"Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Inc. with respect to the accuracy or use of such information, or infringement of patents arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights."
**Table of Contents**

<table>
<thead>
<tr>
<th>SECTION 1</th>
<th>INTRODUCTION TO EMBEDDED SOLUTIONS FROM MICROCHIP</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded Control Overview .................................................</td>
<td>1-1</td>
<td></td>
</tr>
<tr>
<td>PIC16/17 Microcontroller Families Overview/Road Map ..................</td>
<td>1-1</td>
<td></td>
</tr>
<tr>
<td>PIC16/17 Naming Convention ..............................................</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>Low-Voltage Operation of PIC16/17 Microcontrollers ..................</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>Serial EEPROM Overview ..................................................</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>One-Time-Programmable EPROM Overview ...................................</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>The Advantages of One Time Programmable ..............................</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>Application-Specific Standard Product Family ..........................</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>Ease of Production Utilizing Quick Turn Programming (QTP) and ...</td>
<td>1-6</td>
<td></td>
</tr>
<tr>
<td>Serialized Quick Turn Programming (SQTP) ...................................</td>
<td>4-1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION 2</th>
<th>PIC16C5X APPLICATION NOTES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison of 8-Bit Microcontrollers - AN520 .....................</td>
<td>2-1</td>
<td></td>
</tr>
<tr>
<td>Software Interrupt Techniques - AN514 ...............................</td>
<td>2-11</td>
<td></td>
</tr>
<tr>
<td>Software Stack Management - AN527 ....................................</td>
<td>2-15</td>
<td></td>
</tr>
<tr>
<td>Power-Up Considerations - AN522 ......................................</td>
<td>2-19</td>
<td></td>
</tr>
<tr>
<td>Implementation of an Asynchronous Serial I/O - AN510 .............</td>
<td>2-23</td>
<td></td>
</tr>
<tr>
<td>Using PIC16C5X as a Smart PIC™ Peripheral - AN541 ................</td>
<td>2-41</td>
<td></td>
</tr>
<tr>
<td>PLD Replacement - AN511 ..................................................</td>
<td>2-59</td>
<td></td>
</tr>
<tr>
<td>Analog to Digital Conversion - AN513 .....................................</td>
<td>2-79</td>
<td></td>
</tr>
<tr>
<td>Implementing Ohmmeter/Temperature Sensor - AN512 ..................</td>
<td>2-85</td>
<td></td>
</tr>
<tr>
<td>Interfacing to AC Power Lines - AN521 ...................................</td>
<td>2-91</td>
<td></td>
</tr>
<tr>
<td>Implementing Wake-Up on Keystroke - AN528 ...........................</td>
<td>2-93</td>
<td></td>
</tr>
<tr>
<td>Multiplexing LED Drive and Keypad - AN529 ............................</td>
<td>2-97</td>
<td></td>
</tr>
<tr>
<td>Implementing a Simple Serial Mouse Controller - AN519 ............</td>
<td>2-121</td>
<td></td>
</tr>
<tr>
<td>Intelligent Remote Positioner - AN531 ...................................</td>
<td>2-133</td>
<td></td>
</tr>
<tr>
<td>Programming PIC16C5X Devices on Data I/O Unisite - AN524 .........</td>
<td>2-149</td>
<td></td>
</tr>
<tr>
<td>Programming PIC16C5X Devices on Logical Devices ALLPRO - AN525</td>
<td>2-153</td>
<td></td>
</tr>
<tr>
<td>Utility Math Routines - AN526 ...........................................</td>
<td>2-155</td>
<td></td>
</tr>
<tr>
<td>Implementing an LCD Controller - AN563 ...............................</td>
<td>2-215</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION 3</th>
<th>PIC16CXX APPLICATION NOTES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Analog to Digital Converter - AN546 .....................</td>
<td>3-1</td>
<td></td>
</tr>
<tr>
<td>Implementing Wake Up on Keystroke - AN552 .........................</td>
<td>3-21</td>
<td></td>
</tr>
<tr>
<td>PortB as External Interrupt - AN566 ...................................</td>
<td>3-25</td>
<td></td>
</tr>
<tr>
<td>Table Read Using PIC16CXX - AN556 ....................................</td>
<td>3-29</td>
<td></td>
</tr>
<tr>
<td>Software Implementation of PIC™ Bus Master - AN554 ................</td>
<td>3-33</td>
<td></td>
</tr>
<tr>
<td>Software Implementation of Asynchronous Serial I/O - AN555 ........</td>
<td>3-121</td>
<td></td>
</tr>
<tr>
<td>Four Channel Digital Volt Meter with Display and Keyboard - AN557</td>
<td>3-157</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION 4</th>
<th>PIC17CXX APPLICATION NOTES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saving and Restoring Status on Interrupt - AN534 ..................</td>
<td>4-1</td>
<td></td>
</tr>
<tr>
<td>Implementing Table Read and Write - AN548 .........................</td>
<td>4-3</td>
<td></td>
</tr>
<tr>
<td>Frequency and Resolution Options for PWM Outputs - AN539 ..........</td>
<td>4-7</td>
<td></td>
</tr>
<tr>
<td>Using PWM to Generate Analog Output - AN538 .......................</td>
<td>4-15</td>
<td></td>
</tr>
<tr>
<td>Using the PWM - AN564 ..................................................</td>
<td>4-17</td>
<td></td>
</tr>
<tr>
<td>Using the Capture Module - AN545 .....................................</td>
<td>4-29</td>
<td></td>
</tr>
<tr>
<td>Serial Port Utilities - AN547 ..........................................</td>
<td>4-61</td>
<td></td>
</tr>
<tr>
<td>Utility Math Routines - AN544 .........................................</td>
<td>4-71</td>
<td></td>
</tr>
<tr>
<td>Implementing IIR Digital Filters - AN540 .............................</td>
<td>4-121</td>
<td></td>
</tr>
<tr>
<td>Implementation of Fast Fourier Transforms - AN542 ..................</td>
<td>4-139</td>
<td></td>
</tr>
<tr>
<td>Tone Generation - AN543 ..............................................</td>
<td>4-161</td>
<td></td>
</tr>
<tr>
<td>Servo Control of a DC Brush Motor - AN532 ............................</td>
<td>4-197</td>
<td></td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
# Table of Contents

## SECTION 5  INTERFACING PIC16/17 WITH SERIAL EEPROMS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicating with PIC Bus Using PIC16C5X - AN515</td>
<td>5-1</td>
</tr>
<tr>
<td>Interfacing 93CX6 Serial EEPROMs to the PIC16C5X - AN530</td>
<td>5-11</td>
</tr>
<tr>
<td>Logic Powered Serial EEPROMs - AN535</td>
<td>5-29</td>
</tr>
</tbody>
</table>

## SECTION 6  SERIAL EEPROMS TUTORIALS AND APPLICATION NOTES

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Serial EEPROM Operation - AN536</td>
<td>6-1</td>
</tr>
<tr>
<td>Everything a System Engineer Needs to Know About</td>
<td></td>
</tr>
<tr>
<td>Serial EEPROM Endurance - AN537</td>
<td>6-15</td>
</tr>
<tr>
<td>Using the Microchip Endurance Predictive Software - AN562</td>
<td>6-23</td>
</tr>
<tr>
<td>Interfacing 24LCXX Serial EEPROMs to the PIC16C54 - AN567</td>
<td>6-27</td>
</tr>
<tr>
<td>Using the 24C65 and 24C32 with Stand-alone PIC16/17 Code - AN558</td>
<td>6-51</td>
</tr>
<tr>
<td>24C01A Compatibility Issues and Its Mobility for Memory Upgrade - AN517</td>
<td>6-91</td>
</tr>
<tr>
<td>Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559</td>
<td>6-93</td>
</tr>
<tr>
<td>Using the 93LC56 and 93LC66 - AN560</td>
<td>6-97</td>
</tr>
<tr>
<td>ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues - AN516</td>
<td>6-115</td>
</tr>
<tr>
<td>1.8 Volt Technology - Benefits</td>
<td>6-117</td>
</tr>
<tr>
<td>Serial EEPROM Solutions vs. Parallel Solutions</td>
<td>6-119</td>
</tr>
</tbody>
</table>

## SECTION 7  DEVELOPMENT TOOLS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to Microchip Development Tools</td>
<td>7-1</td>
</tr>
<tr>
<td>Application-Specific Standard Products Division:</td>
<td></td>
</tr>
<tr>
<td>PICSEE™ Tools Product Brief</td>
<td>7-3</td>
</tr>
<tr>
<td>Logic Product Division:</td>
<td></td>
</tr>
<tr>
<td>MPALC Cross Assembler Product Brief</td>
<td>7-5</td>
</tr>
<tr>
<td>MPASM Universal Assembler/Linker Product Brief</td>
<td>7-7</td>
</tr>
<tr>
<td>MPSIM Simulator Product Brief</td>
<td>7-9</td>
</tr>
<tr>
<td>PICMASTER™-16X System Product Brief</td>
<td>7-11</td>
</tr>
<tr>
<td>PICMASTER-17 System Product Brief</td>
<td>7-15</td>
</tr>
<tr>
<td>PRO MASTER™ Product Brief</td>
<td>7-19</td>
</tr>
<tr>
<td>PICSTART™-16B Product Brief</td>
<td>7-21</td>
</tr>
<tr>
<td>Memory Products Division:</td>
<td></td>
</tr>
<tr>
<td>Total Endurance™ Predictive Software Model</td>
<td>7-23</td>
</tr>
<tr>
<td>Serial EEPROM Evaluation Board</td>
<td>7-25</td>
</tr>
<tr>
<td>Microchip Bulletin Board Service (BBS)</td>
<td>7-27</td>
</tr>
</tbody>
</table>

## SECTION 8  ARTICLE REPRINTS FOR PIC16/17 MICROCONTROLLERS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16C5X</td>
<td>8-1</td>
</tr>
<tr>
<td>Lean and Mean PIC* Machines</td>
<td></td>
</tr>
<tr>
<td>16C71</td>
<td>8-7</td>
</tr>
<tr>
<td>Enhanced PIC16Cxx Gets ADC and Interrupts</td>
<td></td>
</tr>
<tr>
<td>16C84</td>
<td>8-8</td>
</tr>
<tr>
<td>Microchip PIC Adds EEPROM Data Memory</td>
<td></td>
</tr>
<tr>
<td>17C42</td>
<td>8-9</td>
</tr>
<tr>
<td>Using the PIC Micro</td>
<td></td>
</tr>
<tr>
<td>17C42</td>
<td>8-11</td>
</tr>
<tr>
<td>PIC17C42 Based DC Motor Control (Japanese)</td>
<td></td>
</tr>
<tr>
<td>PICSTART-16B</td>
<td>8-17</td>
</tr>
<tr>
<td>Take your Pick with Controllers</td>
<td></td>
</tr>
<tr>
<td>Serial EEPROM</td>
<td>8-18</td>
</tr>
<tr>
<td>Microchip Technology Rewrites the EEPROM</td>
<td></td>
</tr>
<tr>
<td>Serial EEPROM</td>
<td>8-19</td>
</tr>
<tr>
<td>Serial EEPROM for Embedded Applications</td>
<td></td>
</tr>
<tr>
<td>Endurance</td>
<td>8-23</td>
</tr>
<tr>
<td>A Tool for Calculating EEROM Endurance</td>
<td></td>
</tr>
</tbody>
</table>

## APPENDIX A  OFFICE LOCATIONS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factory Representatives</td>
<td>A-1</td>
</tr>
<tr>
<td>Distributors</td>
<td>A-2</td>
</tr>
<tr>
<td>Field Sales Offices</td>
<td>A-3</td>
</tr>
</tbody>
</table>

*PIC is registered trademark of Microchip Technology Inc. in the U.S.A. and a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.
CROSS REFERENCE GUIDE TO APPLICATION NOTES - ALPHABETICAL

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24C01A Compatibility Issues and its Mobility for Memory Upgrade</td>
<td>6-69</td>
</tr>
<tr>
<td>Analog to Digital Conversion</td>
<td>2-79</td>
</tr>
<tr>
<td>Basic Serial EEPROM Operation</td>
<td>6-1</td>
</tr>
<tr>
<td>Communicating with I^C Bus Using PIC16C5X</td>
<td>5-1</td>
</tr>
<tr>
<td>Comparison of 8-Bit Microcontrollers</td>
<td>2-1</td>
</tr>
<tr>
<td>ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues</td>
<td>6-91</td>
</tr>
<tr>
<td>Everything a System Engineer Needs to Know About</td>
<td></td>
</tr>
<tr>
<td>Four Channel Digital Volt Meter with Display and Keyboard</td>
<td>3-157</td>
</tr>
<tr>
<td>Frequency and Resolution Options for PWM Outputs</td>
<td>4-7</td>
</tr>
<tr>
<td>Implementing a Simple Serial Mouse Controller</td>
<td>2-121</td>
</tr>
<tr>
<td>Implementing an LCD Controller</td>
<td>2-215</td>
</tr>
<tr>
<td>Implementing IIR Digital Filters</td>
<td>4-121</td>
</tr>
<tr>
<td>Implementing Ohmmeter/Temperature Sensor</td>
<td>2-85</td>
</tr>
<tr>
<td>Implementing Table Read and Write</td>
<td>4-3</td>
</tr>
<tr>
<td>Implementing Wake-Up on Keystroke</td>
<td>2-93</td>
</tr>
<tr>
<td>Implementing Wake-Up on Keystroke</td>
<td>3-21</td>
</tr>
<tr>
<td>Implementation of an Asynchronous Serial I/O</td>
<td>2-23</td>
</tr>
<tr>
<td>Implementation of Fast Fourier Transforms</td>
<td>4-139</td>
</tr>
<tr>
<td>Intelligent Remote Positioner</td>
<td>2-133</td>
</tr>
<tr>
<td>Interfacing 93CX6 Serial EEPROMs to the PIC16C5X</td>
<td>5-11</td>
</tr>
<tr>
<td>Interfacing 24LCXX Serial EEPROMs to the PIC16C54</td>
<td>6-27</td>
</tr>
<tr>
<td>Interfacing to AC Power Lines</td>
<td>2-91</td>
</tr>
<tr>
<td>Logic Powered Serial EEPROMs</td>
<td>5-29</td>
</tr>
<tr>
<td>Multiplexing LED Drive and Keypad</td>
<td>2-97</td>
</tr>
<tr>
<td>1.8 Volt Technology - Benefits</td>
<td>6-117</td>
</tr>
<tr>
<td>Optimizing Serial Bus Operations with Proper Write Cycle Times</td>
<td>6-71</td>
</tr>
<tr>
<td>PLD Replacement</td>
<td>2-59</td>
</tr>
<tr>
<td>PortB as External Interrupt</td>
<td>3-25</td>
</tr>
<tr>
<td>Power-Up Considerations</td>
<td>2-19</td>
</tr>
<tr>
<td>Programming PIC16C5X Devices on Data I/O Unisite</td>
<td>2-149</td>
</tr>
<tr>
<td>Programming PIC16C5X Devices on Logical Devices ALLPRO</td>
<td>2-153</td>
</tr>
<tr>
<td>Saving and Restoring Status on Interrupt</td>
<td>4-1</td>
</tr>
<tr>
<td>Serial EEPROM Endurance</td>
<td>6-15</td>
</tr>
<tr>
<td>Serial EEPROM Solutions vs. Parallel Solutions</td>
<td>6-119</td>
</tr>
<tr>
<td>Serial Port Utilities</td>
<td>4-61</td>
</tr>
<tr>
<td>Servo Control of a DC Brush Motor</td>
<td>4-197</td>
</tr>
<tr>
<td>Software Implementation of Asynchronous Serial I/O</td>
<td>3-121</td>
</tr>
<tr>
<td>Software Implementation of I^C Bus Master</td>
<td>3-33</td>
</tr>
<tr>
<td>Software Interrupt Techniques</td>
<td>2-11</td>
</tr>
<tr>
<td>Software Stack Management</td>
<td>2-15</td>
</tr>
<tr>
<td>Table Read Using PIC16CXX</td>
<td>3-29</td>
</tr>
<tr>
<td>Tone Generation</td>
<td>4-161</td>
</tr>
<tr>
<td>Using PIC16C5X as a Smart I^C Peripheral</td>
<td>2-41</td>
</tr>
<tr>
<td>Using PWM to Generate Analog Output</td>
<td>4-15</td>
</tr>
<tr>
<td>Using the Capture Module</td>
<td>4-29</td>
</tr>
<tr>
<td>Using the PWM</td>
<td>4-17</td>
</tr>
<tr>
<td>Using the Analog to Digital Converter</td>
<td>3-1</td>
</tr>
<tr>
<td>Using the Microchip Endurance Predictive Software</td>
<td>6-23</td>
</tr>
<tr>
<td>Using the 24C65 and 24C32 with Standalone PIC16/17 Code</td>
<td>6-33</td>
</tr>
<tr>
<td>Using the 93LC56 and 93LC66</td>
<td>6-75</td>
</tr>
<tr>
<td>Utility Math Routines (PIC16C5X)</td>
<td>2-155</td>
</tr>
<tr>
<td>Utility Math Routines (PIC17CXX)</td>
<td>4-71</td>
</tr>
</tbody>
</table>
## CROSS REFERENCE GUIDE TO APPLICATION NOTES - NUMERICAL

| AN510  | Implementation of an Asynchronous Serial I/O | 2-23 |
| AN511  | PLD Replacement | 2-59 |
| AN512  | Implementing Ohmmeter/Temperature Sensor | 2-85 |
| AN513  | Analog to Digital Conversion | 2-79 |
| AN514  | Software Interrupt Techniques | 2-11 |
| AN515  | Communicating with I²C Bus Using PIC16C5X | 5-1 |
| AN516  | ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues | 6-91 |
| AN517  | 24C01A Compatibility Issues and Its Mobility for Memory Upgrade | 6-369 |
| AN519  | Implementing a Simple Serial Mouse Controller | 2-121 |
| AN520  | Comparison of 8-Bit Microcontrollers | 2-1 |
| AN521  | Interfacing to AC Power Lines | 2-91 |
| AN522  | Power-Up Considerations | 2-19 |
| AN524  | Programming PIC16C5X Devices on Data I/O Unisite | 2-149 |
| AN525  | Programming PIC16C5X Devices on Logical Devices ALLPRO | 2-153 |
| AN526  | Utility Math Routines (PIC16C5X) | 2-155 |
| AN527  | Software Stack Management | 2-15 |
| AN528  | Implementing Wake-Up on Keystroke | 2-93 |
| AN529  | Multiplexing LED Drive and Keypad | 2-97 |
| AN530  | Interfacing 93CX6 Serial EEPROMs to the PIC16C5X | 5-11 |
| AN531  | Intelligent Remote Positioner | 2-133 |
| AN532  | Servo Control of a DC Brush Motor | 4-197 |
| AN534  | Saving and Restoring Status on Interrupt | 4-1 |
| AN535  | Logic Powered Serial EEPROMs | 5-29 |
| AN536  | Basic Serial EEPROM Operation | 6-1 |
| AN537  | Everything a System Engineer Needs to Know About Serial EEPROM Endurance | 6-15 |
| AN538  | Using PWM to Generate Analog Output | 4-15 |
| AN539  | Frequency and Resolution Options for PWM Outputs | 4-7 |
| AN540  | Implementing IIR Digital Filters | 4-121 |
| AN541  | Using PIC16C5X as a Smart I²C Peripheral | 2-41 |
| AN542  | Implementation of Fast Fourier Transforms | 4-139 |
| AN543  | Tone Generation | 4-161 |
| AN544  | Utility Math Routines (PIC17CXX) | 4-71 |
| AN545  | Using the Capture Module | 4-29 |
| AN546  | Using the Analog to Digital Converter | 3-1 |
| AN547  | Serial Port Utilities | 4-61 |
| AN548  | Implementing Table Read and Write | 4-3 |
| AN550  | 1.8 Volt Technology - Benefits | 6-117 |
| AN551  | Serial EEPROM Solutions vs. Parallel Solutions | 6-119 |
| AN552  | Implementing Wake Up on Keystroke | 3-21 |
| AN554  | Software Implementation of PC Bus Master | 3-33 |
| AN555  | Software Implementation of Asynchronous Serial I/O | 3-121 |
| AN556  | Table Read Using PIC16CXX | 3-29 |
| AN557  | Four Channel Digital Volt Meter with Display and Keyboard | 3-157 |
| AN558  | Using the 24XX65 and 24C32 with Standalone PIC16/17 Code | 6-33 |
| AN559  | Optimizing Serial Bus Operations with Proper Write Cycle Times | 6-71 |
| AN560  | Using the 93LC56 and 93LC66 | 6-75 |
| AN562  | Using the Microchip Endurance Predictive Software | 6-23 |
| AN563  | Implementing an LCD Controller | 2-215 |
| AN564  | Using the PWM | 4-17 |
| AN566  | PortB as External Interrupt | 3-25 |
| AN567  | Interfacing 24LCXX Serial EEPROMs to the PIC16C54 | 6-27 |
CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT

Please note that application software written for one family is easily converted to fit another.

<table>
<thead>
<tr>
<th>Subject</th>
<th>PIC16C5X</th>
<th>PIC16CXX</th>
<th>PIC17CXX</th>
<th>Serial EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>24CXX serial EEPROM interface</td>
<td>AN515</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93CXX serial EEPROM interface</td>
<td>AN530</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A/D conversion</td>
<td>AN513</td>
<td>AN546</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC power line, interface to</td>
<td>AN521</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition, 16+16</td>
<td>AN526</td>
<td></td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Addition, floating point</td>
<td>AN526</td>
<td></td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Alarm clock implementation</td>
<td>AN529</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analog to digital conversion</td>
<td>AN513</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asynchronous serial port implementation in software</td>
<td>AN510</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD addition</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD conversion routines</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD subtraction</td>
<td>AN526</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD to binary</td>
<td>AN526</td>
<td></td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Binary to BCD</td>
<td>AN526</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown-out circuits</td>
<td>AN522</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacitance measurement</td>
<td>AN512</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capture routines</td>
<td>AN545</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chip select with Vcc</td>
<td></td>
<td>AN535</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock/calendar implementation</td>
<td>AN529</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatibility - 24C01A</td>
<td></td>
<td>AN517</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatibility - 93C06</td>
<td></td>
<td>AN516</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D/A using PWM</td>
<td>AN538</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital voltmeter</td>
<td></td>
<td>AN557</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital filters</td>
<td></td>
<td>AN540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Division, 16/16, unsigned</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Division, 16/16, signed</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTMF generation</td>
<td>AN543</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Endurance</td>
<td></td>
<td>AN537</td>
<td></td>
<td>AN562</td>
</tr>
<tr>
<td>Endurance predictive model</td>
<td></td>
<td>AN562</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td>AN542</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point addition</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point multiplication</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point routines</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point subtraction</td>
<td>AN526</td>
<td>AN544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency measurement</td>
<td>AN545</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O expansion</td>
<td></td>
<td>AN547</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC implementation (for serial EE interface only)</td>
<td>AN515</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC implementation</td>
<td>AN541</td>
<td>AN554</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIR filter</td>
<td></td>
<td>AN540</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

<table>
<thead>
<tr>
<th>Subject</th>
<th>PIC16C5X</th>
<th>PIC16CXX</th>
<th>PIC17CXX</th>
<th>Serial EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfacing to 24C32/65</td>
<td></td>
<td></td>
<td></td>
<td>AN558</td>
</tr>
<tr>
<td>Interfacing to 24CXX</td>
<td></td>
<td></td>
<td></td>
<td>AN515</td>
</tr>
<tr>
<td>Interfacing to 24LCXX</td>
<td></td>
<td></td>
<td></td>
<td>AN567</td>
</tr>
<tr>
<td>Interfacing to 93CX6</td>
<td></td>
<td></td>
<td></td>
<td>AN530</td>
</tr>
<tr>
<td>Interfacing to 93LC56/66</td>
<td></td>
<td></td>
<td></td>
<td>AN560</td>
</tr>
<tr>
<td>Interrupt, PORTB</td>
<td></td>
<td></td>
<td></td>
<td>AN566</td>
</tr>
<tr>
<td>Interrupt, software</td>
<td></td>
<td></td>
<td>AN557</td>
<td></td>
</tr>
<tr>
<td>Keypad interface</td>
<td>AN528, AN529</td>
<td>AN557</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCD controller</td>
<td></td>
<td></td>
<td>AN563</td>
<td></td>
</tr>
<tr>
<td>LED display interfacing</td>
<td></td>
<td>AN529</td>
<td>AN557</td>
<td></td>
</tr>
<tr>
<td>Math routines</td>
<td></td>
<td>AN526</td>
<td></td>
<td>AN544</td>
</tr>
<tr>
<td>Measuring capacitance</td>
<td></td>
<td>AN512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measuring frequency</td>
<td></td>
<td></td>
<td></td>
<td>AN545</td>
</tr>
<tr>
<td>Measuring period</td>
<td></td>
<td></td>
<td></td>
<td>AN545</td>
</tr>
<tr>
<td>Measuring pulse-width</td>
<td></td>
<td></td>
<td></td>
<td>AN545</td>
</tr>
<tr>
<td>Measuring resistance</td>
<td></td>
<td>AN512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measuring temperature</td>
<td></td>
<td>AN512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microwire interface</td>
<td></td>
<td>AN531</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motor control</td>
<td></td>
<td>AN531</td>
<td>AN532</td>
<td></td>
</tr>
<tr>
<td>Mouse, serial</td>
<td></td>
<td>AN519</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplexing LED and keypad</td>
<td></td>
<td>AN529</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplication, 8 x 8, unsigned</td>
<td></td>
<td>AN526</td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Multiplication, 16 x 16, unsigned</td>
<td></td>
<td>AN526</td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Multiplication, 16 x 16, signed</td>
<td></td>
<td>AN526</td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Multiplication, floating point</td>
<td></td>
<td>AN526</td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Parity generation</td>
<td></td>
<td></td>
<td></td>
<td>AN547</td>
</tr>
<tr>
<td>Period measurement</td>
<td></td>
<td></td>
<td></td>
<td>AN545</td>
</tr>
<tr>
<td>PID</td>
<td></td>
<td>AN531</td>
<td>AN532</td>
<td></td>
</tr>
<tr>
<td>PLA implementation</td>
<td></td>
<td>AN511</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLD replacement using PIC16CXX</td>
<td></td>
<td>AN511</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position control</td>
<td></td>
<td>AN531</td>
<td>AN532</td>
<td></td>
</tr>
<tr>
<td>Positioner</td>
<td></td>
<td>AN531</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power on Reset</td>
<td></td>
<td></td>
<td></td>
<td>AN522</td>
</tr>
<tr>
<td>Pseudo-random number generation</td>
<td></td>
<td></td>
<td></td>
<td>AN544</td>
</tr>
<tr>
<td>Pulse width measurement</td>
<td></td>
<td></td>
<td></td>
<td>AN545</td>
</tr>
<tr>
<td>PWM, using</td>
<td></td>
<td></td>
<td></td>
<td>AN564</td>
</tr>
<tr>
<td>PWM, choosing frequency</td>
<td></td>
<td></td>
<td></td>
<td>AN539</td>
</tr>
<tr>
<td>PWM, choosing resolution</td>
<td></td>
<td></td>
<td></td>
<td>AN539</td>
</tr>
<tr>
<td>PWM generation in software</td>
<td></td>
<td></td>
<td>AN531</td>
<td></td>
</tr>
<tr>
<td>PWM routines</td>
<td></td>
<td></td>
<td>AN539</td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td>PIC16C5X</td>
<td>PIC16C6X</td>
<td>PIC17CXX</td>
<td>Serial EEPROM</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>PWM to analog output</td>
<td></td>
<td></td>
<td></td>
<td>AN538</td>
</tr>
<tr>
<td>Quadrature encoder interface</td>
<td></td>
<td></td>
<td></td>
<td>AN532</td>
</tr>
<tr>
<td>Random number generation, Gaussian</td>
<td></td>
<td></td>
<td></td>
<td>AN544</td>
</tr>
<tr>
<td>Random number generation</td>
<td></td>
<td></td>
<td></td>
<td>AN544</td>
</tr>
<tr>
<td>Real time clock implementation (from AC power line)</td>
<td>AN521</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resistance measurement with PIC16CXX</td>
<td></td>
<td></td>
<td></td>
<td>AN512</td>
</tr>
<tr>
<td>RS232 interface</td>
<td></td>
<td></td>
<td></td>
<td>AN547</td>
</tr>
<tr>
<td>SEEPROM operation</td>
<td></td>
<td></td>
<td></td>
<td>AN536</td>
</tr>
<tr>
<td>Sequencer implementation</td>
<td></td>
<td></td>
<td></td>
<td>AN511</td>
</tr>
<tr>
<td>Serial EEPROM interfacing</td>
<td></td>
<td></td>
<td></td>
<td>AN515</td>
</tr>
<tr>
<td>Serial port (USART) routines</td>
<td></td>
<td></td>
<td>AN555</td>
<td>AN547</td>
</tr>
<tr>
<td>Servo control</td>
<td></td>
<td></td>
<td></td>
<td>AN532</td>
</tr>
<tr>
<td>Sine wave generation</td>
<td></td>
<td></td>
<td></td>
<td>AN543</td>
</tr>
<tr>
<td>Software stack</td>
<td>AN527</td>
<td></td>
<td>AN534</td>
<td></td>
</tr>
<tr>
<td>Square root</td>
<td>AN526</td>
<td></td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Stack management in software</td>
<td>AN527</td>
<td></td>
<td>AN534</td>
<td></td>
</tr>
<tr>
<td>State machine implementation</td>
<td></td>
<td></td>
<td></td>
<td>AN511</td>
</tr>
<tr>
<td>Status save and restore</td>
<td></td>
<td></td>
<td></td>
<td>AN534</td>
</tr>
<tr>
<td>Subtraction, 16+16</td>
<td>AN526</td>
<td></td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Subtraction, floating point</td>
<td>AN526</td>
<td></td>
<td>AN544</td>
<td></td>
</tr>
<tr>
<td>Table Read</td>
<td></td>
<td></td>
<td></td>
<td>AN556</td>
</tr>
<tr>
<td>Temperature measurement</td>
<td>AN512</td>
<td></td>
<td>AN548</td>
<td></td>
</tr>
<tr>
<td>Thermostat implementation</td>
<td></td>
<td></td>
<td></td>
<td>AN512</td>
</tr>
<tr>
<td>Tone generation</td>
<td></td>
<td></td>
<td></td>
<td>AN543</td>
</tr>
<tr>
<td>Trajectory generation</td>
<td></td>
<td></td>
<td></td>
<td>AN532</td>
</tr>
<tr>
<td>UART, software implementation</td>
<td></td>
<td></td>
<td></td>
<td>AN510</td>
</tr>
<tr>
<td>USART routines</td>
<td></td>
<td></td>
<td></td>
<td>AN547</td>
</tr>
<tr>
<td>Velocity control</td>
<td></td>
<td></td>
<td></td>
<td>AN532</td>
</tr>
<tr>
<td>Voltmeter implementation</td>
<td></td>
<td></td>
<td></td>
<td>AN157</td>
</tr>
<tr>
<td>Wake up on key-stroke</td>
<td>AN528</td>
<td></td>
<td>AN552</td>
<td></td>
</tr>
<tr>
<td>Write cycle optimization</td>
<td></td>
<td></td>
<td></td>
<td>AN559</td>
</tr>
<tr>
<td>Zero crossing detect</td>
<td></td>
<td></td>
<td></td>
<td>AN521</td>
</tr>
</tbody>
</table>
SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS

Motivated by customer requirements....

"Microchip Technology draws its impetus from the technology expectations of a large base of longstanding customers. Microchip is small enough to respond quickly with technology to serve our customers' needs. Moreover, as a fully integrated IC manufacturer, Microchip deploys its panoply of resources to act timely and efficiently, and on a worldwide scale: Technology Development, Design, Wafer Fabrication, Assembly and Test, Quality, Reliability and Customer Support.

...and powered by continuous improvement...

"Worldwide competition leaves no room for divergence or mediocrity. Microchip Technology, committed to focus on and continuously improve all the aspects of its business, has a unique corporate culture. To improve performance, our employees are encouraged to analyze their methods continually. Personal empowerment expands the capability of personal responsibility to continually serve our customers better.

...riding, leading and pushing the wave of technological change.

"Our industry's life-line is innovation. The fast pace of technological change is inherent in our industry. Microchip Technology has accelerated the rate of change of its technology and products to leadership in providing user-programmable space-sensitive embedded control solutions.

"Change is our ally. Driving and managing customer-focused change is our winning strategy."

Steve Sanghi
President & Chief Executive Officer
HIGHLIGHTS

• Focused on providing field-programmable embedded control solutions
• Offers RISC 8-bit user-programmable microcontrollers and supporting logic products
• Offers Serial and Parallel EEPROMs and EPROMs
• A unique corporate culture dedicated to continuous improvement
• Research and development of high performance field-programmable products
• A history of innovation
• An experienced executive team focussed on innovation
• Fully integrated manufacturing
• A global network of manufacturing and customer support facilities

BUSINESS SCOPE

Microchip Technology Inc. manufactures and markets a variety of VLSI CMOS semiconductor components to support the field-programmable embedded control market. In particular, the company specializes in highly integrated, field-programmable RISC microcontrollers and related non-volatile memory products to meet growing market requirements for high performance, yet economical embedded control capability in an increasing number of price-sensitive products. Microchip's products feature the industry's most economical OTP (one-time programmable) capability, along with the compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.

MARKET FOCUS

Microchip targets selected markets where our advanced designs, progressive process technology and industry leading operating speeds enable us to deliver decidedly superior performance. The firm has positioned itself to maintain a dominant role as a supplier of high performance field-programmable microcontrollers and associated memory and logic products for embedded control applications.
GUIDING VALUES

Customers Are Our Focus
We establish successful customer partnerships by exceeding customer expectations for products, services and attitude. We earn our credibility through meeting commitments and producing quality products and services in a timely fashion. We believe each employee must effectively serve their internal customers in order for Microchip's external customers to be properly served.

Quality Comes First
We will perform correctly the first time, maintain customer satisfaction and measure our quality against requirements. We practice effective and standardized improvement methods, such as statistical process control to anticipate problems and implement root cause solutions. We believe that when quality comes first, reduced costs follow.

Continuous Improvement Is Essential
We utilize the concept of ‘Vital Few’ to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Employees Are Our Greatest Strength
We design jobs and provide opportunities in a fashion which clearly promotes pride in work, integrity, trust, teamwork, creativity, employee involvement and development, fairness, and productivity. We base recognition, advancement and compensation on an employee's achievement of excellence in team and individual performance. We provide for employee health and welfare by offering a competitive, comprehensive employee benefits program.

Products And Technology Are Our Foundation
We commit to ongoing investments and advancements in the design and development of our manufacturing process, device circuit and system technologies, which provide innovative, reliable and cost-effective products to support current and future market opportunities.

Total Cycle Times Are Competitive
We focus resources to optimize cycle times to our customers by empowering employees to achieve efficient cycle times in their area of responsibility. We believe that cycle time reduction is achieved by streamlining processes through the systematic removal of barriers to productivity.

Safety Is Never Compromised
We place our concern for safety of our employees and community at the forefront of our decisions, policies and actions. Each employee is responsible for safety.

Profits Provide For Everything We Do
We strive to maintain competitive profits as they allow continued investments and future growth, and indicate the overall success of Microchip.

Communication Is Vital
We encourage open, honest, constructive, and ongoing communication in all company and community relationships to resolve issues, exchange information and share knowledge.

Suppliers, Representatives, And Distributors Are Our Partners
We maintain mutually beneficial partnerships with suppliers, representatives, and distributors who are an integral link in the achievement of our mission and guiding values.

Professional Ethics Are Practiced
We manage our business and treat customers, employees, shareholders, investors, suppliers, distributors, representatives, community and government in a manner that exemplifies our honesty, ethics and integrity. We recognize our responsibility to the community and are proud to serve as an equal opportunity employer.
FULLY INTEGRATED MANUFACTURING

Microchip delivers fast turnaround through total control over all phases of production. Research and development, design, mask making, wafer fabrication, assembly and quality assurance testing are conducted at facilities owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced statistical process control (SPC), and a continuous improvement culture has brought forth tight product consistency levels and high yields which enable Microchip to compete successfully in world markets. Microchip's unique approach to SPC provides customers with excellent costs, quality, reliability and on-time delivery.

A GLOBAL NETWORK OF PLANTS AND FACILITIES

Microchip is a global competitor providing local service to the world's technology centers. The Company's focal point is the design and technology advancement facility in Chandler, Arizona. Product and technology development is here, along with front-end wafer fabrication and electrical probing.

Microchip's assembly and test facility in Kaohsiung, Taiwan houses the technology and modern assembly methods necessary for plastic and ceramic packaging. Other quality-conscious firms which fabricate wafers in the Pacific Rim use Microchip's Kaohsiung plant for assembly.

Sales and application offices are located in key cities throughout the Western Hemisphere, Pacific Rim and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical support, purchasing information and failure analysis.

A PRODUCT FAMILY OF SHARED STRENGTHS

Microchip's product focus is CMOS field-programmable microcontrollers, non-volatile memories and peripherals, and application-specific standard products (ASSP). These product lines include PIC16/17 microcontrollers, EEPROMs, high-speed EPROMs, and peripherals in a broad range of product densities, speeds and packages.

MICROCONTROLLERS

PIC16/17 microcontrollers from Microchip combine high performance, low cost and small package size. They offer the best price/performance ratio in the industry. Large numbers of these devices are used in automotive and cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, security and telecommunication applications.

The widely-accepted CMOS PIC16CXX and PIC17CXX families are the industry's only 8-bit microcontrollers using a high-speed RISC architecture. Microchip pioneered the use of RISC architecture to obtain high speed and instruction efficiency. The CMOS PIC16CXX family is in high-volume production, with more than 40 million units shipped, and has achieved more than five thousand design wins worldwide.

The PIC17CXX family offers the world's fastest execution performance of any 8-bit microcontroller family. The PIC17CXX family extends the PIC16/17 microcontroller's high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set, and powerful vectored interrupt handling capabilities. The first member of the family, the PIC17C42, includes a powerful array of intelligent and precise on-chip peripheral features that are ideally suited for many demanding real-time embedded control applications including motor control, process control, security, automotive and medical applications. In addition, the PIC17C42 can function either as a stand-alone microcontroller or can execute instructions from up to 64K words of external program memory. The PIC17C42 features comprehensive timer/counter resources and I/O handling capabilities to address the requirements of complex embedded control applications.
Current CMOS PIC16/17 microcontroller product families include advanced features such as sophisticated timers, embedded A/D, extended instruction/data memory, inter-processor communication and ROM, EPROM and EEPROM memories.

Both PIC16CXX and PIC17CXX families are supported by user-friendly development systems including programmers and emulators.

DEVELOPMENT SYSTEMS

The PICMASTER™ is an advanced real-time in-circuit emulator system using the user-friendly Windows™ software environment. The PICMASTER is a Microchip-designed universal emulator for both PIC16CXX and PIC17CXX families. The PRO MASTER™ is an advanced full-featured programmer. PICSTART™ is a low-cost development kit which includes an assembler, simulator and programmer.

SOFTWARE SUPPORT

Both PIC16/17 microcontroller families are supported by assemblers, linker/loaders, libraries and a source-level debugger. The PIC16CXX family is also supported by a software simulator.

A full-featured C Compiler and Fuzzy Logic support are under development to support both families.

Customers can obtain on-line updates on Microchip Development Systems and Support Software via the Bulletin Board System (BBS). See page 7-27 for access information.

SERIAL EEPROMS

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages. Device densities range from 256K bits up to 64K bits. In addition to 5 volt-only operation, Microchip offers Serial EEPROMs that read and write down to 2.5, 2 or 1.8 volts. PIC™, Microwire™ and 4-wire bus interface protocols are standard. Devices come in three standard operating temperature ranges; commercial, industrial and automotive. Small footprint packages include: 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths and 14-lead SOIC. Other key features of the Serial EEPROM product line include: electrostatic discharge (ESD) protection greater than 4K volts and endurance of 100K cycles minimum and one million typical.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide, including consumer, automotive, industrial, computer and communications. To date, more than 80 million units have been produced. Microchip is continuing to develop additional unique Serial EEPROMs.

PARALLEL EEPROMS

The CMOS Parallel EEPROM devices from Microchip are available in 4K, 16K, and 64K densities. The manufacturing process used for these EEPROMs ensures 10,000 to 100,000 write and erase cycles typically. Data retention is more than 10 years. Fast write times are less than 200 μsec. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from -40°C to +125°C. Microchip’s expertise in advanced SOIC, TSOP and VSOP surface mount packaging supports our customers’ needs in space-sensitive applications.

Typical applications include computer peripherals, engine control, pattern recognition and telecommunications.
EPROMS

Microchip’s CMOS EPROM devices are produced in densities from 64K to 512K. High Speed EPROMs have access times as low as 55 nanoseconds. Typical applications include computer peripherals, military, instrumentation, and automotive devices. Microchip’s expertise in Surface Mount Packaging on SOIC, TSOP and VSOP packages led to the development of the Surface Mount one-time-programmable (OTP) EPROM market where Microchip is the #1 supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

APPLICATION SPECIFIC STANDARD PRODUCTS (ASSP)

Microchip’s new Application-Specific Standard Product (ASSP) Division has the mission of providing value added embedded control solutions through PIC16/17 microcontroller architecture products combined with innovative software, silicon and assembly technology. These products will incorporate technology that will offer a complete solution that is both unique to the customer and standard in manufacture to Microchip. The central theme of this family is to offer a complete solution which reduces or removes the barriers for customers to use Microchip solutions in their products through the use of software embedded in secure OTP- or ROM-based microcontrollers, packaged to provide the highest integration to the customer at the best overall system cost.

The MTA11XXX family is the most accurate and most integrated battery fuel gauge product available today. The family incorporates Microchip/SPAN patented TrueGauge™ technology which digitally integrates battery charge and discharge current to provide an accurate (<3% typical) state of charge indication. The family operates with NiCd and NiMH battery packs from 3 Vdc to 30 Vdc. These products are ideal for portable PC, cellular phone and portable consumer product applications.

Ease of use, low voltage and low cost make the MTA41XXX mouse and trackball MCU firmware solutions ideal for implementing new designs for both PCs and Apple® computers. The products in the MTA41XXX family are 18-lead, low-power CMOS microcontroller ICs combined with application-specific software. By adding a few external components, the user can easily realize a complete mouse or trackball system.

The MTA810XX product line is the first in a series of Multi-Chip Module (MCM) products that integrate PIC16/17 microcontrollers with EEPROM technology. These PICSEE™ devices are ideally suited for security, keyless entry, data logging and telecommunication applications. The combined product assembly techniques provide the user the highest performance solution in a compact and cost-effective package.

Future ASSP products will include advanced features such as mixed analog and digital capability as well as an ever broadening family of turnkey software solutions for the embedded control market.

OTHER MICROCHIP PRODUCTS

Other Microchip products, such as Liquid Crystal Display Drivers, are mature products with proven track record and a large, repeat customer base.

A HISTORY OF INNOVATION

Microchip has a long history of innovation in the semiconductor industry. For more than a quarter century, Microchip and its former parent company have been developers of leading-edge, cost-effective logic and memory products.

Microchip is credited with a number of firsts: The Metal-Oxide-Silicon (MOS) Integrated Circuit, DRAM, Serial EEPROM, Reduced Instruction Set Computer (RISC) microcontroller product family, UART, CMOS 64K EEPROM, and CMOS single chip DSP are all innovations that were originally developed and introduced by Microchip engineers.
Microchip Technology Incorporated

CHANDLER, AZ FACILITY

Chandler Wafer Fabrication: Diffusion Area

Chandler Wafer Fab: Sub-micron Alignment Area

TAIWAN FACILITY

Microchip's assembly and test operation in Kaohsiung, Taiwan received the prestigious Ishikawa Award for assembly and testing excellence.

The Microchip Kaohsiung plant's excellent track record and continuing efforts to achieve higher levels of quality and technological advancement has resulted in superior yields and fast turnaround.
FUTURE PRODUCTS AND TECHNOLOGY

New process technology is constantly being developed for microcontroller, ASSP, EEPROM, and high-speed EPROM products. Advanced process technology modules are being developed that will be integrated into present product lines to continue to achieve a range of compatible processes. Current production technology utilizes dimensions down to 0.9 microns.

Microchip's research and development activities, include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools on line. Cycle times for new technology development are continuously reduced by using in-house mask making, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

More advanced technologies are under development, as well as advanced CMOS RISC-based microcontroller, ASSP and CMOS EEPROM and EPROM products. Objective specifications for new products are developed by listening to our customers and by close cooperation with our many customer-partners worldwide.

QUALITY WITHOUT COMPROMISE

Product reliability is designed into Microchip products at the outset. Wide design margins are established to guarantee that every product can be produced easily, error-free and within the tolerances of the manufacturing process.

All quality assurance tests are tighter than customer specifications. Products are tested at least two machine tolerances tighter than those specified by the customer.

Every new product is qualified under accelerated stress testing. Test samples encompass the full range of processed tolerances at each step. Data sheets detailing these processes enable customers to reach accurate decisions based on known quantitative values.

To determine whether a process is within normal manufacturing variation, industry-leading statistical control techniques are put to work at each process step. In-process controls are performed by operators in the wafer fabrication division and immediate corrective action is taken if they deem a process is out of tight control limits. Products are also sampled weekly through a variety of carefully monitored stress and accelerated life tests.

Microchip's documentation control program assures the correct document is always available at the point of use. Active documents are serialized and stamped to eliminate the possibility of performing a job from obsolete or incorrect instructions.

Individuals in all departments continuously analyze the methods employed at their positions and formulate plans to improve performance. In all areas of our business, everyone is expected to make continuous improvement.

A QUALITY AND RELIABILITY ALLIANCE WITH CUSTOMERS

Microchip works together with customers to establish mutual programs to improve the performance of our products in their systems. We go beyond the incoming inspection level and specification by extending our quality and reliability support to the point where the customer ships the system. Microchip's quality programs ensure that our products can be used with such impunity, a customer can implement improvement programs based on Microchip as your leading supplier.
SECTION 1
INTRODUCTION TO EMBEDDED SOLUTIONS

Embedded Control Overview ................................................................................................................. 1
PIC16/17 Microcontroller Families Overview/Road Map ................................................................. 1
PIC16/17 Naming Convention .......................................................................................................... 1
Low-Voltage Operation of PIC16/17 Microcontrollers .................................................................... 1
Serial EEPROM Overview .............................................................................................................. 1
One-Time-Programmable EPROM Overview ................................................................................... 1
The Advantages of One-Time-Programmable .................................................................................. 1
Application-Specific Standard Product Family .............................................................................. 1
Ease of Production Utilizing Quick Turn Programming (QTP) and Serialized Quick Turn Programming (SQTP) ................................................................................................................... 1
Microchip Technology’s mission is to offer leadership semiconductor products for embedded control system applications. To do this we have focused our technology, engineering, manufacturing and marketing resources on two synergistic product lines: Serial EEPROMs and 8-bit PIC16/17 One Time Programmable (OTP) microcontrollers. These product lines provide the solutions to many of the problems facing designers of embedded control systems.

In keeping with one of our guiding values this Embedded Control Handbook has been produced to assist our customers, existing and new, in their efforts to design and produce a state-of-the-art embedded control system, whether it is intended to assist consumer, automotive, computer, communications or industrial embedded system design.

EMBEDDED CONTROL OVERVIEW

The requirement to manage information processing in today’s systems applications necessitate the implementation of embedded control. As opposed to compute intensive environments like workstations and computers of all types, the embedded control system is intended not only to process specific inputs, but also to create the appropriate control response. The use of embedded control allows the system designer to develop a solution for the specific application with the least amount of overhead cost and space. Normally based around a microcontroller, where on-chip program memory and peripheral functions have been combined on a single chip, the needs of the embedded control system are specifically addressed with application specific software and very few, if any, peripheral devices. In the case of a system application where a larger memory may be required, the use of EPROM, EEPROM or Serial EEPROM, designed to communicate directly and efficiently with the microcontroller will provide this extra level of flexibility.

Microchip Technology has established itself as a world leader in providing user-programmable embedded control solutions. The combination of high performance microcontrollers from both the PIC17CXX and PIC16CXX Families, along with industry leading non-volatile memory products provides the basis for this leadership. Continuous innovation and improvement in both the design and manufacturing of embedded control solutions will allow Microchip to maintain and grow this leadership position.

PIC16/17 FAMILIES OVERVIEW AND ROADMAP

The key to providing a successful Embedded Control solution is to insure that the product available has been, and continues to be designed to meet the requirements of the system designer. In order to accomplish this, Microchip has set itself on a course to provide the most useful matrix of products, allowing the system designer to select the microcontroller which will best suit the needs of his particular application. This matrix of products, as shown below, provides the system designer with the ultimate in flexibility and versatility.

FIGURE 1 - MATRIX OF THE PIC16/17 FAMILY OF 8-BIT MICROCONTROLLERS

A strong foundation has been laid with the popular and versatile PIC16CSX Family of High Performance, OTP Microcontrollers. As the base family of PIC16/17s, the PIC16CSX Family provides low cost, small footprint solutions for most embedded control applications. With the low end microcontroller family, Microchip has become a significant factor in the worldwide microcontroller marketplace.

The addition of the PIC17C42, first member of the PIC17CXX Family of high end microcontrollers, demonstrated Microchips continuing commitment to meet the needs of the embedded control systems designer. Most recently, with the introduction of the PIC16C64 (40-pin), Microchip continues its penetration of the mid level 8-Bit microcontroller arena. The PIC16C71 with on-chip A/D converter and the PIC16C84 with EEPROM program and data memory are members of this “mid-range” family.
<table>
<thead>
<tr>
<th>Status as of October 1993</th>
<th>Core Features</th>
<th>Program Memory</th>
<th>Data Memory</th>
<th>MHz</th>
<th>I/O and Peripheral Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Family</strong></td>
<td><strong>Products</strong></td>
<td><strong>Instruction Memory</strong></td>
<td><strong>Instruction Word Length (bits)</strong></td>
<td><strong>Levels of Hardware Stack</strong></td>
<td><strong>EPROM (Words)</strong></td>
</tr>
<tr>
<td>Low-end</td>
<td>PIC16C54</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>.5K</td>
</tr>
<tr>
<td></td>
<td>PIC16C54A</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>.5K</td>
</tr>
<tr>
<td></td>
<td>PIC16CR54</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>PIC16CR54A*</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>PIC16C55</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>.5K</td>
</tr>
<tr>
<td></td>
<td>PIC16C56</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>1K</td>
</tr>
<tr>
<td></td>
<td>PIC16C57</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>2K</td>
</tr>
<tr>
<td></td>
<td>PIC16CR57A</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>PIC16C58A</td>
<td>33</td>
<td>12</td>
<td>2</td>
<td>2K</td>
</tr>
<tr>
<td>Mid-range</td>
<td>PIC16C64</td>
<td>35</td>
<td>14</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PIC16C71</td>
<td>35</td>
<td>14</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PIC16C84</td>
<td>35</td>
<td>14</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>High-end</td>
<td>PIC17C42</td>
<td>55</td>
<td>16</td>
<td>11</td>
<td>16</td>
</tr>
</tbody>
</table>

* In Development.
** All products have program memory code protect, watchdog timer, and power-on reset.
† The PIC17C42 can be user-configured for two 16-bit timers and two 8-bit timers.
PIC16/17 NAMING CONVENTION

The PIC16/17 family of microcontrollers covers the low-end, mid-range and high-end 8-bit controller applications.

The PIC16C5X family with its 12-bit wide instruction covers the low-end and offers the most cost advantage. In cost, this family competes well against the 4-bit microcontrollers, yet in performance it out-performs other low-end 8-bit controllers.

The PIC16CXX family (PIC16C6X, PIC16C7X, PIC16C8X) with its 14-bit wide instruction set and interrupt capability are most suitable for mid-range applications. This family is well-suited for integrating larger program memory, RAM and a wide variety of peripherals.

Finally, the PIC17CXX family with its 16-bit wide extended instruction set, external code execution capability and more sophisticated interrupt structure is a power high-end 8-bit controller family. (See Table 2.)

TABLE 2 - PIC16/17 NAMING CONVENTION

<table>
<thead>
<tr>
<th>16Cx</th>
<th>Low End</th>
<th>PIC16C5X/CR5X</th>
<th>12-bit Architecture</th>
<th>Digital Only (OTP/ROM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16CXX</td>
<td>Mid-Range</td>
<td>PIC16C6X/CR6X</td>
<td>14-bit Architecture</td>
<td>Digital Only (OTP/ROM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16C7X/CR7X</td>
<td></td>
<td>With Analog Functions, e.g. A/D (OTP/ROM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16C8X/CR8X</td>
<td></td>
<td>With EEPROM Memory (OTP/ROM)</td>
</tr>
<tr>
<td>17CXX</td>
<td>High End</td>
<td>PIC17C4X/CR4X</td>
<td>16-bit Architecture</td>
<td>Digital Only (OTP/ROM)</td>
</tr>
</tbody>
</table>
Introduction

PIC16/17 LOW VOLTAGE OPERATION

A wide operating range (as low as 2.0V for ROM and 2.5V for OTP) has made the PIC16C5X and PIC16CXX families the ideal solution for thousands of applications worldwide.

SERIAL EEPROM OVERVIEW

Serial EEPROMs from Microchip come in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges, and space saving packages.

Densities:
Currently range from 1K to 64K with higher density devices in development.

Bus Interface Protocols:
All major protocols are covered: 2-wire, 3-wire and 4-wire.

Operating Voltages:
In addition to standard 5V devices there are two low voltage families. The “LC” devices operate down to 2.5V, while the “AA” family operates, in both read and write mode, down to a breakthrough 1.8V, making these devices highly suitable for alkaline and NiCad battery powered applications.

Temperature Ranges:
Like all Microchip devices, Serial EEPROMs are offered in Commercial (-40°C to 85°C), Industrial (-40°C to 85°C) and Automotive (-40°C to 125°C) operating temperature ranges.

Packages:
The focus here is on small. Most devices are available in 8 pin PDIP or 8 pin SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Endurance is specified at 1M Erase/Write typical and 100K cycles minimum with a data retention specification greater than 40 years. ESD protection is guaranteed up to 4K volts.

OTP EPROM OVERVIEW

Microchip also provides its customers with a number of CMOS OTP EPROMs. Densities offered include: 64K, 128K, 256K and 512K, all in a X8 organization. Our high speed 256K device also comes in a X16 organization. Access times range from a high performance 55ns to a practical 200ns or greater. Low voltage devices, capable of operating at 3V are available at the 256K and 512K density levels. Surface mounted packages, such as TSOP, PLCC and SOIC as well as the more traditional DIP packages are offered. All EPROMs are available in Commercial, Industrial and Automotive temperature ranges.

A full listing of the Serial EEPROM and EPROM product offerings are shown in Table 3. See the individual Microchip data sheet/data book for detailed information.

THE ADVANTAGES OF ONE-TIME-PROGRAMMABLE

In keeping with Microchip’s goal of providing the embedded control system designer with the best tools available, Microchip has developed world-leading OTP technology. Microchip offers a wide variety of OTP EPROM products. Similarly, by basing the PIC16/17 Family of products around an EPROM program memory capability, all of the advantages of OTP, both in development and production, have been made economically available to the systems manufacturer. The benefits of OTP technology include:

• Lower costs and shorter lead times
• Reduced time to market
• In-circuit programming capability
• Code protection via security fuse
• Reduction of inventory requirements at system manufacturing site
• Quick correction of bugs detected in manufacturing
• Quick product feature changes in response to customer requests
• Reduces wasted inventory

APPLICATION SPECIFIC STANDARD PRODUCTS

The Application-Specific Standard Products (ASSP) Division complements and strengthens Microchip’s leadership position in 8-bit microcontrollers and related specialty memory products for the embedded control market. The ASSP Division employs innovative multi-chip module packaging, applications expertise, firmware and new technology to create integrated, single-chip solutions for specific high-volume embedded control applications such as PC pointing devices and energy management. By offering more complete solutions, Microchip can provide its customers with higher value-added products, with the additional benefits of faster time-to-market and lower design overhead. A full ASSP product listing is shown in Table 4.
<table>
<thead>
<tr>
<th>FAMILY</th>
<th>Device</th>
<th>Density Organization</th>
<th>Max. Clock Frequency (cycles min.)</th>
<th>Temp. Range</th>
<th># Pins</th>
<th>Package Types</th>
<th>Operating Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-Wire (I^2C™) Bus Protocol</td>
<td>24C01A</td>
<td>1K bits (128 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24C02A</td>
<td>2K bits (256 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24C04A</td>
<td>4K bits (512 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>85C12</td>
<td>2K bits (256 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>85C22</td>
<td>4K bits (512 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td>STANDARD 2-WIRE FAMILY</td>
<td>24LC01B</td>
<td>1K bits (128 X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24LC02B</td>
<td>2K bits (256 X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24LC04B</td>
<td>4K bits (512 X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24LC08B</td>
<td>8K bits (1K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24LC16B</td>
<td>16K bits (2K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>24AA01</td>
<td>1K bits (128 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>24AA02</td>
<td>2K bits (256 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>24AA04</td>
<td>4K bits (512 X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM, SL</td>
</tr>
<tr>
<td></td>
<td>24AA08</td>
<td>8K bits (1K X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM, SL</td>
</tr>
<tr>
<td></td>
<td>24AA16</td>
<td>16K bits (2K X 8)</td>
<td>100 kHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM, SL</td>
</tr>
<tr>
<td></td>
<td>24C32</td>
<td>32K bits (4K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>24LC32</td>
<td>32K bits (4K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>24C65</td>
<td>64K bits (8K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>24LC65</td>
<td>64K bits (8K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>24AA65</td>
<td>64K bits (8K X 8)</td>
<td>400 kHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td>SMART SERIAL 2-WIRE FAMILY</td>
<td>93C06</td>
<td>256 bits (16 X 16)</td>
<td>1 MHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>93C46</td>
<td>1K bits (64 X 16)</td>
<td>1 MHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>93C56</td>
<td>2K bits (256 X 8)</td>
<td>1 MHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>93C66</td>
<td>4K bits (512 X 8)</td>
<td>1 MHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>93LC01C</td>
<td>1K bits (128 X 8) or (64 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93LC06</td>
<td>2K bits (256 X 8) or (128 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93LC08</td>
<td>4K bits (512 X 8) or (256 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM, SL</td>
</tr>
<tr>
<td></td>
<td>93LC16</td>
<td>1K bits (64 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93LC66</td>
<td>2K bits (128 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93LC66B</td>
<td>4K bits (256 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C, I</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93AA06</td>
<td>1K bits (128 X 8) or (64 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93AA56</td>
<td>2K bits (256 X 8) or (128 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td></td>
<td>93AA66</td>
<td>4K bits (512 X 8) or (256 X 16)</td>
<td>2 MHz</td>
<td>1M</td>
<td>C</td>
<td>8</td>
<td>P, J, SN, SM</td>
</tr>
<tr>
<td>3-Wire (Microwire™)/4-Wire Bus Protocol</td>
<td>93C11</td>
<td>1K bits (128 X 8) or (64 X 16)</td>
<td>1 MHz</td>
<td>1M</td>
<td>100 K</td>
<td>C, I</td>
<td>8</td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
TABLE 4 - ASSP PRODUCT SELECTION GUIDE

<table>
<thead>
<tr>
<th>Device</th>
<th>Interface</th>
<th>Features</th>
<th>Operating Voltage</th>
<th>Temp. Range</th>
<th>Number of Pins</th>
<th>Package Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FIRMWARE PRODUCTS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTA41300</td>
<td>Serial, PS/2</td>
<td>2 Button Mouse, Trackball</td>
<td>3.0V-6.25V</td>
<td>C, I</td>
<td>18</td>
<td>PDIP, SOIC, SSOP</td>
</tr>
<tr>
<td>MTA41120</td>
<td>ADB</td>
<td>2 Button Mouse, Trackball</td>
<td>3.0V-6.25V</td>
<td>C, I</td>
<td>18</td>
<td>PDIP, SOIC, SSOP</td>
</tr>
<tr>
<td>MTA41110</td>
<td>PS/2</td>
<td>2 Button Mouse, Trackball</td>
<td>3.0V-6.25V</td>
<td>C, I</td>
<td>18</td>
<td>PDIP, SOIC, SSOP</td>
</tr>
<tr>
<td><strong>BATTERY DEVICES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTA11200</td>
<td>RS232      or 1-wire</td>
<td>Fuel Gauge for NiCD, NiMH, Lead Acid</td>
<td>3.0V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC, SSOP</td>
</tr>
<tr>
<td><strong>PICSEE® PRODUCTS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTA81010-RC</td>
<td>DC to 4 MHz</td>
<td>512 x 12 EPROM, 1K EEPROM</td>
<td>3.0V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC, JW</td>
</tr>
<tr>
<td>MTA81010-XT</td>
<td>DC to 4 MHz</td>
<td>512 x 12 EPROM, 1K EEPROM</td>
<td>3.0V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC, JW</td>
</tr>
<tr>
<td>MTA8R1010-RC</td>
<td>DC to 4 MHz</td>
<td>512 x 12 ROM, 1K EEPROM</td>
<td>2.5V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC</td>
</tr>
<tr>
<td>MTA8R1010-XT</td>
<td>DC to 4 MHz</td>
<td>512 x 12 ROM, 1K EEPROM</td>
<td>2.5V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC</td>
</tr>
<tr>
<td><strong>LOW-POWER DEVICES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTA81010-LP</td>
<td>DC to 40 KHz</td>
<td>512 x 12 EPROM, 1K EEPROM</td>
<td>2.5V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC</td>
</tr>
<tr>
<td>MTA8R1010-LP</td>
<td>DC to 40 KHz</td>
<td>512 x 12 ROM, 1K EEPROM</td>
<td>2.0V-6.25V</td>
<td>C, I</td>
<td>28</td>
<td>PDIP, SOIC</td>
</tr>
</tbody>
</table>

EASE OF PRODUCTION UTILIZING QUICK TURN PROGRAMMING (QTP) AND SERIALIZED QUICK TURN PROGRAMMING (SQTP)

Recognizing the needs of high volume manufacturing operations, Microchip has developed two programming methodologies which make the OTP products as easy to use in manufacturing as they are efficient in the system development stage.

Quick-Turn-Programming allows factory programming of OTP product prior to delivery to the system manufacturing operation. PIC16/17, EPROM and serial EEPROM products can be automatically programmed with the users program during the final stages of the test operation at Microchip's assembly and test operation in Kaohsiung, Taiwan. This low cost programming step allows the elimination of programming during system manufacturing and essentially allows the user to treat the PIC16/17 and memory products as custom ROM products. With 1 to 4 week lead times on OTP product, the user no longer needs to plan for the extended ROM masking lead-times and masking charges associated with custom ROM products. This capability, combined with the off-the-shelf availability of standard OTP product, insures the user of product availability and the ability to reduce his time-to-market once product development has been completed.

Unique to the 8-Bit Microcontroller market is Microchip's ability to enhance the OTP capability with SQTP. Serialized Quick-Turn-Programming allows for the programming of devices with unique, random or serialized identification codes. As each PIC16/17 device is programmed with the customer's program code, a portion of the program memory space can be programmed with a unique code, accessible from normal program memory, which will allow the user to provide each device with a unique identification. This capability is ideal for embedded systems applications where the transmission of key codes or identification of the device as a node within a network are essential. Taking advantage of this capability allows the system designer to eliminate the requirement for expensive off-chip code implementation using DIP switches or non-volatile memory components. The SQTP offering, available only from Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.
## SECTION 2
### PIC16C5X APPLICATION NOTES

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison of 8-Bit Microcontrollers - AN520</td>
<td>2- 1</td>
</tr>
<tr>
<td>Software Interrupt Techniques - AN514</td>
<td>2- 11</td>
</tr>
<tr>
<td>Software Stack Management - AN527</td>
<td>2- 15</td>
</tr>
<tr>
<td>Power-Up Considerations - AN522</td>
<td>2- 19</td>
</tr>
<tr>
<td>Implementation of an Asynchronous Serial I/O - AN510</td>
<td>2- 23</td>
</tr>
<tr>
<td>Using PIC16C5X as a Smart I²C™ Peripheral - AN541</td>
<td>2- 41</td>
</tr>
<tr>
<td>PLD Replacement - AN511</td>
<td>2- 59</td>
</tr>
<tr>
<td>Analog to Digital Conversion - AN513</td>
<td>2- 79</td>
</tr>
<tr>
<td>Implementing Ohmmeter/Temperature Sensor - AN512</td>
<td>2- 85</td>
</tr>
<tr>
<td>Interfacing to AC Power Lines - AN521</td>
<td>2- 91</td>
</tr>
<tr>
<td>Implementing Wake-Up on Keystroke - AN528</td>
<td>2- 93</td>
</tr>
<tr>
<td>Multiplexing LED Drive and Keypad - AN529</td>
<td>2- 97</td>
</tr>
<tr>
<td>Implementing a Simple Serial Mouse Controller - AN519</td>
<td>2-121</td>
</tr>
<tr>
<td>Intelligent Remote Positioner - AN531</td>
<td>2-133</td>
</tr>
<tr>
<td>Programming PIC16C5X Devices on Data I/O Unisite - AN524</td>
<td>2-149</td>
</tr>
<tr>
<td>Programming PIC16C5X Devices on Logical Devices ALLPRO - AN525</td>
<td>2-153</td>
</tr>
<tr>
<td>Utility Math Routines - AN526</td>
<td>2-155</td>
</tr>
<tr>
<td>Implementing an LCD Controller - AN563</td>
<td>2-215</td>
</tr>
</tbody>
</table>
INTRODUCTION

The PIC16C5X Family of microcontrollers from Microchip Technology, Inc. provides significant execution speed and code-compaction improvement over any other 8-bit microcontroller in its price range.

The superior performance of the PIC16C5X microcontrollers can be attributed primarily to its RISC-like architecture. The PIC16C5X employs Harvard architecture, i.e., has separate program memory space (12-bit wide instructions) and data memory space (8-bit wide data). It also uses a two stage pipelining instruction fetch and execution. All instructions are executed in a single cycle (200 ns @ 20 MHz clock) except for program branches which take two cycles, and there are only 33 instructions to remember.

Separation of program and data space allows the instruction word to be optimized to any size (12-bit wide in case of PIC16C5X). This makes it possible, for example, to load an 8-bit immediate value in one cycle. First, because there is no conflict between instruction fetch and data fetch (as opposed to von Neumann architecture) and secondary because the instruction word is wide enough to hold the 8-bit data.

In the following sections we will compare the PIC16C5X @ 20 MHz with:

- SGS-Thomson ST62 @ 8MHz
- Motorola MC68HC05 @ 4.2 MHz
- Intel 8048/8049 @ 11 MHz
- Zilog Z86CXX @ 12 MHz
- National COP800 @ 20 MHz

Several coding examples will be considered. While the comparisons are not entirely scientific, they will, nevertheless, demonstrate to the reader the relative superior performance of the PIC16C5X. The examples chosen here are used frequently in microcontroller applications.

PACKING BCD

This example will take two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.

<table>
<thead>
<tr>
<th>PIC16C5X</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAPF REGHl,W</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IORWF REGLO</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

0.4 µs

<table>
<thead>
<tr>
<th>COP800</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>X A[B+]</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SWAP A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OR A[B]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X A[B]</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

B is pointing to the higher BCD digit initially. 5 µs

After auto-increment, it points to the lower BCD digit.

<table>
<thead>
<tr>
<th>ST62</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD A, REGHI</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>RLC A</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>RLC A</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>RLC A</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>ADD A, REGLO</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>LD REGLO, A</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

REGHI & REGLO are registers addressable by short direct addressing mode.

45.5 µs

<table>
<thead>
<tr>
<th>Z86CXX</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAP REGHI</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>OR REGHI, REGLO</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

REGHI and REGLO are addressable via the working register addressing mode.

5.33 µs

<table>
<thead>
<tr>
<th>MC68HC05</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA REGHI</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>ROLA</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ROLA</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ROLA</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ADD REGLO</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>STA REGLO</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

10.5 µs

<table>
<thead>
<tr>
<th>8048/8049</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rx</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,Ry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV Ry,A</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Register Rx contains higher BCD digit, 5.45 µs

Ry holds lower BCD digit.
Microcontroller Comparison

LOOP CONTROL
This example is one of simple loop control where a register containing loop count is decremented, tested for zero and if not branched back to the beginning of the loop.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C5X</td>
<td>DECFSZ COUNT 1 1/2 GOTO BEG_LOOP 2 3/2</td>
<td>0.6µs/0.4 µs</td>
</tr>
<tr>
<td>COP800</td>
<td>DRSZ COUNT 1 3 JP BEG_LOOP 2 6</td>
<td>COUNT is Register (RAM F0h-FFh). 6 µs</td>
</tr>
<tr>
<td>ST62</td>
<td>DEC X JRZ BEG_LOOP 1 4 2 6</td>
<td>9.75 µs</td>
</tr>
<tr>
<td>MC68HC05</td>
<td>DECX 1 3 BEQ 2 6</td>
<td>2.86 µs</td>
</tr>
<tr>
<td>Z86CXX</td>
<td>DJNZ COUNT, BEG_LOOP 2 10/12</td>
<td>1.67 µs/2 µs</td>
</tr>
<tr>
<td>8048/8069</td>
<td>DJNZ Rx, BEG_LOOP 2 2</td>
<td>2.73 µs</td>
</tr>
</tbody>
</table>

Bit Test & Branch
This example tests a single bit in a register or a RAM location and makes a conditional branch. We assume that MSB is tested and a branch is to be taken if the bit is set.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C5X</td>
<td>BTFSC REG,7 1 1/2 GOTO NEWADD 2 3/2</td>
<td>0.6 µs/0.4 µs</td>
</tr>
<tr>
<td>COP800</td>
<td>IFBIT 7,[B] 1 1 JP NEWADD 2 3</td>
<td>B points to the memory location under test. 4 µs</td>
</tr>
<tr>
<td>ST62</td>
<td>JRR 7, NEWADD 3 5</td>
<td>8.125 µs</td>
</tr>
<tr>
<td>MC68HC05</td>
<td>BRCLR,7 NEWADD 3 5</td>
<td>2.38 µs</td>
</tr>
<tr>
<td>Z86CXX</td>
<td>BTJRT NEWADD, REG,7 3 16/18</td>
<td>2.67 µs/3.0 µs</td>
</tr>
<tr>
<td>8048/8049</td>
<td>MOV A,@Rx 1 1 ANL A,#80H 2 2 JNZ NEWADD 2 5</td>
<td>Registers R1 is assumed to be pointing to the memory location under test. 6.82 µs</td>
</tr>
</tbody>
</table>
## Shifting Out 8-Bit Data & Clock

We will now consider the task of serially shifting out an 8-bit data and clock. Data and clock outputs are generated under program control by toggling two output pins.

**Data is transmitted on the rising edge of the clock. No attempt is made to make the clock output symmetrical in order to make the code efficient. Data out is guaranteed on the falling edge of the clock. These conditions are satisfactory for most applications.**

### PIC16C5X

<table>
<thead>
<tr>
<th>XMIT</th>
<th>MOV LW 08H</th>
<th>; Bit Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOVWF BITCNT</td>
<td>;</td>
</tr>
<tr>
<td>XM1</td>
<td>BCF PORTB,0</td>
<td>; Data Out Pin</td>
</tr>
<tr>
<td></td>
<td>BCF PORTB,1</td>
<td>; Clock Out Pin</td>
</tr>
<tr>
<td></td>
<td>RRF XDATA</td>
<td>; Rotate Right thru Carry</td>
</tr>
<tr>
<td></td>
<td>BTFSC STATUS,CARRY</td>
<td>; Test Carry Bit</td>
</tr>
<tr>
<td></td>
<td>BSF PORTB,0</td>
<td>; Data Out Pin</td>
</tr>
<tr>
<td></td>
<td>BSF PORTB,1</td>
<td>; Clock Out Pin</td>
</tr>
<tr>
<td></td>
<td>DECSFZ BITCNT</td>
<td>; Decrement Count</td>
</tr>
<tr>
<td></td>
<td>GOTO XM1</td>
<td>; Skip if Zero</td>
</tr>
<tr>
<td></td>
<td>BCF PORTC,1</td>
<td>; Clock</td>
</tr>
</tbody>
</table>

---

### COP800

<table>
<thead>
<tr>
<th>XMIT</th>
<th>LD A.XDATA</th>
<th>; Load Data in Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LD BITCNT #0F</td>
<td>; Load Bit Count</td>
</tr>
<tr>
<td></td>
<td>LD B.#0F</td>
<td>; B Points to PORTL</td>
</tr>
<tr>
<td>XM1</td>
<td>RBIT 0,[B]</td>
<td>; Clock</td>
</tr>
<tr>
<td></td>
<td>RBIT 1,[B]</td>
<td>; Data</td>
</tr>
<tr>
<td></td>
<td>RRCA</td>
<td>; Rotate A Right thru Carry</td>
</tr>
<tr>
<td></td>
<td>IFC</td>
<td>; Test Carry Bit</td>
</tr>
<tr>
<td></td>
<td>SBIT 1,[B]</td>
<td>; Data</td>
</tr>
<tr>
<td></td>
<td>SBIT 0,[B]</td>
<td>; Clock</td>
</tr>
<tr>
<td></td>
<td>DRSZ BITCNT</td>
<td>; Decrement Bit Count</td>
</tr>
<tr>
<td></td>
<td>JP XM1</td>
<td>; and Go Back if ¬0</td>
</tr>
<tr>
<td></td>
<td>RBIT 0,[B]</td>
<td>; Clock</td>
</tr>
</tbody>
</table>

---

### ST62

<table>
<thead>
<tr>
<th>LDI A, #08</th>
<th>; Bit Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD X, A</td>
<td>; Xmit Data</td>
</tr>
<tr>
<td>LD A, W</td>
<td>;</td>
</tr>
<tr>
<td>XM1 RES 0, DBR</td>
<td>; Clock</td>
</tr>
<tr>
<td>RES 1, DBR</td>
<td>; Data</td>
</tr>
<tr>
<td>SLA A</td>
<td>;</td>
</tr>
<tr>
<td>JRCN XM2</td>
<td>;</td>
</tr>
<tr>
<td>XM2 SET 1, DRB</td>
<td>; Data</td>
</tr>
<tr>
<td>SET 0, DRB</td>
<td>; CLK</td>
</tr>
<tr>
<td>DEC X</td>
<td>;</td>
</tr>
<tr>
<td>JRNZ XM1</td>
<td>;</td>
</tr>
<tr>
<td>RES 0, DBR</td>
<td>; Data</td>
</tr>
</tbody>
</table>

---

Transmit time is the same for 00h or FFh: 74 Tcyc = 14.8 µs. Note that there was no need to load the data in the Accumulator (W) since the PIC can operate directing on file registers.

Accumulator A is first loaded with the data word. Transmit time is maximum for data = FFh: 105 Tcyc = 105 µs.

Register W contains the Data word. Transmit time for FFh = 240 cycles = 390 µs.
## Microcontroller Comparison

### SHIFTING OUT 8-BIT DATA AND CLOCK (CONT.)

<table>
<thead>
<tr>
<th>MC68HC05</th>
<th>LDA XDATA ; Load Xmit Data</th>
<th>Xmit 00h</th>
<th>Xmit FFh</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMIT</td>
<td>LDA XDATA ; Load Xmit Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>LDX #08 ; Load Bit Count</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XM1</td>
<td>BCLR, 0,PORTB ; Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>BCLR 1,PORTB ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ROL PORTB ; Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>BCC XM2 ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>BSET 1,PORTB ; Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>BSET 0,PORTB ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XM2</td>
<td>DECX ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>BNE XM1 ; Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>BCLR 0,PORTB ; Data</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Transmit time is maximum for transmitting FFh = 266 cycles = 126.7 µs.

<table>
<thead>
<tr>
<th>Z86CXX</th>
<th>LD COUNT,#8 ; Load Bit Count</th>
<th>Xmit 00h</th>
<th>Xmit FFh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AND P2,#%FC ; Clock, Data</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>XMIT</td>
<td>LD COUNT,#8 ; Load Bit Count</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>AND P2,#%FC ; Clock, Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XM1</td>
<td>RRC XDATA ; Data, Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>RJ NC,XM2 ; Data, Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>OR</td>
<td>P2,#01 ; Data, Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>P2,#02 ; Data, Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XM2</td>
<td>OR COUNT,XM1 ; Clock, Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL</td>
<td>PORT1,#01H ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ</td>
<td>COUNT,XM1 ; Clock, Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>AND</td>
<td>P2,#%FC ; Clock, Data</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Transmit time is maximum for transmitting FFh = 412 cycles = 68.67 µs.

<table>
<thead>
<tr>
<th>8048/8049</th>
<th>MOV A,@R0 ; R0 Points to Data Word</th>
<th>Xmit 00h</th>
<th>Xmit FFh</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMIT</td>
<td>MOV R1,#08H ; Load Bit Count</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ANL PORT1,#0FCH ; Data, Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XMIT</td>
<td>RRC A ; Rotate Right A thru Carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>JC XM2 ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XMIT</td>
<td>ORL PORT1,#01H ; Data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XMIT</td>
<td>ORL PORT1,#02H ; Clock</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL</td>
<td>PORT1,#02H ; Decrement Count</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Transmit time is maximum for transmitting FFh = 91 cycles = 124.1 µs.
Software Timer

Microcontrollers quite often need to implement time delays. Debouncing key input, pulse width modulation, and phase angle control are just a few examples. Implementing a 10 ms time delay loop subroutine will be considered in this section.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PIC16C5X</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELAY MOV LW 41H ;10 ms Delay Loop</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DELAY MOV WWF COUNT2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DELAY CLR COUNT1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOOP INC FSZ COUNT1 ;This Inner Loop will be Executed 256 Times</td>
<td>1</td>
<td>2/1</td>
</tr>
<tr>
<td>LOOP GOTO LOOP</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LOOP GOTO LOOP</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LOOP RET</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Execution time for the routine = 5 + (255 x 3 + 5) • 65 = 20025 Tcyc = 10.011 ms. The PIC16C5X can implement delay times very precisely (when necessary) because of its fine instruction cycle resolution.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COP600</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELAY LD COUNT1,#0BH ;10 ms Delay Loop</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>DELAY LD B,#0EH</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOOP DRSZ B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOOP JP LOOP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOOP JP LOOP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOOP RET</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Execution time for the routine = (6N1 + 6) N1 + 9 cycles. Here N1 = 0BH and N2 = 0EH, which gives us: 999 Tcyc = 9.99 ms.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ST62</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDI A,#FF</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>LDI A,#04</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>LOOP DEC X</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>LOOP JP LOOP</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LOOP JRNZ LOOP</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Execution time for the subroutine = (6N1 + 6) N2 + 16 cycles, where N1 = FFh, N2 = 04 gives us 10.01 ms.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MC68HC05</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELAY LDX $2D ;10 ms Delay Loop</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DELAY LDX $5C</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LOOP DECA BNE</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>LOOP DECX BNE</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>LOOP RTS</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Execution time for the subroutine = (5 x N1 + 5) N2 + 10, with N1 = 2DH, N2 = 5CH, time delay = 10.081 ms.
## SOFTWARE TIMER (CONT.)

### Z86CXX

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY</td>
<td>LD COUNT1,#%61 ;10 ms Delay Loop</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>LD COUNT2,#%33 ;</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>LOOP</td>
<td>DJNZ COUNT1,LOOP</td>
<td>2</td>
<td>10/12</td>
</tr>
<tr>
<td></td>
<td>DJNZ COUNT2,LOOP</td>
<td>2</td>
<td>10/12</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td>1</td>
<td>14</td>
</tr>
</tbody>
</table>

Total execution time = (12N1 + 10) N2, with N1 = 61H, N2 = 33H, time delay = 59976 cycles = 9.979 ms.

### 8048/8049

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Byte/Words</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY</td>
<td>MOV COUNT1,#13H ;10 ms Delay Loop</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>LOOP1 MOV COUNT2,#AFH ;</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>LOOP2 DJNZ COUNT2,LOOP2 ;</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>DJNZ COUNT1,LOOP1 ;</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

Execution time for the subroutine = (2N1 + 4) N2 + 4 cycles. Here N1 = 13H, N2 = AFH, which gives us: 7354 cycles = 10.028 ms.
SUMMARY

Table 1 summarizes code sizes for different microcontrollers. The overall relative code size number is an average of the individual relative code sizes. Given that the PIC16C5X’s program word size is 12-bit, whereas all the other microcontrollers have 8-bit program memory, a compaction of 1.5 µs is expected. Clearly, the PIC16C5X meets this compaction (except for the COP800) and exceeds in most comparisons.

Table 2 summarizes relative execution speed. The overall speed is an average of relative speed numbers. For example, the COP800 will, on an average, exhibit 27% of the code execution speed of a PIC16C5X. In other words, the PIC16C5X will be $1/0.27 = 3.7$ times faster than a COP800 on an average.

### TABLE 1 - COMPARISON OF CODE EFFICIENCY*

<table>
<thead>
<tr>
<th>Device</th>
<th>Packing BCD</th>
<th>Loop Control</th>
<th>Bit Test &amp; Branch</th>
<th>8-Bit Sync Transmission</th>
<th>10 ms Software Timer</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>COP800</td>
<td>4 2.00</td>
<td>2 1.00</td>
<td>2 1.00</td>
<td>16 1.46</td>
<td>8 1.00</td>
<td>1.29</td>
</tr>
<tr>
<td>ST62</td>
<td>10 5.00</td>
<td>2 1.00</td>
<td>3 1.50</td>
<td>19 1.73</td>
<td>10 1.25</td>
<td>2.10</td>
</tr>
<tr>
<td>MC68HC05</td>
<td>10 5.00</td>
<td>3 1.50</td>
<td>3 1.50</td>
<td>20 1.82</td>
<td>11 1.38</td>
<td>2.24</td>
</tr>
<tr>
<td>Z86CXX</td>
<td>4 2.00</td>
<td>2 1.00</td>
<td>3 1.50</td>
<td>21 1.91</td>
<td>9 1.125</td>
<td>1.51</td>
</tr>
<tr>
<td>8048/8049</td>
<td>4 2.00</td>
<td>2 1.00</td>
<td>5 2.51</td>
<td>14 1.28</td>
<td>9 1.13</td>
<td>1.58</td>
</tr>
<tr>
<td>PIC16C5X @ 8 MHz</td>
<td>2 2 2</td>
<td>11</td>
<td>8 1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* In each box, the top number is the number of program memory locations required to code the application. The bottom number is relative code size compared to the PIC16C5X:

# program memory locations for other microcontroller
# program memory locations for the PIC16C5X
FIGURE 1 - CODE SIZE COMPARISON

- PIC16C5X: 1.00
- COP800: 1.29
- ST92: 2.10
- MC8HC05: 2.24
- Z86CXX: 1.51
- 8048: 1.58

RELATIVE CODE SIZE

© 1993 Microchip Technology Inc.
### TABLE 2 - COMPARISON OF EXECUTION SPEED

<table>
<thead>
<tr>
<th>Device</th>
<th>Packing BCD</th>
<th>Loop Control</th>
<th>Bit Test &amp; Branch</th>
<th>8-Bit Sync Transmission</th>
<th>10 ms Software Timer</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>COP800</td>
<td>5 µs</td>
<td>6 µs</td>
<td>4 µs</td>
<td>105 µs</td>
<td>–</td>
<td>0.108</td>
</tr>
<tr>
<td>@ 20 MHz</td>
<td>0.08</td>
<td>0.0832</td>
<td>0.1252</td>
<td>0.1408</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST62</td>
<td>45.5 µs</td>
<td>9.75 µs</td>
<td>8.125 µs</td>
<td>390 µs</td>
<td>–</td>
<td>0.0455</td>
</tr>
<tr>
<td>@ 8 MHz</td>
<td>0.0088</td>
<td>0.0615</td>
<td>0.0738</td>
<td>0.0339</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC68HC05</td>
<td>10.05 µs</td>
<td>2.86 µs</td>
<td>2.38 µs</td>
<td>126.7 µs</td>
<td>–</td>
<td>0.136</td>
</tr>
<tr>
<td>@ 4.2 MHz</td>
<td>0.038</td>
<td>0.1748</td>
<td>0.21</td>
<td>0.1168</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z86CXX</td>
<td>2.33 µs</td>
<td>1.835 µs</td>
<td>2.835 µs</td>
<td>68.67 µs</td>
<td>–</td>
<td>0.212</td>
</tr>
<tr>
<td>@ 12 MHz</td>
<td>0.172</td>
<td>0.272</td>
<td>0.176</td>
<td>0.224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8048/8049</td>
<td>5.45 µs</td>
<td>2.73 µs</td>
<td>6.82 µs</td>
<td>124.1 µs</td>
<td>–</td>
<td>0.112</td>
</tr>
<tr>
<td>@ 11 MHz</td>
<td>0.0732</td>
<td>0.1824</td>
<td>0.0732</td>
<td>0.1196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC16C5X</td>
<td>0.4 µs</td>
<td>0.6/0.4 µs</td>
<td>0.6/0.4 µs</td>
<td>14.8 µs</td>
<td>–</td>
<td>1.00</td>
</tr>
<tr>
<td>@ 20 MHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* In each box, the top number is the time required to execute the example code, while the bottom number is a measure of relative performance compared to the PIC16C5X:

\[
\frac{\text{time required to execute code by the PIC16C5X}}{\text{time required to execute code by other microcontroller}}
\]

### FIGURE 2 - EXECUTION SPEED COMPARISON
INTRODUCTION

This application note describes a unique method for implementing interrupts in software on the PIC16C5X series of microcontrollers. The method takes advantage of the PIC16C5X's architecture which allows changing the program counter under software control. Up to 8 interrupt lines are possible, but the practical limit for simple code generation is 6 interrupts, or 64 possible input conditions. The interrupt detection time is under software control and standard I/O pins are used as the interrupt lines.

THEORY OF OPERATION

SOFTWARE POLLING OF I/O LINES REPLACES HARDWARE INTERRUPT

The interrupt conditions are determined by detecting changes on the I/O lines that have been selected to be the interrupt lines. These changes are used to create a jump table that allows a different program response to each interrupt condition. The interrupt response time is under software control and can be as short as ten to twenty microseconds, depending on main program and interrupt subroutine program length.

CREATING THE INTERRUPT SUBROUTINE JUMP TABLE

Each I/O condition may have its own unique subroutine to respond to changes on the interrupt lines. Direct access to these routines is achieved by using the PIC16C5X's ability to change the program counter under software control. Here is an example of how two I/O lines may be polled:

```
MOVF CONDTN, W ;LOAD I/O CONDITION INTO W :REGISTER
ANDLW 3 ;MASK OFF TOP 6 BITS
ADDWF 2, 1 ;ADD INPUT TO PROGRAM COUNTER :TO CREATE JUMP TABLE
GOTO MAIN ;FOR NO CHANGE GO TO MAIN :PROGRAM
GOTO INT1 ;FOR CHANGE IN BIT 0 GOTO INT1
GOTO INT2 ;FOR CHANGE IN BIT 1 GOTO INT2
GOTO INT3 ;FOR BOTH CHANGE GOTO INT3
```

The changes to the I/O lines have been used to create a two bit number that is added to the program counter. The GOTO that is executed depends on the new program counter address.

CREATING CONSTANT TIME POLLING

In most applications requiring interrupts, it is important to poll the interrupt lines at fixed time intervals, usually only a few microseconds in length. Two techniques may be used on the PIC16C5X to achieve this. They are dividing the main program into multiple sections and implementing an elapsed time counter (see flow chart). Both of these techniques use the same program jump table concept that was described above. First, the main program is divided into several sections based on the desired I/O polling time. When MAIN is called a branch register is added to the program counter. This determines which section of main code should be executed next. At the end of execution the branch register is decremented so the next section of code will be executed after the next polling. If the branch register is zero then the number of sections of main code is added into it to start the main program over again.

An elapsed time counter can be implemented using the RTCC counter. At the beginning of I/O polling the RTCC register is cleared. It then starts counting the instruction cycles. Then after the main program subsection has been executed, the RTCC register is subtracted from the desired polling time. This determines how many instructions need to be executed before the next polling. A jump table is then created to execute these instructions before the next polling. An example is shown below. This example assumes from zero to 15 additional instruction cycles are needed. Actual numbers need to be computed for each individual application.

```
MOVLW POLL ;POLL=DESIRED POLL CYCLES - 15
SUBWF RTCC, W ;DETERMINE HOW MUCH TIME TO WAIT
ADDWF 2, 1 ;ADD WAIT TIME TO PROGRAM COUNTER
NOP ;15 ADDITIONAL INSTRUCTION CYCLES
NOP ;1 ADDITIONAL INSTRUCTION CYCLES
GOTO START ;0 ADDITIONAL INSTRUCTION CYCLES
```
Software Interrupt Techniques

For example, if the desired instruction time is 50 cycles and the subsection we just executed has consumed a total of 40 instruction cycles (including all overhead cycles) the value of

\[ \text{RTCC(40)} - \text{POLL(50-15(35))} = 5 \]

will be added to the program counter. The program will then jump to the sixth NOP. That NOP plus the 9 following it will be executed for a total of ten more instruction cycles. Note that the final GOTO has two instruction cycles and these must be included in the program overhead.

Example

The following example (see flow chart and code) is the core program for the software interrupt technique described above. This program assumes four interrupt conditions, four main program sections and an eight additional elapsed time instructions.

FIGURE 1 - SOFTWARE INTERRUPT FLOW CHART

![Software Interrupt Flow Chart](image)
Software Interrupt Techniques

APPENDIX A:

MPASM B0.54

LIST
P=16C54

;SOFTWARE INTERRUPT APPLICATIONS
;BRANCH IS MAIN PROGRAM REGISTER

0000 0000 BRANCH EQU 8
0001 0009 CNDTN EQU 9
0002 000A IO EQU 0A
0003 000B TEMP EQU 0B
0004 0069 SETUP CLRF CNDTN
0005 0028 MOVWF BRANCH ;FOUR MAIN PROGRAM SECTIONS
0006 002C OPTION ;SET RTCC TO ONE COUNT PER INSTRUCTION CYCLE
0007 0061 START CLRF ;CLEAR RTCC REGISTER
0008 0026 MOVF 6, W ;READ I/O
0009 002A MOVWF IO ;THIS SECTION OF CODE CALCULATES THE
0010 0009 MOVWF CNDTN,W ;JUMP TABLE, ANY INPUT THAT CHANGES FROM
0011 000A MOVF CNDTN,W ;A ZERO TO A ONE IS CONSIDERED AN INTERRUPT.
0012 000B SUBWF TEMP, W ;THE EQUATION IS:
0013 000C MOVF IO, W ;(IO + CNDTN) - CNDTN = INTERRUPT
0014 000D MOVF TEMP, W ;WHERE IO IS CURRENT INPUT AND
0015 000E ANDLW 3 ;CNDTN IS PREVIOUS INPUT.
0016 0010 ADDWF 2, W ;MASK OFF TOP 6 BITS
0017 0011 GOTO MAIN ;ADD INPUT TO PC TO CREATE JUMP TABLE
0018 GOTO INT1 ;FOR INPUT=00
0019 GOTO INT2 ;FOR INPUT=01
0020 GOTO INT3 ;FOR INPUT=10
0021 GOTO INT4 ;FOR INPUT=11
0022 0000 INT1 NOP ;INTERRUPT LINE 1 CODE
0023 0005 GOTO START
0024 0000 INT2 NOP ;INTERRUPT LINE 2 CODE
0025 0005 GOTO START
0026 0000 INT3 NOP ;INTERRUPT LINES 1 AND 2 CODE
0027 0005 GOTO START
0028 0000 INT4 NOP ;MAIN PROGRAM CODE BANK ONE
0029 0005 GOTO BRNCHK
0030 0000 MAIN MOVF BRANCH, W
0031 0005 GOTO BRNCHK
0032 0000 MAIN2 NOP ;MAIN PROGRAM CODE SECTION TWO
0033 0005 GOTO BRNCHK
0034 0000 MAIN3 NOP ;MAIN PROGRAM CODE SECTION THREE
0035 0005 GOTO BRNCHK
0036 0000 MAIN4 NOP ;MAIN PROGRAM CODE SECTION FOUR
0037 0005 GOTO BRNCHK
0038 0000 MAIN5 DECSZ BRANCH, 1 ;DECREMENT BRANCH REGISTER AND CHECK FOR ZERO
0039 0010 MOVWF BRANCH ;RELOAD BRANCH WITH 4 AT END OF MAIN
003A 0015 TIMCHK MOVILW D'41' ;CHECK TO SEE IF RTCC HAS REACHED 50 (50-7)
Software Interrupt Techniques

002F 0081  SUBWF 1, W ; DETERMINE WAIT TIME
0030 01E2  ADDWF 2, 1 ; ADD WAIT TIME TO PC
0031 0000  NOP
0032 0000  NOP
0033 0000  NOP
0034 0000  NOP
0035 0000  NOP
0036 0000  NOP
0037 0000  NOP
0038 0A05  GOTO START
            END

Errors : 0
Warnings : 0
INTRODUCTION

The PIC16C5X has a stack which is only 2 deep, as a result of which only two nested calls can be made (i.e., only one call within a call routine). If more than two levels of subroutine nesting is required, the following Ap Note can be used to implement a stack manager to handle the flow of the calls.

Note: Since the amount of RAM on the PIC16CXX is limited, it would be prudent to determine the maximum number of nested calls which have to be made in a program and define the stack length appropriately.

IMPLEMENTATION

This Application Brief implements a 5 deep stack, so 5 nested calls can be made without overflowing the stack. NCALL is defined as a MACRO which will be used instead of the mnemonic CALL, when a subroutine call is made. The NCALL routine, “pushes” the return PC value on the “stack” and then executes the called subroutine. At the end of the subroutine, instead of using the RETLW k instruction, a GOTO RETURN is executed, where RETURN is a routine which “pops” the return PC value from the “stack” and resumes the normal flow of the program.

Notes:

Since Software Stack Management utilizes the FSR register, and indirect addressing, the user should restore the “original” values to the FSR register if it is utilized elsewhere in the program.

Author: Stanley D'Souza
Logic Products Division
Software Stack Management

sm.asm:
Routine, demonstrating how to implement a stack
manager capable of handling more than 2
subsequent subroutine calls.
Note: Since this is a demo, NOP has been used
where normally the body of the subroutine would
reside.

```asm
LIST P-16C54
PC EQU 2
FSR EQU 4

ORG 8
STACK RES 5 ; define stack size = 5.

ORG 01FF
GOTO START

ORG 0

INIT MOVLW STACK ; load "stack" as indirect pointer
MOVWF FSR ;
GOTO START ;

;**************
;define NCALL as a MACRO used instead of the
;mnemonic CALL.
;
NCALL MACRO LABEL
MOVF PC,W ; save PC on "stack"
MOVWF 0 ;
INCF FSR ; Inc. "stack" pointer.
GOTO LABEL ; jump to routine
ENDM

;return from subroutine NCALL
;
RET DECF FSR ; point to last "stack" location
MOVWF 3 ; add 3 and output value from FSR
ADDWF 0,W ;
MOVF PC ; load in PC as next executable
instruction

;**************

START NOP
NCALL TOM
MOVF PC,W ; save PC on "stack"
MOVWF 0 ;
INCF FSR ; Inc. "stack" pointer.
GOTO TOM ; jump to routine

NOP ; body of main routine
NOP ;
SLEEP

TOM NOP
NCALL DICK
MOVF PC,W ; save PC on "stack"
MOVWF 0 ;
INCF FSR ; Inc. "stack" pointer.
GOTO DICK ; jump to routine
```
Software Stack Management

0014 0000  NOP ;body of routine TOM
0015 0A03  GOTO RET

0016 0000  ; DICK NOP
             NCALL HARRY
0017 0202  MOVF PC,W ;save PC on "stack"
0018 0020  MOVWF 0 ;
0019 02A4  INCF PSR ;Inc. "stack" pointer.
001A 0A1D  GOTO HARRY ;jump to routine

001B 0000  NOP ;body of routine DICK
001C 0A03  GOTO RET

001D 0000  HARRY NOP ;body of routine HARRY
001E 0000  NOP ;
001F 0A03  GOTO RET

END

Errors : 0
Warnings : 0
INTRODUCTION

When powering up all microcontrollers, it is necessary for the power supply voltage to traverse voltage ranges where the device is not guaranteed to operate before the power supply voltage reaches its final state. Since some circuits on the device (logic) will start operating at voltage levels lower than other circuits on the chip (memory), the device may power-up in an unknown state. To guarantee that the device starts up in a known state, it is necessary that it contain a power-up reset circuit. PIC16C5X microcontrollers are equipped with on-chip power-on reset circuitry, which eliminates the need for external reset logic. This circuit will function in most power-up situations where Vcc rise time is fast enough (50 ms or less). This application note describes the typical power-up sequence for PIC16C5X microcontrollers. Methods of assuring reset on power-up and after a brownout are discussed and simple, low-cost external solutions are discussed for power-up situations where the PIC16C5X’s internal circuitry cannot provide the reset.

POWER-UP SEQUENCE

The PIC16C5X incorporates complex power-on reset (POR) circuitry on-chip which provides solid, reliable internal chip reset for most power-up situations. To use this feature, the user merely needs to tie MCLR to Vdd. A simplified block diagram of the on-chip reset circuitry is shown in Figure 1. On power-up, the reset latch and the start-up timer are reset to appropriate states by the power-on reset (POR). The start-up timer will begin counting once it detects MCLR to be high (i.e., external chip reset goes inactive). After the time-out period, which is typically 18 ms long, the timer will reset the reset latch and thus end the on-chip reset signal.

Figures 2 and 3 are two power-up situations with relative fast rise time on Vdd. In Figure 1, Vdd is stable when MCLR is brought high (i.e., reset pulse is being provided by external source). The chip actually comes out reset about t_{osT} ms after that, where t_{osT} = oscillator start-up timer. (The timer is called oscillator start-up timer because the time-out was incorporated primarily to allow the crystal oscillator to stabilize on power-up.) In Figure 3, the MCLR and Vdd are tied together and clearly the on-chip reset mechanism is being utilized. The Vdd is stable before the start-up timer expires and there is no problem with proper reset.

Figure 4, where Vdd rise time is much greater than t_{osT} (typically 18 ms) clearly is the potentially problematic situation. The POR (power-on reset) pulse comes when Vdd is about 1.5V. Most CMOS logic, including the start-up timer starts functioning between 1.5V to 2.0V. When the start-up timer starts times out, the chip reset is ended and the chip attempts to execute. If by this time the Vdd has reached Vdd MIN value, then all circuits are guaranteed to function correctly and power-up reset is successful. If, however, the Vdd slope was too slow and had not reached Vdd MIN, then the chip may or may not function properly.

FIGURE 1 - PIC16C5X INTERNAL RESET CIRCUIT
Power-Up Considerations

FIGURE 2 - EXTERNAL RESET PULSE

![Diagram of External Reset Pulse]

FIGURE 3 - INTERNAL RESET (VDD AND MCLR TIED TOGETHER)

![Diagram of Internal Reset]

FIGURE 4 - INTERNAL RESET (VDD AND MCLR TIED TOGETHER): SLOW VDD RISE TIME

![Diagram of SLOW VDD RISE TIME]

When VDD rises slowly, the internal time-out period expires long before VDD has reached its final value. In this example, the chip will reset properly if, and only if, V1 ≥ VDDMIN.
EXTERNAL POWER-ON RESET CIRCUIT

To use power supplies with slow rise times it is necessary to use an external power-on reset circuit such as the one shown in Figure 5. This circuit uses an external RC to generate the reset pulse. The time constant of the RC should be long enough to guarantee that the reset pulse is still present until VDD has reached VDD min. R should be 40K or less to guarantee that the MCLR will pull to within 0.2 volts of VDD. (since the leakage spec on MCLR is ±5 µA, a resistor larger than 40K may cause input high voltage on this pin to be less than VDD − 0.2V, the required spec). The diode D is used to rapidly discharge the capacitor on power-down. This is very important as a power-up reset pulse is needed after a short power-down (less than the time constant of RC) or after a power spike. The resistor R1 protects against high current flowing into MCLR pin from fully charged capacitor C in the event MCLR pin breakdown is induced through ESD or EOS. The circuit, however, does not protect against brown-out situations where the power does not drop to zero, but merely dips below VDD MIN. In such a situation, voltage at the MCLR pin will not go low enough (i.e., below V1L) to guarantee a reset pulse. The following section presents an example circuit to protect against such brown-outs.

FIGURE 5 - EXTERNAL POWER-ON RESET CIRCUIT

BROWNOUT PROTECTION

In many applications it is necessary to guarantee a reset pulse whenever VDD is less than VDD min. This can be accomplished using a brownout protection circuit such as the one shown in Figure 6. This is a simple circuit that causes a reset pulse whenever VDD drops below the zener diode voltage plus the Vbe of Q1. A 3.3 volt zener will produce a reset pulse whenever VDD drops below about 4 volts. This circuit has a typical accuracy of about ±100 mV. A less expensive, albeit less precise, brown-out circuit is shown in Figure 7. Transistor Q1 turns off when Vbe = VDD(R1/(R1+R2)) falls below 0.7 V allowing R3 to pull down MCLR input.

FIGURE 6 - BROWNOUT PROTECTION CIRCUIT

FIGURE 7 - BROWNOUT PROTECTION CIRCUIT

Author: Sumit Mitra
Logic Products Division
INTRODUCTION
The PIC16C5X series from Microchip Technology, Inc., are 8 bit, high speed EPROM based microcontrollers. This application note describes the implementation of an Asynchronous serial I/O using Microchip’s PIC16C5X series of high speed 8 bit microcontrollers. These EPROM based microcontrollers can operate at very high speeds with a minimum of 200 ns cycle time @ 20 MHz input clock. Many microcontroller applications require chip to chip serial data communications. Since the PIC16C5X series have no on chip serial ports, serial communication has to be performed in software. For many cost-sensitive high volume applications, implementation of a serial I/O through software provides a more cost effective solution than dedicated logic. This application note provides code for PIC16C5X to simulate a serial port using 2 I/O Pins (one as input for reception and the other as output for transmission).

IMPLEMENTATION
Two programs are provided in this application note. One program simulates a full duplex RS-232 communication and the other provides implementation of half duplex communication. Using Half-Duplex, rates up to 19200 baud can be implemented using an 8 MHz input clock. In case of Full-Duplex, the software can handle up to 9600 baud at 8 MHz and 19200 baud at 20 MHz, one or two stop bits, 8 or 7 data bits, No Parity and can transmit or receive with either LSB first (normal mode) or MSB first (CODEC like mode). It should be noted that the higher the input clock the better the resolution. These options should be set up during assembly time and not during run time. The user simply has to change the header file for the required communication options. The software does not provide any handshaking protocols. With minor modifications, the user may incorporate software handshaking using XON/XOFF. To implement hardware handshaking, an additional 2 digital I/O Pins may be used as RTS (ready to send) and CTS (clear to send) lines.

Figure 1 shows a flow chart for serial transmission and Figure 2 shows a flow chart for reception. The flowcharts show cases for transmission/reception with LSB first and 8 data bits. For reception, the data receive pin, DR, is polled approximately every 8/2 seconds (52 µs in case of 9600 baud) to detect the start bit, where B is the time duration of one bit (B = 1/Baud). If a start bit is found, then the first data bit is checked for after 1.25B seconds. From then on, the other data bits are checked every B seconds (104 µs in case of 9600 baud).

In the case of transmission, first a start bit is sent by setting the transmit data pin, DX to zero for 8 seconds, and from then on the DX pin is set/cleared corresponding to the data bit every B seconds. Assembly language code corresponding to the following flowcharts are given in Figures 3 and 4.
FIGURE 1 - TRANSMISSION FLOW CHART

Input
Load Xmt Reg
Set X-Count = 8
Set DX Pin = 0
Delay
Right Shift Xmt Reg

Is Carry = 1? NO
Set DX Pin = 0
YES
Set DX Pin = 1
X-Count = X-Count - 1

Is X-Count = 0? NO
Transmission Over
YES

FIGURE 2 - RECEPTION FLOW CHART

Input
Test Pin DR

Is DR = 0? NO
YES Start Bit Detected
Set R-Count = 8
Set Rcv Reg = 0
Delay
Clear Carry bit
Right Shift Rcv Reg
Test Pin DR

Is DR = 1? NO
YES
Set MSB of Rcv Reg
R-Count = R-Count - 1

Is R-Count = 0? NO
STOP
FIGURE 3 - TRANSMIT ASSEMBLY CODE (CORRESPONDING TO FIGURE 1)

```assembly
;**************************** Transmitter*****************************
Xmtr movlw 8 ; Assume XmtReg contains data to be Xmted
        movwf XCount ; 8 data bits
        bcf Port_A,DX ; Send Start Bit
X_next call Delay ; Delay for B/2 Seconds
        rrf XmtReg
        btfsc STATUS,CARRY ; Test the bit to be transmitted
        bsf Port_A,DX ; Bit is a one
        btfss STATUS,CARRY
        bcf Port_A,DX ; Bit is zero
        decfsz Count ; If count = 0, then transmit a stop bit
        goto X_next ; transmit next bit
X_Stop call Delay
        bsf Port_A,DX ; Send Stop Bit
X_Over goto X_Over
```

FIGURE 4 - RECEIVE ASSEMBLY CODE (CORRESPONDING TO FIGURE 2)

```assembly
;**************************** Receiver *****************************

Rcvr btfsc Port_A,DR Test for Start Bit
        goto Rcvr ; Start Bit not found
        movlw Start Bit Detected
        movwf RCount 8 Data Bits
        clrf Rev Reg Receive Data Register
R_next call Delay ; Delay for B/2 Seconds, B=Time duration of 1 Bit
        bcf STATUS,CARRY ; Clear CARRY bit
        rrf RevReg ; to set if MSB first or LSB first
        btfsc Port_A,DR ; Is the bit a zero or one ?
        bsf RcvReg,MSB ; Bit is a one
        call Delay
        decfsz RCount
        goto R_next ; Reception done
```

The software is organized such that the communication software acts as a Real Time Operating System (RTOS) which gives control to the User routine for a certain time interval. After this predetermined time slot, the user must give back the control to the Operating System. This is true only in the case of full-duplex implementation. Timing considerations are such that the user gets control for approximately half the time of the bit rate and the rest of the half time is used up by the Operating System (and software delays). Please refer to Table 1 for the delay constants and the time the User gets at 8 MHz input clock. Delay constants and the time that the User gets at 20 MHz and 4 MHz input clock speeds are given in the source code listing of the full duplex routine. At frequencies other than 4, 8, or 20 MHz, the delay constants and the time that the User gets can be computed from the equations given in Figure 6.

FIGURE 5 - FULL DUPLEX BLOCK DIAGRAM

```
```

© 1993 Microchip Technology Inc.
Asynchronous Serial I/O

FIGURE 6 - EQUATIONS FOR DELAY CONSTANTS

Baud_Cycles = Clkout/Baud ;  
User_time = Baud_Cycles • (float) 0.5 ;  
K0 = (1.25 • Baud_Cycles - 2.0 • User_time - 89)/3.0 ; IF (K0 < 0)  
K0 = 0.0 ;  
User_time = 0.50 • (1.25 • Baud_Cycles - 89.0) ;  
K1 = (1.25 • Baud_Cycles - 18 - User_time - 59.0 - 3 • K0 )/3.0 ;  
K2 = (Baud_Cycles - User_time - 41.0 - 3 • K0 )/3.0 ;  
K3 = (Baud_Cycles - User_time - 61.0 - 3 • K0 )/3.0 ;  
K4 = (Baud_Cycles - User_time - 55.0 - 3 • K0 )/3.0 + 1.0 ;  
K5 = 0.0 ;  
K6 = 0.0 ;  
K7 = (1.25 • Baud_Cycles - User_time - 39.0 - 3 • K0 )/3.0 ;

TABLE 1 - DELAY CONSTANTS AT 8 MHz INPUT CLOCK

<table>
<thead>
<tr>
<th>Constant</th>
<th>19200</th>
<th>9600</th>
<th>4800</th>
<th>2400</th>
<th>1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>K0</td>
<td>-0</td>
<td>5</td>
<td>39</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>K1</td>
<td>-39</td>
<td>80</td>
<td>150</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>K2</td>
<td>-27</td>
<td>51</td>
<td>86</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>K3</td>
<td>-21</td>
<td>44</td>
<td>80</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>K4</td>
<td>-23</td>
<td>46</td>
<td>82</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>K5</td>
<td>-24</td>
<td>47</td>
<td>83</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>K6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>K7</td>
<td>45</td>
<td>86</td>
<td>156</td>
<td>295</td>
<td></td>
</tr>
<tr>
<td>User Cycles</td>
<td>-86</td>
<td>208</td>
<td>416</td>
<td>832</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 2 - DELAY CONSTANTS AT 20 MHz INPUT CLOCK

<table>
<thead>
<tr>
<th>Constant</th>
<th>19200</th>
<th>9600</th>
<th>4800</th>
<th>2400</th>
<th>1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>K0</td>
<td>0</td>
<td>13</td>
<td>57</td>
<td>143</td>
<td>317</td>
</tr>
<tr>
<td>K1</td>
<td>49</td>
<td>98</td>
<td>184</td>
<td>358</td>
<td>705</td>
</tr>
<tr>
<td>K2</td>
<td>34</td>
<td>60</td>
<td>103</td>
<td>191</td>
<td>364</td>
</tr>
<tr>
<td>K3</td>
<td>27</td>
<td>53</td>
<td>96</td>
<td>184</td>
<td>357</td>
</tr>
<tr>
<td>K4</td>
<td>29</td>
<td>55</td>
<td>98</td>
<td>186</td>
<td>359</td>
</tr>
<tr>
<td>K5</td>
<td>30</td>
<td>56</td>
<td>99</td>
<td>187</td>
<td>360</td>
</tr>
<tr>
<td>K6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>K7</td>
<td>56</td>
<td>104</td>
<td>190</td>
<td>365</td>
<td>712</td>
</tr>
<tr>
<td>User Cycles</td>
<td>118</td>
<td>260</td>
<td>521</td>
<td>1042</td>
<td>2083</td>
</tr>
</tbody>
</table>

For example, if the baud rate selected is 9600 bps (@ 8 MHz), then the total time frame for one bit is approximately 104 μs. Out of this 104 μs, 61 μs is used by the Operating System and the other 43 μs is available to the User. It is the User's responsibility to return control to the Operating System exactly after the time specified in Table 1. For very accurate timing (with resolution up to one clock cycle) the User may set up the RTCC timer with the Prescaler option for calculating the real time. With RTCC set to increment on internal CLKOUT (500 ns @ 8 MHz CLKIN) and the prescaler assigned to it, very accurate and long timing delay loops may be assigned. This method of attaining accurate delay loops is not used in the RS232 code (RTOS), so that the RTCC is available to the User for other important functions. If the RTCC is not used for other functions, the User may modify the code to replace the software delay loops, by counting the RTCC. For an example of using this method of counting exact timing delays, refer to the “User” routine in Full-Duplex code (Appendix B).

The software uses the minimal processor resources. Only 6 data RAM locations (File Registers) are used. The RTOS uses one level of stack, but it is freed once the control is given back to the user. The watchdog timer and RTCC are not used. The user should clear the watchdog timer at regular intervals, if the WDT is enabled.

The usage of the program is described below. The user should branch to location "Op_Sys" exactly after time as specified in Table 1 or as computed from Equations in Fig. 6. Whereas, the transmission is totally under User control, the Reception is under the control of the Operating System. As long as the user does not set the X_flag, no transmission occurs. On the other hand the Operating System is constantly looking for a start bit and the user should not modify either R_done flag or RcvReg.

TRANSMISSION

Transmit Data is output on DX pin (Bit 0 of Port_A). In the user routine, the user should load the data to be transmitted in the XmtReg and Set the X_flag (bsf FlagRX,X_flag). This flag gets cleared after the transmission. The user should check this flag (X_flag) to see if transmission is in progress. Modifying XmtReg when X_flag is set will transmit erroneous data.

RECEPTION

Data is received on pin DR (Bit 1 of Port_A). The User should constantly check the “R_done” flag to see if reception is over. If the reception is in progress, R_flag is set. If the reception is over, “R_done” flag is set to 1. The “R_done” flag gets reset to zero when a next start bit is detected. The user should constantly check the R_done flag, and if SET, then the received word is in Register “RcvReg”. This register gets cleared when a new start bit is detected. It is recommended that the receive register RcvReg be copied to another register after R_done flag is set. The R_done flag also gets cleared when the next start bit is detected.

The User may modify the code to implement an N deep buffer (limited to the number of Data RAM locations available) for receive. Also, if receiving at high speeds, and if the N deep buffer is full, an XOFF signal (HEX 13) may be transmitted. When ready for receiving more data, an XON signal (HEX 11) should be transmitted.

SUMMARY

PIC16C5X family of microcontrollers allow users to implement half or full duplex RS232 communication.
APPENDIX A: ASSEMBLY LANGUAGE FOR HALF DUPLEX

LIST P=16C54, C=80, T=ON

INCLUDE "PICREG.H"

;************************** PIC16C5X Header ***************

SECTION "F3"

; Define Reset Vectors

PIC54 equ 1FFH
PIC55 equ 1FFH
PIC56 equ 3FFH
PIC57 equ 7FFH

; Reset Vector Assignments

RTCC equ 1h
PC equ 2h
STATUS equ 3h
FSR equ 4h

; I/O Port Assignments

Port_A equ 5h
Port_B equ 6h
Port_C equ 7h

; STATUS REG. Bits

CARRY equ 0h
C equ 0h
DCARRY equ 1h
DC equ 1h
Z_bit equ 2h
Z equ 2h
P_DOWN equ 3h
PD equ 3h
T_OUT equ 4h
TO equ 4h
PA0 equ 5h
PA1 equ 6h
PA2 equ 7h

; Same equ 1h

LSB equ 0h
MSB equ 7h

TRUE equ 1h
YES equ 1h
FALSE equ 0h
NO equ 0h

;PAGE 1

; RS-232 Communication With PIC16C54
; Half Duplex Asynchronous Communication
; This program has been tested at Bauds from 1200 to 19200 Baud
; (@ 8,16,20 MHz CLKIN)
; As a test, this program will echo back the data that has been received.

© 1993 Microchip Technology Inc.
Asynchronous Serial I/O

;************************ Communication Parameters ***************

; 0001 X_MODE equ 1 ; If ( X_MODE=1) Then transmit LSB
;           if ( X_MODE=0) Then transmit MSB
0001 R_MODE equ 1 ; If ( R_MODE=1) Then receive LSB
;           if ( R_MODE=0) Then receive MSB
0001 X_Nbit equ 1 ; if (X_Nbit=1) # of data bits ( T
0001 R_Nbit equ 1 ; if (R_Nbit=1) # of data bits ( R
;
0000 Sbit2 equ 0 ; if Sbit2 = 0 then 1 Stop Bit else
;
;**********************************************************

0005 X_flag equ PA0 ; Bit 5 of F3 ( PA0 )
0006 R_flag equ PA1 ; Bit 6 of F3 ( PA1 )
;
0000 DX equ 0 ; Transmit Pin ( Bit 0 of Port A )
0001 DR equ 1 ; Recive Pin ( Bit 1 of Port A )
;
0044 BAUD_1 equ .68 ; 3+3X = CLKOUT/Baud
0043 BAUD_2 equ .67 ; 6+3X = CLKOUT/Baud
0022 BAUD_3 equ .34 ; 3+3X = 0.5*CLKOUT/Baud
0056 BAUD_4 equ .86 ; 3+3X = 1.25*CLKOUT/Baud
0042 BAUD_X equ .66 ; 11+3X = CLKOUT/Baud
0042 BAUD_Y equ .66 ; 9 +3X = CLKOUT/Baud
;
;************************ Data RAM Assignments ***************

; 0008 0001 RcvReg RES 1 ; Data received
0009 0001 XmtReg RES 1 ; Data to be transmitted
000A 0001 Count RES 1 ; Counter for #of Bits Transmitted
000B 0001 DlyCnt RES 1

;**********************************************************

ORG 08H ; Dummy Origin

; 0000 0068 Talk clrf RcvReg ; Clear all bits of RcvReg
0001 0625 btfs Port_A,DR ; check for a Start Bit
0002 0A30 goto User ; delay for 104/2 uS
0003 0923 call Delay4 ; delay for 104+104/4

; Receiver
;
0004 0C08 IF R_Nbit
0005 0C08 movlw 8 ; 8 Data bits
0006 0C08 ELSE
0007 0C08 movlw 7 ; 7 data bits
0008 0C08 ENDIF
;
0005 02A movwf Count
0006 0403 R_next bcf STATUS,CARRY
Asynchronous Serial I/O

```
0007 0328 IF R_MODE rrf RcvReg,Same ; to set if MSB first or LS
ELSE rlf RcvReg,Same ENDIF
0008 0625 ;
    IF R_MODE
    IF R_Nbit
    bsf RcvReg,MSB ; Conditional Assembly
    ELSE
    bsf RcvReg,MSB-1
    ENDIF
    ELSE
    bsf RcvReg,LSB
    ENDIF
0009 05E8 ;
    call DelayY
    decfsz Count,Same
goto R_next
000A 091F ;******************************************************************************
000B 02EA R_over movf RcvReg,0 ; Send back What is Just Received
000C OA06 ;******************************************************************************
000D 0208 movwf XmtReg
; Transmitter
; Xmtr
000E 0229 IF X_Nbit
000F 0C08 movlw 8
ELSE
movlw 7
ENDIF
0010 002A movwf Count
;
    IF X_MODE
ELSE
    IF X_Nbit
ELSE
rlf XmtReg,Same
ENDIF
ENDIF
0011 0405 bcf Port_A,DX ; Send Start Bit
0012 0925 call Delay1
0013 0403 X_next bcf STATUS,CARRY
;
0014 0329 IF X_MODE
0015 0603 rrf XmtReg,Same ; Conditional Assembly
ELSE
rlf XmtReg,Same ; to set if MSB first or LS
ENDIF
0016 0505 btfsc STATUS,CARRY
0017 0703 bsf Port_A,DX
0018 0405 btfss STATUS,CARRY
0019 0921 bcf Port_A,DX
001A 02EA call DelayX
001B 0A13 decfsz Count,Same
goto X_next
001C 0505 bsf Port_A,DX ; Send Stop Bit
001D 0925 call Delay1
;
    IF Sbit2
001E 032A bsf Port_A,DX
call Delay1
ENDIF
```
Asynchronous Serial I/O

; goto Talk ; Back To Reception & Trans

; End of Transmission

001F DC42 DelayY movlw BAUD_Y
go to save
0021 0C42 DelayX movlw BAUD_X
go to save
0022 0A28 Delay4 movlw BAUD_4
go to save
0024 0A28 Delay1 movlw BAUD_1 ; 104 uS for 9600 baud
go to save
0026 0A28 Delay2 movlw BAUD_2
0028 002B save movwf DlyCnt
0029 02EB redo_1 decfsz DlyCnt,Same
go to redo_1
002A 0A29 retlw 0
002B 0800

; main movlw OEH ; Bit 0 of Port A is Output
002C OC0E tris Port_A ; Set Port_A.0 as output
002D 0055 bsf Port_A,DR
002E 0525 ; goto Talk
002F 0A00

; User movlw BAUD_3
0030 0C22 movwf DlyCnt
0031 002B redo_2 decfsz DlyCnt,Same
go to redo_2
go to Talk ; Loop Until Start Bit Found

; ORG PIC54
01FF 0A2C goto main

END

Errors : 0
Warnings : 0
APPENDIX B: ASSEMBLY LANGUAGE LISTING FOR FULL DUPLEX

MPASM B0.54 PAGE 1
RS232 Communication Using PIC16C54

;*************************************************************************
; TITLE "RS232 Communication Using PIC16C54"
; Comments:
; (1) Full Duplex
; (2) Tested from 1200 to 9600 Baud (@ 8 Mhz )
; (3) Tested from 1200 to 19200 Baud (@ 16 & 20 Mhz)

; The User gets a total time as specified by the User Cycles
; in the table ( or from equations ). The user routine has to
; exactly use up this amount of time. After this time the User
; routine has to give up the control to the Operating System.
; If less than 52 us is used, then the user should wait in a
; delay loop, until exactly 52 us.

; Transmission:
; Transmit Data is output on DX pin (Bit DX of Port_A).
; In the user routine, the user should load the
; data to be transmitted in the XmtReg and Set the
; X_flag ( bsf FlagRX, X_flag ). This flag gets cleared
; after the transmission.

; Reception:
; Data is received on pin DR (bit DR of Port_A).
; The User should constantly check the "R_done" flag
; to see if reception is over. If the reception is
; in progress, R_flag is set to 1.
; If the reception is over, "R_done" flag is set to 1.
; The "R_done" flag gets reset to zero when a next start
; bit is detected. So, the user should constantly check
; the R_done flag, and if SET, then the received word
; is in Register "RcvReg". This register gets cleared
; when a new start bit is detected.

; Program Memory:
; Total Program Memory Locations Used (except initialization
; in "main" & User routine ) = 132 locations.

; Data Memory:
; Total Data memory locations (file registers used) = 6
; 2 File registers to hold Xmt Data & Rcv Data
; 1 File registers for Xmt/Rcv flag test bits
; 3 File registers for delay count & scratch pad

; Stack:
; Only one level of stack is used in the Operating System/RS232
; routine. But this is freed as soon as the program returns to the
; user routine.

; RTCC : Not Used
; WDT : Not Used

; LIST P=16C54
INCLUDE "MPREG.H"
PIC16C5X Header

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
Asynchronous Serial I/O

0004 FSR equ 4h
0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

;*************************************************************************

0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h

;*************************************************************************

;*************************************************************************

INCLUDE "RS232.H"

;*************************************************************************

RS232 Communication Parameters

;*************************************************************************

0001 X_MODE equ 1 ; If ( X_MODE==1) Then transmit LSB first
0001 R_MODE equ 0 ; If ( R_MODE==0) Then receive LSB first
0001 X_Nbit equ 1 ; If ( X_Nbit==1) # of data bits ( Transmission ) is
0001 R_Nbit equ 1 ; # of data bits ( Reception ) is 8

;*************************************************************************

0000 SB2 equ 0 ; if SB2 = 0 then 1 Stop Bit

;*************************************************************************

Transmit & Receive Test Bit Assignments

;*************************************************************************

0000 X_flag equ 0 ; Bit 0 of FlagRX
0002 R_flag equ 2 ; Bit 1 of FlagRX
0003 S_flag equ 3 ; Bit 2 of FlagRX
0004 BitXsb equ 4 ; Bit 3 of FlagRX
0005 A_flag equ 5
0006 S_bit equ 6 ; Xmt Stop Bit Flag( for 2/1 Stop bits )

;*************************************************************************

0001 R_done equ 1 ; When Reception complete, this bit is SET
0000 X_done equ X_flag ; When Xmission complete, this bit is Cleared

;*************************************************************************

0000 DX equ 0 ; Transmit Pin ( Bit 0 of Port A )
0001 DR equ 1 ; Recive Pin ( Bit 1 of Port A )
Asynchronous Serial I/O

;****************************************************************************** Data RAM Assignments ******************************************************************************
; ORG 08H ; Dummy Origin
;
0000 0001
0009 0001
000A 0001
000B 0001
000C 0001
000D 0001

RcvReg RES 1 ; Data received
XmtReg RES 1 ; Data to be transmitted
Xcount RES 1 ; Counter for # of Bits Transmitted
Rcount RES 1 ; Counter for # of Bits to be Received
DlyCnt RES 1 ; Counter for Delay constant
FlagRX RES 1 ; Transmit & Receive test flag hold register

; Constants 19200 9600 4800 2400 1200
; ( @ 20 Mhz )
;
; K0  0 13 57 143 317*
; K1  49 98 184 358* 705*
; K2  34 60 103 191 364*
; K3  27 53 96 184 357*
; K4  29 55 98 186 359*
; K5  30 56 99 187 360*
; K6  0  0  0  0  0
; K7  56 104 190 365* 712*
;
User_Cycles 118 260 521 1042 2083

;******************************************************************************
;
; Constants 19200 9600 4800 2400 1200
; ( @ 8 Mhz )
;
; K0  0  5 39 109
; K1  39 80 150 288*
; K2  27 51 86 155
; K3  21 44 80 148
; K4  23 46 82 150
; K5  24 47 83 151
; K6  0  0  0  0
; K7  45 86 156 295*
;
User_Cycles 86 208 416 832

;******************************************************************************
;
; Constants 19200 9600 4800 2400 1200
; ( @ 4 Mhz )
;
; K0  --  0  5  39
; K1  --  39 80 150
; K2  --  27 51 86
; K3  --  21 44 80
; K4  --  23 46 82
; K5  --  24 47 83
; K6  --  0  0  0
; K7  --  45 86 156
;
User_Cycles 86 208 416

;******************************************************************************

; The constants marked " * " are >255. To implement these constants
; in delay loops, the delay loop should be broken into 2 or more loops.
; For example, 357 - 255+102. So 2 delay loops, one with 255 and
; the other with 102 may be used.
Asynchronous Serial I/O

;*************************************************************************
; Set Delay Constants for 9600 Baud @ CLKIN = 8 Mhz
;
; 0000  K0 EQU .0
0027  K1 EQU .39
001B  K2 EQU .27
0015  K3 EQU .21
0017  K4 EQU .23
0018  K5 EQU .24
0000  K6 EQU .0
002D  K7 EQU .45
;
;*************************************************************************

ORG 0

;*************************************************************************

0000 0C01  Delay    movlw K0+1
0001 002C  movwf DlyCnst ; Total Delay = 3K+6
0002 02EC  redo    decfsz DlyCnst,Same
0003 0A02  goto    redo
0004 0800  retlw    0
;
0005 002C  Delay1   movwf DlyCnst
0006 02EC  redo_1   decfsz DlyCnst,Same
0007 0A06  goto    redo_1
0008 0A8D  goto    User
;
0009 002C  Delay2   movwf DlyCnst
000A 02EC  redo_2   decfsz DlyCnst,Same ; Delay = 260 Cycles
000B 0A0A  goto    redo_2
000C 0A67  goto    User_1
;
000D 0625  R_strt   btfsr Port_A,DR ; check for a Start Bit
000E 0A17  goto    ShellY ; delay for 104/2 us
000F 042D  bcf     FlagRX,R_done ; Reset Receive done flag
0010 054D  bsf     FlagRX,R_flag ; Set flag for Reception in Progress
0011 078D  btfsr   FlagRX,BitXsb
0012 05AD  bsf     FlagRX,A_flag ; A_flag is for start bit detected in R_strt
0013 0068  clrf    RcvReg ; Clear all bits of RcvReg
    IF     R_Nbit
0014 0C08  movlw    8 ; 8 Data bits
    ELSE
0015 0C08  movlw    7 ; 7 data bits
    ENDF
0016 0A7B  movwf    Rcount
0017 0A78  goto    Shell ; delay for 104+104/4
;
0018 078D  ShellY  btfsr   FlagRX,BitXsb
0019 054D  bsf     FlagRX,R_flag
001A 0A78  goto    Shell
;
001B 0403  R_next  bcf     STATUS,CARRY
            IF     R_MODE
001C 0328  rrf     RcvReg,Same ; to set if MSB first or LSB first
            ELSE
001D 0625  btfsr   Port_A,DR
            IF     R_MODE
            IF     R_Nbit

© 1993 Microchip Technology Inc. DS00510B-page 13
Asynchronous Serial I/O

001E 05E8  bsf  RcvReg,MSB  ; Conditional Assembly
            ELSE
            bsf  RcvReg,MSB-1
            ENDIF
            ELSE
            bsf  RcvReg,LSB
            ENDIF

001F 02EB  decfsz Rcount,Same
0020 0A78  goto Shell
0021 044D  bcf  FlagRX, R_flag
0022 056D  bcf  FlagRX, S_flag
0023 052D  bcf  FlagRX, R_done
0024 0A78  goto Shell
            ; Reception Done
            ;
0025 0405  bcf  Port_A,DX  ; Send Start Bit
            IF  X_Nbit
            movlw  8
            ELSE
            movlw  7
            ENDIF

0026 0C08  movwf Xcount
            IF  X_MODE
            ELSE
            IF  X_Nbit
            ELSE
            rlf  XmtReg, Same
            ENDIF
            ENDIF

0027 002A  goto X_SB
            ;
0028 0A50  goto X_SB
            ;
0029 0403  bcf  STATUS, CARRY
            IF  X_MODE
            rrf  XmtReg, Same  ; Conditional Assembly
            ELSE
            rlf  XmtReg, Same
            ENDIF
            ENDIF

002A 0329  btfsc STATUS, CARRY
002B 0603  btfsc STATUS, CARRY
002C 0505  bsf  Port_A, DX
002D 0703  btfss STATUS, CARRY
002E 0405  bcf  Port_A, DX
002F 00EA  decf  Xcount, Same
0030 0A52  goto X_Data
            ;
0031 040D  bcf  FlagRX, X_flag  ; Xmt flag = 0 - transmission over
0032 0C09  movlw  9
0033 002A  movwf Xcount
0034 0505  bsf  Port_A, DX  ; Send Stop Bit
0035 0A60  goto X_Stop
            ;
0036 0505  bcf  Port_A, DX
0037 04CD  bcf  FlagRX, S_bit
0038 0A60  goto X_Stop
            ; End of Transmission
            ;
0039 076D  btfss FlagRX, S_flag
003A 0A8D  goto User
003B 046D  bcf  FlagRX, S_flag
003C 0900  call Delay
003D 0C2E  movlw K7+1
003E 0A05  goto Delay1
            ; R1_X0
003F 0900  call Delay
0040 0C28  movlw K1+1
0041 002C  movwf DlyCnt
            IF  R_Nbit

© 1993 Microchip Technology Inc.
Asynchronous Serial I/O

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0042 OC08</td>
<td>movlw 0</td>
<td>ELSE ; 8 Data bits</td>
</tr>
<tr>
<td>0043 018B</td>
<td>xorwf Rcount, W</td>
<td>movlw 7</td>
</tr>
<tr>
<td>0044 0643</td>
<td>btfsc STATUS, Z_bit</td>
<td>ENDIF</td>
</tr>
<tr>
<td>0045 A006</td>
<td>goto redo_1</td>
<td></td>
</tr>
<tr>
<td>0046 0C1C</td>
<td>movlw K+1</td>
<td></td>
</tr>
<tr>
<td>0047 A05</td>
<td>goto Delay1</td>
<td>; R1_Xl ; same as R0_Xl</td>
</tr>
<tr>
<td>0048 OC09</td>
<td>R0_Xl movlw 9</td>
<td></td>
</tr>
<tr>
<td>0049 008A</td>
<td>subwf Xcount, W</td>
<td></td>
</tr>
<tr>
<td>004A 0643</td>
<td>btfsc STATUS, Z_bit</td>
<td></td>
</tr>
<tr>
<td>004B A025</td>
<td>goto X strt</td>
<td></td>
</tr>
<tr>
<td>004C 022A</td>
<td>movf Xcount, Same ; to check if All data bits Xmted</td>
<td></td>
</tr>
<tr>
<td>004D 0743</td>
<td>btfss STATUS, Z_bit</td>
<td></td>
</tr>
<tr>
<td>004E A029</td>
<td>goto X_next</td>
<td></td>
</tr>
<tr>
<td>004F A031</td>
<td>goto X_SB_1</td>
<td></td>
</tr>
<tr>
<td>ENDIF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050 A051</td>
<td>X_SB goto cycle4</td>
<td></td>
</tr>
<tr>
<td>0051 A052</td>
<td>cycle4 goto X_Data</td>
<td></td>
</tr>
<tr>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0052 06AD</td>
<td>X_Data btfsc FlagRX, A_flag</td>
<td></td>
</tr>
<tr>
<td>0053 A059</td>
<td>goto SbDly</td>
<td></td>
</tr>
<tr>
<td>0054 068D</td>
<td>btfsc FlagRX, BitXsb</td>
<td></td>
</tr>
<tr>
<td>0055 A05D</td>
<td>goto ABC</td>
<td></td>
</tr>
<tr>
<td>0056 0900</td>
<td>call Delay</td>
<td></td>
</tr>
<tr>
<td>0057 0C16</td>
<td>movlw K+1</td>
<td></td>
</tr>
<tr>
<td>0058 A009</td>
<td>goto Delay2</td>
<td>; SbDly bcf FlagRX, A_flag</td>
</tr>
<tr>
<td>0059 04AD</td>
<td>call Delay</td>
<td></td>
</tr>
<tr>
<td>005B 0C18</td>
<td>movlw K+1</td>
<td></td>
</tr>
<tr>
<td>005C A009</td>
<td>goto Delay2</td>
<td></td>
</tr>
<tr>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005D 048D</td>
<td>ABC bcf FlagRX, BitXsb</td>
<td></td>
</tr>
<tr>
<td>005E 0900</td>
<td>call Delay</td>
<td></td>
</tr>
<tr>
<td>005F A067</td>
<td>goto User_1</td>
<td></td>
</tr>
<tr>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0060 06AD</td>
<td>btfsc FlagRX, A_flag</td>
<td></td>
</tr>
<tr>
<td>0061 A059</td>
<td>goto SbDly</td>
<td></td>
</tr>
<tr>
<td>0062 068D</td>
<td>btfsc FlagRX, BitXsb</td>
<td></td>
</tr>
<tr>
<td>0063 A05D</td>
<td>goto ABC</td>
<td></td>
</tr>
<tr>
<td>0064 0900</td>
<td>call Delay</td>
<td></td>
</tr>
<tr>
<td>0065 0C19</td>
<td>movlw K+1</td>
<td></td>
</tr>
<tr>
<td>0066 A009</td>
<td>goto Delay2</td>
<td>; User_1 btfsc FlagRX, R_flag</td>
</tr>
<tr>
<td>0067 064D</td>
<td>goto Sync_1 ; Reception already in progress</td>
<td></td>
</tr>
<tr>
<td>0068 A077</td>
<td>btfsc FlagRX, S_flag</td>
<td></td>
</tr>
<tr>
<td>0069 066D</td>
<td>goto Sync_3</td>
<td></td>
</tr>
<tr>
<td>006A A074</td>
<td>btfsc Port A,DR ; check for a Start Bit</td>
<td></td>
</tr>
<tr>
<td>006B 0625</td>
<td>goto Sync_2 ; No Start Bit - goto User routine</td>
<td></td>
</tr>
<tr>
<td>006C A077</td>
<td>bcf FlagRX, R_done ; Reset Receive done flag</td>
<td></td>
</tr>
<tr>
<td>006D 042D</td>
<td>bcf FlagRX, R_flag</td>
<td></td>
</tr>
<tr>
<td>006E 044D</td>
<td>bcf FlagRX, BitXsb ; Set flag for Reception in Progress</td>
<td></td>
</tr>
<tr>
<td>006F 058D</td>
<td>clr RcvReg ; Clear all bits of RcvReg</td>
<td></td>
</tr>
<tr>
<td>0070 0068</td>
<td>IF R_Nbit</td>
<td></td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
Asynchronous Serial I/O

0071 OC08  movlw 8        ; 8 Data bits
            ELSE
            movlw 7        ; 7 data bits
            ENDIF

0072 002B  movwf  Rcount
0073 0A8D  goto  User
;
0074 046D  Sync_3  bcf  FlagRX,S_flag
0075 0C01  movlw  K6+1
0076 0A05  goto  Delay1
;
0077 0A8D  Sync_2  goto  User
;
*****************************************************************************

Shell  btfs  FlagRX,R_flag
0079 0A7D  goto  Chek_X
007A 060D  btfs  FlagRX,X_flag
007B 0A48  goto  R0_X1
007C 0A39  goto  R0_X0    ; Case for R0_X0
007D 060D  Chek_X  btfs  FlagRX,X_flag
007E 0A48  goto  R1_X1
007F 0A3F  goto  R1_X0
;
*****************************************************************************

Operating System
; The User routine after time = B/2, should branch Here
;
0080 074D  Op_Sys  btfs  FlagRX,R_flag
0081 0A0D  goto  R_strt
0082 0A1B  goto  R_next
;
*****************************************************************************

main  movlw  OEH        ; Bit 0 of Port A is Output
0083 0C0E  tris  Port_A    ; Set Port_A.I as output ( DR )
            ; & Port_A.0 as output ( DX )
0084 0005  movlw  9
0085 0505  bcf  FlagRX,DX
0086 0C09  movlw  Xcount    ; If Xcount == 9, Then send start bit
0087 002A  bcf  FlagRX    ; Clear All flag bits.
            IF SB2
                bcf  FlagRX,S_bit; Set Xmt Stop bit flag(2 Stop Bits)
            ELSE
            ENDIF
0089 04CD  bcf  FlagRX, S_bit    ; Clear Xmt Stop bit flag
008A 0C1F  movlw  IFH        ; Prescaler = 4
008B 0002  OPTION    ; Set RTCC increment on Internal Clock
008C 0A80  goto  Op_Sys
;
*****************************************************************************

User Routine

; The User routine should use up time exactly = User time as given
; in the Constants Table ( or by Equations for constants )
; At 9600, this 86 Clock Cycles. RTCC timer is used here to count
; upto 86 cycles ( From 128-86 To 0 ) by examining Bit 7 of RTCC.
Asynchronous Serial I/O

0030 K_user equ .128+.6-.86

008D 0C30 User movlw K_user
008E 0021 movwf RTCC
008F 062D btfs FlagRX,R_done
0090 0A97 goto ErrChk
0091 060D SetXmt btfs FlagRX,X_flag
0092 0A9C goto Op
0093 0C41 movlw 41H
0094 0029 movwf XmtReg
0095 050D bsf FlagRX,X_flag ; Enable Xmission
0096 0A9C goto Op

; ErrChk
0097 0C5A movlw “Z”
0098 0188 xorwf RcvReg,W
0099 0643 btfs STATUS,Z_bit
009A 0A91 goto SetXmt
009B 0A9B error goto error ; Received word is not “Z”

009C 07E1 Op btfs RTCC,MSB ; Test for RTCC bit 7
009D 0A9C goto Op ; If Set, Then RTCC has incremented
009E 0A80 Oflow goto Op_Sys ; to 128.

; ******************************************************************************
; ORG PIC54
01FF 0A83 goto main

END

Errors : 0
Warnings : 0
INTRODUCTION

The PIC16C5X microcontrollers from Microchip are ideally suited for use as smart peripheral devices under the control of the main processors in systems due to their low cost and high speed. They are capable of performing tasks which would simply overload a conventional microprocessor, or require considerable logic circuitry, at a cost competitive with lower mid-range PLDs. To minimize the engineering overhead of adding multiple controllers to a product, it is convenient for the auxiliary controllers to emulate standard I/O peripherals.

A common interface found in existing products is the I^2C bus. This efficient, 2-wire bi-directional interface allows the designer to connect multiple devices together, with the microprocessor able to send data to and receive data from any device on the bus. This interface is found on a variety of components, such as PLLs, DACs, video controllers, and EEPROMs. If a product already contains one or more I^2C devices, it is simple to add a PIC16C5X emulating a compatible component.

This application note describes the implementation of a standard slave device with multiple, bi-directional registers. A subset of the full I^2C specification is supported, which can be controlled by the same software which would talk to a Microchip 24LCXX series EEPROM.

THE I^2C BUS

The I^2C bus is a master-slave 2-wire interface, consisting of a clock line (SCL) and a data line (SDA). Bidirectional communication (and in a full, multi-master system, collision detection and clock synchronization) is facilitated through the use of a wire-and (ie. active-low, passive high) connection.

The standard-mode I^2C bus supports SCL clock frequency up to 100 KHz. The newly released fast-mode I^2C bus supports clock rate up to 400 KHz. This application note will support 100 KHz (standard-mode) clock rate.

Each device has a unique seven bit address, which the master uses to access each individual slave device.

During normal communication, SDA is only permitted to change while SCL is low, thus providing two violation conditions (see Figure 1) which are used to signal a start condition (SDA drops while SCL is high) and a stop condition (SDA rises while SCL is high), which frame a message.

MESSAGE FORMAT

A message is always initiated by the master, and begins with a start condition, followed by a slave address (7 MSBs) and direction bit (LSB = 1 for READ, 0 for WRITE). The addressed slave must acknowledge this byte if it is ready to communicate any data. If the slave fails to respond, the master should send a stop and retry.

If the direction bit is "0" the next byte is considered the sub-address (this is an extension to I^2C used by most multi-register devices). The sub-address selects which "register" or function subsequent read or write operations will affect. Any additional bytes will be received and stored in consecutive locations until a stop is sent. If the slave is unable to process more data, it could terminate transfer by not acknowledging the last byte.
A Smart I²C Peripheral

If the direction bit is "1", the slave will transfer successive bytes to the master (while the master holds the line at "1"), while the master acknowledges each byte with a "0" in the ninth bit. The master can terminate the transfer by not acknowledging the last byte, while the slave can stop the transfer by generating a stop condition.

The start address of a read operation is set by sending a write request with a sub-address only (no data bytes). For a detailed set of timing diagrams and different communication modes, consult any of the Microchip 24LCXX EEPROM specifications. This program communicates using the same formats.

IMPLEMENTATION

The chip will respond to slave address "DEVICE_ADDRESS", which by default is D6 (D7 for read). This address was chosen because it is the fourth optional address of a Philips PCF8573 clock/calender or a TDA8443 tipple video switch (unlikely that a product would contain four of those).

FIGURE 2 - SCHEMATIC OF I²C CONNECTIONS

The connections to the device are shown in Figure 2. The use of RA0 for data in is required. Data is shifted directly out of the port. The code could be modified to make it port independent, but the loss of efficiency may hinder some real-time applications.

This application emulates an I²C device with 8 registers, accessed as sub-addresses 1 through 8 (modulo 7), plus a data channel (0). The example code returns an ID string when the data channel is accessed. When bytes are written to sub-addresses other than 0, they are stored in I2CR0-I2CR7 (I2CR0 gets data written to sub-address 8).
When the initial sub-address is 0, the flag B:ID is set. This is used to indicate access to a special channel. In this case, the data channel is used to return an ID message, or output data to port B, however the natural extension would be to use this as a data I/O channel.

To make the basic device routines easily adaptable to a variety of uses, macros are used to implement the application specific code. This allows the developer the option of using subroutine calls, or in-line code to avoid the 4 clock cycle overhead and use of the precious stack.

<table>
<thead>
<tr>
<th>Macro</th>
<th>User code function</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_MAIN</td>
<td>Code to execute in the main loop while not in a message. If this code takes too</td>
</tr>
<tr>
<td></td>
<td>long, tSH of 4µs will be violated (see Fig. 1). The slave will simply miss the</td>
</tr>
<tr>
<td></td>
<td>address, not acknowledge, and the master will retry.</td>
</tr>
<tr>
<td>USER_Q</td>
<td>This would be quick user code to implement real-time processes. In most applications,</td>
</tr>
<tr>
<td></td>
<td>this macro would be empty. If used, this routine should be kept under 4µs if possible.</td>
</tr>
<tr>
<td>USER_MSG</td>
<td>This would be user code to process a message. It is inserted after a message is</td>
</tr>
<tr>
<td></td>
<td>successfully received.</td>
</tr>
<tr>
<td>USER_RECV</td>
<td>This would be user code to process a received byte. It allows the user to add extra</td>
</tr>
<tr>
<td></td>
<td>code to implement special purpose sub-addresses such as FIFOs.</td>
</tr>
<tr>
<td>USER_XMIT</td>
<td>This would be user code to prepare an output byte. In the default routine, it traps</td>
</tr>
<tr>
<td></td>
<td>sub-address 0 and calls the ID string function.</td>
</tr>
</tbody>
</table>

References:


The *I²C bus and how to use it* (including specification), Signetics/Phillips Semiconductors, January 1992.


About the Author:

Don Lekei has been designing microprocessor based products over 14 years. He has developed many software and hardware products for a wide variety of applications. Mr. Lekei is Manager of Advanced Technologies at NII Norsat International Inc. at their Canadian headquarters in Surrey, British Columbia. Norsat designs and manufactures products to receive broadcast communications from satellites, terrestrial broadcasting systems and optical fibre. Norsat develops technologies and products for satellite entertainment television, broadcast music and data networks.
LIST P=16C54, C=80, N=0, R=DEC

0676 CPU EQU 1654
0000 SIM EQU 0 ;Change timing constants for

01FF _RESVEC EQU 01FFH ;16c54 start address

IF (CPU==1654) || (CPU==1655)
ENDIF

IF CPU==1656
_ENDIF

IF CPU==1657
_ENDIF

_GOTO INIT

;*** Reset Vector ****************************

ORG _RESVEC
RESVEC

GOTO INIT

;*******************************************************************************

;* Macros to set/clear/branch/skip on bits
;* These macros define and use synthetic “bit labels”
;* Bit labels contain the address and bit of a location
;*
;*******************************************************************************

;/* Usage Description*/

;* BIT label,bit,file ;Define a bit label
;* SEB label ;set bit using bit label
;* CLB label ;clear bit using bit label
;* SKBS label ;SKIP on bit set
;* SKBC label ;SKIP on bit clear
;* BBS label,address ;BRANCH on bit set
;* BBC label,address ;BRANCH on bit clear
;* CBS label,address ;CALL on bit set
;* CBC label,address ;CALL on bit clear

;*******************************************************************************

BIT MACRO label,bit,file ;Define a bit label
label EQU file<<8|bit

ENDM

SEB MACRO label ;Set bit
BSF label>>8,label&7 ; (macro)
ENDM
A Smart I²C Peripheral

CLB MACRO label
    BCF label>>8,label&7 ;Clear bit
    ENDM

SKBS MACRO label
    BTFSS label>>8,label&7 ;Skip on bit set
    ENDM

SKBC MACRO label
    BTFSC label>>8,label&7 ;Skip on bit clear
    ENDM

BBS MACRO label,address
    BTFSC label>>8,label&7 ;Branch on bit set
    GOTO address ;(macro)
    ENDM

BBC MACRO label,address
    BTFSS label>>8,label&7 ;Branch on bit clear
    GOTO address ;(macro)
    ENDM

CBS MACRO label,address
    CALL label>>8,label&7 ;Call on bit set
    ENDM

CBC MACRO label,address
    CALL label>>8,label&7 ;Call on bit clear
    ENDM

;For Assembler portability

0000 W EQU 0 ;For file,W
0000 w EQU 0 ;For file,W
0001 F EQU 1 ;For file,F
0001 f EQU 1 ;For file,F

;***********************************************************************

1* REGISTER DECLARATIONS
;***********************************************************************

ORG 0 ;ORG for register declarations

0000 0001 ind RES 1 ;0=pseudo-reg 0 for indirect

0001 0001 RTCC RES 1 ;1=real time counter

0002 0001 PC RES 1 ;2=PC

0003 0001 STATUS RES 1 ;3=status reg

;* Status reg bits

BIT B_C,0, STATUS ;Carry
    B_C EQU STATUS<<8|0

BIT B_DC,1, STATUS ;Half carry
    B_DC EQU STATUS<<8|1

BIT B_Z,2, STATUS ;Zero
    B_Z EQU STATUS<<8|2

© 1993 Microchip Technology Inc.
A Smart I²C Peripheral

BIT   B_PD,3,STATUS ;Power down
0303

BIT   B_TO,4,STATUS ;Timeout
0304

BIT   B_PA0,5,STATUS ;Page select (56/57 only)
0305

BIT   B_PA1,6,STATUS ;Page select (56/57 only)
0306

BIT   B_PA2,7,STATUS ;GP flag
0307

0004 0001
FSR    RES 1 ;4=file select reg 0-4=indi
0005 0001
PORTA RES 1 ;5=port A I/O register (4 b
0006 0001
PORTB RES 1 ;6=port B I/O register

IF (CPU==1655) 11 (CPU==1657)
PORTC RES 1 ;7=I/O port C on 16C54/56 only
ENDIF

0007 0001
I2CFLG RES 1 ;I2C flag reg
0700

BIT   B_RD,0,I2CFLG ;Flag: l=read
0701

BIT   B_UA,1,I2CFLG ;Flag: 0=reading unit address
0702

BIT   B_SA,2,I2CFLG ;Flag: 1=reading subaddress
0703

BIT   B_ID,3,I2CFLG ;Flag: 1=reading id

I2CFLG EQU $ ;Sub-address 8
0008 0001
I2CREG RES 1 ;I2C I/O register
0009 0001
I2CSUBA RES 1 ;Subaddress
000A 0001
I2CBITS RES 1 ;I2C xmit bit counter

******************************************************************************

; 8 Pseudo registers accessed by sub-addresses 1-8
;*(address 0 accesses the ID string)
;* these are read-write registers
******************************************************************************

000B  I2CRO EQU $ ;Sub-address 8
A Smart I²C Peripheral

000B 0001 RES 1 ; 8 pseudo registers
000C I2CR1 EQU $ ; Sub-address 1
000D I2CR2 EQU $ ; Sub-address 2
000E I2CR3 EQU $ ; Sub-address 3
000F I2CR4 EQU $ ; Sub-address 4
0010 I2CR5 EQU $ ; Sub-address 5
0011 I2CR6 EQU $ ; Sub-address 6
0012 I2CR7 EQU $ ; Sub-address 7

; Constants used by program

00D6 DEVICE_ADDRESS EQU 0D6H ; I²C device address

;**********************************************************************************************

; ** PORTA DEFINITIONS
; ** I²C interface uses PORTA
; ** note SDA goes to A0 for code efficiency
; **
; ;**********************************************************************************************

00F7 TAREAD EQU B'11110111' ; TRISA register for SDA rea
00F6 TAWRITE EQU B'11110110' ; TRISA register for SDA writ
00F7 TAINIT EQU TAREAD ; Initial TRISA value

BIT B_SDA,0,PORTA ; I²C SDA (data) This must be bit 0!

0500 B_SDA EQU PORTA<<8|0

0501 B_SCL EQU PORTA<<8|1

; spare B_???,2,PORTA ; not used
; spare B_???,3,PORTA ; not used

;**********************************************************************************************

; **
; ** Port B definition (Parallel out)
; **
; ;**********************************************************************************************

0000 TBINIT EQU B'00000000' ; Port B tris (all output)
00FF PBINIT EQU B'11111111' ; Port B init

;**********************************************************************************************

; * Macros to contain user POLL loop code.
A Smart I²C Peripheral

These are implemented as macros to allow ease of modification especially in real-time applications. The functions could be in-line code or as subroutines depending on ROM/time trade-offs.

USER_MAIN: Decision or code to perform at idle time

USER_Q: 'Quick' code for use during transfer - max I²C Spec. More than 4 És may result in I²C full spec speed.

USER_MSG: Code to execute at receipt of I²C command.

**********************************************************
USER_MAIN MACRO
;*** This would be user code for idle loop
ENDM

USER_Q MACRO
;*** This would be quick user code
ENDM

USER_MSG MACRO
;*** This would be user code to process a message
ENDM

USER_RECV MACRO
;*** This would be user code to process a received byte
;*** example code sends sub-address 0 to port b
BBC B_Id, _NXI_notid ;Channel 0! Bit set if
MOVFW I2CREG
MOVWF PORTB ;get received byte
GOTO IN_CONT

_NXI_notid

ENDM

USER_XMIT MACRO
;*** This would be user code to prepare an output byte
;*** example code sends id string to output
BBC B_Id, _NXO_notid ;Channel 0! Bit set if
CALL GETID
GOTO OUT_CONT ;get next byte from ID

_NXO_notid

ENDM

**********************************************************
; START OF CODE
**********************************************************

ORG 0

**********************************************************
;* Device ID Table (must be at start)
;* TABLE FOR UNIT ID returns next char in W
**********************************************************

GETID

0000 0209
0001 0E07

MOVFW I2CSUBA ;W=I2CSUBA
ANDLW 07H ;Limit to 8 locations

© 1993 Microchip Technology Inc.
ADDWF PC,F

;**********************************************************************
;* Device ID text: read starting at sub-address 0
;**********************************************************************
RETLW 'P'
RETLW 'I'
RETLW 'C'
RETLW '2'
RETLW 0

;**********************************************************************
;* I2C Device routines
;*
;* Enable must be HIGH, else state goes to 0
;* write is to me, read is from me.
;*<========== first byte / subsequent write
;*
;* SDA    X--X--X--X--X--X--X--X
;* SCL    I--I--I--I--I--I--I--I
;* (bit)  s 7 6 5 4 3 2 1 0
;* STATE: 0 1 2 3 2 3 2 3 2 3 2 3 2 3 2
;*<========== subsequent reads ========
;*
;* SDA    X--X--X--X--X--X--X--X--X
;* (bit)ack 7 6 5 4 3 2 1 0
;* SCL    I--I--I--I--I--I--I--I
;* (bit)ack 7 6 5 4 3 2 1 0
;* STATE: 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8
;*<========== Final READ ===============
;*
;* SDA    X--X--X--X--X--X--X--X--X--X--X
;* (bit)ack 7 6 5 4 3 2 1 0
;* SCL    I--I--I--I--I--I--I--I--I
;*
A Smart I²C Peripheral

;** STATE: 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8
;** STATE B is an ignore bit state for non-addressed bits
;** STATE C indicates last sample had ENA low
;** on rising edge of ENA, DATA LOW = low voltage, DATA&CLOC

;I²C interface uses PORTA
;note SDA must be on PORTA,0 for code efficiency

;******************************************************************************

;** INIT
;** Hardware reset entry point
;**
;******************************************************************************

INIT ;Power-on entry

;******************************************************************************

;** RESET
;** software reset entry point
;**
;******************************************************************************

RESET ;Soft reset

000B 0CF7 MOVLW TAINIT ;Init ports
000C 0005 TRIS PORTA
000D 0000 MOVLW TB INIT
000E 0006 TRIS PORTB
000F 0CF7 MOVLW PB INIT
0010 0026 MOVWF PORTB

;******************************************************************************

Main wait loop while idle. POLL loop should be called here

;******************************************************************************

I2CWAIT

0011 0004 CLRWDT ;Clear watchdog timer
CLB B-UA ;Init state flags
0012 0427 BCF B-UA>>8,B-UA&7

CLB B-SA ;Init state flags
0013 0447 BCF B-SA>>8,B-SA&7

CLB B-RD ;Init state flags
0014 0407 BCF B-RD>>8,B-RD&7

loop1 CLRWDT ;Clear watchdog timer

USER_MAIN ;Call user code while in idle state
;** This would be user code for idle loop

© 1993 Microchip Technology Inc.
A Smart I²C Peripheral

SKBC B_SDA ;Wait for SDA&SCL=H
BTFSC B_SDA>>8,B_SDA&7

loop2

SKBS B_SCL ;
BTFSS B_SCL>>8,B_SCL&7

GOTO loop1 ; No longer valid to wait f

CLRWDT ;Clear watchdog timer

USER_MAIN ;Call user code while in idle state

;** This would be user code for idle loop

;** wait for start **
SKBC B_SCL ;Clock has dropped
BTFSC B_SCL>>8,B_SCL&7

SKBC B_SDA ;Data dropped... Start!
BTFSC B_SDA>>8,B_SDA&7

GOTO loop2

;** START RECEIVED! - wait for first bit!
loop3

BBS B_SDA,I2CWAIT ;Data raised before clock dropped -

BTFSC B_SDA>>8,B_SDA&7

GOTO I2CWAIT

BBS B_SCL,loop3 ;Wait for clock low

BTFSC B_SCL>>8,B_SCL&7

GOTO loop3

NEXTBYTE

CLRWDT ;Clear watchdog timer

MOVlw 1 ;Init receive byte so bit f

MOVWF I2CREG

;** Shift bits! - external poll may be executed during low

;* ENABLE line is checked for loss of enable ONLY during HI

;*** CLOCK IS LOW - DATA MAY CHANGE HERE
;*** We have at least 4Es before any change can occur

loop4

USER_Q

;*** This would be quick user code

loop4A

BBS B_SCL,loop4A ;Wait for clock high

BTFSS B_SCL>>8,B_SCL&7
A Smart I²C Peripheral

0025 0A24
GOTO loop4A

0026 0305
;*** CLOCK IS HIGH - SHIFT BIT - then watch for change
0027 0368
RRF PORTA,W
0028 0603
RLF I2CREG,F ;Shift in bit
0029 0A36
SKPN ;Skip if not done
002A 0608
GOTO ACK_I2C ;Acknowledge byte
002B 0A31
BTFSC I2CREG,0 ;Skip if data bit was 0
ii_0
002C 0725
GOTO i_1 ;This bit was set
002D 0A24
BTFSS B_SCL>>8,B_SCL&7 ;Wait for clock low
002E 0705
SKBS B_SDA ;Data low-high == stop
002F 0A2C
BTFSS B_SDA>>8,B_SDA&7
GOTO i_0 ;process completed message!

0030 0A11
i_1
GOTO I2CWAIT ;back to main loop
0031 0705
BBC B_SDA,I2CWAIT ;Data high-low == start
0032 0A11
GOTO I2CWAIT
0033 0725
BTFSS B_SCL>>8,B_SCL&7 ;Wait for clock low
0034 0A24
GOTO loop4

0035 0A31
GOTO i_1

ACK_I2C
0036 0727
BBC B_UA,ACK-UA ;Not addressed - check unit address
0037 0A8B
BTFSS B_UA>>8,B_UA&7
GOTO ACK-UA
0038 0647
BB B_SA,ACK-SA ;Reading secondary address
0039 09A7
BTFSC B_SA>>8,B_SA&7
GOTO ACK-SA

;***
;** Do what must be done with new data bytes here (before A
;** Don't ack if byte can't be processed!
;****

;*** This would be user code to process a received byte
;*** example code sends sub-address 0 to port b

USER_RECV
A Smart I²C Peripheral

003A 0767  BTFSS B_ID>>8, B_ID4?
003B 0A3F  GOTO _NXI_notid
003C 0208  MOVF B I2CREG
003D 0026  MOVF PORTB
003E 0A47  GOTO IN_CONT
   _NXI_notid
003F 0C07  MOVLW 07H
0040 0169  ANDWF I2CSUBA,f
0041 0C5B  MOVFW I2CRC
0042 01C9  ADDWF I2CSUBA,W
0043 02A9  INCF I2CSUBA
0044 0024  MOVWF FSR
0045 0208  MOVFW I2CREG
0046 0020  MOVFW ind
   IN_CONT

ACKloop
   BBS B_SCL, ACKloop
   BTFSS B_SCL>>8, B_SCL4?
   GOTO ACKloop
0047 0625  BTFSC B_SCL>>8, B_SCL4?
0048 0A47  GOTO ACKloop

0049 0405  CLB B_SDA
   BCF B_SDA>>8, B_SDA4?
004A 0CF6  MOVLW TAWRITE
004B 0005  TRIS PORTA
004C 0405  CLB B_SDA
   BCF B_SDA>>8, B_SDA4?

ACKloop2
   USER_Q
   *** This would be quick user code
   BBS B_SCL, ACKloop2
   BTFSS B_SCL>>8, B_SCL4?
   GOTO ACKloop2
004D 0725  BTFSS B_SCL>>8, B_SCL4?
004E 0A4D  GOTO ACKloop2

ACKloop3
   USER_Q
   *** This would be quick user code
   BBS B_SCL, ACKloop3
   BTFSS B_SCL>>8, B_SCL4?
   GOTO ACKloop3
004F 0625  BTFSS B_SCL>>8, B_SCL4?
0050 0A4F  GOTO ACKloop3

0051 0CF7  MOVLW TAREAD
0052 0005  TRIS PORTA
   BBC B_RD, NEXTBYTE

© 1993 Microchip Technology Inc.

DS00541B-page 13
2-53
A Smart I²C Peripheral

0053 0707          BTFSS  B_RD>>8,B_RD&7
0054 0A21          GOTO  NEXTBYTE

;*******************************************************
; I²C Readback (I²C read request)
; Application specific code to get bytes to send may be add
; This routine gets data from location pointed to by I2Csub
; sends it to I²C. Subsequent reads get sequential address
; AND's the register # with 7 to limit to 8 registers (for
; could be modified to do a comparison to an absolute number
;
;*******************************************************

NEXTOUT
;*** << PUT NEXT BYTE INTO I2CREG HERE NOW! >> ***

USER_XMIT
;*** This would be user code to prepare an output byte
;*** example code sends id string to output

0055 0767          BTFSS  B_ID>>8,B_ID&7
0056 0A59          GOTO  _NXO_notid
0057 0900          CALL  GETID ;get next byte from ID chan
0058 0A60          GOTO  OUT_CONT ;and send it

0059 0C07          MOVLW  07H ;Register count
005A 0169          ANDWF  I2CSUBA,f ;Limit register count
005B 0C0B          MOVLW  I2CR0 ;Pseudo-registers
005C 01C9          ADDWF  I2CSUBA,W ;Offset from buffer start
005D 02A9          INCF  I2CSUBA ;Next sub-address
005E 0024          MOVWF  FSR ;Indirect address
005F 0200          MOVFW  ind ;Get data from register

OUT_CONT
0060 0028          MOVWF  I2CREG ;-- add code here to init I2CREG! when B_ID is clear!
0061 0C08          MOVLW  8 ;Bit counter
0062 002A          MOVWF  I2CBITS ;** OUT bits! -- external poll may be executed during low c
; may also be executed during high cycle if

;* ENABLE line is checked for loss of enable ONLY during HI

;*** CLOCK IS LOW - CHANGE DATA HERE FIRST!

;*** loop l: data was 1
; iIOUT_loop_l

0063 0368          RLF   I2CREG,F ;Shift data out, MSB first
A Smart I²C Peripheral

0064 0603  SKPNC
0065 0A79  GOTO i1OUT_1 ;1->0: change
0066 0405  CLB  B_SDA ;Output another 1!
            BCF  B_SDA>>8,B_SDA&7 ;Output 0

0067 0CF6  MOVLW TAWRITE
0068 0005  TRIS  PORTA
0069 0405  CLB  B_SDA
            BCF  B_SDA>>8,B_SDA&7 ;Set data (just in case docs are

0070 0725  MOVLW TAWRITE
0071 OA6B  GOTO i1OUT_loop_02

006A 0004  i1OUT_0
            CLRWDT ;Clear watchdog timer

USER_Q
;*** This would be quick user code

i1OUT_loop_02
006B 0725  BBC  B_SCL,i1OUT_loop_02 ;Wait for clock high
            BTFSS  B_SCL>>8,B_SCL&7

006C 0A6B  GOTO i1OUT_loop_02

USER_Q
;*** This would be quick user code

i1OUT_loop_03
006D 0625  BBC  B_SCL,i1OUT_loop_03 ;Wait for clock low
            BTFSC  B_SCL>>8,B_SCL&7

006E 0A6D  GOTO i1OUT_loop_03

006F 02EA  DECFSZ I2CBITS ;Count bits
0070 0A74  GOTO i1OUT_loop_0 ;Loop for last bit 0
0071 0CF7  MOVLW TAREAD ;Done with last bit 0... Set
0072 0005  TRIS  PORTA
0073 0A80  GOTO i1OUT_ack ;Get ACK

i1OUT_loop_0
0074 0368  RLF  I2CREG,F ;Shift data out, MSB first
0075 0703  SKPC ;0->1: change
0076 0A6A  GOTO i1OUT_1 ;Output another 0!

0077 0CF7  MOVLW TAREAD ;Set to 1
0078 0005  TRIS  PORTA

i1OUT_1
0079 0004  CLRWDT ;Clear watchdog timer

USER_Q
;*** This would be quick user code

i1OUT_loop_12
007A 0725  BBC  B_SCL,i1OUT_loop_12 ;Wait for clock high
            BTFSS  B_SCL>>8,B_SCL&7
A Smart I2C Peripheral

007B 0A7A
GOTO iiOUT_loop_12

;*** This would be quick user code

iiOUT_loop_13
BBS B_SCL,iiOUT_loop_13 ;Wait for clock low
BTFSC B_SCL>>8,B_SCL&7
GOTO iiOUT_loop_13

007C 0625
007D 0A7C
007E 02EA
007F 0A63
0080 02A9
0081 0725
0082 0A81
0083 0605
0084 0A11
0085 0725
0086 0A55
0087 0605
0088 0A85
0089 0A11
008A 0A11

iiOUT_ack
INCF I2CSUBA ;Get acknowledge
GOTO iiOUT_loop_a2 ;Next sub-address

iiOUT_loop_a2
BBS B_SCL,iiOUT_loop_a2 ;Wait for clock high
BTFSS B_SCL>>8,B_SCL&7
GOTO iiOUT_loop_a2

0081 0725
0082 0A81

--; prepare next character here!

iiOUT_loop_a3
BBS B_SCL,iiOUT_loop_a3 ;Wait for clock low - output next
B_SCL>>B,B_SCL&7
NEXTOUT

0085 0725
0086 0A55
0087 0605
0088 0A85
0089 0A11
008A 0A11

I2CWAIT

GOTO iiOUT_loop_a3 ;Watch out for new start condition

B_SCL>>B,B_SCL&7
GOTO iiOUT_loop_a3

GOTO I2CWAIT ;Stop received!

GOTO I2CWAIT

ACK_UA
A Smart I²C Peripheral

008B 0527   SEB B_UA
            BSF B_UA>>8,B_UA&7
;Flag unit address received

008C 0608   BTFSC I2CREG,0
            SEB B_RD
            BSF B_RD>>8,B_RD&7
;Skip if data coming in
            ;Flag - reading from slave

008D 0507   MOVF I2CREG,W
            ANDLW 0FEH
            XORLW DEVICE_ADDRESS
            BNF I2CREG
            ;Get address
            ;Mask direction flage befor

008E 0208   RNZ I2CREG
008F 0EFE   MOVF I2CREG,W
            BBS B_RD,ACKloop
            ;Device address
            ;Not for me! (skip rest of

0090 0FD6   BTFSC B_RD>>8,B_RD&7
0091 0743 0A11   GO TO ACKloop
            ;Read - no secondary address

0092 0607   SEB B_SA
            BSF B_SA>>8,B_SA&7
            ;Next is secondary address

0093 0A47   GSBO ACKloop
0094 0A47   SEB B_SA
            BSF B_SA>>8,B_SA&7
            ;Yes! ACK address and conti

;*******************************************************
;* Secondary address received - stow it!
;* SA = 0 is converted to 128 to facilitate ID read
;*******************************************************

ACK_SA
0097 0447   CLB B_SA
            BCF B_SA>>8,B_SA&7
;Flag second address received

0098 0467   CLB B_ID
            BCF B_ID>>8,B_ID&7

0099 0208   MOVFW I2CREG
009A 0643   SKPNZ
009B 0567   MOVF I2CREG
            BBS B_ID
            ;Get subaddress
            ;Not 0
            ;Flag - id area selected

009C 0029   SEB B_ID
009D 0A47   BSF B_ID>>8,B_ID&7
            ;Set subaddress

            MOVF I2CSUBA
            ;Set subaddress

            GO TO ACKloop

END

Errors : 0
Warnings : 0
INTRODUCTION

The PIC16CSX microcontrollers are ideal for implementing low cost combinational and sequential logic circuits that traditionally have been implemented using either numerous TTL gates or using programmable logic chips such as PLAs or EPLDs.

PIC16CSX is a family of high-performance 8-bit microcontrollers from Microchip Technology. It employs Harvard architecture, i.e., has a separate data bus (8-bit) and a program bus (12-bit wide). All instructions are single word and execute in one cycle except for program branches. The instruction cycle time is 200 ns at 20 MHz and faster versions with clock frequency of 20 MHz (instruction cycle = 200 ns) are planned. The PIC16CSX microcontrollers are ideal for PLD-type application because:

- Very low cost. Extremely cost effective to replace TTL gates or expensive EPLD’s.
- Fully programmable. PIC16CSX microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
- Available off the shelf from distributors.
- PC board real estate saving can be substantial when replacing multitude of TTL’s or several PLD’s with PIC16CSX microcontrollers which are packaged in 18 and 28 pin packages (DIP, PLCC, SOIC).
- Substantial power savings can be attained by using PIC16CSX’s SLEEP mode. In this mode typical power consumption of PIC16CSX is less than 1uA.
- PIC16CSX’s I/O ports are bidirectional and software configurable as input or output. The user can mix and match number of inputs or outputs as long as the total does not exceed 20 (PIC16CS5/57).
- PIC16CSX’s output pins have large current source/sink capability. They can directly drive LED’s.
- The speed and efficiency of the PIC16CSX allows it to perform other control, timing, and compute functions in addition to implementing a PLA function.

IMPLEMENTING A PLA

To implement a generic combinational logic function, we can simply emulate an AND-OR PLA in software. This will require that the logic outputs be described as sum of products. To describe our algorithm, we will use a simple 8-input, 8-bit output PLA with 24 product terms (Figure 1). We will further use the truth table in Figure 2 as the PLA function being implemented. In this example, only four inputs (A3: A0) are used and the other four inputs (A7: A4) are don’t care. On the output side, seven output pins (Y6: Y0) are used and Y7 is unused. To implement this PLA, the logic inputs A0, A1, ..., A7 can be connected to port RB pins RB0, RB1, ..., RB7 respectively. The logic outputs Y0, Y1, ..., Y7 will appear on port RC pins RC0, RC1, ..., RC7 respectively. Port RB is configured as input and port RC will be configured as output. To evaluate each product term one XOR (exclusive OR) and one AND operation will be required. For example, to determine product term P3 = A3.A2.A1.A0, the expression to evaluate is:

\[ (A<7:0> \text{ XOR } \text{ P3_x}) \text{ AND } 00001111B \]

where P3_x = XXXX0011 B will ensure that if A<3:0> = 0011 B, the least significant 4 bits of the result will be 0000B. The AND constant, referred to here as P3_a (Product term 3, AND constant) basically eliminates the don’t care inputs (here A<7:4>) by masking them. Therefore, if the result of the XOR-AND operation is zero then P3 = 1 else P3 = 0. Once the Product terms are evaluated they are stored in four product registers Preg_0 to Preg_3. To determine an output term:

\[
Y_0 = P_0 + P_2 + P_3 + P_5 + P_6 + P_7 + P_8 + P_9 + P_{10} + P_{12} + P_{13} + P_{14}
\]

we need to evaluate the following expression:

\[
(P_{\text{reg a}} \text{ AND } \text{ OR}_a0) \text{ OR } (P_{\text{reg b}} \text{ AND } \text{ OR}_b0) \text{ OR } (P_{\text{reg c}} \text{ AND } \text{ OR}_c0)
\]

In our case the constant values to implement Y0 are as follows:

<table>
<thead>
<tr>
<th>OR_a0</th>
<th>OR_b0</th>
<th>OR_c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 1 1 0 1</td>
<td>1 1 0 1 1</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

For larger number of inputs, outputs or product terms, the evaluation will be more complex but following the same principle. Appendix A shows the assembly code to implement this 8 input X 8 output X 24 Product PLA. This example optimizes speed as well as program memory requirement. Appendix B shows a slightly different implementation (only EVAL_Y MACRO is different) that optimizes program memory usage over speed. Table 1 shows time and resources required to implement different size PLAs.
FIGURE 1 - A SIMPLE PLA

FIGURE 2 - BINARY TO 7-SEGMENT CONVERSION EXAMPLE

Truth Table:

<table>
<thead>
<tr>
<th>Hex</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>Y0</th>
<th>Y1</th>
<th>Y2</th>
<th>Y3</th>
<th>Y4</th>
<th>Y5</th>
<th>Y6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Product Terms:

- \( P_1 = A_3, A_2, A_1, A_0 \)
- \( P_2 = A_3, A_2, A_1, A_0 \)
- \( P_3 = A_3, A_2, A_1, A_0 \)
- \( P_4 = A_3, A_2, A_1, A_0 \)
- \( P_5 = A_3, A_2, A_1, A_0 \)
- \( P_6 = A_3, A_2, A_1, A_0 \)
- \( P_7 = A_3, A_2, A_1, A_0 \)
- \( P_8 = A_3, A_2, A_1, A_0 \)
- \( P_9 = A_3, A_2, A_1, A_0 \)
- \( P_{10} = A_3, A_2, A_1, A_0 \)
- \( P_{11} = A_3, A_2, A_1, A_0 \)
- \( P_{12} = A_3, A_2, A_1, A_0 \)
- \( P_{13} = A_3, A_2, A_1, A_0 \)
- \( P_{14} = A_3, A_2, A_1, A_0 \)
- \( P_{15} = A_3, A_2, A_1, A_0 \)
SPEED/RESPONSE TIME

The worst case response time of a PLA implemented in this fashion can be calculated as follows. First, we define \( t_d \) = time required to execute the PLA program assuming the worst case program branches are taken. Then the maximum propagation delay time from input change to valid output = 2\( t_d \). This is because if an input changes just after the program reads input port, its effect will not show up until the program completes the current execution cycle, re-reads input and recalculates output. This is shown in Figure 3. There are ways to improve the delay time such as sample inputs several times throughout the PLA program and if input change is sensed, return to the beginning rather than execute the rest of the evaluation code.

A Table Look-Up Method For Small PLA Implementation

If the number of inputs is small (8 or less) then a simple table look-up method can be used to implement the PLA. This will improve execution time to around 5 \( \mu \)s (@ 8 MHz input clock). The following code implements the BCD to 7-segment conversion (Figure 2) using this technique.

```
begin movlw 0fh ;
tris 6 ;Port_b = input
clrw ;
tris 7 ;Port_c = output
pla 88 movf Port_b, w ;Read input
andlw 0fh ;Mask off bits 7:4
call op_tbl ;
movwf Port_c ;Write output
goto pla88 ;

op_tbl addwf pc ;Computed jump for
                  table look-up
retlw b'00111111';
retlw b'00000010';
retlw b'01011101';
retlw b'01001101';
retlw b'01100110';
retlw b'01101101';
retlw b'01111111';
retlw b'01111011';
retlw b'01110111';
retlw b'01111000';
retlw b'00111001';
retlw b'01011110';
retlw b'01110001';
retlw b'01110000';
```

### TABLE 1 - EXECUTION TIME AND RESOURCES NECESSARY FOR DIFFERENT SIZE PLA's

<table>
<thead>
<tr>
<th>Number of Inputs Including fdbk</th>
<th>Number of Outputs Including fdbk and o/e Control</th>
<th>Number of Products</th>
<th>Number of RAM Locations Required NRAM</th>
<th>Number of Program Memory Locations Required NROM</th>
<th>Number of Instruction Cycle To Execute PLA NCYC</th>
<th>Real Time @ 20 MHz to Execute PLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>24</td>
<td>5</td>
<td>228</td>
<td>228</td>
<td>45.6 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>222</td>
<td>352</td>
<td>70.4 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>447</td>
<td>447</td>
<td>89.4 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>384</td>
<td>535</td>
<td>107 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>1042</td>
<td>1042</td>
<td>208.4 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22</td>
<td>843</td>
<td>1250</td>
<td>250 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>1661</td>
<td>1661</td>
<td>372.2 ( \mu )s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>1661</td>
<td>1462</td>
<td>444.2 ( \mu )s</td>
</tr>
</tbody>
</table>

If \( N_i \) = Number of inputs
\( N_{IW} \) = Number of input words, \( N_{IW} = \left\lceil \frac{N_i}{8} \right\rceil \)
\( N_P \) = Number of products
\( N_{PW} \) = Number of product words, i.e. \( N_{PW} = \left\lceil \frac{N_P}{8} \right\rceil \)
\( N_O \) = Number of outputs
\( N_{NOW} \) = Number of output words, i.e. \( N_{NOW} = \left\lceil \frac{N_O}{8} \right\rceil \)

Then, \( N_{RAM} \) @ \( N_{IW} \) + NOW + N PW: Time efficient
\( N_{RAM} \) @ \( N_{IW} + \) NOW + 2 N PW + 2: Code efficient
\( N_{ROM} \) @ \( B + \) N PW + NOW + N P [2 + 3 NIW] + 20 N PW + 3: Code efficient
\( N_{NCYC} \) @ \( 8 + \) N PW + NOW + N P [2 + 3 NIW] + 20 N PW + 4: Time efficient
\( N_{NCYC} \) @ \( 8 + \) N PW + NOW + N P [2 + 3 NIW] + 20 N PW + 5 N [N PW + 1]: Code efficient
FIGURE 3 - PLA PROGRAM FLOW

A very simple PLA evaluation flow.

In this simple implementation, the maximum propagation delay from input change to output change is \( \leq 2td \).

FIGURE 4 - AN ASYNCHRONOUS STATE MACHINE

The concept can be easily extended to implement sequential logic i.e. a state machine. Figure 4 shows a state machine with \( n \) inputs (A0-An), \( m \) outputs (Y0-Ym) and \( p \) states that feedback as inputs to the PLA (F0-Fp). In PIC16C5X the states will be stored as bits in RAM location. Input will now mean input from a port as well as from the feedback registers. Figure 5 shows an example PLA with 8 inputs A0,A1,...,A7 that are connected to port RB pins RB0,RB1,...,RB7. This example PLA has a total of 24 outputs of which 8 are actual outputs, another 8 are output enable control for the outputs and the other 8 are feedbacks (or states). The PLA shown here, therefore, in essence implements an asynchronous state machine (i.e. there is no system clock).
This example shows 16 inputs (including feedback), 64 product terms and 24 outputs including feedback and O/E control. This example demonstrates that O/E control is easily implementable using PIC's bidirectional ports with tristate control.
IMPLEMENTING A SYNCHRONOUS STATE MACHINE

In a synchronous system (see Figure 6) usually all inputs are stable at the falling edge (or rising) edge of the system clock. The state machine samples input on the falling edge, evaluates state and output information. The state outputs are latched by the rising edge of the clock before feeding them back to the input (so that they are stable at the falling edge of the clock). To implement such a state machine, the system clock will have to be polled by an input pin. When a falling edge is detected, the PLA evaluation procedure will be invoked to compute outputs and write them to output pins. The PLA procedure will also determine the new state variables, F0m, F1m,...,Fpm and store them in RAM. The program will then wait until a rising edge on the clock input is detected and copy the "master" state variables (F0m,...,Fpm) to slave state variables (F0s,F1s,...,Fps). This step emulates the feedback flip-flops.

SUMMARY

In conclusion, the PIC16C5X can implement a generic PLA equation and provide quick, low cost solution where system operation speed is not critical.

Author: Sumit Mitra
Logic Products Division
APPENDIX A: PLA IMPLEMENTATION: TIME EFFICIENT APPROACH

MPASM BO.54 PAGE 1

;******************************************************
; plala.asm :
; This procedure implements a simple AND-OR PLA with:
;
; 8 inputs := A7 A6 A5 A4 A3 A2 A1 A0
; 24 product terms := P23 P22 .... P0
; 8 outputs := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
;
; The eight inputs are assumed to be connected to PORT RB such that
; RB0 = A0, RB1 = A1, ..., RB7 = A7.
; The outputs are programmed to appear on port RC such that
; RC0 = Y0, RC1 = Y1, ..., RC7 = Y7.
;
; This implementation optimizes both speed & program memory usage
;
; ******************************************************
;
; define RAM locations used:
;
0000C input equ d'12' ;RAM location 12 holds input
0000D Y_reg equ d'13' ;holds output result
0000E Preg_a equ d'14' ;Product terms P0 to P7, Preg_a<0> = P0
0000F Preg_b equ d'15' ;Product terms P8 to P15, Preg_b<0> = P8
0010 Preg_c equ d'16' ;Product terms P16 to P23, Preg_c<0> = P16

; define some constants and file addresses:
;
0000 bit0 equ 0 
0001 bit1 equ 1 
0002 bit2 equ 2 
0003 bit3 equ 3 
0004 bit4 equ 4 
0005 bit5 equ 5 
0006 bit6 equ 6 
0007 bit7 equ 7 

0003 status equ 3 
0006 port_b equ 6 
0007 port_c equ 7 

; define the AND plane programming variables:
;
0000 P0_x equ b'00000000' ;
000F P0_a equ b'000001111' ;
0010 P1_x equ b'000000001' ;
000F P1_a equ b'000001111' ;
0002 P2_x equ b'000000010' ;
000F P2_a equ b'000001111' ;
0004 P3_x equ b'000000011' ;
000F P3_a equ b'000001111' ;
0004 P4_x equ b'0000000100' ;
000F P4_a equ b'000001111' ;
0005 P5_x equ b'0000000111' ;
000F P5_a equ b'000001111' ;
0006 P6_x equ b'0000000110' ;
000F P6_a equ b'000001111' ;
0007 P7_x equ b'0000000111' ;
000F P7_a equ b'000001111' ;
0008 P8_x equ b'000001000' ;
000F P8_a equ b'000001111' ;
0009 P9_x equ b'000001001' ;
### PLD Replacement

<table>
<thead>
<tr>
<th>Location</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000F P9.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000A P10.x</td>
<td>equ b'00000101' ;</td>
</tr>
<tr>
<td>000F P10.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000B P11_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P11.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000C P12_x</td>
<td>equ b'00000110' ;</td>
</tr>
<tr>
<td>000F P12.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000D P13_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P13.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000E P14_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P14.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P15_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P15.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P16_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P16.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P17_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P17.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000D P18_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000E P18.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P19_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P19.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P20_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P20.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P21_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P21.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P22_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P22.a</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P23_x</td>
<td>equ b'00000111' ;</td>
</tr>
<tr>
<td>000F P23.a</td>
<td>equ b'00000111' ;</td>
</tr>
</tbody>
</table>

; define OR plane programming variables:

<table>
<thead>
<tr>
<th>Location</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000D OR_a0</td>
<td>equ b'11101101' ; for output Y0</td>
</tr>
<tr>
<td>000D OR_b0</td>
<td>equ b'11101111' ;</td>
</tr>
<tr>
<td>0000 OR_c0</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>000F OR_a1</td>
<td>equ b'10011111' ; for output Y1</td>
</tr>
<tr>
<td>0027 OR_b1</td>
<td>equ b'01010111' ;</td>
</tr>
<tr>
<td>0000 OR_c1</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>000B OR_a2</td>
<td>equ b'11111011' ; for output Y2</td>
</tr>
<tr>
<td>002F OR_b2</td>
<td>equ b'00101111' ;</td>
</tr>
<tr>
<td>0000 OR_c2</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>006D OR_a3</td>
<td>equ b'01101101' ; for output Y3</td>
</tr>
<tr>
<td>0079 OR_b3</td>
<td>equ b'01111001' ;</td>
</tr>
<tr>
<td>0000 OR_c3</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>0045 OR_a4</td>
<td>equ b'01000101' ; for output Y4</td>
</tr>
<tr>
<td>00DF OR_b4</td>
<td>equ b'11111111' ;</td>
</tr>
<tr>
<td>0000 OR_c4</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>0071 OR_a5</td>
<td>equ b'01110001' ; for output Y5</td>
</tr>
<tr>
<td>00DF OR_b5</td>
<td>equ b'11011111' ;</td>
</tr>
<tr>
<td>0000 OR_c5</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>007C OR_a6</td>
<td>equ b'01111100' ; for output Y6</td>
</tr>
<tr>
<td>00EF OR_b6</td>
<td>equ b'11101111' ;</td>
</tr>
<tr>
<td>0000 OR_c6</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>0000 OR_a7</td>
<td>equ b'00000000' ; for output Y7</td>
</tr>
<tr>
<td>0000 OR_b7</td>
<td>equ b'00000000' ;</td>
</tr>
<tr>
<td>0000 OR_c7</td>
<td>equ b'00000000' ;</td>
</tr>
</tbody>
</table>

org 01ffh ;

01FF 0A00 begin goto main ;

org 000h ;

; define macro to evaluate 1 product (AND) term:

0000 0902 main call pla88 ;
0001 0A00 goto main ;
PLD Replacement

EVAL_P MACRO Preg_x,bit_n,Pn_x,Pn_a
    movf input,W
    xorlw Pn_x
    andlw Pn_a
    btfsc status,bit2 ; skip if zero bit not set
    bsf Preg_x,bit_n ; product term = 1
ENDM

; define macro to load OR term constants:
EVAL_Y MACRO OR_an,OR_bn,OR_cn,bit_n
    LOCAL SETBIT
    movf Preg_a,W
    andlw OR_an
    btfsc status,bit2
    goto SETBIT
    movf Preg_b,W
    andlw OR_bn
    btfsc status,bit2
    goto SETBIT
    movf Preg_c,W
    andlw OR_cn
    btfsc status,bit2
    SETBIT bsf y_reg,bit_n
ENDM

; now the PLA evaluation procedure:
pla88 movlw 0ffh ;
0003 0006 tris 6 ; port_b = input
0004 0206 movf port_b,W ; read input
0005 002C movwf input ; store input in a register
0006 006E clr Preg_a ; clear Product register a
0007 006F clr Preg_b ; clear Product register b
0008 0070 clr Preg_c ; clear Product register c
0009 006D clr y_reg ; clear output register

EVAL_P Preg_a,bit0,P0_x,P0_a
    movf input,W
    xorlw P0_x
    andlw P0_a
    btfsc status,bit2 ; skip if zero bit not set
    bsf Preg_a,bit0 ; product term = 1

EVAL_P Preg_a,bit1,P1_x,P1_a
    movf input,W
    xorlw P1_x
    andlw P1_a
    btfsc status,bit2 ; skip if zero bit not set
    bsf Preg_a,bit1 ; product term = 1

EVAL_P Preg_a,bit2,P2_x,P2_a
    movf input,W
    xorlw P2_x
    andlw P2_a
    btfsc status,bit2 ; skip if zero bit not set
    bsf Preg_a,bit2 ; product term = 1

EVAL_P Preg_a,bit3,P3_x,P3_a
    movf input,W
    xorlw P3_x
    andlw P3_a
    btfsc status,bit2 ; skip if zero bit not set
    bsf Preg_a,bit3 ; product term = 1
EVAL_P Preg_a,bit4,P4_x,P4_a
  movf input,W ;
  xorlw P4_x ;
  andlw P4_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_a,bit4 ; product term = 1

EVAL_P Preg_a,bit5,P5_x,P5_a
  movf input,W ;
  xorlw P5_x ;
  andlw P5_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_a,bit5 ; product term = 1

EVAL_P Preg_a,bit6,P6_x,P6_a
  movf input,W ;
  xorlw P6_x ;
  andlw P6_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_a,bit6 ; product term = 1

EVAL_P Preg_a,bit7,P7_x,P7_a
  movf input,W ;
  xorlw P7_x ;
  andlw P7_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_a,bit7 ; product term = 1

EVAL_P Preg_b,bit0,P8_x,P8_a
  movf input,W ;
  xorlw P8_x ;
  andlw P8_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_b,bit0 ; product term = 1

EVAL_P Preg_b,bit1,P9_x,P9_a
  movf input,W ;
  xorlw P9_x ;
  andlw P9_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_b,bit1 ; product term = 1

EVAL_P Preg_b,bit2,P10_x,P10_a
  movf input,W ;
  xorlw P10_x ;
  andlw P10_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_b,bit2 ; product term = 1

EVAL_P Preg_b,bit3,P11_x,P11_a
  movf input,W ;
  xorlw P11_x ;
  andlw P11_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_b,bit3 ; product term = 1

EVAL_P Preg_b,bit4,P12_x,P12_a
  movf input,W ;
  xorlw P12_x ;
  andlw P12_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_b,bit4 ; product term = 1

EVAL_P Preg_b,bit5,P13_x,P13_a
  movf input,W ;
  xorlw P13_x ;
  andlw P13_a ;
  btfsc status,bit2 ; skip if zero bit not set
PLD Replacement

bsf Preg_b,bit5 ; product term = 1

EVAL_P Preg_b,bit6,Pl4_x,Pl4_a
movf input,W ;
xorlw P14_x ;
andlw P14_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_b,bit6 ; product term = 1

EVAL_P Preg_b,bit7,Pl5_x,Pl5_a
movf input,W ;
xorlw P15_x ;
andlw P15_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_b,bit7 ; product term = 1

EVAL_P Preg_c,bit0,Pl6_x,Pl6_a
movf input,W ;
xorlw P16_x ;
andlw P16_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit0 ; product term = 1

EVAL_P Preg_c,bit1,Pl7_x,Pl7_a
movf input,W ;
xorlw P17_x ;
andlw P17_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit1 ; product term = 1

EVAL_P Preg_c,bit2,Pl8_x,Pl8_a
movf input,W ;
xorlw P18_x ;
andlw P18_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit2 ; product term = 1

EVAL_P Preg_c,bit3,Pl9_x,Pl9_a
movf input,W ;
xorlw P19_x ;
andlw P19_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit3 ; product term = 1

EVAL_P Preg_c,bit4,P20_x,P20_a
movf input,W ;
xorlw P20_x ;
andlw P20_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit4 ; product term = 1

EVAL_P Preg_c,bit5,P21_x,P21_a
movf input,W ;
xorlw P21_x ;
andlw P21_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit5 ; product term = 1

EVAL_P Preg_c,bit6,P22_x,P22_a
movf input,W ;
xorlw P22_x ;
andlw P22_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_c,bit6 ; product term = 1

EVAL_P Preg_c,bit7,P23_x,P23_a
movf input,W ;
xorlw P23_x ;
andlw P23_a ;
PLD Replacement

0080 0643   btfsc  status,bit2 ; skip if zero bit not set
0081 05F0   bsf    Preg_c,bit7 ; product term = 1

or_pl    EVAL Y  OR_a0,OR_b0,OR_c0,bit0
LOCAL     SETBIT ;
0082 020E   movf  Preg_a,W ;
0083 0EE0   andlw OR_a0 ;
0084 0743   btfss status,bit2 ;
0085 0A8D   goto  SETBIT ;
0086 020F   movf  Preg_b,W ;
0087 0EE7   andlw OR_b0 ;
0088 0743   btfss status,bit2 ;
0089 0A8D   goto  SETBIT ;
008A 0210   movf  Preg_c,W ;
008B 0E00   andlw OR_c0 ;
008C 0743   btfss status,bit2 ;
008D 050D   SETBIT   bsf    Y_reg,bit0 ;

EVAL Y  OR_a1,OR_b1,OR_c1,bit1
LOCAL     SETBIT ;
008E 020E   movf  Preg_a,W ;
008F 0E3F   andlw OR_a1 ;
0090 0743   btfss status,bit2 ;
0091 0A99   goto  SETBIT ;
0092 020F   movf  Preg_b,W ;
0093 0E27   andlw OR_b1 ;
0094 0743   btfss status,bit2 ;
0095 0A99   goto  SETBIT ;
0096 0210   movf  Preg_c,W ;
0097 0E00   andlw OR_c1 ;
0098 0743   btfss status,bit2 ;
0099 052D   SETBIT   bsf    Y_reg,bit1 ;

EVAL Y  OR_a2,OR_b2,OR_c2,bit2
LOCAL     SETBIT ;
009A 020E   movf  Preg_a,W ;
009B 0EFB   andlw OR_a2 ;
009C 0743   btfss status,bit2 ;
009D 0A55   goto  SETBIT ;
009E 020F   movf  Preg_b,W ;
009F 0E2F   andlw OR_b2 ;
00A0 0743   btfss status,bit2 ;
00A1 0A55   goto  SETBIT ;
00A2 0210   movf  Preg_c,W ;
00A3 0E00   andlw OR_c2 ;
00A4 0743   btfss status,bit2 ;
00A5 054D   SETBIT   bsf    Y_reg,bit2 ;

EVAL Y  OR_a3,OR_b3,OR_c3,bit3
LOCAL     SETBIT ;
00A6 020E   movf  Preg_a,W ;
00A7 0E6D   andlw OR_a3 ;
00A8 0743   btfss status,bit2 ;
00A9 0AB1   goto  SETBIT ;
00AA 020F   movf  Preg_b,W ;
00AB 0E79   andlw OR_b3 ;
00AC 0743   btfss status,bit2 ;
00AD 0AB1   goto  SETBIT ;
00AE 0210   movf  Preg_c,W ;
00AF 0E00   andlw OR_c3 ;

© 1993 Microchip Technology Inc.
PLD Replacement

00B0 0743
00B1 056D
  SETBIT
  btfss status, bit2
  bsf Y_reg, bit3

EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
  LOCAL  SETBIT
  00B2 020E
    movf Preg_a, W
  00B3 0B45
    andlw OR_a4
  00B4 0743
    btfss status, bit2
  00B5 0AB0
    goto SETBIT

  00B6 020F
    movf Preg_b, W
  00B7 0BED
    andlw OR_b4
  00B8 0743
    btfss status, bit2
  00B9 0AB0
    goto SETBIT

  00B0 0210
    movf Preg_c, W
  00B1 0B00
    andlw OR_c4
  00B2 0743
    btfss status, bit2
  00B3 058D
    SETBIT
    bsf Y_reg, bit4

EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
  LOCAL  SETBIT
  00B4 020E
    movf Preg_a, W
  00B5 0B71
    andlw OR_a5
  00C0 0743
    btfss status, bit2
  00C1 0AC9
    goto SETBIT

  00C2 020F
    movf Preg_b, W
  00C3 0EDE
    andlw OR_b5
  00C4 0743
    btfss status, bit2
  00C5 0AC9
    goto SETBIT

  00C6 0210
    movf Preg_c, W
  00C7 0B00
    andlw OR_c5
  00C8 0743
    btfss status, bit2
  00C9 05AD
    SETBIT
    bsf Y_reg, bit5

EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
  LOCAL  SETBIT
  00CA 020E
    movf Preg_a, W
  00CB 0B7C
    andlw OR_a6
  00CC 0743
    btfss status, bit2
  00CD 0AD5
    goto SETBIT

  00CE 020F
    movf Preg_b, W
  00CF 0EDE
    andlw OR_b6
  00D0 0743
    btfss status, bit2
  00D1 0AD5
    goto SETBIT

  00D2 0210
    movf Preg_c, W
  00D3 0B00
    andlw OR_c6
  00D4 0743
    btfss status, bit2
  00D5 05CD
    SETBIT
    bsf Y_reg, bit6

EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
  LOCAL  SETBIT
  00D6 020E
    movf Preg_a, W
  00D7 0B00
    andlw OR_a7
  00D8 0743
    btfss status, bit2
  00D9 0AE1
    goto SETBIT

  00DA 020F
    movf Preg_b, W
  00DB 0B00
    andlw OR_b7
  00DC 0743
    btfss status, bit2
  00DD 0AE1
    goto SETBIT

  00DE 0210
    movf Preg_c, W
  00DF 0B00
    andlw OR_c7
  00E0 0743
    btfss status, bit2

© 1993 Microchip Technology Inc. DS00511B-page 13
2-71
APPENDIX B: PLA IMPLEMENTATION: CODE EFFICIENT APPROACH

MPASM B0.54

;********************************************************************************
; pla1b.asm :
; This procedure implements a simple AND-OR PLA with:
; 8 inputs A7 A6 A5 A4 A3 A2 A1 A0
; 24 product terms P23 P22 ..., P0
; 8 outputs Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
; The eight inputs are assumed to be connected to PORT RB such that
; RB0 = A0, RB1 = A1, ..., RB7 = A7.
; The outputs are programmed to appear on port RC such that
; RC0 = Y0, RC1 = Y1, ..., RC7 = Y7.
; This implementation optimizes program memory usage over
; speed
;********************************************************************************
;
; define RAM locations used:
;
LIST P=16C57

input equ d"12" ; RAM location 12 holds input
Y_reg equ d"13" ; holds output result

Preg_a equ d"14" ; Product terms P0 to P7. Preg_a<0> = P0
Preg_b equ d"15" ; Product terms P8 to P15. Preg_b<0> = P8
Preg_c equ d"16" ; Product terms P16 to P23. Preg_c<0> = P16

Pn_x equ d"18" ;
Pn_a equ d"19" ;
OR_a equ d"20" ;
GR_b equ d"21" ;
GR_c equ d"22" ;

; define some constants and file addresses:
;
bit0 equ 0 ;
bit1 equ 1 ;
bit2 equ 2 ;
bit3 equ 3 ;
bit4 equ 4 ;
bit5 equ 5 ;
bit6 equ 6 ;
bit7 equ 7 ;
status equ 3 ;
port_b equ 6 ;
port_c equ 7 ;
; define the AND plane programming variables:

```
0000  P0_x    equ b"00000000"
0000F P0_a    equ b"00000111"
0001   P1_x    equ b"00000001"
0000F P1_a    equ b"00000111"
0002   P2_x    equ b"00000010"
0000F P2_a    equ b"00000111"
0003   P3_x    equ b"00000011"
0000F P3_a    equ b"00000111"
0004   P4_x    equ b"00000100"
0000F P4_a    equ b"00000111"
0005   P5_x    equ b"00000010"
0000F P5_a    equ b"00000111"
0006   P6_x    equ b"00000010"
0000F P6_a    equ b"00000111"
0007   P7_x    equ b"00000011"
0000F P7_a    equ b"00000111"
0008   P8_x    equ b"00000100"
0000F P8_a    equ b"00000111"
0009   P9_x    equ b"00000010"
0000F P9_a    equ b"00000111"
000A   P10_x   equ b"00000101"
0000F P10_a   equ b"00000111"
000B   P11_x   equ b"00000111"
0000F P11_a   equ b"00000111"
000C   P12_x   equ b"00001100"
0000F P12_a   equ b"00000111"
000D   P13_x   equ b"00001101"
0000F P13_a   equ b"00000111"
000E   P14_x   equ b"00001110"
0000F P14_a   equ b"00000111"
000F   P15_x   equ b"00000111"
0000F P15_a   equ b"00000111"
0000   P16_x   equ b"00000000"
0000F P16_a   equ b"00000111"
0000   P17_x   equ b"00000000"
0000F P17_a   equ b"00000111"
0000   P18_x   equ b"00000000"
0000F P18_a   equ b"00000111"
0000   P19_x   equ b"00000000"
0000F P19_a   equ b"00000111"
0000   P20_x   equ b"00000000"
0000F P20_a   equ b"00000111"
0000   P21_x   equ b"00000000"
0000F P21_a   equ b"00000111"
0000   P22_x   equ b"00000000"
0000F P22_a   equ b"00000111"
0000   P23_x   equ b"00000000"
0000F P23_a   equ b"00000111"
```

; define OR plane programming variables:

```
00ED   OR_a0    equ b"11101101" ; for output Y0
0007   OR_b0    equ b"11010111" ;
0000   OR_c0    equ b"00000000" ;
009F   OR_a1    equ b"10011111" ; for output Y1
0027   OR_b1    equ b"00100111" ;
0000   OR_c1    equ b"00000000" ;
00FB   OR_a2    equ b"11111011" ; for output Y2
002F   OR_b2    equ b"00101111" ;
0000   OR_c2    equ b"00000000" ;
006D   OR_a3    equ b"01101101" ; for output Y3
0079   OR_b3    equ b"01111011" ;
0000   OR_c3    equ b"00000000" ;
0045   OR_a4    equ b"01000101" ; for output Y4
00FD   OR_b4    equ b"11111101" ;
0000   OR_c4    equ b"00000000" ;
```
PLD Replacement

0071 OR_a5 equ b"01110001" ; for output Y5
00DF OR_b5 equ b"11011111" ;
0000 OR_c5 equ b"00000000" ;
007C OR_a6 equ b"01111100" ; for output Y6
00EF OR_b6 equ b"11101111" ;
0000 OR_c6 equ b"00000000" ;
0000 OR_a7 equ b"00000000" ; for output Y7
0000 OR_b7 equ b"00000000" ;
0000 OR_c7 equ b"00000000" ;

01FF 0A00 begin
org 01ffh
goto main

org 000h
; define macro to evaluate 1 product (AND) term:
0000 090B main call pla88
0001 0A00 goto main

EVAL_P MACRO Preg_x,bit_n,Pn_x,Pn_a
movf input,W ;
xorlw Pn_x ;
andlw Pn_a ;
btfsc status,bit2 ; skip if zero bit not set
bsf Preg_x,bit_n ; product term = 1
ENDM

; define macro to load OR term constants:
EVAL_Y MACRO OR_an,OR_bn,OR_cn,bit_n
movlw OR_an ; load constants
movwf OR_a ;
movlw OR_bn ;
movwf OR_b ;
movlw OR_cn ;
movwf OR_c ;
call EVAL1 ;
btfss status,bit2 ;
bsf Y_reg,bit_n ;
ENDM

; define procedure to evaluate 1 output (OR) term:
EVAL1 movf Preg_a,W ;
andwf OR_a,1 ;

0004 020F movf Preg_b,W ;
0005 0175 andwf OR_b,1 ;

0006 0210 movf Preg_c,W ;
0007 0156 andwf OR_c,1 ;

0008 0114 iorwf OR_a,W ;
0009 0115 iorwf OR_b,W ;
000A 0800 retlw 0 ; W = 1 implies Yn = 1

; now the PLA evaluation procedure:
pla88 movlw 0ffh ; port_b = input
000B 0CFF tris 6 ; read input
000C 0006 movf port_b,W ;
000D 0206 movwf input ; store input in a register
000E 002C clrf Preg_a ; clear Product register a
000F 006E clrf Preg_b ; clear Product register b
0010 006F clrf Preg_c ; clear Product register c
0011 0070 clrf Y_reg ; clear output register
0012 006D
and_pl EVAL_P Preg_a,bit0,P0_x,P0_a
          movf input, W ;
          xorlw P0_x ;
          andlw P0_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit0 ; product term = 1

EVAL_P Preg_a,bit1,P1_x,P1_a
          movf input, W ;
          xorlw P1_x ;
          andlw P1_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit1 ; product term = 1

EVAL_P Preg_a,bit2,P2_x,P2_a
          movf input, W ;
          xorlw P2_x ;
          andlw P2_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit2 ; product term = 1

EVAL_P Preg_a,bit3,P3_x,P3_a
          movf input, W ;
          xorlw P3_x ;
          andlw P3_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit3 ; product term = 1

EVAL_P Preg_a,bit4,P4_x,P4_a
          movf input, W ;
          xorlw P4_x ;
          andlw P4_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit4 ; product term = 1

EVAL_P Preg_a,bit5,P5_x,P5_a
          movf input, W ;
          xorlw P5_x ;
          andlw P5_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit5 ; product term = 1

EVAL_P Preg_a,bit6,P6_x,P6_a
          movf input, W ;
          xorlw P6_x ;
          andlw P6_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit6 ; product term = 1

EVAL_P Preg_a,bit7,P7_x,P7_a
          movf input, W ;
          xorlw P7_x ;
          andlw P7_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_a, bit7 ; product term = 1

EVAL_P Preg_b,bit0,P8_x,P8_a
          movf input, W ;
          xorlw P8_x ;
          andlw P8_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_b, bit0 ; product term = 1

EVAL_P Preg_b,bit1,P9_x,P9_a
          movf input, W ;
          xorlw P9_x ;
          andlw P9_a ;
          btfsc status, bit2 ; skip if zero bit not set
          bsf Preg_b, bit1 ; product term = 1
EVAL_P Preg_b, bit2, P10_x, P10_a
movf input, W ;
xorlw P10_x ;
andlw P10_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_b, bit2 ; product term = 1

EVAL_P Preg_b, bit3, P11_x, P11_a
movf input, W ;
xorlw P11_x ;
andlw P11_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_b, bit3 ; product term = 1

EVAL_P Preg_b, bit4, P12_x, P12_a
movf input, W ;
xorlw P12_x ;
andlw P12_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_b, bit4 ; product term = 1

EVAL_P Preg_b, bit5, P13_x, P13_a
movf input, W ;
xorlw P13_x ;
andlw P13_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_b, bit5 ; product term = 1

EVAL_P Preg_b, bit6, P14_x, P14_a
movf input, W ;
xorlw P14_x ;
andlw P14_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_b, bit6 ; product term = 1

EVAL_P Preg_b, bit7, P15_x, P15_a
movf input, W ;
xorlw P15_x ;
andlw P15_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_b, bit7 ; product term = 1

EVAL_P Preg_c, bit0, P16_x, P16_a
movf input, W ;
xorlw P16_x ;
andlw P16_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_c, bit0 ; product term = 1

EVAL_P Preg_c, bit1, P17_x, P17_a
movf input, W ;
xorlw P17_x ;
andlw P17_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_c, bit1 ; product term = 1

EVAL_P Preg_c, bit2, P18_x, P18_a
movf input, W ;
xorlw P18_x ;
andlw P18_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_c, bit2 ; product term = 1

EVAL_P Preg_c, bit3, P19_x, P19_a
movf input, W ;
xorlw P19_x ;
andlw P19_a ;
btfs C status, bit2; skip if zero bit not set
bsf Preg_c, bit3 ; product term = 1
EVAL_P Preg_c,bit4,P20_x,P20_a
  movf input,W ;
  xorlw P20_x ;
  andlw P20_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_c,bit4 ; product term = 1

EVAL_P Preg_c,bit5,P21_x,P21_a
  movf input,W ;
  xorlw P21_x ;
  andlw P21_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_c,bit5 ; product term = 1

EVAL_P Preg_c,bit6,P22_x,P22_a
  movf input,W ;
  xorlw P22_x ;
  andlw P22_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_c,bit6 ; product term = 1

EVAL_P Preg_c,bit7,P23_x,P23_a
  movf input,W ;
  xorlw P23_x ;
  andlw P23_a ;
  btfsc status,bit2 ; skip if zero bit not set
  bsf Preg_c,bit7 ; product term = 1

or_pl EVAL_Y OR_a0,OR_b0,OR_c0,bit0
  movlw OR_a0 ; load constants
  movwf OR_a ;
  movlw OR_b0 ;
  movwf OR_b ;
  movlw OR_c0 ;
  movwf OR_c ;
  call EVAL1 ;
  btfss status,bit2 ;
  bsf Y_reg,bit0 ;

EVAL_Y OR_a1,OR_b1,OR_c1,bit1
  movlw OR_a1 ; load constants
  movwf OR_a ;
  movlw OR_b1 ;
  movwf OR_b ;
  movlw OR_c1 ;
  movwf OR_c ;
  call EVAL1 ;
  btfss status,bit2 ;
  bsf Y_reg,bit1 ;

EVAL_Y OR_a2,OR_b2,OR_c2,bit2
  movlw OR_a2 ; load constants
  movwf OR_a ;
  movlw OR_b2 ;
  movwf OR_b ;
  movlw OR_c2 ;
  movwf OR_c ;
  call EVAL1 ;
  btfss status,bit2 ;
PLD Replacement

00A5 054D               bsf Y_reg,bit2 

EVAL_Y OR_a3,OR_b3,OR_c3,bit3
    movlw OR_a3        ; load constants
    movwf OR_a         ;
    movlw OR_b3        ;
    movwf OR_b         ;
    call EVAL1         ;
    btfss status,bit2 ;
    bsf Y_reg,bit3    ;

EVAL_Y OR_a4,OR_b4,OR_c4,bit4
    movlw OR_a4        ; load constants
    movwf OR_a         ;
    movlw OR_b4        ;
    movwf OR_b         ;
    call EVAL1         ;
    btfss status,bit2 ;
    bsf Y_reg,bit4    ;

EVAL_Y OR_a5,OR_b5,OR_c5,bit5
    movlw OR_a5        ; load constants
    movwf OR_a         ;
    movlw OR_b5        ;
    movwf OR_b         ;
    call EVAL1         ;
    btfss status,bit2 ;
    bsf Y_reg,bit5    ;

EVAL_Y OR_a6,OR_b6,OR_c6,bit6
    movlw OR_a6        ; load constants
    movwf OR_a         ;
    movlw OR_b6        ;
    movwf OR_b         ;
    call EVAL1         ;
    btfss status,bit2 ;
    bsf Y_reg,bit6    ;

EVAL_Y OR_a7,OR_b7,OR_c7,bit7
    movlw OR_a7        ; load constants
    movwf OR_a         ;
    movlw OR_b7        ;
    movwf OR_b         ;
    call EVAL1         ;
    btfss status,bit2 ;
    bsf Y_reg,bit7    ;

; Y_reg now contains 8 output values:
    clrw               ;
    tris 7             ; port_c = output
    movf Y_reg,W       ;
    movwf port_c       ; Y_reg -> port_c
    retlw 0            ;

    nop

END

Errors : 0
Warnings : 0
INTRODUCTION

This application note describes a method for implementing analog to digital conversion on the PIC 16C5X series of microcontrollers. The converter requires only four external components and is software and hardware configurable for conversion resolutions from 6 bits up to 10 bits and conversion times of 250us or longer. The method is useable for both voltage and current conversion and uses a software calibration technique that compensates for time and temperature drift as well as component errors. The PIC16C5X microcontrollers are ideal for simple analog applications because:

* Very low cost.
* Few external components required.
* Fully programmable. PIC16C5X microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* Calibration in software for improved measurement accuracy.
* Power savings using PIC16C5X’s SLEEP mode.
* PIC16C5X’s output pins have large, current source/sink capability to drive LED’s directly.

THEORY OF OPERATION

The application uses a capacitive charging circuit (see Figure 1) to convert the input voltage to time, which can be easily measured using a microcontroller. First, the reference voltage is applied to the input voltage to current converter (U1). The equivalent circuit is shown in Figure 2. This circuit provides a linearly variable current as a function of input voltage. The logarithmic characteristic that would occur if the input voltage was applied directly to an RC is not present. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the resistor and capacitor, changes in the input threshold voltage and temperature variations. After the software calibration value is measured, the capacitor is discharged (see Figure 3) and the input voltage is connected to Vin. The time to trip the threshold is measured for the input voltage and compared to the calibration value to determine the actual input voltage.

FIGURE 1 - VOLTMETER A TO D CONVERTER

FIGURE 2 - VOLTMETER MEASUREMENT CYCLE

FIGURE 3 - VOLTMETER DISCHARGE CYCLE
Analog to Digital Conversion

CIRCUIT CONFIGURATION

The values of R and C are selected based upon the number of bits of resolution required.

RC = (Vi • T)/Vt

Where:

Vi = Lowest voltage to be measured (at least ten lsb's)
T = Time to do the number of bits of resolution desired
Vt = Threshold voltage of the PIC16C5X input being used

Actual value for RC should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use a 3 volt input and 8 bits resolution with a 8 MHz clock and 6 instruction cycles per count:

Vi = 100 mV
T = 256 *1/8 MHz * 4 clocks/cycle * 6 cycles = 768 uS
Vt = 3.0V (est)

For input voltages greater than 3 volts a resistor divider network should be used to keep the maximum voltage on Vin to less than 3 Volts. For best performance the reference voltage should be between 2 and 3 volts.

The circuit can also be used as a current mode A to D converter. In this case the input voltage to current converter is not needed and the reference current and input current are both routed via analog switches directly into the capacitor.

FIGURE 4 - TRANSMISSION FLOW CHART

CIRCUIT PERFORMANCE

The calibration cycle removes all first order errors (offset, gain, R and C inaccuracy, power supply voltage and temperature) except the reference voltage drift. Any change in the reference voltage, including noise, between the calibration cycle and the measurement cycle may result in measurement errors. Other error sources are analog switch leakage, resistor and capacitor nonlinearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the converter to be accurate within +/- 1% of full scale.

Example

Assembly code implementing the circuit of figure 1 is listed in Appendix A: This code measures the time up to 16 bits and calculates the results using 16 bit multiply and divide subroutines. In actual applications, if measurement accuracy permits, it may be advantageous to use 8 bits. The math code can be substantially reduced and the measure time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division
APPENDIX A:

MPASM B0.54
VOLTOMETER/AD CONVERTER PROGRAM REV 3-29-90

TITLE 'VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90'
LIST P=16C54,F=inhx16,n=0

EQU
EQU
EQU
EQU
I:QD
ORG 1FF
GOTO VOLTS
;PROGRAM CODE
ORG 0
;SUBROUTINES

MADD MOVF ACCA\+1,W
0000 0209
ADDWF ACCB\+1
0001 01EB
BTFSC 3,0
0002 0603
INCF ACCB
0003 02AA
MOVF ACCA,W
0004 0208
ADDWF ACCB
0005 01EA
RETLW 0
0006 0800
NOP
0007 0000

MPY CALL SETUP
0008 0915
RFF ACCD
0009 032E
RFF ACCD\+1
000A 032F
SKPNC
000B 0603
CALL MADD
000C 0900
RRF ACCB
000D 032A
RRF ACCB\+1
000E 032B
RRF ACCC
000F 032C
RRF ACCC\+1
0010 032D
DECF DSZ TEMP
0011 02F4
RETLW 0
0012 0A09
GOTO MLOOP
0013 0800
NOP
0014 0000

SETUP MOVLW 10
0015 0C10
MOVF TEMP
0016 0034
MOVWF ACCB,W
0017 020A
MOVWF ACCD
0018 002E
MOVWF ACCB\+1,W
0019 020B
MOVWF ACCD\+1
001A 002F
MOVWF ACCC
001B 020C
MOVWF ACCC\+1
001C 0030
MOVF ACCE
001D 020D
MOVF ACCC\+1,W
001E 0031
MOVWF ACCC\+1
001F 006A
CLRF ACCB
0020 006B
CLRF ACCC
0021 0800
RETLW 0

NOP
0022 0000
DIV CALL SETUP
0023 0915
MOVLW 20
0024 0020
MOVF TEMP
0025 0034
MOVF ACCC
0026 006C
CLRF ACCB
0027 006D
CLRF ACCC\+1

© 1993 Microchip Technology Inc.
Analog to Digital Conversion

0028 0403    DLOOP    CLRC
0029 0371    RLF    ACC+1
002A 0370    RLF    ACC
002B 036F    RLF    ACC+1
002C 036E    RLF    ACCD
002D 036D    RLF    ACCC+1
002E 036C    RLF    ACC
002F 0208    MOVF    ACCA+W
0030 008C    SUBWF    ACCC+W
0031 0743    SKPZ
0032 0A35    GOTO    NOCHK
0033 0209    MOVF    ACCA+1,W
0034 008C    SUBWF    ACCC+1,W
0035 0703    NOCHK    SKPC
0036 036E    RLF    ACCD+1
0037 036D    RLF    ACCC
0038 036C    RLF    ACCB
0039 0208    MOVF    ACCA+W
003A 008C    SUBWF    ACCC+W
003B 0209    MOVF    ACCA+1,W
003C 036B    RLF    ACCB+1
003D 036A    RLF    ACCB
003E 020A    MOVF    ACCD+1,W
003F 0209    MOVF    ACCA+W
0040 0209    MOVF    ACCC+1,W
0041 008C    SUBWF    ACCC+W
0042 008D    MOVF    ACCA,W
0043 008C    MOVF    ACCC,W
0044 008D    MOVF    ACCD,W
0045 008C    MOVF    ACCA+W
0046 008D    MOVF    ACCC+W
0047 008C    MOVF    ACCD+W
0048 008D    MOVF    ACCA+1,W
0049 008C    MOVF    ACCC+1,W
004A 008D    MOVF    ACCD+1,W
004B 008C    MOVF    ACCA+1,W
004C 008D    MOVF    ACCC+1,W
004D 008E    MOVF    ACCD+1,W
004E 008F    MOVF    ACCA+1,W
004F 0389    TLOOP    INCFSZ    ACCA+1
0050 0A54    GOTO    ENDCHK
0051 0388    INCFSZ    ACCA
0052 0A54    GOTO    ENDCHK
0053 0A56    GOTO    END_M
0054 0701    ENDCHK    BTFSS    1,0
0055 040F    MOVLW    B'00000000'
0056 0006    TRIS    6
0057 0005    TRIS    5
0058 0005    TRIS    4
0059 0005    TRIS    3
005A 0005    TRIS    2
005B 0005    TRIS    1
005C 0005    TRIS    0
005D 0005    TRIS    3
005E 0005    TRIS    2
005F 0005    TRIS    1
0060 0943    MEAS    CALL    DSCHRG
0061 0C0A    MOVFW    B'00000000'
0062 0209    MOVWF    ACCA+W
0063 0209    MOVWF    ACCC+W
0064 0209    MOVWF    ACC+1,W
0065 0333    MOVWF    TMEAS+1
0066 0328    MOVWF    ACCA+W
0067 0328    MOVWF    TMEAS
0068 0328    MOVWF    TMEAS
0069 0328    MOVWF    TMEAS
Analog to Digital Conversion

0126 064  209  MOVF  ACCA+1,W
0127 065  033  MOVWF  TMEAS+1 ;STORE LSB
0128 066  208  MOVF  ACCA,W
0129 067  032  MOVWF  TMEAS ;STORE MSB
0130 068  026  MOVWF  6
0131 069  943  CALL  DSCHRG ;CHARGE CAPACITOR TO VREF
0132 06A  02 6  MOVWF  6
0133 06B  C09  MOVLW  B'00000101' ;S1 AND S3 ON
0134 06C  02 6  MOVWF  6
0135 06D  C09  MOVLW  B'00001001' ;S1 AND S4 ON
0136 06E  94C  CALL  M_TIME ;MEASURE TIME
0137 06F  CA4  MOVLW  VCALLS
0138 070  02A  MOVWF  ACCB+1
0139 071  C60  MOVLW  VCALLMS
0140 072  02A  MOVWF  ACCB
0141 073  908  CALL  MPY ;MULTIPLY ACCA(TCAL) * ACCB(VREF)
0142 074  213  MOVF  TMEAS+1,W
0143 075  029  MOVWF  ACCA+1
0144 076  212  MOVF  TMEAS,W
0145 077  028  MOVWF  ACCA
0146 078  923  CALL  DIV ;DIVIDE ACCB(TCAL) * V) BY ACCA(TMEAS) 0150
0147 079  A58  GOTO  VOLTS
0148 07A  0150  END

%ASM-I, No Errors, No Warnings
INTRODUCTION

This application note describes a method for implementing an ohmmeter or resistance type temperature sensor using the PIC16C5X series of microcontrollers. The ohmmeter requires only two external components and is software and hardware configurable for resistance measurement with resolutions from 6 bits up to 10 bits with measurement times of 250µs (6 bits at 8 MHz) or longer. The method uses a software calibration technique that compensates for voltage, time, and temperature drift as well as component errors. The PIC16C5X Microcontrollers are ideal for simple analog applications because:

* Very low cost.
* Few external components required.
* Fully programmable. PIC16C5X Microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* Calibration in software for improved measurement accuracy.
* Power savings using PIC16C5X's SLEEP mode.
* PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

The application uses a capacitive charging circuit (Figure 1) to convert the resistance to time, which can be easily measured using a microcontroller. First, a reference voltage (usually VDD) is applied to a calibration resistor, Rc. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the capacitor, changes in the input threshold voltage and temperature variations. After C is discharged, the reference voltage is applied to the resistance to be measured (or thermistor). The time to trip the threshold is then measured and compared to the calibration value to determine the actual resistance (Figure 2). In the temperature sensing mode, the temperature is calculated using a lookup table.

CIRCUIT CONFIGURATION

The values of Rc and C are selected based upon the number of bits of resolution required. Rc should be approximately one half the largest value resistance to be measured and:

$$C = \frac{-T}{R_m \cdot \ln \left(1 - \frac{V_t}{V_r}\right)}$$

Where:

* $V_r$ = Reference voltage
* $T$ = Time to do the number of bits of resolution desired
* $V_t$ = Threshold voltage of the PIC input being used
* $R_m$ = Maximum resistance value to be measured

Actual value for C should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.
Ohmeter/Temperature Sensor

For example use $R_m=200\,K$ for 8 bits resolution with a 8 MHz clock, $V_r=5\,V$, $V_t=3\,V$, $R_c=100\,K$ and 6 instruction cycles per count:

$$T = 256\,\text{counts} \times \frac{1}{8\,\text{MHz}} \times 4\,\text{clocks/instruction} \times 6\,\text{instructions/count} = 768\,\mu\text{s}$$

$$C = 4200\,\text{pF} \quad \text{[Use 3900\,pF]}$$

CIRCUIT PERFORMANCE

The calibration cycle removes all first order errors (offset, gain, $C$ inaccuracy, power supply voltage and temperature) except $R$ absolute accuracy. A low drift resistor should be selected for $R$ and its value stored in software to reduce measurement errors. Other error sources are I/O pin leakage, resistor and capacitor non-linearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the ohmmeter to be accurate within +/- 1% over one decade.

Example

The assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures time up to 16 bits (65535 measure cycles) and calculates the results using 16 bit multiply and divide subroutines. In actual applications, it is more efficient to use 8 bit measurements if application accuracies permit. The math code will be substantially reduced and measurement time is reduced by the simpler code and shorter count.

Author: Doug Cox  
Logic Products Division

FIGURE 3 - TRANSMISSION FLOW CHART

```
START
  Initial Setup
  Call Discharge
  Setup Outputs for Calibration
  Call Measure
  Call Discharge
  Setup Outputs for Measure
  Call Measure
  Compute Result
  END

DISCHARGE
  Set Cap to Ground
  WAIT
  Open Cap
  RETURN

MEASURE
  Clear RTCC
  Clear Counter
  Increment Counter
  Check for RTCC Trip
    YES
    RETURN
    NO
```
APPENDIX A:

LIST P=16C54,F=inhx8M

0008 ACA EQU 8
000A ACCB EQU 0A
000C ACCC EQU 0C
000E ACCD EQU 0E
0010 ACCE EQU 10
0012 TCAL EQU 12
0014 TEMP EQU 14

002F RCALMS EQU 2F ;RCAL MSB VALUE IN HEX
003C RCALLS EQU 3C ;RCAL LSB VALUE IN HEX

ORG 1FF
GOTO OHMS
ORG 0

0000 0209 MADD MOVF ACCA+1,W
0001 01EB ADDWF ACCB+1 ;ADD LSB
0002 0603 BTFSC 3,0 ; ADD IN CARRY
0003 02AA INCF ACCB
0004 0012 MOVF ACCA,W
0005 0800 RETLW 0
0007 0000 NOP

0008 0915 MEY CALL SETUP ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E MSLOOP RRF ACCD ;ROTATE D RIGHT
000A 032F RRF ACCD+1
000B 0603 SKPNC ; NEED TO ADD?
000C 0900 CALL MADD
000D 032A RRF ACCB
000E 032B RRF ACCB+1
000F 032C RRF ACCC
0010 032D RRF ACCC+1
0011 02F4 DECFSZ TEMP ;LOOP UNTIL ALL BITS CHECKED
0012 0A09 GOTO MSLOOP
0013 0800 RETLW 0
0014 0000 NOP

0015 0C10 SETUP MOVFLW 10
0016 0034 MOVWF TEMP
0017 020A MOVF ACCB,W ;MOVE B TO D
0018 002E MOVWF ACCD
0019 020B MOVF ACCB+1,W
001A 002F MOVWF ACCD+1
001B 020C MOVF ACCC,W
001C 0030 MOVF ACCC+1
001D 020D MOVF ACCE
001E 0031 MOVWF ACCE+1
001F 006A CLRF ACCB
0020 006B CLRF ACCB+1
0021 0800 RETLW 0
0022 0000 NOP

0023 0915 DIV CALL SETUP
0024 0C20 MOVFLW 20
0025 0034 MOVWF TEMP
0026 006C CLRWF ACCC
0027 006D CLRWF ACCC+1
0028 0403 DLOOP CLRRC
0029 0371 RLF ACCE+1
002A 0370 RLF ACCE
002B 036F RLF ACCD+1

© 1993 Microchip Technology Inc.

2-87
Ohmeter/Temperature Sensor

002C 036E  RLF  ACCD
002D 036D  RLF  ACCC+1
002E 036C  RLF  ACCC
002F 0208  MOVF  ACCA,W
0030 008C  SUBWF  ACCC,W ;CHECK IF A>C
0031 0743  SKPZ
0032 0A35  GOTO  NOCHK
0033 0209  MOVF  ACCA+1,W
0034 008D  SUBWF  ACCC+1,W ;IF MSB EQUAL THEN CHECK LSB
0035 0703  NOCHK  SKPC ;CARRY SET IF C>A
0036 0A3E  GOTO  NOGO
0037 0209  MOVF  ACCA+1,W ;C-A INTO C
0038 00AD  DECFSZ  TEMP ;LOOP UNTILL ALL BITS CHECKED
0039 0703  NOC  HK  SKPC :CARRY SET IF C>A
003A 0209  MOVF  ACCA+1,W
003B 000B  SUBWF  ACCC
003C 00AC  SETC :SHIFT A 1 INTO B (RESULT)
003D 0503  NOGO  RLF  ACCB+1
003E 036A  RLF  ACCB
0040 02F4  DECFSZ  TEMP ;LOOP UNTILL ALL BITS CHECKED
0041 0A28  GOTO  DLOOP
0042 0800  RETLW 0
0043 0C0B  DSCHRG  MOVLW  B’00001011’ ;ACTIVATE RA2
0044 0005  TRIS 5
0045 0CFF  MOVLW  OFF
0046 0034  MOVF  TEMP
0047 02F4  LOOP  DECFSZ  TEMP ;WAIT
0048 0A47  GOTO  NOGO
0049 0C0F  MOVLW  B’00000111’ ;ALL OUTPUTS OFF
004A 0005  TRIS 5
004B 0800  RETLW 0
004C 0061  M_TIME  CLRWF 1 ;CLEAR RTCC
004D 0069  CLRWF  ACCA+1
004E 0068  CLRWF  ACCA
004F 0369  TLOOP  INCFSZ  ACCA+1
0050 0A54  GOTO  ENDCHK
0051 0368  INCFSZ  ACCA
0052 0A54  GOTO  ENDCHK
0053 0A56  GOTO  END_M
0054 0701  ENDCHK  BTFSS 1,0 ;CHECK FOR RTCC TRIP
0055 0A4F  GOTO  LOOP
0056 0201  END_M  MOVF  1,W
0057 0800  RETLW 0
0058 0C03  OHMS  MOVLW  B’00000011’ ;SET RA0 AND RA1 HIGH (ON WHEN ACTIVATED)
0059 0025  MOVF  5
005A 0C28  MOVLW  B’00101000’ ;SELECT POSITIVE EDGE FOR RTCC
005B 0002  OPTION

005C 0943  CALL  CAL  DSCHRG ;DISCHARGE CAPACITOR
005D 0C0B  MOVLW  B’00001100’ ;ACTIVATE RA0
005E 0005  TRIS 5
005F 094C  CALL  M_TIME ;MEASURE TIME
0060 0209  MOVF  ACCA+1,W
0061 0033  MOVWF  TCAL+1 ;STORE LSB
0062 0208  MOVF  ACCA+1,W
0063 0032  MOVWF  TCAL ;STORE MSB
0064 0943  MEAS  CALL  DSCHRG ;DISCHARGE CAPACITOR
0065 0C0D  MOVLW  B’00001101’ ;ACTIVATE RA1
0066 0005  TRIS 5
0067 094C  CALL  M_TIME ;MEASURE TIME
0068 0C3C  MOVLW  RCALLS ;CALIBRATION LSB VALUE
0069 002B  MOVWF  ACCB+1
006A 0C2F  MOVLW  RCALMS ;CALIBRATION MSB VALUE

© 1993 Microchip Technology Inc.
Ohmeter/Temperature Sensor

006B 002A MOVWF ACCB

006C 0908 CALL MPY ;MULTIPLY ACCA(MEAS) * ACCB(RCAL)

006D 0213 MOVF TCAL+1,W

006E 0029 MOVWF ACCA+1

006F 0212 MOVF TCAL,W

0070 0028 MOVWF ACCA

0071 0923 CALL DIV ;DIVIDE ACCB(MEAS * R) BY ACCA(TCAL)

0072 0A58 GOTO OHMS

END

Errors : 0
Warnings : 0
INTRODUCTION

This application note describes a simple method for measuring parameters from the AC power line. Parameters such as zero crossing, frequency, and relative phase can be measured. The method is useful for measurements on 50, 60, and 400 Hz power systems with voltages up to several hundred volts. The method requires only one external component, a resistor, and is more reliable than previously published methods using capacitors or bulky, expensive transformers.

APPLICATIONS

This measurement method can be used in any application where power line parameters are used for system measurements or control. Typical applications are for switch timing (what part of the power cycle should the system be activated), power factor correction, power measurement, and power line monitor. An additional application is to generate timing or clock functions using the relatively stable power line frequency. The method is also useful for calibrating the oscillator frequency for accurate timing measurements when an inaccurate reference such as an RC oscillator is used to clock the PIC16C5X.

THEORY OF OPERATION

The application takes advantage of the input static protection circuitry that exists on all I/O pins of a CMOS PIC16C5X. These protection circuits are designed to short the inputs to the power supplies when a large overvoltage is applied, thus protecting the chip from static electricity spikes. On the PIC16C5X microcontrollers, this protection circuit is two large P-N diodes on each input (see Figure 1). These diodes will short any voltage higher than VDD to the VDD supply and any voltage less than VSS to the VSS supply. They can take several milliamps of current without any damage to the chip. High voltages can be applied directly to the chip inputs as long as they are current limited.

The-least expensive method of current limiting is using a high value resistor. This method is shown schematically in Figure 2. The power line voltage is current limited by the resistor and then clamped by the input protection diodes internal to the PIC16C5X. A typical input waveform is shown in Figure 3. A 115V AC, 60 cycle sine wave will traverse from 0 to 2 volts in 32 µs so a typical threshold of 2 volts on the PIC16C5X I/O port will permit zero crossing detection accuracy of about 30 µs. If the typical capacitance on an I/O pin is 5 pF, then R should be (T = RC) 6 MEGohm or less for best zero crossing accuracy. A 5 MEGohm resistor with 115V AC applied to it will limit current to 32 µA, a value which is well within the safety margin of the PIC16C5X.
Interfacing to AC Power Lines

The user needs to be aware that the circuit required to connect the RTCC input to AC power line is slightly different. Each of the I/O pins has two diodes for input protection whereas RTCC pin has only one protection diode connecting to Vss (see figure 3). Therefore, it is necessary to connect a diode externally between RTCC pin and Vdd in order to clamp the voltage on RTCC pin to Vdd + 0.6V (approx.). See Figure 4. It is also recommended that resistor R be at least 2MΩ.

RELIABILITY

Reliability of production devices that are directly connected to AC power is always a concern. Two failure modes are possible. First, the series resistor of Figure 1 might fail short, destroying the microcontroller. This is the most unlikely failure mode of a resistor, and resistors are more reliable than transformers or capacitors, which are the alternate components for measuring line parameters. This reliability can be enhanced even further by using two resistors in series. Both would have to fail short to cause catastrophic failure, a very unlikely event. The second possible failure mode is that excessive injection of current into the PIC16C5X input might cause the protection diode to open. This would allow the input to go to the power line peak voltage (162V) and short the input transistor gate oxide, causing device failure. The maximum continuous injection current into an I/O pin is specified ±500 µA. An I/O pin is also capable of handling larger injection current (>100 mA) for a very short period (transient) of time. Therefore higher transient currents due to line voltage surges will be easily handled.

FIGURE 3 - INPUT STRUCTURE AND RTCC PIN

Simplified structure of RTCC and MCLR pins. Grounded gate NMOS device provides ESD and overvoltage protection.

FIGURE 4 - CONNECTING AC POWER LINE TO RTCC PIN

Connecting to RTCC input. R ≥ 2MΩ

FIGURE 5 - INPUT WAVEFORM

Waveform at part pin (RA0) *
R = 100K; Line: 60 Hz, 110 V

Author: Doug Cox
Logic Products Division

© 1993 Microchip Technology Inc.
Implementing Wake-Up on Key Stroke

INTRODUCTION

In certain applications, the PIC16CXX is exercised only when a key is pressed, e.g. remote keyless entry. In such applications, the battery life can be extended by putting the PIC16CXX to sleep during the inactive state and when a key is pressed, the PIC16CXX wakes up does its task and then goes back to sleep.

IMPLEMENTATION

The circuit in Figure 1 depicts an application with two keys. The PIC16C54 is normally in SLEEP mode consuming very little operating current. If either of the two keys is pressed, the PIC16C5X 'wakes up', scans the keys and turns on one of the two LED's. When SW1 is pressed, the green LED is turned on and when SW2 is pressed the red LED is turned on. The LED's are used purely for demonstration purposes. In real life application, a transmission will be completed before putting the PIC16C5X back in sleep. This example can be extended to handle more than two keys.

In the sleep mode, the scan outputs (SCAN1 and SCAN2) are both set to a low logic level. In this state, the capacitor C is fully charged and a high logic level is present at the MCLR pin of the PIC16C5X. When a key is pressed, C discharges through either R2 or R3 (depending on SW1 or SW2 being pressed) and the voltage across C falls rapidly (approx. 1 ms), causing a low at the MCLR pin of the PIC16C5X, which in turn causes the PIC16C5X to wake up and enter its reset state. In reset, the SCAN1 and SCAN2 outputs default to a high impedance mode, so the discharge path for capacitor C is blocked and it charges to a high level through resistor R1. Note that the RC values have been chosen such, that the discharge and charge cycles times are less than the reset time for the PIC16C5X (approx. 18 ms), and certainly far less than the minimum duration of a key-press (approx. 50-100 ms).

After the reset cycle is completed, the code execution momentarily takes the SCAN1 and SCAN2 outputs low in order to sample the key stroke(s). This does not cause the capacitor to discharge since the duration of the low is of the order of 10 micro seconds.

Once the keystroke function has been executed, the program loops until the key has been released, sets the SCAN1 and SCAN2 outputs low and "goes back to sleep". Resistors R4-R8 are not required for functionality. These are recommended to provide protection from electrostatic discharge (ESD). Switches SW1 and SW2, when pressed may frequently pass ESD to the PIC16C54.

FIGURE 1 - TWO KEY INTERFACE TO PIC16C5X

Author: Stan D'Souza
Logic Products Division
Key Stroke Wake-Up

FIGURE 2 - TWO KEY SCAN/WAKE-UP TIMING DIAGRAM

- KEY PRESSED
- WAKE UP OCCURS
- HIGH IMPEDANCE LEVEL
- KEY INPUT SCANNED = 10 µs
- LAST KEY SCAN (KEY RELEASED)

FIGURE 3 - PIC16C5X INTERFACE TO 4 X 4 KEY MATRIX

- 47 K
- 100Ω
- C 0.1µ

4 x 4 key matrix

MCLR

RA0
RA1
RA2
RA3
RB0
RB1
RB2
RB3

PIC16C54

© 1993 Microchip Technology Incorporated
Key Stroke Wake-Up

MPASM B0.54
Key Stroke Wake Up

TITLE "Key Stroke Wake Up"
LIST P = 16C54
"******************************************************
Program demonstrating key stroke wake up for
the PIC16CXX. Program has been implemented for
two keys, but can be extended for more keys.
When SW1 is pressed a green LED lights up.
When SW2 is pressed a red LED lights up.
"******************************************************
Define equates
PC EQU 2
PORT_B EQU 6
SCAN1 EQU 2
SCAN2 EQU 3
SW1 EQU 0
SW2 EQU 1
GRN_LED EQU 4
RED_LED EQU 5
MSEC_20 EQU 0'20'
DB1 EQU 8
GP EQU 8
DB2 EQU 9

;PORT_B ASSIGNMENTS:
  0 -> SW1 INPUT
  1 -> SW2 INPUT
  2 -> SCAN1 OUTPUT
  3 -> SCAN2 OUTPUT
  4 -> GRN_LED OUTPUT
  5 -> RED_LED OUTPUT
  6 & 7 -> ASSIGNED AS DUMMY OUTPUTS

ORG 0

START
CALL INIT_PORT_B ; INITIALIZE PORT B
CALL DELAY ; DELAY 20 MSEC
CALL SCAN.Keys ; GET KEY VALUES
MOVF GP ; SAVE IN RAM
BTFSC GP, SW1 ; SKIP IF SW1 NOT Pressed
CALL TURN_GREEN_ON ; ELSE DO ROUTINE
BTFSC GP, SW2 ; SKIP IF SW2 NOT Pressed
CALL TURN_RED_ON ; ELSE DO ROUTINE

CHKForKey

CALL DELAY ; DELAY FOR 20 MSEC
CALL SCAN.Keys ; GET KEY HIT
XORLW 0 ; EXCL. OR WITH 0
BNZ CHKForKey ; KEY STILL Pressed
; THEN LOOP

NO_KEY_PRESSED

BCF PORT_B, SCAN1 ; SET SCAN LINES LOW
BCF PORT_B, SCAN2 ; / 
SLEEP ; SLEEP

; INITIALIZE PORT B
MOVLW B'00000011' ; config RB0, 1 as i/p’s
TRIS PORT_B ; and RB2-7 as o/p’s
MOVLW 0FFh
MOVF PORT_B ; DEFAULT VALUES FOR PORT B
Key Stroke Wake-Up

RETlw 0 ;RETURN WITH NO ERROR

This routine, scans two keys and returns the following:

0 if no key is pressed
1 if SW1 is pressed
2 if SW2 is pressed
3 if SW1 and SW2 are pressed

SCAN_KEYS

BCF PORT_B, SCAN1 ;ENABLE SCAN FOR SW1
BCF PORT_B, SCAN2 ;ENABLE SCAN FOR SW2
MOVlw B'00000011' ;LOAD MASK IN W
ANDWF PORT_B, 0 ;AND WITH PORT
BSF PORT_B, SCAN1 ;DISABLE SCAN
BSF PORT_B, SCAN2 ;
ADDWF PC, 1 ;GET OFFSET TO TABLE
RETLw 3 ;SW1 AND SW2 PRESSED
RETLw 2 ;SW2 PRESSED
RETLw 1 ;SW1 PRESSED
RETLw 0 ;NO KEY PRESSED

DELAY, IS A APPROX. WAIT FOR 20.4mSECS, FOR A SYSTEM USING A 2 Mhz CRYSTAL CLOCK.

MOVlw MSEC_20
MOVWF DB1

CLRF DB2
DECFSZ DB1
GOTO DLY2
RETLw 0

DECFSZ DB2
GOTO DLY2
GOTO DLY1

TURN_GREEN_ON
BCF PORT_B, GRN_LED
RETLw 0

TURN_RED_ON
BCF PORT_B, RED_LED
RETLw 0

END

Errors : 0
Warnings : 0
INTRODUCTION

Many applications require drive to LEDs along with an interface to a keypad. Implementing such designs usually involves using up significant amounts of the processors I/O lines. This application note describes a method which uses only 16 I/O pins of a PIC16C5X microcontroller to sample a 4x4 keypad matrix, and directly drive four 7 segment LEDs (see Figure 1). Direct drive of the LEDs is possible, because of the high sink and source capabilities of the PIC16C5X microcontroller, thus eliminating the use of external drive transistor, and resulting in reduced cost and complexity of the overall circuit.

Typically applications having LEDs and keypads also keep track of real time, in order to synchronize certain key events. An Industrial Clock/Timer example has been used in this ap note as a demonstration of this technique. The software overhead to keep track of real time is minimal and the user can modify the code to significantly expand the functionality of this circuit.

FIGURE 1 - PIC16C5X INTERFACE TO A SEGMENT DISPLAY AND 4X4 KEYPAD
Multiplexing LEDs/Keypad

PART A: 4X4 KEY MATRIX SAMPLING

Implementation

The 4x4 Key Matrix is connected to port C of the PIC16C5X (Figure 2a). The 4 columns are connected to RC0-RC3 and the 4 rows are connected to RC4-RC7. Each digit is refreshed every 20 ms, with a 5 ms pulse. The keypad is sampled every 20 ms with four 3 µs pulses (Figure 3).

The Keypad sampling is as follows:

1. The columns are connected to output pins, and the rows are connected to input pins.

2. Each column is sequentially driven to a low voltage while at the same instance the four rows are sampled. Since the rows are all held high with pull-up resistors, all four inputs will normally be high. If a key is pressed in a column which is at a low level, that low level will be conducted to the input pin through the closed key and the corresponding row will be sensed as a low.

3. Before a new column is brought low, care should be taken to discharge the input pins (see code section for details).

4. A 50 ms key debounce technique has been implemented in the software, in order to eliminate multiple key strokes.

Notes:

1. Resistors R8-R11 and R12-R14 have been selected such that their ratio is 1:10. This will insure a 0.5 Volt level at the input, when a key is pressed. Also R8-R14 should have a value such that their current contribution to the LEDs segments is negligible.

2. In circuits where there is substantial interference between the key matrix and the LED drive circuit, the alternative circuit (Figure 2b) should be utilized. Diodes in the path of all pins connected to the keypad insure that there is minimal interference from the keypad, when it is not being sampled.

FIGURE 2A • PIC16C5X INDUSTRIAL CLOCK/TIMER SCHEMATIC
PART B: INDUSTRIAL CLOCK/TIMER

Clock Selection

The 4.096 MHz crystal oscillator is the time base. The PIC16C5X internally divides the clock by 4 to give an internal clock of 1.024 MHz. This clock is further divided by 32 (by the prescaler in the OPTIONS register) to give a clock of 32 KHz which is used to increment the RTCC in the PIC16C5X. If the RTCC is initialized to 96, it would overflow to 0 in 5 ms.

\[(256-96) \times (1/32000) = 5.000 \text{ ms}\]

This 5 ms is used to count the seconds, minutes and hours in the clock/timer. It is also used as a time base to update the display digits and sample the keyboard. The clock speed being 4.096 MHz, each instruction will execute in 1 µs. Therefore in 5 ms, approximately 5000 instruction can be executed. This gives us sufficient time to execute a large section of code and not miss the overflow in the RTCC.

**Using a 3.579545 MHz color burst crystal oscillator as a time base**

Some users may want to use a color burst crystal oscillator as a time base, because of its low cost. If a 3.579545 MHz crystal is used, then the internal clock will be 1.117 µs. If this is prescaled by 32, the RTCC will be incremented every 35.758 µs. Initializing the RTCC with 96 will cause it to overflow to 0 in 5.006 ms, giving us an error of 0.12%. This error can be corrected in software by making time adjustments every minute and/or every hour.

**FIGURE 2B - PIC16C5X ALARM CLOCK SCHEMATIC (USING DIODES)**
Multiplexing LEDs/Keypad

FIGURE 3 - KEY SCAN AND LED DIGIT SELECT TIMING

FEATURES

The Flow Chart (Figure 4) shows the sequence of events in the clock/timer software. The clock has the following features:

1. 12 hour clock with a.m./p.m.
2. 12 hour alarm with a.m./p.m.
3. Full function hex keypad (fig. 5).
4. AA audible alarm for 1 minute.
5. 10 minute alarm disable.

SETTING CLOCK/TIMER FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Key Sequence to Activate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Real Time</td>
<td>Set → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set</td>
</tr>
<tr>
<td>View Alarm Time</td>
<td>Alarm (alarm time is displayed for 5 seconds)</td>
</tr>
<tr>
<td>Set Alarm Time</td>
<td>Alarm → Set (must be pressed when alarm LED is flashing) → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set</td>
</tr>
<tr>
<td>Enable/Disable Alarm</td>
<td>Alarm → Alarm (toggles alarm status)</td>
</tr>
<tr>
<td>Disable AA alarm</td>
<td>Disable Alarm (disable audible beep for 10 minutes)</td>
</tr>
<tr>
<td>Clear Alarm</td>
<td>Clear Alarm (clears audible alarm)</td>
</tr>
<tr>
<td>Abort Entry</td>
<td>Clear Entry (aborts data entry mode when setting real and alarm time)</td>
</tr>
</tbody>
</table>

Notes:

1. Valid key strokes will be acknowledged with a beep.
2. Hours and minutes used above correspond to digits 0 - 9 on the keypad.
FIGURE 4 - TIMER/CLOCK FLOW CHART

Start
Initialize RAM
Test LEDs for 2 secs
Update next digit
All digits updated?
Yes
Scan key matrix
Key pressed?
No
Service key
AA alarm on?
Yes
Service AA alarm
No
Update all relevant timers

5ms Timer = 0?
Yes
Update all relevant timers
No

FIGURE 5 - KEYPAD

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>DISABLE ALARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>AM/PM</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>CLEAR ALARM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALARM TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLEAR ENTRY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SET</td>
</tr>
</tbody>
</table>
SUMMARY

This Application Note demonstrates a simple method of interfacing the PIC16C5X to 7 segment LEDs and a keypad. The key features of the PIC16C5X which made this possible are:

1. High sink/source of the I/O ports.
2. Fast instruction cycle for quick key-scan.
3. RISC processor allowing minimal overhead for real time clock maintenance.
4. Re-configurable I/O ports, enabling dual functionality of ports.

Figure 6 depicts a block diagram connecting a PIC16C54/56 to a 4 digit 7 segment LED display and a 4x4 hex keypad. Since only 12 I/O pins are available in the PIC16C54/56, external npn transistor will have to be utilized to sink the current from each digit.

CODE SIZE

Key scan → 97 bytes
Display update → 113 bytes

Author: Stan D'Souza
Logic Products Division
APPENDIX A: CODE LISTING

MPASM B0.54
Alarm Clock

TITLE "Alarm Clock"
LIST P = 16C57

; Define Equates:

PIC57 EQU 7FFH
MSEC5 EQU D'96'

;************************************************************************
; External Osc. used = 4.096Mhz. Prescaler of 32 used, which gives a
; 31.25 microSec increment of the RTCC. If RTCC is intially loaded with 96,
; it would overflow to 0 in 5,000 millisecs. Giving a 0.00% error.

0060

;*************************************************************************

C EQU 0
BEP EQU 0
RTATS EQU 0
DC EQU 1
HR10 EQU 1
Z EQU 2
HR EQU 2
MIN10 EQU 3
MIN EQU 4
FLASH EQU 4
PA0 EQU 5
KEY_BEEP EQU 5
AMPM EQU 5
PA1 EQU 6
F0 EQU 0
KEY_HIT EQU 6
ALED EQU 6
AM_P M EQU 7
COLGN EQU 3
ALRMLED EQU 2
SERVICED EQU 7
ALONOF EQU 0
INAL EQU 1
SILNC EQU 2
INAA EQU 3
INKEYBEP EQU 5

;************************************************************************

RTCC EQU 1
PC EQU 2
STATUS EQU 3
FSR EQU 4
PORT_A EQU 5
PORT_B EQU 6
PORT_C EQU 7

;************************************************************************

MSTMR EQU 8 ; MILLISEC. TIMER
STMR EQU 9 ; SEC. TIMER

;************************************************************************

; Define Real Time Mode Regs (RTM)

MTMR EQU 0A ; MIN. TIMER
HTMR EQU 0A ; HOUR TIMER

; Define Alarm Time Mode Regs (ATM)

MALARM EQU 0C ; MIN. ALARM
HALARM EQU 0D ; HOUR ALARM

; Define Data Entry Mode Regs (DEM)

MENTRY EQU 0E ; MIN. ENTRY
HENTRY EQU 0F ; HOUR ENTRY

;************************************************************************

© 1993 Microchip Technology Incorporated
; DEFINE FLAG REG AND FUNCTION:
FLAG EQU 10
; BIT # 7|6|5|4|3|2|1|0
;-------|---|---|---|---|---|---|---|
; X|X|X|X|X|X|0|1
; X|X|X|X|X|X|1|0
; X|X|X|X|X|Y|0|1
; X|X|X|X|X|Y|1|0
; X|X|X|Y|X|X|0|1
; X|X|X|Y|X|X|1|0
; X|Y|X|X|X|X|0|1
; X|Y|X|X|X|X|1|0
; X = DEFINED ELSEWHERE IN TABLE
; Y = DEFINED AS SHOWN (0/1)

0011
TEMP EQU 11
0012
DIGIT EQU 12
0013
NEW_KEY EQU 13
0014
KEY_NIBL EQU 14
0015
DEBOUNCE EQU 15
0016
MIN_SEC EQU 16 ;MIN/SECONDS TIMER
0017
ENTFLG EQU 17
; flag dedicated to the key entry mode
; BIT # 7|6|5|4|3|2|1|0
;-------|---|---|---|---|---|---|---|
; X|X|X|X|X|X|Y|1
; X|X|X|X|X|Y|X|1
; X|X|X|Y|X|X|1|0
; X|X|X|Y|X|X|0|1
; X|X|Y|X|X|X|1|0
; X|X|Y|X|X|X|0|1
; X|Y|X|X|X|X|1|0
; X|Y|X|X|X|X|0|1
; X = REAL/TIME STATUS
; Y = REAL/ALARM TIME STATUS

0018
ALFLAG EQU 18
; flag dedicated to the alarm
; BIT # 7|6|5|4|3|2|1|0
;-------|---|---|---|---|---|---|---|
; X|X|X|X|X|X|Y|1
; X|X|X|X|X|Y|X|1
; X|X|X|Y|X|X|Y|1
; X|X|X|Y|X|X|1|0
; X|X|X|Y|X|X|0|1
; X|X|X|Y|X|X|0|1
; X|Y|X|X|X|X|Y|1
; X|Y|X|X|X|X|1|0
; X = NOT USED
; Y = NOT USED

0019
AAFLAG EQU 19
; flag dedicated to the AA alarm
001A
AATMR EQU 1A
Port pin definitions:

PORT_A:
- BIT 0  -> BEEPER (ACTIVE LOW) OUTPUT
- BIT 1-3 -> unused I/O

PORT_B: ALL OUTPUTS
- BIT 0&4 -> MSB DIGIT COMMON CATHODE & ALARM
- BIT 1&5 -> 2ND DIGIT CATHODE & COLON
- BIT 2&6 -> 3RD DIGIT COMMON CATHODE & PM
- BIT 3&7 -> LSB DIGIT COMMON CATHODE & AM

PORT_C:
- IN DISPLAY MODE ALL SEG/ANNUN SET AS OUTPUTS
- IN KEY SCAN MODE COLS ARE OUTPUTS ROWS ARE INPUTS

START
GOTO INICLK ; INITIALIZE CLOCK

TEST_HARDWARE

TEST_LOOP
MOVLW d'02'
MOVWF MIN_SEC ; DISPLAY FOR 2 SECS

NORM_TIME
BCF FLAG,0 ; PUT IN REAL TIME
BCF FLAG,1

TIME_LOOP
CALL UPDATE_DISPLAY
BCF STATUS,PA0 ; GOTO PAGE 2
BSF STATUS,P6 ; GOTO PAGE 3

RESET_ATM
CALL RESET_ATM
GOTO TIME_LOOP
Multiplexing LEDs/Keypad

; RESET_ATM
001C 0216 MOVF MIN_SEC,W ; GET MIN/SEC
001D 0E0F ANDLW B'00001111' ; / 
001E 0743 BTFSS STATUS,Z ; Z THEN SKIP
001F 0800 RETLW 0 ; ELSE RETURN
0020 0410 BCF FLAG,0 ; SET TO RTM
0021 0450 BCF FLAG,ALRMLED ; CLEAR LED
0022 0618 BTFSC ALFLAG,ALONGF ; TEST STAT
0023 0550 BSF FLAG,ALRMLED ; SET LED
0024 0800 RETLW 0 ; RETURN

; UPDATE_DISPLAY
0025 0C00 MOVLW B'00000000' ; CLEAR SEG DRIVE
0026 027 MOVWF PORT_C ; / 
0027 0C3F MOVLW B'00111111' ; SEE IF LAST DIGIT
0028 0186 XORWF PORT_B,O ; / 
0029 0643 BTFSC STATUS,Z ; NO THEN SKIP
002A 0A6F GOTO SCAN_KP ; ELSE SCAN KEYPAD

; SELECT DIGIT TO BE DISPLAYED
002B 0246 COMF PORT_B,O ; GET COMPL. PORT B IN W
002C 0643 BTFSC STATUS,Z ; NO DIGIT SELECTED?
002D 0CC0 MOVLW B'11000000' ; THEN SELECT DEFAULT
002E 0031 MOVF TEMP ; SAVE IN TEMP
002F 0271 COMP TEMP ; COMPLEMENT VALUE
0030 0503 BSF STATUS,C ; SET CARRY
0031 0371 RLF TEMP ; SHIFT LEFT
0032 0703 BTFSS STATUS,C ; IF C=1 THEN SKIP
0033 0371 RLF TEMP ; ELSE 3 TIMES...
0034 0371 RLF TEMP ; THRU CARRY
0035 0211 MOVF TEMP,0 ; GET IN W
0036 0026 MOVWF PORT_B ; OUTPUT TO PORT

; NOW THAT DIGIT IS SELECTED, SELECT SEG VALUES FOR THAT DIGIT
0037 0C0A MOVLW MTMR ; LOAD FSR WITH MTMR
0038 0024 MOVWF FSR ; /
0039 0210 MOVF FLAG,0 ; GET FLAG IN W
003A 0E03 ANDLW B'00000011' ; MASK OTHER BITS
003B 0031 MOVF TEMP ; SAVE IN TEMP
003C 0F03 XRORLW B'00000011' ; IN TEST MODE
003D 0643 BTFSC STATUS,Z ; NO THEN SKIP
003E 0A4B GOTO DO_TM ; ELSE TEST MODE
003F 0403 BSF STATUS,C ; CLEAR CARRY
0040 0371 RLF TEMP ; LEFT SHIFT TEMP
0041 0211 MOVF TEMP,0 ; GET IN W
0042 01B4 ADDWF FSR ; CHANGE INDIRECT POINTER
0043 0954 CALL GET_7_SEG ; GET 7 SEG DATA IN W
0044 0032 MOVWF DIGIT ; SAVE IN DIGIT LOC.
0045 09D1 CALL MASK_ANNC ; MASK ANNC TO DIGIT
0046 0690 BTFSC FLAG,FLASH ; NO FLASH THEN SKIP
0047 094E CALL CHK_HALF_SEC ; ELSE CHK. IF ON
0048 0212 MOVF DIGIT,0 ; GET BACK DIGIT
0049 0027 MOVWF PORT_C ; OUTPUT TO PORT
004A 0800 RETLW 0 ; RETURN

; DO_TM
004B 0CFF MOVLW B'11111111' ; LIGHT ALL SEGMENTS
004C 0027 MOVWF PORT_C ; / 
004D 0800 RETLW 0 ; RETURN FROM UPDATE DISPLAY

; CHK_HALF_SEC
004E 0770 BTFSS FLAG,COLON ; IF COLON ON THEN DO
004F 0A51 GOTO BLANK_DSP ; ELSE BLANK DISPLAY
0050 0800 RETLW 0

; BLANK_DSP
0051 0C00 MOVLW B'00000000' ; MAKE PORT C LOW
0052 0032 MOVWF DIGIT
0053 0800 RETLW 0

© 1993 Microchip Technology Incorporated
Multiplexing LEDs/Keypad

;ON ENTRY FSR POINTS TO THE REAL TIME MODE'S MINUTES REGISTER.
;ON RETURN FSR POINTS TO THE TIMER REGISTER TO BE DISPLAYED.
;W REG. CONTAINS THE DECODED 7 SEG. INFO OF THE DIGIT
;TO BE DISPLAYED
;
; GET_7_SEG

;COMPLEMENT B -> W
0054 0246
0055 0EF0
0056 0643
0057 02A4
0058 0200
0059 0031
005A 0643
005B 0643
005C 02A4
005D 0643
005E 0200
005F 0031
0060 0643
0061 02A4
0062 0643
0063 02A4
0064 0643
0065 02A4
0066 0643
0067 02A4
0068 0643
0069 02A4
006A 0643
006B 02A4
006C 0643
006D 02A4
006E 0643
006F 02A4
0070 0643
0071 0643
0072 0643
0073 0643
0074 0643
0075 0643
0076 0643
0077 0643
0078 0643
0079 0643
007A 0643
007B 0643
007C 0643
007D 0643
007E 0643
007F 0643

; This routine scans the 4x4 hex key pad for a key hit.
; If key is pressed, KEY_HIT flag is set and the value of
; the hex key is returned in reg NEW_KEY
; If no key is detected, then a 0xff value is returned in
; register NEW_KEY and the flag KEY_HIT is reset.
;
; SCAN_KP

;KEY UNDER SERVICE?
006F 06D0
0070 0A2B
0071 0CFF
0072 0026
0073 0CFF
0074 0031

;SET PORT C AS OUTPUTS
0075 0C00
0076 0007
0077 0211
0078 0E0F
0079 0027
007A 0CF0
007B 0007
007C 0211
007D 0027
007E 0207
007F 0E0F

; SET PORT C AS INPUTS
0080 0FF0
0081 0743
0082 0A8D

; SET CARRY
0083 0503
0084 0331
0085 0603
0086 0A75
0087 0733
0088 0F03

© 1993 Microchip Technology Incorporated
Multiplexing LEDs/Keypad

```
SKP2

DET_KEY
;key is detected
008D 0293  CLRF  NEW_KEY,W  ;CHK IF KEY ...
008E 0743  INCF  NEW_KEY,W  ;CHK IF KEY ...
008F 0A89  GOTO  DET_KEY  ;NO THEN RETURN
0090 0207  MOVF  PORT_C,W  ;GET RAW KEY...
0091 0D0F  IORLW B'00001111'  ;VALUE.
0092 0151  ANDWF TEMP,W  ;/
0093 0033  MOVWF  NEW_KEY  ;SAVE IN NEW_KEY
0094 0998  CALL  GET_KEY_VAL  ;GET ACTUAL KEY ...
0095 0033  MOVWF  NEW_KEY  ;VALUE
0096 05D0  BSF  NEW_KEY  ;SET KEY HIT FLAG
0097 0A89  GOTO  SKP2  ;RETURN

;This routine decodes the hex value from the "raw" data got
;from scanning the rows and cols.

; actual key value         raw hex value
;    ONE                     EQU 77
;    TWO                     EQU 7B
;    THREE                   EQU 7D
;    C                       EQU 7E
;    FOUR                    EQU 087
;    FIVE                    EQU 08B
;    SIX                     EQU 08D
;    D                       EQU 08E
;    SEVEN                   EQU 0D7
;    EIGHT                   EQU 0DB
;    NINE                    EQU 0DD
;    E                       EQU 0DE
;    A                       EQU 0DE
;    ZERO                    EQU 0DB
;    B                       EQU 0ED
;    F                       EQU 0EE
;

GET_KEY_VAL
0098 060F  ANDLW B'00001111'  ;GET LO NIBBLE
0099 0034  MOVWF  KEYNibL  ;SAVE
009A 0C04  MOVLW  4  ;SET COUNT TO 4
009B 0031  MOVWF  TEMP  ;/

GKV1

BSF STATUS,C  ;SET CARRY
RRF  KEYNibL  ;ROTATE NIBBLE
BTFSS STATUS,C  ;SKIP IF NOT Z
GOTO  GET_HI_KEY  ;GOTO NEXT PART
DECFSZ TEMP  ;DEC COUNT
GOTO  GKV1  ;LOOP

GO_RESET

BSF STATUS,PA0  ;SET MSB
BSF STATUS,PA1  ;/
GOTO  SYS_RESET  ;ELSE BIG ERROR

GET_HI_KEY

DECF TEMP  ;REDUCE BY 1
SWAPF NEW_KEY,W  ;GET HI NIBBLE
ANDLW B'00001111'  ;/
MOVWF  KEYNibL  ;SAVE
MOVF  TEMP,W  ;GET OFFSET TO TBL
ADDWF PC  ;LOAD IN PC
GOTO  GET147A  ;JUMP TO NEXT PART
GOTO  GET369B  ;/
GOTO  GETCDEF  ;/
```

© 1993 Microchip Technology Incorporated
Multiplexing LEDs/Keypad

```assembly
GET147A MOVFL 4 ;SET COUNT TO 4
GETCOM MOVWF TEMP ;
GETCOM1

GETCOM BSF STATUS, C ;SET CARRY
GETCOM RRF KEYNibl ;ROTATE RIGHT
GETCOM BTFS STATUS, C ;CHECK IF DONE
GETCOM GOTO KEY_TBL ;JUMP TO TABLE
GETCOM DECFSZ TEMP ;DEC COUNT
GETCOM GOTO GETCOM1 ;LOOP
GETCOM GO_RESET ;ELSE ERROR

GET2580 MOVFL 8 ;SET COUNT TO 8
GETCOM GOTO GETCOM
GETCOM
GET369B MOVFL D'12' ;SET COUNT TO 12
GETCOM GOTO GETCOM
GETCOM
GETCDEF MOVFL D'16' ;SET COUNT TO 16
GETCOM GOTO GETCOM
GETCOM

KEY_TBL DECF TEMP ;REDUCE BY 1
KEY_TBL MOVE TEMP, W ;GET IN W
KEY_TBL ADDWF PC ;JUMP TO TABLE
KEY_TBL RETLW 1 ;KEY 1
KEY_TBL RETLW 4 ;KEY 4
KEY_TBL RETLW 7 ;KEY 7
KEY_TBL RETLW 0A ;KEY A
KEY_TBL RETLW 2 ;KEY 2
KEY_TBL RETLW 5 ;KEY 5
KEY_TBL RETLW 8 ;KEY 8
KEY_TBL RETLW 0 ;KEY 0
KEY_TBL RETLW 3 ;KEY 3
KEY_TBL RETLW 6 ;KEY 6
KEY_TBL RETLW 9 ;KEY 9
KEY_TBL RETLW 0B ;KEY B
KEY_TBL RETLW 0C ;KEY C
KEY_TBL RETLW 0D ;KEY D
KEY_TBL RETLW 0E ;KEY E
KEY_TBL RETLW 0F ;KEY F

MASK ANNC
MASK ANNC MOVFL B'11111100' ;CHK IF DIGIT 1
MASK ANNC XORWF PORT_B, 0 ; / 
MASK ANNC BTFS STATUS, Z ;NO THEN SKIP
MASK ANNC GOTO MASKAlarm ;ELSE MASK ALARM
MASK ANNC MOVFL B'11110111' ;CHK IF DIGIT 2
MASK ANNC XORWF PORT_B, 0 ; / 
MASK ANNC BTFS STATUS, Z ;NO THEN SKIP
MASK ANNC GOTO MASKCOLON ;ELSE MASK COLON
MASK ANNC MOVFL B'11001111' ;CHK IF DIGIT 3
MASK ANNC XORWF PORT_B, 0 ; / 
MASK ANNC BTFS STATUS, Z ;NO THEN SKIP
MASK ANNC GOTO MASK_PM ;ELSE MASK PM

MASK_AM
MASK_AM INCF FSR ;INC FSR
MASK_AM BTFS F0, AM_PM ;IF 0 THEN AM
MASK_AM BSF DIGIT, 7 ;SET MSB
MASK_AM GOTO BLNK_LEAD_0 ;NEXT

MASK_PM
MASK_PM INCF FSR ;INC FSR
MASK_PM BTFS F0, AM_PM ;IF 1 THEN PM
MASK_PM BSF DIGIT, 7 ;SET MSB
MASK_PM GOTO BLNK_LEAD_0 ;NEXT
```

© 1993 Microchip Technology Incorporated
Multiplexing LEDs/Keypad

MASK_ALARM

00E5 0650 BTFSC FLAG,ALRMLED ;1 THEN LIGHT LED
00E6 05F2 BSF DIGIT,7 ;
00E7 0AEB GOTO BLNK_LEAD_0 ;NEXT

MASK_COLON

00E8 0670 BTFSC FLAG,COLON ;1 THEN LIGHT LED
00E9 05F2 BSF DIGIT,7 ;
00EA 0AEB GOTO BLNK_LEAD_0 ;NEXT

BLNK_LEAD_0

00EB 0210 MOVFS FLAG,W ;GET IN W
00EC OE03 ANDLW B'00000011' ;SEE IF IN DEM
00ED 0F02 XORLW B'00000010' ;CHECK
00EE 0643 BTFSC STATUS,2 ;NO THEN DO
00EF 0800 RETLW 0 ;ELSE RETURN
00F0 0CFE MOVLW B'11111100' ;SEE IF DIGIT 1
00F1 0186 XORWF PORT_8,0
00F2 0743 BTFSS STATUS,2 ;YES THEN SKIP
00F3 0800 RETLW 0 ;RETURN
00F4 0C3F MOVLW B'00111111' ;ELSE MASK G AND ANUNC
00F5 0152 ANDWF DIGIT,0 ;GET IN W
00F6 033F XORLW B'00111111' ;SEE IF 0
00F7 0743 BTFSS STATUS,2 ;YES THEN SKIP
00F8 0800 RETLW 0 ;RETURN
00F9 OC80 MOVLW B'10000000' ;ELSE BLANK D1
00FA 0172 ANDWF DIGIT ;
00FB 0800 RETLW 0 ;RETURN

;THIS ROUTINE SETS UP PORTS A,B,C AND THE INTERNAL
;REAL TIME CLOCK COUNTER.

INIT_CLK

00FC 0C0E MOVLW B'00001111' ;MAKE ACTIVE HIGH
00FD 0025 MOVWF PORT_A ;
00FE OC00 MOVLW B'00000000' ;SET PORT A AS OUTPUTS
00FF 0005 TRIS PORT_A

0100 OCFF MOVLW B'11111111' ;SET LEVELS HIGH
0101 0026 MOVWF PORT_B ;
0102 OC00 MOVLW B'00000000' ;SET PORT B AS OUTPUTS
0103 0006 TRIS PORT_B

0104 OC00 MOVLW B'00000000' ;SET LEVELS LOW
0105 0027 MOVWF PORT_C ;
0106 OC00 MOVLW B'00000000' ;SET PORT C AS OUTPUTS
0107 0007 TRIS PORT_C ;

0108 OC04 MOVLW B'00000100' ;SET UP PRESCALER
0109 0002 OPTION ;

010A OC60 MOVLW MSEC5 ;RTCC = 5 mSEC
010B 0021 MOVWF RTCC ;
010C 0068 CLRWF MSTMR ;CLEAR MSTMR
010D 0069 CLRWF STMR ;& SEC TMR
010E 006A CLRWF MTMR ;& MINUTES
010F OC12 MOVLW 12H ;MAKE HRS = 12
0110 002B MOVWF HTMR ;
0111 002D MOVWF HALARM ;MAKE HRS = 12
0112 006C CLRWF MALARM ;
0113 OC03 MOVWF B'00000001';SET TO TEST MODE
0114 0030 MOVWF FLAG ;
0115 007A CLRWF ALFLAG ;CLEAR ALL FLAG
0116 0079 CLRWF AAFLAG ;
0117 0077 CLRWF ENTFLG ;
0118 0A01 GOTO TEST_HARDWARE
Multiplexing LEDs/Keypad

; All routines related to timer updates are located at
; address 200 and above.
ORG 0200

UPDATE_TIMERS

;RTCC, W
MOVF RTCC,W
BTFSS STATUS,Z
; IF 0 THEN SKIP
GOTO UPDATE_TIMERS
; ELSE LOOP

;RTCC = 5 mSEC
MOVWF RTCC
;

; INC 5 MILLI SEC
INCF MSTMR

; NO KEY HIT THEN SKIP
BTFSF FLAG, KEY_HIT

; CHK DE BOUNCE
GOTO CHK_DE_BOUNCE
; ELSE DEBOUNCE

UP_TMR_1

; ALARM MODE?
MOVF FLAG, W

; SKIP IF YES
ANDLW B'00000001'
;

; SKIP IF YES
XORLW B'00000001'
;

; SKIP IF YES
BTFSF STATUS, Z
;

; SKIP IF YES
GOTO UP_TMR_2
; ELSE NEXT

BSF FLAG, ALRMLED

; LIGHT LED
BSF FLAG, COLON
;

; IF 1/2 SEC
MOVlw D'100'

; SKIP IF YES
SUBWF MSTMR,0

; BLINK
BTFSF STATUS, C
;

; ALARM LED
BCF FLAG, ALRMLED

; ELSE TURN OFF
GOTO UP_TMR_3

UP_TMR_2

; TURN ON
BSF FLAG, COLON

; <100 BLINK COLON
MOVlw D'100'

; SKIP IF YES
SUBWF MSTMR, 0
;

; SKIP IF YES
BTFSF STATUS, C
;

; ELSE TURN OFF
BCF FLAG, COLON

UP_TMR_3

; GET MSTMR IN W
MOVFW MSTMR, W

; 199 THEN SKIP
B'199'

; CLEAR MS_TMR
CLRF MSTMR

; GET MIN_SEC TIMER
MOVFW MIN_SEC, W

; MASK MINUTES
ANDlw B'00000111'

; ZERO THEN SKIP
BTFSF STATUS, Z
;

; REDUCE SECONDS
DECF MIN_SEC

; LOAD FSR WITH S_TMR
MOVFW STMR
;

; INC SECONDS COUNT
INC 60

; GET MIN_SEC IN W
MOVFW MIN_SEC, W

; GET MIN_SEC IN W
ANDlw B'00000111'

; SKIP IF NOT SET
BTFSF STATUS, Z
;

; ELSE DEC
DECF MIN_SEC

; SWAP BACK
SWAPF MIN_SEC

; CALL CHK_SILNC_TMR
CALL CHK_SILNC_TMR

; SIMILE ON?
MOVlw MTMR

; INC MINUTES
INC 60

; DO AN OPERATION
IOWR 0

; BIT OFF
BTFSF STATUS, Z

; IF RETURN = 0 SKIP
GOTO CHK_AL_TIM

; INC MINUTES COUNT

; SWAP MIN SEC
SWAPF MIN_SEC

; GET MIN_SEC IN W
MOVFW MIN_SEC, W

; MASK SECONDS
ANDlw B'00000111'

; SKIP IF NOT SET
BTFSF STATUS, Z
;

; ELSE DEC
DECF MIN_SEC

; SWAP BACK
SWAPF MIN_SEC

; CALL CHK_SILNC_TMR
CALL CHK_SILNC_TMR

; SIMILE ON?
MOVlw MTMR

; INC MINUTES
INC 60

; DO AN OPERATION
IOWR 0

; BIT ON
BTFSF STATUS, Z

; IF 0 THEN SKIP
GOTO CHK_AL_TIM

; CHECK ALARM TIME

; INC HOUR COUNT

; GET HTMR IN FSR
MOVFW HTMR

; GET HTMR IN FSR
MOVFW HTMR

; CALL INC_HR
CALL INC_HR

; INC HOURS

CHK_AL_TIM

0238 071A BTFSS ALFLAG, ALONOF ; IF OFF QUIT
0239 0800 RETLW 0 ; / 
023A 0658 BTFSC ALFLAG, SILNC ; RET IF IN SILENCE
023B 0800 RETLW 0
023C 0638 BTFSC ALFLAG, INAL ; ALREADY DONE
023D 0A4D GOTO CHK_1_MIN ; SEE IF 1 MIN UP
023E 020D MOVF HALARM, W ; CHK HRS
023F 01BB XORWF HTMR, W ; EQUAL?
0240 0743 BTFSS STATUS, Z ; YES THEN SKIP
0241 0800 RETLW 0 ; ELSE RET
0242 020C MOVF MLARM, W ; CHK MIN
0243 0743 BTFSS STATUS, Z ; YES THEN SKIP
0244 0743 RETLW 0 ; ELSE RET
0245 0209 MOVF TMR, W ; SEE IF SEC=0
0246 0743 BTFSS STATUS, Z ; YES THEN SKIP
0247 0743 RETLW 0 ; ELSE RET
0248 0538 BSF ALFLAG, INAL ; SET IN ALARM FLAG
0249 0C10 MOVLW 10 ; SET 1 MIN TIMER
024A 0036 MOVWF MIN_SEC ; /
024B 0800 RETLW 0

; CHK_1_MIN

024C 0396 SWAPF MIN_SEC, W ; SWAP IN W
024E 050F ANDLW B'00001111'; ; CHK MINUTES
024F 0743 BTFSS STATUS, Z ; 0 THEN SKIP
0250 0800 RETLW 0 ; ELSE RET
0251 0438 BCF ALFLAG, INAL ; CLR IN ALARM
0252 0478 BCF ALFLAG, INAA ; CLR IN AA
0253 0505 BSF PORT_A, BEEP ; STOP BEEPER
0254 0800 RETLW 0

; INC_60

0255 02A0 INCF F0 ; INC AND GET IN W
0256 0200 MOVF F0, 0 ; /
0257 050F ANDLW B'00000000'; ; MASK HI BITS
0258 0F0A XORLW B'00000101'; ; = 6 THEN MAKE IT 0
0259 0743 BTFSS STATUS, Z ; /
025A 0801 RETLW 1 ; ELSE RETURN NON ZERO
025B 0CF0 MOVLW B'10100000'; ; ZERO LSB
025C 0160 ANDWF F0 ; /
025D 03A0 SWAPF F0 ; SWAP INDIRECT
025E 02A0 INCF F0 ; INC
025F 0200 MOVF F0, 0 ; GET IN W
0260 03A0 SWAPF F0 ; SWAP F0 BACK
0261 0F06 XORLW D'6'; ; = 6 THEN SKIP
0262 0743 BTFSS STATUS, Z ; /
0263 0801 RETLW 1 ; ELSE RETURN NZ
0264 0060 CLRF F0 ; /
0265 0800 RETLW 0 ; RET 0

; CHK_SILNC_TIM

0266 0758 BTFSS ALFLAG, SILNC ; CHK IF IN SILENCE
0267 0800 RETLW 0 ; NO THEN SKIP
0268 0396 SWAPF MIN_SEC, W ; GET MIN IN W
0269 050F ANDLW B'00000000'; ; MASK SECS
026A 0743 BTFSS STATUS, Z ; ZERO?
026B 0800 RETLW 0 ; NO THEN RET
026C 0458 BCF ALFLAG, SILNC ; RESET SILENCE
026D 0C10 MOVLW 10 ; SET 1 MIN TIMER
026E 0036 MOVWF MIN_SEC ; /
026F 0800 RETLW 0 ;
; CHK_DE_BOUNCE
0270 06B7  BTFSC ENTFLG, INKEYBEP ; IN KEY BEEP?
0271 0A76  GOTO CHK_DEB_1 ; YES THEN DEC TIMER
0272 07B0  BTFSS FLAG, KEY_BEEP ; KEY BEEP SET?
0273 0A7F  GOTO CHK_SERV ; NO, SEE IF SERVICED
0274 0678  BTFSC ALFLAG, INAA ; IN AA?
0275 0A86  GOTO CHK_BEP_ON ; YES THEN SEE IF ON

CHK_DEB_1
0276 05B7  BSF ENTFLG, INKEYBEP ; SET FLAG
0277 0215  MOVF DEBOUNCE, W ; GET IN W
0278 0643  BTFSC STATUS, Z ; NZ THEN SKIP
0279 0C34  MOVFL D'20' ; ELSE DB 100 mSEC
027A 0035  MOVWF DEBOUNCE ; /
027B 0405  BCF PORT_A, BEP ; TURN ON BEEPER
027C 02F5  DECFSZ DEBOUNCE ; DEC AND CHK
027D 0A08  GOTO UP_TMR_1 ; GO BACK
027E 0505  BSF PORT_A, BEP ; TURN OFF BEEPER

CHK_SERV
; CLRF DEBOUNCE
; BSF PORT_A, BEP
027F 07F0  BTFSS FLAG, SERVICED ; SERVICED THEN SKIP
0280 0A08  GOTO UP_TMR_1 ; GO BACK
0281 04F0  BCF FLAG, SERVICED ; ELSE CLEAR FLAGS
0282 0400  BCF KEY_HIT ; /
0283 04B0  BCF KEY_BEEP ; RESET FLAG
0284 04B7  BCF ENTFLG, INKEYBEP ; /
0285 0A08  GOTO UP_TMR_1 ; GO BACK

CHK_BEP_ON
0286 0705  BTFSS PORT_A, BEP ; IF OFF THEN SKIP
0287 0A08  GOTO UP_TMR_1 ; ELSE WAIT
0288 0A76  GOTO CHK_DEB_1 ; RETURN

; INC_HR
0289 02A0  INCF F0 ; INC HOUR TIMER
028A 0200  MOVF F0, W ; GET HR TMR IN W
028B 0031  MOVWF TEMP ; SAVE IN TEMP
028C 050F  ANDLW B'00000111' ; CHK LO BYTE = 10
028D 0F0A  XORLW D'10' ; /
028E 0743  BTFSS STATUS, Z ; YES THEN SKIP
028F 0A93  GOTO INC_AM_PM ; ELSE CHK 12
0290 0C10  MOVFL W'00010000' ; LOAD 1 IN MSB
0291 0020  MOVWF F0
0292 0AA3  GOTO RESTORE_AM_PM ; RESTORE AM/PM

INC_AM_PM
0293 04E0  BCF F0, AM_PM ; CLEAR AM/PM
0294 0200  MOVF F0, W ; GET IN W
0295 0F12  XORLW 12H ; SEE IF 12 HEX
0296 0743  BTFSS STATUS, Z ; YES THEN SKIP
0297 0A9D  GOTO CHK_13 ; ELSE CHK 13
0298 07F1  BTFSS TEMP, AM_PM ; IF SET, SKIP
0299 0A9C  GOTO SET_AM_PM ; ELSE SET
029A 04E0  BCF F0, AM_PM ; CLEAR FLAG
029B 0800  RETLW 0 ; RETURN

SET_AM_PM
029C 05E0  BSF F0, AM_PM ; SET FLAG
029D 0200  MOVF F0, W ; GET IN W
029E 0F13  XORLW 13H ; SEE IF 13
029F 0743  BTFSS STATUS, Z ; YES THEN SKIP
02A0 0AA3  GOTO RESTORE_AM_PM

SET_1_HR
02A1 0C01  MOVFL W'00000001' ; SET TO 1
02A2 0202  MOVWF F0
02A3 06F1  BTFSC TEMP, AM_PM ; SKIP IF AM
02A4 05E0  BSF F0, AM_PM ; ELSE SET TO PM
02A5 0800  RETLW 0 ;
; ORG 400
; KEY DEFINITIONS
000A ALARM_KEY EQU 0A
000B CE_KEY EQU 0B
000C SNOOZE_KEY EQU 0C
000D AM_PM_KEY EQU 0D
000E CLR_ALARM_KEY EQU 0E
000F SET_KEY EQU 0F

; SERVICE KEYS
0400 07D0 BTFSS FLAG,KEY_HIT ;NO KEY HIT THEN ...
0401 0800 RETLW 0 ;RETURN
0402 06F0 BTFSC FLAG,SERVICED ;IF NOT SERVICED SKIP
0403 0800 RETLW 0 ;ELSE RETURN
0404 05F0 BSF FLAG,SERVICED ;SET SERVICED FLAG
0405 0210 MOVF FLAG,W ;GET MODE OF OPERATION
0406 0E03 ANDLW B'00000011' ;
0407 0643 BTFSC STATUS,Z ;00 THEN RTM
0408 0A10 GOTO RTMKS ;RTM KEY SERVICE
0409 0031 MOVWF TEMP ;SAVE IN TEMP
040A 02F1 DECFSZ TEMP ;REDUCE TEMP
040B 0A0D GOTO SK1 ;SKIP
040C 0A1D GOTO ATMSKS ;01, DO ALARM MODE

SK1
040D 02F1 DECFSZ TEMP ;REDUCE TEMP
040E 0800 RETLW 0 ;11 THEN RETURN
040F 0A2A GOTO DEMKS ;10, DATA ENTRY MODE

; REAL TIME MODE KEY SERVICE
RTMKS
0410 09BA CALL CHK_AL_KEYS ;CHK ALARM KEYS
0411 0DD0 IORLW 0 ;SEC IF NZ RET
0412 0643 BTFSC STATUS,Z ;N2 THEN SKIP
0413 0800 RETLW 0 ;ELSE RETURN
0414 0C0F MOVFWSET_KEY ;SEE IF SET KEY
0415 0193 XORWF NEW_KEY,W ;
0416 0643 BTFSC STATUS,Z ;NO THEN SKIP
0417 0A91 GOTO SERV_SET_RTM ;SERVICE SET KEY
0418 0C0A MOVFW ALARM_KEY ;ALARM KEY?
0419 0193 XORWF NEW_KEY,W ;
041A 0643 BTFSC STATUS,Z ;NO THEN SKIP
041B 0AAB GOTO SERV_ALARM_RTM ;ELSE SERVICE ALARM

IGNORE_KEY
041C 0800 RETLW 0 ;ELSE RETURN

; ALARM TIME MODE KEY SERVICE
ATMKS
041D 09BA CALL CHK_AL_KEYS ;CHECK ALARM KEYS
041E 0D00 IORLW 0 ;CHECK IF 0
041F 0643 BTFSC STATUS,Z ;N2 THEN SKIP
0420 0800 RETLW 0 ;ELSE RETURN
0421 0C0F MOVFWSET_KEY ;SEE IF SET KEY
0422 0193 XORWF NEW_KEY,W ;
0423 0643 BTFSC STATUS,Z ;NO THEN SKIP
0424 0A9C GOTO SERV_SET_ATM
0425 0C0A MOVFW ALARM_KEY ;GET ALARM KEY
0426 0193 XORWF NEW_KEY,W ;SEE IF HIT
0427 0643 BTFSC STATUS,Z ;NO THEN SKIP
0428 0A2A GOTO SERV_ALARM_ATM ;ELSE SERVICE
0429 0A1C GOTO IGNORE_KEY
; DATA ENTRY MODE KEY SERVICE
DEMSK

042A 09BA CALL CHK_AL_KEYS ; CHECK ALARM KEYS
DEMSK
042B 0000 IORLW 0 ; CHK IF 0
042C 0643 BTFS C STATUS,2 ; NZ THEN SKIP
DEMSK
042D 0800 RETLW 0 ; ELSE RETURN
042E 0C0F MOVL W SET_KEY ; IF SET KEY THEN END
DEMSK
042F 0193 XORWF NEW_KEY,W ; /
0430 0643 BTFS C STATUS,2 ; NO THEN SKIP
0431 0A3F GOTO DEMKS_END ; GOTO END
0432 0C0B MOVL W CE_KEY ; IF CLEAR ENTRY
DEMSK
0433 0193 XORWF NEW_KEY,W ; /
DEMSK
0434 0643 BTFS C STATUS,2 ; SKI IP IF NO
DEMSK
0435 0A48 GOTO DEMKS_END_1 ; ABANDON ENTRY
DEMSK
0436 0737 BTFS SS ENTFLAG,HR10 ; 10's HRS DONE?
DEMSK
0437 0A54 GOTO ENT HR_10 ; NO THEN GET
DEMSK
0438 0757 BTFS SS ENTFLAG,HR ; HRS DONE?
DEMSK
0439 0A5F GOTO ENT HRS ; NO THEN GET
DEMSK
043A 0777 BTFS SS ENTFLAG,MIN10 ; 10's MIN. DONE?
DEMSK
043B 0A72 GOTO ENT_MIN_10 ; NO THEN GET
DEMSK
043C 0797 BTFS SS ENTFLAG,MIN ; MIN DONE?
DEMSK
043D 0A7F GOTO ENT_MIN ; NO THEN GET
DEMSK
043E 0A87 GOTO ENT_AM_PM ; NO THEN GET
DEMSK
043F 0717 BTFS SS ENTFLAG,RTAT S ; GET OLD STATUS
DEMSK
0440 0A4D GOTO LD_RTM ; LOAD IN TIME
DEMSK
0441 020E MOVF MENTRY,W ; LD IN ALARM
DEMSK
0442 002C MOVWF MALAR M ; /
DEMSK
0443 020F MOVF HENTRY,W ; /
DEMSK
0444 002D MOVWF HALAR M ; /
DEMSK
0445 0450 BCF FLAG,ALARML ED ; CLEAR FLAG
DEMSK
0446 0618 BTFS C ALFL AG,ALONOFF ; SEE IF ON-OFF
DEMSK
0447 0550 B SF FLAG,ALARML ED ; ELSE SET
DEMSK
0448 0410 BCF FLAG,0 ; RTM MODE
DEMSK
0449 0430 BCF FLAG,1 ; /
DEMSK
044A 0490 BCF FLAG,FLASH ; STOP FLASH
DEMSK
044B 0590 BSF FLAG,KEY_BEEP ; RETURN
DEMSK
044C 0800 RETLW 0 ; RETURN
DEMSK

; LD_RTM
DEMSK
044D 020E MOVF MENTRY,W ; LD IN RTM
DEMSK
044E 002A MOVWF MTMR ; /
DEMSK
044F 020F MOVF HENTRY,W ; /
DEMSK
0450 002B MOVWF HTMR ; /
DEMSK
0451 006B CLRF MSTMR ; CLR TIME
DEMSK
0452 0063 CLRF SMTR ; /
DEMSK
0453 0A48 GOTO DEMKS_END_1 ; GO BACK
DEMSK

; ENT HR_10
DEMSK
0454 0213 MOVF NEW_KEY,W ; SEE IF 0
DEMSK
0455 0643 BTFS C STATUS,2 ; NZ THEN SKIP
DEMSK
0456 0A5C GOTO LD_HENTRY_0 ; LOAD 0
DEMSK
0457 02D3 DECF SZ NEW_KEY,0 ; 1 THE SKIP
DEMSK
0458 0A1C GOTO IGNORE_KEY ; ELSE IGNORE KEY
DEMSK
0459 058F BCF HENTRY,4 ; SET TO 1
DEMSK
045A 0537 BSF ENTFLAG,HR10 ; SET FLAG
DEMSK
045B 0A4B GOTO SERV_COM_RET ; GO GET NEXT
DEMSK

LD_HENTRY_0
DEMSK
045C 048F BCF HENTRY,4 ; SET TO 0
DEMSK
045D 0537 BSF ENTFLAG,HR10 ;
DEMSK
045E 0A4B GOTO SERV_COM_RET ;

2-115
Multiplexing LEDs/Keypad

ENT_HRS        MOVFLW HENTRY ;USE INDIRECT ADDR.
045F 0C0F
0460 0024
0461 068F
0462 0A6D
0463 0C0A
0464 0093
0465 0603
0466 0A1C

ENT_LO_COM1
0467 0557

ENT_LO_COM
0468 0200
0469 0F4F
046A 0113
046B 0020
046C 0A4B

ALLOW_2
046D 0C03
046E 0093
046F 0603
0470 0A1C
0471 0A67

ENT_MIN_10
0472 0C0E
0473 0024
0474 0C06
0475 0093
0476 0603
0477 0A1C
0478 0380
0479 0F4F
047A 0113
047B 0020
047C 03A0
047D 0577
047E 0A4B

ENT_MIN
047F 0C0E
0480 0024
0481 0C0A
0482 0093
0483 0603
0484 0A1C
0485 0597
0486 0A68

ENT_AM_PM
0487 0C0D
0488 0193
0489 0743
048A 0A1C
048B 07EF
048C 0A8F
048D 04EF
048E 0A4B

SETAMPM
048F 05EF
0490 0A4B

SERV_SET_RTM
0491 020A
0492 002E
0493 020B
0494 002F

© 1993 Microchip Technology Incorporated
### Multiplexing LEDs/Keypad

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0495 0210</td>
<td>MOVF Flag, W</td>
<td>SAVE IN W</td>
</tr>
<tr>
<td>0496 0001</td>
<td>ANDLW $'00000001'</td>
<td>ATM OR RTM MODE?</td>
</tr>
<tr>
<td>0497 0037</td>
<td>MOVWF ENTFLG</td>
<td>SAVE IN ENTFLG</td>
</tr>
<tr>
<td>0498 0CF2</td>
<td>MOVLW $'11110010'</td>
<td>FORCE IS</td>
</tr>
<tr>
<td>0499 0130</td>
<td>TOCRWF FLAG</td>
<td>/</td>
</tr>
<tr>
<td>049A 0410</td>
<td>BCF FLAG, 0</td>
<td>/</td>
</tr>
<tr>
<td>049B 0800</td>
<td>RETLW 0</td>
<td>;</td>
</tr>
</tbody>
</table>

### SERV_COM

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>049C 020C</td>
<td>MOVF MALARM, W</td>
<td>TRANSFER ALARM</td>
</tr>
<tr>
<td>049D 002E</td>
<td>MOVWF MENTRY</td>
<td>TO DATA ENTRY</td>
</tr>
<tr>
<td>049E 020D</td>
<td>MOVWF MALARM, W</td>
<td>/</td>
</tr>
<tr>
<td>049F 020F</td>
<td>MOVWF HENTRY</td>
<td>/</td>
</tr>
<tr>
<td>04A0 0518</td>
<td>BSF ALFLG, ALONOF</td>
<td>SET FLAG</td>
</tr>
<tr>
<td>04A1 0A95</td>
<td>GOTO SERV_COM</td>
<td>/</td>
</tr>
</tbody>
</table>

### SERV_ALARM_ATM

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04A2 0718</td>
<td>BTFS ALFLG, ALONOF</td>
<td>TEST ON/OFF</td>
</tr>
<tr>
<td>04A3 0A6</td>
<td>GOTO SET_ALONOF</td>
<td>SET ON/OFF FLG</td>
</tr>
<tr>
<td>04A4 0181</td>
<td>BCF ALFLG, ALONOF</td>
<td>CLEAR FLAG</td>
</tr>
<tr>
<td>04A5 0AA7</td>
<td>GOTO SERV_ATM_COMM</td>
<td>SET FLAG</td>
</tr>
<tr>
<td>04A6 0518</td>
<td>BSF ALFLG, ALONOF</td>
<td>SET FLAG</td>
</tr>
</tbody>
</table>

### SERV_ATM_COMM

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04A7 05BO</td>
<td>BSF FLAG, KEY_BEEP</td>
<td>BEEP</td>
</tr>
<tr>
<td>04A8 0CF0</td>
<td>MOVLW $'11110000'</td>
<td>CLEAR SEC COUNT</td>
</tr>
<tr>
<td>04A9 0176</td>
<td>ANDWF MIN_SEC</td>
<td>/</td>
</tr>
<tr>
<td>04AA 0800</td>
<td>RETLW 0</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

### SERV_ALARM_RTM

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04AB 05BO</td>
<td>BSF FLAG, KEY_BEEP</td>
<td>SET BEEP FLAG</td>
</tr>
<tr>
<td>04AC 0510</td>
<td>BSF FLAG, 0</td>
<td>SET TO ALARM TIME</td>
</tr>
<tr>
<td>04AD 0430</td>
<td>BCF FLAG, 1</td>
<td>/</td>
</tr>
<tr>
<td>04AE CO5</td>
<td>MOVLW D'05'</td>
<td>SAVE 5 IN MIN_SEC</td>
</tr>
<tr>
<td>04AF 0036</td>
<td>MOVWF MIN_SEC</td>
<td>/</td>
</tr>
<tr>
<td>04B0 0800</td>
<td>RETLW 0</td>
<td>;</td>
</tr>
</tbody>
</table>

### SERV_SNOOZE

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04B1 0CA0</td>
<td>MOVLW OA0</td>
<td>SNOOZE FOR 10 MINS</td>
</tr>
<tr>
<td>04B2 0036</td>
<td>MOVWF MIN_SEC</td>
<td>/</td>
</tr>
<tr>
<td>04B3 0558</td>
<td>BSF ALFLG, SILNC</td>
<td>SET FLAG</td>
</tr>
</tbody>
</table>

### CLR_AL_COMM

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04B4 05BO</td>
<td>BSF FLAG, KEY_BEEP</td>
<td>SET BEEP FLAG</td>
</tr>
<tr>
<td>04B5 007A</td>
<td>CLR AL AATMR</td>
<td>RESET AA TIMER</td>
</tr>
<tr>
<td>04B6 0079</td>
<td>CLR AL AALF</td>
<td>CLEAR AA FLAGS</td>
</tr>
<tr>
<td>04B7 0478</td>
<td>BCF ALFLG, INAA</td>
<td>RESET INAA FLAG</td>
</tr>
<tr>
<td>04B8 0505</td>
<td>BSF PORT_A, BEEP</td>
<td>TURN OFF BEEPER</td>
</tr>
<tr>
<td>04B9 0800</td>
<td>RETLW 0</td>
<td>;</td>
</tr>
</tbody>
</table>

### CHK_ALKEYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04BA 0718</td>
<td>BTFS ALFLG, ALONOF</td>
<td>ALARM ON?</td>
</tr>
<tr>
<td>04BB 0801</td>
<td>RETLW 1</td>
<td>NO THEN RET</td>
</tr>
<tr>
<td>04BC 0738</td>
<td>BTFS ALFLG, INAL</td>
<td>IN ALARM?</td>
</tr>
<tr>
<td>04BD 0801</td>
<td>RETLW 1</td>
<td>NO THEN SKIP</td>
</tr>
<tr>
<td>04BE 005E</td>
<td>MOVLW CLR_ALARM_KEY</td>
<td>CHECK IF CLR_ALARM</td>
</tr>
<tr>
<td>04BF 0193</td>
<td>XORWF NEW_KEY, W</td>
<td>/</td>
</tr>
<tr>
<td>04C0 0643</td>
<td>BTFS STATUS, Z</td>
<td>NO THEN SKIP</td>
</tr>
<tr>
<td>04C1 0AC7</td>
<td>GOTO CLR_ALARM</td>
<td>ELSE CLEAR ALARM</td>
</tr>
<tr>
<td>04C2 00C8</td>
<td>MOVLW SNOOZE_KEY</td>
<td>SEE IF SNOOZE HIT</td>
</tr>
<tr>
<td>04C3 0193</td>
<td>XORWF NEW_KEY, W</td>
<td>/</td>
</tr>
<tr>
<td>04C4 0743</td>
<td>BTFS STATUS, Z</td>
<td>YES THEN SKIP</td>
</tr>
<tr>
<td>04C5 0801</td>
<td>RETLW 1</td>
<td></td>
</tr>
<tr>
<td>04C6 0AB1</td>
<td>GOTO SERV_SNOOZE</td>
<td></td>
</tr>
</tbody>
</table>
Multiplexing LEDs/Keypad

; CLR_ALARM
04C7 0438 BCF ALFLAG, INAL ; CLEAR ALARM
04C8 0458 BCF ALFLAG, SILNC ; CLEAR SILENCE
04C9 0C0F MOVILW B'00001111' ; CLEAR MINS
04CA 0176 ANDWF MIN_SEC ; /
04CB OAB4 GOTO CLR_AL_COM

; ORG 600
; If the AA alarm is set, then this routine takes care of
; the timing in sounding the alarm.

SOUND_AA
0600 0738 BTFSS ALFLAG, INAL ; SKIP IF IN ALRM
0601 0800 RETLW 0 ; ELSE RETURN
0602 0658 BTFSC ALFLAG, SILNC ; SKIP IF NOT IN SIL
0603 0800 RETLW 0 ; ELSE RET
0604 06B7 BTFSC ENTFLG, INKEYBEP ; SKIP IF NOT IN KEY BEP
0605 OA55 GOTO CHK_COLL; ; CHK COLLISION

SND_AA_0
0606 0778 BTFSS ALFLAG, INAA ; SKIP IF IN AA

SND_AA_1
0607 0919 CALL INIT_AA ; INIT ALL
0608 0719 BTFSS AAFLAG, 0 ; SKIP IF DONE
0609 0A21 GOTO DO_CYCLO ; DO FIRST CYCL
060A 0739 BTFSS AAFLAG, 1 ; SKIP IF DONE
060B 0A29 GOTO DO_CYCLO1 ; ELSE 2ND CYCLE
060C 0759 BTFSS AAFLAG, 2 ; SKIP IF DONE
060D OA31 GOTO DO_CYCLO2 ; ELSE DO 3RD CYCLE
060E 0779 BTFSS AAFLAG, 3 ; SKIP IF DONE
060F 0A39 GOTO DO_CYCLO3 ; DO CYCLE 4
0610 0799 BTFSS AAFLAG, 4 ; SKIP IF DONE
0611 0A3E GOTO DO_CYCLO4 ; DO CYCLE 5
0612 07B9 BTFSS AAFLAG, 5 ; SKIP IF DONE
0613 0A43 GOTO DO_CYCLO5 ; DO CYCLE 6
0614 07D9 BTFSS AAFLAG, 6 ; SKIP IF DONE
0615 0A4B GOTO DO_CYCLO6 ; DO CYCLE 7
0616 07F9 BTFSS AAFLAG, 7 ; SKIP IF DONE
0617 0A50 GOTO DO_CYCLO7 ; DO CYCLE 7
0618 0A07 GOTO SND_AA_1 ; GO BACK

; INIT_AA
0619 0079 CLRF AAFLAG ; CLEAR ALL FLAGS
061A 0578 BSF ALFLAG, INAA ; SET IN AA FLAG
061B 0A2D GOTO PUT_ON_100 ; ON 100 MSEC

; DEC_AA_TMR
061C 00FA DECF AATMR ; REDUCE TIMER
061D 021A MOVF AATMR, W ; GET IN W
061E 0743 BTFSS STATUS, 2 ; CHECK IF Z
061F 0801 RETLW 1 ; NO THEN NZ
0620 0800 RETLW 0 ; ELSE 0

; DO_CYCLO
0621 091C CALL DEC_AA_TMR ; REDUCE TIMER
0622 0743 BTFSS STATUS, 2 ; IF NZ THEN RET
0623 0800 RETLW 0
0624 0519 BSF AAFLAG, 0 ; SET DONE FLAG

PUT_OFF_100
0625 0505 BSF PORT_A, BEP ; TURN OFF BEEPER
0626 0C14 MOVILW D'20' ; FOR 100 MSEC
0627 003A MOVWF AATMR ; /
0628 0800 RETLW 0
; DO_CYCL1
0629 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
062A 0743  BTFSS STATUS,Z ;IF NZ THEN RET
062B 0800  RETLW 0
062C 0539  BSF AAFLAG,1 ;SET DONE FLAG
062D 0405  BCF PORT_A,BEP ;TURN OFF BEEPER
062E 0C14  MOVWF D'20' ;FOR 100 MSEC
062F 003A  MOVWF AATMR
0630 0800  RETLW 0

; DO_CYCL2
0631 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
0632 0743  BTFSS STATUS,Z ;IF NZ THEN RET
0633 0800  RETLW 0
0634 0559  BSF AAFLAG,2 ;SET DONE FLAG
0635 0505  BSF PORT_A,BEP ;TURN OFF BEEPER
0636 0C64  MOVWF D'100' ;FOR 500 MSEC
0637 003A  MOVWF AATMR
0638 0800  RETLW 0

; DO_CYCL3
0639 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
063A 0743  BTFSS STATUS,Z ;IF NZ THEN RET
063B 0800  RETLW 0
063C 0579  BSF AAFLAG,3 ;SET DONE FLAG
063D 0A2D  GOTO PUT_ON_100 ;DO NEXT CYCLE

; DO_CYCL4
063E 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
063F 0743  BTFSS STATUS,Z ;IF NZ THEN RET
0640 0800  RETLW 0
0641 0599  BSF AAFLAG,4 ;SET DONE FLAG
0642 0A25  GOTO PUT_OFF_100 ;DO NEXT CYCLE

; DO_CYCL5
0643 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
0644 0743  BTFSS STATUS,Z ;IF NZ THEN RET
0645 0800  RETLW 0
0646 05B9  BSF AAFLAG,5 ;SET DONE FLAG
0647 0A2D  GOTO PUT_ON_100 ;DO NEXT CYCLE

; DO_CYCL6
0648 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
0649 0743  BTFSS STATUS,Z ;IF NZ THEN RET
064A 0800  RETLW 0
064B 05D9  BSF AAFLAG,6 ;SET DONE FLAG
064C 0505  BSF PORT_A,BEP ;TURN OFF BEEPER
064D 0CC8  MOVWF D'200' ;FOR 1000 MSEC
064E 003A  MOVWF AATMR
064F 0800  RETLW 0

; DO_CYCL7
0650 091C  CALL  DEC_AA_TMR ;REDUCE TIMER
0651 0743  BTFSS STATUS,Z ;IF NZ THEN RET
0652 0800  RETLW 0
0653 05F9  BSF AAFLAG,7 ;SET DONE FLAG
0654 0A2D  GOTO PUT_ON_100 ;DO NEXT CYCLE
Multiplexing LEDs/Keypad

; CHK_COLS

0655 0605  BTFSC PORT_A,BEP ;IF ON THEN SKIP
0656 0A06  GOTO SND_AA_0 ;ELSE RET
0657 021A  MOVF AATMR,W ;GET TIMER
0658 0643  BTFSC STATUS,Z ;NZ THEN SKIP
0659 0A5C  GOTO LD_AAT_1 ;LOAD A 1 IN TMR
065A 00FA  DECF AATMR ;REDUCE TIMER
065B 0800  RETLW 0 ;RETURN

LD_AAT_1

065C 02BA  INCF AATMR ;INC TIMER
065D 0800  RETLW 0 ;RET

ORG PIC57

SYS_RESET

07FF 0A00  GOTO START

END

Errors : 0
Warnings : 0
INTRODUCTION

The mouse is becoming increasingly popular as a standard pointing data entry device. It is no doubt that the demand of the mouse is increasing. Various kinds of mice can be found in the market, including optical mouse, opto-mechanical mouse, and its close relative, trackball. The mouse interfaces to the host via an RS232 port or a dedicated interface card. Their mechanisms are very similar. The major electrical components of a mouse are:

- Microcontroller
- Photo-transistors
- Infrared emitting diode
- Voltage conversion circuit

The intelligence of the mouse is provided by the microcontroller, hence the features and performance of a mouse is greatly related to the microcontroller used. This application note describes the implementation of a serial mouse using the PIC16C54. The PIC16C54 is a high speed 8-bit CMOS microcontroller offered by Microchip Technology, Inc. It is an ideal candidate for a mouse controller.

THEORY OF OPERATION

A mouse can be divided into several functional blocks:

- Microcontroller
- Button detection
- Motion detection
- RS232 signal generation
- 5V DC power supply unit

A typical functional block diagram is shown in Figure 1.

In Figure 2, three push buttons are connected to the input ports of the PIC16C54. When a switch opening or closure is detected, a message is formatted and sent to the host. The X and Y movements are measured by counting the pulses generated by the photo-couplers. In the case of an opto-mechanical mouse, the infrared light emitted by the infrared diode is blocked by the rotating wheel, so that the pulses are generated on the photo-transistor side. In case of an optical mouse, the infrared light emitted by the infrared diode is reflected off the reflective pad patterned with vertical and horizontal grid lines. It is then received by the photo-transistor in the mouse. When any X or Y movement is detected, a message is formatted and sent to the host.
Mouse Controller

The Microsoft® Mouse System and the Mouse Systems® device both use serial input techniques. The Mouse System protocol format contains five bytes of data. One byte describes the status of three push buttons, two bytes for the relative X movements and two bytes for the relative Y movements. The Microsoft protocol format contains three bytes of data describing the status of two push buttons and the relative X and Y movements. The details of these protocols are given in Table 1.

Three lines are connected to the host via the RS232 port:
- Signal Ground
- Received Data
- Request to Send

"Received Data" carries the message sent by the mouse. While "Request to Send" provides a −10V DC for voltage conversion circuitry. A voltage of +5V DC is required for electronic components inside the mouse, however, +5V DC is not part of an RS232 port, so voltage conversion circuitry is required. This circuit is typically composed of a 555 timer, Zener diodes, and capacitors. An example circuit is shown in Figure 3. Since the current supplied through the RS232 port is limited to 10 mA, the mouse cannot be designed to consume more than 10 mA current unless an external power supply is provided. The PIC16C54, running at 4 MHz (1 μs instruction cycle) can provide a very high tracking speed. An 8 MHz version of PIC16C54 is also available if higher performance is desired.

FIGURE 2 - PIC16C54 PIN ASSIGNMENT

FIGURE 3 - VOLTAGE CONVERSION CIRCUITY
ABOUT THE SOFTWARE

The major tasks performed by the software are button scanning, X and Y motion scanning, formatting and sending serial data to the host. These tasks need to be performed in parallel in order to gain better tracking speed. The pulses generated by the photo-couplers are counted while transmitting the serial signals to the RS232 port. The number of pulses reflects the speed of the movement. The more number of pulses, the faster the movement is.

The directions of the movement are determined by the last states and the present states of the outputs of the photo-transistors. In Figure 4, XCLOCK and XDATA are outputs from the photo-transistors corresponding to the X-axis movement. XDATA is read when a rising or a falling edge of XCLOCK is detected. For right movement, XDATA is either LOW at the rising edge of XCLOCK or HIGH at the falling edge of XCLOCK. The up and down movement detections follow the same logic. In Table 1, X7:X0 are data for relative movement. If X is positive, it implies that the mouse is moving to the right. If X is negative, it implies a movement to the left. Similarly, if Y is positive, it indicates that the mouse is moving down and if Y is negative, it indicates that the mouse is moving up. The pulses generated by the photo-couplers are checked before every bit is sent. A bit takes 1/1200 second to send, if the distance between the grid lines is 1 mm, the tracking speed will be up to 1200 mm/second.

FIGURE 4 - VOLTAGE CONVERSION CIRCUITRY

![Voltage Conversion Circuitry Diagram](image)

TABLE 1 - MOUSE SYSTEM AND MICROSOFT PROTOCOLS

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Mouse System Format*</th>
<th>Microsoft Format*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1</td>
<td>1 0 0 0 0 L M R</td>
<td>1 1 L R Y7 Y6 X7 X6</td>
</tr>
<tr>
<td>Byte 2</td>
<td>X7 X6 X5 X4 X3 X2 X1 X0</td>
<td>0 0 X5 X4 X3 X2 X1 X0</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0</td>
<td>0 0 Y5 Y4 Y3 Y2 Y1 Y0</td>
</tr>
<tr>
<td>Byte 4</td>
<td>X7 X6 X5 X4 X3 X2 X1 X0</td>
<td></td>
</tr>
<tr>
<td>Byte 5</td>
<td>Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0</td>
<td></td>
</tr>
</tbody>
</table>

* L = Left Key Status, 1 = Pressed, X7-X0 = X-Axis Movement Data
M = Middle Key Status, 0 = Released, Y7-Y0 = Y-Axis Movement Data
R = Right Key Status

© 1993 Microchip Technology Inc.
Mouse Controller

The buttons are scanned after a message is sent and the time used to send the message is used as the debouncing time. The message is in an RS232 format with 1200 baud, eight data bits, no parity, and two stop bits.

The flow charts of the main program, subroutine BYTE and subroutine BIT are shown in Figures 5, 6, and 7. Figure 5 shows the Trigger Flag is set when any change of button status or X/Y movement is detected. Subroutine BYTE is called in the main program five times to send five bytes of information. Subroutine BYTE controls the status of the "Received Data" (RD) pin. If Trigger Flag is clear, RD will always be HIGH. Hence, no message will be sent even when subroutine BYTE is called. Figure 7 shows that subroutine BIT counts the number of pulses from outputs of the photo-transistors, determines the directions, and generates 1/1200 second delay to get 1200 baud timing.

The mouse has been tested in Mouse System Mode and is functioning properly. A completed listing of the source program is given in Appendix A.

SUMMARY

The PIC16C54 from Microchip Technology, Inc. provides a very cost-effective, high performance mouse implementation. Its low power (typically < 2 mA at 1 μs instruction cycle), small package (18-pin) and high reliability (on-chip watchdog timer to prevent software hang-ups) are among several reasons why the PIC16C54 is uniquely suitable for mouse applications.

This application note provides the user with a simple, fully functional serial mouse implementation. The user may use this as a starting point for a more comprehensive design. For fully implemented and compliant mouse products see Microchip's ASSP device family (MTA41XXX).
FIGURE 6 - FLOW CHART OF ROUTINE BYTE

Mouse Controller

- Byte
  - Count → '0'
    - Trigger Flag Set?
      - Y '0' → RD Pin (Start Bit)
      - N Call Routine Bit
        - Trigger Flag Set?
          - N Count = Count + 1
            - N Count = B?
              - N Return to Caller
FIGURE 7 - FLOW CHART OF ROUTINE BIT

Mouse Controller

© 1993 Microchip Technology Inc.
APPENDIX A:

MPASM B0.54

MOUSE

TITLE " MOUSE "
LIST P=16C54,R=0

;*******************************************************************************
; * MOUSE CONTROLLER
; * VERSION : 25 APRIL, 1990
; * MODE = PIC16C54XT CLK=4.0MHZ
;*******************************************************************************

FILES ASSIGNMENT

STATUS EQU 3
RA EQU 5
RB EQU 6
TIMER1 EQU 10
CSTAT EQU 14
BSTAT EQU 15
DATA0 EQU 16
DATA1 EQU 17
DATA2 EQU 20
DATA3 EQU 21
DATA4 EQU 22
FLAGA EQU 23
XCOUNT EQU 24
YCOUNT EQU 25
DATA_AREA EQU 31

BIT ASSIGNMENT

YC EQU 0
YD EQU 1
UP EQU 1
XC EQU 2
XD EQU 3
RI EQU 3
BU1 EQU 0
BU2 EQU 2
CA EQU 0
RD EQU 7
ZERO_AREA EQU 2
TR EQU 2

SUBROUTINES

ORG 0

;*******************************************************************************
; DELAY A BIT TIME AND CHECK XC & YC STATUS
;*******************************************************************************
Mouse Controller

BIT

0000 0745 BTFSR RA,XC ;XC = 1 ?
0001 0A0A GOTO BIT0
0002 064C BTFSR CSTAT,XC ;(XC=1)
0003 0A11 GOTO BITY ;(XC ALWAYS = 1)
0004 02B4 INCF XCOUNT ;(XC +1)
0005 0476 BCF FLAGB,RI ;DEFAULT LEFT
0006 0765 BTFSR RA, XD ;LEFT / RIGHT ?
0007 0A11 GOTO BITY
0008 0576 BSF FLAGB,RI
0009 0A11 GOTO BITY

BIT0

000A 074C BTFS R CSTAT,XC ;(XC=0)
000B 0A11 GOTO BITY0
000C 02B4 INCF XCOUNT ;(XC +1)
000D 0476 BCF FLAGB,UP ;DEFAULT DOWN
000E 0665 BTFSR RA, XD ;DOWN / UP ?
000F 0A11 GOTO BITY
0010 0576 BSF FLAGB,RI

BITY

0011 0705 BTFSR RA,YC ;YC = 1 ?
0012 0A1B GOTO BITY0
0013 060C BTFSR CSTAT,YC ;(YC=1)
0014 0A22 GOTO BITDY ;(YC ALWAYS = 1)
0015 02B5 INCF YCOUNT ;(YC +1)
0016 0436 BCF FLAGB,UP ;DEFAULT DOWN
0017 0725 BTFSR RA, XD ;DOWN / UP ?
0018 0A22 GOTO BITDY
0019 0536 BSF FLAGB,UP
001A 0A22 GOTO BITDY

BITY0

001B 070C BTFSR CSTAT,YC ;(YC=0)
001C 0A22 GOTO BITDY ;(YC ALWAYS = 0)
001D 02B5 INCF YCOUNT ;(YC +1)
001E 0436 BCF FLAGB,UP ;DEFAULT DOWN
001F 0625 BTFSR RA, XD ;DOWN / UP ?
0020 0A22 GOTO BITDY
0021 0536 BSF FLAGB,UP

BITDY

0022 0205 MOVF RA,W ;SAVE COOR. STATUS
0023 002C MOVWF CSTAT
0024 193D MOVLW 193D ;0.833 MS DELAY
0025 0028 MOVWF TIMER1

BITDO

0026 0000 NOP
0027 0268 DECFSZ TIMER1
0028 0A26 GOTO BIDDO
0029 0B00 RETLW 0

BYTE

002A 0078 CLR F COUNT ;RESET 8 BIT COUNT
002B 0753 BTFSR FLAGA,TR ;ANY TRIGGER
002C 0A2E GOTO BYTE0
002D 0466 BCF RB,RD ;LOW RD FOR START BIT

BYTE0

002E 0900 CALL BIT

BYTE1

002F 0753 BTFSR FLAGA,TR ;ANY TRIGGER ?
0030 0A37 GOTO BYTE3
0031 0339 RRF DATA_AREA ;SHIFT DATA TO CARRY
0032 0703 BTFSR STATUS,CA ;0 / 1 ?
0033 0A36 GOTO BYTE2
Mouse Controller

0034 05E6  
0035 0A37  BSF RB,RD ;SEND A 1
                GOTO BYTE3
BYTE2
0036 04E6  
            BCF RB,RD ;SEND A 0
                GOTO BYTE3
            CALL BIT
            INCF COUNT
            BTFSS COUNT,3 ;COUNT = 8 ?
BYTE1
0038 02B8  GOTO BYTE4
0039 0778  BCF RB,RD ;SEND SENT BIT
003A 0A2F  GOTO BYTE4
003B 07S3  BTFSS FLAGA,TR ;ANY TRIGGER ?
003C 0A42  GOTO BYTE4
003D 04E6  BCF RB,RD
003E 0900  CALL BIT
003F 05E6  CALL BIT
0040 0900  BSF RB,RD
0041 0A44  GOTO BYTE5
BYTE4
0042 0900  CALL BIT
0043 0900  CALL BIT
BYTE5
0044 0800  RETLW 0
;---------------------------------------------
; RESET ENTRY
;---------------------------------------------
; INIT
0045 0CC1  MOVWF B'11000001' ;DISABLE WATCH DOG
0046 0002  OPTION
0047 0CCF  MOVWF B'00001111' ;INIT RB0-3 BE INPUTS
0048 0006  TRIS RB ;RB4-7 BE OUTPUTS
0049 0CCF  MOVWF B'11111111' ;INIT RA0-3 BE INPUTS
004A 0005  TRIS RA
004B 05E6  BSF RB,RD ;HIGH RD PIN
004C 0246  COMF RB,W ;GET INIT BUTTON INPUTS
004D 0E05  ANDLW B'00000101'
004E 0D80  IORLW B'10000000'
004F 002D  MOVWF BSTAT
0050 002E  MOVWF DATA0
0051 0205  MOVF RA,W
0052 002C  MOVWF CSTAT
0053 0073  CLRF FLAGA ;CLEAR TR FLAG
0054 0074  CLRF XCOUNT ;RESET XCOUNT & YCOUNT
0055 0075  CLRF YCOUNT
SCAN
0056 006F  CLRF DATA1 ;UPDATE X,Y MOVEMENT DATA
0057 0070  CLRF DATA2
0058 0071  CLRF DATA3
0059 0072  CLRF DATA4
005A 0214  MOVF XCOUNT,W ;XCOUNT 0
005B 0743  BTFSS STATUS,ZERO_AREA
005C 0A80  GOTO WRITX
SCANA
005D 0215  MOVF Y COUNT,W ;YCOUNT 0
005E 0743  BTFSS STATUS,ZERO_AREA
005F 0A92  GOTO WRITY
SCANN
0060 0246  COMP RB,W ;BUTTON STATUS CHANGE ?
0061 0E05  ANDLW B'00000101'
0062 0D80  IORLW B'10000000'
0063 00AD  SUBWF BSTAT
0064 0643  BTFSC STATUS,ZERO_AREA ;IF CHANGE THEN TRIGGER
0065 0A6B  GOTO SCANC ;(NO CHANGE)
0066 0553  BSF FLAGA,TR ;(CHANGE) SET TRIGGER FLAG
0067 0246  COMP RB,W
0068 0E05  ANDLW B'00000101'
0069 0D80  IORLW B'10000000'

© 1993 Microchip Technology Inc. DS00519B-page 9
Mouse Controller

006A 002E MOVWF DATA0
006B 0246 COMF RB,W
006C 0E05 ANDLW B'00000101'
006D 0D80 IORLW B'10000000'
006E 002D MOVWF BSTAT
006F 020E MOVF DATA0,W ;SEND DATA0,1,2,3,4 TO HOST
0070 0039 MOVWF DATA_AREA
0071 020F MOVF DATA0,W ;SEND DAT~0,1,2,3,4 TO HOST
0072 092A CALL BYTE
0073 020F MOVF DATA2,W
0074 092A CALL BYTE
0075 0210 MOVF DATA2,W
0076 092A CALL BYTE
0077 0211 MOVF DATA3,W
0078 0211 MOVF DATA3,W
0079 092A CALL BYTE
007A 0212 MOVF DATA4,W
007B 092A CALL BYTE
007C 0213 MOVF DATA4,W
007D 092A CALL BYTE
007E 0453 BCF FLAGA,TR ;CLEAR TRIGGER FLAG
007F 0A56 GOTO SCAN

; WRITX
0080 0553 BSF FLAGA,TR ;SET TRIGGER FLAG
0081 0C40 MOVFLW 40H ;IF XCOUNT > 64 THEN XCOUNT <-64
0082 0094 SUBWF XCOUNT,W
0083 0603 BTFSC STATUS,CA
0084 0A8D GOTO WRITR

; WRITS
0085 0776 BTFSS FLAGB,RI ;LEFT / RIGHT ?
0086 0A90 GOTO WRITL
0087 0274 COMF XCOUNT ; (RIGHT) NEG XCOUNT
0088 0295 NCF XCOUNT,W
0089 002F MOVWF DATA1
008A 0031 MOVWF DATA3
008B 0074 CLRF XCOUNT ;RESET XCOUNT
008C 0A5D GOTO SCANA

; WRITR
008D 0C40 MOVFLW 40H ;XCOUNT <- 64
008E 0034 MOVWF XCOUNT
008F 0A85 GOTO WRITS

; WRITL
0090 0214 MOVF XCOUNT,W ;(LEFT)
0091 0A89 GOTO WRITA

; WRITY
0092 0553 BSF FLAGA,TR ;SET TRIGGER FLAG
0093 0C40 MOVFLW 40H ;IF YCOUNT > 64 THEN YCOUNT <-64
0094 0095 SUBWF YCOUNT,W
0095 0603 BTFSC STATUS,CA
0096 0A9F GOTO WRITV

; WRITV
0097 0736 BTFSS FLAGB,UP ;DOWN / UP ?
0098 0AA2 GOTO WRITD
0099 0275 COMF YCOUNT ;(UP) NEG YCOUNT
009A 0295 I NCF YCOUNT,W

; WRITB
009B 0030 MOVWF DATA2
009C 0032 MOVWF DATA4
009D 0075 CLRF YCOUNT ;RESET YCOUNT
009E 0A60 GOTO SCANB
Mouse Controller

WRITV
009F OC40 MOVWM 40H ; YCOUNT <- 64
00A0 0035 MOVWF YCOUNT
00A1 0A97 GOTO WRITW

; WRITD
00A2 0215 MOVF YCOUNT,W ; (DOWN)
00A3 0A9B GOTO WRITB

;===========================================
.RESET ENTRY
;===========================================
ORG 777
01FF 0A45 GOTO INIT ; JUMP TO PROGRAM STARTING
;
END
;
;**************************************************

Errors : 0
Warnings : 0
INTRODUCTION

The excellent cost/performance ratio of the PIC16C5X are well suited for a low-cost proportional d.c. actuator controller. This application note depicts a design of a remote intelligent positioning system using a d.c. motor (up to 1/3 hp) run from 12 to 24 V. The position accuracy is 1 in 8 bits or 0.4%. The PIC16C5X receives its command and control information via a MICROWIRE™ serial bus. However, any serial communication method is applicable.

IMPLEMENTATION

The PIC16C5X based controller receives commands from a host, compares them to the actual position, calculates the desired motor drive level and then pulses a full H-bridge (Figure 2). In this way it serves as a remote intelligent positioner, driving the load until it has reached the commanded position. It can be used to control any proportional d.c. actuator i.e. d.c. motor or proportional valve.

This system is ideally suited to remotely position valves and machinery. It can be used with d.c. motors to easily automate manual equipment. Because of the 5 wire serial interface, the positioner can be installed near its power supply and load. The remote intelligent positioner can then be linked to the central control processor by a small diameter easily routed cable. Since the positioner is running its own closed-loop PIO algorithm (Fig. 3), the host central processor need only send position commands and is therefore free to service the user interface, main application software and command many remote positioners.

The limit switch inputs provide a safety net to keep a system from destroying itself in the event that the feedback device is lost. The optional current sense input can be used to determine if the load has jammed and prevent overheating of the actuator and drive electronics.

The commanded positions are presented to the PIC16C5X via a microwire type protocol at bit-rates of up to 50kb/s for the 4 MHz part. As currently implemented in this application note, the position request is the only communication. There are several variable locations available and they could be utilized to allow downloading of the loop gain parameters, reading positioner information, or for setting a current limit. The host that is sending the position request must set the chip select low, and wait for the PIC16C5X to raise the "busy" (DO) line high. At this point 8 data bits can be clocked into the PIC16C5X. The requested position is sent most signifi-cant bit first and can be any 8 bit value. Values 1 through 255 represent valid positions with 0 being reserved for drive disable.

The PIC16C5X acquires data by way of a microwire A/D converter. This part was chosen for low cost yet it provides adequate performance. The second channel of the A/D is shown hooked up to a peak current detector. If the user desired, the PIC16C5X could monitor and protect the motor from overcurrent by monitoring this information.

The H-bridge power amplifier will deliver 10 or more amps at up to 24 volts when properly heat-sinked. It is wired for a modified 4-quadrant mode of operation. One leg of the bridge is used to control direction and the other leg pulses the low FET and the high FET alternately to generate the desired duty-cycle. In this way the system will operate well to produce a desired "speed" without the use of a separate speed control loop. This allows use of the PIC16C5X to control the PID algorithm for position directly while having reasonable speed control. The capacitance at the gates of the FETs combined with the impedance of the drive circuits provides for turn off of the upper FET before the lower FET turns on... an important criteria.

FIGURE 1 - BLOCK DIAGRAM
The PID algorithm itself is where most of the meat of this application note is located so let's look at it more closely. The Algorithm is formed by summing the contribution of three basic components. The first calculation is the error for that is what the other terms are based on.

The error is the requested position minus the actual position. It is a signed number whose magnitude can be 255. In order not to lose resolution, the error is stored as an 8 bit magnitude with the sign stored separately in the FLAGS register under ER_SGN. This allows us to resolve a full signed 8 bit error with 8 bit math.

The proportional term is merely the algebraic difference of the requested position minus the actual position. It is scaled by a gain term (Kp) called the "proportional gain". The sign of this term is important for it tells the system which direction it must drive to correct the error. The proportional term is limited to +/- 100. Increasing the proportional gain term will improve the dynamic and static accuracy of the system. Increasing it too much will cause oscillations.

The next term that gets calculated is the Integral term. This term is traditionally formed by integrating the error over time. In this application it is done by integrating the Ki term over time. When the error is zero, no integration is performed. This is a more practical way to handle a potentially large number in 8 bit math. By increasing the Ki term the d.c. or static gain of the system is improved. Increasing the integral gain too much can lead to low frequency oscillations.

The differential term (Kd) is a stabilizing term that helps keep the integral and proportional terms from overdriving the system through the desired position and thus creating oscillations. As you use more proportional and integral gain you will need more differential gain as well. The differential gain is calculated by looking at the rate of change of the positional error with respect to time. It is actually formed as "delta error/delta time" with the delta time being a program cycle.

The three terms are summed algebraically and scaled to produce a percentage speed request between 0 and 100%. The sign of the sum is used to control the H-bridge direction. The loop calculations run approximately 20 times per second on a 4 MHz part. This yields sufficient gain-bandwidth for most positioning applications. If higher system performance is desired, the number of pulses can be reduced to 20 and a 16 MHz PIC16C5X can be used. Your Loop gains (Kp, Ki, Kd) will have to be recalculated, but the system sample rate will be increased to 400 Hz. This should be sufficient to control a system that has a response time of 20 milliseconds or more.

The key to using the PIC16C5X series parts for PID control and PWM generation is to separate the two into separate tasks. There is simply not the hardware support or the processing speed to accurately do both concurrently. It is fortunate therefore that it is not necessary to do both concurrently. The systems that are generally controlled can be stabilized with a much lower information update rate than the PWM frequency. This supports the approach of calculating the desired percentage, outputting the PWM for a period of time and then recalculating the new desired percentage. Utilizing this technique the inexpensive PIC16C5X can implement PID control, PWM generation and still have processing time left over for monitor or communication functions.

About the Author:
Steven Frank has been designing analog and digital control systems for 10 years. His background is in medical and consumer electronics. He has received numerous patents in control systems and instrumentation. At Vesta Technology Inc. Mr. Frank works with a number of engineers on custom embedded control systems designs. Vesta Technology Inc. is a provider of embedded control systems from an array of standard products and designs. Vesta offers custom design services and handles projects from concept to manufacturing.
Remote Positioner

FIGURE 2 - PROGRAM FLOW CHART

Start

Limit switches set?

Y

N

New position requested?

Y

N

Get new position = POSR

Get actual position = POSA

Determine P.I.D. term = PENT and direction

Set CNT = 100

PCH ← PCNT; PCL ← 100 - PCNT

Drive Motor hi; PCH ← PCH - 1

PCH = 0?

Y

N

Drive Motor lo; PCL ← PCL - 1

PCH = 0?

Y

N

CNT ← CNT - 1

CNT = 0?

Y

N

© 1993 Microchip Technology Incorporated

FIGURE 3 - P.I.D. ALGORITHM FLOW CHART

Start

ERR = POSR - POSA

Kperr ← Min. of [Kp*ERR or 100]

SUM ← SUM + Kperr

ERR = Positive

Y

N

ACCM = ACCM + KI

Direction = C.W.

ACCM = ACCM - KI

Direction = C.C.W.

SUM ← SUM + Min. of [ACCM or 100]

de/dt = ERR - OLD_ERR

Koerr = KD * de/dt

SUM ← SUM + Min. of [Koerr or 100]

SUM Positive

Y

N

Set Bridge for C.W.

Set Bridge for C.W.

PCNT = Min. of [SUM or 100]

End
Remote Positioner

FIGURE 4 - SCHEMATIC

Notes:
1. All pnp transistors are 2N3906
2. All npn transistors are 2N3904
3. All diodes 1N914 unless otherwise specified
4. All zeners are 1N4742
Remote Positioner

LIST P=16C56
;**************************************************************
REV. A Original release 1/10/92 srf
;**************************************************************
;REGISTER EQUATES

w                   EQU     00H  ; CONTENTS OF POINTER
0000 W               EQU     00H  ; USE THIS VARIABLE LOCATION AS FLAGS
0019 PNTR            EQU     19H  ; 0 BIT IS SIGN OF ERROR 1 IS NEGATIVE
0000 PORTA           EQU     06H  ; 1 BIT IS SIGN OF ERROR ACCUMULATOR
0006 PORTB           EQU     06H  ; 2 BIT IS SIGN OF THE DE/DE TERM
0007 HI              EQU     07H  ; 3 BIT IS DIRECTION 0 IS CW
0008 LO              EQU     08H  ; Bit is sign of the old error
0009 PCNT            EQU     09H  ; STATUS Word Register
0004 FSR             EQU     04H  ; 0 = CARRY
0005 PORTA           EQU     05H  ; 1 = DC
0006 PORTB           EQU     05H  ; 2 = Z, SET IF RESULT IS ZERO
0007 HI              EQU     07H  ; FILE SELECT REGISTER
0008 LO              EQU     08H  ; 1 - DC
0009 PCNT            EQU     09H  ; 2 - Z, SET IF RESULT IS ZERO
000A HI_T            EQU     0AH  ; USE THIS VARIABLE LOCATION AS FLAGS
000B LO_T            EQU     0BH  ; Bit is sign of error
000C ERROR           EQU     0CH  ; Bit is sign of the old error
0003 STATUS          EQU     03H  ; USE THIS VARIABLE LOCATION AS FLAGS
0003 SWR             EQU     03H  ; Bit is sign of the old error
0004 FSR             EQU     04H  ; STATUS Word Register
0005 PORTA           EQU     05H  ; 0 = CARRY
0006 PORTB           EQU     05H  ; 1 = DC
0007 HI              EQU     07H  ; 2 = Z, SET IF RESULT IS ZERO
0008 LO              EQU     08H  ; FILE SELECT REGISTER
0009 PCNT            EQU     09H  ; 1 - DC
000A HI_T            EQU     0AH  ; 2 - Z, SET IF RESULT IS ZERO
000B LO_T            EQU     0BH  ; USE THIS VARIABLE LOCATION AS FLAGS
000C ERROR           EQU     0CH  ; USE THIS VARIABLE LOCATION AS FLAGS
0003 STATUS          EQU     03H  ; STATUS Word Register
0003 SWR             EQU     03H  ; STATUS Word Register
0004 FSR             EQU     04H  ; STATUS Word Register
0005 PORTA           EQU     05H  ; STATUS Word Register
0006 PORTB           EQU     05H  ; STATUS Word Register
0007 HI              EQU     07H  ; STATUS Word Register
0008 LO              EQU     08H  ; STATUS Word Register
0009 PCNT            EQU     09H  ; STATUS Word Register
000A HI_T            EQU     0AH  ; STATUS Word Register
000B LO_T            EQU     0BH  ; STATUS Word Register
000C ERROR           EQU     0CH  ; STATUS Word Register
0000 PWMCW           EQU     00H  ; STATUS Word Register
0001 PWMCNW          EQU     01H  ; STATUS Word Register
0000 CARRY           EQU     00H  ; STATUS Word Register
0002 Z               EQU     02H  ; STATUS Word Register
0001 Same             EQU     01H  ; STATUS Word Register

; PORT ASSIGNMENTS AND CONSTANTS

© 1993 Microchip Technology Incorporated

2-137
Remote Positioner

0000 ER_SGN EQU 0 ; SIGN BIT FOR THE ERROR IN FLAG REGISTER
0001 AC_SGN EQU 1 ; SIGN BIT FOR THE ERROR ACCUMULATOR
0002 DE_SGN EQU 2 ; SIGN BIT FOR DE/DT
0004 OER_SGN EQU 4 ; SIGN BIT FOR THE OLD ERROR
0030 KP EQU 30 ; PROPORTIONAL GAIN
0002 KI EQU 2 ; INTEGRAL GAIN
0020 KD EQU 20 ; DIFFERENTIAL GAIN
0003 DIR EQU 3 ; THE DIRECTION FLAG
0007 CSN EQU 7 ; CHIP SELECT NOT ON A/D
0006 BV EQU 6 ; DATA LINE FOR THE A/D
0005 CK EQU 5 ; CLOCK LINE FOR THE A/D
0002 MWDO EQU 2 ; MICROWIRE DATA OUT FROM POSITIONER
0001 MWDI EQU 1 ; MICROWIRE DATA IN TO POSITIONER
0000 MWCS EQU 0 ; MICROWIRE CHIP SELECT TO POSITIONER
0003 MWCK EQU 3 ; MICROWIRE CLOCK IN TO POSITIONER

•***** MACROS **********************************************
; CLKUP MACRO ; clock up macro for the microwire
BSF PORTB,CK ; data acquisition from the a/d
NOP
ENDM

CLKDN MACRO ; clock down macro for the microwire
BCF PORTB,CK ; data acquisition from the a/d
NOP
ENDM

GET_BIT MACRO ; ** FOR RECEIVING A/D DATA **
BCF SWR,CARRY ; SET CLOCK BIT HIGH
BSF PORTB,CK ; LOOK AT DATA COMMING IN
BTFS C PORTB,BV ; SET THE CARRY FOR A 1
RLF POSA ; ROTATE THE W REG LEFT
BCF PORTB,CK ; SET THE CLOCK LOW
NOP
DELAY
ENDM

0000 0B88

GOTO CLRREG

; ***** MATH ROUTINES ****************************************
; ; **** 8 BIT MULTIPLY ******
; ; ****************************************************************************

mpy_S clrf H_byte
clrf L_byte
rnovlw 8
movlw count
movf rnulcnd,w
bcf STATUS,CARRY

loop rrf mulpr
btfsc STATUS,CARRY ; Clear the carry bit in the status Reg.
bcf STATUS,CARRY

001 0075
0002 0076
0003 0080
0004 0037
0005 0213
0006 01F5
0007 0335
0008 0336
0009 00F7
000A 00F8
000B 00F9
000C 00FA
000D 00FB
000E 00FC

; *******************************
; DOUBLE PRECISION ADD AND SUBTRACT ( ACCb-ACCa->ACCb )

000F 0917

D_sub call neg_A ; At first negate ACCa, then add

; ****************************
; Double Precision Addition ( ACCb+ACCa->ACCb )

0010 0213
0011 01F4

D_add movf ACCaLO,w
addwf ACCbLO ; add lsb

© 1993 Microchip Technology Incorporated

DS00531B-page 6

2-138
Remote Positioner

```
0012  0603  btfsc STATUS,CARRY ; add in carry
0013  0286  incf ACCbHI
0014  0215  movf ACCah,W
0015  01F6  addwf ACCbHI ; add msb
0016  0800  retlw 00

; 0017  0273  neg_A  comf ACCaLO ; negate ACCa
0018  02B3  incf ACCaLO
0019  0643  btfsc STATUS,Z
001A  00F5  decf ACCah1
001B  0275  comf ACCah1
001C  0800  retlw 00

; *****************************************************
; divide by 16 and limit to 100 Decimal

SHIFT MACRO
    BCF    SWR,CARRY
    RRF    L_byte
    BCF    SWR,CARRY
    RRF    H_byte
    BTFSC SWR,CARRY
    BSF    L_byte,7
ENDM

DIV_LMT
    SHIFT
001D  0403  BCF    SWR,CARRY
001E  0336  RRF    L_byte
001F  0403  BCF    SWR,CARRY
0020  0335  RRF    H_byte
0021  0603  BTFSC SWR,CARRY
0022  05F6  BSF    L_byte,7

SHIFT
0023  0403  BCF    SWR,CARRY
0024  0336  RRF    L_byte
0025  0403  BCF    SWR,CARRY
0026  0335  RRF    H_byte
0027  0603  BTFSC SWR,CARRY
0028  05F6  BSF    L_byte,7

SHIFT
0029  0403  BCF    SWR,CARRY
002A  0336  RRF    L_byte
002B  0403  BCF    SWR,CARRY
002C  0335  RRF    H_byte
002D  0603  BTFSC SWR,CARRY
002E  05F6  BSF    L_byte,7

SHIFT
002F  0403  BCF    SWR,CARRY
0030  0336  RRF    L_byte
0031  0403  BCF    SWR,CARRY
0032  0335  RRF    H_byte
0033  0603  BTFSC SWR,CARRY
0034  05F6  BSF    L_byte,7

LMT100
0035  00C1  MOVLW 1H ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0036  0095  SUBWF  H_byte,0 ; IF THERE IS ANYTHING THERE, IF NOT,
0037  0703  BTFSS SWR,CARRY ; THEN LEAVE THE LOW BYTE ALONE
0038  0A3C  GOTO LB_E ; OTHERWISE GIVE THE LOW BYTE A FULL
0039  0C64  MOVLW 64H ; COUNT AND IT WILL HAVE BEEN LIMITED
003A  0036  MOVWF L_byte ; TO 100
003B  0A42  GOTO LMT_EXIT

LB_E  MOVLW 64H ; LIMIT THE MAGNITUDE OF THE VALUE TO
```
Remote Positioner

```assembly
003D 0096 SUBWF L_BYTE,0 ; 100 DECIMAL
003E 0703 BTFSS SWR,CARRY
003F 0A42 GOTO LMT_EXIT
0040 0C64 MOVLW 64H
0041 0036 MOVWF L_BYTE

LMT_EXIT
0042 0800 RETLW 00

;THE ROUTINE CALCTIMES DOES THE FOLLOWING: PCNT = DUTY CYCLE IN %
; 100 - PCNT -> LO AND PCNT -> HI. ZERO VALUES IN EITHER LO OR HI
;ARE FORCED TO 1.
CALCTIMES
0043 0209 MOVF PCNT,W ; PUT REQUESTED % INTO W REGISTER
0044 0027 MOVWF HI ; COPY ON MICROSECONDS IN TO HI TIME
0045 0C64 MOVLW 64H
0046 0028 MOVWF LO
0047 0209 MOVWF PCNT,O
0048 00A8 SUBWF LO,1 ; LEAVE 100-HI TIME IN LO TIME
0049 0207 MOVF HI,0 ; INSPECT THE HIGH TIME
004A 0643 BTFSC SWR,2 ; IF ITS IS ZERO
004B 02A7 INCF HI,1 ; INCREMENT IT
004C 0208 MOVF LO,0 ; INSPECT THE LO TIME
004D 0643 BTFSC SWR,2 ; IF ITS ZERO
004E 02A8 INCF LO,1 ; INCREMENT IT
004F 0800 RETLW 00

;*******************************************************************
BEGIN
0050 0000 NOP ; STUBBED BEGINNING

;****CHECKING THE LIMIT SWITCHES AND CHECKING FOR MW**************
; This will check the switch inputs for closure and will terminate
; pulsing is one is closed. It doesn't distinguish between the switches
; so they are not dedicated to cw end and ccw end.
SW_TRAP
0051 0004 CLRWDI
0052 0746 BTFSS PORTB,2 ; THIS WILL TEST ALL THREE OF THE
0053 0A51 GOTO SW_TRAP ; SWITCH INPUTS. IF ANY ONE IS
0054 0766 BTFSS PORTB,3 ; SET THEN EXECUTION OF THE CODE
0055 0A51 GOTO SW_TRAP ; WILL BE LIMITED TO LOOKING FOR
0056 0766 BTFSS PORTB,4 ; IT TO BE CLEARED
0057 0A51 GOTO SW_TRAP

;****RECEIVING THE POSITIONAL REQUEST****************************
; The host system that wishes to send positional requests to the positioner
; servo makes its desire known by setting the chip select to the positioner
; low. It then monitors the busy (Data Out) line from the positioner. When
; the positioner sets the busy line high, the host may begin sending its 8
; request. The data bits should be valid on the rising edge of the clock.
; After 8 bits have been received by the positioner it will begin operation
; to send the system to the received position. It can be interrupted at any
; point during the positioning process by the host sending a new command.
; opportunity to update the command is issued every 100 pwm pulses (every 50
; milliseconds).
; If the host sends a zero positional command the positioner will stop the
; system and remain inactive.
; If the host does not successfully complete a mtcrowire transmission of 8
; data bits the watchdog timer will trip and reset the system to an inactive
; "stopped" state.

REC_MW
0058 0C0B MOVLW 0BH ; RESET THE PORT FOR THREE INPUTS
```

© 1993 Microchip Technology Incorporated
Remote Positioner

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0059 0005</td>
<td>TRIS PORTA</td>
<td>; AND ONE OUTPUT</td>
</tr>
<tr>
<td>005A 0445</td>
<td>BCF PORTA,MWDO</td>
<td>; SET THE DATA OUT LOW FOR BUSY</td>
</tr>
<tr>
<td>005B 0C20</td>
<td>MOVWF count</td>
<td></td>
</tr>
<tr>
<td>005C 0037</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005D 0705</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005E 0A62</td>
<td>BTFSS PORTA,MWCS</td>
<td>; CHECK FOR INCOMING REQUESTS</td>
</tr>
<tr>
<td>005F 0F7F</td>
<td>GOTO REC_CMD</td>
<td>; RECEIVE A NEW POSITION REQUEST</td>
</tr>
<tr>
<td>0060 0A5D</td>
<td>DECFZ count,1</td>
<td></td>
</tr>
<tr>
<td>0061 0A71</td>
<td>GOTO WATCH_CS</td>
<td>; NO REQUEST WAS MADE IN THE TIME ALLOCATED</td>
</tr>
<tr>
<td>0062 0545</td>
<td>BSF PORTA,MWDO</td>
<td>; SET THE DATA OUT HIGH FOR &quot;OK TO SEND&quot;</td>
</tr>
<tr>
<td>0063 0C08</td>
<td>MOVWL 4H</td>
<td>; SET TO RECEIVE 8 BITS</td>
</tr>
<tr>
<td>0064 0037</td>
<td>MOVWF count</td>
<td></td>
</tr>
<tr>
<td>0065 0765</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0066 0A65</td>
<td>BTFSS PORTA,MWCK</td>
<td>; WAIT FOR A RISING EDGE</td>
</tr>
<tr>
<td>0067 0403</td>
<td>GOTO WAIT_UP</td>
<td></td>
</tr>
<tr>
<td>0068 0625</td>
<td>BCF SWR,CARRY</td>
<td>; RESET THE CARRY TO A DEFAULT ZERO</td>
</tr>
<tr>
<td>0069 0503</td>
<td>BTFSC PORTA,MWDI</td>
<td>; READ THE DATA IN</td>
</tr>
<tr>
<td>006A 0370</td>
<td>BSF SWR,CARRY</td>
<td>; SET THE CARRY FOR A ONE</td>
</tr>
<tr>
<td>006B 02F7</td>
<td>DECFZ count,1</td>
<td>; DECREMENT THE BIT COUNTER</td>
</tr>
<tr>
<td>006C 0A6E</td>
<td>GOTO WAIT_DN</td>
<td>; WAIT FOR THE FALLING EDGE</td>
</tr>
<tr>
<td>006D 0A71</td>
<td>GOTO REC_EXIT</td>
<td>; LAST BIT RECEIVED</td>
</tr>
<tr>
<td>006E 0665</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006F 0A6E</td>
<td>BTFSC PORTA,MWCK</td>
<td>; CHECK THE INCOMING CLOCK</td>
</tr>
<tr>
<td>0070 0A65</td>
<td>GOTO WAIT_DN</td>
<td>; IF IT IS STILL HIGH WAIT FOR IT TO GO LOW</td>
</tr>
<tr>
<td>0070 0A65</td>
<td>GOTO WAIT_UP</td>
<td>; IF IT GOES LOW GO BACK TO RECEIVE NEXT BIT</td>
</tr>
<tr>
<td>0071 0445</td>
<td>BCF PORTA,MWDO</td>
<td>; SET THE BUSY FLAG</td>
</tr>
</tbody>
</table>

;********** CHECK FOR THE DISABLE REQUEST ***********************
; Position 0 is considered a request to not drive the system. In this way
; the positioner will come up from a reset in a safe state and will not
; try to move the system to some arbitrary location.

MOVE?
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0072 0210</td>
<td>MOVF POSR,W</td>
<td>; CHECK THE REQUESTED POSITION</td>
</tr>
<tr>
<td>0073 0643</td>
<td>BTFSC SWR,Z</td>
<td>; IF IT IS ZERO THEN WAIT FOR A NON-ZERO</td>
</tr>
<tr>
<td>0074 0A50</td>
<td>GOTO BEGIN</td>
<td>; REQUEST BY BRANCHING BACK TO THE BEGINNING</td>
</tr>
</tbody>
</table>

;****READING THE A/D VALUES***************************************
; Read the positional a/d channel (1) and store the value in the actual
; position variable (POSA).
; This is written in line to minimize the use of variables

READ_POS
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0075 0071</td>
<td>CLR PORTA,WS</td>
<td>; CLEAN THE POSITION ACTUAL HOLDER</td>
</tr>
<tr>
<td>0076 0466</td>
<td>BCF PORTB,CSN</td>
<td>; SET THE CHIP SELECT LOW TO A/D</td>
</tr>
<tr>
<td>0077 0C1C</td>
<td>MOV PORTB,1CH</td>
<td>; SET THE DATA LINE TO OUTPUT</td>
</tr>
<tr>
<td>0078 0006</td>
<td>TRIS PORTB</td>
<td>; FOR SENDING SET-UP BITS</td>
</tr>
<tr>
<td>0079 05C6</td>
<td>BSF PORTB,RV</td>
<td>; SET FOR &quot;START&quot; BIT</td>
</tr>
<tr>
<td>007A 0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLKUP</td>
<td>; CLOCK IN THE START BIT</td>
</tr>
<tr>
<td>007B 05A6</td>
<td>BSF PORTB,CK</td>
<td>; data acquisition from the a/d</td>
</tr>
<tr>
<td>007C 0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLKDN</td>
<td></td>
</tr>
<tr>
<td>007D 04A6</td>
<td>BCF PORTB,CK</td>
<td>; data acquisition from the a/d</td>
</tr>
<tr>
<td>007E 0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLKUP</td>
<td>; CLOCK IN SINGLE-ENDED</td>
</tr>
<tr>
<td>007F 05A6</td>
<td>BSF PORTB,CK</td>
<td>; data acquisition from the a/d</td>
</tr>
<tr>
<td>0080 0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLKDN</td>
<td></td>
</tr>
</tbody>
</table>
Remote Positioner

0081 04A6  BCF  PORTB,CK  ; data acquisition from the a/d
0082 0000  NOP

0083 05A6  BCF  PORTB,CK  ; CLOCK IN CHANNEL 1
0084 0000  NOP

0085 04A6  BCF  PORTB,CK  ; TO THE MUX
0086 0000  NOP

0087 0C5C  MOVW  5CH
0088 0006  TRIS PORTB
0089 05A6  BSF  PORTB,CK  ; data acquisition from the a/d
008A 0000  NOP

008B 04A6  BCF  PORTB,CK  ; CLOCK DN TO LET MUX SETTLE
008C 0000  NOP

008D 0403  BCF  SWR,CARRY
008E 05A6  BSF  PORTB,CK  ; SET CLOCK BIT HIGH
008F 06C6  BFSC PORTB,BV
0090 0503  BSF  SWR,CARRY
0091 0371  RLF  POSA
0092 04A6  BCF  PORTB,CK
0093 0000  NOP

0094 0403  BCF  SWR,CARRY
0095 05A6  BSF  PORTB,CK
0096 06C6  BFSC PORTB,BV
0097 0503  BSF  SWR,CARRY
0098 0371  RLF  POSA
0099 04A6  BCF  PORTB,CK
009A 0000  NOP

009B 0403  BCF  SWR,CARRY
009C 05A6  BSF  PORTB,CK
009D 06C6  BFSC PORTB,BV
009E 0503  BSF  SWR,CARRY
009F 0371  RLF  POSA
00A0 04A6  BCF  PORTB,CK
00A1 0000  NOP

00A2 0403  BCF  SWR,CARRY
00A3 05A6  BSF  PORTB,CK
00A4 06C6  BFSC PORTB,BV
00A5 0503  BSF  SWR,CARRY
00A6 0371  RLF  POSA
00A7 04A6  BCF  PORTB,CK
00A8 0000  NOP

00A9 0403  BCF  SWR,CARRY
00AA 05A6  BSF  PORTB,CK
00AB 06C6  BFSC PORTB,BV
00AC 0503  BSF  SWR,CARRY
00AD 0371  RLF  POSA
00AE 04A6  BCF  PORTB,CK
00AF 0000  NOP

00B0 0403  BCF  SWR,CARRY
00B1 05A6  BSF  PORTB,CK
00B2 06C6  BFSC PORTB,BV

© 1993 Microchip Technology Incorporated
Remote Positioner

00B3 0503  BSF  SWR, CARRY  ; SET THE CARRY FOR A 1
00B4 0371  RLF  POSA   ; ROTATE THE W REG LEFT
00B5 04A6  BCF  PORTB, CK  ; SET THE CLOCK LOW
00B6 0000  NOP  ; DELAY

GET_BIT

00B7 0403  BCF  SWR, CARRY  ; BIT 1
00B8 05A6  BSF  PORTB, CK  ; SET CLOCK BIT HIGH
00B9 06C6  BTFSC  PORTB, BV  ; LOOK AT DATA COMMING IN
00BA 0503  BSF  SWR, CARRY  ; SET THE CARRY FOR A 1
00BB 0371  RLF  POSA   ; ROTATE THE W REG LEFT
00BC 04A6  BCF  PORTB, CK  ; SET THE CLOCK LOW
00BD 0000  NOP  ; DELAY

GET_BIT

00BE 0403  BSF  SWR, CARRY  ; BIT 0
00BF 05A6  BSF  PORTB, CK  ; SET CLOCK BIT HIGH
00C0 06C6  BTFSC  PORTB, BV  ; LOOK AT DATA COMMING IN
00C1 0503  BSF  SWR, CARRY  ; SET THE CARRY FOR A 1
00C2 0371  RLF  POSA   ; ROTATE THE W REG LEFT
00C3 04A6  BCF  PORTB, CK  ; SET THE CLOCK LOW
00C4 0000  NOP  ; DELAY
00C5 05E6  BSF  PORTB, CSN  ; DESELECT THE CHIP

;*********************** CALCULATING THE PIO TERMS ***********************

;****CALCULATE THE ERROR********
; The error is very simply the signed difference between where the
; system is and where it is supposed to be at a particular instant
; in time. It is formed by subtracting the actual position from the
; requested position (Position requested - Position actual). This
; difference is then used to determine the proportional, integral and
; differential term contributions to the output.

C_ERR

00C6 0211  MOVF  POSA, 0  ; LOAD THE ACTUAL POSITION INTO W
00C7 0090  SUBWF  POSR, 0  ; SUBTRACT IT FROM THE REQUESTED POSITION
00C8 0603  BTFSC  SWR, CARRY  ; CHECK THE CARRY BIT TO DETERMINE THE SIGN
00C9 0ACB  GOTO  PLS_ER  ; ITS POSITIVE (POSR>POSA)
00CA 0ACE  GOTO  MNS_ER  ; ITS NEGATIVE (POSA>POSR)

PLS_ER

00CB 002C  MOVWF  ERROR  ; SAVE THE DIFFERENCE IN "ERROR"
00CC 0419  BCF  FLAGS, ER_SGN  ; SET THE SIGN FLAG TO INDICATE POSITIVE
00CD 0AD2  GOTO  CE_EXIT

MNS_ER

00CE 0210  MOVF  POSR, 0  ; RE-DO THE SUBTRACTION
00CF 0091  SUBWF  POSA, 0  ; ACTUAL - REQUESTED
00D0 002C  MOVWF  ERROR  ; STORE THE DIFFERENCE IN "ERROR"
00D1 0519  BSF  FLAGS, ER_SGN  ; SET THE SIGN FLAG FOR NEGATIVE

CE_EXIT

00D2 006D  CLRF  SUMLO  ; CLEAN OLD VALUES OUT TO PREPARE
00D3 0078  CLRF  SUMHI  ; FOR THIS CYCLES SUMMATION

;****CALCULATE THE PROPORTIONAL TERM********
; The proportional term is the error times the proportional gain term.
; This term simply gives you more output drive the farther away you are
; from where you want to be (error) * Kp.
; The proportional gain term is a signed term between -100 and 100. The
; more proportional gain you have the lower your system following error
; will be. The higher your proportional gain, the more integral and
differential term gains you will have to add to make the system stable.
; The sum is being carried as a 16 bit signed value.

C_PROP
Remote Positioner

;****CALCULATE THE INTEGRAL TERM******

\text{; The integral term is an accumulation of the error thus far. Its purpose}
\text{; is to allow even a small error to affect a large change. It does this}
\text{; by adding a small number into an accumulator each cycle through the pro}
\text{; Thusly even a small error that exist for a while will build up to a large}
\text{; enough number to effect an output sufficient to move the system. The}
\text{; that this integral accumulator has is modulated by the integral gain term}
\text{; The integral of the error over time is multiplied be KI and the result is}
\text{; contribution to the final summation for determining the output value. This}
\text{; term helps to insure the long-term accuracy of the system is good. A}
\text{; amount is necessary for this purpose but too much will cause oscillations.}
\text{; The integral is bounded in magnitude for two purposes. The first is so}
\text{; it never rolls over and changes sign. The second is that it may saturate}
\text{; long moves forcing an excessively large overshoot to "de-integrate" the}
\text{; accumulated during the first of the move}

\text{C_INT}

;****CALCULATE THE INTEGRAL TERM******

\text{; The integral term is an accumulation of the error thus far. Its purpose}
\text{; is to allow even a small error to affect a large change. It does this}
\text{; by adding a small number into an accumulator each cycle through the pro}
\text{; Thusly even a small error that exist for a while will build up to a large}
\text{; enough number to effect an output sufficient to move the system. The}
\text{; that this integral accumulator has is modulated by the integral gain term}
\text{; The integral of the error over time is multiplied be KI and the result is}
\text{; contribution to the final summation for determining the output value. This}
\text{; term helps to insure the long-term accuracy of the system is good. A}
\text{; amount is necessary for this purpose but too much will cause oscillations.}
\text{; The integral is bounded in magnitude for two purposes. The first is so}
\text{; it never rolls over and changes sign. The second is that it may saturate}
\text{; long moves forcing an excessively large overshoot to "de-integrate" the}
\text{; accumulated during the first of the move}
Remote Positioner

Remote Positioner

BTFSC SWR,CARRY
GOTO ADDINT
; NON-CARRY CONDITION INDICATING A ROLL-OVER
MOV LW 9CH
; IF NOT THEN LEAVE THE ACCUMULATOR ALONE
MOVWF ACCUM
; IF SO THEN LIMIT IT TO -100 BY

ADDINT
MOVF ACCUM,W
; ADD THE INTEGRAL ACCUMULATOR TO
ADDWF SUMHI,1
; THE LOW BYTE OF THE SUM
BTFSC SWR,CARRY
; TEST FOR OVERFLOW, IF SO THEN
INCF SUMHI,1
; INCREMENT THE HI BYTE
MOV LW 0
; LOAD 0 INTO THE W REGISTER
BTFSC ACCUM,7
; IF THE INTEGRAL ACCUMULATOR WAS NEGATIVE
COMF W,W
; COMPLEMENT THE 0 TO GET SIGN FOR HIGH BYTE
ADDWF SUMHI,1
; ADD INTO THE HIGH BYTE OF THE SUM

U_DEXIT
; EXIT POINT FOR THE UP/DOWN CONTROL OF ACCUM

;****CALCULATING THE DIFFERENTIAL TERM*******************************
; The differential term examines the error and determines how much
; it has changed since the last cycle. It does this by subtracting the
; old error from the new error. Since the cycle time is relatively fixed
; we can use it as the “dt” of the desired “de/dt”. This derivative of the
; error is then multiplied by the differential gain term KD and becomes the
; differential term contribution for the final summation.

; First, create the “de” term by doing a signed subtraction of new error
; minus the old error. (new_error - old_error)

C_DIFF
MOVF ERROR,W
; LOAD THE NEW ERROR INTO REGISTER
BTFSS FLAGS,ER_SGN
GOTO LO_BYTE
MOV LW 00
; CORRECT THE VALUE TO BE 16 BIT
BTFSC FLAGS,ER_SGN
GOTO LO_BYTE
MOVF ERR_O,W
; RESTORE IT FOR FUTURE USE TO 8 BIT MAGNITUDE
LOAD THE OLD ERROR INTO OTHER REGISTER
BTFSC FLAGS,OER_SGN
GOTO LO_BYTE
MOV LW 0
; SIGN EXTEND THE UPPER BYTE
BTFSC FLAGS,DE_SGN
GOTO LO_BYTE
MOVWF ERR_O,7
; COUNT THE SIGN OF THE RESULT
BTFSC ERR_O,1
; ITS NEGATIVE SO SET THE FLAG AND
INC ERR_O,1
; COMPLEMENT THE VALUE
BCF DE_SGN,7
; ITS POSITIVE SO SET RESET THE FLAG

STRIP SGN
BTFSS ACChI,7
GOTO NEG ABS
GOTO POS ABS
NEG ABS
BSF FLAGS,DE_SGN
; ITS NEGATIVE SO SET THE FLAG AND
COMF ACChI,1
; COMPLEMENT THE VALUE
INC ACChI,W
MOVWF ERR_O
GOTO MULT_KD
POS ABS
BCF FLAGS,DE_SGN
; ITS POSITIVE SO SET RESET THE FLAG
Remote Positioner

0126 0214 MOVF ACCbLO,W ; AND SAVE THE VALUE
0127 002F MOVWF ERR_O

; Then multiply by Kd

MULT_KD
0128 020F MOVF ERR_O,W
0129 0033 MOVWF mulcnd ; MOVE THE DE/DT TERM INTO THE MULCND REG.
012A 0C20 MOVlw KD ; MOVE THE DIFFERENTIAL GAIN TERM INTO
012B 0034 MOVWF mulplr ; MULPLR TO MULTIPLY THE DE/DT
012C 0901 CALL mpy_S ; DO THE MULTIPLICATION
012D 0910 CALL DIV_LMT ; SCALE AND LIMIT TO 100

RE_SIGN
012E 0759 BTFSS FLAGS,DE_SGN ; IF THE DE SIGN IS NEGATIVE THEN
012F 0B32 GOTO SAVE_DIFF ; PUT THE SIGN INTO THE LOW BYTE
0130 0276 COMF L_byte,W
0131 02B6 INCF L_byte,W

SAVE_DIFF
0132 0216 MOVWF L_byte,W
0133 0643 BTFSC SWR,Z
0134 0045 GOTO ROLL_ER
0135 002F MOVWF ERR_O

; ADD THE DIFF TERM INTO THE SUMM **************

ADDDIF
0136 0C00 MOVLW 00
0137 0659 BTFSC FLAGS,DE_SGN ; PUT THE KD*(DE/DT) TERM INTO THE
0138 00FF MOVLW OFF ; REGISTERS TO ADD.
0139 0036 MOVWF ACCbHI ; SIGN EXTEND THE UPPER BYTE
013A 020F MOVW ERR_O,W
013B 0034 MOVWF ACCbLO
013C 020D MOVF SUMIO,W ; LOAD THE CURRENT SUM INTO THE
013D 0033 MOVWF ACCaLO ; REGISTERS TO ADD
013E 0218 MOVF SUMHI,W
013F 0035 MOVWF ACCaHI
0140 0910 CALL D_add ; ADD IN THE DIFFERENTIAL TERM
0141 0214 MOVF ACCbLO,W ; SAVE THE RESULTS BACK
0142 002D MOVWF SUMLO ; INTO SUMLO AND HI
0143 0216 MOVF ACCbHI,W
0144 0038 MOVWF SUMHI

ROLL_ER
0145 020C MOVF ERROR,W ; TAKE THE CURRENT ERROR
0146 002F MOVWF ERR_O ; AND PUT IT IN THE ERROR HISTORY
0147 0499 BCF FLAGS,ER_SGN ; SAVE THE CURRENT ERROR SIGN
0148 0619 BTFSC FLAGS,ER_SGN ; IN THE OLD ERROR SIGN FOR
0149 0599 BSF FLAGS,ER_SGN ; NEXT TIME THROUGH

;****SET UP THE DIRECTION FOR THE BRIDGE***************

; After the sum of all the components has been made, the sign of the
; sum will determine which way the bridge should be powered.
; If the sum is negative the bridge needs to be set to drive ccw; if the
; sum is positive then the bridge needs to be set to drive cw. This
; is purely a convention and depends upon the polarity the motor and feedback
; element are hooked up in.

SET_DIR
014A 0479 BCF FLAGS,DIR ; SET FOR DEFAULT CLOCKWISE
014B 0669 BTFSC SUMHI,7 ; LOOK AT THE SIGN BIT, IF IT IS SET
014C 0579 BSF FLAGS,DIR ; THEN SET FOR CCW BRIDGE DRIVE

;**** SCALE THE NUMBER TO BETWEEN 0 AND 100% ***************

; After the direction is set the request for duty cycle is limited to between
; 0 and 100 percent inclusive. This value is passed to the dutycycle setting
Remote Positioner

; routine by loading it in the variable "PCNT".

L_SUMM
014D 07F8  BTFSS  SUMHI,7  ; CHECK TO SEE IF IT IS NEGATIVE
014E 0B52  GOTO  POS_1M
014F 0278  COMF  SUMHI,1
0150 026D  COMF  SUMLO,1
0151 02AD  INCF  SUMLO,1

POS_1M
0152 0C01  MOVLW 1H  ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0153 0098  SUBWF  SUMHI,0  ; IF THERE IS ANYTHING THERE, IF NOT,
0154 0703  BTFSS  SWR,CARRY  ; THEN LEAVE THE LOW BYTE ALONE
0155 0B59  GOTO  LB_L  ; OTHERWISE GIVE THE LOW BYTE A FULL
0156 0C64  MOVLW 64H  ; COUNT AND IT WILL HAVE BEEN LIMITED
0157 002D  MOVWF  SUMLO  ; TO 100
0158 0B5F  GOTO  LP_EXIT  ; GOTO LIMIT PERCENT EXIT

LB_L
0159 0C64  MOVLW 64H  ; LIMIT THE MAGNITUDE OF THE VALUE TO
015A 008D  SUBWF  SUMLO,0  ; 100 DECIMAL
015B 0703  BTFSS  SWR,CARRY
015C 0B5F  GOTO  LP_EXIT
015D 0C64  MOVLW 64H
015E 002D  MOVWF  SUMLO

LP_EXIT
015F 020D  MOVF  SUMLO,W  ; STORE THE LIMITED VALUE IN
0160 0029  MOVWF  PCNT  ; THE PERCENT DUTYCYCLE REQUEST

;**********************************************************
; PWM GENERATING ROUTINE
;
; The important thing here is not to have to do too many decisions or
; calculations while you are generating the 100 or so
; pulses. These will
; take time and limit the minimum or maximum duty cycle.

WHICH_DIR
0161 0679  BTFSC  FLAGS,DIR  ; CHECK THE DIRECTION FLAG
0162 0B76  GOTO  GOCW  ; DO CCW PULSES FOR 1
0163 0B64  GOTO  GOCW  ; DO CW PULSES FOR 0

GOCW
0164 0426  BCF  PORTB,PWMCW  ; SET THE BRIDGE FOR CW MOVE
0165 0C64  MOVWF  CYCLES  ; SET UP CYCLES COUNTER FOR 100 PULSES
0166 0032  MOVF  CWHI  ; CALL CALCTIMES
0167 0943  CALL  CALCTIMES  ; CALCULATE THE HI AND LO TIMES

RLDCW
0168 0207  MOVF  HI,0  ; RE LOAD THE HI TIMER
0169 0202A  MOVWF  HI_T  ; WITH THE CALCULATED TIME
016A 0208  MOVF  LO,0  ; RE LOAD THE LO TIMER
016B 0202B  MOVWF  LO_T  ; WITH THE CALCULATED TIME
016C 0004  CLRWD  ; TAG THE WATCHDOG TIMER

CWHI
016D 0506  BSF  PORTB,PWMCW  ; SET THE CLOCKWISE PWMBIT HIGH
016E 028A  DECFSZ  HI_T,1  ; DECREMENT THE HI USEC. COUNTER
016F 0B6D  GOTO  CWHI  ; DO ANOTHER LOOP

CWLO
0170 0406  BCF  PORTB,PWMCW  ; SET THE CLOCKWISE PWMBIT LOW
0171 025B  DECFSZ  LO_T,1  ; DECREMENT THE LO USEC. COUNTER
0172 0B70  GOTO  CWLO  ; DO ANOTHER LOOP
0173 02FF  DECFSZ  CYCLES,1  ; DECREMENT THE NUMBER OF CYCLES LEFT
0174 0B68  GOTO  RLDCW  ; DO ANOTHER PULSE
0175 0A50  GOTO  BEGIN  ; DO ANOTHER MAIN SYSTEM CYCLE
GOCCW
0176 0406 BCF PORTB,PWMCCW ; SET THE BRIDGE FOR CCW MOVE
0177 0C64 MOV LW 64H
0178 0032 MOVWF CYCLES ; SET UP CYCLE COUNTER FOR 100 PULSES
0179 0943 CALL CALCTIMES ; CALCULATE THE HI AND LO TIMES

RLDCW
017A 0207 MOVF HI,0 ; RE LOAD THE HI TIMER
017B 002A MOVF LO,0 ; RE LOAD THE LO TIMER
017C 0208 MOVF LO_T,0 ; WITH THE CALCULATED TIME
017D 002B MOVF HI_T,0 ; WITH THE CALCULATED TIME
017E 0004 CLRWD ; TAG THE WATCHDOG

CCWHI
017F 0526 BSF PORTB,PWMCCW ; SET THE COUNTERCLOCKWISE PWM BIT HIGH
0180 02EA DECFSZ HI_T,1 ; DECREMENT THE HI USEC. COUNTER
0181 0B7F GOTO CCWHI ; DO ANOTHER LOOP

CCWLO
0182 0426 BCF PORTB,PWMCCW ; SET THE COUNTERCLOCKWISE PWM BIT LOW
0183 0426 DECFSZ LO_T,1 ; DECREMENT THE LO USEC. COUNTER
0184 0B82 GOTO CCWLO ; DO ANOTHER LOOP
0185 02E2 DECFSZ CYCLES,1 ; DECREMENT THE NUMBER OF CYCLES LEFT
0186 087A GOTO RLDCW ; DO ANOTHER PULSE
0187 0A50 GOTO BEGIN ; DO ANOTHER MAIN SYSTEM CYCLE

;************* START VECTOR **********************
CLRREG ;INITIALIZE REGISTERS
0188 0COB MOVLW 0B H ; SET PORT A FOR 3 INPUTS AND
0189 0005 TRIS PORTA ; AN OUTPUT
018A 0C1C MOVLW 1CH ; SET PORT B FOR INPUTS AND OUTPUTS
018B 0006 TRIS PORTB ; THIS SETTING FOR SENDING TO A/D
018C 0040 CLR W ; CLEAR THE W REGISTER
018D 0002 OPTION ; STORE THE W REG IN THE OPTION REG
018E 0C08 MOVLW 0BH ; STARTING REGISTER TO ZERO
018F 0024 MOVF FSR ;

GCLR
0190 0060 CLR F ;
0191 03B4 INCFSZ FSR ; SKIP AFTER ALL REGISTERS
0192 0B90 GOTO GCLR ; HAVE BEEN INITIALIZED
0193 0A50 GOTO BEGIN ; START AT THE BEGINING OF THE PROGRAM

01FE 0888 ORG 01FF ; START VECTOR
01FF 0888 GOTO CLRREG ; START VECTOR

Errors : 0
Warnings : 0
INTRODUCTION

UNISITE, from Data I/O Corporation, is a universal programming system that supports a wide variety of programmable IC on the market, including Microchip’s PIC16C5X series of EPROM-based single-chip microcontrollers.

The UNISITE is available in various versions and options for:
- single unit programming
- gang/set programming
- IC-handler interfacing
- support for devices in surface mountable packages

This application note describes three ways of programming PIC16C5X devices on this programmer for prototyping during development and production programming in medium and high volumes. Only PIC16C5X specific topics will be covered. For more information on using the UNISITE in general, please refer to Data I/O’s literature.

REQUIRED EQUIPMENT OPTIONS

The basic version of the UNISITE with 28-pin support (SITE 48") and software/firmware release V3.0 is the base requirement to program PIC16C5X devices (see Note 1). In addition, a “dumb” terminal with RS232 interface, or a personal computer with Data I/O’s proprietary “PROMLINK” software, any standard communication software package (e.g., PROCOMM") is required to control UNISITE and/or to download object files.

Note 1: Software release V2.8 supports the PIC16C5X series except for the code protect function. Do not use this function with V2.8 as the devices may not be functional in the application afterwards.

Note 2: The “SETSITE” option does not support microcontrollers.

Note 3: The “CHIPSITE” option is not supported by the software release V3.0 regarding PIC16C5X programming. To program PIC16C5Xs in surface mount packages (such as PLCC and SOIC) use adapter sockets that plug into a DIP socket.

THREE WAYS TO PROGRAM PIC16C5X DEVICES ON THE UNISITE

Duplication of a “Master” Device

This is the easiest way to program PIC16C5Xs on the UNISITE. Only a “dumb” terminal is required and an already programmed “Master” PIC16C5X. The “Master” can be generated by either using Microchip’s PICPRO programmer or by using the method as described in the section titled “Prototype Programming” of this application note.

Step 1: Select the appropriate PIC16C5X device from UNISITE software menu.

Step 2: Load the “Master” PIC16C5X into the UNISITE’s RAM using the corresponding menu function. All program memory locations of the PIC16C5X, including the configuration EPROM (watchdog timer disable fuse and oscillator selection) and the customer ID bits will be loaded and stored. Note that the “Master” PIC16C5X must not be code protected.

Step 3: Remove the “Master” PIC16C5X and insert a blank PIC into the programming socket.

When programming OTP devices, make sure that the oscillator type of the OTP part matches the oscillator selection of your “Master” PIC16C5X as the oscillator type of an OTP part cannot be changed by the programmer.

When programming windowed devices, the oscillator selection of the “Master” PIC16C5X will be duplicated as well.

Step 4: Activate the “Program” menu function of the UNISITE.

A. If no code protection is desired:

Hit the <CR> key.

The UNISITE will produce a 1:1 copy of the “Master”, including customer ID code and configuration EPROM. After successful programming a checksum is being displayed in the message line. Note that this checksum is different from the one generated by Microchip’s PICPRO due to different ways of computation.

B. If code protection is desired:

Select “security fuse data” to be “1” and enable “security fuse option” (“Y”).

Then hit <CR>. 

© 1993 Microchip Technology Inc.

DS00524B-page 1

2-149
Programming PIC16C5X on Data I/O's Unisite

The "Master" unit will be copied 1:1 into the blank device, will be verified, and then the code protection fuse will be blown. After successful programming a checksum is being displayed in the message line.

Note that this checksum is generated before the code protection has been activated. Thus, it is the same as in Option A. Steps 3 and 4 can be executed as often as required to produce any amount of "duplicates".

As the programming time for PIC16C54 or PIC16C55, respectively, is less than five seconds, this method is very effective and fast for any kind of production programming of PIC16C5X devices.

An alternative way to duplicate a PIC16C5X device, if no code protection is desired, is given with the "Quick Copy" command of the UNISITE.

**PROTOTYPE PROGRAMMING**

This method allows the generation of a prototype or "Master" PIC16C5X during the product development cycle. A personal computer is required to:

A. Generate a formatted PIC16C5X object file using MPALC and,
B. Download the contents of this file to the UNISITE.

Besides the normal object code, the configuration EPROM of the PIC16C5X device and, optionally, the customer ID bits have to be specified before programming a PIC16C5X device on the UNISITE.

**TABLE 1 - PIC16C5X ID CODE**

<table>
<thead>
<tr>
<th>Device</th>
<th>&quot;N&quot;</th>
<th>Block Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C54</td>
<td>200 (hex)</td>
<td>40A (hex)</td>
</tr>
<tr>
<td>PIC16C55</td>
<td>200 (hex)</td>
<td>40A (hex)</td>
</tr>
<tr>
<td>PIC16C56</td>
<td>400 (hex)</td>
<td>80A (hex)</td>
</tr>
<tr>
<td>PIC16C57</td>
<td>800 (hex)</td>
<td>100A (hex)</td>
</tr>
</tbody>
</table>

Note: The block size in UNISITE is the total number of bytes. For PIC16C54, for example, total number of words = 1FFh + 4 (ID locations) + 1 (configuration word) = 205h. The total number of bytes, is therefore, 40Ah.

**TABLE 2 - PIC16C5X CONFIGURATION EPROM FUSES**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td></td>
<td>WDT Enabled (Default)</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td></td>
<td>WDT Disabled</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td></td>
<td>LP Oscillator</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
<td>XT Oscillator</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
<td>HS Oscillator</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td></td>
<td>RC Oscillator</td>
</tr>
</tbody>
</table>

Example:

```
ORG 200H ;PIC16C54 or PIC16C55
DATA 0,0A,3,0F ;ID code = 0A3F (hex)
DATA B'001' ;WDT disabled, XT oscillator
```

Step 4: Re-assemble the modified source code with MPALC, specifying the "merged hex" output file format (INHXBM) in the command line.

Example: MPALC -F INHXBM <filename>

Step 5: Establish communication with UNISITE and select the desired PIC16C5X device from the UNISITE menu.

Step 6: Select the file transfer menu of the UNISITE software. Specify data translation format as "83". (Note that the file size and addresses relate to byte counts - not 12-bit words; one PIC16C5X location corresponds to two bytes.) You may want to "fill" the memory with "FFh" before downloading.

© 1993 Microchip Technology Inc.
Step 7: Download the object file into the UNISITE RAM using the ASCII file transfer option of the serial communication software or the corresponding PROMLINK function.

Step 8: Select “program” menu. Handle the code protect option the same as in Step 4 of the section titled Duplication of a “Master” Device.

Alternative to Define Customer ID code and Configuration EPROM

If you are familiar with the UNISITE, you may skip Step 3 and edit the customer ID code and configuration EPROM directly in the UNISITE RAM in Step 6.

When doing so, you have to reduce the file size and address limits of the UNISITE default values (Step 6) by ten before downloading the object file (Step 7).

After downloading, edit the data of the customer ID location (see Table 1) one hex digit per location and the location immediately after ID locations per Table 2 (Step 7A). Note that you have to convert the binary CONFIG value to a hex value first. The most significant bits are “don’t cares” in both cases.

Then continue with Step 8.

High Volume Programming with IC-Handler

For the fastest and most efficient programming of PIC16C5X devices in high volumes, an IC-handler should be employed. The UNISITE provides with its “HANDLERSITE” option, support for this application.

Data I/O and the manufacturer of the IC-handler should be contacted for further setup information.

This is the only recommended way for high volume programming the PIC16C5X devices in surface mountable packages (e.g., SOIC packages) at the customer’s facilities.

The object data can be loaded into the UNISITE’s RAM with any of the methods described above.

Author: Sumit Mitra
Logic Products Division
INTRODUCTION

The ALLPRO, from Logical Devices, is a universal programming system supporting a wide range of programmable devices, including Microchip’s PIC16C5X series of EPROM based single-chip microcontrollers.

This application note describes two ways of programming PIC16C5X devices on this programmer for prototyping during development and production programming. Only PIC16C5X specific topics will be covered. For more information on using the ALLPRO in general, please refer to Logical Devices’ literature.

REQUIRED EQUIPMENT OPTIONS

The basic version of the ALLPRO with 40-pin support and software/firmware release V1.49C is sufficient to program PIC16C5X devices. In addition, a personal computer with ALLPRO’s proprietary software and interface card is required to control the ALLPRO and to download object files.

TWO WAYS TO PROGRAM PIC16C5X DEVICES ON THE ALLPRO

Duplication of a “Master” Device

This is the easiest way to program PICs on the ALLPRO. Only an already programmed “Master” PIC16C5X is required. The “Master” can be generated by either using Microchip’s PICPRO programmer or by using the method as described in the section titled “Prototype Programming” of this application note.

Step 1: Run the ALLPRO program on the PC and select the “DEVICE SELECT” menu. Type “MICMIC” where the first MIC is for microcontrollers and the second MIC is for Microchip. Then select the appropriate device.

Step 2: Place the “Master” device into the ALLPRO socket and select “LOAD DEVICE” from the menu to load the “Master” PIC16C5X into the ALLPRO’s RAM. All program memory locations of the PIC, including the configuration EPROM (watchdog timer disable fuse and oscillator selection) and the customer ID bits will be loaded and stored. Note that the “Master” PIC16C5X must not be code protected.

Step 3: Remove the “Master” PIC16C5X and insert a blank PIC16C5X into the programming socket. When programming OTP devices, make sure that the oscillator type of the OTP part matches the oscillator selection of your “Master” PIC16C5X as the oscillator type of an OTP part cannot be changed by the programmer.

When programming windowed devices, the oscillator selection of the “Master” PIC16C5X will be duplicated as well.

Step 4: Select the “PROGRAM DEVICE” menu function of the ALLPRO and hit enter.

The ALLPRO will produce a 1:1 copy of the “Master”, including customer ID code and configuration EPROM. After successful programming a checksum will be displayed in the message line. Note that this checksum is different from the one generated by Microchip’s PICPRO due to different ways of computation.

If code protection is desired, select “SECURE DEVICE” on the menu, then hit enter. This will blow the code protection fuse.

Steps 3 and 4 can be executed as often as required to produce any amount of “duplicates”. The “Master” device only needs to be downloaded the first time.

As the programming time for a PIC16C54 or PIC16C55 is less than ten seconds, this method is very effective and fast for any kind of production programming of PIC16C5X devices.

Prototype Programming

This method allows the generation of a prototype or “Master” PIC16C5X during the product development cycle. A personal computer is required to:

A. generate a formatted PIC16C5X object file using MPALC and,

B. download the contents of this file to the ALLPRO.

Besides the normal object code, the configuration EPROM of the PIC16C5X device and, optionally, the customer ID bits have to be specified before programming a PIC16C5X device on the ALLPRO.

Step 1: Write the application source code with the processor selection of your choice specified in the LIST assembler directive.

Example: LIST P=16C55

Step 2: Assemble your source code and debug with PICMASTER or MPSIM as required.
Programming PIC16C5X on Logical Devices' ALLPRO

Step 3: Modify your debugged source file:

A. Change the processor selection in the LIST directive to:

P=16C62

This enables MPALC to accept addresses above the normal program memory space.

B. Define customer ID code by adding before the END statement:

Example:

ORG N ;see Table 1 for value of N
DATA ID3,ID2,ID1,ID0 ;four hex digits representing the ID code
ORG N+40
DATA CONFIG ;last three bits = config - see Table 2

<table>
<thead>
<tr>
<th>Device</th>
<th>&quot;N&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C54</td>
<td>200 (hex)</td>
</tr>
<tr>
<td>PIC16C55</td>
<td>200 (hex)</td>
</tr>
<tr>
<td>PIC16C56</td>
<td>400 (hex)</td>
</tr>
<tr>
<td>PIC16C57</td>
<td>800 (hex)</td>
</tr>
</tbody>
</table>

TABLE 2 - PIC16C5X CONFIGURATION EPROM FUSES

<table>
<thead>
<tr>
<th>Configuration*</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 2</td>
<td>Bit 1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

* Bits 3-7 must always be 1.

Example:

ORG 200(HEX) ;PIC16C54 or PIC16C55
DATA 0,0A,3,0F ;ID code = 0A3F (hex)
ORG 240
DATA 0F9 ;WDT disabled, XT oscillator

Step 4: Re-assemble the modified source code with MPALC, specifying the "merged hex" output file format (INHX8M) in the command line

Example: MPALC-F INHX8M <filename>

or the LIST directive of your source file.

Example: LIST P=16C62, F=INHX8M

Step 5: Upload the data file into the ALLPRO RAM using the "READ DATA FILE" menu command..

Step 6: Select "PROGRAM DEVICE" menu and handle code protection option as described in Step 4 from the section titled "Duplication of Master Device".

Alternative Method to Load ID Code and Configuration EPROM

You may skip Step 3 and edit the customer ID code and configuration EPROM directly in the ALLPRO RAM.

After downloading the file (without ID and configuration data) to the ALLPRO's RAM, use the "EDIT DATA" menu command to view the ALLPRO RAM. Push F8 to see and edit the ID locations. The ID can be entered directly on line 0. For example, the ID code ABCD would be entered as:

0 - 0a 00 0b 00 0c 00 0d 00...

where the first zero is the line number. Note that the INHX8M format has LSB first and MSB second so the code 0a 00 is actually 000a (the first ID code).

To edit the configuration EPROM push F8 again. Enter the configuration data into the data location per Table 2 above. The example (WDT disabled, crystal oscillator) would be:

0 - f9 0f

Note that the MSB is a "don't care" and will show a different value when data is loaded from the PC (00) than from a "master" device (0f).

Author: Sumit Mitra
Logic Products Division
INTRODUCTION

This application note provides some utility math routines for Microchip's PIC16C5X series of 8-bit microcontrollers. The following math routines are provided:

- 8 x 8 unsigned multiply
- 16 x 16 double precision multiply
- 16 x 16 double precision divide
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- floating point multiplication
- floating point addition
- floating point subtraction
- BCD to binary conversion routines
- binary to BCD conversion routines
- BCD addition
- BCD subtraction
- square root

These are written in native assembly language and the listing files are provided. They are also available on a disk (MS-DOS®). All the routines provided can be called as subroutines. Most of the routines have two different versions: one optimized for speed and the other optimized for code size. The calling sequence of each routine is explained at the beginning of each listing file.

SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices A and B. The performance specs for the routines are shown in Table 1.

![Flow Chart for Unsigned 8 x 8 Multiply]

TABLE 1 · PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Efficient</td>
<td>35</td>
<td>37</td>
</tr>
<tr>
<td>Code Efficient</td>
<td>16</td>
<td>71</td>
</tr>
</tbody>
</table>
DOUBLE PRECISION MULTIPLICATION

This routine computes the product of two 16-bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic are supported. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices C and D. The performance specs for the routines are shown in Table 2.

TABLE 2 - PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Efficient</td>
<td>240</td>
<td>233</td>
</tr>
<tr>
<td>Code Efficient</td>
<td>33</td>
<td>333</td>
</tr>
</tbody>
</table>

The code in Appendices C and D has been setup for unsigned arithmetic and the performance specs in the table above is for unsigned arithmetic. If signed arithmetic is desired, edit the line with "Signed equ FALSE" to "Signed equ TRUE" then re-assemble the code.

In case of signed multiply, both operands are assumed to be 16-bit 2's complement numbers.

Conditional assembly is supported by MPALC (PIC16C5Xcross assembler) V2.1 or higher. If you have an older version, please contact the Microchip Technology sales office nearest you.

DOUBLE PRECISION DIVISION

This routine divides two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic are supported. Two routines are provided. One routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices E and F. The performance specs for the routines are shown in Table 3.

TABLE 3 - PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Efficient</td>
<td>370</td>
<td>263</td>
</tr>
<tr>
<td>Code Efficient</td>
<td>37</td>
<td>310</td>
</tr>
</tbody>
</table>

As can be seen from the above table, writing a speed efficient (straight line code) for this routine does not produce an efficient code. In this case, the looped code has a better overall performance (code size versus the speed). The code in Appendices E and F has been setup for unsigned arithmetic and the performance specs in the table above is for unsigned arithmetic. If signed arithmetic is desired, edit the line with "Signed equ FALSE" to "Signed equ TRUE" and then re-assemble the code. Signed division assumes two 16-bit 2's complement numbers.

DOUBLE PRECISION ADDITION & SUBTRACTION

This routine adds or subtracts two 16-bit numbers and produces a 16-bit result. This routine is used by other double precision routines. The listing of these routines is given in Appendix G. The performance specs for the routines are shown below:

TABLE 4 - PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Subtraction</td>
<td>14</td>
<td>17</td>
</tr>
</tbody>
</table>

FLOATING POINT ROUTINES

Four routines are implemented: addition, subtraction, multiplication and division. A floating point number is implemented as a number with 16-bit signed Mantissa and an 8-bit signed binary exponent, i.e., a number "Num" is represented as:

Num = \( <\pm 16\text{-bit Mantissa}> \times 2^{<\pm 8\text{-bit Exponent}>} \)

Also, a general purpose normalization routine is provided and it is recommended that the user call this routine as often as possible to avoid loss of precision. The normalization routine is written to normalize the value in locations "ACCbHI and ACCbLO". With minor modifications, the user may change this routine to normalize any two consecutive File registers (16 bits) by using indirect addressing scheme.

In case of multiplication, if a 32-bit result (with 8-bit exponent) is desired, then the user should edit the line "Mode16 equ TRUE" and change to "Mode16 equ FALSE".

No attempt has been made to optimize the code for speed. Only a looped version of the code is provided (see Appendix H). Also, only the code size (memory requirement) is shown in the table below and not the execution time because, the execution time varies significantly depending upon the value of the operands (and how they are normalized).

TABLE 5 - PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Subtraction</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>102</td>
<td></td>
</tr>
</tbody>
</table>

The flow charts of the algorithm for addition/subtraction and multiplication are shown in Figures 2, 3 and 4, respectively.

Please contact Microchip office for any code update.
**FIGURE 2 - FLOW CHART FOR FLOATING POINT ADDITION AND SUBTRACTION**

**F_SUB**
- Subtraction Routine
- Obtain 2's Complement of ACCA (Call Nega)

**F_ADD**
- Addition Routine
- Subtract Exponent A from Exponent B
  - If \( A = B \) then
    - Subtract A from B, B Register has the Difference
  - If \( A < B \) then
    - Swap Them (Call FSWAP)

- Shift ACCB Right Preserving the Sign, Increment B
- B Register has the Difference

- Check & Correct for Overflow
- Return

**FIGURE 3 - FLOW CHART FOR FLOATING POINT MULTIPLICATION**

**F_MPY**
- Multiplication Routine
- Determine Sign of Final Answer, Obtains 2's Complement of -VE #s

- Move ACCB to ACCD, Clear ACCB, Initiate Temp to 16

- Shift ACCD Right (One Bit)
  - Add Multiplicand to the Partial Product (Call MADD) in ACCB

- Rotate ACCB Right (One Bit)
- Decrement Temp Register
- Temp = 0?

- Add the Exponents \( A + B \rightarrow B \)
  - Sign of Answer + or -?

- Obtains 2's Complement of ACCB
  - Normalize ACCB to Maintain Precision
- Return
FIGURE 4 - FLOW CHART FOR FLOATING POINT DIVISION

DIV Division Routine

- Determine Sign of Final Answer. Obtain 2's Complement of -VE #s
- Move ACCB to ACCD. Clear ACCB, ACCC. Initiate Temp to 16
- Shift ACCD Left (One Bit) into ACCC. Check if ACCA is >, =, or < ACCC

ACCA ≤ ACCC ? Y

- Subtract ACCA from ACCC
- Subtract ACCA from ACCC

N

- Temp = 0

- Subtract Exponents

- Adjust ACCB if Final Answer was to be -VE

Return

BCD TO BINARY CONVERSION

This routine converts a five digit BCD number to a 16-bit binary number. The listing of this routine is given in Appendix I. The performance spec for the routine is shown below:

TABLE 6 - PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCD to Binary</td>
<td>30</td>
<td>121</td>
</tr>
</tbody>
</table>

FIGURE 5 - FLOW CHART FOR BINARY TO BCD CONVERSION

Bin to BCD

- Count = 16
- R0 = 0, R1 = 0
- R2 = 0

Binary to BCD Conversion

- In: BCD # in R0, R1, R2
- Out: Binary # in S0, S1

Shift S0, S1 Left into R0, R1, R2 (One Bit)

- Count = 0 ?

Y

- Adjust BCD
- Adjust R2

N

- Adjust BCD
- Adjust R1

FSR = 2 Digit BCD #

LSD + 3 > 7 ?

Y

- LSD = LSD + 3

N

MSD + 3 > 7 ?

Y

- MSD = MSD + 3

N

Return

TABLE 7 - PERFORMANCE SPECS

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary (8-Bit) to BCD</td>
<td>10</td>
<td>81 (Worst Case)</td>
</tr>
<tr>
<td>Binary (16-Bit) to BCD</td>
<td>30</td>
<td>719</td>
</tr>
</tbody>
</table>
BCD ADDITION & SUBTRACTION

These two routines perform a two digit unsigned BCD addition and subtraction. The results are the sum (or difference) in one File register and with a overflow carry-bit in another File register. The performance specs for the routines are shown below:

**TABLE 8 - PERFORMANCE SPECS**

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCD Addition</td>
<td>29</td>
<td>23 (Worst Case)</td>
</tr>
<tr>
<td>BCD Subtraction</td>
<td>31</td>
<td>21 (Worst Case)</td>
</tr>
</tbody>
</table>

The listing files for the above two routines are given in Appendices L and M. The flow charts for BCD addition and BCD subtraction are given in Figures 6 and 7, respectively.

**FIGURE 6 - FLOW CHART FOR BCD ADDITION**

```
U BCD AD
Perform Binary Addition

DC = 1 ?
Y
N
Least Sig Dig > 9
N
Add 6 to LSD
CY = 1 ?
Y
N
Most Sig Dig > 9
N
Add 6 to MSD
Return
```

**SQUARE ROOT**

Often in many applications one needs to find the square root of a number. Of many numerical methods to find the square root of a number, the Newton-Raphson method is very attractive because of its fast convergence rate. In this method the square root of a number, "N", is obtained from the approximate solution of:

\[ f(Y) = Y^2 - N = 0 \]

The function "f(Y)" can be expanded about \( Y_0 \) using first order Taylor polynomial expansion as:

**Equation 1:** \[ f(Y) = f(Y_0) + (Y - Y_0) f'(Y_0) + \frac{(Y - Y_0)^2}{2} f''(Y_0) + \cdots \]

If \( X \) is a root of \( f(Y) \), then \( f(X) = 0 \):

\[ f(X) = f(Y_0) + (X - Y_0) f'(Y_0) + \frac{(X - Y_0)^2}{2} f''(Y_0) + \cdots = 0 \]

If \( Y_0 \) is an approximate root of \( f(Y) \), then higher order terms are negligible. Therefore:

**Equation 2:** \[ f(Y_0) + (X - Y_0) f'(Y_0) \]

i.e., \[ X = Y_0 - \frac{f(Y_0)}{f'(Y_0)} \]

Thus, \( X \) is a better approximation for \( Y_0 \). From this, the sequence \( \{X_n\} \) can be generated:

**Equation 3:** \[ X_n = X_{n-1} - \frac{f(X_{n-1})}{f'(X_{n-1})}, n \geq 1 \]

From equation 1 and equation 3 we get,

**Equation 4:** \[ X_n = 0.5 \cdot \{X_{n-1} + N/X_{n-1}\} \]

An assembly language subroutine implementing the above algorithm is given in Figure 1.8. The initial approximate root of \( N \) is taken to be \( N/2 \). If the approximate range of \( N \) is known a priori, then the total number of iterations may be cut down by starting with a better approximate root than \( N/2 \).

This program, as listed in Appendix N, computes the square root of a 16-bit number. This routine uses a double precision math routine (division and addition) as described in the previous pages of this application note.
APPENDIX A
MPASM B0.54

*******************************************************************
8x8 Software Multiplier
(Code Efficient : Looped Code)
*******************************************************************

The 16 bit result is stored in 2 bytes

Before calling the subroutine "mpy", the multiplier should be loaded in location "mulplr", and the multiplicand in "mulcnd". The 16 bit result is stored in locations H_byte & L_byte.

Performance:
Program Memory: 15 locations
# of cycles: 71
Scratch RAM: 0 locations

This routine is optimized for code efficiency (looped code)
For time efficiency code refer to "mult8x8F.asm" (straight line code)

*******************************************************************
LIST P=16C54

mulcnd equ 09 ; 8 bit multiplicand
mulplr equ 10 ; 8 bit multiplier
H_byte equ 12 ; High byte of the 16 bit result
L_byte equ 13 ; Low byte of the 16 bit result
count equ 14 ; loop counter

include "mpreg.h"

;*******************************************************************
PIC16C5X Header ***********************************************

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

RTCC equ 1h
PC equ 2h
STATUS equ 3h ; F3 Reg is STATUS Reg.
FSR equ 4h

Port_A equ 5h
Port_B equ 6h ; I/O Port Assignments
Port_C equ 7h

;*******************************************************************
; STATUS REG. Bits
CARRY equ 0h ; Carry Bit is Bit.0 of F3
C equ 0h
DCARRY equ 1h
DC equ 1h
Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
Z equ 2h
P_DOWN equ 3h
PD equ 3h
T_OUT equ 4h
TG equ 4h
PA0 equ 5h
PA1 equ 6h
PA2 equ 7h

© 1993 Microchip Technology Inc.
; PIC16C5X Math Routines

; Same equ 1h
; LSB equ 0h
; MSB equ 7h
; TRUE equ 1h
; YES equ 1h
; FALSE equ 0h
; NO equ 0h
;
; ;********************************************************************
; Begin Multiplier Routine

0000 0072
0001 0073
0002 0C08
0003 0034
0004 0209
0005 0403
0006 0330
0007 0603
0008 01F2
0009 0332
000A 0333
000B 02F4
000C 0A06
;
000D 0800
;
000E 0CFF
000F 0030
0010 0CFF
0011 0029
;
0012 0900
;
0013 0A13
;
01FF 0A0E
;

Errors : 0
Warnings : 0
8x8 Software Multiplier
(Fast Version: Straight Line Code)

The 16 bit result is stored in 2 bytes.
Before calling the subroutine "mpy", the multiplier should be loaded in location "mulplr", and the multiplicand in "mulcnd". The 16 bit result is stored in locations H_byte & L_byte.

Performance:
Program Memory: 35 locations
# of cycles: 37
Scratch RAM: 0 locations

This routine is optimized for speed efficiency (straight line code).
For code efficiency, refer to "mult8x8S.asm" (looped code).

LIST

P=16C54
mu lend equ 09 ; 8 bit multiplicand
mulplr equ 10 ; 8 bit multiplier
H_byte equ 12 ; High byte of the 16 bit result
L_byte equ 13 ; Low byte of the 16 bit result

include "picreg.h"

P16C5X Header

PIF54 equ 1FFH ; Define Reset Vectors
PIF55 equ 1FFH
PIF56 equ 3FFH
PIF57 equ 7FFH

RTCC equ 1h
PC equ 2h
STATUS equ 3h ; F3 Reg is STATUS Reg.
FSR equ 4h

Port_A equ 5h
Port_B equ 6h ; I/O Port Assignments
Port_C equ 7h

CARRY equ 0h ; Carry Bit is Bit.0 of F3
C equ 0h
DCARRY equ 1h
DC equ 1h
Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
Z equ 2h
P_DOWN equ 3h
PD equ 3h
T_OUT equ 4h
T0 equ 4h
PA0 equ 5h
PA1 equ 6h
PA2 equ 7h

Same equ 1h

© 1993 Microchip Technology Inc.
PIC16C5X Math Routines

0000 0007
  LSB  equ  0h
  MSB  equ  7h

0000 TRUE  equ  1h
0001 YES  equ  1h
0000 FALSE equ  0h
0000 NO equ  0h

;*************************************************************************

; Define a macro for adding & right shifting

mult MACRO bit
    btfsc mulplr,bit
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same
ENDM

;*************************************************************************

BEGIN Multiplier Routine

mpy_F
    clrf  H_byte
    clrf  L_byte
    movf  mulcnd, w
    bcf   STATUS, CARRY

mult 0
    btfsc mulplr, 0
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

mult 1
    btfsc mulplr, 1
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

mult 2
    btfsc mulplr, 2
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

mult 3
    btfsc mulplr, 3
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

mult 4
    btfsc mulplr, 4
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

mult 5
    btfsc mulplr, 5
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

mult 6
    btfsc mulplr, 6
    addwf H_byte,Same
    rrf  H_byte,Same
    rrf  L_byte,Same

END
mult 7
btfsc mulplr,7
addwf H_byte,Same
rrf H_byte,Same
rrf L_byte,Same

retlw 0

;********************************************************************
;Test Program
;********************************************************************
main movlw OFF ; multiplier (in mulplr) = OFF
movwf mulplr
movlw OFF ; multiplicand (in mulcnd ) = 0FF
movwf mulcnd

; call mpy_F ; The result 0FF*0FF = FEOl is in locations
; H_byte & L_byte

self goto self
org 01FF
self goto main

END

Errors : 0
Warnings : 0
APPENDIX C: DOUBLE PRECISION MULTIPLICATION LISTING (LOOPED)

**PIC16C5X Math Routines**

**MPASM B0.54**

PAGE 1

```assembly
:"************************************************************
:  Double Precision Multiplication
:  ( Optimized for Code Size : Looped Code )
:  "************************************************************;
:        Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
:        (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
:        (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
:        (c) CALL D__mpy
:        (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
:  Performance :
:  Program Memory :  033
:  Clock Cycles :  333
:  Note : The above timing is the worst case timing, when the
:  register ACCb = FFFF. The speed may be improved if
:  the register ACCb contains a number ( out of the two
:  numbers ) with less number of 1s.
:  The performance specs are for Unsigned arithmetic ( i.e,
:  with "SIGNED equ FALSE ").
:  The performance specs are for Unsigned arithmetic ( i.e,
:  with "SIGNED equ FALSE ").
:  "************************************************************;
:LIST P=16C54

0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 ACCbLO equ 12
0013 ACCbHI equ 13
0014 ACCcLO equ 14
0015 ACCcHI equ 15
0016 ACCdLO equ 16
0017 ACCdHI equ 17
0018 temp equ 18
0019 sign equ 19

; include "picreg.h"

;************************************************************ PIC16C5X Header ************************************************************
01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

; 0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h

; 0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

;************************************************************ ;
: ; STATUS REG. Bits
: 0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
```

© 1993 Microchip Technology Inc.
PIC16C5X Math Routines

0003   P_DOWN equ 3h
0003   PD equ 3h
0004   T_OUT equ 4h
0004   TO equ 4h
0005   PA0 equ 5h
0006   PA1 equ 6h
0007   PA2 equ 7h
0001   Same equ 1h
0000   LSB equ 0h
0007   MSB equ 7h
0001   TRUE equ 1h
0001   YES equ 1h
0000   FALSE equ 0h
0000   NO equ 0h

;*************************************************************************

org 0

;*************************************************************************

0001   SIGNED equ TRUE    ; Set This To 'TRUE' if the routines
0000   for Multiplication & Division needs
0000   to be assembled as Signed Integer
0000   ; Routines. If 'FALSE' the above two
0000   ; routines ( D.mpy & D.div ) use
0000   ; unsigned arithmetic.
0000   ;*************************************************************************

; Double Precision Subtraction ( ACCb - ACCa -> ACCb )

0000 0210   D_add movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
0001 01F2   addwf ACCbLO ;add lsb
0002 0603   btfsC STATUS,CARRY ;add in carry
0003 02B3   incf ACCbHI
0004 0211   movf ACCaHI,w
0005 01F3   addwf ACCbHI ;add msb
0006 0800   retlw 0

;*************************************************************************

; Double Precision Multiply ( 16x16 -> 32 )
; ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
; in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO )
;
D_mpyS ;results in ACCb(16 msb's) and ACCc(16

0007 0930   CALL S_SIGN
0008 0921 ENDIF

0008 0921   call setup
0009 0337   mloop rrf ACCbHI ;rotate d right
000A 0336   rrf ACCbLO
000B 0603   btfsC STATUS,CARRY ;need to add?
000C 0900   call D_add
000D 0333   rrf ACCbHI
000E 0332   rrf ACCbLO
000F 0335   rrf ACCcHI
0010 0334   rrf ACCcLO
0011 02F8   decfsz temp ;loop until all bits checked
0012 0A09   goto mloop
PIC16C5X Math Routines

; IF SIGNED
0013 07F9 btfs s sign,MSB
0014 0800 retlw 0
0015 0274 comf ACCcLO ; negate ACCa ( -ACCa -> ACCa )
0016 02B4 incf ACCcLO
0017 0643 btfsc STATUS, Z_bit
0018 00F5 decf ACCcHI
0019 0275 comf ACCcHI
001A 0643 btfsc STATUS, Z_bit
001B 0272 neg_B comf ACCcLO ; negate ACCb
001C 02B2 incf ACCcLO
001D 0643 btfsc STATUS, Z_bit
001E 00F3 decf ACCcHI
001F 0273 comf ACCcHI
0020 0800 retlw 0
ELSE
retlw 0
ENDIF

;******************************************************************************
;
0021 0C10 setup movlw .16 ; for 16 shifts
0022 0038 movwf temp
0023 0213 movf ACCcHI, W ; move ACCb to ACCd
0024 0037 movwf ACCcHI
0025 0212 movwf ACCcLO, W
0026 0036 movwf ACCcLO
0027 0073 clrf ACCcHI
0028 0072 clrf ACCcLO
0029 0800 retlw 0

;******************************************************************************
;
002A 0270 neg_A comf ACCaLO ; negate ACCa ( -ACCa -> ACCa )
002B 02B0 incf ACCaLO
002C 0643 btfs  STATUS, Z_bit
002D 00F1 decf ACCaHI
002E 0271 comf ACCaHI
002F 0800 retlw 0

;******************************************************************************
;
0030 0211 S_SIGN movf ACCaHI, W
0031 0193 xorwf ACCaHI, W
0032 0039 movwf sign
0033 07F3 btfs ACCaHI, MSB ; if MSB set go & negate ACCb
0034 0A3A goto chek_A

; 0035 0272 comf ACCcLO ; negate ACCb
0036 02B2 incf ACCcLO
0037 0643 btfsc STATUS, Z_bit
0038 00F3 decf ACCcHI
0039 0273 comf ACCcHI

; 003A 07F1 chek_A btfs ACCaHI, MSB ; if MSB set go & negate ACCa
003B 0800 retlw 0
003C 0A2A goto neg_A

; ENDIF
Test Program

; Load constant values to ACCa & ACCb for testing
;
003D 0C01
003E 0031
003F 0CFF
0040 0030
0041 0C7F
0042 0033
0043 0CFF
0044 0032
0045 0907
0046 0A46
01FF 0A3D

Errors : 0
Warnings : 0
APPENDIX D: DOUBLE PRECISION MULTIPLICATION LISTINGS (FAST)

MPASM B0.54

;******************************************************************************
; Double Precision Multiplication
; ( Optimized for Speed : straight Line Code )
;******************************************************************************;
;
; Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_mpy
; (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
;
; Performance :
; Program Memory : 240
; Clock Cycles : 233
;
; Note : The above timing is the worst case timing, when the
; register ACCb = FFFF. The speed may be improved if
; the register ACCb contains a number ( out of the two
; numbers ) with less number of ls.
; The performance specs are for Unsigned arithmetic ( i.e,
; with "SIGNED equ FALSE ").
;
;******************************************************************************;

LIST p=16c54

0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 ACCbLO equ 12
0013 ACCbHI equ 13
0014 ACCcLO equ 14
0015 ACCcHI equ 15
0016 ACCdLO equ 16
0017 ACCdHI equ 17
0018 temp equ 18
0019 sign equ 19

; include "mpreg.h"

;******************************************************************************

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH
0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h
0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

; STATUS REG. Bits
0000 C equ 0h ; Carry Bit is Bit.0 of F3
0001 D equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h

© 1993 Microchip Technology Inc.
PIC16C5X Math Routines

```assembly
0003   PD  equ    3h
0004   T_OUT equ    4h
0004   T0 equ    4h
0005   PA0 equ    5h
0006   PA1 equ    6h
0007   PA2 equ    7h

; Same equ    1h
;
0000   LSB equ    0h
0007   MSB equ    7h
;
0001   TRUE equ    1h
0001   YES equ    1h
0000   FALSE equ    0h
0000   NO equ    0h
;
*************************************************************************
org 0 ;*******************************************************************
SIGNED equ FALSE Set This To 'TRUE' if the routines
; for Multiplication & Division needs
; to be assembled as Signed Integer
; Routines. If 'FALSE' the above two
; routines ( D_mpy & D_div ) use
; unsigned arithmetic.
;*************************************************************************
; multiplication macro

mulMac MACRO
LOCAL NO_ADD
;
rdf ACCbHI ;rotate d right
rdf ACCbLO
btfss STATUS,CARRY ;need to add?
goto NO_ADD ;no addition necessary
movf ACCaLO, w ;Addition ( ACCb + ACCa -> ACCb )
addwf ACCbLO ;add lab
btfsc STATUS,CARRY ;add in carry
incf ACCbHI
movf ACCaHI, w
addwf ACCbHI ;add msb
NO_ADD rdf ACCbHI
rdf ACCbLO
rdf ACCbHI
rdf ACCbLO
;
ENDM
;
***************************************************************************;
; Double Precision Multiply ( 16x16 -> 32 )
; ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
; in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
; D_mpyF ;results in ACCb(16 msb's) and ACCc(16 lsb's)
;
IF SIGNED CALL S_SIGN
ENDIF

0000 09E2 ; call setup
```
; use the mulMac macro 16 times
;
mulMac
    LOCAL   NO_ADD
    
    rrf ACCdHI ; rotate d right
    rrf ACCdLO
    btfss STATUS,CARRY ; need to add?
    goto NO_ADD ; no addition necessary
    movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
    addwf ACCbLO ; add lsb
    btfsc STATUS,CARRY ; add in carry
    incf ACCbHI
    movf ACCaHI,w
    addwf ACCbHI ; add msb
    rrf ACCbHI
    rrf ACCbLO
    rrf ACCbHI
    rrf ACCbLO

mulMac
    LOCAL   NO_ADD
    
    rrf ACCdHI ; rotate d right
    rrf ACCdLO
    btfss STATUS,CARRY ; need to add?
    goto NO_ADD ; no addition necessary
    movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
    addwf ACCbLO ; add lsb
    btfsc STATUS,CARRY ; add in carry
    incf ACCbHI
    movf ACCaHI,w
    addwf ACCbHI ; add msb
    rrf ACCbHI
    rrf ACCbLO
    rrf ACCbHI
    rrf ACCbLO

mulMac
    LOCAL   NO_ADD
    
    rrf ACCdHI ; rotate d right
    rrf ACCdLO
    btfss STATUS,CARRY ; need to add?
    goto NO_ADD ; no addition necessary
    movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
    addwf ACCbLO ; add lsb
    btfsc STATUS,CARRY ; add in carry
    incf ACCbHI
    movf ACCaHI,w
    addwf ACCbHI ; add msb
    rrf ACCbHI
    rrf ACCbLO
    rrf ACCbHI
    rrf ACCbLO

;
mulMac
LOCAL NO_ADD

002B 0337    rrf   ACCdHI    ;rotate d right
002C 0336    rrf   ACCdLO
002D 0703    btfss  STATUS,CARRY ;need to add?
002E 0A35    goto   NO_ADD    ; no addition necessary
002F 0210    movf   ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
0030 01F2    addwf  ACCbLO    ;add lsb
0031 0603    btfsc  STATUS,CARRY ;add in carry
0032 02B3    incf   ACCdHI
0033 0211    movf   ACCaHI,w
0034 01F3    addwf  ACCdHI    ;add msb
0035 0333    NO_ADD  rrf   ACCdHI
0036 0332    rrf   ACCbLO
0037 0335    rrf   ACCcHI
0038 0334    rrf   ACCcLO

mulMac
LOCAL NO_ADD

0039 0337    rrf   ACCdHI    ;rotate d right
003A 0336    rrf   ACCdLO
003B 0703    btfss  STATUS,CARRY ;need to add?
003C 0A43    goto   NO_ADD    ; no addition necessary
003D 0210    movf   ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
003E 01F2    addwf  ACCbLO    ;add lsb
003F 0603    btfsc  STATUS,CARRY ;add in carry
0040 02B3    incf   ACCdHI
0041 0211    movf   ACCaHI,w
0042 01F3    addwf  ACCdHI    ;add msb
0043 0333    NO_ADD  rrf   ACCdHI
0044 0332    rrf   ACCbLO
0045 0335    rrf   ACCcHI
0046 0334    rrf   ACCcLO

mulMac
LOCAL NO_ADD

0047 0337    rrf   ACCdHI    ;rotate d right
0048 0336    rrf   ACCdLO
0049 0703    btfss  STATUS,CARRY ;need to add?
004A 0A51    goto   NO_ADD    ; no addition necessary
004B 0210    movf   ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
004C 01F2    addwf  ACCbLO    ;add lsb
004D 0603    btfsc  STATUS,CARRY ;add in carry
004E 02B3    incf   ACCdHI
004F 0211    movf   ACCaHI,w
0050 01F3    addwf  ACCdHI    ;add msb
0051 0333    NO_ADD  rrf   ACCdHI
0052 0332    rrf   ACCbLO
0053 0335    rrf   ACCcHI
0054 0334    rrf   ACCcLO

mulMac
LOCAL NO_ADD

0055 0337    rrf   ACCdHI    ;rotate d right
0056 0336    rrf   ACCdLO
0057 0703    btfss  STATUS,CARRY ;need to add?
0058 0A5F    goto   NO_ADD    ; no addition necessary
0059 0210    movf   ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
005A 01F2    addwf  ACCbLO    ;add lsb
005B 0603    btfsc  STATUS,CARRY ;add in carry
005C 02B3    incf   ACCdHI
005D 0211    movf   ACCaHI,w
005E 01F3    addwf  ACCdHI    ;add msb

© 1993 Microchip Technology Inc.
PIC16C5X Math Routines

mulMac
LOCAL NO_ADD

0063 0337 rrf ACCdHI ; rotate d right
0064 0336 rrf ACCdLO
0065 0703 btfs S STATUS,CARRY ; need to add?
0066 0A6D gOto NO_ADD ; no addition necessary
0067 0210 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
0068 01F2 addwf ACCbLO ; add lsb
0069 0603 btfs C STATUS,CARRY ; add in carry
006A 02B3 incf ACCbHI
006B 0211 movf ACCaHI,w
006C 01F3 addwf ACCbHI ; add msb

mulMac
LOCAL NO_ADD

0071 0337 rrf ACCdHI ; rotate d right
0072 0336 rrf ACCdLO
0073 0703 btfs S STATUS,CARRY ; need to add?
0074 0A7B gOto NO_ADD ; no addition necessary
0075 0210 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
0076 01F2 addwf ACCbLO ; add lsb
0077 0603 btfs C STATUS,CARRY ; add in carry
0078 02B3 incf ACCbHI
0079 0211 movf ACCaHI,w
007A 01F3 addwf ACCbHI ; add msb

mulMac
LOCAL NO_ADD

007F 0337 rrf ACCdHI ; rotate d right
0080 0336 rrf ACCdLO
0081 0703 btfs S STATUS,CARRY ; need to add?
0082 0A89 gOto NO_ADD ; no addition necessary
0083 0210 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
0084 01F2 addwf ACCbLO ; add lsb
0085 0603 btfs C STATUS,CARRY ; add in carry
0086 02B3 incf ACCbHI
0087 0211 movf ACCaHI,w
0088 01F3 addwf ACCbHI ; add msb

mulMac
LOCAL NO_ADD

0089 0333 rrf ACCbHI
008A 0332 rrf ACCbLO
008B 0335 rrf ACCcHI
008C 0334 rrf ACCcLO

mulMac
LOCAL NO_ADD

005F 0333 NO_ADD rrf ACCbHI
0060 0332 rrf ACCbLO
0061 0335 rrf ACCcHI
0062 0334 rrf ACCcLO

mulMac
LOCAL NO_ADD
mulMac
LOCAL NO_ADD

008D 0337
 rrf ACCdHI ; rotate d right
008E 0336
 rrf ACCdLO
008F 0703
 btfs STATUS,CARRY ; need to add?
0090 0AA7
 goto NO_ADD ; no addition necessary
0091 0210
 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
0092 01F2
 addwf ACCbLO ; add lsb
0093 0603
 btfs STATUS,CARRY ; add in carry
0094 02B3
 incf ACCbHI
0095 0211
 movf ACCaHI,w
0096 01F3
 addwf ACCbHI ; add msb
0097 0333
 NO_ADD rrf ACCbHI
0098 0332
 rrf ACCbLO
0099 0335
 rrf ACCCHI
009A 0334
 mulMac
LOCAL NO_ADD

009B 0337
 rrf ACCbHI ; rotate d right
009C 0336
 rrf ACCbLO
009D 0703
 btfs STATUS,CARRY ; need to add?
009E 0AA5
 goto NO_ADD ; no addition necessary
009F 0210
 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
00A0 01F2
 addwf ACCbLO ; add lsb
00A1 0603
 btfs STATUS,CARRY ; add in carry
00A2 02B3
 incf ACCbHI
00A3 0211
 movf ACCaHI,w
00A4 01F3
 addwf ACCbHI ; add msb
00A5 0333
 NO_ADD rrf ACCbHI
00A6 0332
 rrf ACCbLO
00A7 0335
 rrf ACCCHI
00A8 0334
 mulMac
LOCAL NO_ADD

00A9 0337
 rrf ACCbHI ; rotate d right
00AA 0336
 rrf ACCbLO
00AB 0703
 btfs STATUS,CARRY ; need to add?
00AC 0AB3
 goto NO_ADD ; no addition necessary
00AD 0210
 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
00AE 01F2
 addwf ACCbLO ; add lsb
00AF 0603
 btfs STATUS,CARRY ; add in carry
00B0 02B3
 incf ACCbHI
00B1 0211
 movf ACCaHI,w
00B2 01F3
 addwf ACCbHI ; add msb
00B3 0333
 NO_ADD rrf ACCbHI
00B4 0332
 rrf ACCbLO
00B5 0335
 rrf ACCCHI
00B6 0334
 mulMac
LOCAL NO_ADD

00B7 0337
 rrf ACCbHI ; rotate d right
00B8 0336
 rrf ACCbLO
00B9 0703
 btfs STATUS,CARRY ; need to add?
00BA 0AC1
 goto NO_ADD ; no addition necessary
00BB 0210
 movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
00BC 01F2
 addwf ACCbLO ; add lsb
00BD 0603
 btfs STATUS,CARRY ; add in carry
00BE 02B3
 incf ACCbHI
00BF 0211
 movf ACCaHI,w
00C0 01F3
 addwf ACCbHI ; add msb
PIC16C5X Math Routines

; rotate d right
; need to add?
; no addition necessary
; Addition ( ACCb + ACCa -> ACCb )
; add lsb
; add in carry
; add msb
; rotate d right
; need to add?
; no addition necessary
; Addition ( ACCb + ACCa -> ACCb )
; add lsb
; add in carry
; add msb
; negate ACCa ( -ACCa -> ACCa )
; negate ACCb
; negate ACCb
; negate ACCb

00C1 0333  NO_ADD rrf  ACCbHI
00C2 0332  rrf  ACCbLO
00C3 0335  rrf  ACCcHI
00C4 0334  rrf  ACCcLO

; mulMac
; LOCAL  NO_ADD
;
00C5 0337  rrf  ACCdHI
00C6 0336  rrf  ACCdLO
00C7 0703  btfsb STATUS,CARRY
00C8 0ACF  goto  NO_ADD
00C9 0210  movf ACCaLO,w
00CA 01F2  addwf ACCaLO
00CB 0603  btfsb STATUS,CARRY
00CC 02B3  incf ACCbHI
00CD 0211  movf ACCaHI,w
00CE 01F3  addwf ACCbHI
00CF 0333  NO_ADD rrf  ACCbHI
00D0 0332  rrf  ACCbLO
00D1 0335  rrf  ACCcHI
00D2 0334  rrf  ACCcLO

; mulMac
; LOCAL  NO_ADD
;
00D3 0337  rrf  ACCdHI
00D4 0336  rrf  ACCdLO
00D5 0703  btfsb STATUS,CARRY
00D6 0ADD  goto  NO_ADD
00D7 0210  movf ACCaLO,w
00D8 01F2  addwf ACCaLO
00D9 0603  btfsb STATUS,CARRY
00DA 02B3  incf ACCbHI
00DB 0211  movf ACCaHI,w
00DC 01F3  addwf ACCbHI
00DD 0333  NO_ADD rrf  ACCbHI
00DE 0332  rrf  ACCbLO
00DF 0335  rrf  ACCcHI
00E0 0334  rrf  ACCcLO

;
;
IF SIGNED
btfsb sign,MSB
retlw 0
comf ACCcLO
incf ACCcLO
btfsb STATUS,Z_bit
decf ACCcHI
comf ACCcHI
btfsb STATUS,Z_bit
neg_B
comf ACCbLO
incf ACCbLO
btfsb STATUS,Z_bit
decf ACCbHI
comf ACCbHI
retlw 0
ELSE
retlw 0
ENDIF
; PIC16C5X Math Routines

;******************************************************************************

setup movlw .16 ; for 16 shifts
00E2 0C10
00E3 0038
00E4 0213
00E5 0037
00E6 0212
00E7 0036
00E8 0073
00E9 0072
00EA 0800

;******************************************************************************

neg_A comf ACCaLO ; negate ACCa ( -ACCa -> ACCa )
00EB 0270
00EC 02B0
00ED 0643
00EE 00F1
00EF 0271
00F0 0800

;******************************************************************************

; Assemble this section only if Signed Arithmetic Needed
; IF SIGNED
;
S_SIGN movf ACCaHI,W
xorwf ACCbHI,W
movwf sign
btfss ACCbHI,MSB if MSB set go & negate ACCb
goto chek_A
comf ACCbLO ; negate ACCb
incf ACCbLO
btfsc STATUS, Z_bit
decf ACCbHI
comf ACCbHI
;
chek_A btfss ACCaHI,MSB ; if MSB set go & negate ACCa
retlw 0
goto neg_A
;
ENDIF
;

;******************************************************************************

DS00526C-page 22 © 1993 Microchip Technology Inc.

2-176
;*******************************************************************
; Test Program
;*******************************************************************
; Load constant values to ACCa & ACCb for testing
;
loadAB	movlw 1
00F1 0C01
movwf ACCaHI
00F2 0031
movlw OFF
00F3 0CFF
movwf ACCaLO
00F4 0030
;
movlw 07F
00F5 0C7F
movwf ACCbHI
00F6 0033
movlw OFF
00F7 0CFF
movwf ACCbLO
00F8 0032
retlw 0
00F9 0800
;
main	nop
00FA 0000
;
call loadAB
00FB 09F1
; result of multiplying ACCb*ACCa=
call D_mpyF
00FC 0900
; Here (ACCb, ACCc) = 00FF 7E01
;
self	goto self
00FD 0AFD
; org PIC54
org
01FF 00AF
;
goto main
END

Errors : 0
Warnings : 0
APPENDIX E: DOUBLE PRECISION DIVISION LISTING (LOOPED)

MPASM B0.54

;***************************************************************************
; Double Precision Division
; ( Optimized for Code Size : Looped Code )
;***************************************************************************

Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
Remainder in ACCc (16 bits)
(a) Load the Denominator in location ACCaHI & ACCaLO (16 bits)
(b) Load the Numerator in location ACCbHI & ACCbLO (16 bits)
(c) CALL D_div
(d) The 16 bit result is in location ACCbHI & ACCbLO
(e) The 16 bit Remainder is in locations ACCcHI & ACCcLO

Performance:
Program Memory : 037
Clock Cycles : 310

NOTE :
The performance specs are for Unsigned arithmetic (i.e.,
with "SIGNED equ FALSE").

;***************************************************************************

LIST P-16C54

0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 ACCbLO equ 12
0013 ACCbHI equ 13
0014 ACCcLO equ 14
0015 ACCcHI equ 15
0016 ACCdLO equ 16
0017 ACCdHI equ 17
0018 temp equ 18
0019 sign equ 19

;***************************************************************************

include "mpreg.h"

;***************************************************************************

PIC16C5X Header

;***************************************************************************

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h

0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

;***************************************************************************

© 1993 Microchip Technology Inc.

DS00526C-page 24 2-178
**PIC16C5X Math Routines**

```plaintext
; STATUS REG. Bits
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h

; Same equ 1h
;
0000 LSB equ 0h
0007 MSB equ 7h
;
0001 TRUE equ 1h
0001 YES equ 1h
0000 FALSE equ 0h
0000 NO equ 0h

;*************************************************************************

org 0

;*************************************************************************

org 0

SIGNED equ FALSE ; Set This To 'TRUE' if the routines
; for Multiplication & Division needs
; to be assembled as Signed Integer
; Routines. If 'FALSE' the above two
; routines ( D_mpy & D_div ) use
; unsigned arithmetic.
;
; Double Precision Divide ( 16/16 --> 16 )
; ( ACCb/ACCa --> ACCb with remainder in ACCc ) ; 16 bit output
; With Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc

(ACCcHI,ACCcLO).

; NOTE : Before calling this routine, the user should make sure that
; the Numerator(ACCb) is greater than Denominator(ACCa). If
; the case is not true, the user should scale either Numerator
; or Denominator or both such that Numerator is greater than
; the Denominator.
;
; D_divs
;
IF SIGNED
CALL S_SIGN
ENDIF

0000 091C call setup
0001 0075 clrf ACCcHI
0002 0074 clrf ACCcLO
0003 0403 dloop bcf STATUS,CARRY
0004 0376 rlf ACCdLO
0005 0377 rlf ACCdHI
0006 0374 rlf ACCcLO
0007 0375 rlf ACCcHI
```
PIC16C5X Math Routines

0008 0211   movf ACCaHI,w
0009 0095   subwf ACCcHI,w ;check if a>c
000A 0743   btfss STATUS,Z_bit
000B 0A0E   goto nochk
000C 0210   movf ACCaLO,w
000D 0094   subwf ACCcLO,w ;if msb equal then check lsb
000E 0703   nochk btfss STATUS,CARRY ;carry set if c>\n
000F 0A17   goto nogo
0010 0210   movf ACCaLO,w
0011 0094   subwf ACCcLO
0012 0703   btfss STATUS,CARRY
0013 0095   decf ACCcHI
0014 0211   movf ACCaHI,w
0015 0085   subwf ACCcHI
0016 0503   bsf STATUS,CARRY ;shift a 1 into b (result)
0017 0372   nogo rlf ACCbLO
0018 0373   rlf ACCbHI
0019 02F8   decfsz temp ;loop untill all bits checked
001A 0A03   goto dloop

; IF SIGNED
001B 0800   ;*******************************************************************
001C 0C10   setup movlw .16 ; for 16 shifts
001D 0038   movwf temp
001E 0213   movf ACCbHI,w ;move ACCb to ACCd
001F 0037   movwf ACCdHI
0020 0212   movf ACCbLO,w
0021 0036   movwf ACCdLO
0022 0073   clrf ACCcHI
0023 0072   clrf ACCcLO
0024 0800   retlw 0

;*******************************************************************

0025 0270   neg_A movf ACCaLO ; negate ACCa ( -ACCa -> ACCa )
0026 02B0   incf ACCaLO
0027 0643   btfsc STATUS,Z_bit
0028 00F1   decf ACCaHI
0029 0271   comf ACCcHI
002A 0800   retlw 0

;*******************************************************************

; Assemble this section only if Signed Arithmetic Needed
; IF SIGNED
002B 0270   ; S_SIGN movf ACCaHI,w
002C 02B0   xorwf ACCbHI,w
002D 00F1   movwf sign
002E 0703   btfss ACCbHI,MSB ; if MSB set go & negate ACCb
002F 0A17   goto chek_A

; comf ACCbLO ; negate ACCb
0030 02B0   incf ACCbLO
0031 0643   btfsc STATUS,Z_bit
0032 00F1   decf ACCcHI
0033 0072   comf ACCcHI

;
chek_A btfss ACCaHI,MSB ; if MSB set go & negate ACCa
    retlw 0
    goto neg_A
;
    ENDIF
;
;******************************************************************************
; Test Program
;******************************************************************************
;
Load constant values to ACCa & ACCb for testing
;
main movlw 1
    movwf ACCaHI
    movlw 0FF ; loads ACCa = 01FF
002E 0030
    movwf ACCaLO
;
002F 0C7F
    movlw 07F
0030 0033
    movwf ACCbHI
0031 0CFF
    movlw 0FF ; loads ACCb = 7FFF
0032 0032
    movwf ACCbLO
;
0033 0900
    call D_div5 ; remainder in ACCc. Here ACCb =0040 &
    ACCc=003F
;
0034 0A34
    self goto self
;
    org PIC54
01FF 0A2B
    goto main
END
;******************************************************************************

Errors : 0
Warnings : 0
APPENDIX F: DOUBLE PRECISION DIVISION LISTING (FAST)

;*******************************************************************
; Double Precision Division
; ( Optimized for Speed : straight Line Code )
;*******************************************************************;
; Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits)
; Remainder in ACCc (16 bits)
; (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
; (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
; (c) CALL D_div
; (d) The 16 bit result is in location ACCbHI & ACCbLO
; (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;
; Performance :
; Program Memory : 370
; Clock Cycles : 263
;
; NOTE :
; The performance specs are for Unsigned arithmetic (i.e.,
; with "SIGNED equ FALSE ").

;*******************************************************************;

LIST P=16C54

0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 ACCbLO equ 12
0013 ACCbHI equ 13
0014 ACCcLO equ 14
0015 ACCcHI equ 15
0016 ACCdLO equ 16
0017 ACCdHI equ 17
0018 temp equ 18
0019 sign equ 19

#include "picreg.h"

;----------------------------------------------------------------------
PIC16C5X Header
;----------------------------------------------------------------------

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h

0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

;----------------------------------------------------------------------
; STATUS REG. Bits
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
PIC16C5X Math Routines

0002  Z_bit equ 2h  ; Bit 2 of F3 is Zero Bit
0002  Z equ 2h
0003  P_DOWN equ 3h
0003  PD equ 3h
0004  T_OUT equ 4h
0004  TO equ 4h
0005  PA0 equ 5h
0006  PA1 equ 6h
0007  PA2 equ 7h

; Same equ 1h
; LSB equ 0h
; MSB equ 7h
; TRUE equ 1h
; YES equ 1h
; FALSE equ 0h
; NO equ 0h
;
;*************************************************************************
org 0

SIGNED equ FALSE  ; Set This To 'TRUE' if the routines
; for Multiplication & Division needs
; to be assembled as Signed Integer
; Routines. If 'FALSE' the above two
; routines ( D~mpy & D~div ) use
; unsigned arithmetic.
;
;****************************************************************************;
divMac MACRO
LOCAL NOCHK
LOCAL NOGO
;
bcf STATUS,CARRY
rlf ACCdLO
rlf ACCdHI
rlf ACCcLO
rlf ACCcHI
movf ACCaHI,w
subwf ACCcHI,w  ;check if a>c
btfss STATUS,Z_bit
go NOCHK
movf ACCaLO,w
subwf ACCcLO,w  ;if msb equal then check lsb
NOCHK btfs STATUS,CARRY  ;carry set if c>a
go NOGO
movf ACCaLO,w
subwf ACCcLO
btfs STATUS,CARRY
decf ACCcHI
movf ACCaHI,w
subwf ACCcHI
bsf STATUS,CARRY  ;shift a 1 into b (result)
NOGO rlf ACCbLO
rlf ACCbHI
;
ENDM
;
;*****************************************************************************
; Double Precision Divide ( 16/16 -> 16 )
;
; ( ACCb/ACCa -> ACCb with remainder in ACCc ) ; 16 bit output
; with Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
;
NOTE: Before calling this routine, the user should make sure that the Numerator(ACCb) is greater than Denominator(ACCa). If the case is not true, the user should scale either Numerator or Denominator or both such that Numerator is greater than the Denominator.

setup movlw .16 ; for 16 shifts
movwf temp
movf ACCbHI,w ;move ACCb to ACCd
movwf ACCdHI
movf ACCbLO,w
movwf ACCdLO
clrf ACCbHI
clrf ACCbLO
retlw 0

neg_A comp ACCaLO ; negate ACCa (-ACCa -> ACCa)
incf ACCaLO
btfsc STATUS,Z_bit
cornf ACCaHI
retlw 0

D_divF
IF SIGNED
CALL S_SIGN
ENDIF

call setup
clrf ACCcHI
clrf ACCcLO

; use the mulMac macro 16 times

divMac
LOCAL NOCHK
LOCAL NOGO

bcf STATUS,CARRY
rlf ACCcLO
rlf ACCcHI
rlf ACCcLO
rlf ACCcHI
movf ACCaHI,w
subwf ACCcHI,w ;check if a>c
btfss STATUS,Z_bit
goto NOCHK
movf ACCcLO,w
subwf ACCcLO,w ;if msb equal then check lsb
NOCHK btfss STATUS,CARRY ;carry set if c>a
goto NOGO
movf ACCcLO,w ;c-a into c
subwf ACCcLO
btfss STATUS,CARRY
movf ACCbHI
subwf ACCcHI
bsf STATUS,CARRY ;shift a 1 into b (result)
; PIC16C5X Math Routines

0026 0372  NOGO  rlf  ACCbLO
0027 0373  rlf  ACCbHI

; divMac
  LOCAL  NOCHK
  LOCAL  NOGO

  0028 0403  bcf  STATUS,CARRY
  0029 0376  rlf  ACCdLO
  002A 0377  rlf  ACCdHI
  002B 0374  rlf  ACCcLO
  002C 0375  rlf  ACCcHI
  002D 0211  movf  ACCaHI,w
  002E 0095  subwf  ACCcHI,w  ;check if a>c
  002F 0743  btfs  STATUS,Z_bit
  0030 0A33  goto  NOCHK
  0031 0210  movf  ACCaLO,w
  0032 0094  subwf  ACCcLO,w  ;if msb equal then check lsb
  0033 0703  NOCHK  btfs  STATUS,CARRY  ;carry set if c>a
  0034 0A3C  goto  NOGO
  0035 0210  movf  ACCaLO,w  ;c-a into c
  0036 0084  subwf  ACCcLO
  0037 0703  btfs  STATUS,CARRY
  0038 00F5  decf  ACCcHI
  0039 0211  movf  ACCaHI,w
  003A 0085  subwf  ACCcHI
  003B 0503  bsf  STATUS,CARRY  ;shift a 1 into b (result)
  003C 0372  NOGO  rlf  ACCbLO
  003D 0373  rlf  ACCbHI

; divMac
  LOCAL  NOCHK
  LOCAL  NOGO

  003E 0403  bcf  STATUS,CARRY
  003F 0376  rlf  ACCdLO
  0040 0377  rlf  ACCdHI
  0041 0374  rlf  ACCcLO
  0042 0375  rlf  ACCcHI
  0043 0211  movf  ACCaHI,w
  0044 0095  subwf  ACCcHI,w  ;check if a>c
  0045 0743  btfs  STATUS,Z_bit
  0046 0A49  goto  NOCHK
  0047 0210  movf  ACCaLO,w
  0048 0094  subwf  ACCcLO,w  ;if msb equal then check lsb
  0049 0703  NOCHK  btfs  STATUS,CARRY  ;carry set if c>a
  004A 0A52  goto  NOGO
  004B 0210  movf  ACCaLO,w  ;c-a into c
  004C 0084  subwf  ACCcLO
  004D 0703  btfs  STATUS,CARRY
  004E 00F5  decf  ACCcHI
  004F 0211  movf  ACCaHI,w
  0050 0085  subwf  ACCcHI
  0051 0503  bsf  STATUS,CARRY  ;shift a 1 into b (result)
  0052 0372  NOGO  rlf  ACCbLO
  0053 0373  rlf  ACCbHI

; divMac
  LOCAL  NOCHK
  LOCAL  NOGO

  0054 0403  bcf  STATUS,CARRY
  0055 0376  rlf  ACCdLO
  0056 0377  rlf  ACCdHI
  0057 0374  rlf  ACCcLO
  0058 0375  rlf  ACCcHI
PIC16C5X Math Routines

```
0059 0211  movf  ACCaHI, w
005A 0095  subwf  ACCcHI, w ;check if a>c
005B 0743  btfss  STATUS, Z_bit
005C 0A5F  goto  NOCHK
005D 0210  movf  ACCcLO, w
005E 0094  subwf  ACCcLO, w ;if msb equal then check lab
005F 0703  NOCHK  btfss  STATUS, CARRY ;carry set if c>a
0060 0A68  goto  NOGO
0061 0210  movf  ACCcLO, w
0062 00B4  subwf  ACCcLO
0063 0703  btfss  STATUS, CARRY
0064 00F5  decf  ACCcHI
0065 0211  movf  ACCaHI, w
0066 00B5  subwf  ACCcHI
0067 0503  bsf  STATUS, CARRY ;shift a 1 into b (result)
0068 0372  NOGO  rlf  ACCcLO
0069 0373  rlf  ACCcHI

; divMac
LOCAL NOCHK
LOCAL NOGO

006A 0403  bcf  STATUS, CARRY
006B 0376  rlf  ACCcLO
006C 0377  rlf  ACCcHI
006D 0374  rlf  ACCcLO
006E 0375  rlf  ACCcHI
006F 0211  movf  ACCaHI, w
0070 0095  subwf  ACCcHI, w ;check if a>c
0071 0743  btfss  STATUS, Z_bit
0072 0A75  goto  NOCHK
0073 0210  movf  ACCcLO, w
0074 0094  subwf  ACCcLO ;if msb equal then check lab
0075 0703  NOCHK  btfss  STATUS, CARRY ;carry set if c>a
0076 0A7E  goto  NOGO
0077 0210  movf  ACCcLO, w
0078 00B4  subwf  ACCcLO
0079 0703  btfss  STATUS, CARRY
007A 00F5  decf  ACCcHI
007B 0211  movf  ACCaHI, w
007C 00B5  subwf  ACCcHI
007D 0503  bsf  STATUS, CARRY ;shift a 1 into b (result)
007E 0372  NOGO  rlf  ACCcLO
007F 0373  rlf  ACCcHI

; divMac
LOCAL NOCHK
LOCAL NOGO

0080 0403  bcf  STATUS, CARRY
0081 0376  rlf  ACCcLO
0082 0377  rlf  ACCcHI
0083 0374  rlf  ACCcLO
0084 0375  rlf  ACCcHI
0085 0211  movf  ACCaHI, w
0086 0095  subwf  ACCcHI, w ;check if a>c
0087 0743  btfss  STATUS, Z_bit
0088 0AB8  goto  NOCHK
0089 0210  movf  ACCcLO, w
008A 0094  subwf  ACCcLO ;if msb equal then check lab
008B 0703  NOCHK  btfss  STATUS, CARRY ;carry set if c>a
008C 0A94  goto  NOGO
008D 0210  movf  ACCcLO, w
008E 00B4  subwf  ACCcLO
008F 0703  btfss  STATUS, CARRY
0090 00F5  decf  ACCcHI
0091 0211  movf  ACCaHI, w
```
0092 00B5  subwf  ACCcHI
0093 0503  bsf  STATUS, CARRY  ;shift a 1 into b (result)
0094 0372  \textbf{NCGO}  rlf  ACCbLO
0095 0373  rlf  ACCbHI

;  \texttt{divMac}
\texttt{LOCAL NOCHK}
\texttt{LOCAL NOGO}

0096 0403  bcf  STATUS, CARRY
0097 0376  rlf  ACCcLO
0098 0377  rlf  ACCcHI
0099 0374  rlf  ACCbLO
009A 0375  rlf  ACCcHI
009B 0211  movf  ACCcHI, w
009C 0095  subwf  ACCcHI, w  ;check if \texttt{a>c}
009D 0743  btfss STATUS, Z_bit  \texttt{goto NOCHK}
009E 00A1  movf  ACCaLO, w
009F 0210  \texttt{goto NOGO}
00A0 0094  subwf  ACCcLO, w  ;if msb equal then check lsb
00A1 0703  \texttt{NOCHK}  btfss STATUS, CARRY  \texttt{carry set if c>a}
00A2 0AAA  \texttt{goto NOGO}
00A3 0210  movf  ACCcLO, w  \texttt{c-a into c}
00A4 00B4  subwf  ACCcLO
00A5 0703  btfss STATUS, CARRY
00A6 00F5  decf ACCcHI
00A7 0211  movf  ACCcHI, w
00A8 00B5  subwf  ACCcHI
00A9 0503  bcf  STATUS, CARRY  ;shift a 1 into b (result)
00AA 0372  \textbf{NCGO}  rlf  ACCbLO
00AB 0373  rlf  ACCbHI

;  \texttt{divMac}
\texttt{LOCAL NOCHK}
\texttt{LOCAL NOGO}

00AC 0403  bcf  STATUS, CARRY
00AD 0376  rlf  ACCdLO
00AE 0377  rlf  ACCcHI
00AF 0374  rlf  ACCbLO
00B0 0375  rlf  ACCcHI
00B1 0211  movf  ACCcHI, w
00B2 0095  subwf ACCcHI, w  ;check if \texttt{a>c}
00B3 0743  btfss STATUS, Z_bit
00B4 00A7  \texttt{goto NOCHK}
00B5 0210  movf  ACCaLO, w
00B6 0094  subwf  ACCcLO, w  ;if msb equal then check lsb
00B7 0703  \texttt{NOCHK}  btfss STATUS, CARRY  \texttt{carry set if c>a}
00B8 00A0  \texttt{goto NOGO}
00B9 0210  movf  ACCcLO, w  \texttt{c-a into c}
00BA 00B4  subwf  ACCcLO
00BB 0703  btfss STATUS, CARRY
00BC 00F5  decf ACCcHI
00BD 0211  movf  ACCcHI, w
00BE 00B5  subwf  ACCcHI
00BF 0503  bcf  STATUS, CARRY  ;shift a 1 into b (result)
00C0 0372  \textbf{NCGO}  rlf  ACCbLO
00C1 0373  rlf  ACCbHI

;  \texttt{divMac}
\texttt{LOCAL NOCHK}
\texttt{LOCAL NOGO}

00C2 0403  bcf  STATUS, CARRY
00C3 0376  rlf  ACCdLO
00C4 0377  rlf  ACCbHI

© 1993 Microchip Technology Inc.
PIC16C5X Math Routines

00C5 0374 rlf ACCcLO
00C6 0375 rlf ACCcHI
00C7 0211 movf ACCcHI,w
00C8 0095 subwf ACCcHI,w ;check if a>c
00C9 0743 btfss STATUS,Z_bit
00CA 0A6D goto NOCHK
00CB 0210 movf ACCcLO,w
00CC 0094 subwf ACCcLO,w ;if msb equal then check lsb
00CD 0703 NOCHK btfs STATUS,CARRY ;carry set if c>a
00CE 0A06 goto NOGO
00CF 0210 movf ACCcLO,w ;c-a into c
00D0 0084 subwf ACCcHI
00D1 0703 btfs STATUS,CARRY
decf ACCcLO
00D2 00F5 movf ACCcHI,w
00D3 0211 subwf ACCcHI
00D5 0503 bsf STATUS,CARRY ;shift a 1 into b (result)
00D6 0372 NOGO rlf ACCbLO
00D7 0373 rlf ACCbHI

; divMac
LOCAL NOCHK
LOCAL NOGO

00D8 0403 bcf STATUS,CARRY
00D9 0376 rlf ACCbLO
00DA 0377 rlf ACCbHI
00DB 0374 rlf ACCbLO
00DC 0375 rlf ACCbHI
00DD 0211 movf ACCcHI,w
00DE 0095 subwf ACCcHI,w ;check if a>c
00DF 0743 btfs STATUS,Z_bit
00E0 0A63 goto NOCHK
00E1 0210 movf ACCcLO,w
00E2 0094 subwf ACCcLO,w ;if msb equal then check lsb
00E3 0703 NOCHK btfs STATUS,CARRY ;carry set if c>a
00E4 0A0C goto NOGO
00E5 0210 movf ACCcLO,w ;c-a into c
00E6 0084 subwf ACCcHI
decf ACCcLO
00E7 0703 btfs STATUS,CARRY
00E8 00F5 movf ACCcHI,w
00E9 0211 subwf ACCcHI
00EA 0085 bsf STATUS,CARRY ;shift a 1 into b (result)
00EC 0372 NOGO rlf ACCcLO
00ED 0373 rlf ACCcHI

; divMac
LOCAL NOCHK
LOCAL NOGO

00EE 0403 bcf STATUS,CARRY
00EF 0376 rlf ACCbLO
00F0 0377 rlf ACCbHI
00F1 0374 rlf ACCbLO
00F2 0375 rlf ACCbHI
00F3 0211 movf ACCcHI,w
00F4 0095 subwf ACCcHI,w ;check if a>c
00F5 0743 btfs STATUS,Z_bit
00F6 0A69 goto NOCHK
00F7 0210 movf ACCcLO,w
00F8 0094 subwf ACCcLO,w ;if msb equal then check lsb
00F9 0703 NOCHK btfs STATUS,CARRY ;carry set if c>a
00FA 0802 goto NOGO
00FB 0210 movf ACCcLO,w ;c-a into c
00FC 0084 subwf ACCcLO
00FD 0703 btfs STATUS,CARRY
PIC16C5X Math Routines

```assembly
00FE 00F5 decf ACCCHI,
00FF 0211 movf ACCAHI,w
0100 0085 subwf ACCCHI
0101 0503 bsf STATUS,CARRY ;shift a 1 into b (result)
0102 0372 NOGO rlf ACCBL0
0103 0373 rlf ACCBHI

; divMac
LOCAL NOCHK
LOCAL NOGO

0104 0403 bcf STATUS,CARRY
0105 0376 rlf ACCBL0
0106 0377 rlf ACCdHI
0107 0374 rlf ACCcLO
0108 0375 rlf ACCcHI
0109 0211 movf ACCAHI,w
010A 0095 subwf ACCCHI,w ;check if a>c
010B 0743 btfs STATUS,Z_bit
010C 080F goto NOCHK
010D 0210 movf ACCALO,w
010E 0094 subwf ACCcLO,w ;if msb equal then check lsb
010F 0703 NOCHK btfs STATUS,CARRY ;carry set if c>a
0110 0818 goto NOGO
0111 0210 movf ACCALO,w ;c-a into c
0112 0084 subwf ACCcLO,w
0113 0703 btfs STATUS,CARRY
0114 00F5 decf ACCcHI
0115 0211 movf ACCAHI,w
0116 00B5 subwf ACCcHI
0117 0503 bsf STATUS,CARRY ;shift a 1 into b (result)
0118 0372 NOGO rlf ACCBLO
0119 0373 rlf ACCBHI

; divMac
LOCAL NOCHK
LOCAL NOGO

011A 0403 bcf STATUS,CARRY
011B 0376 rlf ACCdLO
011C 0377 rlf ACCcLO
011D 0374 rlf ACCcHI
011E 0375 rlf ACCcHI
011F 0211 movf ACCAHI,w
0120 0095 subwf ACCCHI,w ;check if a>c
0121 0743 btfs STATUS,Z_bit
0122 0825 goto NOCHK
0123 0210 movf ACCALO,w
0124 0094 subwf ACCcLO,w ;if msb equal then check lsb
0125 0703 NOCHK btfs STATUS,CARRY ;carry set if c>a
0126 082E goto NOGO
0127 0210 movf ACCALO,w ;c-a into c
0128 00B4 subwf ACCcLO
0129 0703 btfs STATUS,CARRY
012A 00F5 decf ACCcHI
012B 0211 movf ACCAHI,w
012C 00B5 subwf ACCcHI
012D 0503 bsf STATUS,CARRY ;shift a 1 into b (result)
012E 0372 NOGO rlf ACCBLO
012F 0373 rlf ACCBHI

; divMac
LOCAL NOCHK
LOCAL NOGO

0130 0403 bcf STATUS,CARRY
```
```assembly
0131 0376      rlf    ACCdLO
0132 0377      rlf    ACCdHI
0133 0374      rlf    ACCcLO
0134 0375      rlf    ACCcHI
0135 0211      movf   ACCAHI,w
0136 0095      subwf   ACCAHI,w       ;check if a>c
0137 0743      btfss   STATUS, Z_bit
0138 0838      goto   NOCHK
0139 0210      movf   ACCcLO,w
013A 0094      subwf   ACCcLO,w       ;if msb equal then check lsb
013B 0070      btfss   STATUS, CARRY
013C 0844      goto   NOGO
013D 0210      movf   ACCcHI,w
013E 008B      subwf   ACCcHI
013F 0703      btfss   STATUS, CARRY
0140 00F5      decf   ACCcHI
0141 0211      movf   ACCcHI,w
0142 0085      subwf   ACCcHI
0143 0503      bsf     STATUS, CARRY
0144 0372      rlf     ACCcLO
0145 0373      rlf     ACCcHI

; divMac
LOCAL      NOCHK
LOCAL      NOGO

0146 0403      bcf     STATUS, CARRY
0147 0376      rlf     ACCdLO
0148 0377      rlf     ACCdHI
0149 0374      rlf     ACCcLO
014A 0375      rlf     ACCcHI
014B 0211      movf   ACCAHI,w
014C 0095      subwf   ACCAHI,w       ;check if a>c
014D 0743      btfss   STATUS, Z_bit
014E 0851      goto   NOCHK
014F 0210      movf   ACCcLO,w
0150 0094      subwf   ACCcLO,w       ;if msb equal then check lsb
0151 0703      btfss   STATUS, CARRY
0152 085A      goto   NOGO
0153 0210      movf   ACCcLO,w       ;c-a into c
0154 008B      subwf   ACCcLO
0155 0703      btfss   STATUS, CARRY
0156 00F5      decf   ACCcHI
0157 0211      movf   ACCcHI,w
0158 0085      subwf   ACCcHI
0159 0503      bsf     STATUS, CARRY
015A 0372      rlf     ACCcLO
015B 0373      rlf     ACCcHI

; divMac
LOCAL      NOCHK
LOCAL      NOGO

015C 0403      bcf     STATUS, CARRY
015D 0376      rlf     ACCdLO
015E 0377      rlf     ACCdHI
015F 0374      rlf     ACCcLO
0160 0375      rlf     ACCcHI
0161 0211      movf   ACCAHI,w
0162 0095      subwf   ACCAHI,w       ;check if a>c
0163 0743      btfss   STATUS, Z_bit
0164 0857      goto   NOCHK
0165 0210      movf   ACCcLO,w
0166 0094      subwf   ACCcLO,w       ;if msb equal then check lsb
0167 0703      btfss   STATUS, CARRY
0168 0870      goto   NOGO
0169 0210      movf   ACCcLO,w       ;c-a into c
```
PIC16C5X Math Routines

016A 00B4 subwf ACCcLO
016B 0703 btfss STATUS, CARRY
016C 00F5 decf ACCcHI
016D 0211 movf ACCcHI,w
016E 00B5 subwf ACCcHI
016F 0503 bsf STATUS, CARRY ; shift a 1 into b (result)
0170 0372 NOGO rlf ACCcLO
0171 0373 rlf ACCcHI

; IF SIGNED
0172 0800 btfss sign, MSB ; check sign if negative
retlw 0
; goto neg_B ; negate ACCa ( -ACCa -> ACCa )
ELSE
retlw 0
ENDIF

; Assemble this section only if Signed Arithmetic Needed
; IF SIGNED
$SIGN movf ACCcHI, W
xorwf ACCcHI, W
movwf sign
btfss ACCcHI, MSB ; if MSB set go & negate ACCb
goto chek_A
;
comf ACCcLO ; negate ACCb
incf ACCcLO
btfsc STATUS, Z_bit
decf ACCcHI
comf ACCcHI
;
chek_A btfss ACCcHI, MSB ; if MSB set go & negate ACCa
retlw 0
; goto neg_A
;
ENDIF
;
; Test Program
; Load constant values to ACCa & ACCb for testing
;
0173 0C01 main movlw 1
0174 0031 movwf ACCcHI
0175 0CFF movlw 0FF ; loads ACCa = 01FF
0176 0030 movwf ACCcLO
;
0177 0C7F movlw 07F
0178 0033 movwf ACCcHI
0179 0CFF movlw 0FF ; loads ACCb = 07FF
017A 0032 movwf ACCcLO
017B 090F call D_divF ; remainder in ACCc. Here ACCc = 0040 &
ACCc=003F
;
017C 0B7C self goto self
;
sel org PIC54
LIST p=16c54
01FF 0B73 goto main
END

Errors : 0
Warnings : 0
APPENDIX G: DOUBLE PRECISION ADDITION AND SUBTRACTION LISTING

PAGE 1

;*******************************************************************;
; Double Precision Addition & Subtraction
;*******************************************************************;
; Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
; (c) CALL D_add
; (d) The result is in location ACCbLO & ACCbHI (16 bits)
;
; Performance :
; Program Memory : 07
; Clock Cycles : 08
;*******************************************************************;

; Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
; (c) CALL D_sub
; (d) The result is in location ACCbLO & ACCbHI (16 bits)
;
; Performance :
; Program Memory : 14
; Clock Cycles : 17
;*******************************************************************;

LIST P=16C54
0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 ACCbLO equ 12
0013 ACCbHI equ 13

#include "mpreg.h"

;************************** PIC16C5X Header **************************
01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

; 0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h

; 0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

;**********************************************

; STATUS REG. Bits
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 P equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h

;
PIC16C5X Math Routines

Same equ 1h
; LSB equ 0h
MSB equ 7h
;
TRUE equ 1h
YES equ 1h
FALSE equ 0h
NO equ 0h
;
;*************************************************************************
or 0
;*************************************************************************
org 0
;*************************************************************************
Double Precision Subtraction ( ACCb - ACCa -> ACCb )
;
0000 0908 D_sub call neg_A ; At first negate ACCa; Then add
;*************************************************************************
Double Precision Addition ( ACCb + ACCa -> ACCb )
;
0001 0210 # D_add movf ACCaLo, w
0002 01F2 addwf ACCbLo ;add lsb
0003 0603 btsc STATUS, CARRY ;add in carry
0004 02B3 incf ACCbHi
0005 0211 movf ACCaHi, w
0006 01F3 addwf ACCbHi ;add msb
0007 0800 retlw 0

neg_A comf ACCaLo ; negate ACCa ( -ACCa -> ACCa )
0008 0270 incf ACCaLo
0009 02B0 btsc STATUS, Z_bit
000A 0643 decf ACCaHi
000B 00F1 comf ACCaHi
000C 0271 retlw 0
000D 0800
;
;*************************************************************************
Test Program
;*************************************************************************
Load constant values to ACCa & ACCb for testing
;
000E 0C01 loadA8 movlw 1
000F 0031 movwf ACCaHi
0010 0CFF movlw 0FF ; loads ACCa = 01FF
0011 0030 movwf ACCaLo
;
0012 0C7F movlw 07F
0013 0033 movwf ACCbHi
0014 0CFF movlw 0FF ; loads ACCb = 7FFF
0015 0032 movwf ACCbLo
0016 0800 retlw 0
0017 0000
main nop
;
0018 090E call loadA8 ; result of adding ACCb+ACCa->ACCb
0019 0901 call D_add ; Here Accb = 81FE
;
001A 090E call loadA8 ; result of subtracting ACCb - ACCa->ACCb
001B 0900 call D_sub ; Here Accb = 7E00
;
001C 0A1C self goto self
;
01FF 0A17 org PIC54
2-193
APPENDIX H: FLOATING POINT ROUTINES LISTING

PIC16C5X Math Routines

MPASM B0.54

PAGE 1

:************************** Binary Floating Point Addition, Subtraction, and Multiplication routines: 

:**************************

; Addition: ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits) with the 8 bit exponent in EXPa.
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits) with the 8 bit exponent in EXPb.
; (c) CALL F_add
; (d) The result is in location ACCbLO & ACCbHI (16 bits) with the 8 bit exponent in EXPb.

; Program Memory: 55 locations

;*****************************

; Subtraction: ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits) with the 8 bit exponent in EXPa.
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits) with the 8 bit exponent in EXPb.
; (c) CALL F_sub
; (d) The result is in location ACCbLO & ACCbHI (16 bits) with the 8 bit exponent in EXPb.

; Program Memory: 61 locations

;*****************************

; Multiplication:
; ACCb(16 bits)EXP(b) * ACCa(16 bits)EXPa -> ACCb(16 bits)EXPb
; where, EXP(x) represents an 8 bit exponent.
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits) with an 8 bit exponent in location EXPa.
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits) with an 8 bit exponent in location EXPb.
; (c) CALL F_mpy
; (d) The 16 bit result overwrites ACCb(ACCbLO & ACCbHI). The exponent is stored in EXPb and the results are normalized.

; NOTE: If one needs to get a 32 bit product ( & an 8 bit exponent ), reassemble the program after changing the line " Mode16 equ TRUE " to " Mode16 equ FALSE ". If this option is chosen, then the 32 bit result is returned in ACCbHI, ACCbLO, ACCcHI, ACCcLO and the 8 bit exponent in EXPb. This method ( with " Mode16 equ FALSE " ) is NOT Recommended.

; Program Memory: 102 locations

;*****************************

LIST n=0,p-16C54

0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 EXPa equ 12
0013 ACCbLO equ 13
0014 ACCbHI equ 14
0015 EXPb equ 15
PIC16C5X Math Routines

#include "mpreg.h"

;************************** PIC16C5X Header

PIC16C5X Math Routines

0016 ACCcLO equ 16
0017 ACCcHI equ 17
0018 ACCdLO equ 18
0019 ACCdHI equ 19
001A temp equ 1A
001B sign equ 1B

; include "mpreg.h"

;************************** PIC16C5X Header

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

; 0001 RTCC equ 1
0002 PC equ 2
0003 STATUS equ 3 ; F3 Reg is STATUS Reg.
0004 FSR equ 4

; 0005 Port_A equ 5
0006 Port_B equ 6 ; I/O Port Assignments
0007 Port_C equ 7

; ; STATUS REG. Bits

0000 CARRY equ 0 ; Carry Bit is Bit.0 of F3
0000 C equ 0
0001 DCARRY equ 1
0001 DC equ 1
0002 Z.bit equ 2 ; Bit 2 of F3 is Zero Bit
0002 Z equ 2
0003 P_DOWN equ 3
0003 P_DOWN equ 3
0003 PD equ 3
0004 T_OUT equ 4
0004 TO equ 4
0005 PA0 equ 5
0006 PA1 equ 6
0007 PA2 equ 7

; 0001 Same equ 1
0000 W equ 0

; 0000 LSB equ 0
0007 MSB equ 7

; 0001 TRUE equ 1
0001 YES equ 1
0000 FALSE equ 0
0000 NO equ 0

; ;************************** PIC16C5X Header

; Model16 equ TRUE ; Change this to FALSE if a 32 bit product
; is desired for F_mpy routine.
;
; org 0

; Floating Point Subtraction ( ACCb - ACCa -> ACCb )

; 0000 0968 F_sub call neg_A ; At first negate ACCa; Then add
;
; 0001 0212 F_add movf EXPa,w ; scale mantissas
0002 0095 subwf EXPb,w ; find the greater exponent
0003 0643 btfsc STATUS,Z_bit
0004 0A0E goto pad
; exponents are equal
0005 0693 btfsc STATUS,CARRY ;
```assembly
; PIC16C5X Math Routines

0006 0986  call F_swap ; if A > B then swap ( A<>B )
0007 0212  movf EXPa,w
0008 00B5  subwf EXPb
0009 0920  scloop call shftSR
000A 03F5  infsz EXPb
000B 0A09  goto scloop
000C 0212  movf EXPa,w
000D 0035  movwf EXPb
000E 0211  padd movf ACCaHI,w
000F 0114  iorwf ACCbHI,w
0010 003B  movf sign
0011 0919  call D_add ; compute double precision integer add
0012 07FB  btfss sign,MSB
0013 07F4  retlw 0
0014 0800  bcf STATUS,CARRY
0015 0403  goto shftR
0016 02B5  incf EXPb

0017 0A23  ;*******************************************************
0018 0968  ; Double Precision Subtraction ( ACCb - ACCa -> ACCb )
0019 0210  D_sub call neg_A ; At first negate ACCa; Then add
001A 01F3  D_add movf ACCaLO,w ; Addition ( ACCb + ACCa -> ACCb )
001B 0603  addwf ACCbLO ;add lsb
001C 02B4  btfsc STATUS,CARRY ;add in carry
001D 0211  incf ACCbHI
001E 01F4  movf ACCaHI,w
001F 0800  addwf ACCbHI ;add msb

0020 0403  retlw 0
0021 06F4  shftSR bcf STATUS,CARRY
0022 0503  btfsc ACCbHI,MSB
0023 0334  shftR rrf ACCbHI ; set carry if < 0
0024 0333  rrf ACCbLO
0025 0800  retlw 0

; 0026 0403  shftSL bcf STATUS,CARRY
; if Model6
0027 0376  rlf ACCcLO
0028 0377  rlf ACCcHI
endif

0029 0373  rlf ACCbLO
002A 0374  rlf ACCbHI
002B 04F4  bcf ACCbHI,MSB
002C 0603  btfsc STATUS,CARRY
002D 05F4  bcf ACCbHI,MSB
002E 0800  retlw 0

;*******************************************************
; Binary Floating Point Multiplication :
; ACCb(16 bits)EXP(b) * ACCa(16 bits)EXPa -> ACCb(16 bits)EXPb
; F_mpy
002F 096E  call S_SIGN
0030 095F  call setup
0031 0403  mloop bcf STATUS,CARRY ; clear carry bit ?????????
0032 0339  rrf ACCcHI ;rotate d right
0033 0338  rrf ACCcLO
0034 0603  btfsc STATUS,CARRY ;need to add?
0035 0919  call D_add
0036 0334  rrf ACCbHI
0037 0333  rrf ACCbLO
```

© 1993 Microchip Technology Inc.
PIC16C5X Math Routines

0038 0337  rrf  ACCcHI
0039 0336  rrf  ACCcLO
003A 02FA  decfsz temp  ;loop until all bits checked
003B 0A31  goto  mloop
    movf  EXPa,w
003C 0212  addwf  EXPb
    ; IF Model6
003D 01F5  movf  ACCbHI
003E 0234  btfs  STATUS,Z_bit
003F 0743  goto  finup  ; if ACCbHI != 0
0040 A51  movf  ACCbLO
0041 0233  btfs  STATUS,Z_bit
0042 0743  goto  Shift08  ; if ACCbLO != 0 && ACCbHI == 0
0043 0A4B  movf  ACCcHI,w
0044 0217  movwf  ACCbHI  ; if ACCb == 0, then move ACCc to ACCb
0045 0034  movf  ACCcLO,w
0046 0216  movwf  ACCbLO
0047 0033  movlw  .16
0048 0C10  addwf  EXPb
0049 01F5  goto  finup
004A OA51  ; Shift08 movf  ACCbLO,w
004B 0213  movwf  ACCcHI
004C 0034  movwf  ACCcLO
004D 0217  movf  ACCcHI,w
004E 0033  movwf  ACCbLO
004F OC08  movlw  .8
0050 01F5  addwf  EXPb
    ; ENDIF  ; matching endif for IF Model6
0051 07FB  finup  btfs  sign,MSB
0052 0A7B  goto  F_norm
    ;
0053 00F6  decf  ACCcLO  ; negate ACCc
0054 0276  comf  ACCcLO
0055 0643  btfs  STATUS,Z_bit
0056 00F7  decf  ACCcHI
0057 0277  comf  ACCcHI
0058 0643  btfs  STATUS,Z_bit
    ;
0059 00F3  neg_B  decf  ACCbLO  ; negate ACCb
005A 0273  comf  ACCbLO
005B 0643  btfs  STATUS,Z_bit
005C 00F4  decf  ACCbHI
005D 0274  comf  ACCbHI
    ;
005E OA7B  goto  F_norm
    ;
    ;******************************************************************************
    ;
005F OC10  setup  movlw  .16  ; for 16 shifts
0060 003A  movwf  temp
0061 0214  movf  ACCbHI,w  ;move ACCb to ACCd
0062 0039  movwf  ACCbHI
0063 0213  movf  ACCbLO,w
0064 0038  movwf  ACCdLO
0065 0074  cclf  ACCbHI
0066 0073  cclf  ACCbLO  ; clear ACCb ( ACCbLO & ACCbHI )
0067 0800  retlw  0
    ;******************************************************************************
    ;
0068 0270  neg_A  comf  ACCaLO  ; negate ACCa ( -ACCa -> ACCa )
0069 02B0  incf  ACCaLO
PIC16C5X Math Routines

006A 0643  btfsc STATUS,Z_bit
006B 00F1  decf ACCaHI
006C 0271  comm ACCaHI
006D 0800  retlw 0
;
;*******************************************************************
;
006E 0211  S_SIGN movf ACCaHI,W
006F 0194  xorwf ACCbHI,W
0070 003B  movwf sign
0071 07F4  btfss ACCbHI,MSB ; if MSB set go & negate ACCb
0072 0A78  goto chek_A
;
0073 0273  comf ACCbLO ; negate ACCb
0074 02B3  incf ACCbLO
0075 0643  btfsc STATUS,Z_bit
0076 00F4  decf ACCbHI
0077 0274  comf ACCbHI
;
0078 07F1  chek_A btfss ACCaHI,MSB ; if MSB set go & negate ACCa
0079 0800  retlw 0
007A 0A68  goto neg_A
;
;*******************************************************************
;
Normalize Routine
; Normalizes ACCb for use in floating point calculations.
; Call this routine as often as possible to minimize the loss
; of precision. This routine normalizes ACCb so that the
; mantissa is maximized and the exponent minimized.
;
;
007B 0234  F_norm movf ACCbHI
007C 0743  btfss STATUS,Z_bit
007D 0A81  goto C_norm
007E 0233  movf ACCbLO
007F 0643  btfsc STATUS,Z_bit
0080 0800  retlw 0
0081 06D4  C_norm btfsc ACCbHI,6
0082 0800  retlw 0
0083 0926  call shftSL
0084 00F5  decf EXPb
0085 0A81  goto C_norm
;
;*******************************************************************
;
Attach ACCa & ACCb [ (ACCa,EXPa) <-> (ACCb,EXPb) ]
;
F_swap
0086 0211  movf ACCaHI,w
0087 003A  movwf temp
0088 0214  movf ACCbHI,w ;ACCaHI <-> ACCbHI
0089 0031  movwf ACCaHI
008A 021A  movf temp,w
008B 0034  movwf ACCbHI
;
008C 0210  movf ACCaLO,w
008D 003A  movwf temp
008E 0213  movf ACCbLO,w ;ACCaLO <-> ACCbLO
008F 0030  movwf ACCaLO
0090 021A  movf temp,w
0091 0033  movwf ACCbLO
;
0092 0212  movf EXPa,w
0093 003A  movwf temp
0094 0215  movf EXPb,w ;EXPa <-> EXPb
0095 0032  movwf EXPa
0096 021A  movf temp,w
0097 0035  movwf EXPb
;
PIC16C5X Math Routines

retlw 0
;
;******************************************************************************
; Test Program
;******************************************************************************
; Load constant values to (ACCa, EXPa) & (ACCb, EXPb) for testing
;
loadAB movlw 1
movwf ACCaHI
movlw 0F
movwf ACCaLO
movlw 04
movwf EXPa
movlw 07F
movwf ACCbHI
movlw 0F
movwf ACCbLO
movlw 06
movwf EXPb
retlw 0

main nop
; call loadAB ; result of adding b)+ACCa(EXPa)-
call F_add ; Here Accb = 403F, EXPb = 07
;
call loadAB ; result of subtracting b(EXPb)-
call F_sub ; Here Accb = 7F7F, EXPb = 06
;
call loadAB ; result of multiplying b(EXPb) *
call F_mpy ; Here Accb = FF7E, EXPb = 12
;
self goto self
;
org PIC54
org PIC54

END

Errors : 0
Warnings : 0
APPENDIX I: BCD TO BINARY CONVERSION LISTING

This routine converts a 5 digit BCD number to a 16 bit binary number. The input 5 digit BCD numbers are assumed to be in locations R0, R1 & R2 with R0 containing the MSD in its right most nibble. The 16 bit binary number is output in registers H_byte & L_byte (high byte & low byte respectively).

The method used for conversion is:
Input number X = abcd (the 5 digit BCD number)

\[ X = abcd = 10[10[10a+b]+c+d]+e \]

Performance:
Program Memory: 30
Clock Cycles: 121

LIST p=16C54

H_byte equ 1d
L_byte equ 1e
R0 equ 1f
R1 equ 1g
R2 equ 1h
H_temp equ 1i
L_temp equ 1j

INCLUDE "mpreg.h"

PIC54 equ 1ffh ; Define Reset Vectors
PIC55 equ 2ffh
PIC56 equ 3ffh
PIC57 equ 4ffh
RTCC equ 1h
PC equ 2h
STATUS equ 3h ; F3 Reg is STATUS Reg.
FSR equ 4h
Port_A equ 5h
Port_B equ 6h ; I/O Port Assignments
Port_C equ 7h

CARRY equ 0h ; Carry Bit is Bit.0 of F3
C equ 0h
DCARRY equ 1h
DC equ 1h
Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
Z equ 2h
P_DOWN equ 3h
PD equ 3h
T_OUT equ 4h
TO equ 4h
PA0 equ 5h
PA1 equ 6h
PIC16C5X Math Routines

0007    PA2    equ    7h
0001    Same    equ    1h
0000    LSB    equ    0h
0007    MSB    equ    7h
0001    TRUE    equ    1h
0001    YES    equ    1h
0000    FALSE    equ    0h
0000    NO    equ    0h

;*****************************************?*******************************

0000    OEOF    mpylOb    andlw    0F
0001    01F1    addwf    L_byte
0002    0603    btfsc    STATUS,CARRY
0003    02B0    incf    H_byte
0004    0403    mpyl0a    bcf    STATUS,CARRY    ; multiply by 2
0005    0351    rlf    L_byte,w
0006    0336    movwf    L_temp
0007    0350    rlf    H_byte,w    ; (H_temp,L_temp) = 2*N
0008    0335    movwf    H_temp

0009    0403    bcf    STATUS,CARRY    ; multiply by 2
000A    0371    rlf    L_byte
000B    0370    rlf    H_byte
000C    0403    bcf    STATUS,CARRY    ; multiply by 2
000D    0371    rlf    L_byte
000E    0370    rlf    H_byte
000F    0403    bcf    STATUS,CARRY    ; multiply by 2
0010    0371    rlf    L_byte
0011    0370    rlf    H_byte    ; (H_byte,L_byte) = 8*N

0012    0216    movf    L_temp,w
0013    01F1    addwf    L_byte
0014    0603    btfsc    STATUS,CARRY
0015    02B0    incf    H_byte
0016    0215    movf    H_temp,w
0017    01F0    addwf    H_byte
0018    0800    retlw    0    ; (H_byte,L_byte) = 10*N

0019    0070    BCDtoB    clrf    H_byte
001A    0212    movf    R0,w
001B    00F0    andlw    0F
001C    0031    movwf    L_byte
001D    0904    call    mpyl0a    ; result = 10a+b
001E    0393    swapf    R1,w
001F    0900    call    mpyl0b    ; result = 10(10a+b)

0020    0213    movf    R1,w
0021    0900    call    mpyl0b    ; result = 10(10(10a+b)+c)

0022    0394    swapf    R2,w
0023    0900    call    mpyl0b    ; result = 10(10(10(10a+b)+c)+d)

0024    0214    movf    R2,w
0025    00F0    andlw    0F
0026    01F1    addwf    L_byte
0027    0603    btfsc    STATUS,CARRY
0028    02B0    incf    H_byte    ; result = 10(10(10(10(10a+b)+c)+d)+e
0029    0800    retlw    0    ; BCD to binary conversion done

© 1993 Microchip Technology Inc.
Test Program

;** Test Program

main movlw 06
   movwf R0 ; Set R0 = 06

movlw 55
   movwf R1 ; Set R1 = 55

movlw 35
   movwf R2 ; Set R2 = 35 (R0, R1, R2 = 6, 55, 35)

; call BCDtoB ; After conversion H_Byte = FF & L_Byte = FF

self goto self

org 1FF

goto main

END

Errors : 0
Warnings : 0
LIST p=16c54,n=0

; Binary To BCD Conversion Routine
; This routine converts the 8 bit binary number in the W Register
; to a 2 digit BCD number.
; The least significant digit is returned in location LSD and
; the most significant digit is returned in location MSD.
;
; Performance :
; Program Memory : 10
; Clock Cycles : 81 (worst case when W = 63 Hex )
; ( i.e max Decimal number 99 )

#include "mpreg.h"

;************************** PIC16C5X Header

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH
;
0001 RTCC equ 1
0002 PC equ 2
0003 STATUS equ 3 ; F3 Reg is STATUS Reg.
0004 FSR equ 4
;
0005 Port_A equ 5
0006 Port_B equ 6 ; I/O Port Assignments
0007 Port_C equ 7
;
; STATUS REG. Bits
0000 CARRY equ 0 ; Carry Bit is Bit.0 of F3
0000 C equ 0
0001 DCARRY equ 1
0001 DC equ 1
0002 Z_bit equ 2 ; Bit 2 of F3 is Zero Bit
0002 Z equ 2
0003 P_DOWN equ 3
0003 PD equ 3
0004 T_OUT equ 4
0004 TO equ 4
0005 PA0 equ 5
0006 PA1 equ 6
0007 PA2 equ 7
;
0001 Same equ 1
0000 W equ 0
;
0000 LSB equ 0
0007 MSB equ 7
;
0001 TRUE equ 1
PIC16C5X Math Routines

0001    YES       equ       1
0000    FALSE     equ       0
0000    NO        equ       0

*************************************************************************
BinBCD  clrf  MSD
movwf   LSD
gtenth  movlw  .10
subwf   LSD,W
BTFSS   STATUS,CARRY
goto    over
movwf   MSD
incf    MSD
goto    gtenth
over    retlw  0
*************************************************************************
main    movlw  63  ; W reg = 63 Hex
          call   BinBCD  ; after conversion, MSD = 9 & LSD = 9
self    goto   self  ; ( 63 Hex = 99 Decimal )
org     1FF
goto    main
END

Errors   :  0
Warnings :  0
APPENDIX K: BINARY (16-BIT) TO BCD LISTING

; Binary To BCD Conversion Routine
; This routine converts a 16 Bit binary Number to a 5 Digit
; BCD Number. This routine is useful since PIC16C55 & PIC16C57
; have two 8 bit ports and one 4 bit port (total of 5 BCD digits)
;
; The 16 bit binary number is input in locations H_byte and
; L_byte with the high byte in H_byte.
; The 5 digit BCD number is returned in R0, R1 and R2 with R0
; containing the MSD in its right most nibble.
;
; Performance:
; Program Memory: 35
; Clock Cycles: 885

;***************************************************************************;
LIST P=16CS4

COUNT equ 16
TEMP equ 17

H_BYTE equ 10
L_BYTE equ 11
R0 equ 12
R1 equ 13
R2 equ 14

include "mpreg.h"

PICS4 equ lFFH
PICSS equ lFFH
PICS6 equ 3FFH
PICS7 equ 7FFH
RTCC equ lh
PC equ 2h
STATUS equ 3h
FSR equ 4h
Port A equ Sh
Port B equ 6h
Port C equ 7h

STA equ 0h
CARRY equ 0h
C equ 0h
DCARRY equ 1h
DC equ 1h
Z_BIT equ 2h
Z equ 2h
P_DOWN equ 3h
PD equ 3h
T_OUT equ 4h
T0 equ 4h
PA0 equ 5h
PA1 equ 6h
PA2 equ 7h
Same equ 1h
LSB equ 0h
PIC16C5X Math Routines

0007 MSB equ 7h
0001 TRUE equ 1h
0001 YES equ 1h
0000 FALSE equ 0h
0000 NO equ 0h

0000 0403 B2 BCD bcf STATUS,0 ; clear the carry bit
0001 0C10 movlw .16
0002 0036 movwf count
0003 0072 clrf RO
0004 0073 clrf R1
0005 0074 clrf R2
0006 0371 loop16 rlf L_byte
0007 0370 rlf H_byte
0008 0374 rlf R2
0009 0373 rlf R1
000A 0372 rlf R0
000B 02F6 decf sz count
000C 00AE goto adj DEC
000D 0800 RETLW 0
000E OC14 ; adjDEC movlw R2
000F 0024 movwf FSR
0010 0918 call adjBCD
0011 0C13 movlw R1
0012 0024 movwf FSR
0013 0918 call adjBCD
0014 0C12 movlw R0
0015 0024 movwf FSR
0016 0918 call adjBCD
0017 0A06 goto loop16
0018 0C03 adjBCD movlw 3
0019 01C0 addwf 0,W
001A 0037 movwf temp
001B 0677 btfsC temp,3 ; test if result > 7
001C 0020 movwf 0
001D 0C30 movlw 30
001E 01C0 addwf 0,W
001F 0037 movwf temp
0020 06F7 btfsC temp,7 ; test if result > 7
0021 0020 movwf 0 ; save as MSD
0022 0800 RETLW 0

; Test Program

0023 OCFF main movlw OFF
0024 0030 movwf H_byte
0025 0031 movwf L_byte ; The 16 bit binary number = FFFF
0026 0900 call B2_BCD ; After conversion the Decimal Number
0027 0A27 self goto self ; in R0,R1,R2 = 06,55,35
0028 0A27 org 1FF
0029 0A23 goto main

END
APPENDIX L: UNSIGNED BCD ADDITION LISTING

PIC16C5X Math Routines

MPASM B0.54

PAGE 1

;*******************************************************************;
;******************* Unsigned BCD Addition *******************
;*******************************************************************;

; This routine performs a 2 Digit Unsigned BCD Addition
; It is assumed that the two BCD numbers to be added are in
; locations Num_1 & Num_2. The result is the sum of Num_1+Num_2
; and is stored in location Num_2 and the overflow carry is returned
; in location Num_1
;
; Performance:
; Program Memory : 2b
; Clock Cycles : 23 (worst case)
; Rev 2.0 changed on 7/30/92.
;
;***************************************************************************;

LIST P=16C54

0000 Num_1 equ 8 ; Overflow flow carry overwrites Num_1
0002 result equ 8 ;
0003 Num_2 equ 9 ; Num_2 + Num_1 overwrites Num_2
0004 O_flow equ 9 ;

; include "mpreg.h"

;***************************************************************************;

PICS4 equ lFFH ; Define Reset Vectors
PICS5 equ lFFH
PICS6 equ 3FFH
PICS7 equ 7FFH

RTCC equ lh
PC equ 2h
STATUS equ 3h
FSR equ 4h
Port A equ Sh
Port B equ 6h
Port C equ 7h

;***************************************************************************;

STATUS REG. Bits
CARRY equ 0h
C equ 0h
DCARRY equ 1h
DC equ 1h
Z bit equ 2h
Z equ 2h
P_DOWN equ 3h
PD equ 3h
T_OUT equ 4h
TO equ 4h
PA0 equ 5h
PA1 equ 6h
PA2 equ 7h

Same equ 1h

LSB equ 0h
MSB equ 7h
TRUE equ 1h
YES equ 1h
FALSE equ 0h
PIC16C5X Math Routines

0000

NO equ 0h

;*************************************************************************

BCDAdd movf Num_1,w
addwf Num_2 ; do binary addition
clrf Num_1
rlf Num_1
btsc STATUS,DC ; Is DC = 0 ?
goto adjust ; adjust LSD
movlw 6
addwf Num_2 ; Test for LSD > 9 ( by adding 6
btsc STATUS,CARRY
incf Num_1
subwf Num_2 ; & checking Digit Carry
addwf Num_2 ; LSD < 9, so get back original value.
goto over1
adj ust movlw 6
addwf Num_2
movlw 60 ; add 6 to MSD
addwf Num_2
btsc STATUS,CARRY
goto over3
btss Num_1,0
subwf Num_2
RETLW 0
movlw 1
movwf Num_1
RETLW 0

;*************************************************************************

Test Program

;*************************************************************************

main movlw 99
movwf Num_1 ; Set Num_1 = 99 ( max BCD digit )
movlw 99
movwf Num_2 ; Set Num_2 = 99
; call BCDAdd ; After addition, Num_2 = 98
; and Num_1 = 01 ( 99+99 - 198 -> max number
)
;
self goto self
;
org 1FF
01FF 0A19

goto main
;
END

Errors : 0
Warnings : 0
APPENDIX M: UNSIGNED BCD SUBTRACTION LISTING

;*************************************************** Unsigned BCD Subtraction ***************************************************

; This routine performs a 2 Digit Unsigned BCD Subtraction.
; It is assumed that the two BCD numbers to be subtracted are in
; locations Num_1 & Num_2. The result is the difference of Num_1 & Num_2
; ( Num_2 - Num_1) and is stored in location Num_2 and the overflow carry
; is returned in location Num_1.
;
; Performance :
; Program Memory :  31
; Clock Cycles : 21 ( worst case )
;
;***************************************************

LIST P=16C54

0008 Num_1 equ 8 ; Overflow flow carry overwrites Num_1
0008 result equ 8

0009 Num_2 equ 9 ; Num_2 - Num_1 overwrites Num_2
0009 O_flow equ 9

;include "mpreg.h"
PIC16C5X Header ***************************************************

01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH

0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h

0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h

; STATUS REG. Bits
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h

0001 Same equ 1h

0000 LSB equ 0h
0007 MSB equ 7h

0001 TRUE equ 1h
0001 YES equ 1h
0000 FALSE equ 0h
0000 NO equ 0h

;***************************************************
PIC16C5X Math Routines

; BCDSUB movf Num_1,w
; subwf Num_2
; clrf Num_1
; rlf Num_1
; btfss STATUS,DC
; goto adjst1
; btfss Num_2,3 ; Adjust LSD of Result
; goto Over_1
; btfsc Num_2,2
; goto adjst_1 ; Adjust LSD of Result
; goto Over_1 ; No : Go for MSD
; adjst1 movlw 6
; subwf Num_2
; Over_1 btfss Num_1,0 ; CY = 0 ?
; goto adjst2 ; Yes, adjust MSD of result
; clrf Num_1
; btfss Num_2,7 ; No, test for MSD >9
; RETLW 0
; btfsc Num_2,6
; goto adjst2
; btfss Num_2,5
; RETLW 0
; adjst2 movlw 60 ; add 6 to MSD
; subwf Num_2
; clrf Num_1
; btfss STATUS,CARRY ; test if underflow
; RETLW 0
; movlw 1
; movwf Num_1
; RETLW 0
; Over RETLW 0
;
; ; Test Program
; ;********************************************************************
; main movlw 23
; movwf Num_1 ; Set Num_1 = 23
; movlw 99
; movwf Num_2 ; Set Num_2 = 99
; call BCDSUB ; After subtraction, Num_2 = 76 (99-23)
; ; ;
; movlw 99
; movwf Num_1 ; Set Num_1 = 99
; movlw 0
; movwf Num_2 ; Set Num_2 = 0
; call BCDSUB ; After subtraction, Num_2 = 1
; ; ;
; call BCDSUB ; After subtraction, Num_2 = 1
; ; ;
; self goto self
; org 1FF
; goto main
; END

Errors : 0
Warnings : 0
APPENDIX N: SQUARE ROOT BY NEWTON-RAPHSON METHOD

```
list p=16c54,f=inhx8m,n=0

;***************************************************************
; Square Root By Newton Raphson Method
;
; This routine computes the square root of a 16 bit number (with
; low byte in NumLo & high byte in NumHi). After loading NumLo &
; NumHi with the desired number whose square root is to be computed,
; branch to location Sqrt (by "GOTO Sqrt"). "CALL Sqrt" cannot
; be issued because the Sqrt function makes calls to Math routines
; and the stack is completely used up.
;
; The result = sqrt(NumHi, NumLo) is returned in location SqrtLo.
; The total number of iterations is set to ten. If more iterations
; are desired, change "LupCnt equ .10" to the desired value. Also,
; the initial guess value of the square root is given set as
; input/2 (in subroutine "init"). The user may modify this scheme
; if a better initial approximation value is known. A good initial
; guess will help the algorithm converge at a faster rate and thus
; less number of iterations required.
;
; Two utility math routines are used by this program: D_divS
; and D_add. These two routines are listed as separate routines
; under double precision Division and double precision
; addition
; respectively.
;
; Note: If square root of an 8 bit number is desired, it is probably
; better to have a table look scheme rather than using numerical
; methods.
;
; Performance:
; Program Memory : 27 (excluding Math Routines
; D_divS & D_add)
; Clock Cycles : 3600 (approximately)
;
; To assemble this program, two routines, namely "D_add" &
; "D_divS" must be included into this program. These two routines
; are listed as separate programs in files "DBL_ADD.ASM" &
; "DBL_DIVS.ASM" respectively.
;***************************************************************

ACCaLO equ 10
ACCaHI equ 11
EXPa   equ 12
ACCbLO equ 13
ACCbHI equ 14
EXPb   equ 15
ACCcLO equ 16
ACCcHI equ 17
ACCdLO equ 18
ACCdHI equ 19
temp   equ 1A
sign    equ 1B

LupCnt equ .10 ; Number of iterations
```

© 1993 Microchip Technology Inc. DS00526C-page 57
PIC16C5X Math Routines

; 0010  SqrtLo  equ  ACCaLO
; 0011  SqrtHi  equ  ACCaHI
; 001D  NumLo  equ  lD
; 001E  NumHi  equ  lE
; 001F  count  equ  1F
;
; init
0000 0C0A  movlw  LupCnt
0001 003F  movwf  count
0002 021E  movf  NumHi,W
0003 0031  movwf  SqrtHi
0004 021D  movf  NumLo,W ; set initial guess root = NUM/2
0005 0030  movwf  SqrtLo
0006 0403  bcf  STATUS,CARRY
0007 0331  rrf  SqrtHi
0008 0330  rrf  SqrtLo
0009 0800  retlw  0
;
000A 0403  div2  bcf  STATUS,CARRY
000B 0314  rrf  ACCbHI,W
000C 0031  movwf  SqrtHi
000D 0313  rrf  ACCbLO,W
000E 0030  movwf  SqrtLo
000F 0800  retlw  0
;
0010 0900  Sqrt  call  init
0011 021D  sloop  movf  NumLo,W
0012 0033  movwf  ACCbLO
0013 021E  movf  NumHi,W
0014 0034  movwf  ACCbHI
;
0015 091B  call  D_divS ; double precision division
0016 0947  call  D_add ; double precision addition
; the above 2 routines are listed
; as separate routines

0017 090A  call  div2
0018 02FF  decfsz  count
0019 0A11  goto  sloop
001A 0A53  goto  over ; all iterations done
;
; branch back to desired location
;******************************************************
; double precision division is placed here
include "d_divS.asm"

;******************************************************
; double precision addition is placed here
include "d_add.asm"

;******************************************************
;
;******************************************************
; Test Program
;******************************************************
;
004E OCF3  main  movlw  OF3
004F 003E  movwf  NumHi
0050 0CF6  movlw  OF6 ; Set input test number = 62454
0051 003D  movwf  NumLo ; = F3F6h
;
0052 0A10  goto  Sqrt ; cannot use CALL : Math routines
;
0053 0000  over  nop ; use up all the stack.
;
0054 0A54  self  goto  self ; result = 00F9h = 249
; exact sqrt(62454) = 249.9

org PIC54

goto main

Errors : 0
Warnings : 0
INTRODUCTION

This application report describes an LCD controller implementation using a PIC16C55 microcontroller. This technique offers display capabilities for applications that require a small display at a low cost together with the capabilities of the standard PIC15C55 microcontroller. We start by an overview of LCD devices and their theory of operation followed by software implementation issues of the controller. The source code for controlling a multiplexed LCD display is included in Appendix A.

LIQUID CRYSTAL DISPLAYS

The Liquid Crystal Display (LCD) is a thin layer of "Liquid Crystal Material" deposited between two plates of glass. The raw LCD is often referred to as "glass". Electrodes are attached to both sides of the glass. One side is referred to as common or backplane, while the other side is referred to as segment.

An LCD is modeled as a capacitor, with one side connected to the common plane and the other side connected to the segment, as shown in Figure 1. LCDs are sensitive to Root Mean Square Voltage levels. When a $V_{\text{RMS}}$ level of zero volts is applied to the LCD, the LCD is practically transparent.

To turn an LCD segment "on", which makes the segment turn dark or opaque, an LCD RMS voltage that is greater than the LCD threshold voltage is applied to the LCD. The RMS LCD voltage is the RMS voltage across the capacitor C in Figure 1, which is equal to the potential difference between SEG and COM values.

Different LCDs have different characteristics; Figure 2 shows typical voltage vs relative contrast characteristics. Notation on curve shows operating points for multiplex operation with the threshold voltage set to 1.7 Vrms. This voltage is often used as the measure of voltage for LCD to be "off" or transparent. The curve is normalized and assumes a viewing angle of 90° to the plane of the LCD.

Contrast control, the process of turning on a segment, is achieved by moving the operating point of the LCD by applying voltage to the LCD that is greater than the LCD threshold voltages. A typical circuit to accomplish this task is shown in Figure 3.

Driving a liquid crystal display at direct current (DC) will cause permanent damage to the display unit. In order to prevent irreversible electrochemical action from destroying the display, the voltage at all segment locations must reverse polarity periodically so that a zero net voltage is applied to the device. This process is referred to as AC voltage application. There are two LCD driving methods available: Static driving method and multiplexed driving method.

FIGURE 1: ELECTRICAL MODEL OF AN LCD SEGMENT WITH DRIVING VOLTAGES

![Electrical Model of an LCD Segment with Driving Voltages](image-url)
Using PIC16C5X Microcontrollers as LCD Drivers

Conventional LCDs have separate external connections for each and every segment plus a common plane. This is the most basic method that results in good display quality. The main disadvantage of this driving method is that each segment requires one liquid crystal driver. The static driving method uses the frame frequency, defined as a period of the common plane signal, of several tens to several hundred Hz. A lower frequency would result in blinking effects and higher frequencies would increase power requirements. To turn a segment on, a voltage that has an opposite polarity to the common plane signal must be applied resulting in a large RMS voltage across the plates. To turn off a segment a voltage that is of the same polarity to common plane signal is applied. This drive method is universal to driving LCD segments. Figure 1 shows an example of this driving method.

The LCD frequency is defined as the rate of output changes of the common plane and segment signals, whereas the frame rate is defined as

$$f_{\text{frame}} = \frac{f_{\text{frame}}}{N}$$

where N is the multiplex rate or number of backplane. Typically, $f_{\text{frame}}$ ranges from 25HZ to 300HZ. The most commonly used frame frequency is 40-70HZ. A lower frequency would result in flicker effects and higher frequency would increase power requirements.

Multiplexed LCDs maintain their liquid crystal characteristics. These are low power consumption, high contrast ratio under high ambient light levels, and reduce the number of external connections necessary for dot matrix and alphanumeric displays. The multiplex driving method reduces the number of driver circuits, or microcontroller I/O pins if a software method is used. The method of drive for multiplexed displays is Time Division Multiplex (TDM) with the number of time divisions equal to twice the number of common planes used in a given format. In order to prevent permanent damage to the LCD display, the voltage at all segment locations must reverse polarity periodically so that zero net voltage is applied. This is the reason for the doubling in time divisions; each common plane must be alternately driven with a voltage pulse of opposite polarity. The drive frequency should be greater than the flicker rate of 25Hz. Since increasing the drive frequency significantly above this value increases current demand by the CMOS circuitry, an upper drive frequency level of 60Hz is recommended by most LCD manufacturers. We have chosen a drive rate of 50Hz for this application report which results in a frame period of 20 ms. The most commonly available formats are 2x4, 3x3, and 5x7. In this report we use a 2x4 format LCD to display hexadecimal digits.

**FIGURE 2: TYPICAL LCD CHARACTERISTICS**

<table>
<thead>
<tr>
<th>Drive Voltage Vrms</th>
<th>% Relative Contrast</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.7 Vrms</td>
<td>100%</td>
</tr>
<tr>
<td>V = 4.1 Vrms</td>
<td></td>
</tr>
</tbody>
</table>

Two Common Planes

**FIGURE 3: CONTRAST CONTROL CIRCUIT**
Using PIC16C5X Microcontrollers as LCD Drivers

To better understand multiplexed LCD control it is best to look at the general case. The segments in a multiplexed LCD are arranged in an X-Y grid form as shown in Figure 4. The common plane signals maintain their relative shape at all times, as shown in Figure 5. To turn on segment 1 (SEG1), we need to apply a voltage Vd, such that Vs+Vd turns the segment on and Vs-Vd turns the segment off. Note that the segment signal Vd is symmetrical. This is a consequence of the intervals that the common plane signal is not present at all times. Use of nonsymmetrical waveform will result in a higher Vrms present on the unaddressed segments. The symmetrical nature of the waveforms theoretically result in a zero DC voltage levels. CMOS drivers (e.g. microcontrollers) operate at 0 to +5V levels (rail voltage levels). This would require driving voltages beyond the range of operation. This constraint is addressed by a technique referred to as “level shifting” or “biasing”. Level shifting allows application of voltages in the range of 0 to +2.5V, which is compatible with these drivers. This would require an additional voltage level of +2.5V, which can be implemented through a simple resistive voltage divider circuit.

FIGURE 4: MULTIPLEXED LCD SEGMENT ARRANGEMENT

FIGURE 5: MULTIPLEXED LCD DRIVE WAVEFORMS
IMPLEMENTATION

The ideas presented in the previous section can be applied to any size multiplexed LCD display. In our implementation we used a 4-digit LCD from Ocular Inc. [1]. The circuit diagram used in this application report is shown in Figure 6. Each I/O pin on the PIC16C55 device controls the state of two segments (see Figure 6) which requires a total of 16 I/O pins. The reference voltages are generated through a simple resistive voltage divider circuit. The voltage levels are generated by taking advantage of PIC16C5X I/O pin set to input, which tristates the voltage level seen on the pin. This method uses 4 I/O pins to generate the proper voltage levels. Figure 7 shows the truth table for generating the voltage levels. Figure 8 shows how to create a bitmap for different digits. Figure 9 shows the waveforms generated for the accompanying software which implements a hexadecimal counter.

FIGURE 6: SYSTEM CONFIGURATION WITH LCD PINOUT

FIGURE 7: COMMON PLANE SIGNAL GENERATION

FIGURE 8: LCD CHARACTER BITMAP

<table>
<thead>
<tr>
<th>Digit</th>
<th>COM0 SEG 0</th>
<th>COM1 SEG 1</th>
<th>COM0 SEG 2</th>
<th>COM1 SEG3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F  E  D  DP</td>
<td>A  G  C  B</td>
<td>F  E  D  DP</td>
<td>A  G  C  B</td>
</tr>
<tr>
<td>0</td>
<td>0  0  0  1</td>
<td>0  0  1  0</td>
<td>1  1  1  0</td>
<td>1  1  0  1</td>
</tr>
<tr>
<td>1</td>
<td>1  1  1  1</td>
<td>1  0  1  0</td>
<td>0  0  0  0</td>
<td>0  1  0  1</td>
</tr>
<tr>
<td>2</td>
<td>1  0  0  1</td>
<td>0  0  0  1</td>
<td>0  1  1  0</td>
<td>1  1  1  0</td>
</tr>
<tr>
<td>3</td>
<td>1  1  0  1</td>
<td>0  0  0  0</td>
<td>0  0  1  0</td>
<td>1  1  1  1</td>
</tr>
<tr>
<td>4</td>
<td>0  1  1  1</td>
<td>0  0  0  0</td>
<td>1  0  0  0</td>
<td>0  1  1  0</td>
</tr>
<tr>
<td>5</td>
<td>0  1  0  1</td>
<td>0  1  0  0</td>
<td>1  0  1  0</td>
<td>1  0  1  1</td>
</tr>
<tr>
<td>6</td>
<td>1  1  1  1</td>
<td>1  1  1  1</td>
<td>1  1  1  0</td>
<td>1  0  1  1</td>
</tr>
<tr>
<td>7</td>
<td>1  1  1  1</td>
<td>0  0  1  0</td>
<td>0  0  0  0</td>
<td>1  1  0  1</td>
</tr>
<tr>
<td>8</td>
<td>0  0  0  1</td>
<td>0  0  0  0</td>
<td>1  1  1  0</td>
<td>1  1  1  1</td>
</tr>
<tr>
<td>9</td>
<td>0  1  0  1</td>
<td>0  0  0  0</td>
<td>1  0  1  0</td>
<td>1  1  1  1</td>
</tr>
<tr>
<td>a</td>
<td>0  0  1  1</td>
<td>0  0  0  0</td>
<td>1  1  0  0</td>
<td>1  1  1  1</td>
</tr>
<tr>
<td>b</td>
<td>0  0  0  1</td>
<td>1  1  0  0</td>
<td>1  1  1  0</td>
<td>0  0  1  1</td>
</tr>
<tr>
<td>c</td>
<td>1  0  0  1</td>
<td>1  1  0  1</td>
<td>0  1  1  0</td>
<td>0  0  1  0</td>
</tr>
<tr>
<td>d</td>
<td>1  0  0  1</td>
<td>1  0  0  0</td>
<td>0  1  1  0</td>
<td>0  1  1  1</td>
</tr>
<tr>
<td>e</td>
<td>0  0  0  1</td>
<td>0  1  0  1</td>
<td>1  1  1  0</td>
<td>1  0  1  0</td>
</tr>
<tr>
<td>f</td>
<td>0  0  1  1</td>
<td>0  1  0  1</td>
<td>1  1  0  0</td>
<td>1  0  1  0</td>
</tr>
</tbody>
</table>
Using PIC16C5X Microcontrollers as LCD Drivers

FIGURE 9: EXAMPLE OF OUTPUT WAVEFORMS FOR DIGIT 4

CONCLUSION

In this application report we have demonstrated the use of PIC16C5X devices to implement a simple LCD controller. As discussed earlier, it is important to keep the generated DC voltage to a minimum to extend the life of the LCD. Ideally one should switch all the I/O lines simultaneously, however, a software implementation of the LCD controller will necessarily introduce a delay which is proportional to the instruction cycle of the microcontroller, as shown in Figure 10. Therefore it is necessary to keep the switching time to a minimum. Our implementation introduced less than 50 mV of DC voltage on the segment lines which is below the manufacturer's recommended DC offset voltage of 60 mV.

REFERENCES


AUTHOR: Al Lovrich
Logic Products Division
Using PIC16C5X Microcontrollers as LCD Drivers

LIST C=132,n=0,p=16C55,r=dec

;********************************************************
; Project: PIC16C5X as a multiplexed LCD driver. *
;
; Revision history:
;
; 04/14/93 original
;
;********************************************************

; Equates

01FF pic54 equ 0x1ff ; Define Reset Vectors
01FF pic55 equ 0x1ff
03FF pic56 equ 0x3ff
07FF pic57 equ 0x7ff

0001 rtcc equ 1 ; f1
0002 pc equ 2 ; f2
0003 status equ 3 ; f3
0004 fsr equ 4 ; f4

0005 porta equ 5 ; f5
0006 portb equ 6 ; f6
0007 portc equ 7 ; f8

; realtime mode registers

0008 currentState equ 8 ; Millisecond timer
0009 msTimer equ currentState+1
000A sTimerLow equ msTimer+1 ; Lower byte second timer
000B sTimerHigh equ sTimerLow+1 ; Upper byte second timer
000C digit56 equ sTimerHigh+1
000D digit34 equ digit56+1

; Misc definitions

0060 FIVEMSEC equ 96 ; Assuming 4.096 MHz crystal
0000 w equ 0
0001 f equ 1
0002 z equ 2

; Status register bits

;********************************************************

; Port assignments
;
; porta - bit0: Common Plane 0 *
; bit1: Common Plane 0 *
; bit2: Common Plane 1 *
; bit3: Common Plane 1 *
;
; portb - bit0: 6B/DP *
; bit1: 6C/6D *
; bit2: 6G/6E *
; bit3: 6A/6F *
; bit4: 5B/DP *
; bit5: 5C/5D *
; bit6: 5G/5E *
Using PIC16C5X Microcontrollers as LCD Drivers

; bit7: SA/SF *
; *
; portc - bit0: 4B/DP *
; - bit1: 4C/4D *
; - bit2: 4G/4E *
; - bit3: 4A/4F *
; - bit4: 3B/3D *
; - bit5: 3C/3D *
; - bit6: 3G/3E *
; - bit7: 3A/3F *
;
;********************************************************
;********************************************************
; Macro definitions *
;********************************************************
UpdateState macro State, Table

swapf sTimerLow, w
andlw 0xf
call Table
movwf digit56
swapf digit56, f
movf sTimerLow, w
andlw 0xf
call Table
iorwf digit56, f
swapf sTimerHigh, w
andlw 0xf
call Table
movwf digit34
swapf digit34, f
movf sTimerHigh, w
andlw 0xf
call Table
iorwf digit34, f
movf digit34, w
movwf portc
movf digit56, w
movwf portb
endm

org 0

; Initialize ports A, B, and C and RTCC. In case of output data
; values, set the data latch first, then set the port direction.

Initialize

0000 0C01 movlw 00000001b ; Set data latch
0001 025 movwf porta
0002 0C08 movlw 00001000b ; Set I/O direction
0003 0005 tris porta
0004 0C00 movlw 00000000b ; Set levels to low
0005 026 movwf portb
0006 0C00 movlw 00000000b ; Set as outputs
0007 0006 tris portb
0008 0C00 movlw 00000000b ; Set levels to low
0009 027 movwf portc
00A0 0C00 movlw 00000000b ; Set as outputs
000B 0007 tris portc
000C 0C04 movlw 0x04 ; Set prescaler
000D 0002 option

© 1993 Microchip Technology Inc.
Using PIC16C5X Microcontrollers as LCD Drivers

```
0008 0060 movlw FIVEMSEC movwf rtcc  ; rtcc = 5ms
000F 0021 movwf rtcc
0010 00C0 movlw 4
0011 0028 movwf currentState
0012 00C0 movlw Ox0h
0013 0029 movwf msTimer  ; Initialize millisecond timer
0014 006A clrf sTimerLow  ; Clear second counter
0015 006B clrf sTimerHigh
0016 0800 retlw 0

; Check timer register for timing out (rtcc = 0). Remain in the
; loop until the timer times out.

; Wait for 5ms timer timeout

Timer_Check
0017 0201 movf rtcc, w
0018 0743 btfss status, z
go to Timer_Check
0019 0A17 goto Timer_Check
001A 0060 movlw FIVEMSEC
001B 0021 movwf rtcc
001C 02E9 decfsz msTimer, f
go to Update_Backplane
001D 0A21 goto Update_Backplane
001E 03EA incfsz sTimerLow  ; Update second counter
001F 0A21 goto Update_Backplane
0020 02AB incf sTimerHigh, f

; RA0 and RA1 are used to control voltage level for common plane 0.
; RA2 and RA3 are used to control voltage level for common plane 1.
; There are four possible states with different voltage levels as
; follows:
;
; State 0 - cp0 = +5v ra0=1, ra1=x
; cp1 = +2.5v ra2=1, ra3=x
; State 1 - cp0 = +2.5v ra0=1, ra1=0
; cp1 = +5v ra2=1, ra3=x
; State 2 - cp0 = 0v ra0=0, ra1=x
; cp1 = +2.5v ra2=0, ra3=x
; State 3 - cp0 = +2.5v ra0=1, ra1=0
; cp1 = 0v ra2=0, ra3=x

Update_Backplane
0021 0004 clrwdt  ; Reset watchdog timer
0022 00C8 decf currentState, w  ; Update w register
0023 00E3 andlw 0x03  ; Use only bit0/1
0024 0028 movwf currentState
0025 01E2 addwf pc, f  ; Update currentState
0026 0A2D goto State3
0027 0A81 goto State2
0028 0A55 goto State1
; goto State0
; State 0
State0
UpdateState State0, S0_Table
```
Using PIC16C5X Microcontrollers as LCD Drivers

```assembly
0029 038A swapf sTimerLow, w
002A 00EF andlw 0xf ; Isolate digit 5 (offset)
002B 0944 call S0_Table
002C 002C movwf digit56
002D 03AC swapf digit56, f

002E 020A movf sTimerLow, w
002F 020F andlw 0xf ; Isolate digit 6 (offset)
0030 0944 call S0_Table
0031 012C iorwf digit56, f

0032 038B swapf sTimerHigh, w
0033 00EF andlw 0xf ; Isolate digit 5 (offset)
0034 0944 call S0_Table
0035 002D movwf digit34
0036 03AD swapf digit34, f

0037 020B movf sTimerHigh, w
0038 00EF andlw 0xf ; Isolate digit 6 (offset)
0039 0944 call S0_Table
003A 012D iorwf digit34, f

003B 020D movf digit34, w ; Display digits 3 & 4
003C 0027 movwf portc
003D 020C movf digit56, w
003E 0026 movwf portb ; Display digits 5 & 6

003F 0C05 movlw 00000101b
0040 0025 movwf porta
0041 0C02 movlw 00000101b
0042 0005 tris porta

0043 0B00 retlw 0

S0_Table

0044 01E2 addwf pc, f ; Add offset to pc
0045 0B04 retlw 0100b 0
0046 0B0C retlw 1100b 1
0047 0B02 retlw 0010b 2
0048 0B00 retlw 0000b 3
0049 0B08 retlw 1000b 4
004A 0B01 retlw 0001b 5
004B 0B0F retlw 1111b 6
004C 0B04 retlw 0100b 7
004D 0B00 retlw 0000b 8
004E 0B00 retlw 0000b 9
004F 0B00 retlw 0000b a
0050 0B01 retlw 1001b b
0051 0B0B retlw 1011b c
0052 0B08 retlw 1000b d
0053 0B03 retlw 0011b e
0054 0B03 retlw 0011b f

; State 1

State1

UpdateState State1, S1_Table

0055 038A swapf sTimerLow, w
0056 00EF andlw 0xf ; Isolate digit 5 (offset)
0057 0970 call S1_Table
0058 002C movwf digit56
0059 03AC swapf digit56, f

005A 020A movf sTimerLow, w
005B 00EF andlw 0xf ; Isolate digit 6 (offset)
005C 0970 call S1_Table
```

© 1993 Microchip Technology Inc.
Using PIC16C5X Microcontrollers as LCD Drivers

005D 012C   iorwf digit56, f
005E 03BB   swapf sTimerHigh, w
005F 020F   andlw 0xf            ; Isolate digit 5 (offset)
0060 0970   call S1_Table
0061 002D   movwf digit34
0062 03AD   swapf digit34, f
0063 020B   movf sTimerHigh, w
0064 080F   andlw 0xf            ; Isolate digit 6 (offset)
0065 0970   call S1_Table
0066 012D   iorwf digit34, f
0067 020D   movf digit34, w      ; Display digits 3 & 4
0068 0027   movwf portc
0069 020C   movf digit56, w      ; Display digits 5 & 6
006A 0026
006B 0C05   movlw 00000101b
006C 0025   movwf porta
006D 0C08   movlw 00001000b
006E 0005   tris porta
006F 0800   retlw 0
0070 01E2   addwf pc, f
0071 0801   retlw 0001b          ; 0
0072 080F   retlw 1111b          ; 1
0073 0809   retlw 1001b          ; 2
0074 080D   retlw 1101b          ; 3
0075 0807   retlw 0111b          ; 4
0076 0805   retlw 0101b          ; 5
0077 080F   retlw 1111b          ; 6
0078 080F   retlw 1111b          ; 7
0079 0801   retlw 0001b          ; 8
007A 0805   retlw 0101b          ; 9
007B 0803   retlw 0011b          ; a
007C 0801   retlw 0001b          ; b
007D 0809   retlw 1001b          ; c
007E 0809   retlw 1001b          ; d
007F 0801   retlw 0001b          ; e
0080 0803   retlw 0011b          ; f

S1_Table

; State 2

State2
UpdateState State2, S2_Table

0081 038A   swapf sTimerLow, w   ; Isolate digit 5 (offset)
0082 080F   andlw 0xf            ; Isolate digit 6 (offset)
0083 099C   call S2_Table
0084 002C   movwf digit56
0085 03AC   swapf digit56, f
0086 020A   movf sTimerLow, w    ; Isolate digit 5 (offset)
0087 080F   andlw 0xf            ; Isolate digit 6 (offset)
0088 099C   call S2_Table
0089 012C   iorwf digit56, f
008A 03BB   swapf sTimerHigh, w  ; Isolate digit 5 (offset)
008B 080F   andlw 0xf            ; Isolate digit 6 (offset)
008C 099C   call S2_Table
008D 002D   movwf digit34
008E 03AD   swapf digit34, f
Using PIC16C5X Microcontrollers as LCD Drivers

008F 020B  movf    sTimerHigh, w
0090 020F  andlw  0xf ; Isolate digit 6 (offset)
0091 099C  call    S2_Table
0092 012D  iorwf   digit34, f

0093 020D  movf    digit34, w ; Display digits 3 & 4
0094 0027  movwf    portc
0095 020C  movf    digit56, w
0096 0026  movwf    portb ; Display digits 5 & 6

0097 0804  movlw    00000000b
0098 0025  movwf    porta
0099 0802  movlw    00000000b
009A 0005  tris    porta
009B 0800  retlw    0

S2_Table

009C 01E2  addwf   pc, f

009D 080B  retlw    0111b ; 0
009E 0803  retlw    0011b ; 1
009F 080D  retlw    1111b ; 2
00A0 080F  retlw    1111b ; 3
00A1 0807  retlw    1111b ; 4
00A2 080E  retlw    1111b ; 5
00A3 080E  retlw    1111b ; 6
00A4 080B  retlw    0111b ; 7
00A5 080F  retlw    1111b ; 8
00A6 080F  retlw    1111b ; 9
00A7 080F  retlw    1111b ; a
00A8 0806  retlw    0110b ; b
00A9 0804  retlw    0100b ; c
00AA 0807  retlw    0111b ; d
00AB 080C  retlw    1100b ; e
00AC 080C  retlw    1100b ; f

; State 3

State3
UpdateState   State3, S3_Table

00AD 038A  swapf   sTimerLow, w
00AE 080F  andlw   0xf ; Isolate digit 5 (offset)
00AF 099C  call    S3_Table
00B0 002C  movwf    digit56
00B1 03AC  swapf   digit56, f

00B2 020A  movf    sTimerLow, w ; Isolate digit 6 (offset)
00B3 080F  andlw   0xf
00B4 099C  call    S3_Table
00B5 012C  iorwf   digit56, f

00B6 038B  swapf   sTimerHigh, w
00B7 080F  andlw   0xf ; Isolate digit 5 (offset)
00B8 099C  call    S3_Table
00B9 002D  movwf    digit34
00BA 03AD  swapf   digit34, f

00BB 020B  movf    sTimerHigh, w ; Isolate digit 6 (offset)
00BC 080F  andlw   0xf
00BD 099C  call    S3_Table
00BE 012D  iorwf   digit34, f

00BF 020D  movf    digit34, w ; Display digits 3 & 4
00C0 0027  movwf    portc
Using PIC16C5X Microcontrollers as LCD Drivers

Using PIC16C5X Microcontrollers as LCD Drivers

OOC1 020C movf digit56, w
OOC2 0026 movwf portb  ; Display digits 5 & 6

OOC3 00C1 movlw 00000001b
OOC4 0025 movwf porta
OOC5 0C08 movlw 00001000b
OOC6 0005 tris porta
OOC7 0800 retlw 0

S3_Table
  OOC8 01E2 addwf pc, f

  OOC9 080E retlw 1110b  ; 0
  Ooca 0800 retlw 0000b  ; 1
  Oocb 0806 retlw 0110b  ; 2
  Oocc 0802 retlw 0010b  ; 3
  Oocd 0808 retlw 1000b  ; 4
  Ooce 080A retlw 1010b  ; 5
  Oocf 080E retlw 1110b  ; 6
  00D0 0800 retlw 0000b  ; 7
  00D1 080E retlw 1110b  ; 8
  00D2 080A retlw 1010b  ; 9
  00D3 080C retlw 1100b  ; a
  00D4 080E retlw 1110b  ; b
  00D5 0806 retlw 0110b  ; c
  00D6 0806 retlw 0110b  ; d
  00D7 080E retlw 1110b  ; e
  00D8 080C retlw 1100b  ; f

; Main code

Start  call Initialize
Repeat  call Timer_Check
         goto Repeat
         org pic55

System_Reset
  01FF 0AD9 goto Start

END

Errors : 0
Warnings : 0
SECTION 3
PIC16CXX APPLICATION NOTES

Using the Analog to Digital Converter - AN546 ................................................................. 3- 1
Implementing Wake Up on Keystroke - AN552 ................................................................. 3- 21
PortB as External Interrupt - AN566 .................................................................................... 3- 25
Table Read Using PIC16CXX - AN556 ............................................................................. 3- 29
Software Implementation of I²C Bus Master - AN554 ....................................................... 3- 33
Software Implementation of Asynchronous Serial I/O - AN555 ........................................ 3- 121
Four Channel Digital Volt Meter with Display and Keyboard - AN557 ............................. 3- 157
INTRODUCTION

This application note is intended for PIC16C71 users with various degrees of familiarity with analog system design. The various sections discuss the following topics:

- Commonly used A/D terminology
- How to configure and use the PIC16C71 A/D
- Various ways to generate external reference voltage (VREF)
- Configuring RA0-RA3 pins

COMMONLY USED A/D TERMINOLOGY

The Ideal Transfer Function

In an A/D converter, an analog voltage is mapped into an N-bit digital value. This mapping function is defined as the transfer function. An ideal transfer is one in which there are no errors or non-linearity. It describes the “ideal” or intended behavior of the A/D. Figure 1 shows the ideal transfer function for the PIC16C71 A/D. Note that the digital output value is 00h for analog input voltage range of 0 to 1LSB. In some converters, the first transition point is at 0.5LSB and not at 1LSB as shown in Figure 2. Either way, knowing the transfer function the user can appropriately interpret the data.

Transition Point

It is the analog input voltage at which the digital output switches from one code to the next. The transition point is typically not a single threshold, rather a small region of uncertainty (see Figure 3). The transition point is therefore defined as the statistical average of many conversions. Stated differently, it is the voltage input at which the uncertainty of the conversion is 50%.

Code Width

It is the distance (voltage differential) between two transition points. Ideally the Code Width should be 1LSB. See Figure 1.

FIGURE 1 - PIC16C71 IDEAL TRANSFER FUNCTION

FIGURE 2 - ALTERNATE TRANSFER FUNCTION
Using the Analog to Digital Converter

**Center of Code Width**

It is the midpoint between two transition points. See Figure 3.

**FIGURE 3 - TRANSITION POINTS**

![Transition Points Diagram](image)

**Differential Non-Linearity (DNL)**

It is the deviation in code-width from 1 LSB (Figure 7). The difference is calculated for each and every transition. The largest difference is reported as DNL.

It is important to note that the DNL is measured after the transfer function is normalized to match offset error and gain error.

Note that the DNL cannot be any less than -1 LSB. In the other direction, DNL can be >1 LSB.

**FIGURE 7 - DIFFERENTIAL NON-LINEARITY**

![Differential Non-Linearity Diagram](image)

**Absolute Error**

The maximum deviation between any transition point from the corresponding ideal transfer function is defined as the absolute error. This is how it is measured and reported in the PIC16C71 (Figure 8). The notable difference between absolute error and INL is that the measured data is not normalized for full scale and offset errors.

It is probably the first parameter the user will look at to evaluate an A/D. Sometimes absolute error is reported as the sum of offset, full-scale and integral non-linearity errors.

**Total Unadjusted Error**

It is the same as absolute error. Again, sometimes it is reported as the sum of offset, full-scale and integral non-linearity errors.

**No Missing Code**

No missing code implies that as the analog input voltage is gradually increased from zero to full scale (or vice versa), all digital codes are produced. Stated otherwise, changing analog input voltage from one quantum of the analog range to the next adjacent range will not produce a change in the digital output by more than one code count.

**Monotonic**

Monotonicity guarantees that an increase (or decrease) in the analog input value will result in an equal or greater digital code (or less). Monotonicity does not guarantee that there are no missing codes. However, it is an important criterion for feedback control systems. Non monotonicity may cause oscillations in such a system.

The first derivative of a monotonic function always has the same sign.

**FIGURE 8 - ABSOLUTE ERROR**

![Absolute Error Diagram](image)
Using the Analog to Digital Converter

**Ratiometric Conversion**

It is the A/D conversion process where the binary result is a ratio of the supply voltage or reference voltage, the latter being equal to full-scale value by default. The PIC16C71 is a ratiometric A/D converter where the result depends on VDD or VREF.

In some A/D's, an absolute reference is provided resulting in "absolute conversion".

**Sample and Hold**

In sample and hold type A/D converters, the analog input has a switch (typically a FET switch in CMOS) which is opened for a short duration to capture the analog input voltage onto an on-chip capacitor. Conversion is typically started after the sampling switch is closed.

**Track and Hold**

It is basically the same as sample and hold, except the sampling switch is typically left on. Therefore the voltage on the on-chip holding capacitor "tracks" the analog input voltage. To begin a conversion, the sampling switch is shut off.

The PIC16C71 A/D falls in this category.

**Sampling Time**

It is the time required to charge the on-chip holding capacitor to the same value as on the analog input pin. The sampling time depends on the magnitude of the holding capacitor and the source impedance of the analog voltage input.

**Offset Error (or Zero Error)**

It is the difference between the first actual (measured) transition point and the first ideal transition point as shown in Figure 4. It can be corrected by the user by subtracting the offset error from each conversion result.

**Full Scale Error (or Gain Error)**

It is the difference between the ideal Full Scale and the actual (measured) full scale range (see Figure 5). It is also called gain error, because the error changes the slope of the ideal transfer function creating a gain factor. It can be corrected by the user by multiplying each conversion result by the inverse of the gain.

**Integral Non-Linearity (INL), or Relative Error**

It is the deviation of a transition point from its corresponding point on the ideal transfer curve (Figure 6). The maximum difference is reported as the INL of the converter.

It is important to note that Full Scale Error and the Offset Error are normalized to match end transition points before measuring the INL.
Using the Analog to Digital Converter

HOW TO USE THE PIC16C71 A/D

The A/D in the PIC16C71 is easy to set up and use. There are a few considerations:

1. Select either VDD or VREF as reference voltage. More on using VREF input later.
2. Select A/D conversion clock (tad): 2 tosc, 8 tosc, 32 tosc or trc (internal RC clock). For the first three options, make sure that \( \text{tad} \geq 2.0 \mu s \). If deterministic conversion time is required, select tosc time base. If conversion during SLEEP is required, select trc.
3. Channel Selection: If only one A/D channel is required, program the ADCON1 register to 03h. This configures the A/D pins as digital I/O. If multiple channels are required, prior to each conversion the new channel must be selected.
4. Sampling and Conversion: After a new channel is selected, a minimum amount of sampling time must be allowed before GO bit in ADCONO is set to begin conversion. Once conversion begins, it is OK to select the next channel, \textit{but sampling does not begin until current conversion is complete}. Therefore, it is always necessary to provide minimum required sampling time
   i) after a conversion
   ii) after a new channel is selected
   iii) after A/D is turned on (ADON = 1).
5. Reading Result: Completion of conversion can be determined by either polling GO/DONE bit to cleared, polling the ADIF bit to be set, or waiting for an ADIF interrupt.

Example 1: How to do a simple ADC conversion.

```plaintext
; InitializeAD, initializes and sets up the A/D hardware.
; Always ch2, internal RC OSC.
InitializeAD
    bsf STATUS, 5 ; select pgl
    movlw B'00000000' ; select RA0-RA3...
    movwf ADCON1 ; as analog inputs
    bcf STATUS, 5 ; select pg0
    movlw B'11010001' ; select: RC osc, ch2...
    movwf ADCON0 ; turn on A/D
Convert
    call sample-delay ; provide necessary sampling time
;
    bsf ADCON0, 2 ; start new A/D conversion
loop
    btfsc ADCON0, 2 ; A/D over?
    goto loop ; no then loop
;
    movf ADRES, w ; yes then get A/D value

A detailed code listing is in Appendix A.
```

ADDITIONAL TIPS:

1. The GO bit and the ADON bit may not be set at once. After the A/D is turned on by setting ADON, at least 5\( \mu s \) time must be allowed before conversion begins, longer if sampling time requirement is not met within 5\( \mu s \).
2. Aborting a conversion: A conversion can be aborted by clearing GO bit. The A/D converter will stop conversion and revert back to sampling state.
3. Using ADRES register as a normal register: The A/D only writes to ADRES at the end of a conversion. Therefore, it is possible to use ADRES as a normal file register between conversions and when A/D is off.

The following are a few examples of using the A/D.
Example 2: How to do sequential channel conversions.

```assembly
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3 in a round robin fashion, internal RC OSC.
; Load results in 4 consecutive addresses starting at ADTABLE (10h)

InitializeAD

bsf STATUS, 5  ; select pg1
movlw B'00000000'  ; select RA0-RA3...
movwf ADCON1  ; as analog inputs
bcf STATUS, 5  ; select pg0
movlw B'11000000'  ; select: RC osc, ch0...
movwf ADCON0  ; turn on A/D
movlw ADTABLE  ; point fsr to top of...
movwf FSR  ; table

new_ad call sample_delay  ; provide necessary sampling time
bsf ADCON0, 2  ; start new A/D conversion

loop

btfsc ADCON0, 2  ; A/D over?
goto loop  ; no then loop

movf adres, w  ; yes then get A/D value
movwf 0  ; load indirectly
movlw 4  ; select next channel
addwd ADCON0  ;
bcf ADCON0, 5  ; reset carry over bit.

; increment pointer to correct table offset.
clr temp  ; clear temp register
btfsc ADCON0, 3  ; test lsb of channel select
bsf temp, 0  ; set if ch1 selected
btfsc ADCON0, 4  ; test msb of channel select
bsf temp, 1  ;
movlw ADTABLE  ; get table address
addwf temp, w  ; add with temp
movwf FSR  ; move into indirect

goto new_ad
```

A detailed code listing is in Appendix B.
Using the Analog to Digital Converter

Example 3: How to write the interrupt handler for the ADC.

```
org 0x00
goto start
org 0x04
goto service_ad ; interrupt vector

; org 0x10
start
movlw B'00000000' ; init I/O ports
movwf PORT_B
tris PORT_B

; call InitializeAD
update
bcf flag,adover ; reset software A/D flag
call SetupDelay ; setup delay >= 10uS.
bcf ADCON0,adif ; reset A/D int flag (ADIF
bsf ADCON0,adgo ; start new A/D conversion
bsf INTCON,gie ; enable global interrupt

loop
btfsc flag,adover ; A/D over?
goto update ; yes start new conv.
goto loop ; no then keep checking

; InitializeAD, initializes and sets up the A/D hardware.
; select ch0 to ch3, RC OSC., a/d interrupt.
InitializeAD
bsf STATUS, 5 ; select pg1
movlw B'00000000' ; select RA0-RA3...
movwf ADCON1 ; as analog inputs
bcf STATUS, 5 ; select pg0
clrf INTCON ; clr all interrupts
bsf INTCON, 6 ; enable A/D int.
movlw B'11010001' ; select: RC osc, ch2...
movwf ADCON0 ; turn on A/D
return ;

service_ad
btfss ADCON0, 1 ; A/D interrupt?
retfie ; no then ignore
movf ADRES, W ; get A/D value
return ; do not enable int
```

A detailed code listing is in Appendix C.
Example 4: How to do conversions during sleep mode.

```
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3, internal RC OSC.
; While doing the conversion put unit to sleep. This will
; minimize digital noise interference.
; Note that ad's RC osc. has to be selected in this instance.
;
InitializeAD
    bsf      STATUS, 5 ; select p1
    movlw    B'00000000' ; select RA0-RA3...
    movwf    ADCON1    ; as analog inputs
    bcf      STATUS, 5 ; select pg0
    movlw    B'1100001' ; select: RC osc, ch0...
    movwf    ADCON0    ; turn on A/D & ADIE
    movlw    ADTABLE   ; point fsr to top of...
    movwf    FSR       ; table

; new_ad
    bsf      ADCON0, 2 ; start new A/D conversion
    sleep    ; goto sleep
; when A/D is over program will continue from here
;
    movf     ADRES, w ; get A/D value
```

A detailed code listing is in Appendix D.
Using the Analog to Digital Converter

USING EXTERNAL REFERENCE VOLTAGE

When using external reference voltage, keep in mind that any analog input voltage must not exceed VREF.

An inexpensive way to generate VREF is by employing zener diode (Figure 9). Most common zener diodes offer 5% accuracy. Reverse bias current may be as low as 10 µA. However, larger currents (1mA - 20mA) are recommended for stability, as well as lower impedance of the VREF source.

FIGURE 9 - LOW COST VOLTAGE REFERENCE

An inexpensive way to generate VREF is by employing zener diode (Figure 9). Most common zener diodes offer 5% accuracy. Reverse bias current may be as low as 10 µA. However, larger currents (1mA - 20mA) are recommended for stability, as well as lower impedance of the VREF source.

C = 0.01 to 0.1µF

POWER MANAGEMENT IN USING VREF

In power sensitive applications, user may turn on VREF generator using another I/O pin as shown in Figure 10. Drive a “1” on RB1 pin in this example when using the A/D. Drive a “0” on RB1 pin when not using the A/D converter.

Note that this way RB1 is not floating. Even if VREF decays to some intermediate voltage, it will not cause the input buffer on RB1 to draw current.

Alternately, use RA0, RA1 or RA2 pin to supply the current instead of RB1. Configure the RA pin as analog (this will turn off its input buffer). Then use it as a digital output (Figure 11).

FIGURE 10 - POWER-SENSITIVE APPLICATIONS #1

ZENERS AND REFERENCE GENERATORS

Finally, various reference voltage generator chips (typically using on-chip band-gap reference) are available. These are more accurate.

TABLE 1 - ZENERS AND REFERENCE GENERATORS

<table>
<thead>
<tr>
<th>Zeners</th>
<th>Vz</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1N746</td>
<td>3.3V</td>
<td>±5%</td>
</tr>
<tr>
<td>1N747</td>
<td>3.6V</td>
<td>±5%</td>
</tr>
<tr>
<td>1N748</td>
<td>3.9V</td>
<td>±5%</td>
</tr>
<tr>
<td>1N749</td>
<td>4.3V</td>
<td>±5%</td>
</tr>
<tr>
<td>1N750</td>
<td>4.7V</td>
<td>±5%</td>
</tr>
<tr>
<td>1N751</td>
<td>5.1V</td>
<td>±5%</td>
</tr>
<tr>
<td>1N752</td>
<td>5.6V</td>
<td>±5%</td>
</tr>
</tbody>
</table>

Voltage References VREF Tolerance

<table>
<thead>
<tr>
<th>Voltage References</th>
<th>VREF</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD580 (Maxim)</td>
<td>2.5V</td>
<td>±3% to ±0.4%</td>
</tr>
<tr>
<td>LM385</td>
<td>2.5V</td>
<td>±1.5%</td>
</tr>
<tr>
<td>LM1004</td>
<td>2.5V</td>
<td>±1.2%</td>
</tr>
<tr>
<td>LT1009 (LIN. Tech.)</td>
<td>2.5V</td>
<td>±0.2%</td>
</tr>
<tr>
<td>LT1019 (LIN. Tech.)</td>
<td>5.0V</td>
<td>±0.2%</td>
</tr>
<tr>
<td>LT1021 (LIN. Tech.)</td>
<td>5.0V</td>
<td>±0.05% to ±1%</td>
</tr>
<tr>
<td>LT1029 (LIN. Tech.)</td>
<td>5.0V</td>
<td>±0.2% to ±1%</td>
</tr>
</tbody>
</table>
Using the Analog to Digital Converter

Vref Impedance and Current Supply Requirements

Ideally, Vref should have as low a source impedance as possible. Referring to Figure 9, Vref source impedance = R. However, smaller R increases current consumption. Since Vref is used to charge capacitor arrays inside the A/D converter and the holding capacitor Chold ≈ 51 pF, the following guideline should be met:

\[ \text{tad} = 6 (1K + R) \times 51.2 \text{ pF} + 1.677 \mu\text{s} \]

\[ \text{tad} = \text{conversion clock}. \text{ For tad} = 2\mu\text{s and for Chold} = 50 \text{ pF, RVREF} = 50\Omega. \]

For Vref impedance higher than this, the conversion clock (tad) should be increased appropriately.

FIGURE 11 - POWER-SENSITIVE APPLICATIONS #2

Table 2 gives examples of the maximum rate of conversion per bit, relating to the voltage reference impedance.

**TABLE 2 - MAXIMUM RATE OF CONVERSION / BIT**

<table>
<thead>
<tr>
<th>RVref</th>
<th>Tad (Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>2.29 \mu s</td>
</tr>
<tr>
<td>5K</td>
<td>3.52 \mu s</td>
</tr>
<tr>
<td>10K</td>
<td>5.056 \mu s</td>
</tr>
<tr>
<td>50K</td>
<td>16.66 \mu s</td>
</tr>
<tr>
<td>100K</td>
<td>32.70 \mu s</td>
</tr>
</tbody>
</table>

Assumes no external capacitors.

To achieve a low source impedance when using a Zener diode, a voltage follower circuit is recommended. This is shown in Figure 11A.

FIGURE 11A - VOLTAGE FOLLOWER CIRCUIT

Any general purpose op-Amp (LM358, LM324, ...)

Configuring Port A Inputs as Analog or Digital

Two bits in ADCON1 register PCFG1 and PCFG0 control how pins RA0–RA3 are configured. When any of these pins are selected as analog:

- The digital input buffer is turned off to save current (see Figure 12). Reading the port will read this pin as '0'.
- TRIS bit still controls the output buffer on this pin. So, normally the TRIS bit will be set (input).
- However, if the TRIS bit is cleared, then the pin will output whatever is in the data latch.

When any of these pins are cleared, then the pin can be used as analog input.

The user has, therefore, great flexibility in configuring these pins.
Using the Analog to Digital Converter

**CURRENT CONSUMPTION THROUGH INPUT BUFFER**

A CMOS input buffer will draw current when the input voltage is around its threshold. (See Figure 13.)

In power-sensitive applications, the RA pins when used as analog inputs should be configured as "analog" to avoid unintended power drain.

Other considerations and tips:

1. If possible, avoid any digital output next to analog inputs.
2. Avoid digital inputs that switch frequently (e.g., clocks) next to analog inputs.
3. If VREF is used, then no analog pin being sampled should exceed VREF.

**SUMMARY**

The PIC16C71 A/D converter is simple to use. It is versatile and low power.

**AUTHORS:** Sumit Mitra, Stan D'Souza, Russ Cooper,
Logic Products Division

---

**FIGURE 13 - A SIMPLE CMOS INPUT BUFFER:**

\[ V_{TH} = \text{Threshold of the inverter} \]
\[ V_{TN} = \text{Device threshold of NMOS pull-down} \]
\[ V_{TP} = \text{Device threshold of PMOS pull-up} \]
\[ I = \text{On-current (or through current) of the inverter} \]
\[ I_{MAX} = \text{Maximum on-current occurs when } V_{IN} = V_{TH}. \text{ Value of } I_{MAX} \text{ depends on the sizes of the devices. The larger the devices, the faster the input buffer, and the larger the value of } I_{MAX}. \text{ Typically, } I_{MAX} = 0.2\text{mA} - 1\text{mA}. \]
Using the Analog to Digital Converter

APPENDIX A - SINGLE CHANNEL A/D (SAD)

This program is a simple implementation of the PIC16C71's
A/D. 1 Channel is selected (CHO).
The A/D is configured as follows:
Vref = +5V internal.
A/D Osc. = internal RC
A/D Channel = CHO
Hardware: PICDEMO board.

LIST P=16C71,F=INHX8M
include "picreg.equ"

org 0x00
goto service_int ;interrupt vector

start
org 0x10

0010 TEMP equ 10h
0011 adif equ 1
0012 adgo equ 2

movlw B'00000000'
PORT B

tris PORT_B

0014 0089
0015 0086
0016 0205
0017 1088
0018 1508
0019 1888
001A 2814
001B 2819

001C 0008

return ;do not enable global.

Stan D'Souza 7/6/93

LIST P=16C71,F=INHX8M
include "picreg.equ"

org 0x00
goto service_int ;interrupt vector

start
org 0x10

0010 TEMP equ 10h
0011 adif equ 1
0012 adgo equ 2

movlw B'00000000'
PORT B

tris PORT_B

0014 0089
0015 0086
0016 0205
0017 1088
0018 1508
0019 1888
001A 2814
001B 2819

001C 0008

return ;do not enable global.

Stan D'Souza 7/6/93

LIST P=16C71,F=INHX8M
include "picreg.equ"

org 0x00
goto service_int ;interrupt vector

start
org 0x10

0010 TEMP equ 10h
0011 adif equ 1
0012 adgo equ 2

movlw B'00000000'
PORT B

tris PORT_B

0014 0089
0015 0086
0016 0205
0017 1088
0018 1508
0019 1888
001A 2814
001B 2819

001C 0008

return ;do not enable global.

Stan D'Souza 7/6/93

LIST P=16C71,F=INHX8M
include "picreg.equ"

org 0x00
goto service_int ;interrupt vector

start
org 0x10

0010 TEMP equ 10h
0011 adif equ 1
0012 adgo equ 2

movlw B'00000000'
PORT B

tris PORT_B

0014 0089
0015 0086
0016 0205
0017 1088
0018 1508
0019 1888
001A 2814
001B 2819

001C 0008

return ;do not enable global.
;InitializeAD, initializes and sets up the A/D hardware.
;Select ch0 to ch3 as analog inputs, fosc/2 and read ch0.
InitializeAD
    bsf STATUS,5 ;select pg1
    movlw B'00000000' ;select ch0-ch3...
    movwf ADCON1 ;as analog inputs
    bcf STATUS,5 ;select pg0
    movlw B'11000001' ;select:RC,ch0:
    movwf ADCON0 ;select:RC,ch0:
    clrf ADRES ;clr result reg.
    return

;This routine is a software delay of 10uS for the a/d setup.
;At 4MHz clock, the loop takes 3uS, so initialize TEMP with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.
SetupDelay
    movlw .3
    movwf TEMP
    SD
    decfsz TEMP
    goto SD
    return
END

Errors : 0
Warnings : 0
Using the Analog to Digital Converter

APPENDIX B

MPASM B0.44   PAGE 1

;TITLE   "A/D in Sleep Mode"
;This program is a simple implementation of the PIC16C71's
;A/D feature. This program demonstrates;
;how to do a A/D in sleep mode on the PIC16C71.
;The A/D is configured as follows:
;   Vref = +5V internal.
;   A/D Osc. = internal RC
;   A/D Interrupt = OFF
;   A/D Channels = ch 0
;The ch0 A/D result is displayed as a 8-bit binary value
;on 8 LEDs connected to port b.
;Hardware: PICDEMO board.
;       Stan D'Souza 7/6/93
;
LIST P=16C71,F=INHX8M
;
   include "picreg.equ"
;
0010   TEMP   equ   10h
0001   adif   equ   1
0002   adgo   equ   2
;
   org   0x00
;
0000   2810
;
   goto   start
;
   org   0x04
0004   281B
;
   goto   service_int ;interrupt vector
;
   org   0x10
0010   3000
0011   0086
0012   0066
;
   movlw   B'00000000' ;make port b all
   movwf   PORT_B ;outputs.
0013   201C
;
   call   InitializeAD
0014   0809
0015   0086
0016   2025
0017   1088
0018   1508
;
   movf   ADRES,W ;save in table
   movwf   PORT_B ;
call   SetupDelay ;
bcf   ADCON0.adif ;clr A/D flag
bsf   ADCON0.adgo ;start new A/D conversion
;
0019   0063

sleep
Using the Analog to Digital Converter

MPASM B0.44

PAGE 2

001A 2814 goto update ;wake up and update
001B 0008 service_int ;do not enable int

;InitializeAD, initializes and sets up the A/D hardware.
InitializeAD

001C 1683 bsf STATUS,5 ;select pg1
001D 3000 movlw B’00000000’ ;select ch0-ch3...
001E 0108 movwf ADCON1 ;as analog inputs
001F 1283 bcf STATUS,5 ;select pg0
0020 30C1 movlw B’11000001’ ;select:internal RC, ch0.
0021 0088 movwf ADCON0 ;turn on A/D
0022 018B clrf INTCON ;clear all interrupts
0023 170B bsf INTCON,ADIE ;enable A/D
0024 0008 return

;This routine is a software delay of 10us for the A/D setup.
;At 4Mhz clock, the loop takes 3us, so initialize TEMP with
;a value of 3 to give 9us, plus the move etc should result in
;a total time of > 10us.
SetupDelay

0025 3003 movlw .3
0026 0090 movwf TEMP
0027 0B90 decfsz TEMP
0028 2827 goto SD
0029 0008 return

; END

Errors : 0
Warnings : 0
Using the Analog to Digital Converter

APPENDIX C

MPASM B0.44

;TITLE "Single channel A/D with interrupts"
;This program is a simple implementation of the PIC16C71's
;A/D. 1 Channel is selected (CHO). A/D interrupt is turned on,
; hence on completion of A/D conversion, an interrupt is generated.
;The A/D is configured as follows:
; Vref = +5V internal.
; A/D Osc. = Internal RC Osc.
; A/D Interrupt = On
; A/D Channel = CHO
;
;The A/D result is displayed as a 8-bit value on 8 LEDs connected
;to portb.
;Hardware: PICDEMO board.
;
; LIST P=16C71,F=INHX8M
;
; include "picreg.equ"
;
0010 flag equ 10
0011 TEMP equ 11
0000 adover equ 0
0001 adif equ 1
0002 adgo equ 2
0006 adie equ 6
0007 gie equ 7
0005 rp0 equ 5
;
org 0x00
goto start
;
org 0x04
goto service_ad ;interrupt vector
;
org 0x10
start
movlw B'00000000' ;init I/O ports
movwf PORT_B
tris PORT_B
;
0013 2022 call InitializeAD
update

0014 1010 bcf flag,adover ;reset software A/D flag
0015 202B call SetupDelay ;setup delay >= 10uS.
0016 108B bcf ADCON0,adif ;reset A/D int flag (ADIF)
0017 1508 bsf ADCON0,adgo ;start new A/D conversion
0018 170B bsf INTCON,gie ;enable global interrupt
loop

0019 1810 btfsc flag,adover ;A/D over?
001A 2814 goto update ;yes start new conv.
001B 2819 goto loop ;no then keep checking
Using the Analog to Digital Converter

; InitializeAD, initializes and sets up the A/D hardware.
; select ch0, RC OSC., A/D interrupt.
InitializeAD
  bsf STATUS, rp0 ; select pg1
  movlw B'00000000' ; select ch0-ch3...
  movwf ADCON1 ; as analog inputs
  bcf STATUS, rp0 ; select pg0
  clrf INTCON ; clr all interrupts
  bsf INTCON, adie ; enable A/D int.
  movlw B'11000001' ; select: RC osc, ch0...
  movwf ADCONO ; turn on A/D
  return

; This routine is a software delay of 10uS for the A/D setup.
; At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
; a value of 3 to give 9uS, plus the move etc should result in
; a total time of > 10uS.
SetupDelay
  movlw .3
  movwf TEMP
  decfsz TEMP
  goto SD
  return

END

Errors : 0
Warnings : 0
Using the Analog to Digital Converter

APPENDIX D

MPASM B0.44  PAGE 1

;TITLE "A/D using Multiple Channels"
;This program is a simple implementation of the PIC16C71's
;A/D feature. This program demonstrates
;how to select multiple channels on the PIC16C71.
;The A/D is configured as follows:
;Vref = +5V internal.
;A/D Osc. = internal RC osc.
;A/D Interrupt = Off
;A/D Channels = all in a "Round Robin" format.
;A/D results are stored in ram locations as follows:
;ch0 -> ADTABLE + 0
;ch1 -> ADTABLE + 1
;ch2 -> ADTABLE + 2
;ch3 -> ADTABLE + 3
;
;The ch0 A/D result is displayed as a 8-bit value on 8 LEDs
;connected to port b.
;Hardware: PICDEM board.
;Stan D'Souza 7/6/93.
;
;LIST P-16C71,F-INHXSM
;include "picreg.equ"

TEMP equ l0h
adif equ 1
adgo equ 2

ch2 equ 6
ch3 equ 7
flag equ 0C
ADTABLE equ 20

org Ox00
goto start
org Ox04
goto service_int ;interrupt vector

org Ox10
start
movlw B'00000000' ;make port b
movwf PORT_B ;as all outputs
tris PORT_B ;/

call InitializeAD
Using the Analog to Digital Converter

MPASM B0.44

update

0014 0809 movf ADRES, W
0015 0080 movwf 0 ; save in table
0016 0020 movlw ADTABLE ; chk if ch0
0017 0204 subwf FSR, W ; /
0018 1003 btfss STATUS, Z ; yes then skip
0019 281C goto NextAd ; else do next channel
001A 0809 movf ADRES, W ; get A/D value
001B 0086 movwf PORT B ; output to port b

NextAd

001C 202E call NextChannel ; select next channel
001D 203A call SetupDelay ; set up > = 10uS
001E 1088 bcf ADCONO, adif ; clear flag
001F 0088 movwf ADCONO ; turn on A/D

loop

0020 1888 btfsc ADCONO, adif ; A/D done?
0021 2814 goto update ; yes then update
0022 2820 goto loop ; wait till done

; service_int

0023 0008 return ; do not enable int

; InitializeAD, initializes and sets up the A/D hardware.

InitializeAD

0024 1683 bsf STATUS, 5 ; select pg1
0025 3000 movlw B’00000000’ ; select ch0-ch3...
0026 0108 movwf ADCON1 ; as analog inputs
0027 1283 bcf STATUS, 5 ; select pg0
0028 30C1 movlw B’11000001’ ; select: fosc/2, ch0.
0029 0080 movwf ADCON0 ; turn on A/D
002A 3020 movlw ADTABLE ; get top of table address
002B 0084 movwf FSR ; load into indirect reg
002C 0189 clrf ADRES ; clr result reg.
002D 0008 return

Using the Analog to Digital Converter

Errors : 0
Warnings : 0
INTRODUCTION

Microchip’s PIC16CXX family of microcontrollers are ideally suited to directly interface to a keypad. The high 4 bits of PortB (RB4 - RB7) have internal pull-ups and can trigger a “change on port state” interrupt. This interrupt, if enabled, will wake the microcontroller from sleep. In most battery powered applications a microcontroller is exercised when a key is pressed, e.g. in a remote keyless entry system. The life of the battery can be extended by using PIC16CXX microcontrollers. This can be done by putting the PIC16CXX microcontroller into sleep mode for most of the time and wake-up only when a key is pressed.

IMPLEMENTATION

Figure 1 depicts an application where four keys are connected to RB4 - RB7. Internal pull-ups are used to maintain a high level on these inputs. In this example, LEDs are connected to RB0 - RB3. When SW1 is pressed, LED1 is turned on and when SW2 is pressed, LED2 is turned on and so on. The PIC16CXX is normally in sleep mode with the "change on port state" interrupt enabled. When SW1 is pressed, RB4 goes low and triggers an interrupt. Since the PIC16CXX is in sleep, it first wakes up and starts executing code at the interrupt vector. Note that if the global interrupt is enabled, the program execution after an interrupt is at the interrupt vector, if the global interrupt is not enabled, the program starts executing right after the sleep instruction.

After waking up, a 20 - 40 msec. de-bounce delay is executed which checks the port for a key hit and depending on which key is hit, its associated LED is turned on. The LEDs are used purely for demonstration purposes. In a remote keyless entry application, the remote code would be transmitted when the appropriate key is hit.

Figure 2 depicts a 4x4 keypad interface to the PIC16CXX. When using the PIC16CXX in a keypad application, the internal pull-ups on RB4 - RB7 can be enabled eliminating the need for external pull-up resistors. The series 100Ω resistors are used for ESD protection, and are recommended in keypad interface applications.

AUTHOR: Stan D’Souza, Logic Products Division

SUMMARY

The PIC16CXX is ideally suited to interface directly to a Keypad application. Built in pull up resistors and very low sleep current make it a very good candidate for battery powered remote operations and applications.

Performance:
Code Size: 64 words
RAM Locations Used: 0 bytes

FIGURE 1 - 4 KEY INTERFACE TO PIC16CXX

FIGURE 2 - 4X4 KEYPAD INTERFACE TO PIC16CXX
Implementing Wake-up on Key Stroke

This program demonstrates the wake-up on Keystroke feature of the PIC16C71. Port B pins RB4 - RB7 can be configured as inputs with internal pull up resistors, also the interrupt associated with the change on input on RB4 - RB7 can be set up to wake the chip from sleep. If the global interrupt is enabled just before sleep, the program will vector to the interrupt vector (0004). If not the chip will continue execution just after the next instruction following sleep.

In this example code, the port B is initialized to input 4 keys at RB4 - RB7. RB0 - RB3 are configured to drive LEDs corresponding to which key is hit (LED on RB0 when RB4 is hit and so on). Sleep is executed. When any keys is hit, the processor wakes up and jumps to the interrupt vector. The corresponding LED is turned on and after the key is released, the whole process is repeated.

```
LIST P=16C71, F=INHX8M

org 0
start
loop
sleep
goto loop

ServiceInterrupt

InitPortB

int

change on RB int?

yes then service

clear RTCC int mask

clear flag

return
```
Implementing Wake-up on Key Stroke

Line   PC   Opcode
0046   ;
0047   ;This routine checks which keys is hit and lights up the
0048   ;corresponding LED associated with it. eg. RB6's LED when
0049   ;RB4's key is pressed. Finally it waits till all keys have
0050   ;been released before returning form the service routine.
0051   ServiceWakup
0052   000D 118B bcf INTCON, RBIE ;clear mask
0053   000E 0906 comf PORT_B, w ;read PORT_B
0054   000F 100B bcf INTCON, RBIF ;clear flag
0055   0010 2035 call delay16 ;do de-bounce for 16mSecs
0056   0011 0906 comf PORT_B, w ;read port B again
0057   0012 39FO andlw B'11110000' ;mask outputs
0058   0013 0090 movwf temp ;save in temp
0059   0014 0810 swapf temp, w ;switch low and high
0060   0015 0086 movwf PORT_B ;send as outputs.
0061   0016 2018 call KeyRelease ;check for key release
0062   0017 0009 retfie
0063   ;
0064   ;This sub-routine, waits till all key have been released
0065   ;In order to save power, the chip is in sleep mode till
0066   ;all keys are released.
0067   KeyRelease
0068   0018 2035 call delay16 ;do debounce
0069   0019 0906 comf PORT_B, w ;read PORT_B
0070   001A 100B bcf INTCON, RBIF ;clear flag
0071   001B 158B bcf INTCON, RBIE ;enable mask
0072   001C 39FO andlw B'11110000' ;clear outputs
0073   001D 1903 btfsc STATUS, z ;key still pressed?
0074   001E 0008 return ;no then return
0075   001F 0063 sleep ;else save power
0076   0020 118B bcf INTCON, RBIE ;on wake up clear mask
0077   0021 0906 comf PORT_B, w
0078   0022 100B bcf INTCON, RBIF ;clear flag
0079   0023 2818 goto KeyRelease ;try again
0080   ;
0081   ;This sub-routine, initializes PortB.
0082   InitPortB
0083   0024 1683 bsf STATUS, R0 ;select bank 1
0084   0025 3003 movlw B'00000011' ;Port_A digital I/O
0085   0026 0088 movwf ADCON1 ;
0086   0027 3000 movlw 0 ;
0087   0028 0085 movwf PORT_A ;set port a as outputs
0088   0029 30F0 movlw B'11110000' ;RB0-RB3 outputs
0089   002A 0086 movwf PORT_B ;RB4-RB7 inputs
0090   002B 1381 bcf OptionReg, RBPU ;enable pull up
0091   002C 1283 bcf STATUS, RPU ;select page 0
0092   002D 0186 clrf PORT_B ;init port B
0093   002E 0185 clrf PORT_A ;make port a all low
0094   002F 1405 bsf PORT_A, 0 ;make first bit high
0095   0030 118B bcf INTCON, RBIE ;disable mask
0096   0031 0806 movf PORT_B, w ;read port
0097   0032 100B bcf INTCON, RBIF ;clear flag
0098   0033 158B bcf INTCON, RBIE ;enable mask
0099   0034 0009 retfie ;enable global and return
0100   ;
## Implementing Wake-up on Key Stroke

- **16c5x/XX Cross-Assembler V3.05.06 BETA Wed Apr 28 16:09:44 1993 Page 3**

<table>
<thead>
<tr>
<th>Line</th>
<th>PC</th>
<th>Opcode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0102</td>
<td>0102</td>
<td><code>delayl6</code></td>
<td>waits for approx 16.4mSecs using RTCC interrupts</td>
</tr>
<tr>
<td>0103</td>
<td>0103</td>
<td>;fosc speed is 4Mhz.</td>
<td></td>
</tr>
<tr>
<td>0104</td>
<td>0104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0105</td>
<td>0025</td>
<td>1683</td>
<td><code>bsf STATUS,RPO</code></td>
</tr>
<tr>
<td>0106</td>
<td>0036</td>
<td>3007</td>
<td><code>movlw B'00000111'</code></td>
</tr>
<tr>
<td>0107</td>
<td>0037</td>
<td>0081</td>
<td><code>movwf OptionReg</code></td>
</tr>
<tr>
<td>0108</td>
<td>0038</td>
<td>1283</td>
<td><code>bcf STATUS,RPO</code></td>
</tr>
<tr>
<td>0109</td>
<td>0039</td>
<td>0181</td>
<td><code>clrf RTCC</code></td>
</tr>
<tr>
<td>0110</td>
<td>003A</td>
<td>110B</td>
<td><code>bcf INTCON,RTIF</code></td>
</tr>
<tr>
<td>0111</td>
<td>003B</td>
<td>16B</td>
<td><code>bsf INTCON,RTIE</code></td>
</tr>
<tr>
<td>0112</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0113</td>
<td>003C</td>
<td>1D6B</td>
<td><code>btfss INTCON,RTIF</code></td>
</tr>
<tr>
<td>0114</td>
<td>003D</td>
<td>283C</td>
<td><code>goto CheckAgain</code></td>
</tr>
<tr>
<td>0115</td>
<td>003E</td>
<td>128B</td>
<td><code>bcf INTCON,RTIE</code></td>
</tr>
<tr>
<td>0116</td>
<td>003F</td>
<td>110B</td>
<td><code>bcf INTCON,RTIF</code></td>
</tr>
<tr>
<td>0117</td>
<td>0040</td>
<td>0008</td>
<td><code>return</code></td>
</tr>
<tr>
<td>0118</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0119</td>
<td>0000</td>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>
INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. To address these applications, there is the PIC16CXX microcontroller family of products. This family has numerous peripheral and special features to better address user applications.

One feature is the interrupt on change of the PORTB pins. This "interrupt on change" is caused when any of the RB<7:4> pin, configured as input, changes levels. When used in conjunction with the software programmable weak internal pull-ups, a direct interface to a keypad is possible. This is shown in application note AN552 (Implementing Wake-up on Key Stroke). Another way to use the "interrupt on change" feature would be as additional external interrupt sources. This allows the PIC16CXX devices to support multiple external interrupts, in addition to the INT pin.

This application note will discuss some of the issues in using PortB as additional external interrupt pins, and will show some examples. These examples can be easily modified to suit your particular needs.

USING A PORTB INPUT FOR AN EXTERNAL INTERRUPT

The interrupt source(s) cannot simply be directly connected to the PortB pins, and expect the interrupt to function the same as the interrupt (INT) pin. The characteristic of the interrupt signal must also be known to develop the microcontrollers hardware/software. After we know this, we can determine the best way to structure the program to handle this signal. These characteristics include:

1. Trigger interrupt on rising, falling, or both edges
2. What is the pulse width of the interrupt (high time / low time)

It is easy to understand the need of knowing which edge triggers the external interrupt service routine. This allows one to ensure that the interrupt service routine is only entered for the desired edge, with all other edges ignored. Not so clear is pulse width of the interrupt. This determines the amount of additional overhead that the software routine may need.
Figure 1 shows the two cases for the interrupt signal versus the time to complete the interrupt service routine. The first waveform is when the signal makes the low-to-high transitions before the interrupt service routine has completed (interrupt flag cleared). When the interrupt flag has been cleared the interrupt signal has already returned to the inactive level. The next transition of the signal is due to another interrupt request. An interrupt signal with this characteristic will be called a small pulse width signal. The second waveform is when the signal only makes the low-to-high transitions before the interrupt service routine has completed (interrupt flag cleared). The next transition (high-to-low) will return the interrupt signal to the inactive level. This will generate a “false” interrupt, that will need to be cleared. Then the following transition (low-to-high) will be a “true” interrupt. An interrupt signal with this characteristic will be called a wide pulse width signal.

An interrupt pulse with a small pulse width requires less overhead than a wide pulse width. A small pulse width signal must be less then the minimum execution time of the interrupt service routine, while a wide pulse width must be greater then the maximum time through the interrupt service routine.

Example 1 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a small pulse width. In this case since the interrupt pulse width is small, the pulse has gone high and then low again before PortB is read to end the mismatch condition. So when PortB is read it will read a low signal and will again be waiting for the rising edge transition.

***FIGURE 1 - INTERRUPT STEPS FOR SMALL AND WIDE PULSE WIDTHS***

**Small Pulse Width**
- RBx
- Rising Edge
- Triggers Interrupt
- PortB Interrupt Service Routine is complete.
- PortB Interrupt Flag is cleared, and mismatch is ended.
- Wait for next Interrupt edge.
- Signal returns to “Inactive State”

**Large Pulse Width**
- RBx
- Rising Edge
- Triggers Interrupt
- PortB Interrupt Service Routine is complete.
- PortB Interrupt Flag is cleared, and mismatch is ended.
- Wait for next Interrupt edge.
- Falling Edge
- Triggers “False” Interrupt
- PortB Interrupt Service Routine is complete.
- PortB Interrupt Flag is cleared, and mismatch is ended.
- Wait for “False” interrupt edge.

**Example 1: Single Interrupt with a Small Pulse Width**

```assembly
PER_INT BTFSS INTCON, RBIF
  GOTO OTHER_INT
  ; PortB interrupt?
  ; Other interrupt
  ; Do task for INT on RB7
  
  CLR_RBINTF
  MOVF PORTB, 1
  BCF INTCON, RBIF
  RETFIE
  ; Read PortB (to itself) to end mismatch condition
  ; Clear the RB interrupt flag.
  ; Return from interrupt
  ; Do what you need to here

OTHER_INT
  RETFIE
  ; Return from interrupt
```

© 1993 Microchip Technology Inc.
Example 2 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a wide pulse width. In this case since the interrupt pulse width is large, the pulse is still high before PortB is read to end the mismatch condition. So when PortB is read it will read a high signal and will generate an interrupt on the next falling edge transition (which should be ignored).

Example 3 shows a interrupt on change with the interrupt source on PortB (RB7). This executes the interrupt service routine on a both edges. The interrupt source must have a minimum pulse width to ensure that both edges can be "seen". The minimum pulse width is the maximum time from the interrupt edge to the reading of PortB and clearing the interrupt flag.

**Example 2: Single Interrupt with a Wide Pulse Width**

```
PER_INT BTFSS INTCON, RBIF
   ; PortB interrupt?
GOTO OTHER_INT
   ; Other interrupt
BTFSC PORTB, RB7
   ; Check for rising edge
GOTO CLR_RBINTF
   ; Falling edge, clear PortB int
CLR_RBINTF MOVF PORTB, 1
   ; Read PortB (to itself) to end
BCF INTCON, RBIF
   ; mismatch condition
RETFIE
   ; Clear the RB interrupt flag.
OTHER_INT RETFIE
   ; Return from interrupt
   ; Do what you need to here
   ; Return from interrupt
```

**Example 3: Interrupt on Change**

```
PER_INT BTFSS INTCON, RBIF
   ; PortB interrupt?
GOTO OTHER_INT
   ; Other interrupt
CLR_RBINTF MOVF PORTB, 1
   ; Read PortB (to itself) to end
BCF INTCON, RBIF
   ; mismatch condition
RETFIE
   ; Clear the RB interrupt flag.
RETFIE
   ; Return from interrupt
OTHER_INT : RETFIE
   ; Do what you need to here
   ; Return from interrupt
```
PortB as External Interrupt

Using PortB Inputs for Multiple Interrupts

The previous examples have been for a single external interrupt on PORTB. This can be extended to support up to 4 external interrupts. To do this requires additional software overhead, to determine which of the PortB pins (RB<7:4>) caused the interrupt. Care should be taken in the software to ensure that no interrupts are lost.

In this example, the interrupt sources on RB7, RB5, and RB4 have a small pulse width, while the interrupt source on pin RB6 is wide and should cause a trigger on the rising edge.

Example 4: Multiple Interrupts with Different Pulse Widths

```Assembly
PER_INT BTFSS INTCON, RBIF
    ; PortB interrupt?
    GOTO OTHER_INT
    ; Other interrupt
    ; PortB change interrupt has occurred. Must determine which pin caused
    ; interrupt and do appropriate action. That is service the interrupt,
    ; or clear flags due to other edge.

    ; Move PortB value to the W register
    MOVF PORTB, 0
    ; This ends mismatch conditions
    MOVF TEMP, 1
    ; Need to save the PortB reading.
    XORWF LASTPB,
    ; XOR last PortB value with the new
    BTFSC CK_R7
    ; PortB value.
    BTFSC CK_R6
    ; Did pin RB7 change
    CALL RB7_CHG
    ; RB7 changed and caused the interrupt
    BTFSC CK_R5
    ; Did pin RB6 change
    CALL RB6_CHG
    ; RB6 changed and caused the interrupt
    BTFSC CK_R4
    ; Did pin RB5 change
    CALL RB5_CHG
    ; RB5 changed and caused the interrupt
    BTFSC CK_RB4
    ; Did pin RB4 change
    CALL RB4_CHG
    ; RB4 changed and caused the interrupt

    ; Do task for INT on RB7
    RB7_CHG
    ;
    RETURN

RB6_CHG BTFSC PORTB, RB6
    ; Check for rising edge
    RETURN
    ; Falling edge, Ignore
    ; Do task for INT on RB6

    ; Do task for INT on RB5
    RB5_CHG
    ;
    RETURN

    ; Do task for INT on RB4
    RB4_CHG
    ;
    RETURN

CLR_RBINTF MOVF TEMP, 0
    ; Move the PortB read value to the
    MOVF LASTPB
    ; register LASTPB
    BCF INTCON, RBIF
    ; Clear the RB interrupt flag.
    RETFIE
    ; Return from interrupt

    ; Do what you need to here
    OTHER_INT
    ;
    RETFIE
    ; Return from interrupt
```

SUMMARY

The PortB interrupt on change feature is both a very convenient method for direct interfacing to an external keypad, with no additional components, but is also versatile in its uses. The ability to add up to four additional external interrupt. Of course hybrid solutions are also possible. That is, for example, using PORTB<6:1> as a 3x3 keypad, with PORTB7 as an external interrupt and PORTB0 as a general purpose I/O. The flexibility of this feature allows the user to implement a best fit design for the application.

AUTHOR: Mark Palmer, Logic Products Division
INTRODUCTION

To access data in Program Memory, a table read operation must be performed. The Table consists of a series of \texttt{RETLW \textit{K}} instructions where, the 8 bit table constants are assigned to the literal \textit{K}. The first instruction in the Table computes the offset to the table by using "addwf pcl" and consequently the program branches to the appropriate \texttt{RETLW \textit{X}} instruction (see Example 1).

Example 1.

\begin{verbatim}
movlw offset ;load offset in w reg
call Table

Table:
addwf pcl ;add offset to pc to
 ;generate a computed goto
\texttt{RETLW 'A'} ;return the ASCII char A
\texttt{RETLW 'B'} ;return the ASCII char B
\texttt{RETLW 'C'} ;return the ASCII char C
\end{verbatim}

The method is straightforward, however certain precautions have to be exercised when doing a Table read in the PIC16CXX.

IMPLEMENTATION

Program Counter Loading

The PC in the PIC16CXX is 13 bits wide. The low 8 bits (PCL) are mapped in RAM at location 02 and are directly readable and writable. The high 5 bits are not accessible directly and can only be written through the PCLATH (see Figure 1). The PCLATH is a r/w register with only 5 of its bits implemented <4:0>, all other bits are read as '0'.

FIGURE 1 - LOADING OF PC IN DIFFERENT SITUATIONS
SECTION 1

Call and Goto Instructions

When executing a call or goto, the low 11 bits are loaded directly from the instruction opcode. The high 2 bits are loaded from bits 3 and 4 of the PCLATH. It is a good practice to pre-load the PCLATH with the high byte of the location of the routine before executing the routine. This can be done as follows:

Example 2.

```
movlw HIGH Table ;load high 8 bit
                  ;address of Table
movwf PCLATH     ;into PCLATH
call Routine     ;execute Call
                  ;instruction
```

Note: If the program memory size is less than 2K bytes, then the above precaution is not necessary.

Computed Goto Instruction

Any instruction with PCL as the destination, will load the PCH with the 5 low bits from the PCLATH (see Fig 1). In Program 3, if the address where the call was made, were on page 0 and the address of the actual table were on say page 3, then on executing the computed goto, the program will goto a location in page 0 instead of a location on page 3. To avoid the program from branching "erratically" when doing a table read, the PCLATH should be pre-loaded with the high byte of the "Table" address. Example 3 shows how this can be done.

Example 3.

```
org Ox80           ;code location in page 0
movlw offset      ;load offset in w reg
call Table
    ;execute Call
    ;instruction

org 0x0320         ;Table located in page 3
Table
addwf pcl          ;add offset to pc to
                   ;generate a computed goto
retlw 'A'           ;return the ASCII char A
retlw 'B'           ;return the ASCII char B
retlw 'C'           ;return the ASCII char C
```
When doing a computed goto for a table read, care should be taken about page boundaries. The ADDWF PCL instruction will not compute a value greater than 8 bits. In Example 4, the result of the computed goto will result in a branch to an unintended portion of the code. The user either has to be cautious as to where in a page the Table is resident or will have to monitor page roll-over and add it to the PCLATH ahead of the computed goto.

Example 4.

```
org 0x80 ;code location in page 0
movlw HIGH Table ;load PCLATH with hi address
movwf PCLATH ;
movlw offset ;load offset in w reg
call Table

org 0x02ff ;Table located end of page 2
Table:
addwf pcl ;value in pc will not roll over to page 3
retlw 'A' ;return the ASCII char A
retlw 'B' ;return the ASCII char B
retlw 'C' ;return the ASCII char C
```

To take care of both table location and page boundary crossing, it is necessary to do a 13 bit computed goto operation as shown in Example 5.

The code in Example 5 will allow the user to place and access a table anywhere in program memory.

Example 5

```
movlw LOW Table ;get low 8 bits of address
addwf offset ;do a 8 bit add operation
movlw HIGH Table ;get high 5 bits of address
btfsc status,c ;page crossed?
addlw 1 ;yes then inc high address
movwf PCLATH ;load high address in latch
movf offset,w ;load computed offset in w reg
call Table

Table:
addwf pcl ;load computed offset in pcl
retlw 'A' ;return the ASCII char A
retlw 'B' ;return the ASCII char B
retlw 'C' ;return the ASCII char C
```

**Interruption**

Interruptions are handled like a call, i.e. the present PC+1 address is saved on the stack and the program vectors to location 0x04. In Example 5 if the interrupt occurs just before the "movwf pcl" instruction (at label "Table"), then the present fetched instruction will be executed, i.e. movwf pcl, the new computed PC will be incremented and saved on stack (as the return from interrupt address) and the program will vector to the interrupt vector. On return from interrupt the program will go to the intended offset of the table + 1. This is a very undesirable result, so interrupts must be disabling during a table read operation.
SECTION 2

**Differences with PIC16C5X Code**

The PIC16C5X has no PCH or PCLATH register, so the user has to take into consideration all the precautions mentioned in Section 1. In PIC16C5X, the location of the Table has to be in the top half of a 512 byte page. This restriction is not valid for the PIC16CXX family. To convert a table read operation from PIC16C5X code to the PIC16CXX code, the following should be done:

1. Remove any program memory bank select instructions (PIC16C56/57) if present. These are not necessary for the PIC16CXX.

2. Do a 13 bit computed goto operation (as shown in Example 5), when doing a table read operation.

**Differences with PIC17C42**

Unlike the PIC16CXX family, the PIC17C42 loads the PCLATH with the PCH value during a call or goto operation, however a computed goto, does not take care of page boundary crossing. A 16 bit computed jump address should be calculated before the table read is executed. Example 6 show how this is done.

**Example 6**

```
 movlw LOW Table ;get low 8 bits of address
 addwf offset ;do a 8 bit add operation
 movlw HIGH Table ;get high 8 bits of address
 btfsc status, c ;page crossed?
 addlw 1 ;yes then inc high address
 movwf PchBuffer ;load in temp location
 call Table

 Table:
 movf PchBuffer, w; get high offset
 movwf PCLATH ;load in latch
 movf offset, w ;get low offset
 movwf pcl ;load computed offset in PCL
 retlw 'A' ;return the ASCII char A
 retlw 'B' ;return the ASCII char B
 retlw 'C' ;return the ASCII char C
```

Example 6 allows the user to locate his Table at any program memory location, however for large table or tables which cross page boundaries, it is recommended that the "tablrd / tlrd" instruction be used, these instructions are specific for table read operations and are very code efficient.

As mentioned in Section 1, interrupts must be disabled during a table read using the "retlw K" instruction (Example 6). Note it is not necessary to disable interrupts when using the "tablrd/tlrd" instruction for table read operations.

**AUTHOR: Stan D'Souza, Logic Products Division**
**INTRODUCTION**

A set of software routines for Microchip's mid-range, high speed 8-bit EPROM-based microcontrollers to implement I²C Bus Master Mode is given. Some of the members of PIC16CXX family (e.g., PIC16C71, PIC16C84) do not have an on-chip hardware implementation of I²C Bus interface. This application note describes the software implementation of I²C interface routines. Only the master mode of I²C interface is implemented with the PIC16CXX Microcontroller being the only master and communicating to multiple I²C slaves.

This application note does not describe the I²C Bus specifications and the user is assumed to have an understanding of the I²C Bus. For detailed information on the bus, the user is advised to read the I²C Bus Specification document from Philips/Signetics (order number 98-8080-575). The I²C Bus is a 2-wire serial bus with multiple masters and multiple slaves connected to these two wires. The two wires consist of a clock line (SCL) and a data line (SDA) with both the lines being bi-directional. Bi-directional communication is facilitated through the use of wire-and connection (the lines are either active-low or passive high). The I²C Bus protocol also allows collision detection, clock synchronization and hand-shaking for multi-master systems. The clock is always generated by the master, but the slave may hold it low to generate a wait state.

In most of the systems the microcontroller is the master and the external peripheral devices are slaves. In these cases this application note can be used to attach I²C slaves to PIC16CXX (the master) microcontroller. The multi-master system is not implemented because it is extremely difficult to meet all the I²C Bus timing specifications using software. For a true slave or multi-master system, some interface hardware is necessary (like START & STOP bit detection).

In addition to the low-level single master I²C routines, a collection of high level routines with various message structures is given. These high level macros/routines can be used as canned routines to interface to most I²C Slave devices. As an example, the test program talks to two Serial EEPROMs (Microchip's 24LC04 & 24LC01).

**IMPLEMENTATION**

Two levels of software routines are provided. The low-level routines are given in "i2c_low.asm" and the high level routines are given in "i2c_high.asm". The routines are described later. The messages passed (communicated on the two wire network) are abbreviated and certain notation is used to represent Start, Stop and other conditions. These abbreviations are described at first in Table 1.

**TABLE 1 - DESCRIPTION OF ABBREVIATIONS USED**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Start Condition</td>
</tr>
<tr>
<td>P</td>
<td>Stop Condition</td>
</tr>
<tr>
<td>SlvAR</td>
<td>Slave Address (for read operation)</td>
</tr>
<tr>
<td>SlvAW</td>
<td>Slave Address (for write operations)</td>
</tr>
<tr>
<td>A</td>
<td>Acknowledge condition (positive ACK)</td>
</tr>
<tr>
<td>N</td>
<td>Negative Acknowledge condition (NACK)</td>
</tr>
<tr>
<td>D</td>
<td>Data byte, D[0] represents byte 0, D[1] represents second byte</td>
</tr>
</tbody>
</table>
Software Implementation of I^2C Bus Master

**Message Format**

In the high level routines, the basic structure of the message implemented is given. Every I^2C slave supports one or more message structures. For example, Microchip's 24LC04 Serial EEPROM supports the following message (to write a byte to Serial EEPROM at current address counter) S-SlvAW-A.-D-A-P which basically means the following sequence of operations are required:

(a) Send Start Bit
(b) Send Slave Address for Write Operations
(c) Expect Acknowledge
(d) Send Data Byte
(e) Expect Acknowledge
(f) Issue a STOP Condition

**Slave Address**

Both 10 bit and 7 bit addressing schemes are implemented as specified by the I^2C Bus specification. Before calling a certain sub-routine (high level or low-level), the address of the slave being addressed must be loaded using either "LOAD_ADDR_B" (for 7 bit address slaves) or "LOAD_ADDR_B" macro (for 10 bit address slaves). These macros not only load the address of the slaves for all the following operations, but also setup conditions for 7 or 10 bit addressing modes. See the macros section for more details.

**CLOCK STRETCHING**

In I^2C Bus, the clock (SCL Line) is always provided by the master. However, the slave can hold the line low even though the master has released it. The master must check this condition and wait for the slave to release the clock line. This provides a built in wait state for the I^2C Bus. The slave may pull the clock low and ask the master to wait indicating it is busy. This feature is implemented and can be turned on or off as an assembly time option (by setting _ENABLE_BUS_FREE_TIME flag to be TRUE or FALSE). If the clock is held low for too long, say 1 msec, then an error condition is assumed and an RTCC interrupt is generated.

**ARBITRATION**

The I^2C Bus specifies both bit by bit and byte mode arbitration procedure for multi-master systems. However, the arbitration is not needed in a single master system, and therefore not implemented in this application note.

**HARDWARE**

Two I/O pins are used to emulate the Clock Line SCL and the data line SDA. In the example test program, RB0 is used as SCL and RB1 as SDA line. On initialization, these I/O lines are configured as input pins (tri-state) and their respective latches are loaded with 0s. To emulate the high state (passive), these lines are turned as inputs and to emulate the active low state, the pins are turned as outputs (with the assumption of having external pull-up resistors on both the lines).
Software Implementation of I²C Bus Master

I²C ROUTINES

Status Register (File Register “Bus Status”):

The bit definitions of the status register are described in the table given below. These bits reflect the status of the I²C Bus.

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>_Bus_Busy</td>
<td>1 = Start Bit transmitted 0 = STOP condition.</td>
</tr>
<tr>
<td>1</td>
<td>_Abort</td>
<td>It is set when a fatal error condition is detected. The user must clear this bit. This bit is set when Clock Line (SCL) is stuck low.</td>
</tr>
<tr>
<td>2</td>
<td>_Txmt_Progress</td>
<td>1 = transmission in progress.</td>
</tr>
<tr>
<td>3</td>
<td>_Rcv_Progress</td>
<td>1 = reception in progress.</td>
</tr>
<tr>
<td>4</td>
<td>_Txmt_Success</td>
<td>1 = transmission successfully completed. 0 = error condition.</td>
</tr>
<tr>
<td>5</td>
<td>_Rcv_Success</td>
<td>1 = reception successfully completed. 0 = error condition.</td>
</tr>
<tr>
<td>6</td>
<td>_Fatal_Error</td>
<td>1 = FATAL error occurred. The communication was aborted.</td>
</tr>
<tr>
<td>7</td>
<td>_ACK_Error</td>
<td>1 = slave sent not ACK while the master was expecting an ACK. This may happen for example if the slave was not responding to a message.</td>
</tr>
</tbody>
</table>

Control Register (File Register “Bus Control”):

The bit definitions of the control register are described in the table given below. These bits must be set by the software prior to performing certain operations. Some of the high level routines described later in this section set these bits automatically.

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10BitAddr</td>
<td>1 = 10 bit slave addressing 0 = 7 bit addressing.</td>
</tr>
<tr>
<td>1</td>
<td>_Slave_RW</td>
<td>1 = READ operation 0 = WRITE operation.</td>
</tr>
<tr>
<td>2</td>
<td>_Last_Byte_Rcv</td>
<td>1 = last byte must be received. Used to send not ACK.</td>
</tr>
<tr>
<td>3,4,5</td>
<td>Unused bits</td>
<td>Unused bits, can be used as general purpose bits.</td>
</tr>
<tr>
<td>6</td>
<td>_SlaveActive</td>
<td>A status bit indicating if a slave is responding. This bit is set or cleared by calling the PC_TEST_DEVICE macro. See description of this I²C_TESTDEVICE macro.</td>
</tr>
<tr>
<td>7</td>
<td><em>TIME_OUT</em></td>
<td>A status bit indicating if a clock is stretched low for more than 1 msec, indicating a bus error. On this time out, the operation is aborted.</td>
</tr>
</tbody>
</table>
Software Implementation of I²C Bus Master

Low Level:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitI²CBus_Master</td>
<td>Initializes Control/Status Registers, and set SDA &amp; SCL lines. Must be called on initialization.</td>
</tr>
<tr>
<td>TxmtStartBit</td>
<td>Transmits a START (S) condition.</td>
</tr>
<tr>
<td>TxmtStopBit</td>
<td>Transmits a STOP (P) condition.</td>
</tr>
<tr>
<td>LOAD_ADDR_8</td>
<td>The 7 bit slave’s address must be passed as a constant parameter.</td>
</tr>
<tr>
<td>LOAD_ADDR_10</td>
<td>The 10 bit slave’s address must be passed as a constant parameter.</td>
</tr>
<tr>
<td>Txmt_Slave_Addr</td>
<td>Transmits a Slave address. Prior to calling this routine, the address of the slave being addressed must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 routines. Also the Read/Write condition must be set in the control register.</td>
</tr>
<tr>
<td>SendData</td>
<td>Transmits a byte of data. Prior to calling this routine, the byte to be transmitted must be loaded into DataByte file register.</td>
</tr>
<tr>
<td>GetData</td>
<td>Receives a byte of data in DataByte file register. If the data byte to be received is the last byte, the _Last_Byte_Rcv bit in control register must be set prior to calling this routine.</td>
</tr>
</tbody>
</table>
Software Implementation of I²C Bus Master

MACROS

**High Level:**

The high level routines are implemented as a mixture of function calls and macros. These high level routines call the low level routines described above. In most cases only a few of the high level routines may be used and the user can remove or not include the routines not necessary to conserve program memory space. Examples are given for a few functions.

**FC_TEST_DEVICE**

| Parameters | None |
| Purpose    | To test if a slave is present on the network |
| Description | Before using this macro, the address of the slave being tested must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 macro. Is the slave under test is present, then the "SlaveActive" status bit (in Bus_Control file register) is set. If not, then this bit is set 0, indicating that the slave is either not present on the network or is not listening. |
| Message    | S-SlvAW-A-P |
| Example    | LOAD_ADDR_8 0xA0 ; 24LC04 address  
FC_TEST_DEVICE  
btfss _SlaveActive ; See If slave is responding  
goto SlaveNotPresent ; 24LC04 is not present  
; Slave is present  
; Continue with program |

**FC_WR**

| Parameters | _BYTES_, _SourcePointer_ |
| Purpose    | A basic macro for writing a block of data to a slave |
| Description | This macro writes a block of data (no of bytes = _BYTES_) to a slave. The starting address of the block of data is _SourcePointer_. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Txmt_Success bit) |
| Message    | S-SlvAW-A-D[0]-A....A-D[N-1]-A-P |
| Example    |  
btfsc _Bus_Busy ; Check if bus is free  
goto $-1  
LOAD_ADDR_8 _Slave_1.Addr  
FC_WR 0xF9, DataBegin ; |
**FC_WR_SUB**

Parameters:  
BYTES, SourcePointer, Sub_Address

BYTES: Number of bytes starting from RAM pointer.
SourcePointer: Data Start Buffer pointer in RAM (file registers).
Sub_Address: Sub-address of the Slave.

Purpose: Write a block of data to a slave starting at slave's sub-addr.

Description: Same as I2C_WR function, except that the starting address of the slave is also specified. For example, while writing to an I2C Memory Device, the sub-addr specifies the starting address of the memory. The I2C_WR may prove to be more efficient than this macro in most situations. Advantages will be found for Random Address Block Writes for Slaves with Auto Increment Sub-addresses (like Microchip's 24CXX series Serial EEPROMs).


Example:
LOAD_ADDR_8 Slave_2_Addr ; Load addr of 7 bit slave
I2C_WR_SUB 0x08, DataBegin+1, Ox30
In the above example, 8 Bytes of data starting from addr (DataBegin+1) is written to 24LC04 Serial EEPROM beginning at 0x30 address.

**FC_WR_SUB_SWINC**

Parameters:  
BYTES, SourcePointer, Sub_Address

BYTES: Number of bytes starting from RAM pointer.
SourcePointer: Data Start Buffer pointer in RAM (file registers).
Sub_Address: Sub-address of the Slave.

Purpose: Write a block of data to a slave starting at slave's sub-addr.

Description: Same as I2C_WR_SUB function, except that the sub-address (incremented) is sent after every data byte. A very inefficient message structure and the Bus is given up after each data byte. This is useful for when the slave does not have an auto-increment sub-address feature.

Message:  
S-SlvAW-A-(SubA+0)-A-D[0]-A-P
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
and so on until #of Bytes
**FC_WR_BYTE_MEM**

Parameters:

- `BYTES_`, `SourcePointer_`, `Sub_Address_`
  
  `BYTES_` Number of bytes starting from RAM pointer `SourcePointer_`

- `SourcePointer_` Data Start Buffer pointer in RAM (file Registers)

- `Sub_Address_` Sub-address of the Slave

Purpose:

Write a block of data to a slave starting at slave's sub-address

Description:

Same as `FC_WR_SUB_SWINC`, except that a delay is added between each message. This is necessary for some devices like EEPROMs which accept only a byte at a time for programming (devices without on chip ram buffer) and after each byte a delay is necessary before a next byte is written.

Message:

- `S-SlvA-W-A-(SubA+0)-A-D[0]-A-P`
  
  Delay 1 msec

- `S-SlvA-W-A-(SubA+1)-A-D[1]-A-P`
  
  Delay 1 msec

... and so on until # of Bytes

**FC_WR_BUF_MEM**

Parameters:

- `BYTES_`, `SourcePointer_`, `Sub_Address_`, `Device_BUF_SIZE_`
  
  `BYTES_` Number of bytes starting from RAM pointer `SourcePointer_`

- `SourcePointer_` Data Start Buffer pointer in RAM (file Registers)

- `Sub_Address_` Sub-address of the Slave

- `Device_BUF_SIZE_` the slaves on-chip buffer size

Purpose:

Write a block of data to a slave starting at slave’s sub-address

Description:

This Macro/Function writes # of `BYTES_` to an `I²C` memory device. However some devices, esp. EEPROMs must wait while the device enters into programming mode. But some devices have an on-chip temp. data hold buffer and is used to store data before the device actually enters into programming mode. For example, the 24C04 series of Serial EEPROMs from Microchip have an 8 byte data buffer. So one can send 8 bytes of data at a time and then the device enters programming mode. The master can either wait until a fixed time and then retry to program or can continuously poll for ACK bit and then transmit the next Block of data for programming.

Message:

`I²C_SUB_WR` operations are performed in loop and each time data buffer of `BUF_SIZE` is output to the device. Then the device is checked for busy and when not busy another block of data is written.
Software Implementation of I²C Bus Master

**FC_READ**

**Parameters** :

- `_BYTES_`, `_DestPointer_`
  - `_BYTES_` : Number of bytes starting from RAM pointer `_SourcePointer_`
  - `_DestPointer_` : Data Start Buffer pointer in RAM (file Registers)

**Purpose** :
A basic macro for reading a block of data from a slave

**Description** :
This macro reads a block of data (no of bytes = `_BYTES_`) from a slave. The starting address of the block of data is `_DestPointer_`. If an error occurs, the message is aborted and the user must check Status flags (e.g. `_Rcv_Success` bit). Note that on the last byte to receive, NACK is sent.

**Message** :
`S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P`

**Example** :

```c
LOAD_ADDR_10 _Slave_3_Addr
I²C_READ 8, DataBegin
btfss _Rcv_Success
goto ReceiveError
goto ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10 bit slave and stored in the master’s ram starting at address DataBegin.
Software Implementation of I²C Bus Master

**FC_READ_SUB**

**Parameters**:

- \_BYTES\_
- \_DestPointer\_
- \_SubAddress\_

- \_BYTES\_: Number of bytes starting from RAM pointer \_SourcePointer\_.
- \_DestPointer\_: Data Start Buffer pointer in RAM (file Registers).
- \_SubAddress\_: Sub-address of the slave.

**Purpose**:

A basic macro for reading a block of data from a slave.

**Description**:

This macro reads a block of data (no of bytes = \_BYTES\_) from a slave starting at slave's sub-address \_SubAddress. The data received is stored in master's ram starting at address \_DestAddress. If an error occurs, the message is aborted and the user must check Status flags (e.g., \_Rcv\_Success bit).

This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address between the sub-address write & the following reads, a STOP condition is not issued and a "REPEATED START" condition is used so that another master will not take over the bus, and also that no other master will overwrite the sub-address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a memory device (e.g. : 24CXX serial of Serial EEPROMs from Microchip).

**Message**:


**Example**:

```assembly
LOAD_ADDR_10 _Slave_3.Addr
i²C_READ_SUB 8, DataBegin, 0x60
btfss _Rcv_Success
    goto ReceiveError
    goto ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10 bit slave (starting at address 0x60) and stored in the master's ram starting at address DataBegin.

**FC_READ_BYTE or FC_READ_STATUS**

**Parameters**:

- \_DestPointer\_

- \_DestPointer\_: Data Start Buffer pointer in RAM (file Registers).

**Purpose**:

To read a Status Byte from Slave.

**Description**:

Several I²C Devices can send a Status Byte upon reception of the control byte. This Macro reads a Status byte from a slave to the master's RAM location \_DestPointer\. This function is basically the same as i²C_READ for a single byte read. As an example of this command, the 24Cxx serial Serial EEPROM from Microchip will send the memory data at the current location when i²C_READ_STATUS function is called. On successful operation of this command, WREG = 1 else WREG = 0 on errors.

**Message**:

S-SlvAR-A-D-A-N-P

© 1993 Microchip Technology Inc. DS00554A-page 9
Software Implementation of I²C Bus Master

**FC_WR_SUB_WR**

Parameters :

- _Bytes1_
- _SrcPtr1_
- _SubAddr_
- _Bytes2_
- _SrcPtr2_

_Purpose:_ To send two blocks of data to a slave at its sub address

_Description:_ This Macro writes two blocks of data (of variable length) starting at slave’s sub-address _SubAddr_. This Macro is essentially the same as calling I²C_WR_SUB twice, but a STOP bit is not sent in between the transmission of the two blocks. This way the Bus is not given up.

This function may be useful for devices which need two blocks of data in which the first block may be an extended address of a slave device. For example, a large I²C memory device, or a teletext device with an extended addressing scheme, may need multiple bytes of data in the first block that represents the actual physical address and is followed by a second block that actually represents the data.

_Message:_ S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-....-A-D2[M-1]-A-P

**FC_WR_SUB_RD**

Parameters :

- _Count1_
- _SrcPtr_
- _SubAddr_
- _Count2_
- _DestPtr_

_Purpose:_ To send a block of data and then receive a block of data

_Description:_ This macro writes a block of data (of length _Count1_) to a slave starting at sub-address _SubAddr_ and then reads a block of data (of length _Count2_) to the master’s destination buffer (starting at address _DestPtr__). Although this operation can be performed using previously defined Macros, this function does not give up the bus in between the block writes and block reads. This is achieved by using the Repeated Start Condition.

Software Implementation of I²C Bus Master

**FC_WR_COM_WR**

Parameters:
- `_Count1_`, `_SrcPtr1_`, `_Count2_`, `_SrcPtr2_`
  - `_Count1_` Length of the first data block
  - `_SrcPtr1_` Source pointer of the first data block
  - `_Count2_` Length of the second data block
  - `_SrcPtr2_` Source pointer of the second data block

Purpose: To send two blocks of data to a slave in one message.

Description:
This macro writes a block of data (of length `_Count1_`) to a slave and then sends another block of data (of length `_Count2_`) without giving up the bus.

For example, this kind of transaction can be used in an I²C LCD driver where a block of control & address information is needed and then another block of actual data to be displayed is needed.

Message:

```
S-SlvW-A-D1[0]-A-......-A-D1[N-1]-A-D2[0]-A-......-A-D2[M-1]-A-P
```

**APPLICATIONS**

The I²C Bus is a simple two wire serial protocol for inter-IC communications. A lot of peripheral devices (acting as slaves) are available in the market with I²C interface (e.g. serial EEPROM, clock/calendar, I/O Port expanders, LCD drivers, A/D converters, etc.). Although some of the PIC16CXX devices do not have an on chip I²C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I²C bus can be implemented using software.

**AUTHOR:** Amar Palacherla,
Logic Products Division
Appendix A - I'CH

;******************************************************************************
; I2C Bus Header File
;******************************************************************************

_CLKOut equ (_ClkIn >> 2)

; Compute the delay constants for setup & hold times

_40uS_Delay set (_ClkOut/250000)
_47uS_Delay set (_ClkOut/212766)
_50uS_Delay set (_ClkOut/200000)

#define _OPTION_INIT (0xC0 | 0x03) ; Prescaler to RTC for Appox 1 mSec timeout
#define SCL _ portb,0
#define SDA _ portb,1
#define SCL_TRIS trisb,0
#define SDA_TRIS trisb,1
#define WRITE_ 0
#define READ_ 1

; Register File Variables

CBLOCK 0x0C
SlaveAddr ; Slave Addr must be loader into this reg
SlaveAddrHi ; for 10 bit addressing mode
DataByte ; load this reg with the data to be transmitted
BitCount ; The bit number (0:7) transmitted or received
Bus_Status ; Status Reg of I2C Bus for both TXMT & RCVE
Bus_Control ; control Register of I2C Bus
DelayCount
DataByteCopy ; copy of DataByte for Left Shifts (destructive)
SubAddr ; sub-address of slave (used in I2C_High.ASM)
SrcPtr ; source pointer for data to be transmitted
tempCount ; a temp variable for scratch RAM
StoreTemp_1 ; a temp variable for scratch RAM, do not disturb contents
End_I2C_Ram ; unused, only for ref of end of RAM allocation

ENDC
;**********************************************************************************************************
; I2C Bus Status Reg Bit Definitions
;**********************************************************************************************************
#define _Bus_Busy
#define _Abort
#define _Txmt_Progress Bus_Status, 0
#define _Rcv_Progress Bus_Status, 1
#define _Txmt_Success Bus_Status, 2
#define _Rcv_Success
#define _Fatal_Error
#define _ACK_Error
#define_ ACK_Error
#define _Bus_Busy
#define _Abort
#define _Txmt_Progress Bus_Status, 0
#define _Rcv_Progress Bus_Status, 1
#define _Txmt_Success Bus_Status, 2
#define _Rcv_Success
#define _Fatal_Error
#define _ACK_Error
;**********************************************************************************************************
; I2C Bus Control Register
;**********************************************************************************************************
#define _lOBitAddr
#define _Slave_RW
#define _Last_Byte_Rcv Bus_Control, 0
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 1
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 2
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 3
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 4
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 5
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 6
#define _SlaveActive
#define _TIME_OUT_ Bus_Control, 7
;**********************************************************************************************************
; General Purpose Macros
;**********************************************************************************************************
RELEASE_BUS MACRO
bsf _rp0 ; select page 1
bsf _SDA ; tristate SDA
bsf _SCL ; tristate SCL
bsf _Bus_Busy ; Bus Not Busy, TEMP ?????, set/clear on Start & Stop
ENDM
;**********************************************************************************************************
; A MACRO To Load 8 OR 10 Bit Address To The Address Registers
; SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending on 8 or 10 bit addressing modes
;**********************************************************************************************************
LOAD_ADDR_10 MACRO SLAVE_ADDRESS
bsf _lOBitAddr ; Slave has 10 bit address
movlw (SLAVE_ADDRESS & 0xff)
movwf SlaveAddr
movlw (((SLAVE_ADDRESS >> 7) & 0x06) | 0xFO)
movwf SlaveAddr+1

ENDM

LOAD_ADDR_8 MACRO SLAVE_ADDRESS
bcf 10BitAddr
movlw (SLAVE_ADDRESS & 0xff)
movwf SlaveAddr

ENDM
Appendix B - TEST.ASM

Title "I2C Master Mode Implementation"
Subtitle "Rev 0.1 : 01 Mar 1993"

;****************************************************************************************
; Software Implementation Of I2C Master Mode
; * Master Transmitter & Master Receiver Implemented in software
; * Slave Mode implemented in hardware
; * Refer to Signetics/Philips I2C-Bus Specification
; The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
; RBl is SDA (Any I/O Pin May Be used instead)
; RBO/INT is SCL (Any I/O Pin May Be used instead)

Processor 16C71
Radix DEC

_ClkIn   equ 16000000  ; Input Clock Frequency Of PIC16C71

#include "d:\pictools\16Cxx.h"

#define Slave_1.Addr 0xA0  ; Serial EEPROM #1
#define Slave_2.Addr 0xAC  ; Serial EEPROM #2
#define Slave_3.Addr 0xD6  ; Slave PIC16CXX

#define ENABLE_BUS_FREE_TIME TRUE
#define CLOCK_STRETCH_CHECK TRUE
#define INCLUDE_HIGH_LEVEL_I2C TRUE

#include "i2c.h"

CBLOCK End_I2C_Ram
    SaveStatus  ; copy of STATUS Reg
    SaveWReg   ; copy of WREG
    byteCount
    HoldData
ENDC

CBLOCK 0x20
DataBegin
  ; Data to be read or written is stored here
ENDC

ORG 0x00
  ;
  ; ORG 0x04
  ;*********************************************************************************************************
  ; Interrupt Service Routine
  ;
  ; For I2C routines, only RTCC interrupt is used
  ; RTCC Interrupts enabled only if Clock Stretching is Used
  ; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
  ; MUST set _TIME_OUT_ flag saying possibly a FATAL error ocured
  ; The user may choose to retry the operation later again
  ;*********************************************************************************************************

Interrupt:
  ; Save Interrupt Status (WREG & STATUS regs)

  ;
  ; movwf    SaveWReg            ; Save WREG
  ; swapf_   status,w            ; affects no STATUS bits : Only way OUT to save STATUS Reg ????
  ; movwf    SaveStatus          ; Save STATUS Reg
  ; if _CLOCK_STRETCH_CHECK     ; RTCC Interrupts enabled only if Clock Stretching is Used
  ; btfss    rtif                ; other Interrupts
  ; bsf      TIME_OUT_           ; MUST set this Flag, can take other desired actions here
  ; bcf      rtif                ;
  ; endif

  ; Check For Other Interrupts Here, This program useds only RTCC & INT Interrupt
  ;
  ; MaybeOtherInt:
  ;     NOP

  ; Restore Interrupt Status
  ;
  ; movwf    status             ; restore STATUS Reg
  ; swapf    SaveWReg            ; restore WREG
  ; swapf    SaveStatus          ;
  ; retfie

  ;*********************************************************************************************************

  ; Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C

  ; include  "i2c_high.asm"
endif

;*********************************************************************************************************
 ReadSlave1:
 ;
 ; EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
 ;
 LOAD_ADDR 8 Slave1_Addr
 wait1:
 I2C_TEST_DEVICE
 btfss SlaveActive ; See If slave is responding
 goto wait1 ; if stuck for ever, recover from WDT, can use other schemes
 clrwdt
 I2C_READ_SUB 8, DataBegin+1, 0x50

; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
 ;
 LOAD_ADDR 8 Slave2_Addr
 wait2:
 I2C_TEST_DEVICE
 btfss SlaveActive ; See If slave is responding
 goto wait2 ; if stuck for ever, recover from WDT, can use other schemes
 clrwdt
 I2C_READ_SUB 8, DataBegin+1, 0x60

 return

;*********************************************************************************************************
 ReadSlave3:

 LOAD_ADDR 8 Slave3_Addr
 wait3:
 I2C_TEST_DEVICE
 btfss SlaveActive ; See If slave is responding
 goto wait3 ; if stuck for ever, recover from WDT, can use other schemes
 clrwdt
 I2C_READ_SUB 8, DataBegin, 0

 return

;******************************************************************************
 Fill Data Buffer With Test Data ( 8 bytes of Ox55, OxAA pattern)
 ;
FillDataBuf:

```assembly
movlw 0x00 ; start address location of EEPROM array
movwf DataBegin ; lst byte of data to be sent is start address
movlw DataBegin+1 ; data starts following address (RAM Pointer)
movwf Xl
movlw 8 ; fill RAM with 8 bytes , this data is written to EEPROM (slave)
movwf-byteCount
movlw 0x55 ; pattern to fill with is 0x55 & 0xAA
movwf HoldData

X1:
    comf HoldData
    movf HoldData,w
    incf fsr
    decfsz byteCount
    goto X1
    return
```

;**************************************************************************************
;
; Main Routine (Test Program)
;
; SINGLE MASTER, MULTIPLE SLAVES
;
;**************************************************************************************

Start:
call InitI2CBus_Master ; initialize I2C Bus
bsf gie ; enable global interrupts

call FillDataBuf ; fill data buffer with 8 bytes of data (0x55, 0xAA)
;
; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit Addr
;
; Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
;
    btfsc Bus_Busy ; is Bus Free, ie. has a start & stop bit been detected (only for multi master system)
    goto $-1 ; a very simple test, unused for now
LOAD_ADDR_8 _Slave_1_Addr
I2C_WR 0x09, DataBegin
;
; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30

;**************************************************************************************
;
;**************************************************************************
btfsc  Bus_Busy  ; is Bus Free, ie, has a start & stop bit been detected (only for multi master system)
goto  $-1  ; a very simple test, unused for now

LOAD_ADDR_8  _Slave_2_Addr
I2C_WR_SUB  0x08, DataBegin+1, 0x30

call ReadSlave1  ; read a byte from slave from current address

LOAD_ADDR_8  _Slave_3_Addr
movlw 0xCC
movwf DataBegin
I2C_WR_SUB  0x01,DataBegin, 0x33

call ReadSlave3  ; Read From Slave PIC

;clrwdt
goto self  

;****************************************************************************************************''****

END
Appendix C - LOW.ASM

;******************************************************************************
;
; Low Level I2C Routines
;
; Single Master Transmitter & Single Master Receiver Routines
; These routines can very easily be converted to Multi-Master System
; when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
; detection is available.
;
; The generic high level routines are given in I2C_HIGH.ASM
;
;******************************************************************************

;******************************************************************************
;
; I2C Bus Initialization
;
;******************************************************************************

InitI2CBus_Master:
    bcf     rp0
    movf    portb,w
    andlw   0xFC ; do not use BSF & BCF on Port Pins
    movwf   portb ; set SDA & SCL to zero. From Now on, simply play with tris
    RELEASE_BUS
    clrf    Bus_Status ; reset status reg
    clrf    Bus_Control ; clear the Bus_Control Reg, reset to 8 bit addressing
    return

;******************************************************************************
;
; Send Start Bit
;
;******************************************************************************

TxmtStartBit:
    bsf     _rp0 ; select page 1
    bsf     _SDA ; set SDA high
    bsf     _SCL ; clock is high
    ; Setup time for a REPEATED START condition (4.7 uS)
    ; call    Delay40uSec ; only necessary for setup time
bcf _SDA          ; give a falling edge on SDA while clock is high
;
call Delay47uSec  ; only necessary for START HOLD time
;
bsf _Bus_Busy     ; on a start condition bus is busy
;
return

;******************************************************************************
;                Send Stop Bit
;******************************************************************************

TxmtStopBit:
    bsf _rp0        ; select page 1
    bcf _SCL
    bcf _SDA        ; set SDA low
    bcf _SCL        ; Clock is pulled up
    call Delay40uSec ; Setup Time For STOP Condition
    bsf _SDA        ; give a rising edge on SDA while CLOCK is high
;
    if _ENABLE_BUS_FREE_TIME
    ; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
    ;
    call Delay47uSec
    endif
;
    bcf _Bus_Busy    ; on a stop condition bus is considered Free
;
    return

;******************************************************************************
;                Abort Transmission
;******************************************************************************

AbortTransmission:
    call TxmtStopBit
    bsf _Abort
    return

;******************************************************************************
;                Transmit Address (1st Byte)& Put in Read/Write Operation
;******************************************************************************

; Transmits Slave Addr On the 1st byte and set LSB to R/W operation
Slave Address must be loaded into SlaveAddr reg
The R/W operation must be set in Bus Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
On Success, return TRUE in WREG, else FALSE in WREG
If desired, the failure may tested by the bits in Bus Status Reg

.Txmt_Slave_Addr:
  bcf ACK_Error ; reset Acknowledge error bit
  btfs s 10BitAddr
  goto SevenBitAddr
  ;
  btfs s Slave_RW
  goto TenBitAddrWR ; For 10 Bit WR simply send 10 bit addr
  ; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
  ; and then Hi Byte Only for read operation
  ;
  TenBitAddrRd:
  bcf Slave_RW ; temporarily set for WR operation
  call TenBitAddrWR
  btfs s Txmt_Success ; skip if successful
  retlw FALSE
  ;
  call TxmtStartBit ; send A REPEATED START condition
  bcf Slave_RW ; For 10 bit slave Read
  movf SlaveAddr+1,W
  movwf DataByte
  bcf DataByte,LSB ; WR Operation
  ; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)
; then save RETURN Address Pointer
;
call     SendData ; send high byte of 10 bit addr slave
;
; if successfully transmitted, expect an ACK bit
;
  btfss  Txmt_Success ; if not successful, generate STOP & abort transfer
  goto   AddrSendFail
;
  movf   SlaveAddr,W
  movwf   DataByte
  goto   EndTxmtAddr

SevenBitAddr:
  movf   SlaveAddr,W
  movwf   DataByte
  bcf   DataByte,LSB
  btfsc  Slave_RW
  bsf   DataByte,LSB

EndTxmtAddr:
  call     SendData ; send 8 bits of address, bus is our's
;
; if successfully transmitted, expect an ACK bit
;
_AddrSendTest:
  btfss   Txmt_Success
  goto   AddrSendFail
  clrwdt
  retlw   TRUE
;
_AddrSendFail:
  clrwdt
  btfss   ACK_Error
  retlw   FALSE
  ; Addr Txmt Unsuccessful, so return 0
  ; Address Not Acknowledged, so send STOP bit
  ;
call     TxmtStopBit
  retlw   FALSE
  ; Addr Txmt Unsuccessful, so return 0

;*********************************************************************************************************

Transmit A Byte Of Data

; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
; and INT Interrupt on Rising edge is enabled and also RTCC timeout interrupt is enabled
; The clock stretching slows down the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock stretching, then
this feature can be disabled during Assembly time by setting
_CLOCK_STRETCH_ENABLED must be set to FALSE.

SendData:

TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
For future compatibility, the user MUST call SendData & NOT TxmtByte

goto TxmtByte

TxmtByte:

movf DataByte,w
movwf DataByteCopy
bsf Txmt_Progress
bcf Txmt_Success
movlw 0x08
movwf BitCount
bsf _rp0
if CLOCK_STRETCH_CHECK
; set RTCC to INT CLK timeout for 1 mSec
; do not disturb user's selection of RPUB in OPTION Register
movf option,w
andlw OPTION_INIT
movwf option
endif

TxmtNextBit:
clrwdt
bcf SCL
rlf DataByteCopy
bcf _SDA
btfsc c
bsf SDA
call Delay47uSec
bsf SCL
call Delay40uSec
if CLOCK_STRETCH_CHECK
bcf _rp0
clrff rtcc
bcf rtif
bsf rtie
bcf TIME_OUT
endif

Check_SCL_1:
btfsc TIME_OUT
if RTCC timeout or Error then Abort & return
subscribe Bus_Fatal_Error

bcf rp0
btfss SCL ; if clock not being stretched, it must be high
goto Check_SCL_1 ; loop until SCL high or RTCC timeout interrupt
bcf rtie ; Clock good, disable RTCC interrupts
bsf rp0
endif
decfsz BitCount
goto TxmtNextBit
;
; Check For Acknowledge
;
bcf SCL ; reset clock
bsf SDA ; Release SDA line for Slave to pull down
call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
bsf SCL ; clock for slave to ACK
call Delay40uSec ; guarantee min HIGH TIME tHIGH
bcf rp0 ; select PAGE 0 to test PortB pin SDA
btfsc SDA ; SDA should be pulled low by slave if OK
goto _TxmtErrorAck
;
bcf rp0
bcf SCL ; reset clock
bcf Txmt_Progress ; reset TXMT bit in Bus Status
bsf Txmt_Success ; transmission successful
bcf ACK_Error ; ACK OK
return
_TxmtErrorAck:
RELEASE_BUS
bcf Txmt_Progress ; reset TXMT bit in Bus Status
bcf Txmt_Success ; transmission NOT successful
bsf ACK_Error ; No ACK From Slave
return
;
******************************************************************************
;
Receive A Byte Of Data From Slave
;
; assume address is already sent
; if last byte to be received, do not acknowledge slave (last byte is tested from
; _Last_Byte_Rcv bit of control reg)
; Data Received on successful reception is in DataReg register
;
******************************************************************************

GetData:
goto RcvByte
; RcvByte:

    bsf Rcv_Progress                   ; set Bus status for txmt progress
    bcf Rcv_Success                    ; reset status bit

    movlw 0x08                         
    movwf BitCount

    if _CLOCK_STRETCH_CHECK
    bsf _rp0
    ; set RTCC to INT CLK timeout for 1 mSec
    ; do not disturb user's selection of RPUB in OPTION Register
    
    movf option, w
    andlw OPTION_INIT                 ; defined in I2C.H header file
    movwf option

    endif

RcvNextBit:
    clrwdt                             ; clear WDT, set for 18 mSec
    bsf rp0                             ; page 1 for TRIS manipulation
    bcf SCL                             ; can be removed from loop
    call Delay47uSec                    ; guarantee min LOW TIME tLOW & Setup time
    bsf SCL                             ; clock high, data sent by slave
    call Delay40uSec                    ; guarantee min HIGH TIME tHIGH

    if _CLOCK_STRETCH_CHECK
    bcf rp0
    clrdf rtcc                           ; clear RTCC
    bcf rtif                             ; clear any pending flags
    bsf rtie                             ; enable RTCC Interrupt
    bcf TIME_OUT                         ; reset timeout error flag

    Check_SCL_2:
    btfscc TIME_OUT                      ; if RTCC timeout or Error then Abort & return
    goto Bus_Fatal_Error                 ; Possible FATAL Error on Bus
    bcf rp0
    btfs SCL                             ; if clock not being stretched, it must be high
    goto Check_SCL_2                     ; loop until SCL high or RTCC timeout interrupt
    bcf rtie                             ; Clock good, disable RTCC interrupts
    bcf rp0

    endif

    bcf rp0                              ; select page 0 to read Ports
    bcf c
    btfscc SDA                            
    bcf c

    ; TEMP ????? DO 2 out of 3 Majority detect
    rlf DataByte                         ; left shift data (MSB first)
    decfsz BitCount
    goto RcvNextBit
; Generate ACK bit if not last byte to be read,
; if last byte Generate NACK ; do not send ACK on last byte, main routine will send a STOP bit
; 
bsf   rp0
bcf   SCL
bcf   SDA
btfsc Last.Byte_Rcv
bsf   SDA ; if last byte, send NACK by setting SDA high
   call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
   bcf   SCL
   call Delay40uSec ; guarantee min HIGH TIME tHIGH

RcvEnd:
   bcf   SCL ; reset clock
   bcf   Rcv_Success ; transmission successful
   bcf   ACK_Error ; ACK OK
   return
   
   if _CLOCK_STRETCH_CHECK
   ;.Fatal Error On I2C Bus
   ;
   ; Slave pulling clock for too long or if SCL Line is stuck low.
   ; This occurs if during Transmission, SCL is stuck low for period longer than approx 1mS
   ; and RTCC times out ( approx 4096 cycles : 256 * 16 - prescaler of 16).
   ;
Bus_Fatal_Error:
   ; disable RTCC Interrupt
   ;
   bcf   rtie ; disable RTCC interrupts, until next TXMT try
   
RELEASE_BUS
   ;
   ; Set the Bus_Status Bits appropriately
   ;
   bsf   Abort ; transmission was aborted
   bsf   Fatal_Error ; FATAL Error occurred
   bcf   Txmt_Progress ; Transmission Is Not in Progress
   bcf   Txmt_Success ; Transmission Unsuccessful
   ;
   call   TxmtStopBit ; Try sending a STOP bit, may be not successful
   ;
   return
General Purpose Delay Routines

Delay4uS is wait loop for 4.0 uSec
Delay47uS is wait loop for 4.7 uSec
Delay50uS is wait loop for 5.0 uSec

Delay50uSec:
    movlw ((_50uS_Delay-5)/3 + 1)
    DlyK
    movwf DelayCount
    decfsz DelayCount
    goto $-1
    return

Delay47uSec:
    movlw ((_47uS_Delay-8)/3 + 1)
    goto DlyK

Delay40uSec:
    movlw ((_40uS_Delay-8)/3 + 1)
    goto DlyK
Appendix D - HIGH.ASM

;**********************************************************************************************************
; I2C Master : General Purpose Macros & Subroutines
; High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
;**********************************************************************************************************

;**********************************************************************************************************
; I2C TEST DEVICE
; MACRO
;
; If Slave Device is listening, then _SlaveActive bit is set, else is cleared
;
; Parameter : NONE
;
; Sequence Of Operations :
; S-SlvAW-A-P
; If A is +ve device is listening, else either busy, not present or error condition
;
; This test may also be used to check for example if a Serial EEPROM is in internal programming
; mode
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
; mode addressing must be set
;**********************************************************************************************************

I2C_TEST_DEVICE       MACRO
            call   IsSlaveActive            ; TEMP ????? : Assembler Error with this MACRO
            ENDM

; Test If A Device of SlaveAddr Is Present on Bus
;
; The Slave Address Is put on the bus and if ACK it is present, if NACK not present
; or may be device is not responding. The presence can be checked constantly by a master
; (for ex. the Operating System on an Access.Bus may constantly issue this command)
;
; Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
; Set _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
;
; Returns 1 in _SlaveActive Bit if slave is responding else a 0
IsSlaveActive:
  bcf Slave_RW ; set for write operation
  call TxmtStartBit ; send START bit
  call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
  
  bcf SlaveActive
  btfss ACK_Error ; skip if NACK, device is not present or not responding
  bsf SlaveActive ; ACK received, device present & listening
  call TxmtStopBit
  return

I2C_WRITE

; A basic macro for writing a block of data to a slave

; Parameters:
; _BYTES_ #of bytes starting from RAM pointer _SourcePointer_
; _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)

; Sequence:
; S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P

; If an error occurs then the routine simply returns and user should check for
; flags in Bus_Status Reg (for eg. _Txmt_Success flag)

; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
; mode addressing must be set

I2C_WR MACRO _BYTES_, _SourcePointer_

  movlw _BYTES_
  movwf tempCount
  movlw _SourcePointer_
  movwf fsr
  call i2c_block_write
  call TxmtStopBit ; Issue a stop bit for slave to end transmission

ENDM

_i2c_block_write:
  call TxmtStartBit ; send START bit
  bcf Slave_RW ; set for write operation
call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set

_block_wrl_loop:
  btfss Txmt_Success
  return
  movf indf, w
  movwf DataByte ; start from the first byte starting at _DataPointer_
  incf fsr
  call SendData ; send next byte, bus is our's!
  decfsz tempCount
  goto block_wrl_loop ; loop until desired bytes of data transmitted to slave
  return

************************************************************************************************************

I2C_WRITE_SUB

; Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address
; to a slave device.
; Eg. : A serial EEPROM would need an address of memory location for Random Writes
;
; Parameters :
; _BYTES_ #of bytes starting from RAM pointer _SourcePointer_ (constant)
; _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
; _Sub_Address_ Sub-address of Slave (constant)
;
; Sequence :
; S-SlvAV-A-SubA-A-D[0]-A......A-D[N-1]-A-P
;
; If an error occurs then the routine simply returns and user should check for
; flags in Bus_Status Reg (for eg. _Txmt_Success flag
;
; Returns : WREG = 1 on success, else WREG = 0
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
; mode addressing must be set
;
; COMMENTS :
; I2C_WR may prove to be more efficient than this macro in most situations
; Advantages will be found for Random Address Block Writes for Slaves with
; Auto Increment Sub-Addresses (like Microchip's 24CXX series Serial EEPROMs)
;
*****************************************************************************

I2C_WR_SUB MACRO _BYTES_, _SourcePointer_, _Sub_Address_
    movlw (_BYTES_ + 1)
    movwf tempCount
movlw (_SourcePointer_ - 1)
movwf fsr
movf indf, w
movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ -1)
movlw Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ -1)
call i2c_block_write ; write _BYTES_+1 block of data
movf StoreTemp_1, w
movwf (_SourcePointer _ - 1) ; restore contents of (_SourcePointer_ - 1)
call TxmtStopBit ; Issue a stop bit for slave to end transmission
ENDM

;*******************************************************************************;
; I2C_WR_SUB_SWINC
;
; Parameters :
; _BYTES_ #of bytes starting from RAM pointer _SourcePointer_ (constant)
; _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
; _Sub_Address_ Sub-address of Slave (constant)
;
; Sequence :
; S-SlvAW-A-(SubA+0)-A-D[0]-A-P
; S-SlvAW-A-(SubA+1)-A-D[1]-A-P
; and so on until #of Bytes
;
; If an error occurs then the routine simply returns and user should check for
; flags in Bus_Status Reg (for eg. _Txrnt_Success flag
;
; Returns : WREG = 1 on success, else WREG = 0
;
; COMMENTS : Very In-efficient, Bus is given up after every Byte Write
; ; Some I2C devices addressed with a sub-address do not increment automatically
; ; after an access of each byte. Thus a block of data sent must have a sub-address
; ; followed by a data byte.
; ;*******************************************************************************

I2C_WR_SUB_SWINC MACRO _BYTES_, _SourcePointer_, _Sub_Address_

variable i ; TEMP ????: Assembler Does Not Support This

i = 0
.while (i < _BYTES_)
movf (_Source_Pointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call i2c_byte_wr_sub ; write a byte of data at sub address
i++
.endw

ENDM

; ; Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
; ;
_i2c_byte_wr_sub:
call TxmtStartBit ; send START bit
bcf Slave_RW ; set for write operation
call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
btfss Txmt_Success
goto block_wrl_fail ; end
movf SubAddr,w
movwf DataByte ; start from the first byte starting at _DataPointer_
call SendData ; send next byte
btfss Txmt_Success
goto block_wrl_fail ; end
movf SrcPtr,w
movwf DataByte ; start from the first byte starting at _DataPointer_
call SendData ; send next byte
btfss Txmt_Success
goto block_wrl_fail ; failed, return 0 in WREG
goto block_wrl_pass ; successful, return 1 in WREG

; return back to called routine from either _block_wrl_pass or _block_wrl_fail
;
_block_wrl_fail:
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw FALSE
_block_wrl_pass:
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw TRUE

;**********************************************************************************************************

I2C WR MEM BYTE
Some I2C devices like a EEPROM need to wait some time after every byte write
(when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
but in addition adds a delay after each byte.
Some EEPROM memories (like Microchip's 24Cxx Series have on-chip data buffer), and hence
this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
for a block of data and then insert a delay until the whole buffer is written.

Parameters:
BYTES: # of bytes starting from RAM pointer _SourcePointer_ (constant)
_SourcePointer_: Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_: Sub-address of Slave (constant)

Sequence:
S-SlvAW-A-(SubA+0)-A-D[0]-A-P
Delay 1 mSec ; The user can change this value to desired delay
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
Delay 1 mSec
and so on until # of Bytes

I2C_WR_BYTE_MEM MACRO _BYTES_, _SourcePointer_, _Sub_Address_

variable i ; TEMP ????: Assembler Does Not Support This
i = 0

_while (i < _BYTES_)
movf (_Source_Pointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call _I2C_byte_wr_sub ; write a byte of data at sub address
call Delay50uSec
i++

_.endw

ENDM

I2C_WR_MEM_BUF MACRO _BYTES_, _SourcePointer_, _Sub_Address_

variable i ; TEMP ????: Assembler Does Not Support This
i = 0

_while (i < _BYTES_)
movf (_Source_Pointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call _I2C_byte_wr_sub ; write a byte of data at sub address
call Delay50uSec
i++

_.endw

ENDM

This Macro/Function writes # of _BYTES_ to an I2C memory device. However
some devices, esp. EEPROMs must wait while the device enters into programming
mode. But some devices have an on-chip temp data hold buffer and is used to
store data before the device actually enters into programming mode.
For example, the 24C04 series of Serial EEPROMs from Microchip
have an 8 byte data buffer. So one can send 8 bytes of data at a time
and then the device enters programming mode. The master can either wait
until a fixed time and then retry to program or can continuously poll
for ACK bit and then transmit the next block of data for programming

Parameters:
- BYTES_ # of bytes to write to memory
- SourcePointer_ Pointer to the block of data
- SubAddress_ Sub-address of the slave
- Device_BUF_SIZE The on chip buffer size of the i2c slave

Sequence of operations
- I2C_SUB_WR operations are performed in loop and each time
  data buffer of BUF_SIZE is output to the device. Then
  the device is checked for busy and when not busy another
  block of data is written

I2C_WR_BUF_MEM MACRO _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_

variable i, j

if ( !_BYTES_)
    exitm

elif ( _BYTES_ <= _Device_BUF_SIZE )
    I2C_WR_SUB _BYTES_, _SourcePointer_, _SubAddress_
        exitm

else
    i = 0
    j = (_BYTES_ / _Device_BUF_SIZE_)
    while ( i < j )
        I2C_WR_SUB _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ +
        i*_Device_BUF_SIZE_)

call IsSlaveActive
btfs SlaveActive
goto $-2

i++
.endw

j = (_BYTES_ - i*Device_BUF_SIZE_)

if (j)
  I2C_WR_SUB j, (_SourcePointer_ + i*Device_BUF_SIZE_), (_SubAddress_ + i*Device_BUF_SIZE_)
endif

endif

ENDM

;********************************************************************************************************************
; I2C_READ
; The basic MACRO/procedure to read a block message from a slave device
;
; Parameters :
; _BYTES_ : constant : #of bytes to receive
; _DestPointer_ : destination pointer of RAM (File Registers)
;
; Sequence :
; S-SlvAR-A-D[0]-A-......-A-D[N-1]-N-P
;
; If last byte, then Master will NOT Acknowledge (send NACK)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
; mode addressing must be set
;
;***************************************************************************************

I2C_READ MACRO _BYTES_, _DestPointer_

movlw (_BYTES_ -1)
movwf tempCount ; -1 because, the last byte is used out of loop
movlw DestPointer_
movwf fsr ; FIFO destination address pointer
call i2c_block_read

ENDM

_i2c_block_read:
call TxmtStartBit ; send START bit
bsf Slave_RW ; set for read operation
bcf Last_Byte_Rcv ; not a last byte to rcv
call Txmt_Slave.Addr ; if successful, then _Txmt_Success bit is set
btfsc Txmt_Success ; end
goto block_rdl_loop
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw FALSE ; Error: may be device not responding

_block_rdl_loop:
call GetData
movf DataByte,w
movwf indf ; start receiving data, starting at Destination Pointer
incf far
decfsz tempCount
goto block_rdl_loop ; loop until desired bytes of data transmitted to slave
bsf Last_Byte_Rcv
movf DataByte,w
movwf indf
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw TRUE

;**********************************************************************************************************
I2C_READ_SUB
This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address
Between the sub-address write & the following reads, a STOP condition is not issued and
a “REPEATED START” condition is used so that an other master will not take over the bus,
and also that no other master will overwrite the sub-address of the same slave.

This function is very commonly used in accessing Random/Sequential reads from a
memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).

Parameters :
BYTES_ # of bytes to read
_DestPointer_ The destination pointer of data to be received.
_SubAddress_ The sub-address of the slave

Sequence :

;**********************************************************************************************************
I2C_READ_SUB MACRO _BYTES_, _DestPointer_, _SubAddress_
bcf Slave_RW ; set for write operation
call TxmtStartBit ; send START bit
call Txmt_Slave.Addr ; if successful, then _Txmt_Success bit is set
movlw SubAddress_;  ; START address of EEPROM(slave 1)
movwf DataByte_   ; write sub address
; do not send STOP after this, use REPEATED START condition
;
I2C_READ _BYTES_, _DestPointer_

ENDM

;**********************************************************************************************************
;

I2C_READ_STATUS
;
This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
; send a status byte upon reception of a control byte
; This is basically same as I2C_READ MACRO for reading a single byte
;
; For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory
; data at the current address location
;
; On success WREG = 1 else = 0
;
;**********************************************************************************************************

I2C_READ_STATUS MACRO _DestPointer_
call TxmtStartBit_  ; send START bit
bsf Slave_RW_  ; set for read operation
call Txmt_Slave_Addr_  ; if successful, then _Txmt_Success bit is set
btfsc Txmt_Success_  ; read a byte
goto byte_rdl_loop_  ; read a byte

call TxmtStopBit_  ; Issue a stop bit for slave to end transmission
retlw FALSE_  ; Error : may be device not responding

byte_rdl_loop:  ; last byte to rcv, so send NACK
    bsf Last_Byte_Rcv_  ; last byte to rcv, so send NACK
    call GetData_  ; read a byte
    movf DataByte,w_  ; read a byte
    movwf DestPointer_  ; read a byte
    call TxmtStopBit_  ; Issue a stop bit for slave to end transmission
    btfss Rcv_Success_  ; Issue a stop bit for slave to end transmission
    retlw FALSE_  ; Error : may be device not responding
    retlw TRUE_  ; Error : may be device not responding
This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This
may be useful for devices which need 2 blocks of data in which the first block may be an
extended address of a slave device. For example, a large I2C memory device, or a teletext
device with an extended addressing scheme, may need multiple bytes of data in the 1st block
that represents the actual physical address and is followed by a 2nd block that actually
represents the data.

Parameters:

- BYTES1_ 1st block # of bytes
- SourcePointer1_ Start Pointer of the 1st block
- SubAddress_ Sub-Address of slave
- BYTES2_ 2nd block # of bytes
- SourcePointer2_ Start Pointer of the 2nd block

Sequence:

S-SlvW-A-SubA-A-D1[0]-A-...-D1[N-1]-A-D2[0]-A-...-D2[M-1]-A-P

Note: This MACRO is basically same as calling I2C_WR_SUB twice, but
a STOP bit is not sent (bus is not given up) in between
the two I2C_WR_SUB

Check Txmt_Success flag for any transmission errors

I2C_WR_SUB_WR MACRO _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_

movlw (_COUNT1_ + 1)
movwf tempCount
movlw (_SourcePointer1_ - 1)
movwf fsr
movf indf,w
I2C WR SUB RD

; This macro writes a block of data from SourcePointer of length _ COUNT1_ to a slave
; at sub-address and then Reads a block of Data of length _ COUNT2_ to destination
; address pointer
;
; Message Structure :
; S-SlvW-A-SubA-A-Dl[0]-A-....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-....-A-D2[M-1]-N-P
;
; Parameters :
; _ COUNT1_       Length Of Source Buffer
; _SourcePointer_  Source Pointer Address
; _Sub_Address_    The Sub Address Of the slave
; _ COUNT2_       The length of Destination Buffer
; _DestPointer_   The start address of Destination Pointer
;
I2C_WR_SUB_RD MACRO _ COUNT1_, _SourcePointer_, _Sub_Address_, _ COUNT2_, _ DestPointer_

    movlw (_ COUNT1_ + 1) ; temporarily store contents of (_SourcePointer_ -1) 
    movlw Sub Address_ ; store temporarily the sub-address at (_SourcePointer_ -1) 
    movwf indf          ; write _BYTES_+1 block of data
    call i2c_block_write

    movf StoreTemp_1,w  ; Block 1 write over
    movwf (_SourcePointer_ - 1) ; restore contents of (_SourcePointer_ - 1)

    ; Send Block 2
    movlw COUNT2_       
    movwf tempCount     
    movlw SourcePointer2_ 
    movwf fsr            
    call block_wri_loop

    call TxmtStopBit     ; Issue a stop bit for slave to end transmission

ENDM
movwf StoreTemp1       ; temporarily store contents of (_SourcePointer_ -1)
movlw Sub_Address_    ; store temporarily the sub-address at (_SourcePointer_ -1)
call i2c_block_write  ; write _BYTES_+1 block of data

movf StoreTemp1,W    ; temporarily store the sub-address at (_SourcePointer_ -1)
movwf (_SourcePointer_ - 1) ; restore contents of (_SourcePointer_ - 1)

; Without sending a STOP bit, read a block of data by using a REPEATED
; Start Condition
;
I2C_READ  _COUNT2_, _DestPointer_
ENDM

;**********************************************************************************************************

I2C_WR_COM_WR

; This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
; the bus after sending the first block.
; For example, this kind of transaction is used in an LCD driver where a
; a block of control & address info is needed and then another block of actual data
; to be displayed is needed.

Message Structure :
; S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P

NOTE : This message is same as calling two I2C_WR Macros, except that
; the bus is not given up between the sending of 2 blocks (this is
; done by not sending a STOP bit inbetween)
;
Parameters :
; _COUNTl_ Length Of Source Buffer #1
; _SourcePointer1_ Source Pointer Address of 1st buffer
; _COUNT2_ The length of Destination Buffer
; _SourcePointer2_ Source Pointer Address of 2nd Buffer

;**********************************************************************************************************

I2C_WR_COM_WR MACRO _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_

movlw _COUNT1_ 
movwf tempCount 
movlw _SourcePointer1_ 
movwf fsr 
call i2c_block_write 

;
; First block sent, now send 2nd block of data
;
    movlw COUNT2_
    movwf tempCount
    movlw SourcePointer2_
    movwf fsr
    call block_wrl_loop
;
    call TxmtStopBit ; End of Double buffer txmt

ENDM

**************************************************************************

INCLUDE I2C Low Level Routines Here
**************************************************************************

include "i2c_low.asm"
Software Implementation of I2C Bus Master

Title "I2C Master Mode Implementation"
Subtitle "Rev 0.1 : 01 Mar 1993"

;*****************************************************************************
; Software Implementation Of I2C Master Mode
;
; * Master Transmitter & Master Receiver Implemented in software
; * Slave Mode implemented in hardware
;
; * Refer to Signetics/Philips I2C-Bus Specification
;
; The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
;
; RB1 is SDA    (Any I/O Pin May Be used instead)
; RB0/INT is SCL (Any I/O Pin May Be used instead)
;
;*****************************************************************************

Processor 16C71
Radix DEC

0044 2400  _ClkIn  equ  16000000 ; Input Clock Frequency Of PIC16C71

include  "d:\pictools\16Cxx.h"
include "i2c.h"

; I2C Bus Header File

; Software Implementation of I2C Bus Master

; Prescaler to NTCC for approx. 1 sec timeout

; Configure the delay constants for setup & hold times

#define OPTION_INIT
#define SCL -
#define SDA
#define SCL_TRIS -
#define SDA_TRIS
#define WRITE
#define READ

#define 40uS_Delay
#define 47uS_Delay
#define 50uS_Delay

#define _ClkOut

#define _ClkOut/250000
#define _ClkOut / (212766)
#define _ClkOut/200000

#define 0x00
#define 0x02
#define 0x03

#define Portb
#define Portb, 0
#define Portb, 1
#define trisb, O
#define trisb, 1

#define INCLUDE HIGH_LEVEL TRUE
#define ENABLE_BUS_FREE_TIME TRUE
#define CLOCK STRETCH_CHECK TRUE
#define Slave 1 Addr OxA0 Serial EEPROM #1 - - - -
#define Slave 2 Addr OxAC Serial EEPROM #2 - - - -
#define Slave 3 Addr OxD6 Slave PIC16CXX - - - -
#define ENABLE BUS FREE TIME TRUE - - - -
#define CLOCK STRETCH_CHECK TRUE - - - -
CBLOCK Ox0C

000C 0001 SlaveAddr ; Slave Addr must be loader into this reg
000D 0001 SlaveAddrHi ; for 10 bit addressing mode
000E 0001 DataByte ; load this reg with the data to be transmitted
000F 0001 BitCount ; The bit number (0:7) transmitted or received
0010 0001 Bus_Status ; Status Reg of I2C Bus for both TXMT & RCV
0011 0001 Bus_Control ; control Register of I2C Bus
0012 0001 DelayCount
0013 0001 DataByteCopy ; copy of DataByte for Left Shifts (destructive)
0014 0000
0014 0001 SubAddr ; sub-address of slave (used in I2C_HIGH.ASM)
0015 0001 SrcPtr ; source pointer for data to be transmitted
0016 0000
0016 0001 tempCount ; a temp variable for scratch RAM
0017 0001 StoreTemp_1 ; a temp variable for scratch RAM, do not disturb contents
0018 0000
0018 0001 End_I2C_Ram ; unused, only for ref of end of RAM allocation

ENDC

;************************************************************************************
#define _Bus_Busy Bus_Status,0
#define _Abort Bus_Status,1
#define _Txmt_Progress Bus_Status,2
#define _Rcv_Progress Bus_Status,3
#define _Txmt_Success Bus_Status,4
#define _Rcv_Success Bus_Status,5
#define _Fatal_Error Bus_Status,6
#define _ACK_Error Bus_Status,7

;************************************************************************************
; I2C Bus Control Register

#define _10BitAddr Bus_Control,0
#define _Slave_RW Bus_Control,1
#define _Last_Byte_Rcv Bus_Control,2
#define _SlaveActive Bus_Control,3
#define _TIME_OUT_ Bus_Control,7

; General Purpose Macros

RELEASE_BUS MACRO
bsf rp0 ; select page 1
bsf SDA ; tristate SDA
bsf SCL ; tristate SCL
bcf _Bus_Busy ; Bus Not Busy, TEMP ???, set/clear on Start & Stop
ENDM

A MACRO To Load 8 OR 10 Bit Address To The Address Registers

; SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s)
; depending on 8 or 10 bit addressing modes

LOAD_ADDR_10 MACRO SLAVE_ADDRESS
bsf _ 10BitAddr ; Slave has 10 bit address
movlw (SLAVE_ADDRESS & 0xff) ; load low byte of address
movwf SlaveAddr
movlw (((SLAVE_ADDRESS >> 7) & 0x06) | 0xf0) ; 10 bit addr is 11110XX0
movwf SlaveAddr+1 ; hi order address

ENDM

LOAD_ADDR_8 MACRO SLAVE_ADDRESS

bcf _ 10BitAddr ; Set for 8 Bit Address Mode
movlw (SLAVE_ADDRESS & 0xff)
movwf SlaveAddr

ENDM

CBLOCK _End_I2C_Ram

0018 0001 SaveStatus ; copy of STATUS Reg
0019 0001 SaveWReg ; copy of WREG
001A 0001 byteCount
001B 0001 HoldData

ENDC

CBLOCK 0x20

0020 0001 DataBegin ; Data to be read or written is stored here

ENDC

ORG 0x00

0000 2956 goto Start ;

ORG 0x04

;***************************************************************************************************

; Interrupt Service Routine
;
; For I2C routines, only RTCC interrupt is used
; RTCC Interrupts enabled only if Clock Stretching is Used
; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
; MUST set _TIME_OUT_ flag saying possibly a FATAL error occurred
; The user may choose to retry the operation later again

;*********************************************************************************************************

Interrupt:

; Save Interrupt Status (WREG & STATUS regs)

0004 0099 movwf SaveWReg ; Save WREG
0005 0E03 swapf status, w ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098 movwf SaveStatus ; Save STATUS Reg if _CLOCK.Stretch_CHECK RTCC Interrupts enabled only if Clock Stretching is Used
0007 1D0B btfss rtf
0008 280B goto MayBeOtherInt ; other Interrupts
0009 1791 bsf TIME_OUT_; MUST set this Flag, can take other desired actions here
000A 110B bcf rtf
endif

; Check For Other Interrupts Here, This program used only RTCC & INT Interrupt

MayBeOtherInt:

000B 0000 NOP

; Restore Interrupt Status

000C 0E18 swapf SaveStatus, w
000D 0083 movwf status ; restore STATUS Reg
000E OE99 swapf SaveWReg
000F OE19 swapf SaveWReg, w ; restore WREG
0010 0009 retfie

;*********************************************************************************************************

; Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
include "i2c_high.asm"

;**********************************************************************************************************
; I2C Master : General Purpose Macros & Subroutines
; High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
;**********************************************************************************************************

;**********************************************************************************************************
;
;
I2C_TEST_DEVICE
;
MACRO
;
If Slave Device is listening, then _SlaveActive bit is set, else is cleared
;
Parameter : NONE
;
Sequence Of Operations :
S-SlvAW-A-P
;
If A is +ve device is listening, else either busy, not present or error condition
;
This test may also be used to check for example if a Serial EEPROM is in internal programming mode
;
NOTE : The address of the slave must be loaded into SlaveAddress Registers,
and 10 or 8 bit mode addressing must be set
;******************************************************************************
I2C_TEST_DEVICE MACRO

call IsSlaveActive ; TEMP ????: Assembler Error with this MACRO
ENDM
;

Test If A Device of SlaveAddr Is Present on Bus

The Slave Address Is put on the bus and if ACK it is present, if NACK not present or may be device is not responding. The presence can be checked constantly by a master (for ex. the Operating System on an Access Bus may constantly issue this command)

Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr

Set _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0

Returns 1 in _SlaveActive Bit if slave is responding else a 0

IsSlaveActive:

```assembly
0011 1091
bcf Slave_RW ; set for write operation
0012 2057
   call TxmtStartBit ; send START bit
0013 206B
   call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
;
0014 1311
bcf SlaveActive
0015 1F90
   btfss ACK_Error ; skip if NACK, device is not present or not responding
0016 1711
   bsf SlaveActive ; ACK received, device present & listening
0017 205F
   call TxmtStopBit
0018 0008
   return
```

I2C_WRITE

A basic macro for writing a block of data to a slave

Parameters:

- _BYTES_ #of bytes starting from RAM pointer _SourcePointer_
- _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
; Sequence:
; S-SlvAV-A-D[0]-A-A-D[D[N-1]]-A-P
;
; If an error occurs then the routine simply returns and user should check
; for flags in Bus_Status Reg (for eg. _Txmt_Success flag)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers,
; and 10 or 8 bit mode addressing must be set
;
I2C_WR MACRO _BYTES_, _SourcePointer_

movlw _BYTES_
movwf tempCount
movlw _SourcePointer_
movwf _fsr

call _i2c_block_write

call TxmtStopBit ; Issue a stop bit for slave to end transmission

ENDM

_i2c_block_write:

0019 2057 call TxmtStartBit ; send START bit
001A 1091 bcf Slave_RW ; set for write operation
001B 2068 call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set

_block_wr_loop:

001C 1E10 btfss Txmt_Success
001D 0008 return
001E 0800 movf indf, w
001F 008B movwf DataByte ; start from the first byte starting at DataPointer_
0020 0A84  incf  fsr
0021 2095  call  SendData  ; send next byte, bus is our's!
0022 0B96  decfsz  tempCount
0023 281C  goto  block_wrl_loop  ; loop until desired bytes of data transmitted to slave
0024 0008  return

;**********************************************************************************************************
;**********************************************************************************************************

I2C_WRITE_SUB
;
; Writes a message just like I2C_WRITE, except that the data is preceded by a sub-address to a slave device.
; Eg. : A serial EEPROM would need an address of memory location for Random Writes
; Parameters:
; BYTES  # of bytes starting from RAM pointer _SourcePointer_ (constant)
; SourcePointer_  Data Start Buffer pointer in RAM (file Registers)
; Sub_Address_  Sub-address of Slave (constant)
;
; Sequence:
;  S-SlvAW-A-SubA-A-D[0]-A-.....A-D[N-1]-A-P
;
; If an error occurs then the routine simply returns and user should check
; for flags in Bus_Status Reg (for eg. _Txmt_Success flag
;
; Returns:  WREG = 1 on success, else WREG = 0
;
; NOTE: The address of the slave must be loaded into SlaveAddress Registers,
; and 10 or 8 bit mode addressing must be set
;
; COMMENTS:
; I2C_WR may prove to be more efficient than this macro in most situations
; Advantages will be found for Random Address Block Writes for Slaves with
; Auto Increment Sub-Addresses (like Microchip's 24CXX series Serial EEPROMS)
I2C_WR_SUB MACRO _BYTES_, _SourcePointer_, _Sub_Address_

movlw (_BYTES_ + 1)
movwf tempCount

movlw (_SourcePointer_ - 1)
movwf far

movf indf, w
movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ -1)
movlw _Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ -1)
call i2c_block_write ; write _BYTES_+1 block of data

movf StoreTemp_1, w
movwf (_SourcePointer_ - 1) ; restore contents of (_SourcePointer_ - 1)
call TxmtStopBit ; Issue a stop bit for slave to end transmission

ENDM

;--------------------------------------------------------------------------------------------------------------------------
;
;
; Parameters :
;
; _BYTES_ #of bytes starting from RAM pointer _SourcePointer_ (constant)
; _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
; _Sub_Address_ Sub-address of Slave (constant)
;
; Sequence :
;
; S-S1vA=B-(SubA+0)-A-D(0)-A-P
S-SlvWA-A-(SubA+1)-A-D[1]-A-P

and so on until #of Bytes

If an error occurs then the routine simply returns and user should check
for flags in Bus_Status Reg (for eg. _Txmt_Success flag

Returns : WREG = 1 on success, else WREG = 0

COMMENTS : Very In-efficient, Bus is given up after every Byte Write

Some I2C devices addressed with a sub-address do not increment
automatically after an access of each byte. Thus a block of data
sent must have a sub-address followed by a data byte.

I2C_WR_SUB_SWINC MACRO _BYTES_, _SourcePointer_, _Sub_Address_

variable i ; TEMP ???? : Assembler Does Not Support This

i = 0

.whil (i < _BYTES_)

movf (_Source_Pointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call i2c_byte_wr_sub ; write a byte of data at sub address
i++

.endw

ENDM
;  Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
;

i2c_byte_wr_sub:

 0025 2057   call   TxmtStartBit           ; send START bit
 0026 1091   bcf    Slave_RW              ; set for write operation
 0027 206B   call   Txmt_Slave.Addr       ; if successful, then _Txmt_Success bit is set
 0028 1E10   btfss  Txmt_Success
 0029 2835   goto   block_wr1_fail        ; end
 002A 0814   movf   SubAddr,w
 002B 008E   movwf  DataByte
 002C 2095   call   SendData
 002D 1E10   btfss  Txmt_Success
 002E 2835   goto   block_wr1_fail        ; end
 002F 0815   movf   SrcPtr,w
 0030 008E   movwf  DataByte
 0031 2095   call   SendData
 0032 1E10   btfss  Txmt_Success
 0033 2835   goto   block_wr1_fail        ; failed, return 0 in WREG
 0034 2837   goto   block_wr1_pass         ; successful, return 1 in WREG

; return back to called routine from either block_wr1_pass or block_wr1_fail
block_wr1_fail:

 0035 205F   call   TxmtStopBit           ; Issue a stop bit for slave to end transmission
 0036 3400   retlw  FALSE
block_wr1_pass:

 0037 205F   call   TxmtStopBit           ; Issue a stop bit for slave to end transmission
 0038 3401   retlw  TRUE

;**********************************************************************************************
I2C_WR_MEM_BYTE

; Some I2C devices like a EEPROM need to wait some time after every byte write (when entered into
; internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC, but in addition adds a delay
; after each byte. Some EEPROM memories (like Microchip's 24Cxx Series have on-chip data buffer),
; and hence this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB for a
; block of data and then insert a delay until the whole buffer is written.

Parameters:
BYTES #of bytes starting from RAM pointer _SourcePointer_ (constant)
_SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_ Sub-address of Slave (constant)

Sequence:
S-SlvAW-A-(SubA+0)-A-D[0]-A-P
Delay 1 mSec ; The user can change this value to desired delay
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
Delay 1 mSec
and so on until #of Bytes

;****************************************************************************************************************************

I2C_WR_BYTE_MEM MACRO _BYTES_, _SourcePointer_, _Sub_Address_

variable i ; TEMP ????? : Assembler Does Not Support This

i = 0

.while (i < _BYTES_)
movf (_Source_Pointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call  i2c_byte_wr_sub        ; write a byte of data at sub address

i++

.endw

ENDM

;**************************************************************************;
;
;   I2C_WR_MEM_BUF
;
;
; This Macro/Function writes #of _BYTES_ to an I2C memory device. However some devices, esp. EEPROMs must
; wait while the device enters into programming mode. But some devices have an onchip temp data hold buffer
; and is used to store data before the device actually enters into programming mode. For example, the 24C04
; series of Serial EEPROMs from Microchip have an 8 byte data buffer. So one can send 8 bytes of data at a
; time and then the device enters programming mode. The master can either wait until a fixed time and then
; retry to program or can continouslly poll for ACK bit and then transmit the next Block of data for programming
;
; Parameters :
;   _BYTES_          # of bytes to write to memory
;   SourcePointer_   Pointer to the block of data
;   SubAddress_      Sub-address of the slave
;   Device_BUF_SIZE_ The on chip buffer size of the i2c slave
;
; Sequence of operations
;   I2C_SUB_WR operations are performed in loop and each time
;   data buffer of BUF_SIZE is output to the device. Then
;   the device is checked for busy and when not busy another
;   block of data is written
;
;**************************************************************************
I2C_WR_BUF_MEM MACRO _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_

    variable i, j

    if ( !( _BYTES_ )

        exitm

    endif

    elseif ( _BYTES_ <= _Device_BUF_SIZE_ )

        I2C_WR_SUB _BYTES_, _SourcePointer_, _SubAddress_

        exitm

    else

        i = 0
        j = ( _BYTES_ / _Device_BUF_SIZE_ )

        .while ( i < j )

            I2C_WR_SUB _Device_BUF_SIZE_, (_SourcePointer_ + i* _Device_BUF_SIZE_), (_SubAddress_ + i* _Device_BUF_SIZE_)

            call IsSlaveActive
            btfss _SlaveActive
            goto $-2

            i++

        .endw

        j = ( _BYTES_ - i* _Device_BUF_SIZE_ )

        if ( j )

            I2C_WR_SUB j, (_SourcePointer_ + i* _Device_BUF_SIZE_), (_SubAddress_ + i* _Device_BUF_SIZE_)

        endif

    endif

endm
The basic MACRO/procedure to read a block message from a slave device

Parameters:

- `BYTES_` : constant: #of bytes to receive
- `_DestPointer_` : destination pointer of RAM (File Registers)

Sequence:

- `S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P`

If last byte, then Master will NOT Acknowledge (send NACK)

NOTE: The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit mode addressing must be set

```
I2C_READ MACRO _BYTES_, _DestPointer_

movlw (_BYTES_ -1)
movwf tempCount  ; -1 because, the last byte is used out of loop
movlw DestPointer_
movwf fsr         ; FIFO destination address pointer

call i2c_block_read
ENDM
```
_i2c_block_read:

0039 2057  call  TxmtStartBit  ; send START bit
003A 1491  bsf  Slave_RW  ; set for read operation
003B 1111  bcf  Last_Byte_Rcv  ; not a last byte to rcv
003C 206B  call  Txmt_Slave_Addr  ; if successful, then _Txmt_Success bit is set
003D 1A10  btfsc  _Txmt_Success
003E 2841  goto  block_rdl_loop  ; end
003F 205F  call  TxmtStopBit  ; Issue a stop bit for slave to end transmission
0040 3400  retlw  FALSE  ; Error : may be device not responding

; block_rdl_loop:

0041 20CC  call  GetData
0042 080E  movf  DataByte,w
0043 0080  movwf  indf  ; start receiving data, starting at Destination Pointer

0044 0A84  incf  fsr
0045 0B96  decfsz  tempCount
0046 2841  goto  block_rdl_loop  ; loop until desired bytes of data transmitted to slave
0047 1511  bsf  Last_Byte_Rcv  ; last byte to rcv, so send NACK
0048 20CC  call  GetData
0049 080E  movf  DataByte,w
004A 0080  movwf  indf
004B 205F  call  TxmtStopBit  ; Issue a stop bit for slave to end transmission
004C 3401  retlw  TRUE

;******************************************************************************

I2C_READ_SUB

This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address Between the
sub-address write & the following reads, a STOP condition is not issued and a "REPEATED START" condition is
used so that an other master will not take over the bus, and also that no other master will overwrite the sub-
address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a
; memory device (e.g., 24Cxx serial of Serial EEPROMs from Microchip).
;
Parameters:
;    _BYTES_        # of bytes to read
;    _DestPointer_  The destination pointer of data to be received.
;    _SubAddress_   The sub-address of the slave
;
Sequence:
;    S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
******************************************************************************************

I2C_READ_SUB  MACRO  _BYTES_, _DestPointer_, _SubAddress_

bcf      Slave_RW      ; set for write operation
call    TxmtStartBit   ; send START bit
call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set

movlw    SubAddress_;
movwf    DataByte      ; START address of EEPROM(slave 1)
call    SendData       ; write sub address
     ; do not send STOP after this, use REPEATED START condition

I2C_READ  _BYTES_, _DestPointer_

ENDM

;******************************************************************************
;
I2C_READ_STATUS

; This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can send a status byte
; upon reception of a control byte. This is basically same as I2C_READ MACRO for reading a single byte.
;
; For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory data at the
; current address location
; On success WREG = 1 else = 0
;
;**********************************************************************************************************

;******************************************************************************

I2C_READ_STATUS MACRO _DestPointer_

;******************************************************************************

call TxmtStartBit ; send START bit
bsf Slave_RW ; set for read operation
call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
btfsc Txmt_Success
goto byte_rdl_loop ; read a byte
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw FALSE ; Error : may be device not responding
_byte_rdl_loop:

bsf Last_Byte_Rcv ; last byte to rcv, so send NACK
call GetData
movf DataByte,w
movwf DestPointer_
call TxmtStopBit ; Issue a stop bit for slave to end transmission
btfss Rcv_Success
retlw FALSE
retlw TRUE
ENDM
I2C_READ_BYTE MACRO _DestPointer_

I2C_READ_STATUS MACRO _DestPointer_

ENDM

I2C_WR_SUB WR

This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This may be useful for devices which need 2 blocks of data in which the first block may be an extended address of a slave device. For example, a large I2C memory device, or a teletext device with an extended addressing scheme, may need multiple bytes of data in the 1st block that represents the actual physical address and is followed by a 2nd block that actually represents the data.

Parameters:

BYTES1 1st block #of bytes
SourcePointer1 Start Pointer of the 1st block
SubAddress Sub-Address of slave
BYTES2 2st block #of bytes
SourcePointer2 Start Pointer of the 2nd block

Sequence:
S-Svw-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-....-A-D2[N-1]-A-P

Note: This MACRO is basically same as calling I2C_WR_SUB twice, but a STOP bit is not sent (bus is not given up) in between the two I2C_WR_SUB
Check Txmt_Success flag for any transmission errors

******************************************************************************

I2C_WR_SUB_WR MACRO _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_

movlw  (_COUNT1_ + 1)
movwf  tempCount
movlw  (_SourcePointer1_ - 1)
movwf  fsr
movf  indf,w
movwf  StoreTemp_1    ; temporarily store contents of (_SourcePointer_ -1)
movlw  _Sub_Address_
movwf  _indf
    ; store temporarily the sub-address at (_SourcePointer_1)
call  i2c_block_write  ; write _BYTES_+1 block of data
    ;
movf  StoreTemp_1,w
movwf  (_SourcePointer1_ - 1)    ; restore contents of (_SourcePointer_ -1)
    ; Block 1 write over
    ; Send Block 2
movlw  COUNT2_
movwf  tempCount
movlw  _SourcePointer2_
movwf  fsr
call  block_wr1_loop
    ;
call  TxmtStopBit    ; Issue a stop bit for slave to end transmission
ENDM

******************************************************************************
; I2C_WR_SUB_RD
;
; This macro writes a block of data from SourcePointer of length _COUNT1_ to a
; slave at sub-address and then Reads a block of Data of length _COUNT2_ to
; destination address pointer
;
; Message Structure :
;
; Parameters :
; _COUNT1_ Length Of Source Buffer
; _SourcePointer_ Source Pointer Address
; _Sub_Address_ The Sub Address Of the slave
; _COUNT2_ The length of Destination Buffer
; _DestPointer_ The start address of Destination Pointer
;
;******************************************************************************

I2C_WR_SUB_RD MACRO _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_

 movlw (COUNT1_ + 1)
 movwf tempCount
 movlw (SourcePointer_ - 1)
 movwf fsr
 movf indf, w
 movwf StoreTemp_l ; temporarily store contents of (_SourcePointer_ -1)
 movlw Sub_Address_
 movwf indf ; store temporarily the sub-address at (_SourcePointer_ -1)
 call i2c_block_write ; write _BYTES+1 block of data

 movf StoreTemp_l, w
 movwf (SourcePointer1_- 1) ; restore contents of (_SourcePointer_ - 1)
; Without sending a STOP bit, read a block of data by using a REPEATED
; Start Condition
;
I2C_READ  _COUNT2_, _DestPointer_
ENDM

;******************************************************************************;
;
I2C_WR_COM_WR
;
; This Macro write 2 blocks of data buffers to a slave in one message. This
; way no need to give up the bus after sending the first block.
; For example, this kind of transaction is used in an LCD driver where a
; block of control & address info is needed and then another block of actual
; data to be displayed is needed.
;
; Message Structure :
; S-SlvW-A-D1[0]-A-A-D1[1]-A-D2[0]-A-A-D2[1]-A-P
; NOTE : This message is same as calling two I2C_WR Macros, except that
; the bus is not given up between the sending of 2 blocks (this is
; done by not sending a STOP bit inbetween)
;
; Parameters :
; _COUNT1_  Length Of Source Buffer #1
; _SourcePointer1_  Source Pointer Address of 1st buffer
; _COUNT2_  The length of Destination Buffer
; _SourcePointer2_  Source Pointer Address of 2nd Buffer
;
;******************************************************************************;
I2C_WR_COM_WR MACRO _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_

.movlw COUNT1_
.movwf tempCount
.movlw _SourcePointer1_
.movwf fsr
.call i2c_block_write
;
; First block sent, now send 2nd block of data
;
.movlw _COUNT2_
.movwf tempCount
.movlw _SourcePointer2_
.movwf fsr
.call block_wr1_loop
;
call TxmtStopBit ; End of Double buffer txmt

ENDM

;***************************************************************
; INCLUDE I2C Low Level Routines Here
;***************************************************************

.include "i2c_low.asm"

;***************************************************************
; Low Level I2C Routines
;***************************************************************

; Single Master Transmitter & Single Master Receiver Routines
; These routines can very easily be converted to Multi-Master System
; when PIC16CEX with on chip I2C Slave Hardware, Start & Stop Bit
; detection is available.
;
; The generic high level routines are given in I2C_HIGH.ASM
;
;********************************************************************************************************
;
;********************************************************************************************************
;
; I2C Bus Initialization
;
;********************************************************************************************************

InitI2CBus_Master:

004C 1283 bcf _ rp0
004E 0806 movf _ portb,w
004F 39FC andlw 0xFC ; do not use BSF & BCF on Port Pins
0050 0086 movwf portb ; set SDA & SCL to zero. From Now on, simply play with tris
          ; RELEASE BUS

0051 1683
0051 1486
0052 1406
0054 0190 clrf Bus_Status ; reset status reg
0055 0191 clrf Bus_Control ; clear the Bus_Control Reg, reset to 8 bit addressing
0056 0008 return

;********************************************************************************************************
;
; Send Start Bit
;
;********************************************************************************************************

TxmtStartBit:

0057 1683 bsf rp0 ; select page 1
0058 1486 bsf SDA ; set SDA high
bsf SCL ; clock is high
; Setup time for a REPEATED START condition (4.7 uS)

005A 210D call Delay40uSec ; only necessary for setup time

005B 1086 bcf SDA ; give a falling edge on SDA while clock is high

005C 210B call Delay47uSec ; only necessary for START HOLD time

005D 1410 bsf Bus_Busy ; on a start condition bus is busy

005E 0008

return

;******************************************************************************1=******************
; Send Stop Bit
;
;******************************************************************************1=******************

TxmtStopBit:

bsf rp0 ; select page 1

0060 1006 bcf SCL

0061 1086 bcf SDA ; set SDA low

0062 1406 bsf SCL ; Clock is pulled up

0063 210D call Delay40uSec ; Setup Time For STOP Condition

0064 1486 bsf SDA ; give a rising edge on SDA while CLOCK is high

if _ENABLE_BUS_FREE_TIME
; delay to make sure a START bit is not sent immediately after a STOP,
; ensure BUS Free Time tBUF
;

0065 210B call Delay47uSec
endif
;
0066 1010  bcf  Bus_Busy  ; on a stop condition bus is considered Free
   
0067  0008  return

;*******************************************************************************
;    Abort Transmission
;
;    Send STOP Bit & set Abort Flag
;*******************************************************************************

AbortTransmission:

0068  205F  call  TxmtStopBit
0069  1490  bsf  Abort
006A  0008  return

;*******************************************************************************
;    Transmit Address (1st Byte)& Put in Read/Write Operation
;
;    Transmits Slave Addr On the 1st byte and set LSB to R/W operation
;    Slave Address must be loaded into SlaveAddr reg
;    The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for
;        Write & 1 for Read
;    On Success, return TRUE in WREG, else FALSE in WREG
;
;    If desired, the failure may tested by the bits in Bus Status Reg
;*******************************************************************************

Txmt_Slave.Addr:
006B 1390 bcf ACK_Error ; reset Acknowledge error bit
006C 1C11 btfs 10BitAddr
006D 2886 goto SevenBitAddr

006E 1C91 btfs Slave_RW
006F 2870 goto TenBitAddrWR ; For 10 Bit WR simply send 10 bit addr

; Required to READ a 10 bit slave, so first send 10 Bit for WR & then
; Repeated Start and then Hi Byte Only for read operation

TenBitAddrRd:

0070 1091 bcf Slave_RW ; temporarily set for WR operation
0071 2070 call TenBitAddrWR
0072 1E10 btfs Txmt_Success ; skip if successful
0073 3400 retlw FALSE
0074 2057 call TxmtStartBit ; send a REPEATED START condition
0075 1491 bsf Slave_RW ; For 10 bit slave Read

0076 080D movf SlaveAddr+1,W
0077 008E movwf DataByte
0078 140E bsf DataByte,LSB ; Read Operation
0079 2095 call SendData ; send ONLY high byte of 10 bit addr slave
007A 288C goto AddrSendTest ; 10 Bit Addr Send For Slave Read Over

; if successfully transmitted, expect an ACK bit

007B 1E10 btfs Txmt_Success ; if not successful, generate STOP & abort transfer
007C 2890 goto _ AddrSendFail

TenBitAddrWR:
007D 080D  movf  SlaveAddr+1,W
007E 008E  movwf  DataByte
007F 100E  bcf  DataByte,LSB ; WR Operation
               ; Ready to transmit data: If Interrupt Driven (i.e. if Clock Stretched LOW
               ; Enabled) then save RETURN Address Pointer

0080 2095  call  SendData ; send high byte of 10 bit addr slave
               ; if successfully transmitted, expect an ACK bit

0081 1E10  btfss  Txmt_Success ; if not successful, generate STOP & abort transfer
0082 2890  goto  AddrSendFail

0083 080C  movf  SlaveAddr,W
0084 008E  movwf  DataByte ; load addr to DataByte for transmission
0085 288B  goto  EndTxmtAddr

SevenBitAddr:

0086 080C  movf  SlaveAddr,W
0087 008E  movwf  DataByte ; load addr to DataByte for transmission
0088 100E  bcf  DataByte,LSB
0089 1891  btfsc  Slave_RW ; if skip then write operation
008A 140E  bsf  DataByte,LSB ; Read Operation

EndTxmtAddr:

008B 2095  call  SendData ; send 8 bits of address, bus is our's
               ; if successfully transmitted, expect an ACK bit

  _AddrSendTest:

008C 1E10  btfss  Txmt_Success ; skip if successful
008D 2890  goto  AddrSendFail
008E 0064 clrwdt
008F 3401 retlw TRUE

_AddrSendFail:
0090 0064 clrwdt
0091 1F90 btfs s ACK_Error
0092 3400 retlw FALSE ; Addr Txmt Unsuccessful, so return 0

; Address Not Acknowledged, so send STOP bit

0093 205F call TxmtStopBit
0094 3400 retlw FALSE ; Addr Txmt Unsuccessful, so return 0

**************************************************************************************

Transmit A Byte Of Data

; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low,
; then, the stretch is detected and INT Interrupt on Rising edge is enabled
; and also RTCC timeout interrupt is enabled The clock stretching slows down
; the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock
; stretching, then this feature can be disabled during Assembly time by setting
; _CLOCK_STRETCH_ENABLED must be set to FALSE.

;**************************************************************************************

SendData:

; TxmtByte & Send Data are same, Can check errors here before calling TxmtByte
; For future compatibility, the user MUST call SendData & NOT TxmtByte

0095 2896 goto TxmtByte
TxmtByte:
0096 080E movf DataByte,w
0097 0093 movwf DataByteCopy ; make copy of DataByte
0098 1510 bsf Txmt_Progress ; set Bus status for txmt progress
0099 1210 bcf Txmt_Success ; reset status bit
009A 3008 movlw 0x08
009B 008F movwf BitCount
009C 1683 bsf rp0
if _CLOCK_STRETCH_CHECK

; set RTCC to INT CLK timeout for 1 mSec
; do not disturb user's selection of RPUB in OPTION Register

009D 0801 movf option,w
009E 39C3 andlw OPTION_INIT ; defined in I2C.H header file
009F 0081 movwf option
endif

TxmtNextBit:
00A0 0064 clrwdt ; clear WDT, set for 18 mSec
00A1 1006 bcf SCL
00A2 0D93 rlf DataByteCopy ; MSB first, Note DataByte Is Lost
00A3 1086 bcf SDA
00A4 1803 btfsc c
00A5 1486 bsf SDA
00A6 210B call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
00A7 1406 bsf SCL ; set clock high, check if clock is high, else
00A8 210D call Delay40uSec ; guarantee min HIGH TIME tHIGH
if _CLOCK_STRETCH_CHECK

; clear RTCC
00A9 1283 bcf rp0
00AA 0181 clrf rttcc
00AB 110B bcf rttif
00AC 168B bsf rtie ; enable RTCC Interrupt
00AD 1391  bcf   TIME_OUT_      ; reset timeout error flag
     Check_SCL_1:
    00AE 1B91  btfs  TIME_OUT_    ; if RTCC timeout or Error then Abort & return
    00AF 28FC  goto  Bus_Fatal_Error  ; Possible FATAL Error on Bus
    00B0 1283  bcf   rp0
    00B1 1C06  btfs  SCL       ; if clock not being stretched, it must be high
    00B2 28AE  goto  Check_SCL_1   ; loop until SCL high or RTCC timeout interrupt
    00B3 1288  bcf   rtie       ; Clock good, disable RTCC interrupts
    00B4 1683  bsf   rp0
         endif
    00B5 0B8F  decfsz BitCount
    00B6 28A0  goto  TxmtNextBit

; Check For Acknowledge
;
    00B7 1006  bcf   SCL        ; reset clock
    00B8 1486  bsf   SDA        ; Release SDA line for Slave to pull down
    00B9 210B  call Delay47uSec  ; guarantee min LOW TIME tLOW & Setup time
    00BA 1406  bsf   SCL        ; clock for slave to ACK
    00BB 210D  call Delay40uSec  ; guarantee min HIGH TIME tHIGH
    00BC 1283  bcf   rp0        ; select PAGE 0 to test PortB pin SDA
    00BD 1886  btfs  SDA        ; SDA should be pulled low by slave if OK
    00BE 28C5  goto  TxmtErrorAck

;    00BF 1683  bsf   rp0       ; reset clock
    00C0 1006  bcf   SCL        ; reset TXMT bit in Bus Status
    00C1 1110  bcf  Txmt_Progress ; transmission successful
    00C2 1610  bsf  Txmt_Success  ; ACK_OK
    00C3 1390  bcf   ACK_Error   ; ACK_OK
    00C4 0008  return

                 TxmtErrorAck:
RELEASE_BUS
00C5 1683
00C5 1486
00C6 1406
00C8 1110  bcf  Txmt_Progress  ; reset TXMT bit in Bus Status
00C9 1210  bcf  Txmt_Success  ; transmission NOT successful
00CA 1790  bsf  ACK_Error    ; No ACK From Slave
00CB 0008  return
;
/*************************************************************************************************************/

Receive A Byte Of Data From Slave
;
; assume address is already sent
; if last byte to be received, do not acknowledge slave (last byte is
; tested from Last_Byte_Rcv bit of control reg)
; Data Received on successful reception is in DataReg register
;
/*****************************************************************************/

GetData:
00CC 28CD  goto  RcvByte
;
RcvByte:
00CD 1590  bsf  Rcv_Progress  ; set Bus status for txmt progress
00CE 1290  bcf  Rcv_Success  ; reset status bit
00CF 3008  movlw 0x08
00D0 008F  movwf  BitCount
00D1 1683  if  _CLOCK_STRETCH_CHECK
00D2 0008  movlw 0x08
00D3 008F  movwf  BitCount
00D4 1683  bsf  rp0
; set RTCC to INT CLK timeout for 1 mSec
; do not disturb user's selection of RPB in OPTION Register

```
00D2 0801  movf option, w
00D3 39C3  andlw OPTION_INIT ; defined in I2C.H header file
00D4 0081  movwf option

endif

RcvNextBit:
00D5 0064  clrwdt ; clear WDT, set for 18 mSec
00D6 1603  bsf rpo ; page l for TRIS manipulation
00D7 1006  bcf SCL
00D8 1486  bsf SDA ; can be removed from loop
00D9 210B  call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
00DA 1406  bsf SCL ; clock high, data sent by slave
00DB 210D  call Delay40uSec ; guarantee min HIGH TIME tHIGH

if _CLOCK_STRETCH_CHECK

00DC 1283  bcf rpo
00DD 0181  clrf rtcc ; clear RTCC
00DE 110B  bcf rtif ; clear any pending flags
00DF 168B  bsf rtie ; enable RTCC Interrupt
00E0 1391  bcf TIME_OUT_ ; reset timeout error flag

Check_SCL_2:
00E1 1B91  btfsc TIME_OUT_ ; if RTCC timeout or Error then Abort & return
00E2 28FC  goto Bus_Fatal_Error ; Possible FATAL Error on Bus
00E3 1283  bcf rpo
00E4 1C06  btfss SCL ; if clock not being stretched, it must be high
00E5 28E1  goto Check_SCL_2 ; loop until SCL high or RTCC timeout interrupt
00E6 128B  bcf rtie ; Clock good, disable RTCC interrupts
00E7 1683  bsf rpo

endif

00E8 1283  bcf rpo ; select page 0 to read Ports
00E9 1003  bcf c
```
00EA 1886    btfsb  SDA
00EB 1403    bsf  c                                  ; TEMP ????. DO 2 out of 3 Majority detect
          ;
00EC 008E    rlf  DataByte                             ; left shift data ( MSB first)
00ED 0B8F    decfsz  BitCount
00EE 28D5    goto  RcvNextBit
          ;
          ; Generate ACK bit if not last byte to be read,
          ; if last byte Generate NACK
          ; do not send ACK on last byte, main routine will send a STOP bit

00EF 1683    bsf  rp0
00F0 1006    bcf  SCL
00F1 1086    bcf  SDA                                  ; ACK by pulling SDA low
00F2 1911    btfsb  Last_Byte_Rcv
00F3 1486    bsf  SDA                                  ; if last byte, send NACK by setting SDA high
00F4 210B    call  Delay47uSec                       ; guarantee min LOW TIME tLOW & Setup time
00F5 1406    bsf  SCL
00F6 210D    call  Delay40uSec                       ; guarantee min HIGH TIME tHIGH
          
          ; RcvEnd:
00F7 1006    bcf  SCL                                 ; reset clock
00F8 1190    bcf  Rcv_Progress                       ; reset TXMT bit in Bus Status
00F9 1690    bsf  Rcv_Success                        ; transmission successful
00FA 1390    bcf  ACK_Error                          ; ACK OK
00FB 0008    return

if  _CLOCK_STRETCH_CHECK
;
;*********************************************************************************************************
;                                      ; Fatal Error On I2C Bus
; ; Slave pulling clock for too long or if SCL Line is stuck low.
; This occurs if during Transmission, SCL is stuck low for period longer
; than approx. 1mS and RTCC times out (approx 4096 cycles : 256 * 16 -
; prescaler of 16).
Bus_Fatal_Error:

; disable RTCC Interrupt

00FC 128B bcf rtie ; disable RTCC interrupts, until next TXMT try

RELEASE_BUS

00FD 1683
00FD 1486
00FE 1406

; Set the Bus_Status Bits appropriately

0100 1490 bsf Abort ; transmission was aborted
0101 1710 bsf Fatal_Error ; FATAL_Error occurred
0102 1110 bcf Txmt_Progress ; Transmission Is Not in Progress
0103 1210 bcf Txmt_Success ; Transmission Unsuccesful

0104 205F call TxmtStopBit ; Try sending a STOP bit, may be not successful

0105 0008 return

;*******************************************************************************
;****************************************************************************

General Purpose Delay Routines

; Delay4uS is wait loop for 4.0 uSec
; Delay47uS is wait loop for 4.7 uSec
; Delay50uS is wait loop for 5.0 uSec
;
;******************************************************************************

;******************************************************************************

Delay50uSec:
0106 3006
movlw (50uS_Delay-5)/3 + 1
DlyK
0107 0092
movwf DelayCount
0108 0B92
decfsz DelayCount
0109 2908
goto $-1
010A 0008
return
;

Delay47uSec:
010B 3004
movlw (47uS_Delay-8)/3 + 1
010C 2907
goto DlyK
;

Delay40uSec:
010D 3003
movlw (40uS_Delay-8)/3 + 1
010E 2907
goto DlyK
;

******************************************************************************

 endif
******************************************************************************

ReadSlavel:

;
;
EEPROM (24C04) may be in write mode (busy), check for ACK by sending a
;
control byte
LOAD_ADDR_8 _Slave_1_Addr

010F 1011
010F 30A0
wait1:
    I2C_TESTDEVICE

0110 008C

    ;
    ; SlaveActive ; See If slave is responding
    ; if stuck for ever, recover from WDT, can use other schemes
    ;
    I2C_READ_SUB 8, DataBegin+1, 0x50

0115 0064 clrwdt

0112 2011
0113 1F11 btfss SlaveActive ; See If slave is responding
0114 2912 goto wait1 ; if stuck for ever, recover from WDT, can use other schemes

0116 1091
0116 2057
0117 206B
0119 3050
0119 008E
011A 2095
011C 3007
011C 0096
011D 3021
011E 0084
0120 2039

; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60

0121 1011
0121 30AC
0122 008C
wait2:
I2C_TEST_DEVICE

0124 2011
0125 1F11 btfss SlaveActive ; See If slave is responding
0126 2924 goto wait2 ; if stuck for ever, recover from WDT, can use other schemes
0127 0064 clrwdt

I2C_READ_SUB 8, DataBegin+1, 0x60

0128 1091
0128 2057
0129 206B
012B 3060
012B 008E
012C 2095
012E 3007
012E 0096
012F 3021
0130 0084
0132 2039
0133 0008
0134 0008
0134 1011
0134 30D6
0135 008C

ReadSlave3:
LOAD_ADDR_8 _Slave_3_Addr
0137 2011
0138 1F11 btfss SlaveActive ; See If slave is responding
0139 2937 goto wait3 ; if stuck for ever, recover from WDT, can use other schemes
013A 0064 clrwdt
   I2C_READ_SUB 8, DataBegin, 0

013B 1091
013B 2057
013C 206B
013E 3000
013E 008E
013F 2095
0141 3007
0141 0096
0142 3020
0143 0084
0145 2039

; return

0146 0008

;*******************************************************************************
; ;
; Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
; ;*******************************************************************************

FillDataBuf:
0147 3000  movlw 0x00   ; start address location of EEPROM array
0148 00A0  movwf DataBegin ; 1st byte of data to be sent is start address
0149 3021  movlw DataBegin+1 ; data starts following address (RAM Pointer)
014A 0084  movwf fsr
014B 3008  movlw 8        ; fill RAM with 8 bytes, this data is written to
014C 009A  movwf byteCount
014D 3055  movlw 0x55     ; pattern to fill with is 0x55 & 0xAA
014E 009B  movwf HoldData
014F 099B  comf HoldData
0150 081B  movf HoldData,w
0151 0080  movwf indf
0152 0A84  incf fsr       ; point to next location
0153 0B9A  decfsz byteCount
0154 294F  goto X1
0155 0008  return

;****************************************************************************
; Main Routine (Test Program)
; SINGLE MASTER, MULTIPLE SLAVES
;****************************************************************************

Start:
0156 204D  call InitI2CBus_Master ; initialize I2C Bus
0157 178B  bsf gie               ; enable global interrupts

0158 2147  call FillDataBuf      ; fill data buffer with 8 bytes of data (0x55, 0xAA)
; Use high level Macro to send 9 bytes to Slave (1 & 2: 24C04) of 8 bit
; Addr
; Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin

0159 1810 btfsc Bus_Busy ; is Bus Free, ie. has a start & stop bit been
; detected (only for multi master system)
015A 2959 goto $-1 ; a very simple test, unused for now

LOAD_ADDR_8 _Slave_1.Addr

015B 1011
015C 30A0
015D 008C

I2C_WR 0x09, DataBegin

015E 3009
015F 0096
015F 3020
0160 0084
0162 2019
0162 205F

; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30

0164 1810 btfsc Bus_Busy ; is Bus Free, ie. has a start & stop bit been
; detected (only for multi master system)
0165 2964 goto $-1 ; a very simple test, unused for now
LOAD_ADDR_8  Slave 2 Addr

0166  1011
0166  30AC
0167  008C

I2C_WR_SUB  0x08, DataBegin+1, 0x30

0169  3009
0169  0096
016B  3020
016B  0084
016D  0800
016D  0097
016E  3030
016F  0080
0171  2019
0172  0817
0172  00A0
0174  205F
0175  210F  call  ReadSlave ; read a byte from slave from current address

LOAD_ADDR_8  Slave 3 Addr

0176  1011
0176  30D6
0177  008C
0179  30CC  movlw  0xCC
017A  00A0  movwf  DataBegin
I2C_WR_SUB  0x01,DataBegin,  0x33

017B  3002
017B 0096
017D 301F
017D 0084
017F 0800
017F 0097
0180 3033
0181 0080
0183 2019
0184 0817
0184 009F
0186 205F
0187 2134    call ReadSlave3       ; Read From Slave PIC
              
0188 0064    self clrwdt
0189 2988    goto self
              ******************************************

END
INTRODUCTION

The PIC16CXX series from Microchip Technology, Inc., are mid-range, high performance EPROM based 8-bit microcontrollers. Some of the members of this series (like PIC16C71 and PIC16C84) do not have on-chip hardware asynchronous serial port. This application note describes the Interrupt driven Software implementation of Asynchronous Serial I/O (Half Duplex RS-232 Communications) using PIC16CXX microcontrollers. These microcontrollers can operate at very high speeds with a minimum of 250 ns cycle time (with input clock frequency of 16 MHz). To test the RS-232 routines, a simple Digital Volt Meter (DVM)/Analog Data Acquisition Systems has been implemented using PIC16C71 in which upon reception of a command from host (IBM™ PC), an 8-bit value of the selected A/D channel is transmitted back to host.

IMPLEMENTATION

A half duplex Interrupt driven software implementation of RS-232 communications using PIC16C71 is described in detail below. The transmit pin used in the example code is RB7 and receive pin is connected to RTCC/RA4 pin (see Figure 2). Of course these pins are connected with appropriate voltage translation to/from RS-232/CMOS levels. The voltage translation is given described with schematics in the hardware section of this application note.

Transmit Mode

The transmit mode in software is quite straightforward to implement using interrupts. Once the input clock frequency and baud rate is known, the number of clock cycles per bit can be computed. The on-chip Real Time Clock Counter (RTCC) along with the prescaler can be used to generate interrupt on RTCC overflow. This RTCC overflow interrupt can be used as timing to send each bit. The input clock frequency ("_Clkln") and the Baud Rate ("_BaudRate") are programmable by the user and the RTCC time-out value (the period for each bit) is computed at assembly time. Whether the prescaler must be assigned to RTCC or not is also determined at assembly time. This computation is done in the header file "rs232.h". Note that very high speed transmissions can be obtained if transmission is done with pulserly software delays instead of interrupt driven. However the processor will be totally dedicated to this job.

Transmission of a byte is performed by calling "PutChar" function and the data byte in the "TxReg" is transmitted out. Before calling this function ("PutChar"), the data must be loaded into TxReg and also made sure that serial port is free. The serial port is free when both _txmtProgress and _rcvOver bits are cleared (see description of these bits in the Serial Status/Control Reg table given later).

Summary of "PutChar" function:
1) Make sure _txmtProgress & _rcvOver bits are cleared
2) Load TxReg with data to be transmitted
3) CALL PutChar function

Receive Mode

The reception mode implementation is slightly different from the transmit mode. Unlike the transmit Pin (TX pin in the example code is RB7, but could be any I/O pin), the receive pin (RX Pin) must be connected to RTCC/RA4 Pin. This is because in reception, the Start Bit which is asynchronous in nature, must be detected. To detect the start bit, when put in Reception mode, the RTCC module is configured to counter mode. The OPTION register is configured so that RTCC module is put in counter mode (increment on external clock on RTCC/RA4 Pin) and set to increment on falling edge on RTCC/RA4 pin with no prescaler assigned. After this configuration setup, RTCC (File Reg 1) is loaded with 0xFF. A falling edge on RTCC Pin will make RTCC roll over from 0xFF to 0x00, thus generating an interrupt indicating a Start Bit. The RTCC/RA4 pin is sampled again to make sure the transition on RTCC is not a glitch. Once the start bit has been detected, the RTCC module is reconfigured to increment on internal clock and the prescaler is assigned to it depending on input master clock frequency and the baud rate (configured same way as the transmission mode).

The software serial port is put in reception mode when a call is made to function "GetChar". Before calling this function make sure serial port is free (i.e., _txmtProgress and _rcvOver status bits must be 0). On completion of reception of a byte, the data is stored in RxReg and _rcvOver bit is set to 0.

Summary of "GetChar" function:
1) Make sure _txmtProgress & _rcvOver bits are cleared
2) CALL GetChar function
3) The received Byte is in TxReg after _rcvOver bit is cleared
Parity Generation

Parity can be enabled at assembly time by setting "_PARITY_ENABLE" flag to TRUE. If enabled, the parity can be set to either EVEN or ODD parity. In transmission mode, if parity is enabled, the parity bit is computed and transmitted as the 9th bit. On reception, the parity is computed on the received byte and compared to the 9th bit received. If a match does not occur the parity error bit is set in the RS-232 Status/Control Register (_ParityErr bit of SerialStatus reg). The parity bit is computed using the algorithm shown in Figure 1. This algorithm is highly efficient using PIC16CXX's SWAPF and XORWF instructions (with ability to have the destination as either file register itself or W register) and the sub-routine (called "GenParity") is in file "txmtr.asm".

FIGURE 1 - AN EFFICIENT PARITY GENERATION SCHEME IN SOFTWARE

Assembly Time Options

The firmware is written as a general purpose routines and the user must specify the following parameters before assembling the program. The Status/Control register is also described below:

TABLE 1 - LIST OF ASSEMBLY TIME OPTIONS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ClkIn</td>
<td>Input clock frequency of the processor.</td>
</tr>
<tr>
<td>_BaudRate</td>
<td>Desired Baud Rate. Any valid value can be used. The highest Baud Rate achievable depends on Input Clock Freq. 600 to 4800 Baud was tested using 4 MHz Input Clock. 600 to 19200 Baud was tested using 10 MHz Input Clock. Higher rates can be obtained using higher Input Clock Frequencies. Once the _BaudRate &amp; _ClkIn are specified, the program automatically selects all the appropriate timings.</td>
</tr>
<tr>
<td>_DataBits</td>
<td>Can specify 1 to 8 data bits.</td>
</tr>
<tr>
<td>_StopBits</td>
<td>Limited to 1 Stop Bit. Must be set to 1.</td>
</tr>
<tr>
<td>_PARITY_ENABLE</td>
<td>Parity Enable Flag. Set it to TRUE or FALSE. If PARITY is used, then set it to TRUE, else FALSE. See &quot;_ODD_PARITY&quot; flag description below.</td>
</tr>
<tr>
<td>_ODD_PARITY</td>
<td>Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else mEVEN Parity Scheme is used. This Flag is ignored if _PARITY_ENABLE is set to FALSE.</td>
</tr>
<tr>
<td>_USE_RTSCTS</td>
<td>RTS &amp; CTS Hardware handshaking signals. If set to FALSE, no hardware handshaking is used. If set to TRUE, RTS &amp; CTS use up 2 I/O Pins of PortB.</td>
</tr>
</tbody>
</table>
### TABLE 2 - BIT ASSIGNMENTS OF SERIAL STATUS/CONTROL REGISTER ("SERIALSTATUS" REG)

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>_txmtProgress</td>
<td>1 = Transmission in progress. 0 = Transmission line free.</td>
</tr>
<tr>
<td>1</td>
<td>_txmtEnable</td>
<td>Set this bit to 1 on initialization to enable transmission. This bit may be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>used to abort a transmission. The transmission is aborted if in the middle of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a transmission (i.e. when _txmtProgress bit is 1) _txmtEnable bit is set to 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit gets automatically set when PutChar function is called.</td>
</tr>
<tr>
<td>2</td>
<td>_rcvProgress</td>
<td>1 = Middle of a byte reception. 0 = Reception of a byte (in RxReg) is complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and is set to 1 when a valid start bit is detected in reception mode.</td>
</tr>
<tr>
<td>3</td>
<td>_rcvOver</td>
<td>1 = Completion of reception of a byte. The user's code can poll this bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>after calling &quot;GetChar&quot; function and check to see if it is set. When set,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the received byte is in RxReg. Other status bits should also be checked for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>any reception errors.</td>
</tr>
<tr>
<td>4</td>
<td>_ParityErr</td>
<td>1 = Parity error on reception (irrespective of Even Or Odd parity chosen).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not applicable if No Parity is used.</td>
</tr>
<tr>
<td>5</td>
<td>_FrameErr</td>
<td>1 = Framing error on reception.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>7</td>
<td>_parityBit</td>
<td>The 9th bit of transmission or reception. In transmission mode, the parity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit of the byte to be transmitted is set in this bit. In receive mode, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9th bit (or parity bit) received is stored in this bit. Not Applicable if no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parity is used.</td>
</tr>
</tbody>
</table>
**Hardware**

The hardware is primarily concerned with voltage translation from RS-232 to CMOS levels and vice versa. Three circuits are given below and the user may choose which ever best suits his application. The primary difference between each solution is cost versus number of components. Circuits in Figure 3 and 4 are very low cost but have more components than the circuit in Figure 2. The circuit in Figure 2 interfaces to RS-232 line using a single chip (MAX-232) and single +5V supply. The circuit in Figure 3 is a low cost RS-232 Interface but requires two chips and a single +5V supply source.

Figure 4 shows a very low cost RS-232 Interface to an IBM PC with no external power requirements. The circuit draws power from RS-232 line (DTR) and meets the spec of drawing power less than 5mA. This requires that the host to communicate must assert DTR high and RTS low. The power is drawn from DTR line and this requires that DTR to be asserted high and must be at least 7V. The negative -5 to -10 V required by LM339 is drawn from RTS line and thus the host must assert RTS low. This circuit is possible because of the low current consumption of PIC16C71 (typical 2 mA).

**FIGURE 2 - SINGLE CHIP FOR RS-232 INTERFACE (SINGLE +5V SUPPLY)**

![Circuit Diagram](image)

**FIGURE 3 - LOW COST RS-232 INTERFACE (TWO CHIPS, SINGLE +5V SUPPLY)**

![Circuit Diagram](image)
FIGURE 4 - LOW COST, LOW POWER RS-232 INTERFACE (POWER SUPPLIED BY RS-232 LINES)

Test Program

To test the transmission and reception modules, a main program is written in which the PIC16C71 waits to receive a command from a host through the RS-232. On reception of a byte (valid commands are Ox00, Ox01, Ox02 & Ox03), the received byte is treated as the PIC16C71's A/D channel number and the requested channel is selected, an A/D conversion is started and when the conversion is complete (in about 20 µS) the digital data (8 bits) are transmitted back to the host. A Microsoft Windows program running on an IBM PC/AT was written to act as a host and collect the A/D data from PIC16C71 via an RS-232 port. The Windows program (DVM.EXE) runs as a background job and displays the A/D data in a small window (similar to the CLOCK program that comes with MS Windows). The windows program and the PIC16C71 together act like a data acquisition system or a digital volt meter (DVM). The block diagram of the system is shown in Figure 2. The input clock frequency is fixed at 4 MHz and RS-232 parameters are set to 1200 Baud, 8-bits, 1 Stop Bit and No Parity. The program during development stage was also tested at 1200, 2400, 4800 Baud Rates @ 4 MHz Input Clock and up to 19200 Baud @ 10 MHz input clock frequency (all tests were performed with No Parity, Even Parity and Odd Parity at 8 and 7 Data Bits).
Source Code

The PIC16CXX source code along with the MS Windows DVM Program (executable running on an IBM PC/AT under MS Windows 3.1 or higher) is available on Microchip’s BBS. The assembly code for PIC16CXX must be assembled using Microchip’s Universal Assembler MPASM. The code cannot be assembled using the older assemblers without significant modifications. It is suggested that users who do not have the new assembler MPASM, must change to the new version.

The MS Windows Program (DVM.EXE) runs under MS Windows 3.1 or higher. The program does not have any menus and shows up as a small window displaying A/D Data and runs as a background job. There are a few command line options and are described below:

-Px : x is the comm port number (e.g. - P2 selects COM2). Default is COM1
-Cy : y is the number of A/D channels to display. Default is one channel (channel #1)
-Sz : z is a floating point number that represents the scaling factor (For example - S5.5 would display the data as 5.5*<8bit A/D>/256). The default value is 5.0 volts. -S0 will display the data in raw format without any scaling.
Appendix A - RS232.H

NOLIST

;*****************************************************************************************
; RS-232 Header File
; PIC16C6X/7X/8X
;*****************************************************************************************

ClkOut equ (ClkIn >> 2) ; Instruction Cycle Freq = CLKIN/4

CyclesPerBit set (ClkOut/_BaudRate)
tempCompute set (CyclesPerBit >> 8)

;*****************************************************************************************
; Auto Generation Of Prescaler & Rtcc Values
; Computed during Assembly Time
;*****************************************************************************************

; At first set Default values for RtccPrescale & RtccPreLoad

RtccPrescale set 0
RtccPreLoad set CyclesPerBit
UsePrescale set FALSE

if (_tempCompute >= 1)
RtccPrescale set 0
RtccPreLoad set (CyclesPerBit >> 1)

UsePrescale set TRUE
endif

if (tempCompute >= 2)
RtccPrescale set 1
RtccPreLoad set (CyclesPerBit >> 2)
endif

if (tempCompute >= 4)
RtccPrescale set 2
RtccPreLoad set (CyclesPerBit >> 3)
endif

if (tempCompute >= 8)
RtccPrescale set 3
RtccPreLoad set (CyclesPerBit >> 4)
endif
if (tempCompute >= 16)
    RtccPrescale set 4
    RtccPreLoad set (CyclesPerBit >> 5)
endif

if (tempCompute >= 32)
    RtccPrescale set 5
    RtccPreLoad set (CyclesPerBit >> 6)
endif

if (tempCompute >= 64)
    RtccPrescale set 6
    RtccPreLoad set (CyclesPerBit >> 7)
endif

if (tempCompute >= 128)
    RtccPrescale set 7
    RtccPreLoad set (CyclesPerBit >> 8)
endif

; if (RtccPrescale == 0) && (RtccPreLoad < 60))
    messg "Warning : Baud Rate May Be Too High For This Input Clock"
endif ;
; Compute RTCC & Prescaler Values For 1.5 Times the Baud Rate for Start Bit Detection ;
SBitCycles set (ClkOut/_BaudRate) + (_ClkOut/4)/_BaudRate
tempCompute set (SBitCycles >> 8)
SBitPrescale set 0
SBitRtccLoad set _SBitCycles

if (tempCompute >= 1)
    SBitPrescale set 0
    SBitRtccLoad set (SBitCycles >> 1)
endif

if (tempCompute >= 2)
    SBitPrescale set 1
    SBitRtccLoad set (SBitCycles >> 2)
endif
if (tempCompute >= 4)
SBitPrescale set 2
SBitRtccLoad set (SBitCycles >> 3)
endif

if (tempCompute >= 8)
SBitPrescale set 3
SBitRtccLoad set (SBitCycles >> 4)
endif

if (tempCompute >= 16)
SBitPrescale set 4
SBitRtccLoad set (SBitCycles >> 5)
endif

if (tempCompute >= 32)
SBitPrescale set 5
SBitRtccLoad set (SBitCycles >> 6)
endif

if (_tempCompute >= 64)
SBitPrescale set 6
SBitRtccLoad set (SBitCycles >> 7)
endif

if (tempCompute >= 128)
SBitPrescale set 7
SBitRtccLoad set (SBitCycles >> 8)
endif

; ;***************************************************************************************
; #define _Cycle_Offset1 18
; LOAD_RTCC MACRO K, Prescale
if (UsePrescale == 0)
movlw -K + _Cycle_Offset1
else
movlw -K + (_Cycle_Offset1 >> (Prescale+1)) ; Re Load RTCC init value + INT Latency Offset
endif
movwf _rtcc ; Note that Prescaler is cleared when RTCC is written
ENDM

LOAD_BITCOUNT MACRO
    movlw _DataBits+ StopBits
    movwf BitCount
    if _PARITY.Enable
        movlw 2
        movwf ExtraBitCount
    endif
    ENDM

;***********************************************************************************************
LOAD_BAUD MACRO
    ; Set baud rate
    ;_BaudRate is the Baud Rate to be set

;***********************************************************************************************
OPTION SEIT set
if UsePrescale
    OPTION INIT set 0x38
    if UsePrescale
        OPTION_INIT set 0x00
        Prescaler is used depending on Input Clock & Baud Rate
    else
        OPTION_INIT set 0x08
    endif
else
    _OPTION_INIT set 0x38
    ; Increment on Ext Clock (falling edge), for START Bit Detect
endif

; Pin Assignments
#define RX MASK
#define RX Pin
#define TX
#define RTS
#define CTS

#define _txmtProgress SerialStatus, 0
#define txmtEnable SerialStatus, 1
#define rcvProgress SerialStatus, 2
#define rcvOver SerialStatus, 3
#define ParityErr SerialStatus, 4
#define _FrameErr SerialStatus, 5
#define _parityBit SerialStatus, 7

;***********************************************************************************************
CBLOCK 0x0C
TxReg ; Transmit Data Holding/Shift Reg
RxReg ; Rcv Data Holding Reg
RxTemp
SerialStatus ; Txmt & Rev Status/Control Reg
BitCount
if _PARITY_ENABLE
ExtraBitCount ; Parity & Stop Bit Count
endif
SaveWReg ; temp hold reg of WREG on INT
SaveStatus ; temp hold reg of STATUS Reg on INT
temp1, temp2
ENDC

;***************************************************************************************************

LIST
Appendix B - RS232.ASM

TITLE "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
SUBTITLE "Software Implementation : Interrupt Driven"

;******************************************************************************
; Software Implementation of RS232 Communications Using PIC16CXX
; Half-Duplex
;
These routines are intended to be used with PIC16C6X/7X family. These routines can be
used with processors in the 16C6X/7X family which do not have on board Hardware Async
Serial Port.

Description:
Half Duplex RS-232 Mode Is implemented in Software.
Both Reception & Transmission are Interrupt driven
Only 1 peripheral (RTCC) used for both transmission & reception
RTCC is used for both timing generation (for bit transmission & bit polling)
and Start Bit Detection in reception mode.
This is explained in more detail under Interrupt Subroutine.
Programmable Baud Rate (speed depending on Input Clock Freq.), programmable
# of bits, Parity enable/disable, odd/even parity is implemented.
Parity & Framing errors are detected on Reception

RS-232 Parameters
The RS-232 Parameters must be defined as shown below:

ClkIn : Input Clock Frequency of the processor
(NOTE : RC Clock Mode Is Not Suggested due to wide variations)
BaudRate : Desired Baud Rate. Any valid value can be used.
The highest Baud Rate achievable depends on Input Clock Freq.
600 to 4800 Baud was tested using 4 MHz Input Clock
600 to 19200 Baud was tested using 10 MHz Input Clock
Higher rates can be obtained using higher Input Clock Frequencies.
Once the _BaudRate & _ClkIn are specified the program
automatically selects all the appropriate timings
DataBits : Can specify 1 to 8 Bits.
StopBits : Limited to 1 Stop Bit. Must set it to 1.
PARITY_ENABLE : Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
description below
ODD_PARITY : Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
EVEN Parity Scheme is used.
This Flag is ignored if _PARITY_ENABLE is set to FALSE.
Usage:

An example is given in the main program on how to Receive & Transmit Data.

In the example, the processor waits until a command is received. The command is interpreted as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.

The RS-232 Control/Status Reg’s bits are explained below:

"SerialStatus" : RS-232 Status/Control Register

- **Bit 0** : txmtProgress (1 if transmission in progress, 0 if transmission is complete)
  - After a byte is transmitted by calling “PutChar” function, the user’s code can poll this bit to check if transmission is complete.
  - This bit is reset after the STOP bit has been transmitted.

- **Bit 1** : txmtEnable
  - Set this bit to 1 on initialization to enable transmission.
  - This bit can be used to Abort a transmission while the transmitter is in progress (i.e when _txmtProgress = 1)

- **Bit 2** : rcvProgress
  - Indicates if the receiver is in middle of reception. It is reset when a byte is received.

- **Bit 3** : rcvOver
  - This bit indicates the completion of Reception of a Byte. The user’s code can poll this bit after calling “GetChar” function. Once “GetChar” function is called, this bit is 0 and is set to 1 after reception of a complete byte (parity bit if enabled & stop bit)

- **Bit 4** : ParityErr
  - A 1 indicates Parity Error on Reception (for both even & odd parity)

- **Bit 5** : FrameErr
  - A 1 indicates Framing Error On Reception

- **Bit 6** : unused
  - Unimplemented Bit

- **Bit 7** : parityBit
  - The 9th bit of transmission or reception (status of PARITY bit if parity is enabled)

To Transmit A Byte Of Data:
1) Make sure _txmtProgress & _rcvOver bits are cleared
2) Load TxReg with data to be transmitted
3) CALL PutChar function

To Receive A Byte Of Data:
1) Make sure _txmtProgress & _rcvOver bits are cleared
2) CALL GetChar function
3) The received Byte is in TxReg after _rcvOver bit is cleared

Processor: 16C71
Radix: DEC
EXPAND

include "d:\pictools\16Cxx.h"
;*******************************************************************************
; Setup RS-232 Parameters
;*******************************************************************************

_Clkn equ 1000000 ; Input Clock Frequency is 4 Mhz
_BaudRate set 1200 ; Baud Rate (bits per second) is 1200
_DataBits set 8 ; 8 bit data, can be 1 to 8
_StopBits set 1 ; 1 Stop Bit, 2 Stop Bit is not implemented

#define _PARITY_ENABLE FALSE ; NO Parity
#define _ODD_PARITY FALSE ; EVEN Parity, if Parity enabled
#define _USE_RTSCTS FALSE ; NO Hardware Handshaking is Used

#include "rs232.h"

;*******************************************************************************

ORG ResetVector
goto Start

ORG IntVector
goto Interrupt

;*******************************************************************************
; Table Of ADCONO Reg
; Inputs : WREG (valid values are 0 thru 3)
; Returns In WREG, ADCONO Value, selecting the desired Channel
;
; Program Memory : 6 locations
; Cycles : 5
;
;*******************************************************************************

GetADConO:
  ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
  andlw 0x03
  addwf pcl
  retlw (0xc1 | (0 << 3)) ; channel 0
  retlw (0xc1 | (1 << 3)) ; channel 1
  retlw (0xc1 | (2 << 3)) ; channel 2
GetADConO_End:
  retlw (0xc1 | (3 << 3)) ; channel 3

if( (GetADConO & 0xff) >= (GetADConO_End & 0xff) )
  MESSG "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
endif
; Initialize A/D Converter
; <RA0:RA3> Configure as Analog Inputs, VDD as Vref
; A/D Clock Is Internal RC Clock
; Select Channel 0
;
; Program Memory : 6 locations
; Cycles : 7
;
;******************************************************************************
; InitAtoD:
;  bsf rp0
;  clrf adcon1
;  bcf rp0
;  movlw 0xCl
;  movwf adcon0
;  return
;
;******************************************************************************

; Main Program Loop
;
; After appropriate initialization, The main program wait for a command from RS-232
; The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
; After a command is received, the appropriate A/D Channel is selected and when conversion is
; completed the A/D Data is transmitted back to the Host. The controller now waits for a new
; command.
;******************************************************************************

Start:
call InitSerialPort

call InitAtoD
;
WaitForNextSel:
if _ USE RTSCTS
  bcf rp0
  bcf RTS
endif

  call GetChar
  btfsc rcvOver goto rcvOver
  ;_rcvOver Gets Cleared When a Byte is Received (in RxReg)
  ; USER can perform other jobs here, can poll _rcvOver bit
;
; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
;
  bcf rp0
  movf RxReg, w
  ;_WREG = Commanded Channel = (0 thru 3)
  call GetADCon0
  ; Get ADCON0 Reg Constant from table Lookup
  movwf adcon0
  ; Load ADCON0 reg, selecting the desired channel
  nop
;
  bsf go
  ; start conversion
btfsc done
  goto $-1 ; Loop Until A/D Conversion Done
;
  movf adress, w
  movwf TxReg
  if _USE_RTSCTS
    bsf RTS ; Half duplex mode, transmission mode, ask host not to send data
    btfsc CTS ; Check CTS signal if host ready to accept data
    goto $-1
  endif
  call PutChar
  btfsc txmtProgress
  goto $-1 ; Loop Until Transmission Over, User Can Perform Other Jobs
;
  goto WaitForNextSel; wait for next selection (command from Serial Port)
;
;*********************************************************************************************************
RS-232 Routines
;*********************************************************************************************************

Interrupt Service Routine
;
; Only RTCC Interrupt Is used. RTCC Interrupt is used as timing for Serial Port Receive & Transmit
; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
; Transmit Modules.
;  Transmission :
;    RTCC is setup for Internal Clock increments and interrupt is generated when
;    RTCC overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
;    desired BAUD RATE.
;  Reception :
;    When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
;    and preloaded with 0xFF. When a Falling Edge is detected on RTCC Pin, RTCC is
;    rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
;    bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
;    with a certain value for regular timing interrupts to Poll RTCC Pin (i.e RX pin).
;
;*********************************************************************************************************

Interrupt:
  btfss rtif ; other interrupt, simply return & enable GIE
  retfie

  ; Save Status On INT : WREG & STATUS Regs
  movwf SaveWReg
  swapf status, w ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
  movwf SaveStatus
  btfsc txmtProgress
gto TxmNextBit ; Txmt Next Bit
tfsc rcvProgress
gto - RcvNextBit ; Receive Next Bit
goto SBitDetected ; Must be start Bit
;
RestoreIntStatus:
swapf SaveStatus,w
movwf status ; restore STATUS Reg
swapf SaveWReg
swapf SaveWReg,w ; restore WREG
bcf rtf
retfie

*****************************************************************************

Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
Configure, RX_Pin (RTCC pin) as Input, which is used to poll data on reception
;
Program Memory : 8 locations
Cycles : 9
*****************************************************************************

InitSerialPort:
clr f SerialStatus
;
bcf rp0 ; select Page 0 for Port Access
bsf TX ; make sure TX Pin is high on powerup, use RB Port Pullup
bsf rp0 ; Select Page 1 for TrisB access
bcf TX ; set TX Pin As Output Pin, by modifying TRIS
if _USE_RTSCTS
bcf RTS ; RTS is output signal, controlled by PIC16Cxx
bsf CTS ; CTS is input signal, controlled by the host
endif
bsf RX_Pin ; set RX Pin As Input for reception
return

*****************************************************************************

include "txmtr.asm" ; The Transmit routines are in file "txmtr.asm"
ineclude "rcvr.asm" ; The Receiver Routines are in File "rcvr.asm"
*****************************************************************************

END
Appendix C - RCVR.ASM

;*****************************************************************************************
; GetChar Function
; Receives a Byte Of Data
; When reception is complete, _rcvOver Bit is cleared
; The received data is in RxReg
;
; Program Memory : 15 locations (17 locations if PARITY is used)
; Cycles : 16 (18 if PARITY is USED)
;
;*****************************************************************************************
GetChar:
bsf _rcvOver
LOAD_BITCOUNT
        clr _RxReg
bcf _FrameErr
bcf _ParityErr
bsf _rp0
movlw _OPTION_SBIT
movwf _option
bcf _rp0
movlw 0xFF
movwf _rtcc
bsf _rtie
        ; Enable RTCC Interrupt
        ; Enable Global Interrupt
retfie

;*****************************************************************************************
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 14 locations
; Cycles : 12 (worst case)
;
;*****************************************************************************************
_SBitDetected:
bcf _rp0
btssc _RX_Pin
        ; Make sure Start Bit Interrupt is not a Glitch
goto FalseStartBit
bsf _rcvProgress
bsf _rp0
movlw (OPTION_INIT | SBitPrescale)
movwf _option
        ; Switch Back to INT Clock
        ; Set Option Reg Located In Page 1
        ; make sure to select Page 0
LOAD_RTCC (SBitRtccLoad), SBitPrescale
goto RestoreIntStatus

_FalseStartBit:
    movlw 0xFF
    movwf __rtcc
    ; reload RTCC with 0xFF for start bit detection
    goto RestoreIntStatus
;
*******************************************************************************************
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 28 locations ( 43 locations with PARITY enabled)
; Cycles : 24 Worst Case
;
*******************************************************************************************
_RcvNextBit:
    bsf __ rp0
    movlw (__OPTION_INIT | __RtccPrescale) ; Switch Back to INT Clock
    movwf __ option
    ; Set Option Reg Located In Page 1
    bcf __ rp0
    movf __ porta, w
    movwf __ RxTemp
    LOAD_RTCC __RtccPreLoad, __RtccPrescale ; Macro to reload RTCC
    movf __ porta, w
    xorwf __ RxTemp, w
    andlw __ RX_MASK ; mask for only RX PIN (RA4)
    btfsc __ z
    goto _SampleAgain:
    movf __ porta, w
    movwf __ RxTemp
    ; 2 out of 3 majority sampling done
    _PinSampled:
    if __PARITY_ENABLE
    movf __ BitCount
    btfsc __ z
    goto _RcvP_Or_S
    endif
    ;
    decfsz __ BitCount
    goto _NextRcvBit
    ;
    if __PARITY_ENABLE
    _RcvP_Or_S:
    decfsz __ ExtraBitCount
    goto _RcvParity
    endif
    ;
_RcvStopBit:
    btfss __ RX
    bsf __ FrameErr
    ; may be framing Error or Glitch
bcf rtie               ; disable further interrupts
bcf _ rcvProgress    ; Byte Received, Can RCV/TXMT an other Byte if _PARITY_ENABLE
movf _ rcvOver       ; Generate Parity, for Parity check
 call GenParity      
 movlw 0             ; to mask off Received Parity Bit in _ParityErr
 btfsc _ parityBit   ; _ParityErr bit is set accordingly
 movlw 0x10
 xorwf SerialStatus
 endif
 goto RestoreIntStatus

 ;_NextRcvBit:
 bcf _ carry
 btfsc RX            ; may be a glitch, check again
 bsf _ carry
 rrf RxReg           ; shift in received data
 goto RestoreIntStatus

 ; if _PARITY_ENABLE
 _RcvParity:
 bcf _ ParityErr     ; Temporarily store PARITY Bit in _ParityErr
 btfsc RX            ; Sample again to avoid any glitches
 bsf _ ParityErr     
 goto RestoreIntStatus
 endif

 ;*****************************************************************************************
Appendix D - TXMTR.ASM

;*********************************************************************************************************
; PutChar Function
; Function to transmit A Byte Of Data
; Before calling this routine, load the Byte to be transmitted into TxReg
; Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
; calling this routine
;
; Program Memory : 6 locations (10 locations if PARITY is Used)
; Cycles : 8 (13 if PARITY is Used)
;
;*********************************************************************************************************

PutChar:
  bsf      txmtEnable             ; enable transmission
  bsf      txmtProgress
  load     BITCOUNT               ; Macro to load bit count
  if _PARITY_ENABLE
  movf     TxReg,W
  call      GenParity              ; If Parity is used, then Generate Parity Bit
  endif
  call      TxmtStartBit
  bsf      rtie                    ; Enable RTCC Overflow INT
  retfie   ; return with _GIE Bit Set

;*********************************************************************************************************

Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 15 locations (25 locations if PARITY is Used)
; Cycles : 13 Worst Case
;
;*********************************************************************************************************

_TxmtNextBit:
  bcf _rp0
  load    RTCC                  ; Macro to reload RTCC
  if _PARITY_ENABLE
  movf    BitCount
  btfsc    z
  goto    ParityOrStop
  endif
  decfsz   BitCount

goto _ NextTxmtBit
;
if _ PARITY_ENABLE
  _ParityOrStop:
    decsz ExtraBitCount
    goto _ SendParity
  endif
;
  _StopBit:
    bsf TX ; STOP Bit is High
    bcf rtie ; disable further interrupts
    bcf txmtProgress
    goto RestoreIntStatus
;
  _NextTxmtBit:
    rrf TxReg
    btfss carry
    bcf TX
    btfsc carry
    bsf TX
    btfss txmtEnable
    bsf rtie ;disable further interrupts, Transmission Aborted
    goto RestoreIntStatus
;
  if _ PARITY_ENABLE
    _SendParity:
      btfss parityBit
      bcf TX
      btfsc parityBit
      bsf TX
      goto RestoreIntStatus
  endif

;******************************************************************************
; Internal Subroutine entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 9 locations
; Cycles : 10
;
;******************************************************************************

_TXmtStartBit:
  bsf rpl
  movlw (OPTION_INIT | RtcPrescale)
  movwf option ; Set Option Reg Located In Page 1
bcf _    rp0 ; make sure to select Page 0
bcf _    TX ; Send Start Bit
movlw _ -RtccPreLoad ; Prepare for Timing Interrupt
movwf _  rtcc
bcf _    rtif
return

;*****************************************************************************************************
; *** Generate Parity for the Value in WREG
; *** The parity bit is set in _parityBit (SerialStatus, 7)
; Common Routine for Both Transmission & Reception
; Program Memory : 13 locations
; Cycles : 14
;*****************************************************************************************************

if _PARITY_ENABLE

GenParity:
    movwf    temp2
    swapf    temp2, w
    xorwf    temp2, w
    movwf    temp1
    rrf      temp1
    rrf      temp1
    xorwf    temp1
    incf     temp1
    rrf      temp1 ; parity bit in Bit 0
    ; Parity bit is in Bit 0 of temp1
    
    if _ODD_PARITY
        bsf _    parityBit
        btfsc    temp1, 0
        bcf _    parityBit
        else
            bcf _    parityBit
            btfsc temp1, 0
            bsf _    parityBit
            endif
        endif

return
endif
;*****************************************************************************************************
Appendix E - RS232.LST

MPASM B0.24
"RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
"Software Implementation : Interrupt Driven"

TITLE "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
SUBTITLE "Software Implementation : Interrupt Driven"

*********************************************************************************************************
Software Implementation Of RS232 Communications Using PIC16CXX
Half-Duplex

These routines are intended to be used with PIC16C6X/7X family. These routines can be
used with processors in the 16C6X/7X family which do not have on board Hardware Async
Serial Port.
MX,

Description:

Half Duplex RS-232 Mode Is implemented in Software.
Both Reception & Transmission are Interrupt driven
Only 1 peripheral (RTCC) used for both transmission & reception
RTCC is used for both timing generation (for bit transmission & bit polling)
and Start Bit Detection in reception mode.
This is explained in more detail under Interrupt Subroutine.
Programmable Baud Rate (speed depending on Input Clock Freq.), programmable
#of bits, Parity enable/disable, odd/even parity is implemented.
Parity & Framing errors are detected on Reception
RS-232 Parameters

The RS-232 Parameters must be defined as shown below:

ClkIn : Input Clock Frequency of the processor
(NOTE : RC Clock Mode Is Not Suggested due to wide variations)
BaudRate : Desired Baud Rate. Any valid value can be used.
The highest Baud Rate achievable depends on Input Clock Freq.
600 to 4800 Baud was tested using 4 Mhz Input Clock
600 to 19200 Baud was tested using 10 Mhz Input Clock
Higher rates can be obtained using higher Input Clock Frequencies.
Once the _BaudRate & _ClkIn are specified the program
automatically selects all the appropiate timings
DataBits : Can specify 1 to 8 Bits.
StopBits : Limited to 1 Stop Bit. Must set it to 1.
PARITY_ENABLE : Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
description below
ODD_PARITY : Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
EVEN Parity Scheme is used.
This flag is ignored if _PARITY_ENABLE is set to FALSE.

Usage:
An example is given in the main program on how to Receive & Transmit Data
In the example, the processor waits until a command is received. The command is interpreted
as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
The RS-232 Control/Status Reg's bits are explained below:

"SerialStatus" : RS-232 Status/Control Register

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>txmtProgress</th>
<th>(1 if transmission in progress, 0 if transmission is complete)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>After a byte is transmitted by calling &quot;PutChar&quot; function, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user's code can poll this bit to check if transmission is complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit is reset after the STOP bit has been transmitted</td>
</tr>
<tr>
<td>Bit 1</td>
<td>txmtEnable</td>
<td>Set this bit to 1 on initialization to enable transmission.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit can be used to Abort a transmission while the transmitter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is in progress (i.e when _txmtProgress = 1)</td>
</tr>
<tr>
<td>Bit 2</td>
<td>rcvProgress</td>
<td>Indicates if the receiver is in middle of reception. It is reset when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a byte is received.</td>
</tr>
<tr>
<td>Bit 3</td>
<td>rcvOver</td>
<td>This bit indicates the completion of Reception of a Byte. The user's</td>
</tr>
<tr>
<td></td>
<td></td>
<td>code can poll this bit after calling &quot;GetChar&quot; function. Once &quot;GetChar&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>function is called, this bit is 0 and is set to 1 after reception of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a complete byte (parity bit if enabled &amp; stop bit)</td>
</tr>
<tr>
<td>Bit 4</td>
<td>ParityErr</td>
<td>A 1 indicates Parity Error on Reception (for both even &amp; odd parity)</td>
</tr>
<tr>
<td>Bit 5</td>
<td>FrameErr</td>
<td>A 1 indicates Framing Error On Reception</td>
</tr>
<tr>
<td>Bit 6</td>
<td>unused_</td>
<td>Unimplemented Bit</td>
</tr>
<tr>
<td>Bit 7</td>
<td>parityBit</td>
<td>The 9 th bit of transmission or reception (status of PARITY bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if parity is enabled)</td>
</tr>
</tbody>
</table>

To Transmit A Byte Of Data:
1) Make sure _txmtProgress & _rcvOver bits are cleared
2) Load TxReg with data to be transmitted
3) CALL PutChar function

To Receive A Byte Of Data:
1) Make sure _txmtProgress & _rcvOver bits are cleared
2) CALL GetChar function
3) The received Byte is in TxReg after _rcvOver bit is cleared

Processor 16C71
Radix DEC
EXPAND

include "d:\pictools\16Cxx.h"

;*********************************************************************************************************
; Setup RS-232 Parameters
;*********************************************************************************************************

000F 4240 Clkin equ 1000000 ; Input Clock Frequency is 4 MHz
0480 BaudRate set 1200 ; Baud Rate (bits per second) is 1200
0008 DataBits set 8 ; 8 bit data, can be 1 to 8
0001 StopBits set 1 ; 1 Stop Bit, 2 Stop Bit is not implemented
0046 #define _ PARITY_ENABLE FALSE ; NO Parity
0047 #define _ ODD_PARITY FALSE ; EVEN Parity, if Parity enabled
0048 #define _ USE_RTSCTS FALSE ; NO Hardware Handshaking is Used

include "rs232.h"

;*********************************************************************************************************
; Setup RS-232 Parameters
;*********************************************************************************************************

0000 2811 ORG ResetVector
     goto Start

0004 2824 ORG IntVector
     goto Interrupt

;*********************************************************************************************************
; Table Of ADCONO Reg
; Inputs : WREG (valid values are 0 thru 3)
; Returns In WREG, ADCONO Value, selecting the desired Channel
; Program Memory : 6 locations
; Cycles : 5
;*********************************************************************************************************

GetADCon0:
0005 3903 andlw 0x03 ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
addwf pcl
retlw (0xc1 | (0 << 3)) ; channel 0
retlw (0xc1 | (1 << 3)) ; channel 1
retlw (0xc1 | (2 << 3)) ; channel 2
GetADCon0_End:
retlw (0xc1 | (3 << 3)) ; channel 3
if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
    MESSG "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
endif

;******************************************************************************
; Initialize A/D Converter
; <RA0:RA3> Configure as Analog Inputs, VDD as Vref
; A/D Clock Is Internal RC Clock
; Select Channel 0
;
; Program Memory : 6 locations
; Cycles : 7
;
;******************************************************************************

InitAtoD:
    bsf rp0
    clrf adcon1
    bcf rp0
    movlw 0xc1
    movwf adcon0
    return

;******************************************************************************
; Main Program Loop
;
; After appropriate initialization, The main program wait for a command from RS-232
; The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
; After a command is received, the appropriate A/D Channel is seleted and when conversion is
; completed the A/D Data is transmitted back to the Host. The controller now waits for a new
; command.
;******************************************************************************

Start:
    call InitSerialPort
    call InitAtoD
;
    WaitForNextSel:
    if _USE_RTSCTS
        bcf rp0
        bcf RTS
    ; ready to accept data from host
    endif

call GetChar          ; wait for a byte reception
btfsc rcvOver         ; _rcvOver gets cleared when a Byte Is Received (in RxReg)
goto $-1              ; USER can perform other jobs here, can poll _rcvOver bit

; A Byte is received, Select The Desired Channel & TXMC the desired A/D Channel Data

bcf rp0               ; make sure to select Page 0
movf RxReg,w          ; _REG = Commanded Channel # (0 thru 3)
call GetADC0nC        ; Set ADC0nC Reg Constant from Table Lookup
ncp

bsf go                ; start conversion
btfas done

bcf rp0
movf addr,w
movwf TxReg

if _USE_RTSCTS
    bcf RTS         ; Half duplex mode, transmission mode, ask host not to send data
    btfsc CTS      ; Check CTS signal if host ready to accept data
    goto $-1
endif

call PutChar

btfas txmProgress

movf adres,w
movwf TxReg

rs-232 routines

interrupt service routine

only RTCC interrupt is used. RTCC interrupt is used as timing for serial port receive & transmit
since RS-232 is implemented only as a half duplex system, the RTCC is shared by both receive &
transmit modules.

transmission:
RTCC is setup for internal clock increments and interrupt is generated when
RTCC overflows. Prescaler is assigned, depending on the input clock & the
desired baud rate.

reception:
when put in receive mode, RTCC is setup for external clock mode (falling edge)
and preloaded with OxFF. When a falling edge is detected on RTCC pin, RTCC is
rolls over and an interrupt is generated (thus start bit detect). Once the start
bit is detected, RTCC is change to internal clock mode and RTCC is preloaded
with a certain value for regular timing interrupts to poll RTCC pin (i.e. RX pin).

*********************************************************************************************************
RS-232 Routines
*********************************************************************************************************

interrupt service routine

only RTCC interrupt is used. RTCC interrupt is used as timing for serial port receive & transmit
since RS-232 is implemented only as a half duplex system, the RTCC is shared by both receive &
transmit modules.

transmission:
RTCC is setup for internal clock increments and interrupt is generated when
RTCC overflows. Prescaler is assigned, depending on the input clock & the
desired baud rate.

reception:
when put in receive mode, RTCC is setup for external clock mode (falling edge)
and preloaded with OxFF. When a falling edge is detected on RTCC pin, RTCC is
rolls over and an interrupt is generated (thus start bit detect). Once the start
bit is detected, RTCC is changed to internal clock mode and RTCC is preloaded
with a certain value for regular timing interrupts to poll RTCC pin (i.e. RX pin).

*********************************************************************************************************
Interrupt:

```assembly
  btfs rtif
  retfie ; other interrupt, simply return & enable GIE

; Save Status On INT: WREG & STATUS Regs

  movwf SaveWReg

; affects no STATUS bits: Only way OUT to save STATUS Reg ??????

  swapf status,w

  movwf SaveStatus

;........

  btfs 4 rtmxtProgress
  goto 7 XmtNextBit ; Xmt Next Bit

  btfs 9 rctvProgress
  goto RcvNextBit ; Receive Next Bit

  goto SBitDetected ; Must be start Bit

;*******************************************************************************~*************************
```

```assembly
  ; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
  ; Configure, RX Pin (RTCC pin) as Input, which is used to poll data on reception
  ; Program Memory: 8 locations
  ; Cycles: 9
  ;*******************************************************************************~****************************************
```

```assembly
  ; InitSerialPort:
  clrf SerialStatus

  bcf rp0 ; select Page 0 for Port Access

  bcf TX ; make sure TX Pin is high on powerup, use RB Port Pullup

  bcf rp0 ; Select Page 1 for TrisB access

  bcf TX ; set TX Pin As Output Pin, by modifying TrisS

  if _USE_RTSCTS
    bcf RTS ; RTS is output signal, controlled by PIC16Cxx
  bcf CTS ; CTS is input signal, controlled by the host
  endif

  bcf RX_Pin ; set RX Pin As Input for reception

  return
```

```assembly
  ;*******************************************************************************~*************************
```
include "txmtr.asm"

; The Transmit routines are in file "txmtr.asm"

; PutChar Function
;
; Function to transmit A Byte Of Data
; Before calling this routine, load the Byte to be transmitted into TxReg
; Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
; calling this routine
;
; Program Memory : 6 locations (10 locations if PARITY is Used)
; Cycles : 8 (13 if PARITY is Used)
;
;********************************************************************************************

PutChar:

bsf txmtEnable ; enable transmission
bsf txmtProgress ; Macro to load bit count

LIGAP_BITCOUNT

movlw DataBits+_StopBits
movwf BitCount

if _PARITY_ENABLE
    movlw 2
    movwf ExtraBitCount
endif

if _PARITY_ENABLE
    movf TxReg,W
    call GenParity ; If Parity is used, then Generate Parity Bit
endif

    call TxmtStartBit
    bsf rtie ; Enable RTCC Overflow INT
    retfie ; return with _GIE Bit Set.

;********************************************************************************************

; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 15 locations (25 locations if PARITY is Used)
; Cycles : 13 Worst Case
;------------------------------------------------------------------------------------------------------------------

TxmtNextBit:
  bcf rp0
LOAD_RTCC  RttcPreLoad, RttcPrescale  ; Macro to reload RTCC

  if(UsePrescale == 0)
    movlw -RttcPreLoad + Cycle_Offset1
  else
    movlw -RttcPreLoad + (Cycle_Offset1 >> (RttcPrescale+1))
  endif

  movwf rttcc  ; Note that Prescaler is cleared when RTCC is written

  if _PARITY_ENABLE
    movf BitCount
    btfsc z
    goto ParityOrStop
  endif

  decfsz BitCount
  goto NextTxmtBit

  if _PARITY_ENABLE
    decfsz ExtraBitCount
    goto SendParity
  endif

  _StopBit:
    bsf TX  ; STOP Bit is High
    bcf rtie  ; disable further interrupts

  _NextTxmtBit:
    rrf TxReg
    btfss carry
    bcf TX
    btfsc carry
    bcf TX

    btfss txmtEnable
    bcf rtie  ; disable further interrupts, Transmission Aborted
goto RestoreIntStatus

if _PARITY_ENABLE
_SendParity:
btfss parityBit
bcf TX
btfsc parityBit
bsf TX
goto RestoreIntStatus
eendif

*********************************************************************************************************
Program Memory Cycles 10 locations
;**************************************************************************************~******************************
TxmtStartBit:
bsf rp0
movlw (OPTION_INIT | RtcPrescale) ; Set Option Reg Located In Page 1
movwf option ; make sure to select Page 0
bcf rp0 ; Send Start Bit
bcf TX
movlw -RtcPreLoad ; Prepare for Timing Interrupt
movwf rtcc
bcf rtif
return

*******************************************************************************~***************************
Generate Parity for the Value in WREG
*******************************************************************************~***************************
if _PARITY_ENABLE
GenParity:
movwf temp2
swapf temp2,w
xorwf temp2,w
movwf temp1
rrf temp1
rrf templ
xorwf templ
incf templ
rrf templ
; Parity bit is in Bit 0 of templ

if _ODD_PARITY
   bsf parityBit
   btfsc temp1,0
   bcf parityBit
else
   bcf parityBit
   btfsc temp1,0
   bsf parityBit
endif

return

;***********************************************************************
The Receiver Routines are in File “rcvr.asm”
;***********************************************************************

getchar:
bsf rcvOver

;***********************************************************************
; GetChar Function
; Receives a Byte Of Data
; When reception is complete, _rcvOver Bit is cleared
; The received data is in RxReg
; Program Memory : 15 locations (17 locations if PARITY is used)
; Cycles : 16 (18 if PARITY is USED)
;***********************************************************************

movlw DataBits+_StopBits
movwf BitCount

if PARITY_ENABLE
movlw 2
movwf ExtraBitCount
endif
0060 018D  clrf RxReg
0061 128F  bcf FrameErr
0062 120F  bcf ParityErr
0063 1683  bsf rp0
0064 3038  movlw OPTION_SBIT  ; Inc On Ext Clk Falling Edge
0065 0081  movwf option       ; Set Option Reg Located In Page 1
0066 1293  bcf rpC          ; make sure to select Page 0
0067 30FF  movlw 0xFF
0068 0081  movwf rttc
0069 110B  bcf rtif
006A 168B  bsf rtie
006B 0009  retifie

;******************************************************************************
;  Internal Subroutine
;  entered from Interrupt Service Routine when Start Bit Is detected.
;  Program Memory    :       14 locations
;  Cycles            :       12 (worst case)
;******************************************************************************
SBitDetected:
006C 1283  bcf rp0
006D 1A05  btfsc RX_Pin       ; Make sure Start Bit Interrupt is not a Glitch
006E 2877  goto FalseStartBit
006F 150F  bsf rcvProgress
0070 1683  bsf rp0
0071 3008  movlw (_OPTION_INIT | SBitPrescale); Switch Back to INT Clock
0072 0081  movwf option       ; Set Option Reg Located In Page 1
0073 1293  bcf rp0           ; make sure to select Page 0
0074 3090  LOAD_RTCC     (SBitRttcLoad), SBitPrescale
    if(UsePrescale == 0)
      movlw -(SBitRttcLoad) + _Cycle_Offset1
    else
      movlw -(SBitRttcLoad) + (_Cycle_Offset1 >> (SBitPrescale+1)) ; Re Load RTCC init value + INT La
    endif
0075 0081  movwf rtcc       ; Note that Prescaler is cleared when RTCC is written
0076 282E  goto RestoreIntStatus

; FalseStartBit:
0077 30FF  movlw 0xFF
0078 0081  movwf rtcc       ; reload RTCC with 0xFF for start bit detection
goto RestoreIntStatus
;
;*****************************************************************************************
; Program Memory
; Cycles : 28 locations (43 locations with PARITY enabled)
; Cycles : 24 Worst Case
;
_RcvNextBit:
  bsr rp0
  movlw (OPTION_INIT | RtccPrescale) ; Switch Back to INT Clock
  movwf option ; Set Option Reg Located In Page 1
  bcf rp0
  movf porta, w
  movwf Rx Temp
  LOAD_RRTC RtccPreLoad, RtccPrescale ; Macro to reload RTCC

if(UsePrescale == 0)
  movlw -RtccPreLoad + _Cycle_Offset1
else
  movlw -RtccPreLoad + (Cycle_Offset1 >> (RtccPrescale+1))
  ; Re Load RTCC init value + INT La
endif

movwf rtcc ; Note that Prescaler is cleared when RTCC is written
movf porta, w
xorwf RxTemp, w
andlw RX_MASK
bfsz z
goto PinSampled
SampleAgain:
  movf porta, w
  movwf RxTemp
PinSampled:
  if _PARITY_ENABLE
    movf BitCount
    bsrz z
    goto RcvP_Or_S
  endif
  decfsz BitCount
  goto NextRcvBit
RcvP_Or_S:
decfsz ExtraBitCount
  goto RcvParity
endif

_RcvStopBit:
  btfss RX
  bcf FrameErr ; may be framing Error or Glitch
  bcf rtie ; disable further interrupts
  bcf rcvProgress
  bcf rcvOver ; Byte Received, Can RCV/TXMT an other Byte
  if _PARITY_ENABLE
    movf RxReg, w
    call GenParity ; Generate Parity, for Parity check
    movlw 0
    btfsr parityBit
    movlw 0x10 ; to mask off Received Parity Bit in _ParityErr
    xorwf SerialStatus ; _ParityErr bit is set accordingly
  endif

  goto RestoreIntStatus

_RcvStopBit:
  bcf carry
  btfsr RX ; may be a glitch, check again
  bcf carry
  rrf RxReg ; shift in received data
  goto RestoreIntStatus

  if _PARITY_ENABLE
    RcvParity:
      bcf ParityErr ; Temporarily store PARITY Bit in _ParityErr
      btfsr RX
      bcf ParityErr
      goto RestoreIntStatus
      endif

END
INTRODUCTION

The PIC16C71 is a member of a new family of 8-bit microcontrollers, namely the PIC16CXX. The salient features of the PIC16C71 are:

- Improved and enhanced instruction set
- 14-bit instruction word
- Interrupt capability
- On-chip four channel, 8-bit A/D converter

This application note demonstrates the capability of the PIC16C71. To make this application note easier for the end user, it has been broken down into four sub-sections:

Section 1: Implements the multiplexing of four 7 segment LEDs with the PIC16C71.
Section 2: Implements the multiplexing of four 7 segment LEDs as well as the sampling of a 4x4 Keypad.
Section 3: Implements the multiplexing of four 7 segment LEDs as well as the sampling of one A/D channel.
Section 4: Implements the multiplexing of four 7 segment LEDs, sampling a 4x4 keypad and four A/D channels.

IMPLEMENTATION

SECTION 1: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS

Hardware

The PIC16C71's I/O ports have an improved sink/source specification. Each I/O pin can sink up to 25 mA and source 20 mA, in addition total Port B source current is 100 mA and sink current is 150 mA. Port A is rated for 50 mA source current and 80 mA sink current. This makes the PIC16C71 ideal for driving 7 segment LEDs. Since the total number of I/O is limited to 13, the 8-bit Port B is used to drive the 8 LEDs, while external sink transistors or MOSFETs are used to sink the digit current (See Figure 1). Another alternative is to use ULN2003 open collector sink current drivers, which are available in 16 pin DIP or very small S0-16 packages. Each transistor on the ULN2003 can sink a maximum of 500 mA and the base drive can be directly driven from the Port A pins.

FIGURE 1 - MULTIPLEXING FOUR 7 SEGMENT LEDS
**Software**

The multiplexing is achieved by turning on each LED for a 5 msec duration every 20 msec. This gives an update rate of 50 Hz, which is quite acceptable to the human eye as a steady display. The 5 msec time base is generated by dividing the 4.096 MHz oscillator clock. The internal prescaler is configured to be a divide by 32 and assigned to the RTCC. The RTCC is pre-loaded with a value = 96. The RTCC will increment to FF and then roll over to 00 after a period = \((256-96)*(32*4/4096000)\) = 5 msec. When the RTCC rolls over, the RTIF flag is set and since the RTIE and GIE bits are enabled, an interrupt is generated.

The software implements a simple timer which increments at a 1 second rate. Every second, the 4 nibble (two 8-bit registers MsdTime and LsdTime) are incremented in a BCD format. The lower 4 bits of LsdTime correspond to the least significant digit (LSD) on the display. The high 4 bits of LsdTime correspond to the second significant digit of the display and so on. Depending on which display is turned on, the corresponding 4 bit BCD value is extracted from either MsdTime or LsdTime, and decoded to a 7 segment display. The RTCC interrupt is generated at a steady rate of 5 msec and given an instruction time of 1 us. The entire display update program can reside in the interrupt service routine with no chance of getting an interrupt within an interrupt. The Code listing for Section 1 is in Appendix A.

**SECTION 2: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND SCANNING A 4X4 KEYPAD**

**Hardware**

A 4x4 keypad can be very easily interfaced to the PIC16C71’s Port B (see Figure 2). Internal pull-ups on pins RB4 to RB7 can be enabled and disabled by setting the RBPU bit in the OPTION register. The internal pull-ups have a value of 20K at 5V (typical). In order to sense a low level at the input, the switch is "connected" to ground through a 2.2K resistor. A key hit normally lasts from 50 msec to as long as a person holds the key down. In order not to miss any key hits, the keypad is sampled every 20 msec (just after the update of the MSD).

**Software**

To sample the keypad, the digit sinks are first disabled. Port B is then configured with RB4-RB7 as inputs and RB0-RB3 as outputs driven high. The pull-ups on RB4-RB7 are enabled. Sequentially RB0 to RB3 are made low while RB4 to RB7 are checked for a key hit (a low level). One key hit per scan is demonstrated in this program. Multiple key hits per scan can very easily be implemented. Once the key hit is sensed, a 40 msec debounce period elapses before key sampling is resumed. No more key hits are sensed until the present key is released. This prevents erroneous key inputs.

The program basically inputs the key hit and displays its value as a hexadecimal character on the multiplexed 7-segment LEDs. The Code Listing for Section 2 is in Appendix B.
SECTION 3: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND THE A/D CHANNEL 0

Hardware
The four analog channels are connected to RA0-RA3. If any of these pins are used normally as digital I/O, they can momentarily be used as analog inputs. In order to avoid interference from the analog source, it is advisable to buffer the analog input through a voltage follower op amp, however, it is not always necessary. Figure 3A and 3B show some typical configurations. In this application, the analog input is a potentiometer whose wiper is connected through an RC network to channel 0. The RC is necessary in order to smooth out the analog voltage. The RC does contribute to a delay in the sampling time, however the stability of the analog reading is greatly improved.

Software
The analog input is sampled every 20 msec. The digit sinks and the drivers are turned off i.e. Port A is configured as an input and Port B outputs are made low. A 1msec settling time is allowed for the external RC network connected to the analog input to settle and then the A/D conversion is started. The result is read then converted from an 8-bit binary value to a 3 digit BCD value which is then displayed on the 7 Segment LEDs. The Code Listing for Section 3 is in Appendix C.
SECTION 4: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS WITH A 4X4 KEYPAD AND 4 A/D CHANNELS

Hardware

This section essentially incorporated Sections 1, 2 and 3 to give a complete four channel voltmeter. Figure 4 shows a typical configuration. The analog channels are connected through individual potentiometers to their respective analog inputs and are sampled every 20 msec in a round robin format. The sampling rate can be increased to as fast as once every 5 msec if required. The keypad sampling need not be any faster than once every 20 msec.

Software

The program samples the analog inputs and saves the result in four consecutive locations starting at "ADVALUE", with channel 0 saved at the first location and so on. By default, channel 0 is displayed. If Key 1 is pressed, channel 1 is displayed and so on. Key hits > 3 are ignored. The Code Listing for Section 4 is in Appendix D.

Conclusion

The 4 A/D channels on the PIC16C71 can be multiplexed with digital I/O, thus reducing overall pin counts and improving I/O pin usage in an analog application.

AUTHOR: Stan D'Souza, Logic Products Division
This program is to demonstrate how to multiplex four 7 segment LED digits using a PIC16C71. The four digits will start at 0000 and increment at a 1 sec rate up to 9999. The LEDs are updated every 5 msec, for a multiplexing rate of 20 msec. The RTCC timer is used in internal interrupt mode to generate the 5 msec.

Stan D'Souza 5/8/93

LIST P=16C71, F=INHX8M

;*****************************************************************************
;This program is to demonstrate how to multiplex four 7 segment LED digits using a PIC16C71. The four digits will start at 0000 and increment at a 1 sec rate up to 9999. The LEDs are updated every 5 msec, for a multiplexing rate of 20 msec. The RTCC timer is used in internal interrupt mode to generate the 5 msec.
;
;*****************************************************************************

PAGE

;*****************************************************************************
;*************** include "picreg.equ"***************

;000C
TempC equ 0x0c ;temp general purpose regs
000D
TempD equ 0x0d
000E
TempE equ 0x0e
000F
Count equ 0x0f ;count
0010
MsdTime equ 0x10 ;most significant Timer
0011
LsdTime equ 0x11 ;least significant Timer
0012
OptionReg equ 1
0026
PCL equ 2
0027
BcdMsd equ 26
0028
Bcd equ 27

org
goto Start ;skip over interrupt vector

org
goto ServiceInterruptions

Start
call InitPorts
call InitTimers
loop
goto loop

;InitPorts

0009 3003
movlw 3 ;make RA0-3 digital I/O
000A 0108
movwf ADCON1 ;
000B 0205
clf TRISA ;make RA0-4 outputs
000C 0206
clf TRISB ;make RB0-7 outputs
000D 1283
bcf STATUS,RP0 ;select page 0
000E 0185
clf PORT_A ;make all outputs low
000F 0186
clf PORT_B ;
0010 1585
bsf PORT_A,3 ;enable MSB digit sink
0011 0008
return

;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler, the rtcc will be incremented every 31.25uS. If rtcc is preloaded with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the end result is that we get a rtcc interrupt every 5msec.

InitTimers

0012 0190
cclf MsdTime ;clr timers
0013 0191
cclf LsdTime ;
0014 1683
bsf STATUS,RP0 ;select pg 1
0015 3084
movlw B'10000100' ;assign ps to rtcc

;*****************************************************************************
Four Channel Digital Voltmeter with Display and Keyboard

0016 0081          movwf OptionReg               ;ps - 32
0017 1283          bcf STATUS,RP0               ;select pg 0
0018 3020          movlw B'00100000'           ;enable rtcc interrupt
0019 008B          movwf INTCON               ;
001A 3060          movlw .96                  ;preload rtcc
001B 0081          movwf RTCC                 ;start counter
001C 0009          retfie

; ServiceInterrupts
001D 190B          btfsc INTCON,RTIF           ;rtcc interrupt?
001E 2822          goto ServiceRTCC           ;yes then service
001F 3020          movlw B'00100000'           ;else clr rest
0020 008B          movwf INTCON               ;
0021 0009          retfie

; ServiceRTCC
0022 3060          movlw .96                  ;initialize rtcc
0023 0081          movwf RTCC
0024 110B          bcf INTCON,RTIF           ;clr int flag
0025 2028          call IncTimer             ;inc timer
0026 2050          call UpdateDisplay        ;update display
0027 0009          retfie

; The display is incremented every 200*5msec = 1 Sec.

IncTimer
0028 0A0F          incf Count,w               ;inc count
0029 3AC8          xorlw .200                ;= 200?
002A 1903          btfsc STATUS,Z            ;no then skip
002B 282E          goto DoIncTime             ;else inc time
002C 0A8F          incf Count
002D 0008          return

DoIncTime
002E 018F          clrf Count                 ;clr count
002F 0A11          incf LsdTime,w             ;get lsd
0030 390F          andlw 0x0F                ;mask high nibble
0031 3AOA          xorlw 0x0a                 ;= 10?
0032 1903          btfsc STATUS,Z            ;no then skip
0033 2836          goto IncSecondLsd         ;inc next lsd
0034 0A91          incf LsdTime
0035 0008          return

IncSecondLsd
0036 0E11          swapf LsdTime,w            ;get hi in low nibble
0037 390F          andlw 0x0F                ;mask hi nibble
0038 3E01          addlw 1                   ;inc it
0039 090F          movwf LsdTime              ;restore back
003A 0E91          swapf LsdTime
003B 3AOA          xorlw 0x0a                 ;= 10?
003C 1903          btfsc STATUS,Z            ;no then skip
003D 283F          goto IncThirdLsd          ;else inc next lsd
003E 0008          return

IncThirdLsd
003F 0191          clrf LsdTime               ;get 3rd lsd
0040 0A10          incf MsdTime,w             ;mask hi nibble
0041 390F          andlw 0x0F                ;= 10?
0042 3AOA          xorlw 0x0a                 ;no then skip
0043 1903          btfsc STATUS,Z            ;else Msd
0044 2847          goto IncMsd               ;else inc timer
0045 0A90          incf MsdTime
0046 0008          return

IncMsd
0047 0E10          swapf MsdTime,w            ;get hi in lo nibble
0048 390F          andlw 0x0F                ;mask hi nibble
0049 3E01          addlw 1                   ;inc timer
004A 090F          movwf MsdTime              ;restore back
004B 0E90          swapf MsdTime
004C 3AOA          xorlw 0x0a                 ;= 10?
004D 1903          btfsc STATUS,Z            ;no then skip
004E 0190          clrf MsdTime               ;clr msd
004F 0008          return
Four Channel Digital Voltmeter with Display and Keyboard

UpdateDisplay

0050 0805 movf PORT_A, w ; present sink value in w
0051 0185 clrf PORT_A ; disable all digits sinks
0052 390F andlw 0x0f
0053 008C movwf TempC ; save sink value in tempC
0054 160C bsf TempC, 4 ; preset for lsd sink
0055 0C8C rrf TempC ; determine next sink value
0056 1C03 btfs STATUS, CARRY ; c=1?
0057 180C bcf TempC, 3 ; no then reset Lsd sink
0058 18C0 bcf TempC, 0 ; else see if Msd
0059 286B goto UpdateMsd ; yes then do Msd
005A 190C btfsc TempC, 1 ; see if 3rd Lsd
005B 2866 goto Update3rdLsd ; yes then do 3rd Lsd
005C 190C btfsc TempC, 2 ; see if 2nd Lsd
005D 2961 goto Update2ndLsd ; yes then do 2nd lsd

UpdateLsd

005E 0811 movf LsdTime, w ; get Lsd in w
005F 390F andlw 0x0f ;
0060 286F goto DisplayOut ; enable display

Update2ndLsd

0061 2080 call Chk2LsdZero ; msd = 0 & 2 lsd 0?
0062 1D03 btfs STATUS, Z ; yes then skip
0063 0E11 swapf LsdTime, w ; get 2nd Lsd in w
0064 390F andlw 0x0f ; mask rest
0065 286F goto DisplayOut ; enable display

Update3rdLsd

0066 2088 call ChkMsdZero ; msd = 0?
0067 1D03 btfs STATUS, Z ; yes then skip
0068 0810 movf MsdTime, w ; get 3rd Lsd in w
0069 390F andlw 0x0f ; mask low nibble
006A 286F goto DisplayOut ; enable display

UpdateMsd

006B 0E10 swapf MsdTime, w ; get Msd in w
006C 390F andlw 0x0f ; mask rest
006D 1903 btfs STATUS, Z ; msd != 0 then skip
006E 300A movlw 0x0a

DisplayOut

006F 2074 call LedTable ; get digit output
0070 0086 movwf PORT_B ; drive leds
0071 080C movwf PORT_C, w ; drive leds
0072 0085 movwf PORT_A ; drive leds
0073 0008 return

LedTable

0074 0782 addwf PCL ; add to PC low
0075 343F retlw B'00111111' ; led drive for 0
0076 3406 retlw B'00000110' ; led drive for 1
0077 345B retlw B'01011101' ; led drive for 2
0078 344F retlw B'01001111' ; led drive for 3
0079 3466 retlw B'01100110' ; led drive for 4
007A 346D retlw B'01101101' ; led drive for 5
007B 347D retlw B'01111101' ; led drive for 6
007C 3407 retlw B'00000111' ; led drive for 7
007D 347F retlw B'01111111' ; led drive for 8
007E 3467 retlw B'01100111' ; led drive for 9
007F 3400 retlw B'00000000' ; blank led drive

Chk2LsdZero

0080 2088 call ChkMsdZero ; msd = 0?
0081 1D03 btfs STATUS, Z ; yes then skip
0082 0008 return ; else return
0083 0E11 swapf LsdTime, w ; get 2nd Lsd
0084 390F andlw 0x0f ; mask of Lsd
0085 1D03 btfs STATUS, Z ; 0? then skip
0086 0008 return
Four Channel Digital Voltmeter with Display and Keyboard

0087 340A  retlw 10 ;else return with 10

; ChkMsdZero
0088 0810  movf  MsdTime,w ;get Msd in w
0089 1D03  btfss STATUS,Z ;= 0? skip
008A 0008  return ;else return
008B 340A  retlw .10 ;ret with 10
Appendix B

MPASM B0.50

;*********************************************************************
;This program is to demonstrate how to multiplex four 7 segment LED
digits and a 4X4 keypad using a PIC16C71.
The four digits will start as '0000' and when a key is hit
it is displayed on the 7 segment leds as a hex value 0 to F. The last
digit hit is always displayed on the right most led with the rest of
the digits shifted to the left. The left most digit is deleted.
The LEDs are updated every 20msec, the keypad is scanned at a rate of 20
msec.
The RTCC timer is used in internal interrupt mode to generate the 5 msec.
; Stan D'Souza 5/8/93
;*********************************************************************

LIST P=16C71, F=INHX8M
; include "picreg.equ"

; temp general purpose regs
TempC equ Ox De
TempD equ Ox0d
TempE equ Ox0e
PABuf equ Ox20
PBBuf equ Ox21
Count equ Ox Of
MsdTime equ OxlO
LsdTime equ Oxll
KeyFlag equ Ox12
DebnceOn equ 0
noentry equ 2
ServKey equ 3
Debnce eq 0
NewKey equ Ox14
WBuf fer equ Ox2f
StatBuffer equ Ox2e
OptionReg equ 1
PCL equ 2

; push macro
movwf WBuf fer ;save w reg in Buffer
swapf WBuffer ;swap it
swapf STATUS,w ;get status
movwf StatBuffer ;save it
endm

; pop macro
swapf StatBuffer,w ;restore status
movwf STATUS ;/
swapf WBuffer,w ;restore W reg
endm

org 0
goto Start ;skip over interrupt vector

org 4
; It is always a good practice to save and restore the w reg,
; and the status reg during a interrupt.
push
movwf WBuffer ;save w reg in Buffer
swapf WBuffer ;swap it
swapf STATUS,w ;get status
movwf StatBuffer ;save it
call ServiceInterrupty

Four Channel Digital Voltmeter with Display and Keyboard

0009 0E2E
000A 0083
000B 0E2F
000C 0009

;pop
;swapf StatBuffer,w
;movwf STATUS
;swapf WBuref,w
;
;retfie

;Start

call InitPorts

call InitTimers

loop

btfs KeyFlag,ServKey ;key service pending

call ServiceKey ;yes then service

goto loop

;ServiceKey, does the software service for a keyhit. After a key service, ;the ServKey flag is reset, to denote a completed operation.

ServiceKey

NewKey,w ;get key value

TempE ;save in TempE

MsdTime,w ;move MSD out

andw B'11110000'

MsdTime ;clr lo nibble

movwf MsdTime ;save back

LsdTime,w ;get Lsd

andw B'00001111'

MsdTime ;mask off lsd

MovB MadTime ;and left shift 3rd

Iorwf TempE,w ;get Mad again

MsdTime ;or with new lsd

movwf LsdTime ;make Lsd

bcf KeyFlag,ServKey ;reset service flag

InitPorts

STATUS ;select pg 1

ADCON1 ;make RA0-3 digital I/O

clrf TRISA ;make RA0-4 outputs

clrf PORT_B ;make RB0-7 outputs

bcf STATUS,RPO ;select page 0

clrf PORT_A ;make all outputs low

clrf PORT_A,3 ;enable MSB digit sink

return

;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler, ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded ;with 96, it will take (256-96) *31.25us to overflow i.e. 5msec. So the ;end result is that we get a rtcc interrupt every 5msec.

InitTimers

clrf MadTime ;clr timers

clrf LsdTime ;

clrf KeyFlag ;clr all flags

bsf STATUS,RPO ;select pg 1

movlw B'10000100' ;movps to rtcc

movwf OptionReg ;ps = 32

bcf STATUS,RPO ;select pg 0

movlw B'00100000' ;enable rtcc interrupt

movwf INTCON ;

movlw .96 ;preload rtcc

movwf RTCC ;start counter

retfie

ServiceInterrupts

bsf INTCON,RTIF ;rtcc interrupt?

goto ServiceRTCC ;yes then service

clrf INTCON ;else clr all int

bsf INTCON,RTIE

© 1993 Microchip Technology Inc.
Four Channel Digital Voltmeter with Display and Keyboard

ServiceRTCC
movlw .96 ;initialize rtcc
movwf RTCC
bcf INTCON,RTIF ;clr int flag
btfsc PORT_A,0 ;if mb on then do
call ScanKeys ;do a quick key scan
call UpdateDisplay ;update display
return

ScanKeys
btfss KeyFlag,DebnceOn ;debounce on?
goto Scan1 ;no then scan keypad
decfsz Debnce ;else dec debounce time
return
bcf KeyFlag,DebnceOn ;over, clr debounce flag
return ;and return

Scan1
call SavePorts ;save port values
movlw B'11110111' ;init TempD
movwf TempD
ScanNext
movf PORT_B,w ;read to init port
bcf INTCON,RBIF ;clr flag
rff TempD ;get correct column
btfss STATUS,C ;if carry set?
goto NoKey ;no then end
movf TempD,w ;else output
movwf PORT_B ;low column scan line
ncp
btfss INTCON,RBIF ;flag set?
goto ScanNext ;no then next
btfsc KeyFlag,keyhit ;last key released?
goto SReturn ;no then exit
bsf KeyFlag,keyhit ;set new key hit
swapf PORT_B,w ;read port
movwf TempE ;save in TempE
call GetKeyValue ;get key value 0 - F
movwf NewKey ;save as New key
bsf KeyFlag,ServKey ;set service flag
bsf KeyFlag,DebnceOn ;set flag
movlw 4
movwf Debnce ;load debounce time
SReturn
call RestorePorts ;restore ports
return

NoKey
bcf KeyFlag,keyhit ;clr flag
goto SReturn

GetKeyValue gets the key as per the following layout

<table>
<thead>
<tr>
<th>Coll1</th>
<th>Coll2</th>
<th>Coll3</th>
<th>Coll3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RB3)</td>
<td>(RB2)</td>
<td>(RB1)</td>
<td>(RB0)</td>
</tr>
<tr>
<td>Row1(RB4)</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Row2(RB5)</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Row3(RB6)</td>
<td>8</td>
<td>9</td>
<td>A</td>
</tr>
<tr>
<td>Row4(RB7)</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>
; GetKeyValue
0064 018C  clr    TempC
0065 1D8D  btfss TempD,3 ; first column
0066 286E  goto RowValEnd
0067 0A8C  incf TempC
0068 1D8D  btfss TempD,2 ; second col.
0069 286E  goto RowValEnd
006A 0A8C  incf TempC
006B 1C8D  btfss TempD,1 ; 3rd col.
006C 286E  goto RowValEnd
006D 0A8C  incf TempC ; last col.
RowValEnd
006E 1C0E  btfss TempE,0 ; top row?
006F 2878  goto GetValCom ; yes then get 0,1,2,3
0070 1C0E  btfss TempE,1 ; 2nd row?
0071 2877  goto Get4567 ; yes then get 4,5,6,7
0072 1D0E  btfss TempE,2 ; 3rd row?
0073 2875  goto Get89ab ; yes then get 8,9,a,b
GetDef
0074 150C  bsf    TempC,2 ; set msb bits
Get89ab
0075 158C  bsf    TempC,3 ; /
0076 2878  goto GetValCom ; do common part
Get4567
0077 150C  bsf    TempC,2
GetValCom
0078 080C  movf TempC,w
0079 0782  addwf PCL
007A 3400  retlw 0
007B 3401  retlw 1
007C 3402  retlw 2
007D 3403  retlw 3
007E 3404  retlw 4
007F 3405  retlw 5
0080 3406  retlw 6
0081 3407  retlw 7
0082 3408  retlw 8
0083 3409  retlw 9
0084 340A  retlw 0a
0085 340B  retlw 0b
0086 340C  retlw 0c
0087 340D  retlw 0d
0088 340E  retlw 0e
0089 340F  retlw 0f

; SavePorts, saves the porta and portb condition during a key scan
; operation.
SavePorts
008A 0805  movf PORT_A, w ; get sink value
008B 00A0  movwf PABuf ; save in buffer
008C 0185  clr    PORT_A ; disable all sinks
008D 0806  movf PORT_B, w ; get port b
008E 00A1  movwf PBBuf ; save in buffer
008F 30FF  movlw 0xff ; make all high
0090 0086  movwf PORT_B ; on port b
0091 1683  bsf STATUS, Rp0 ; select page 1
0092 1381  bcf OptionReg, 7 ; enable pull ups
0093 30F0  movlw B'11110000' ; port b hi nibble inputs
0094 0106  movwf TRISB ; io nibble outputs
0095 1283  bcf STATUS, Rp0 ; page 0
0096 0008  return

; RestorePorts, restores the condition of porta and portb after a
; key scan operation.
RestorePorts
0097 0821  movf PBBuf, w ; get port n
0098 0086  movwf PORT_B
0099 0820  movf PABuf, w ; get port a value
movwf PORT_A
bsf STATUS,RP0 ;select page 1
clrf OptionReg,7 ;disable pull ups
clrf TRISA ;make port a outputs
clrf TRISB ;as well as PORTB
bcf STATUS,RP0 ;page 0
return

UpdateDisplay
movf PORT_A,w ;present sink value in w
andlw 0x0f
goto DisplayOut

UpdateLsd
movf LsdTime,w ;get Lsd in w
andlw 0x0f
goto DisplayOut

Update2ndLsd
swapf LsdTime,w ;get 2nd Lsd in w
andlw 0x0f
goto DisplayOut

Update3rdLsd
movf MsdTime,w ;get 3rd Lsd in w
andlw 0x0f
goto DisplayOut

UpdateMsd
swapf MsdTime,w ;get Msd in w
andlw 0x0f

DisplayOut
call LedTable ;get digit output
movwf PORT_B ;drive leds
movwf PORT_A
return

LedTable
addwf PCL ;add to PC low
retlw B'00000010' ;led drive for 0
retlw B'00000110' ;led drive for 1
retlw B'00110111' ;led drive for 2
retlw B'01001111' ;led drive for 3
retlw B'01100110' ;led drive for 4
retlw B'01101101' ;led drive for 5
retlw B'01111101' ;led drive for 6
retlw B'00000011' ;led drive for 7
retlw B'00111111' ;led drive for 8
retlw B'01011111' ;led drive for 9
retlw B'01111111' ;led drive for A
retlw B'01111100' ;led drive for B
retlw B'00111101' ;led drive for C
retlw B'01011110' ;led drive for D
retlw B'01111101' ;led drive for E
retlw B'01111001' ;led drive for F

end
This program is to demonstrate how to multiplex four 7 segment LED and sample ch0 of the a/d in a PIC16C71. The a/d value is displayed as a 3 digit decimal value of the a/d input (0 - 255). The LEDs are updated every 20msec, the a/d is sampled every 20 msec. The RTCC timer is used in internal interrupt mode to generate the 5 msec.

Stan D'Souza 5/8/93

LIST P=16C71, F=INHX8M

include "picreg.equ"

BcdMsd equ 26
Bcd equ 27
TempC equ 0x0c ;temp general purpose regs
TempD equ 0x0d
TempE equ 0x0e
PABuf equ 0x20
PBBuf equ 0x21
Count equ 0x0f ;count
MedTime equ 0x10 ;most significant Timer
LedTime equ 0x11 ;Least significant Timer
AdFlag equ 0x12 ;flags related to key pad
ADOver equ 5 ;bit 5 \rightarrow a/d over
WBuffer equ 0x2f
StatBuffer equ 0x2e
OptionReg equ 1
PCL equ 2

push macro
movwf WBuffer ;save w reg in Buffer
swapf WBuffer ;swap it
swapf STATUS,w ;get status
movwf StatBuffer ;save it
endm

pop macro
swapf StatBuffer,w ;restore status
movwf STATUS ;/
swapf WBuffer,w ;restore W reg
endm

org 0
goto Start ;skip over interrupt vector

org 4
;It is always a good practice to save and restore the w reg, and the status reg during a interrupt.
push
movwf WBuffer ;save w reg in Buffer
swapf WBuffer ;swap it
swapf STATUS,w ;get status
movwf StatBuffer ;save it

0004 00AF
movwf WBuffer ;save w reg in Buffer
0005 00AE
swapf WBuffer ;swap it
0006 003
swapf STATUS,w ;get status
0007 00AE
movwf StatBuffer ;save it

0008 2039
call ServiceInterrupts

0009 00E2
swapf StatBuffer,w ;restore status
000A 0083
movwf STATUS ;/
; The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler, 
; the rtcc will be incremented every 31.25us. If rtcc is preloaded 
; with 96, it will take (256-96)*31.25us to overflow i.e. 5msec. So the 
; end result is that we get a rtcc interrupt every 5msec.

InitTimers

; select pg 1
clrf MsdTime ; clr timers
clrf LsdTime ; / 
bsf STATUS,RP0 ; select pg 1
movlw B'10000100' ; assign pg to rtcc
movwf OptionReg ; ps = 32
bcf STATUS,RP0 ; select pg 0
movlw B'00100000' ; enable rtcc interrupt
movwf INTOCON ;
movwf .96 ; preload rtcc
movwf RTCC ; start counter
retfie ;

InitAd

movlw B'11001000' ; init a/d
movwf ADCONO
return ;
ServiceInterrupts
039 190B  btsc  INTC,RTIF  ;rtcc interrupt?
03A 283E  goto  ServiceRTCC  ;yes then service
03B 018B  clrf  INTC
03C 168B  bsf  INTC,RTIE
03D 0008  return

ServiceRTCC
03E 3060  movlw .96  ;initialize rtcc
03F 0081  movwf  RTCC
040 110B  bcf  INTC,RTIF  ;clr int flag
041 1C05  btfsc  PORT_A,0  ;last digit?
042 2045  call  SampleAd  ;then sample a/d
043 2071  call  UpdateDisplay  ;else update display
044 0008  return

SampleAd
045 205A  call  SavePorts
046 204C  call  DoAd  ;do a ad conversion

AdDone
047 1908  btsc  ADCON0,GO  ;ad done?
048 2847  goto  AdDone  ;no then loop
049 1692  bsf  ADFlag,ADOVer  ;set a/d over flag
04A 2067  call  RestorePorts  ;restore ports
04B 0008  return

DoAd
04C 0186  clrf  PORT_B  ;turn off leds
04D 1683  bsf  STATUS,RP0  ;select pg 1
04E 300F  movlw Ox0f  ;make port a hi-Z
04F 0105  movwf  TRIA
050 1283  bcf  STATUS,RP0  ;select pg 0
051 1408  bsf  ADCON0,ADON  ;start a/d
052 307D  movlw .125
053 2056  call  Wait
054 1508  bsr  ADON,GO  ;start conversion
055 0008  return

Wait
056 0008  movwf  TempC  ;store in temp
057 088C  decfsz  TempC
058 2857  goto  Next
059 0008  return

SavePorts
05A 0805  movf  PORT_A,w  ;Get sink value
05B 00A0  movwf  PABuf  ;save in buffer
05C 0185  clrf  PORT_A  ;disable all sinks
05D 0806  movf  PORT_B,w  ;get port b
05E 00A1  movwf  PBBuf  ;save in buffer
05F 30FF  movlw Oxff  ;make all high
060 008F  movwf  PORT_B  ;on port b
061 1683  bcf  STATUS,RP0  ;select page 1
062 1381  bcf  OptionReg,?  ;enable pull ups
063 300F  movlw B'11110000'  ;port b hi nibble inputs
064 0106  movwf  TRIB  ;io nibble outputs
065 1283  bcf  STATUS,RP0  ;page 0
066 0008  return

RestorePorts  ;restore the condition of porta and portb after a
;key scan operation.
Four Channel Digital Voltmeter with Display and Keyboard

```
0067 0B21 movf PBBuf,w ;get port n
0068 0086 movwf PORT_B
0069 0B20 movf PABuf,w ;get port a value
006A 0085 movwf PORT_A
006B 1683 bsf STATUS,RP0 ;select page 1
006C 1781 bsf OptionReg,7 ;disable pull ups
006D 0205 clrf TRISA ;make port a outputs
006E 0206 clrf TRISB ;as well as PORTB
006F 1283 bcf STATUS,RP0 ;page 0
0070 0008 return

; RestorePorts
0071 0B05 movf PORT_A,w ;present sink value in w
0072 0B05 clrf PORT_A ;disable all digits sinks
0073 390F andlw 0x0f
0074 008C movwf TempC ;save sink value in tempC
0075 160C bsf TempC,4 ;preset for lsd sink
0076 0C8C rrf TempC ;determine next sink value
0077 1C03 btfs STATUS,CARRY ;c=1?
0078 118C bcf TempC,3 ;no then reset LSD sink
0079 180C btfsc TempC,0 ;else see if Msd
007A 288C goto UpdateMsd ;yes then do Msd
007B 188C btfsc TempC,1 ;see if 3rdLsd
007C 2887 goto Update3rdLsd ;yes then do 3rd Lsd
007D 190C btfsc TempC,2 ;see if 2nd Lsd
007E 2882 goto Update2ndLsd ;yes then do 2nd lsd

; UpdateLsd
007F 0B11 movf LsdTime,w ;get Lsd in w
0080 390F andlw 0x0f ;
0081 2890 goto DisplayOut ;enable display

; Update2ndLsd
0082 20A1 call Chk2LsdZero ;mds = 0 & 2 lsd 0?
0083 1D03 btfs STATUS,Z ;yes then skip
0084 0E11 swapf LsdTime,w ;get 2nd Lsd in w
0085 390F andlw 0x0f ;mask rest
0086 2890 goto DisplayOut ;enable display

; Update3rdLsd
0087 20A9 call ChkMsdZero ;mds = 0?
0088 1D03 btfs STATUS,Z ;yes then skip
0089 0411 movf MsrdTime,w ;get 3rd Lsd in w
008A 390F andlw 0x0f ;mask low nibble
008B 2890 goto DisplayOut ;enable display

; UpdateMsd
008C 0E10 swapf MsrdTime,w ;get Msd in w
008D 390F andlw 0x0f ;mask rest
008E 1903 btfsc STATUS,Z ;msd != 0 then skip
008F 300A movlw 0x0a

; DisplayOut
0090 2095 call LedTable ;get digit output
0091 0086 movwf PORT_B ;drive leds
0092 080C movf TempC,w ;get sink value in w
0093 0085 movwf PORT_A
0094 0008 return

; LedTable
0095 0782 addwf PCL ;add to PC low
0096 343F retlw B'00111111' ;led drive for 0
0097 3406 retlw B'00000000' ;led drive for 1
0098 345B retlw B'01011011' ;led drive for 2
0099 344F retlw B'01001111' ;led drive for 3
009A 3466 retlw B'01100010' ;led drive for 4
009B 346D retlw B'01101101' ;led drive for 5
009C 347D retlw B'01111011' ;led drive for 6
009D 3407 retlw B'00000011' ;led drive for 7
009E 347F retlw B'01111111' ;led drive for 8
009F 3467 retlw B'01100011' ;led drive for 9
```
Four Channel Digital Voltmeter with Display and Keyboard

00A0 3400 retlw B’00000000’ ; blank led drive
;
; Chk2LsdZero
00A1 20A9 call ChkMsdZero ; msd = 0?
00A2 1D03 btfss STATUS, Z ; yes then skip
00A3 0008 return ; else return
00A4 0E11 swapf LsdTime, w ; get 2nd lsd
00A5 390F andlw 0x0F ; mask of LSD
00A6 1D03 btfss STATUS, Z ; 0? then skip
00A7 0008 return ; else return with 10
00A8 340A retlw .10 ; else return

; ChkMsdZero
00A9 0810 movf MsdTime, w ; get Msd in w
00AA 1D03 btfss STATUS, Z ; = 0? skip
00AB 0008 return ; else return
00AC 340A retlw .10 ; ret with 10

; count equ 26
0026
temp equ 27
0027

H_byte equ 20
0020
L_byte equ 21
0021
R0 equ 22
0022
R1 equ 23
0023
R2 equ 24
0024

B2_BCD bcf STATUS, 0 ; clear the carry bit
00AD 1003

movlw .16
00AF 00A6
movwf count
00B0 01A2
clr R0
00B1 01A3
clr R1
00B2 01A4
clr R2
00B3 0DA1
loop16 rlf L_byte
00B4 0DA0
rlf H_byte
00B5 0DA4
rlf R2
00B6 0DA3
rlf R1
00B7 0DA2
rlf R0
00B8 0BA6

decfsz count
00B9 28B8
goto adjDEC
00BA 3400
RETLW 0
;
00BB 3024
adjDEC movlw R2
00BC 0084
movwf FSR
00BD 20C5
call adjBCD
;
00BE 3023
movlw R1
00BF 0084
movwf FSR
00C0 20C5
call adjBCD
;
00C1 3022
movlw R0
00C2 0084
movwf FSR
00C3 20C5
call adjBCD
;
00C4 28B3
goto loop16
;
00C5 3003
adjBCD movlw 3
00C6 0700
addwf 0, w
00C7 00A7
movwf temp
00C8 19A7
btfsc temp, 3 ; test if result > 7
00C9 0080
movwf 0
00CA 3030
movlw 30
00CB 0700
addwf 0, W
00CC 00A7
movwf temp
Four Channel Digital Voltmeter with Display and Keyboard

btfsc temp,7 ; test if result > 7
movwf 0 ; save as MSD
RETLW 0

; ; end ;
Four Channel Digital Voltmeter with Display and Keyboard

APPENDIX D

MPASM B0.50

; This program is to demonstrate how to multiplex four 7 segment LED
digits and a 4X4 keypad using a PIC16C71, along with 4 channels of A/D.
At start the display reads the A/D value of channel 0. When a key is hit
the A/D value of the corresponding channel's 0 to 3 is displayed.
All key hits > 4 are ignored.
The LEDs are updated every 20msec, the keypad is scanned at a rate of 20 msec.
The RTCC timer is used in internal interrupt mode to generate the 5 msec.

Stan D'Souza 5/8/93

LIST P=16C71, F=INHX8M

; include "picreg.equ"

000C TempC equ 0x0c ; temp general purpose regs
000D TempD equ 0x0d
000E TempE equ 0x0e
0020 PABuf equ 0x20
0021 PBuf equ 0x21
000F Count equ 0x0f ; count
0010 MsdTime equ 0x10 ; most significant Timer
0011 LsdTime equ 0x11 ; least significant Timer
0012 Flag equ 0x12 ; general purpose flag reg
0001 #define keyhit Flag, 0 ; bit 0 -> key-press on
0002 #define DebnceOnFlag, 1 ; bit 1 -> debounce on
0003 #define nOentryFlag, 2 ; no key entry = 0
0004 #define ServKey Flag, 3 ; bit 3 -> service key
0005 #define DOWNFlag, 4 ; bit 4 -> a/d conv. over
0013 debounce equ 0x13 ; debounce counter
0014 NewKey equ 0x14
0016 ADTABLE equ 0x16 ; 4 locations are reserved here
                      ; from Ox16 to Ox19f

002F WBuffer equ 0x2f
002E StatBuffer equ 0x2e
0001 OptionReg equ 1
0002 PCL equ 2

push macro
movwf WBuffer
swapf StatBuffer
movwf STATUS
swapf WBuffer
swapf STATUS
movwf StatBuffer
swapf WBuffer
endm

pop macro
swapf StatBuffer,w ; restore status
movwf STATUS ;
swapf WBuffer,w ; restore W reg
endm

0000 280D goto Start ; skip over interrupt vector

org 0

org 4

; It is always a good practice to save and restore the w reg,
; and the status reg during a interrupt.
push
0004 00AF movwf WBuffer ; save w reg in Buffer
0005 0EAF swapf WBuffer ; swap it

© 1993 Microchip Technology Inc.
Four Channel Digital Voltmeter with Display and Keyboard

0006 0B03  swapf STATUS, w ; get status
0007 00AE  movwf StatBuffer ; save it
0008 204E  call ServiceInterrups
0009 0E2E  swapf StatBuffer, w ; restore status
000A 0083  movwf STATUS ; /  /
000B 0B2E  swapf WBuffer, w ; restore W reg
000C 0009  retfie ;
000D 2038  call InitPorts
000E 20E0  call initAd
000F 2042  call InitTimers
0010 1992  loop
0011 2015  btfsc ServKey ;key service pending
0012 1A12  call ServiceKey ;yes then service
0013 2026  btfsc ADOver ; a/d pending?
0014 2810  call ServiceAD ;yes the service a/d
0015 0814  movf NewKey, w ; get key value
0016 3C03  sublw 3 ; key > 3?
0017 1C03  btfss STATUS, C ; no then skip
0018 0008  return ; else ignore key
0019 3016  movlw ADTABLE ; get top of table
001A 0714  addlw NewKey, w ; add offset
001B 0084  movwf FSR ; init FSR
001C 0800  movf 0, w ; get a/d value
001D 00A1  movwf L_BYTE ;
001E 01A0  clrf H_BYTE ;
001F 2102  call B2_BCD
0020 0824  movf R2, w ; get LSD
0021 0991  movwf LsdTime ; save in LSD
0022 0823  movf R1, w ; get MSD
0023 0090  movwf MsdTime ; save in Msd
0024 1192  bcf ServKey ; reset service flag
0025 0008  return
0026 0808  movf ADCON0, w ; get adcon0
0027 00BC  movwf TempC ; save in temp
0028 3008  movlw B'00001000' ; select next channel
0029 0708  addwf ADCON0, w ; /
002A 1A88  btfsc ADCON0, 5 ; if <= ch3
002B 30C0  movlw B'11000000' ; select ch0
002C 0088  movwf ADCON0 ; now load adres in the table
002D 3016  movlw ADTABLE
002E 0084  movwf FSR ; load FSR with top
002F 0DBC  rlf TempC
0030 0DBC  rlf TempC
0031 00DC  rlf TempC, w ; get in w reg
0032 3903  andlw 3 ; mask off all but last 2
0033 0784  addwf FSR ; add offset to table
0034 0809  movw ADRES, w ; get a/d value
0035 0080  movwf 0 ; load indirectly
0036 1212  bcf ADOver ; clear flag
0037 0008  return ;
0038 1683  bsf STATUS, RPO ; select pg 1
0039 3003  movlw 3 ; make RA0-3 digital I/O
003A 0088  movwf ADCON1 ; /

; InitPorts

© 1993 Microchip Technology Inc. DS00557A-page 21
The clock speed is 4.096MHz. Dividing internal clk. by a 32 prescaler, the rtcc will be incremented every 31.25uS. If rtcc is preloaded with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the end result is that we get a rtcc interrupt every 5msec.

InitTimers

clrft

; ScanKeys, scans the 4x4 keypad matrix and returns a key value in NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
; Debounce for a given keyhit is also taken care of.
; The rate of key scan is 20msec with a 4.096MHz clock.

ScanKeys

crft

; make RA0-4 outputs
clr f TRISA
; make RB0-7 outputs
clr f TRISB
; select page 0
bcf STATUS,RPO
; make all outputs low
bcf PORT_A
; enable MSB digit sink
bsf PORT_A,3

; The clock speed is 4.096MHz. Dividing internal clk. by a 32 prescaler, the rtcc will be incremented every 31.25uS. If rtcc is preloaded with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the end result is that we get a rtcc interrupt every 5msec.

InitTimers

clrft

; make RA0-4 outputs
clr f TRISA
; make RB0-7 outputs
clr f TRISB
; select page 0
bcf STATUS,RPO
; make all outputs low
bcf PORT_A

; enable MSB digit sink
bsf PORT_A,3

; The clock speed is 4.096MHz. Dividing internal clk. by a 32 prescaler, the rtcc will be incremented every 31.25uS. If rtcc is preloaded with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the end result is that we get a rtcc interrupt every 5msec.

InitTimers
Four Channel Digital Voltmeter with Display and Keyboard

006D 1COB btfss INTCON,RB1IF ;flag set?
006E 2865 goto ScanNext ;no then next
006F 1B12 btfsc keyhit ;last key released?
0070 287A goto SKreturn ;no then exit
0071 1412 bsf keyhit ;set new key hit
0072 0086 swapf PORT_B,w ;read port
0073 008E movwf TempE ;save in TempE
0074 207E call GetKeyValue ;get key value 0 - F
0075 0094 movwf NewKey ;save as New key
0076 1492 bsf ServKey ;set service flag
0077 1592 bsf DebnceOn ;set flag
0078 2004 movlw 4 ;load debounce time
0079 0093 movwf Debnce ;save
SKreturn
007A 2081 call RestorePorts ;restore ports
007B 0008 return

NoKey
007C 1012 bcf keyhit ;clr flag
007D 287A goto SKreturn

;GetKeyValue gets the key as per the following layout
;
;                 Coll  Col2  Col3  Col4
;                  (RB3) (RB4) (RB5) (RB6) (RB7)
;Row1 (RB4) 0 1 2 3
;Row2 (RB5) 4 5 6 7
;Row3 (RB6) 8 9 A B
;Row4 (RB7) C D E F

007E 018C clrf TempC
007F 1D8D btfss TempD,3 ;first column
0080 2888 goto RowValEnd
0081 0A8C incf TempC
0082 1D0D btfss TempD,2 ;second col.
0083 2888 goto RowValEnd
0084 0A8C incf TempC
0085 1C0D btfss TempD,1 ;third col.
0086 2888 goto RowValEnd
0087 0A8C incf TempC ;last col.

RowValEnd
0088 1COE btfss TempE,0 ;top row?
0089 2892 goto GetValCom ;yes then get 0,1,2,3
008A 1CE1 btfss TempE,1 ;2nd row?
008B 2891 goto Get4567 ;yes the get 4,5,6,7
008C 1D0E btfss TempE,2 ;3rd row?
008D 288F goto Get89ab ;yes then get 8,9,a,b

Getcdef
008E 150C bsf TempC,2 ;set msb bits
Get89ab
008F 158C bsf TempC,3 ;
0090 2892 goto GetValCom ;do common part
Get4567
0091 150C bsf TempC,2

GetValCom
0092 008C movf TempC,w
0093 0782 addwf PCL
0094 3400 retlw 0
0095 3401 retlw 1
0096 3402 retlw 2
0097 3403 retlw 3
0098 3404 retlw 4
0099 3405 retlw 5
009A 3406 retlw 6
009B 3407 retlw 7

© 1993 Microchip Technology Inc.
Four Channel Digital Voltmeter with Display and Keyboard

009C 3408  retlw  8
009D 3409  retlw  9
009E 340A  retlw  0a
009F 340B  retlw  0b
00A0 340C  retlw  0c
00A1 340D  retlw  0d
00A2 340E  retlw  0e
00A3 340F  retlw  0f

;SavePorts, saves the porta and portb condition during a key scan operation.
SavePorts

00A4 0805  movf  PORT_A,w  ;Get sink value
00A5 00A0  movwf  PABuf    ;save in buffer
00A6 0185  clrf  PORT_A    ;disable all sinks
00A7 0806  movf  PORT_B,w  ;get port b
00A8 00A1  movwf  PBBuf    ;save in buffer
00A9 30FF  movlw  0xff    ;make all high
00AA 0086  movwf  PORT_B    ;on port b
00AB 1683  bsf  STATUS,RPO ;select page 1
00AC 1381  bcf  OptionReg,7 ;enable pull ups
00AD 30FO  movlw  B'11110000' ;port b hi nibble inputs
00AE 0086  movwf  TRISB ;lo nibble outputs
00AF 1283  bcf  STATUS,RPO ;page 0
00BA 0008  return

;RestorePorts, restores the condition of porta and portb after a key scan operation.
RestorePorts

00BB 0821  movf  PBBuf,w  ;get port b
00B2 0086  movwf  PORT_B  ;get port a value
00B3 0820  movf  PABuf,w  ;select page 1
00B4 0085  movwf  PORT_A  ;disable pull ups
00B5 1683  bsf  STATUS,RPO ;make port a outputs
00B6 1381  bcf  OptionReg,7 ;as well as PORTB
00B7 0185  clrf  TRISA  ;page 0
00B8 0186  clrf  TRISB  ;present sink value in w
00B9 1283  bcf  STATUS,RPO ;disable all digits sinks
00BA 0008  return

;UpdateDisplay

00BB 0805  movf  PORT_A,w  ;present sink value in w
00BC 0185  clrf  PORT_A  ;save sink value in tempC
00BD 390F  movlw  0x0f  ;disable all digits sinks
00BE 008C  movwf  TempC  ;preset for lsd sink
00BF 160C  bsf  TempC,4  ;determine next sink value
00C0 0C8C  rrf  TempC  ;r:=7
00C1 1C03  btfss  STATUS,CARRY  ;no then reset LSD sink
00C2 118C  bcf  TempC,3  ;else see if Mad
00C3 180C  btfsc  TempC,0  ;get Lsd in w
00C4 28D2  goto  UpdateLsd  ;get 2nd Lsd in w
00C5 188C  btfsc  Tempc,1  ;mask low nibble
00C6 28CF  goto  Update3rdLsd  ;enable display
00C7 190C  btfsc  Tempc,2  ;get 3rd Lsd in w
00C8 28CC  goto  Update2ndLsd  ;enable display
00C9 0811  movf  LsdTime,w  ;get Lsd in w
00CA 390F  andlw  0x0f  ; / 
00CB 28D4  goto  DisplayOut  ;
00CC 0E11  swapf  LsdTime,w  ;get 2nd Lsd in w
00CD 390F  andlw  0x0f  ;mask rest
00CE 28D4  goto  DisplayOut  ;enable display
00CF 0B10  movf  MsdTime,w  ;get 3rd Lsd in w
00D0 390F  andlw  0x0f  ;mask low nibble
00D1 28D4  goto  DisplayOut  ;enable display
00D2 0E10  swapf  MsdTime,w  ;get Mad in w

DS00557A-page 24  © 1993 Microchip Technology Inc.

3-180
Four Channel Digital Voltmeter with Display and Keyboard

```
0003 390F  andlw   0x0f ;mask rest
0004 20D9  call     LedTable ;get digit output
0005 0086  movwf     PORT_B ;drives leds
0006 080C  movf      TempC,w ;get sink value in w
0007 0085  movwf     PORT_A
0008 0008  return

; display output
0009 0782  addwf   PCL ;add to PC low
000A 343F  retlw   B'00111111' ;led drive for 0
000B 3406  retlw   B'00000110' ;led drive for 1
000C 345B  retlw   B'01011011' ;led drive for 2
000D 344F  retlw   B'01001111' ;led drive for 3
000E 3466  retlw   B'01100110' ;led drive for 4
000F 346D  retlw   B'01101101' ;led drive for 5
0010 347D  retlw   B'01111101' ;led drive for 6
0011 3407  retlw   B'00000111' ;led drive for 7
0012 347F  retlw   B'01111111' ;led drive for 8
0013 3467  retlw   B'01100111' ;led drive for 9
0014 3477  retlw   B'01111011' ;led drive for A
0015 347C  retlw   B'01111100' ;led drive for B
0016 3439  retlw   B'00111101' ;led drive for C
0017 345E  retlw   B'01011110' ;led drive for D
0018 3479  retlw   B'01111001' ;led drive for E
0019 3471  retlw   B'01110001' ;led drive for F

; init ad
001A 30C0  movlw   B'11000000' ;internal rc for tad
001B 0088  movwf   ADCONO ;/note that adcon1 is set in initports
001C 0008  return

; sample ad
001D 20A4  call     SavePorts
001E 20F4  call     DoAd ;do a ad conversion
001F 1908  btfsc   ADCON0, GO ;ad done?
0020 28E8  goto    AdDone ;no then loop
0021 1612  bsf     ADOver ;set a/d over flag
0022 20B1  call     RestorePorts ;restore ports
0023 0008  return

; do ad
0024 0186  clrf    PORT_B ;turn off leds
0025 1883  bsf     STATUS,RP0 ;select pg 1
0026 300F  movlw   0x0f ;make port a hi-Z
0027 0085  movwf   TRISA ;/
0028 1283  bcf     STATUS,RP0 ;select pg 0
0029 1408  bsf     ADCONO, ADON ;start a/d
002A 307D  movlw   .125
002B 20FE  call     Wait
002C 1508  bsf     ADCONO, GO ;start conversion
002D 0008  return

; wait
002E 008C  movwf   TempC ;store in temp
002F 0B8C  decfsz   TempC Next
0030 28FF  goto    Next
0031 0008  return

; count
0032 count equ 26
```
Four Channel Digital Voltmeter with Display and Keyboard

0027    temp     equ  27
          ;
0020    H_byte    equ  20
0021    L_byte    equ  21
0022    R0        equ  22
0023    R1        equ  23
0024    R2        equ  24
          ;
0102    1003    R2_BCD  bcf STATUS,0
0103    3010    movlw .16
0104    00A6    movwf count
0105    01A2    clrf R0
0106    01A3    clrf R1
0107    01A4    clrf R2
0108    0DA1    loop16 rlf L_byte
0109    0A00    rlf H_byte
010A    0DA4    rlf R2
010B    0DA3    rlf R1
010C    0DA2    rlf R0
          ;
010D    0BA6    decfsz count
010E    2910    goto adjDEC
010F    3400    RETLW 0
          ;
0110    3024    adjDEC movlw R2
0111    0084    movwf FSR
0112    211A    call adjBCD
          ;
0113    3023    movlw R1
0114    0084    movwf FSR
0115    211A    call adjBCD
          ;
0116    3022    movlw R0
0117    0084    movwf FSR
0118    211A    call adjBCD
          ;
0119    2908    goto loop16
          ;
011A    3003    adjBCD movlw 3
011B    0700    addwf 0,W
011C    00A7    movwf temp
011D    19A7    btfsc temp,3 ; test if result > 7
011E    0080    movwf 0
011F    3030    movlw 30
0120    0700    addwf 0,W
0121    00A7    movwf temp
0122    18A7    btfsc temp,7 ; test if result > 7
0123    0080    movwf 0 ; save as MSD
0124    3400    RETLW 0
          ;
          ;
          ;
          ;
end
## SECTION 4
### PIC17CXX APPLICATION NOTES

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saving and Restoring Status on Interrupt - AN534</td>
<td>4-1</td>
</tr>
<tr>
<td>Implementing Table Read and Write - AN548</td>
<td>4-3</td>
</tr>
<tr>
<td>Frequency and Resolution Options for PWM Outputs - AN539</td>
<td>4-7</td>
</tr>
<tr>
<td>Using PWM to Generate Analog Output - AN538</td>
<td>4-15</td>
</tr>
<tr>
<td>Using the PWM - AN564</td>
<td>4-17</td>
</tr>
<tr>
<td>Using the Capture Module - AN545</td>
<td>4-29</td>
</tr>
<tr>
<td>Serial Port Utilities - AN547</td>
<td>4-61</td>
</tr>
<tr>
<td>Utility Math Routines - AN544</td>
<td>4-71</td>
</tr>
<tr>
<td>Implementing IIR Digital Filters - AN540</td>
<td>4-121</td>
</tr>
<tr>
<td>Implementation of Fast Fourier Transforms - AN542</td>
<td>4-139</td>
</tr>
<tr>
<td>Tone Generation - AN543</td>
<td>4-161</td>
</tr>
<tr>
<td>Servo Control of a DC Brush Motor - AN532</td>
<td>4-197</td>
</tr>
</tbody>
</table>
INTRODUCTION

The PIC17C42 has a 16 level deep hardware stack. The program counter is pushed into this stack on interrupts and subroutine calls. However, other key registers are not saved in the stack. Registers such as W, ALUSTA (which has carry, zero and other flag bits) and the bank select register (BSR) must be saved in an interrupt service routine. The following macros, PUSH and POP implement a parameter stack in data memory to accomplish this.

The indirect addressing register FSRO is used to implement this parameter stack. It is assumed that FSRO and its control bits are not used or modified elsewhere. The stack pointer (FSRO) is initialized at the highest RAM location (FFh).

Main_prog

```asm
SETF FSRO ;Initialize and dedicate FSRO as stack pointer
BCF ALUSTA,FSO 
BCF ALUSTA,FS1 ;Set-up FSRO for auto-dec

PUSH MACRO
BCF ALUSTA,FSO ;Set-up FSRO for auto-dec
MOVFP ALUSTA,INDO ;Save ALUSTA first
MOVFP BSR,INDO ;
MOVFP W,INDO ;
MOVFP RAM_x, INDO ;Now save general
MOVFP RAM_y, INDO ;Purpose registers
ENDM

POP MACRO
BSF ALUSTA,FS0 ;Set-up for auto-inc
INCF FSR0 ;
MOVFF INDO, RAM_y ;
MOVFF INDO, RAM_x ;
MOVFF INDO,W ;
MOVFF INDO,BSR ;
MOVFF INDO,ALUSTA ;restore ALUSTA last
DECFF FSR0 ;Adjust stack pointer
ENDM

interrupt_routine

PUSH ;save registers

;main body of interrupt service

POP ;restore status
RETFIE ;return
```
While the macros are quite self-explanatory, the user should note a few subtle points.

1. **MOVFP** instruction does not affect status flags while **MOVPF** does.

2. **MOVFP** and **MOVPF** are used such that any register can be saved and restored. Note that register being saved or restored has address f (which can be 00h to FFh) and other address is INDO (indirect), therefore can be any address.

3. FSR auto-increments or auto-decrements after the operation ("post"). Therefore in the POP macro pre-increment is simulated.

Using this scheme, interrupts and subroutine calls can be nested, since the stack will grow and shrink. The stack can be used to pass parameters to subroutines.

**Author:** Stanley D'Souza  
*Logic Products Division*
INTRODUCTION

This application brief discusses how to read data from program memory to data memory and write data from data memory to program memory.

RETLW K Instruction

As in all PIC17CXX family parts, the simplest method used to retrieve data from program memory to data memory is to use the RETLW K instruction. For example:

```assembly
; simple program to transfer ; table values to PortB
Main
  movlw 5,W ;load offset
  call SimpleTableRead
  movwf PortB ;output to PortB

SimpleTableRead
  .
  .
  .

Table retlw 0 ;return a known ;table value based ;on the OFFSET.
  .
  .
  .
  retlw 10
```

In the example above, OFFSET is loaded with the required offset to the Table and the subroutine SimpleTableRead is called. The table value is returned in the W register. In this manner program memory can be transferred to data memory.

Table Read Instruction

The PIC17C42 has an expanded instruction set which includes the TABLRD and TLRD instructions. These instructions are specifically constructed to transfer data from program memory to data memory.

The instruction syntax is: TABLRD t,i,f.

The sequence in which this instruction is executed is as follows:

- if \( t = 1 \) then the high byte of the table latch (TBLATH) is loaded in the file register \( f \).
- else (if \( t = 0 \)) the low byte of the table latch (TBLATL) is loaded in the file register \( f \).
- next, the 16 bit data pointed to by the table pointer (TBLPTR) is loaded into the table latch.
- lastly, if \( i = 1 \) the table pointer (TBLPTR) is incremented.

Note: The first time this instruction is executed in a sequence, the table latch will not be initialized, hence an unknown value will be loaded in the file register. This is not a problem if the user overwrites the same \( f \) register in the next subsequent instruction.

The instruction syntax is: TLRD t,f.

The sequence in which this instruction is executed is as follows:

- if \( t = 1 \) then the high byte of the table latch (TBLATH) is loaded in the file register \( f \).
- else (if \( t = 0 \)) the low byte of the table latch (TBLATL) is loaded in the file register \( f \).

FIGURE 1 - TABLE READ
Implementing Table Read and Write in PIC17C42

Read In Line

A simple method of transferring data from program memory to data memory is to use the tablrd and tlrd instruction in sequence as shown in the example below:

;transfer 6 bytes of data in program memory at 0x500, to data memory at 0x80:

```
;Read In Line
movlw 05  ;load table pointer with 0x500
movwf TBLPTRH ;
clr TBLPTRL ;
tablrd 0,1,0x80  ;get 16 bit value in table latch.
    ;low byte (1st) \( \text{H} 80 \)
tlr 0,0x80  ;
tablrd 1,1,0x81  ;high byte (2nd) \( \text{H} 81 \)
tlr 0,0x82  ;
tablrd 1,1,0x83  ;4th byte \( \text{H} 83 \)
tlr 0,0x84  ;
tablrd 1,1,0x85  ;6th byte \( \text{H} 85 \)
```

Reading a Block of Data

In instances where a block of N bytes needs to be transferred from program memory to data memory, the tablrd and tlrd instruction need to be included in a loop which checks for N transfers.

;transfer 'COUNT' bytes (even values only) of data at program memory 'MESSAGE' to data memory at: 'RAM_BUFFER'

```
;Read Block
movlw high MESSAGE  ;load table pointer
movpf W,TBLPTRH ;
movlw low MESSAGE ;
movpf W,TBLPTRL ;
bcf ALUSTA,5 ;enable post auto increment of FSRO
movlw RAM_BUFFER ;initialize FSRO to RAM_BUFFER
movfp W,FSRO ;
movlw COUNT/2 ;initialize count
tablrd 1,1,RAM_BUFFER ;initialize latch

ReadBlockLoop
    tlrd 1,0x00  ;do indirect read of high byte
tablrd 0,1,0x00  ;do indirect read of low byte
decfsz W ;check if count = 0
goto ReadBlockLoop ;no then do next
return ;else end of transfer.
```

<table>
<thead>
<tr>
<th>Program</th>
<th>Code Size</th>
<th>Transfer Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Table Read</td>
<td>N + 3</td>
<td>6 cycles/byte</td>
</tr>
<tr>
<td>Read In-Line</td>
<td>4 + N + N/2</td>
<td>1.5 cycles/byte</td>
</tr>
<tr>
<td>Read Block (using loop)</td>
<td>14 + N/2</td>
<td>3 cycles/byte</td>
</tr>
</tbody>
</table>

\( N = \text{Number of bytes to transfer} \)

<table>
<thead>
<tr>
<th>N = 10</th>
<th>13</th>
<th>19</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>N = 20</td>
<td>23</td>
<td>34</td>
<td>24</td>
</tr>
</tbody>
</table>

Conclusion:

In cases where the number of bytes to be transferred is small, the Read In-Line offers small code size for fast transfer rate. However, as the number of bytes to be transferred increases, the Read Block offers optimum code size for a decent transfer rate.
Implementing Table Read and Write in PIC17C42

Table Write Instruction

The PIC17C42 has a TABLWT and a TLWT instruction which transfer data from data memory to program memory. Note in cases where the table pointer points to internal EPROM, the table write instruction will try to program the EPROM, hence the programming voltage must be present on the VPP line to successfully program the part.

The instruction syntax is: TABLWT t, i, f.

The sequence in which this instruction is executed is as follows:

- if t = 1 then the file register f is loaded to the high byte of the table latch (TBLATH).
- else (if t = 0) the file register f is loaded to the low byte of the table latch (TBLATL).
- next, the 16 bit data in the table latch is transferred to the program memory pointed to by the table pointer (TBLPTR).
- lastly, if i = 1 the table pointer (TBLPTR) is incremented.

The instruction syntax is: TLWT t, f.

The sequence in which this instruction is executed is as follows:

- if t = 1 then the file register f is loaded to the high byte of the table latch (TBLATH).
- else (if t = 0) the file register f is loaded to the low byte of the table latch (TBLATL).

Write in Line

A simple method of transferring data from data memory to program memory is to use the tablwt and tlwt instruction in sequence as shown in the example below:

; transfer 6 bytes of data in data memory at 0x80, to program memory at 0x5000:

ReadInLine
movlw 50 ; load table pointer with 0x5000
movwf TBLPTRH ;
clf TRIPTRL ;
tlwt 1,0x80 ; high byte & table latch.
tablwt 0,1,0x81 ; low byte & table latch;
; latch & prog. mem.
tlwt 1,0x82 ; 3rd and 4th byte & prog. mem.
tablwt 0,1,0x83 ;
tlwt 1,0x84 ; 5th and 6th byte & prog. mem.
tablwt 0,1,0x85 ;

Writing a Block of Data

In instances where a block of N bytes needs to be transferred from data memory to program memory, the tablwt and tlwt instruction need to be included in a loop which checks for N transfers.

; transfer 'COUNT' bytes (even values only) of data at 'RAM BUFFER' to program memory at 'MESSAGE'

WriteBlock
movlw high MESSAGE ; load table pointer
movpf W,TBLPTRH ;
movlw low MESSAGE ;
movpf W,TBLPTRL ;
bcf ALUSTA,5 ; enable post auto
; increment of F SR0
movlw RAM_BUFFER ; initialize F SR0 to
; RAM_BUFFER
movfp W,FSRO ;
movlw COUNT/2 ; initialize count
WriteBlockLoop
; high byte & table
; latch
; low byte & table
; latch;
; table latch & prog.
; mem.
; check if count = 0
; no then do next
; else end of transfer.

Author: Stanley D’Souza
Logic Products Division
INTRODUCTION

The PIC17C42 is equipped with two high frequency Pulse Width Modulation (PWM) outputs. In a pulse width modulated signal the period of the signal is (usually) kept fixed, while the duty cycle is varied. In this application note, we will discuss options in selecting their frequency and resolution.

This application brief assumes that internal clock is used for the time-base, which is typically the preferred set-up. Also, throughout this application brief, PWM1 output is used in examples, timer1 is assumed to be the time-base.

Definition of terms:

- **Period** of a PWM output is the duration after which the PWM pattern will repeat itself.
- **Frequency** of a PWM output is \( 1 / \text{Period} \).
- **Resolution** of a PWM output is the granularity with which the duty cycle can be modulated.

In the case of the PIC17C42, when using PWM1 with timer1 as time-base the:

\[
\text{PWM1 period} = [(\text{PR1}) + 1] \times 4\text{tosc}
\]

\[
\text{PWM1 duty cycle} = (\text{DC}1) \times \text{tosc}
\]

where

- \( \text{PR1} \) = period register for timer1
- \( \text{DC}1 \) = PW1DCH, PW2DCL concatenated (10-bit value)
- \( \text{tosc} \) = oscillator period

At 16 MHz oscillator frequency, \( \text{tosc} = 62.5 \text{ ns} \). The user can control the frequency of the PWM output by altering the 'period' value of the time-base. For example, if period is chosen to be 100 osc (PR1 = 18h), then PWM frequency is \( 1(100 \times 62.5) \text{ ns} = 160 \text{ KHz} \). Note however that duty cycle resolution is a little less than 7-bits.

Useful and Common PWM Modes

While a variety of period values can be selected, the following modes would be most commonly used:

10-Bit Mode: In this mode PWM duty cycle has full 10-bit resolution (maximum offered by the PIC17C42). The period register PR1 is set at FFh. PWM period is \( 1024\text{tosc} = 64 \mu\text{s} \). PWM frequency is 15.625 KHz. The user must write both PW1DCH and PW1DCL to update PWM output. See Appendix A for an example that code modules 10-bit resolution PWM output (PWM10.LST).

8-Bit Hi-Resolution Mode: In this mode, the user has only an 8-bit quantity to write to the duty-cycle register. Period register is set at 3Fh (63 decimal), such that PWM period is 256 tosc. To write the 8-bit duty-cycle value, first the 8-bit is right shifted two bits. The upper six bits are written to PW1DCH and the lower two bits are written to PW1DCL as follows:

\[ \text{8-bit duty-cycle value is in W reg} \]

\[ \text{CLR} \text{F TEMP} ; \]

\[ \text{RRCF WREG} ; \]

\[ \text{RRCF TEMP} ; \]

\[ \text{RRCF WREG} ; \]

\[ \text{RRCF TEMP} ; \text{Shift right twice} \]

\[ \text{ANDLW } 00111111b ; \text{Mask off two-high bits} \]

\[ \text{MOVFP WREG, PW1DCH} ; \text{Write duty-cycle values} \]

\[ \text{MOVFP TEMP, PW1DCL} ; \]

Note that in 8-bit, hi-resolution mode, maximum PWM frequency is attained. For example, at 16 MHz clock, PWM period = 256 tosc = 16 \mu\text{s}; PWM frequency = 62.5 KHz. See appendix B for an example code that generates 8-bit low high resolution PWM output (PWM8Hi.LST).

FIGURE 1 - PWM OUTPUT

![PWM Output Diagram]

\[ \text{PWM frequency = } 1 / \text{period} \]
PWM Frequency and Resolution

FIGURE 2 - VARIOUS PWM MODES

<table>
<thead>
<tr>
<th>DC9</th>
<th>DC2</th>
<th>DC9</th>
<th>DC2</th>
<th>DC7</th>
<th>DC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWIDCH</td>
<td></td>
<td>PWIDCH</td>
<td></td>
<td>PWIDCH</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td></td>
<td>0 0</td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

Note: DC registers suited for 16 bit computation; utilizing 10 MSB only.

8-Bit Low Resolution Mode

In this mode, the user still has only an 8-bit quantity to write to duty cycle register. However, the desired frequency of the PWM output is less, due to the nature of the application. For example, if the PWM output is being used to drive a motor through a power stage, the power transistors (or devices) due to their switching time will prefer PWM frequency not to exceed certain frequency. In the previous section, we derived an 8-bit resolution PWM output at 62.5 KHz.

To attain a low-resolution PWM output, the PW1DCL is always kept at zero. The 8-bit value is written to PW1DCH. The period (PR1) is set at FFh, i.e. 256 Tcy equals 1024 tosc (15.625 KHz). See Appendix C for an example code that produces 8-bit low resolution PWM output (PWM8LO.LST).

Choosing Resolution and Frequency of PWM Output

Actually, the resolution and the frequency of the PWM output is selectable within certain limits. The user will need to first define the requirements based on the application. There may be an upper limit to the frequency if the PWM is being used to drive motors. On the other hand, if the PWM is being filtered to generate an analog signal, higher frequency may be desirable. In any case, the lowest frequency achievable (using internal clock for the timer) is (OSC freq/1024). At 16 MHz oscillator input, the lowest PWM frequency possible is 15.625 KHz. At resolutions less than 10-bit higher frequencies are possible (see Figure 3). For example if 7-bit resolution is chosen, then the PWM frequencies can be 15-625 KHz, 31.25 KHz, 62.5 KHz or 125 KHz. The reader will note that its how the 7-bits are placed within the 10-bit possible duty cycle value.

Conversely, if a certain frequency is desired, such as 44 KHz, then referring to Figure 1, resolution can be 8.5-bit or 7.5-bit or 6.5-bit etc.

Summary

The frequency and resolution of the PWM outputs of the PIC17C42 can be traded off against each other to best suit the application. The oscillator frequency can also be varied to adjust PWM frequency, if necessary. External clock should be used as timer time-base to generate very low frequency PWM output.

Authors: Stanley D'Souza  
Sumit Mitra  
Logic Products Division

FIGURE 3 - PWM FREQ VS RESOLUTION

PWM frequency vs resolution  
for osc freq = 16 MHz  
Timer time-base = internal clock

At a given oscillator frequency (16 MHz in this figure),  
a family of curves represents the PWM freq/resolution combinations possible.
APPENDIX A: PWM10.LST

MPASM B0.54
PULSE WIDTH MODULATION 10 BIT RESOLUTION

TITLE "PULSE WIDTH MODULATION 10 BIT RESOLUTION"
LIST P=17C42, C=8C, T=ON

#include "17c42.h"

PWM_HI equ 0x21
PWM_LO equ 0x20
TEMP equ 0x22

;The user would generate a 16 bit value which is saved in r
;locations PWM_HI and PWM_LO byte. In 10 bit mode, the prog
;transfers these values directly to the Duty Cycle (DC) reg
;generate the required 10 bit PWM.
;
;This is a short program to demonstrate how to generate PWM
;10 bit resolution. Since a 10MHz crystal was used in the t
;The max. period = 1024x100ns = 102.4 us or 9.8 KHz. This p
;keeps the period constant and varies the duty cycle (which
;to the most significant 10 bits of the 16 bit value PWM_LO
;This program is interrupt driven, i.e. the update to the D
;r is done in the rtcc interrupt, which then enables the pwm
;The period update is done during the pwm interrupt. The pw
;ramps up from 0% to 100% duty cycle and then repeats. The
;swep takes approx. 52 secs.
;
ORG 0
goto start

ORG 0x10 ;vector for rtcc interrupt
rtcc_int
goto service_rtcc ;service rtcc

ORG 0x0020 ;vector for pwm interrupt
pwm_int
goto service_pwm ;service pwm only

ORG 0x0030

;initialize internal hardware to generate the output
;for 10 bit resolution pwm.
init_pwm10

movlw 0x058
movb 2
clr tml1 ;clear timer 1
rused to "drive" pwm1
movb 3
set pr1 ;set period=9.8 khz
movf PWM_HI,pwldch ;load duty cyl. hi byte
movf PWM_LO,pwldcl ;load duty cycle lo byte
movfb tcon1 ;tmrl inc. internally
ras 8 bit counter
movlw 0x00010001B ;start tmrl and
movfb wreg,tcon2 ;enable pwm2
movb 1
clr p1e ;clr all int. enables
clr p1r ;clear all interrupts
bsf peie ;except peripheral int.
retfie
PWM Frequency and Resolution

;everytime a new value is written to the PWM_HI, PWM_LO reg
;tmrl interrupts is enabled. The DC value are written just
;the "pwm interrupt" is enabled. Here the new period regist
;updated. In this example, period is kept constant at 0xff

service_pwm

    ;if the period changed, write new value here.
    movlb 2 ;select bank 2
    setf pr1 ;period <- 0xff
    movlb 1 ;disable tmrl int
    bcf _tmiie ;
    retfie

;This part of the program is basically used to simulate a
;which would be used to drive the pwm output.
;
;the rtcc is set up to interrupt every 52 ms.
init_rtcc

    movlw 00100000b ;set up rtcc timer
    movfp wreg,rtcsta ;
    clrf rtccl ;clear rtcc
    clrf rtcch ; /
    movlw 08x80 ;start pwm at 50%
    movwf PWM_HI ; /
    clrf PWM_LO ; /
    bsf _rtcle ;enable rtcc int.
    return

;Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
;Only the 10 most significant bits are incremented. Once th
;
;service_rtcc

    bcf _rtcir ;reset int flag
    ;do a pseudo inc of the 10 bit PWM_HI, PWM_LO.
    bcf _carry ;clear carry
    movlw 01000000b ;load lab for 10 bit
    addwf PWM_LO,1 ;add to LSB
    btfsc _carry ;carry?
    incf PWM_HI ;yes then inc PWM_HI
    now load the values into the Duty Cycle registers

    movlb 3 ;bank 3
    movfp PWM_LO,pwlcl ;load lo value
    movfp PWM_HI,pwlch ;load hi value
    movlb 4
    bsf _tmiie ;enable tmrl int

;start

    bsf _glintd ;disable interrupts
    call init_rtcc ;initialize the RTCC tmr
    ;for test purposes
    call init_pwm10 ;initialize pwm
    loop goto loop ;spin wheels

END

Errors : 0
Warnings : 0

© 1993 Microchip Technology Inc.
PWM Frequency and Resolution

Appendix B: PWM8Hi.LST

TITLE "PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION"
LIST P=17C42, C=80, T=ON

include "17c42.h"

PWM_HI equ 0x21
PWM_LO equ 0x20
TEMP equ 0x22

;The user would generate a 16 bit value which is saved in registers PWM_HI and PWM_LO byte. In 8 bit hi-res mode, the
;transfers the 8 bit values to the lo Duty Cycle (DC) register to generate the required 8 bit hi-res PWM.

;This is a short program to demonstrate how to generate PWM 8 bit resolution. Since a 10Mhz crystal was used in the test
;the max. period = 256x100ns = 25.6us or 39KHz. This program keeps the period constant and varies the duty cycle (which is the most significant 10 bits of the 16 bit value PWM_LO).
;This program is interrupt driven, i.e. the update to the Duty Cycle register is done in the rtc clock interrupt, which then enables the pwm.
;The period update is done during the pwm interrupt. The pwm period ramps up from 0% to 100% duty cycle and then repeats. The sweep takes approx. 13.3 secs.

org 0x010
org 0x020
org 0x030

org 0

goto start

rtcc_int

org 0x054

goto service_rtcc

org 0x062

org 0x070

pwm_int

org 0x086

org 0x094

 initialize internal hardware to generate the pwm output

 init_pwm8hi

org 0x0a2

movlw 2

cirf tmrl

org 0x0b0

movlw 62

cirf prl

org 0x0b8

movlw 3

cirf TEMP

org 0x0c6

movwf PWM_HI

rcrf wreg

org 0x0d4

movf PWM_HI

rcrf TEMP

org 0x0e2

movf PWM_HI

rcrf wreg

org 0x0f0

movf PWM_HI

rcrf TEMP

org 0x0f8

movlw b'00111111'

org 0x106

movwf W

org 0x114

movf TEMP

rcrf wreg

org 0x122

movf tcon1

rcrf wreg

org 0x130

movlw b'00010001'

org 0x138

movlw 1

© 1993 Microchip Technology Inc.
PWM Frequency and Resolution

0042 2917      clr pie  ;clr all int. enables
0043 2916      clr pir  ;clear all interrupts
0044 8307      bsf _peie  ;except peripheral int.
0045 0005      retfie

; everytime a new value is written to the PWM_HI, PWM_LO reg
; tmrl interrupts is enabled. The DC value are written just
; the "pwm interrupt" is enabled. Here the new period regist
; updated. In this example, period is kept constant at 62 Tc
; service_pwm
0046 B802      movlb 2   ;select bank 2
0047 B062      movlw 62   ;period = 62 Tcyl.
0048 0114      movwf prl  ;
0049 B801      movlb 1   ;disable tmrl int
004A BC17      bcf _tmile ;
004B 0005      retfie

; This part of the program is basically used to simulate a
; which would be used to drive the pwm output.
; the rtcc is set up to interrupt every 52 ms.
; init_rtcc
004C B200      movlw b'00100000' ;set up rtcc timer
004D 650A      movfp wreg,rtcsta  ;
004E 290B      clrf rtccl  ;clear rtcc
004F 290C      clrf rtcch  ;
0050 B031      movlw 31   ;init pwm at 50%
0051 0121      movwf PWM_HI  ;save in high
0052 8107      bsf _rtcle  ;enable rtcc int.
0053 0002      return

; Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
; Only the 8 most significant bits are incremented.
; service_rtcc
0054 BD07      bcf _rtcrl  ;reset int flag
0055 1521      incf PWM_HI  ;do a pseudo inc of the 8 bit PWM_HI.
0056 8801      movlb 3   ;temp = mask for pwldcl
0057 2922      clrf TEMP  ;TEMP - mask for pwldcl
0058 6A21      movfp PWM_HI,wreg  ;get duty cyl. hi byte
0059 190A      rrcf wreg  ;rotate hi through carry
005A 1902      rrcf TEMP  ;rotate into lo byte
005B 190A      rrcf wreg  ;repeat for 2nd lsb
005C 1922      rrcf TEMP  ;
005D B53F      andlw b'00111111' ;mask hi bits
005E 4A12      movfp wreg,pwldch  ;save in high
005F 7022      movfp TEMP,pwldcl  ;save in low
0060 B801      movlb 1
0061 8417      bsf _tmile  ;encode tmrl int
0062 0005      retfie

; start
0063 8406      bsf _glintd  ;disable interrupts
0064 E04C      call init_rtcc  ;initialize the RTCC tmr
0065 E030      call init_pwm8hi  ;initialize 8 bit pwm
0066 C066      loop goto loop  ;spin wheels.

END

Errors : 0
Warnings : 0
Appendix C: PWM8LO.LST

PULSE WIDTH MODULATION 8 BIT LOW RESOLUTION

include "17c42.h"

PWM_HI equ 0x21
PWM_LO equ 0x20
TEMP equ Ox22

;The user would generate a 16 bit value which is saved in registers PWM_HI and PWM_LO byte. In 8 bit low-res mode, the
;transfers the 8 hi-byte value directly to the PWLDCR register.

;This is a short program to demonstrate how to generate PWM
;8 bit low resolution. Since a 5.068MHz crystal was used in
;The max. period = 1024×100ns = 102.4 μS or 9.8 KHz. This program keeps the period constant and varies the duty cycle (which
;to the most significant 8 bits of the 16 bit value PWM_LO &
;This program is interrupt driven, i.e. the update to the DTP is done in the rtc interrupt, which then enables the pwm
;The period update is done during the pwm interrupt. The pwm
;ramps up from 0% to 100% duty cycle and then repeats. The
;sweep takes approx. 52 secs.

ORG

;initialize internal hardware to generate the output
;for 8 bit low resolution pwm
init_pwm8lo

0030 B802
0031 2910
0032 2B14
0033 B003
0034 7221
0035 2910
0036 2916
0037 B011
0038 4A17
0039 B001
003A 2917
003B 2916
003C 8307
003D 0005

;
; everytime a new value is written to the PWM_HI, PWM_LO register, the "pwm interrupt" is enabled. Here the new period register is updated. In this example, period is kept constant at 0xff

; service_pwm

; if the period changed, write new value here.

movlb 2 ; select bank 2
setf prl ; period <- 0xff
movlb 1 ; disable tmrl int
bcf _tmlie ;
retfie

; This part of the program is basically used to simulate a
; which would be used to drive the pwm output.
;
; the rtcc is set up to interrupt every 52 ms.

init_rtcc

movlw b'00100000' ; set up rtcc timer
movf pwm8lo, rtcsta
clr rtcc1 ; clear rtcc
movlw Ox BO ; begin PWM at 50% dc
movf PWM_HI ;
clr PWM_LO ;
bsf _rtcie ; enable rtcc int.
return

; Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremented.
; Only the 8 most significant bits are incremented.

service_rtcc

bcf _rtcir ; reset int flag
incf PWM_HI ; do a inc of the 8 bit PWM_HI
movb 3
movfp PWM_HI,pwldch ; load hi byte
movb 1
bsf _tmlie ; enable tmrl int
retfie

; start

bsf _glintd ; disable interrupts
call init_rtcc ; initialize the RTCC timer
for test purposes

call init_pwm8lo ; initialize pwm

loop goto loop ; spin wheels.

END

Errors : 0
Warnings : 0
Pulse Width Modulated (PWM) waveforms which are basically digital waveforms, can be used as cheap Digital-to-Analog (D/A) converters with few external components. A wide variety of microcontroller applications exists that need analog output but do not require high resolution D/A converters. Some speech applications (talk back units, speech synthesis systems in toys, etc.) also do not require high resolution D/A converters. For these applications, Pulse Width Modulated outputs may be used to convert to analog outputs.

Conversion of PWM waveforms to analog signals involves the use of analog low-pass filters. This brief application note describes the design criteria of the analog filters necessary and the requirements of the PWM frequency. Later in this application note, a simple RC low-pass filter is designed to convert pulse-width modulated speech signals of 4 KHz bandwidth.

In a typical PWM signal, the base frequency is fixed, but the pulse width is a variable. The pulse width is directly proportional to the amplitude of the original unmodulated signal. In other words, in a PWM signal, the frequency of the waveform is a constant while the duty cycle varies (from 0 % to 100 %) according to the amplitude of the original signal. A typical PWM signal is shown in Figure 1.

A Fourier analysis of a typical PWM signal (like the one depicted in Figure 1) shows that there is a strong peak at frequency \( F_n = \frac{1}{T} \). Other strong harmonics also exist at \( F = K\frac{1}{T} \), where \( K \) is an integer. These peaks are unwanted noise and should be eliminated. This requires that the PWM signal be low-pass filtered, thus eliminating these inherent noise components (see Figure 2).

The band-width of the desired signal should thus be

\[ f_{bw} < \left( \frac{f_{pwm}}{1/T} \right) \]

If \( f_{bw} \) is selected such that \( f_{bw}=f_{pwm} \), then the external low-pass filter should be a brick-wall type filter. Brick-wall type analog filters are very difficult and expensive to build. So, for practical purpose, the external low pass filter should be as shown in Figure 3.

This means, \( f_{bw} << f_{pwm} \)

or \( f_{pwm} >> f_{bw} \)

\[ f_{pwm} = K * f_{bw} \]  \hspace{1cm} (1)

where, \( K \) is a constant such that \( K >> 1 \)

The value of \( K \) should be chosen depending upon by how many dB the inherent fundamental noise component of PWM be rejected. An example follows:

Example: It is required to design a simple RC low-pass filter to obtain an analog output from a pulse width modulated speech signal of bandwidth 4 KHz.

From eqn (1), choosing arbitrarily \( K = 5 \),

\[ f_{pwm} = 5 * f_{bw} = 5 * 4 \text{ KHz} = 20 \text{ KHz}. \]
Using PWM to Generate Analog Output

FIGURE 4 - RC FILTER CONNECTED TO PWM1 OF PIC17C42

Choosing, the -3 dB point at 4 KHz, and using the relation $RC = \frac{1}{(2\pi f^2)}$, we get $R = 4 \text{ K} \Omega$, if $C$ is chosen as 0.01 µF:

- $R = 4.0 \text{ K} \Omega$
- $C = 0.01 \mu F$

Since the PWM frequency is selected as 20 KHz, the fundamental noise peak to be filtered is at 20 KHz. Now, let's calculate by how many dB the main peak of PWM signal is cut-off at 20 KHz:

$$(\text{dB})_{20 \text{KHz}} = -10 \log [1 + (2\pi fRC)^2] = -14 \text{ dB}.$$ 

For many applications, this rejection of -14 dB will not suffice. Therefore instead of a simple RC low-pass filter, a higher order active low-pass filter may be necessary. Or if the microcontroller is capable of modulating at higher PWM frequencies, the rejection of noise will be more.

For example, using 8-bit resolution, the PIC17C42 can generate PWM frequency of 62.5 KHz.

At this frequency the attenuation of the PWM frequency is:

$$(\text{dB})_{62.5 \text{KHz}} = -10 \log [1 + (2\pi fRC)^2] = -24 \text{ dB}.$$ 

The higher frequency of the PIC17C42 PWM outputs makes it easier to generate analog output.

Author: Amar Palacherla
Logic Products Division
INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application report provides examples which illustrate the uses of Pulse Width Modulation (PWM) using the PIC17C42 Timer1 or Timer2 modules. These examples may be modified to suit the specific needs of your application.

This Application Note describes the operation of the PWM. They include the following topics:

1. Simple PWM Operation
2. Variable Period / Variable Duty Cycle PWM
3. External Clock for Timer Timebase
   (ramifications/issues)

The listing file for the Variable Period / Variable Duty Cycle example can be found in Appendix A. The source files can be found on the Microchip BBS. On directions on how to access the Microchip BBS please refer to DS30128, which can also be found in the Microchip Embedded Control Handbook (Literature Number DS00092).

FIGURE 1 - TIMER1 AND TIMER2 BLOCK DIAGRAM WITH PWM MODE
PWM Operation

The control registers that are utilized by Timer1 and Timer2 are shown in Table 1. Shaded boxes are control bits that are not used by the Timer1 nor Timer2 module.

**TABLE 1 - REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE**

<table>
<thead>
<tr>
<th>Name</th>
<th>BANK</th>
<th>ADDR</th>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTSTA</td>
<td>0x06</td>
<td>0x06</td>
<td>PEIR</td>
<td>RTXIR</td>
<td>RTCIR</td>
<td>INTIR</td>
<td>PEIE</td>
<td>RTXIE</td>
<td>RTCIE</td>
<td>INTIE</td>
</tr>
<tr>
<td>CPUSTA</td>
<td>0x07</td>
<td>0x10</td>
<td></td>
<td>STKAV</td>
<td>GLINTD</td>
<td>TO</td>
<td>P0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR1</td>
<td>2</td>
<td>0x10</td>
<td></td>
<td>Timer1 Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR2</td>
<td>2</td>
<td>0x11</td>
<td></td>
<td>Timer2 Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR1</td>
<td>2</td>
<td>0x14</td>
<td></td>
<td>Timer1 Period Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR2</td>
<td>2</td>
<td>0x15</td>
<td></td>
<td>Timer2 Period Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIR</td>
<td>1</td>
<td>0x16</td>
<td>IEB</td>
<td>IRB</td>
<td>TM3IR</td>
<td>TM2IR</td>
<td>TM1IR</td>
<td>CA2IR</td>
<td>CA1IR</td>
<td>TMR3C</td>
</tr>
<tr>
<td>PIE</td>
<td>1</td>
<td>0x17</td>
<td></td>
<td>IEB</td>
<td>TM3IE</td>
<td>TM2IE</td>
<td>TM1IE</td>
<td>CA2IE</td>
<td>CA1IE</td>
<td>TXIE</td>
</tr>
<tr>
<td>PW1DCL</td>
<td>3</td>
<td>0x10</td>
<td></td>
<td>PWM1 Duty Cycle Low Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW2DCL</td>
<td>3</td>
<td>0x11</td>
<td></td>
<td>PWM2 Duty Cycle Low Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW1DCL</td>
<td>3</td>
<td>0x12</td>
<td></td>
<td>PWM1 Duty Cycle High Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW2DCL</td>
<td>3</td>
<td>0x13</td>
<td></td>
<td>PWM2 Duty Cycle High Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCON1</td>
<td>3</td>
<td>0x16</td>
<td>CA2ED1</td>
<td>CA2ED0</td>
<td>CA1ED1</td>
<td>CA1ED0</td>
<td>CA1PR3</td>
<td>TMR3C</td>
<td>TMR2C</td>
<td>TMR1C</td>
</tr>
<tr>
<td>TCON2</td>
<td>3</td>
<td>0x17</td>
<td>CA2OVF</td>
<td>CA1OVF</td>
<td>PWM2ON</td>
<td>PWM1ON</td>
<td>CA1/PR3</td>
<td>TMR3ON</td>
<td>TMR2ON</td>
<td>TMR1ON</td>
</tr>
</tbody>
</table>

Care must be taken when loading values into the PWM registers. These registers are the duty cycle registers (PWxDCH:PWxDCL) and the period register (PRx). Figure 2 shows the proper update timing of these values.

**FIGURE 2 - TIMING FOR UPDATING THE DUTY CYCLE REGISTERS AND PERIOD REGISTER**

![Diagram showing PWM timing and update](image)

**NOTE:** When updating the period register, the value loaded must be greater than the timer value. If the period value is less than the timer value, the duty cycle value is not latched and the timer is not reset (to 00h) until the next Timer = Period match. This causes the current cycle to not output as expected and cause a "glitch".

**NOTE:** It is generally good practice to load the new period value into the PRx register as soon as possible after Point A.
### SIMPLE PWM OPERATION

Simple PWM operation is where the period of the PWM output remains constant, and only the duty cycle is modified. The PWM can operate in either of two modes:

- Hi-resolution mode - the PWxDCL register is modified
- Standard resolution mode - the PWxDCL register is not modified

When operating in the high-resolution mode, only the PWxDCH register is ever modified. Since this takes only a single cycle, this can be done at any time. Also since the period is remaining constant this may be done without any PWM interrupt software overhead.

When operating in the high-resolution mode both the PWxDCH:PWxDCL register pair is modified. Since this is a multicycle update, care needs to be taken that the “new” PWM duty cycle value is not latched until the update is complete. If the duty cycle is latched before this update is complete, the duty cycle will display a “glitch”. If the PWxDCH is written first, the maximum error is 3 Q-cycles (187.5 ns @ 16 MHz). If the PWxDCL is written first, the maximum error is also 3 Q-cycles (187.5 ns @ 16 MHz), with the PWxDCH delayed by one PWM period. This may be acceptable for some applications. If this is not acceptable for your application then a subroutine can be written to ensure that these duty cycle writes are not done when the timer will equal the period. One implementation of this subroutine (PWM_UD) is used in the Variable Period / Variable Duty Cycle PWM example. This is discussed in the following section, with the listing in Appendix A.

Additional code examples can be found in application note AN539 in the Embedded Control Handbook.

### VARIABLE PERIOD / VARIABLE DUTY CYCLE PWM

In a variable period / variable duty cycle PWM both the duty cycle of the PWM as well as the frequency (period) of the PWM are modified.

The 17C42's hardware double buffers the duty cycle registers, but the period registers are not double buffered. What this means is that you can modify the duty cycle registers, but the value will only be latched when the timer register equals period register. Since the period register is not buffered, as the period register is modified this becomes the “new” period. This means that care must be taken when modifying the period register. The most common problem would be to modify the period register resulting in a “glitch” to occur. This “glitch” occurs when the period register is modified with a value that is less than the present timer value. The timer does not have a match with the old period value, and continues to count until the timer register equals period register.

Figure 3, shows an example where the period (PR1) register = 7Fh. Then the period is modified to a smaller value (PR1 = 1Fh) without checking that the value in Timer1 (TMR1) register = 3Eh. Since the new period (PR1) value is less than the present timer (TMR1) value, a glitch has occurred.

![Figure 3 - Modifying Period Register “Glitch”](image-url)
PWM Operation

Care must be taken when writing a 10-bit duty cycle value. Since this requires two register writes, the Timer equals period could occur between these two writes, which would give a duty cycle that was not as expected. The cases are as follows:

1. If the duty cycle low (DCL) register is written, and then the Timer equals period. The old DCH register and the new DCL register becomes the duty cycle.

2. If the duty cycle high (DCH) register is written, and then the Timer equals period. The new DCH register and the old DCL register becomes the duty cycle.

At the following occurrence of the timer equaling the period, the second register written would be updated. The subroutine PWM_UD (Appendix A) ensures that these duty cycle writes are not done when the timer will equal the period.

A software example of a variable period / variable duty cycle is shown in appendix A. In this example the period is double buffered in software, and the new period value is loaded in the timer overflow interrupt service routine. When the new duty cycle needs to be loaded. The device connections are shown in Figure 4. This program has two PWM settings (period / duty cycle combinations) that are switched between depending on the level on pin RB0. A frequency generator was used to give a low frequency signal on the RB0 pin. Figure 5 shows an example of the input and output waveforms.

![FIGURE 4 - APPLICATION HARDWARE SETUP](image)

The program listing in appendix A implements this example, Figure 8 is the flowchart of the program. This example may be modified to suit the particular needs of your application. The Table 3 is a summary of the requirements for this program (@ 16 MHz):

**TABLE 3 - PROGRAM REQUIREMENTS**

<table>
<thead>
<tr>
<th>Code Size:</th>
<th>52 Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used:</td>
<td>11 Bytes</td>
</tr>
<tr>
<td>Interrupt Service Routine time</td>
<td>3.0 usec</td>
</tr>
<tr>
<td>Subroutine time</td>
<td>4.5 usec</td>
</tr>
<tr>
<td>Maximum PWM frequency:</td>
<td>200 KHz</td>
</tr>
<tr>
<td>PWM Accuracy:</td>
<td>62.5 nsec</td>
</tr>
</tbody>
</table>

**FIGURE 5: EXAMPLE APPLICATION WAVEFORMS.**

![Waveforms](image)
EXTERNAL CLOCK FOR TIMER TIMEBASE

The counters used for the time base of the PWM outputs can be software selected to operate from an external clock source. This allows a lower frequency PWM to be achieved. Doing this brings up new issues that must be understood for the application.

One of these issues is clock synchronization. All external clocks must be synchronized to the internal operating speed of the microcontroller, as shown in figure 9. When this synchronization occurs the PWM output is not truly operating from the external clock, but actually the internal synchronized clock. This leads to a "jitter" of the output to the clock. This jitter is caused from the delta time between the external clock and the synchronized clock not being constant. The synchronization errors are:

- Duty cycle error = \( \pm T_{cy} \)
- Period error = \( \pm T_{cy} \)

If you needed to run the PWM at a low frequency, and also want to reduce the "jitter" from the use of an external asynchronous clock, a PWM output could be used as the synchronous clock source. When the clock is synchronized to the device the clock error is always constant, so there is no jitter. Figure 7 shows this example.

**FIGURE 7 - PWM OUTPUT TO GENERATE A SYNCHRONOUS CLOCK**

**FIGURE 6 - EXTERNAL CLOCK SYNCHRONIZATION**

[Diagram showing external and synchronized clocks]
The PWM outputs could be programmed to have a frequency of 20 KHz, so to reduce audible noise. The PWM2 signal is connected to the RB4/TCLK12, as shown in Figure 11. The PR2 register could be loaded with 14h (20), to give a interrupt every 1 KHz. This interrupt can then trigger tasks, such as updating the duty cycle of PWM1. This is useful in motor control as well as other applications where the update rate is less then the PWM frequency.

CONCLUSION

The PIC17C42s PWM features offer a high performance solution at a lower system cost then previously available. The versatility of PWM’s make the PIC17C42 ideal for motor control applications (ses AN532) and many industrial control applications.

Author: Mark Palmer
Logic Products Division

FIGURE 11 - SAMPLING SCHEME
This is the basic outline for a program that generates a variable PWM output. The PWM's period and duty cycle can be varied. The new period (NEW_PR1) and the new duty cycle (NEW_DC1 and NEW_DC1Q) are loaded by the user program. The peripheral interrupt routine loads the new period value (frequency) into the PR1 register. A subroutine (PWM_UD) is also used to ensure that the 10-bit duty cycle registers are updated in the same PWM cycle, i.e. the timer match does not occur between two duty cycle register writes.

The duty cycle value gets latched on the overflow (Period match) of the timer. The period value gets modified as soon as the period register is changed. Therefore care must be taken in updating the period register. In cases where the period value is modified to a smaller value, we must ensure that the Timer counter is less than this value when the period register is updated (TMRI < new PR1).

If TMRI is greater than PR1, the counter will count to FFh, rollover to OOH, and only cause the overflow interrupt when it then reaches the period value. This would give a wrong PWM output.

In this example the event which cause the PWM to be updated is an asynchronous event. A low frequency signal was placed on port pin RBO. For a high level the PWM registers are updated as follows:

- PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
- For a low level the PWM registers are updated as follows:
  - PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h

Do the EQUate table

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0020</td>
<td>NEW_DC1</td>
<td>0x20</td>
</tr>
<tr>
<td>0021</td>
<td>NEW_DC1Q</td>
<td>0x21</td>
</tr>
<tr>
<td>0022</td>
<td>NEW_PR1</td>
<td>0x22</td>
</tr>
<tr>
<td>0025</td>
<td>PWM_WIN</td>
<td>0x25</td>
</tr>
<tr>
<td>0026</td>
<td>CALC_PR</td>
<td>0x26</td>
</tr>
<tr>
<td>0027</td>
<td>FLAG_REG</td>
<td>0x27</td>
</tr>
<tr>
<td>001A</td>
<td>DC1H</td>
<td>0x1A</td>
</tr>
<tr>
<td>001B</td>
<td>DC1QH</td>
<td>0x1B</td>
</tr>
<tr>
<td>001C</td>
<td>PR1H</td>
<td>0x1C</td>
</tr>
<tr>
<td>001D</td>
<td>DC1L</td>
<td>0x1D</td>
</tr>
<tr>
<td>001E</td>
<td>DC1QL</td>
<td>0x1E</td>
</tr>
<tr>
<td>001F</td>
<td>PR1L</td>
<td>0x1F</td>
</tr>
<tr>
<td>07FF</td>
<td>END_OF_PROG_MEM</td>
<td>0x07FF</td>
</tr>
<tr>
<td>0004</td>
<td>ALUSTA</td>
<td>0x04</td>
</tr>
<tr>
<td>0006</td>
<td>CPUSTA</td>
<td>0x06</td>
</tr>
<tr>
<td>0007</td>
<td>INTSTA</td>
<td>0x07</td>
</tr>
<tr>
<td>000A</td>
<td>W</td>
<td>0x0A</td>
</tr>
<tr>
<td>0011</td>
<td>DDRB</td>
<td>0x11</td>
</tr>
<tr>
<td>0012</td>
<td>PORTB</td>
<td>0x12</td>
</tr>
<tr>
<td>0016</td>
<td>PIR</td>
<td>0x16</td>
</tr>
<tr>
<td>0017</td>
<td>PIE</td>
<td>0x17</td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
PWM Operation

; Bank 2
0010 TMR1 EQU 0x10
0011 TMR2 EQU 0x11
0012 TMR3 EQU 0x12
0013 TMR3h EQU 0x13
0014 PR1 EQU 0x14
0015 PR2 EQU 0x15
0016 PR3L EQU 0x16
0017 PR3h EQU 0x17
0010 PW1DCD EQU 0x10
0011 PW2DCD EQU 0x11
0012 PW1DCH EQU 0x12
0013 PW2DCH EQU 0x13
0016 TCON1 EQU 0x16
0017 TCON2 EQU 0x17
0000 C02B
  ORG 0x0000
  GOTO START ; Origin for the RESET vector
  ; On reset, go to the start of
  ; the program
0008 C07C
  ORG 0x0008
  GOTO EXT_INT ; Origin for the external RA0/INT
  ; interrupt vector
0010 C07D
  ORG 0x0010
  GOTO RTCCINT ; Goto the RTCC overflow interrupt
  ; routine
0018 C07E
  ORG 0x0018
  GOTO RT_INT ; Goto the ext. interrupt on
  ; RA1/RT routine
0020 B801
  MOVLB PER_INT, 1 ; Select register Bank 1
0021 9416
  BTSS PIR, 4 ; Did Timerl overflow?
0022 C022
  GOTO ERROR ; Not a Timerl overflow.
  ; No other interrupts should
  ; be enabled, so error.

; The interrupt routine for any peripheral interrupt, This routine
; only deals with Timerl interrupt.
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routine)
; case1 - only Tl overflow = 12 cycles
; case2 - Other = Infinite Loop

; Once the enabled Timerl overflow occurs, the period register
; is loaded. This PWM waveform will remain until the PWM duty
; cycle and / or period is updated. Until such update, there is no
; S/W overhead from Tl interrupts (Tl interrupts can be disabled).
; NOTE: If PW1DCD => PR1, then the duty cycle of this PWM output
; is 100%.
; NOTE: The new Period register (PR1) value, must always be greater
; than the value in the Timerl register (TMR1). If a PR1 value
; is loaded that is less then the TMR1 value, the timer will
; continue to count until it reaches the PR1 value, i.e. TMR1
; will overflow at FFh and the count to the new PR1 value.
; Minimum PR1 value is 0Ah, due to time to load new values and
; execute the peripheral interrupt service routine.

0023 8C16
  TIOVFL BCF PIR, 4 ; Clear Overflow interrupt flag
0024 B802
  MOVLB 2 ; Bank2
PWM Operation

0025 7422 MOVFP  NEW_PR1,PR1 ; Load this period value
0026 B801 MOVLB  3 ; Bank 0
0027 8C17 BCF PIE, 4 ; Disable T1 interrupt
0028 B800 MOVLB  0 ; (until transition on PORTB0)
0029 3F12 BTG PORTB, 7 ; Bank 0
002A 0005 RETFIE ; Transition PortB 7 pin (H->L,
002B 8406 ; Return from Interrupt

; This is the start of the program.

; START BSF CPUSTA,4 ; Disable ALL interrupts via the
; Global Interrupt Disable
; (GLINTD) bit.

; MAIN

; Place Main program here

002C B803 MOVLB  3 ; Select register Bank 3
002D 2817 CLRFL TCON2,0 ; Stop the timers, Single Capture
002E B070 MOVLW 0x670 ; Initialize TCON1 so that
002F 0116 MOVWF TCON1 ; T1 (8-bit), T2 (8-bit),
0030 B00D and T3 run off the internal
0031 0125 ; system clock. Timer3 uses
0032 0125 ; period register

; Load this period value

0033 2811 MOVLB  0 ; Bank 0
0034 2912 SIEF DDRB, 1 ; Port B is an input
0035 8F11 CLRF PORTB, 1 ; Set output values to 0 (for
0036 2927 BCF DDRB, 7 ; PORTB7 is an output. Used to

; Clear the Flag registers

; Load registers with the PWM values that we will switch between. One set
; for the time PORTBO is high and another set for when low.

; For a high level the PWM registers are updated as follows:
; PR1 = 7Fh, PWLDC = 3Fh, and PW1DCL = 40h
; At 16Mhz this gives a period of 31.75 us, and a duty cycle of 16.625 us
; For a low level the PWM registers are updated as follows:
; PR1 = 1Fh, PWLDC = 07h, and PW1DCL = 80h
; At 16Mhz this gives a period of 7.75 us, and a duty cycle of 6.00 us

0037 B803 MOVLB  3 ; Bank 3
0038 B03F MOVLW 0x3F ; The Duty Cycle initial value is
0039 4A1A MOVFP W, DC1H ; 50% of the initial period
003A B040 MOVLW 0x40 ;
003B 4A1B MOVFP W, DC1QH ; Duty Cycle low = 01
003C B007 MOVLW 0x07 ; The Duty Cycle initial value is
003D 4A1D MOVFP W, DC1L ; 25% of the initial period
003E B080 MOVLW 0x80 ;
003F 4A1E MOVFP W, DC1QL ; Duty Cycle low = 10

; Bank 2

0040 B02 MOVLB  2 ;
0041 B07F MOVLW 0x7F ;
0042 4A1C MOVFP W, PR1H ; The initial period value is 50%
; of full scale (for High)
0043 B01F MOVLW 0x1F ;
0044 4A1F MOVFP W, PR1L ; The initial period value is
; of full scale (for Low)

; Default PWM values should be set, and the timer should be started
; and the interrupts enabled.

; Load the Period register

0045 B0F0 MOVLW 0x70 ;
0046 0114 MOVWF PR1 ;
0047 B803 MOVLB  3 ; Select register Bank 3
0048 B0C0 MOVLW 0xC0 ;
0049 0112 MOVWF PW1DCH ;
PWM Operation

; PWM Operation

; Only need to update PWM values on the first occurrence of a new level on
; Else loop waiting for level to change.

; Here is where we update the PWM values (period and Duty cycle) for high

; Here is where we update the PWM values (period and Duty cycle) for low

; This code segment ensure that all PWM values (period and duty cycle)
; are updated at the same time. This is done by ensuring that the Timer
; is at least PWM_WIN (ODh) cycles before the PR1 value (PR1 - PWM_WIN >
; If not a "glitch" could occur in the PWM waveform. When only the 1st duty
; cycle register is latched for this PWM cycle, and the following PWM period
; will latch the 2nd duty cycle register.

; PWM Operation

0053 B827
MOVWF PW1DCL ; effectively loaded with 0
; Only need to update PWM values on the first occurrence of a new level on
; Else loop waiting for level to change.

HIGH1ST
BCF FLAG_REG, 0 ; First time in loop (this

; HIGHCYC
BTFSS PORTB, 0 ; Is PortBO low

; LOW1ST
BCF PORTB, 0 ; PORTBO = L

; LOWCYC
BTFSC PORTB, 0 ; Is PortBO high

; Here is where we update the PWM values (period and Duty cycle) for high

; Here is where we update the PWM values (period and Duty cycle) for low

; This code segment ensure that all PWM values (period and duty cycle)
; are updated at the same time. This is done by ensuring that the Timer
; is at least PWM_WIN (ODh) cycles before the PR1 value (PR1 - PWM_WIN >
; If not a "glitch" could occur in the PWM waveform. When only the 1st duty
; cycle register is latched for this PWM cycle, and the following PWM period
; will latch the 2nd duty cycle register.

; PWM Operation

0065 B803
MOVFB PW1DCL ; effectively loaded with 0

0066 8406
PW1UD
BSF CPUSTA, 4 ; Disable Global Interrupts

0067 B802
MOVFB PW1DCL ; effectively loaded with 0

0068 8406
PW1UD
BSF CPUSTA, 4 ; Disable Global Interrupts

0069 B802
MOVFB PW1DCL ; effectively loaded with 0

006A 8406
PW1UD
BSF CPUSTA, 4 ; Disable Global Interrupts

© 1993 Microchip Technology Inc.
PWM Operation

<table>
<thead>
<tr>
<th>Location</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0072 6A20</td>
<td>MOVFP</td>
<td>NEW_DCl, W</td>
<td>Your New PWM MSB</td>
</tr>
<tr>
<td>0073 0112</td>
<td>MOVWF</td>
<td>PW1BCH</td>
<td>Loaded in duty cycle buffer</td>
</tr>
<tr>
<td>0074 6A21</td>
<td>MOVFP</td>
<td>NEW_DClQ, W</td>
<td>Your New PWM LSB</td>
</tr>
<tr>
<td>0075 0110</td>
<td>MOVWF</td>
<td>PW1DCL</td>
<td>Loaded in duty cycle buffer</td>
</tr>
<tr>
<td>0076 B801</td>
<td>MOVLB</td>
<td>1</td>
<td>Back to Bank 1</td>
</tr>
<tr>
<td>0077 8C16</td>
<td>BCF</td>
<td>PIR, 4</td>
<td>Clear T1 Overflow interrupt</td>
</tr>
<tr>
<td>0078 B417</td>
<td>BSF</td>
<td>PIE, 4</td>
<td>Enable T1 int</td>
</tr>
<tr>
<td>0079 8C06</td>
<td>BCF</td>
<td>CPUSTA, 4</td>
<td>Enable Global Interrupts</td>
</tr>
<tr>
<td>007A B800</td>
<td>MOVLB</td>
<td>0</td>
<td>Bank 0</td>
</tr>
<tr>
<td>007B 0002</td>
<td>RETURN</td>
<td></td>
<td>** this does not need to be as a subroutine.</td>
</tr>
</tbody>
</table>

; Other Interrupt routines. (Not utilized in this example)

<table>
<thead>
<tr>
<th>Location</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>007C 0005</td>
<td>EXT_INT</td>
<td>RETFIE</td>
<td>RA0/INT interrupt routine</td>
</tr>
<tr>
<td>007D 0005</td>
<td>RTCCINT</td>
<td>RETFIE</td>
<td>RTCC overflow interrupt routine</td>
</tr>
<tr>
<td>007E 0005</td>
<td>RT_INT</td>
<td>RETFIE</td>
<td>RA1/RT interrupt routine</td>
</tr>
<tr>
<td>007F C02B</td>
<td>SRESET</td>
<td>GOTO START</td>
<td>If program became lost, goto START and reinitalize.</td>
</tr>
</tbody>
</table>

; When the executed address is NOT in the program range, the 16-bit address should contain all l's (a CALL $FF01). At this location you could branch to a routine to recover or shut down from the invalid program execution.

<table>
<thead>
<tr>
<th>Location</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>07FF C07F</td>
<td>ORG END_OF_PROG_MEM</td>
<td>GOTO SRESET</td>
<td>The program has lost its mind, do a system reset</td>
</tr>
</tbody>
</table>

Errors : 0
Warnings : 0
INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application report provides examples which illustrate the uses of input capture using the PIC17C42 Timer3 module. These examples may be modified to suit the specific needs of an application.

This Application Note has 4 examples that utilize the Timer3 input capture. They are:

1. Frequency Counter (Period Measurement)
2. Frequency Counter (Period Measurement) using a Free Running Timer
3. Pulse Width Measurement
4. Frequency Counter (Period Measurement) with Input Prescaler

TIMER3 DESCRIPTION

Timer3 is a 16-bit counter that has two modes of operation which can be software selected. The CA1/PR3 bit (TCON2<3>) selects the mode of operation. The two modes are:

1. Timer3 with Period Register and Single Capture Register (see Figure 1)
2. Timer3 and Dual Capture Registers (see Figure 2)

Timer3 is the time-base for capture operations.

FIGURE 1 - TIMER3 WITH PERIOD REGISTER AND SINGLE CAPTURE REGISTER

[Diagram showing Timer3 with Period Register and Single Capture Register]

Timer + Period Reg + One Capture Mode (CA1/PR3 = 0)
The period register allows the time base of Timer3 to be something other than the 2\(^{16}\) counter overflow value, which corresponds to FFFFh (65536) cycles. This is accomplished by loading the desired period value into the PR3H/CA1H:PR3L/CA1L register pair. The overflow time can be calculated by this equation:

\[ T_{off} = T_{clk} \times (\text{value in PR3H/CA1H:PR3L/CA1L register pair} + 1). \]

Where \( T_{clk} \) is either the internal system clock (Tcy) or the external clock cycle time. Table 1 shows time-out period for different period values at different frequencies. The values in the register are the closest approximation for the period value. All examples in this application report uses a Timer3 overflow value of FFFFh.
### TABLE 1 - TIMER3 OVERFLOW TIMES

<table>
<thead>
<tr>
<th>Overflow Time</th>
<th>@ 16 MHz (250 ns)</th>
<th>@ 10 MHz (400 ns)</th>
<th>@ 8 MHz (500 ns)</th>
<th>@ 5 MHz (800 µs)</th>
<th>@ 2 MHz (2.0 µs)</th>
<th>@ 32 KHz (125 µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.192 s</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>131.072 ms</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>0xFFFF</td>
<td>0x0418</td>
</tr>
<tr>
<td>52.428 ms</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>0xFFFF</td>
<td>0x6666</td>
<td>0x01A3</td>
</tr>
<tr>
<td>32.7675 ms</td>
<td>N.A.</td>
<td>0xFFFF</td>
<td>0x9FF</td>
<td>0x3FFF</td>
<td>0x0106</td>
<td></td>
</tr>
<tr>
<td>26.214 ms</td>
<td>N.A.</td>
<td>0xFFFF</td>
<td>0xCE20</td>
<td>0x3388</td>
<td>0x00D3</td>
<td></td>
</tr>
<tr>
<td>16.384 ms</td>
<td>0xFFFF</td>
<td>0xA000</td>
<td>0x8000</td>
<td>0x5000</td>
<td>0x2000</td>
<td>0x0083</td>
</tr>
<tr>
<td>10.0 ms</td>
<td>0x9C40</td>
<td>0x61A8</td>
<td>0x4E20</td>
<td>0x30D4</td>
<td>0x1388</td>
<td>0x0050</td>
</tr>
<tr>
<td>4.0 ms</td>
<td>0x3E80</td>
<td>0x2710</td>
<td>0x1F40</td>
<td>0x1388</td>
<td>0x07D0</td>
<td>0x0020</td>
</tr>
<tr>
<td>1.0 ms</td>
<td>0x0FA0</td>
<td>0x09C4</td>
<td>0x07D0</td>
<td>0x04E2</td>
<td>0x01F4</td>
<td>0x0008</td>
</tr>
<tr>
<td>600 µS</td>
<td>0x0960</td>
<td>0x05DC</td>
<td>0x04B0</td>
<td>0x02EE</td>
<td>0x012C</td>
<td>0x0005</td>
</tr>
<tr>
<td>100 µS</td>
<td>0x0190</td>
<td>0x00FA</td>
<td>0x00C8</td>
<td>0x007D</td>
<td>0x0032</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

The uses of an input capture are all for time based measurements. These include:

- Frequency measurement
- Duty cycle and pulse width measurements

The PIC17C42 has two pins (RB0/CAP1 and RB1/CAP2) which can be used for capturing the Timer3 value, when a specified edge occurs. The input capture can be specified to occur on one of the following four events:

- Falling Edge
- Rising Edge
- 4th Rising Edge
- 16th Rising Edge

These are specified, for both capture pins, by the register TCON1<7:4>.

This flexibility allows an interface without the need of additional hardware to change polarity or specify an input prescaler.
Capture Module

The control registers that are utilized for by Timer3 are shown in Table 2. Shaded Boxes are control bits that are not used by the Timer3 module, the Peripheral Interrupt enable and flag bits, and the Global Interrupt enable bit.

### Table 2 - Registers Associated with Timer3 and Capture:

<table>
<thead>
<tr>
<th>Name</th>
<th>BANK</th>
<th>ADDR</th>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTSTA</td>
<td>All</td>
<td>6h</td>
<td>PEIR</td>
<td>RTXIR</td>
<td>RTCIR</td>
<td>INTIR</td>
<td>PEIE</td>
<td>RTXIE</td>
<td>RTCIE</td>
<td>INTIE</td>
</tr>
<tr>
<td>CPUSTA</td>
<td>All</td>
<td>7h</td>
<td>-</td>
<td>-</td>
<td>STKAV</td>
<td>GLINTD</td>
<td>TO</td>
<td>PD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR3L</td>
<td>2</td>
<td>12h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer3 LSB Register</td>
<td></td>
</tr>
<tr>
<td>TMR3H</td>
<td>2</td>
<td>13h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer3 MSB Register</td>
<td></td>
</tr>
<tr>
<td>CA2L</td>
<td>3</td>
<td>14h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer3 Capture2 LSB Register</td>
<td></td>
</tr>
<tr>
<td>CA2H</td>
<td>3</td>
<td>15h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer3 Capture2 MSB Register</td>
<td></td>
</tr>
<tr>
<td>PIR</td>
<td>1</td>
<td>16h</td>
<td>IRB</td>
<td>TM3IR</td>
<td>TM2IR</td>
<td>TM1IR</td>
<td>CA2IR</td>
<td>CA1IR</td>
<td>TBMT</td>
<td>RBFL</td>
</tr>
<tr>
<td>PIE</td>
<td>1</td>
<td>17h</td>
<td>IEB</td>
<td>TM3IE</td>
<td>TM2IE</td>
<td>TM1IE</td>
<td>CA2IE</td>
<td>CA1IE</td>
<td>TXIE</td>
<td>RCIE</td>
</tr>
<tr>
<td>PR3L/CA1L</td>
<td>2</td>
<td>16h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer3 Period MSB Register/Capture1 MSB Register</td>
<td></td>
</tr>
<tr>
<td>PR3H/CA1H</td>
<td>2</td>
<td>17h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer3 Period MSB Register/Capture1 MSB Register</td>
<td></td>
</tr>
<tr>
<td>TCON1</td>
<td>3</td>
<td>16h</td>
<td>CA2ED1</td>
<td>CA2ED0</td>
<td>CA1ED1</td>
<td>CA1ED0</td>
<td>16/8</td>
<td>TMR3C</td>
<td>TMR2C</td>
<td>TMR1C</td>
</tr>
<tr>
<td>TCON2</td>
<td>3</td>
<td>17h</td>
<td>CA2OVF</td>
<td>CA1OVF</td>
<td>PWM2ON</td>
<td>PWM1ON</td>
<td>CA1/PR3</td>
<td>TMR3ON</td>
<td>TMR2ON</td>
<td>TMR1ON</td>
</tr>
</tbody>
</table>

This Application Note has 4 examples that utilize the Timer3 input capture. They are:

1. Frequency Counter (Period Measurement)
2. Frequency Counter (Period Measurement) using a Free Running Timer
3. Pulse Width Measurement
4. Frequency Counter (Period Measurement) with Input Prescaler

All these examples can be run from a simple setup, this is show in Figure 3.

### Figure 3 - Application Hardware Setup

A discussion of each application with the operation of the software and application issues. The source listings for these are in appendices A-D.
PERIOD MEASUREMENT  
(FREQUENCY COUNTER)

Period measurement is simply done by setting the counter to 0000h, then starting the counter on the 1st rising edge. On the following rising edge, the capture 2 register is loaded with the Timer3 value and the Timer3 (TMR3) register is cleared. If the period is greater than the overflow rate of Timer3, the timer overflows, causing an interrupt. With a TMR3 overflow, an interrupt occurs and the overflow counter may need to be incremented. The overflow counter should be incremented if:

1. The TMR3 overflow is the only interrupt source.
2. Both the TMR3 overflow and capture2 interrupts occurred at near the same time, but the TMR3 overflow occurred first (Most Significant Byte of the Capture2 register = 00h).

Once a capture2 has occurred, the capture registers are moved to data RAM, the capture2 interrupt flag is cleared and the TMR3 register is loaded with an offset value. This offset value is the number of cycles from the time the interrupt routine is entered to when the TMR3 register is reloaded. In this example a data RAM location is used as an overflow counter. This gives in effect a 24-bit timer. The software flow for this routine is shown in Figure 4.

FIGURE 4 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2

The program listing in Appendix A implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

- Code size: 30 Words
- RAM used: 4 Bytes
- Maximum frequency that can be measured: 130 KHz
- Minimum frequency that can be measured: 0.25 Hz
- Measurement Accuracy: +/- 1cy (+/- 250 nsec)
PERIOD MEASUREMENT (FREQUENCY COUNTER) USING A FREE RUNNING TIMER

In many applications the timer (TMR3) would need to be used for multiple tasks, and could not be reset (modified) by any one of these tasks. This is called a free running timer. To do period measurement in a free running timer application, the program needs to store each capture in a data RAM location pair (word). The 1st capture in data RAM locations input capture2A (IC2AH:IC2AL) and the 2nd capture in data RAM locations input capture2B (IC2BH:IC2BL). Once the two captures have occurred, the values in these two words is subtracted. Since this is a free running timer, the value in input capture2B may be less than the value in input capture2A. This is if the 1st capture occurs, then the Timer3 overflows, and then the 2nd capture occurs. So an overflow counter should only be incremented if the Timer3 overflow occurs after a capture1 but before the capture2 occurs. With the use of an overflow counter this becomes an effective 24-bit period counter. The software flow for this routine is shown in Figure 5.

The program listing in Appendix B implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

- Code size: 41 Words
- RAM used: 76 Bytes
- Maximum frequency that can be measured: 80 KHz
- Minimum frequency that can be measured: 0.25 Hz
- Measurement Accuracy: +/- Tcy (± 250 nsec)

FIGURE 5 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2

![Diagram showing software timing flow relative to input signal on RA1/CAP2](image-url)
PULSE WIDTH MEASUREMENT USING A FREE RUNNING TIMER

Applications that require the measurement of a pulse width can also be easily handled. The PIC17C42 can be programmed to measure either the low or the high pulse. The software example in Appendix C measures the High pulse time. The program is initialized to capture on the rising edge of the RA1/CAP2 pin. After this event occurs, the capture mode is switched to the falling edge of the RA1/CAP2 pin. When the capture edge is modified (rising to falling, or falling to rising) a capture interrupt is generated. This "false" interrupt request must be cleared before leaving the interrupt service routine, or the program will immediately re-enter the interrupt service routine due to this "false" request. When the falling edge of the RA1/CAP2 pin occurs, the difference of the two capture values is calculated. The flow for this is shown in Figure 6.

Due to the software overhead of the Peripheral Interrupt routine the following are the limitations on the input signal on RA1/CAP2 pin. This does not include any software overhead that may be required in the main routine, or if additional peripheral interrupt features need to be included. This is shown in Table 3. If you assume that the input is a square wave (high time = low time), one needs to take the worst case time of the two minimum pulse times (11 µS) times two, to determine the period. The maximum continuous input frequency would then be approximately 45.5 KHz. For a single pulse measurement, minimum pulse width is 4.5 µS.

The program listing in Appendix C implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

- Code size: 51 Words
- RAM used: 7 Bytes
- Maximum frequency that can be measured: 71 KHz
- Minimum frequency that can be measured: 0.25 Hz
- Measurement Accuracy: +/- Tcy (±250 nsec)

FIGURE 6 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2

TABLE 3 - PERIPHERAL INTERRUPT ROUTINE

<table>
<thead>
<tr>
<th>EVENT</th>
<th># of Cycles</th>
<th>Time @ 16 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st CAPTURE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capture1 Only</td>
<td>18</td>
<td>4.5 µS</td>
</tr>
<tr>
<td>Capture1 and Timer Overflow</td>
<td>30</td>
<td>7.5 µS</td>
</tr>
<tr>
<td>2nd CAPTURE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capture Only</td>
<td>35</td>
<td>8.75 µS</td>
</tr>
<tr>
<td>Capture and Timer Overflow</td>
<td>41</td>
<td>10.25 µS</td>
</tr>
<tr>
<td>Minimum Pulse High</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capture1 and Timer Overflow + INT Latency</td>
<td>33</td>
<td>8.25 µS</td>
</tr>
<tr>
<td>Minimum Pulse Low</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capture2 and Timer Overflow + INT Latency</td>
<td>44</td>
<td>11 µS</td>
</tr>
<tr>
<td>Minimum Period</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(square wave)</td>
<td>2 * (Minimum Pulse Low)</td>
<td>88</td>
</tr>
</tbody>
</table>
PERIOD MEASUREMENT (FREE RUNNING TIMER) WITH A PRESCALER

Occasionally the application may require a prescaler on the input signal. This may be due to application requirements, such as:

- Require higher resolution measurement of the input signal
- Reduce interrupt service overhead
- The input frequency is higher than interrupt service routine

The software selectable prescaler of the PIC17C42 allows the designer to easily implement this in their system without the cost of additional hardware. Care must be taken in determining if this option is appropriate. For example, if the input frequency is not stable (excessive frequency change per period) then the prescaler will give a less accurate capture value than the individual measurements.

In cases where the resolution of the input frequency is important, the Prescaler can be used to reduce the input capture error. There are two components to input capture:

1. Resolution Error
2. Input Synchronization Error

These two errors add together to form the total capture error. Resolution error is dependent on the rate at which the timer is incremented, remember the timer may be based on an external clock (must be slower than Tcy). The input synchronization error is dependent on the system clock speed (Tcy), and will be less than Tcy.

It is easy to see that when a capture occurs the synchronization error (Tesync) can be up to Tcy (see Figure 7). This error is constant regardless of the number of edges that occur before the capture is taken. So a capture on the 1st edge gives a synchronization error per sample up to Tcy. While a capture taken on the 16th edge gives a synchronization error per sample only up to Tcy / 16, by achieving a smaller percentage of error, the captured value becomes more accurate.

FIGURE 7 - SYNCHRONIZATION ERROR WITH NO CAPTURE PRESCALER

Notes:
1. Capture edge to actual capture register update latency is 1.75 Tcy max., 0.75 Tcy min. This implies that when measuring a pulse or a period, the measurement error is ± Tcy.
2. With no prescaler on the input capture, two consecutive capturing edges must be apart by at least Tcy. This allows the internal "capture edge detect latch" to reset.
Another scenario is when the signal on the input capture pin is different than the system cycle time (Tcy), for example 1.6 Tcy. If you tried to capture this you would have a capture value of 1. If you set the prescaler to actually capture on the 16th edge you would have 16*1.6 Tcy = 25.6 Tcy, which would be latched on the 26th Tcy (see Figure 8). This 0.4 Tcy error is over 16 samples, which therefore gives an effective error/sample of 0.025 Tcy.

The program listing in Appendix D implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

- Code size: 41 Words
- RAM used: 7 Bytes
- Maximum frequency that can be measured: 80 KHz
- Minimum frequency that can be measured: 0.25 Hz
- Measurement Accuracy: +/- Tcy (± 16.625 nsec)

**FIGURE 8 - INPUT CAPTURE DIVIDED BY 16 PRESCALE EXAMPLE**

<table>
<thead>
<tr>
<th>Tclk</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal on</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPl Pin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal capture edge detect latch (input divided by 16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Author:** Mark Palmer  
Logic Products Division
This is the basic outline for a program that can determine the frequency of an input, via input capture. This routine uses an 8-bit register to count the times that timer3 overflowed. At the max crystal frequency of 16 MHz, this gives an overflow time of 
\[(2^{16}) (256 + 1) (250 \text{ nS}) > 4.21 \text{ sec} \] or a frequency < 0.25 Hz. If measurement of longer time intervals is required, the overflow counter could be extended to 16 (or more) bits.

; Do the EQUate table

LIST P=17C42, C=80, T=ON

ORG Ox0000
GOTO START

ORG Ox0008
GOTO EXT_INT

ORG Ox000C
GOTO RTCCINT

ORG Ox0010
GOTO RT_INT

© 1993 Microchip Technology Inc.
Capture Module

ORG 0x0020

0020 C040 GOTO PER_INT

ORG 0x0028

0028 8406 START BSF CPUSTA,4

MAIN

0029 B803 MOVLB 3

002A 2817 CLRF TCON2,0

002B 8317 BSF TCON2,3

002C B070 MOVLW 0x070

002D 0116 MOVWF TCON1

; RA1/RT routine
; Origin fo
; of any enabled peripheral
; Goto the
; peripheral routine
; Origin fo
; program memory
; Disable A
; Global Interrupt Disable
; (GLINTD) bit.
; Place Main program here
; Select re
; Stop the
; Dual in
; Initialize
; TI (8-b
; and T3 run off the internal
; system clock. Capture2
; captures on the rising edge.

; Initialize Timer 3, so we can count clock cycles. For the
; measurement Timer3 is not started exactly at the RA1/CAP2
; transition. Therefore an offset is required due to softwa
; overhead (3 cycle).

; 002E B802
; MOVLB 2
; Select re

; 002F 280A
; CLRF W,0
; Clear the

; 0030 0113
; MOVWF TMR3H
; Timer3 MS

; 0031 B003
; MOVLB 0x03
; Timer3 LS

; 0032 0112
; MOVWF TMR3L
;
; We need to wait for an event to happen to start timer 3.
; be when the RA1/CAP2 pin to does a Low to High transition
; signifies the start of the first interval to measure. Thi
; by looping while the RA1/CAP2 pin is high, and then loopi
; it is low. When that 2nd loop is exited, there has been a
; high transition and the timer should be started and inter
; enabled.
;
; 0033 B800
; MOVLB 0
; Select re

; 0034 9912
; RA1HIGH BTFSC PORTB,1
; Is the CA

; 0035 C034
; GOTO RA1HIGH
; Loop here

; 0036 9112
; RA1LOW BTFSS PORTB,1
; Is the CA
Capture Module

0037 C036  GOTO RA1LOW ; Loop here
0038 B803  MOVLB 3  ; Select re
0039 8217  BSF TCON2,2 ; Turn on t
003A 8307  BSF INTSTA,3 ; Turn on P
003B B801  MOVLB 1  ; Select re
003C B048  MOVLW 0x48 ; Enable Ca
003D 0117  MOVF PIE ; Interr
003E 8C06  WAIT BCF CPUSTA,4 ; Enable AL
003F C03E  GOTO WAIT ; Loop here

; The interrupt routine for any peripheral interrupt. This
; only deals with Timer3 (T3) interrupts.

0040 B801  PER_INT MOVLB 1  ; Select re
0041 9E16  BTFSC PIR,6 ; Did T3 ov
0042 C057  GOTO T3OVFL ; If not skip next
0043 9316  CK_CAP BTFSS PIR,3 ; Did the R
0044 0005  RETFIE ; No RB1/CA

; If Both a timer 3 overflow and CAP2 interrupt occurred near
; same time. Need to determine which one occurred first. Ther
; two possibilities when the PIR register has both the time
; overflow interrupt and input capture 2 interrupt bits set
; 1. the input capture occurred and then timer 3 overfl
; 2. timer 3 overflowed and then the input capture occ
; in case 2. the MSB of the capture register would be 0h.

0045 B803  CAPTURE MOVLB 3  ; Select re
0046 5422  MOVPP CA2L,IC2L ; Move the
0047 5521  MOVPP CA2H,IC2H ; tempora
0048 B802  MOVLB 2  ; (to prevent
0049 B00A  MOVLW 0x000A ; Select re
004A 0112  MOVF TMR3L ; Reload thou

© 1993 Microchip Technology Inc.
Capture Module

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>004B 280A</td>
<td>CLRF W, 0</td>
<td>Clear the</td>
</tr>
<tr>
<td>004C 0113</td>
<td>MOVWF TMR3H</td>
<td>Timer3 MS</td>
</tr>
<tr>
<td>004D B801</td>
<td>MOVLB l</td>
<td>Select re</td>
</tr>
<tr>
<td>004E 3121</td>
<td>CPFSEQ IC2H</td>
<td>If the hi</td>
</tr>
<tr>
<td>004F C051</td>
<td>GOTO SDEC</td>
<td>Skip the</td>
</tr>
<tr>
<td>0050 0723</td>
<td>DECF T3OFLCNTR, 1</td>
<td>Capture o</td>
</tr>
<tr>
<td>0051 6A23</td>
<td>SDEC MOVFP T3OFLCNTR, W</td>
<td>overflow, so</td>
</tr>
<tr>
<td>0052 4A20</td>
<td>MOVFP W, IC20F</td>
<td>overflow counter.</td>
</tr>
<tr>
<td>0053 2823</td>
<td>CLRF T3OFLCNTR, 0</td>
<td>Store the</td>
</tr>
</tbody>
</table>

; Clear the Capture2 and T3 Overflow interrupt flags. T3 Ov |
; should be cleared again since the overflow may have occur |
; while we were in this (PER_INT) interrupt routine. This w |
; be between the compare of the T3 overflow interrupt flag |
; the resetting of the timer 3 counter, which is about 6 cy |

0054 8B16 | BCF PIR, 3 | Clear Cap |
0055 8E16 | BCF PIR, 6 | Clear Tim |
0056 0005 | RETFIE | Return fr |

; When Timer 3 has overflowed, the overflow counter should |
; incremented. Then the Timer 3 overflow flag, the source o |
; the interrupt, should be cleared. then return to see if a |
; capture also occurred. |

; NOTE: Timer 3 has a process dependent silicon problem whi |
; may give 2 overflow interrupts for each “real” over |
; interrupt. The first ("false") interrupt will occur |
; the timer 0xFEFF to 0xFF00 transition. The second |
; interrupt then occurs at the “real” overflow. A soft |
; fix is included in this routine (T3OVFL). These add |
; lines are designated by ;** as a comment. Once this |
; been corrected, these lines of code can be removed. |
; The routine lable must remain.

0057 B816 T3OVFL BCF PIR,6 ; Clear int
0058 B802 MOVLB 02 ;** Switch
0059 9F13 BTFSC TMR3H,7 ;** Is Time
    ;** the "false"
    ;** Return
005A 0005 RETFIE ;** interrupt.
    ;** "real" overflow.
    ;** Return
005B B801 MOVLB 01 ;** Return
005C 1523 INCF T30FLCNTR,1 ; increment
005D C043 GOTO CK_CAP ; return to
    ; interrupt also occurred

; Other Interrupt routines. (Not utilized in this example)

005E 0005 EXT_INT RETFIE ; RA0/INT i
005F 0005 RTCCINT RETFIE ; (NOT used in
                          ; RTCC over
0060 0005 RT_INT RETFIE ; (NOT used in
                          ; RA1/RT in
0061 C028 SRESET GOTO START ; (NOT used in
    ; If progra
    ; START and reinitalize.

; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL Ox1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.

2 ORG END_OF_PROG_MEM ;
07FF C061 GOTO SRESET ; The progr
    ; do a system reset

Errors : 0
Warnings : 0
This is the basic outline for a program that can determine the frequency of an input, via input capture. This routine uses an 8-bit register to count the times that timer3 overflowed. At the Max crystal frequency of 16 MHz, this gives an overflow time of (2^16) (256 + 1) (250 nS) > 4.21 sec or a frequency < 0.25 Hz. If measurement of longer time intervals is required, the overflow counter could be extended to 16 (or more) bits.

Timer 3 in this example is a free running timer. The input capture is generated on the RB1/CAP2 pin. There is a flag that specifies if this is the 1st or 2nd capture. The first capture is the start of the period measurement. The second capture value gives the end of the period. In this type of measurement If the 2nd capture value < the 1st capture value then the overflow counter should be decremented.

Do the EQUate table

LIST P=17C42, C=80, T=ON

0020 IC20F EQU 0x20 ; T3 overflow regs
0021 IC2BH EQU 0x21 ; T3 ICA2 MSB regs
0022 IC2BL EQU 0x22 ; T3 ICA2 LSB regs
0023 IC2AH EQU 0x23 ; T3 ICB2 MSB regs
0024 IC2AL EQU 0x24 ; T3 ICB2 LSB regs
0025 T3OFLCNTR EQU 0x25 ; Temperay T3 overf
0026 FLAG_REG EQU 0x26 ; Register that has

07FF END_OF_PROG_MEM EQU 0x07FF

0004 ALUSTA EQU 0x04
0006 CPUSTA EQU 0x06
0007 INTSTA EQU 0x07
000A W EQU 0x0A

0012 PORTB EQU 0x12 ; Bank 0
0016 PIR EQU 0x16 ; Bank 1
0017 PIE EQU 0x17

0012 TMR3L EQU 0x12 ; Bank 2
0013 TMR3H EQU 0x13
0016 T3PRL EQU 0x16
0017 T3PRH EQU 0x17

© 1993 Microchip Technology Inc.
Capture Module

0014 CA2L EQU 0x14 ; Bank 3
0015 CA2H EQU 0x15
0016 TCON1 EQU 0x16
0017 TCON2 EQU 0x17

ORG 0x0000 ; Origin fo

0000 C028 GOTO START ; On reset,

ORG 0x0008 ; Origin fo

0008 C068 GOTO EXT_INT ; interrupt vector

ORG 0x0010 ; Origin fo

0010 C069 GOTO RTCCINT ; overflow interrupt vector

ORG 0x0018 ; Origin fo

0018 C06A GOTO RT_INT ; RA1/RT interrupt vector

ORG 0x0020 ; Origin fo

0020 C03E GOTO PER_INT ; of any enabled peripheral

ORG 0x0028 ; Origin fo

0028 8406 START BSF CPUSTA, 4 ; Disable A

MAIN

MOVLB 3 ; Select re

002A 2817 CLRF TCON2, 0 ; Stop the

002B B070 MOVILW 0x070 ; Initialize

002C 0116 MOVWF TCON1 ; T1 (8-b

; and T3 run off the internal
; system clock. Capture2
; captures on the rising edge.

; Initialize Timer 3, load the timer with the number of cycl
; the device executes (from RESET) before the timer is turn
; Therefore the offset is required due to software overhead

002D B802 MOVLB 2 ; Select re

002E 280A CLRF W, 0 ; Clear the

002F 0126 MOVWF FLAG_REG ; Initialize

0030 0113 MOVWF TMR3H ; Timer3 MS
Capture Module

MOVLW 0x13 ; Timer 3 LS
MOVWF TMR3L ; Load the Timer 3 period register with 0xFFFF, which will
; interrupt on the overflow of Timer3

MOVLW 0xFF ;
MOVWF T3PRH ;
MOVWF T3PRL ;
; the timer should be started and interrupts enabled.

MOVLB 3 ; Select re
BSF TCON2,2 ; Turn on t
BSF INTSTA,3 ; Turn on P
MOVLB 1 ; Select re
MOVWLW 0x48 ; Enable Car
MOVWF PIE ; Interr

; This is where you would do the things you wanted to do.
; this example will only loop waiting for the interrupts.

WAIT BCF CPUSTA, 4 ; Enable AL
GOTO WAIT ; Loop here
; Interrupt

; The interrupt routine for any peripheral interrupt. This
; only deals with Timer3 (T3) interrupts.
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routi

; case1 - only T3 overflow = 12 cycles
; case2 - 1st capture = 14 cycles
; case3 - 2nd capture = 30 cycles
; case4 - T3 overflow and 1st capture = 34 cycles
; case5 - T3 overflow and 2nd capture = 50 cycles

; Select re
BTFSC PIR,6 ; Did T3 ov
; If not skip next Instruction
GOTO T3OVFL ; Inc overf
BTFSS PIR,3 ; Did the R
; interrupt?
; No RB1/CA
; Return from Interrupt

; This portion of the code takes the 1st capture and stores
value in register pair IC2AH:IC2AL. When the 2nd capture
is take, its value is stored in register pair IC2BH:IC2BL
A 16-bit subtract is performed, with the final 24-bit res
being stored in IC20F:IC2BH:IC2BL. This value will no lon
be correct after the next capture occurs (IC2BH:IC2BL wil
change), so the main routine must utilize this value befo
it changes.

0043 8B16 CAPTURE BCF PIR,3 ; Clear Cap
0044 B803 MOVLB 3 ; Select re
0045 9826 BTFSC FLAG_REG,0 ; 1st or 2n
0046 C04B GOTO CAP2 ; It was th
0047 5424 CAP1 MOVFF CA2L,IC2AL ; Move the
0048 5523 MOVFF CA2H,IC2AH ; tempora
0049 8026 BSF FLAG_REG,0 ; Have 1st
004A 0005 RETFIE ; Return fr
004B 5422 CAP2 MOVFF CA2L,IC2BL ; Move the
004C 5521 MOVFF CA2H,IC2BH ; tempora
004D E061 CALL SUB16 ; (to perrvent
004E 9926 BTFSC FLAG_REG,1 ; Underflow
004F 0725 DECF T30FLCNTR,1 ; Since und
0050 2926 CLRF FLAG_REG,1 ; overflow counter.
0051 6A25 MOVFP T30FLCNTR,W ; Clear the
0052 4A20 MOVFP W,IC2OF ; underflow and
0053 2825 CLRF T30FLCNTR,0 ; Store the
0054 0005 RETFIE ; counts how
; overflowed.
; Return fr

; When Timer 3 has overflowed, the overflow counter only sh
be incremented when the overflow occurs after a capture 1
but before the capture 2. The 4 possible cases when enter
the T3OVFL section of the PER_INT routine are as follows:
Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
; for Capture 1 to occur). Do Not increment count

; Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
; for Capture 2 to occur). Increment counter

; Case 3: T3 Overflow happened after Capture. Do Not
; increment overflow counter

; Case 4: T3 Overflow occurred before Capture 2 and FLAG_REG
; (waiting for Capture 2 to occur). Increment count

0055 8E16 T3OVFL BCF PIR,6 ; Clear Over
0056 9316 BTFSS PIR,3 ; Did the R
0057 C05E GOTO FR0 ; cause an interrupt?
   ; No, Check
0058 B803 MOVLB 3 ; and 2nd capture
0059 280A CLRF W,0 ; Bank 3
005A 3118 CPFSEQ CA2H ; if CA2H =
005B C05E GOTO FR0 ; first,
005C B801 MOVLB 1 ; bit 0
005D C043 GOTO CAPTURE ; Back to b
005E 9826 FR0 BTFSC FLAG_REG,0 ; Capture h
005F 1525 INCF T3FLCNTR,1 ; Increment
0060 0005 RETFIE ; and do capture
0061 6A24 SUB16 MOVFP IC2AL,W ; Between C
0062 0522 SUBWF IC2BL,1 ; Yes, Inc.
0063 6A23 MOVFP IC2AH,W ;
0064 0321 SUBWFB IC2BH,1 ;
0065 9004 BTFSS ALUSTA,0 ; Is the re
0066 8126 BSF FLAG_REG,1 ; neg., Set
0067 0002 RETURN ; Return fr

; Other Interrupt routines. (Not utilized in this example)

0068 0005 EXT_INT RETFIE ; RA0/INT i
0069 0005 RTCCINT RETFIE ; (NOT used in
006A 0005 RT_INT RETFIE ; RTCC over
006B C028 SRESET GOTO START ; (NOT used in
   ; RA1/INT in
   ; (NOT used in
   ; If progra
; START and reinitalize.
;
; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.
;
; ORG END_OF_PROG_MEM
ORG C06B
GOTO SRESET
; The progr
; do a system reset

END

Errors : 0
Warnings : 0
This is the basic outline for a program that can determine the
Pulse Width of an input, via input capture. This routine uses an
8-bit register to count the times that timer3 overflowed. At the
Max crystal frequency of 16 MHz, this gives an overflow time of
(2**16)(256 + 1)(250 nS) > 4.21 sec or a frequncy < 0.25 Hz. If
measurement of longer time intervals is required, the overflow
counter could be extended to 16 (or more) bits.

; Do the EQUate table

LIST P=17C42, C=80, T=ON

0020 IC2OF EQU 0x20 ; T3 overflow reg
0021 IC2BH EQU 0x21 ; T3 ICA2 MSB reg
0022 IC2BL EQU 0x22 ; T3 ICA2 LSB reg
0023 IC2AH EQU 0x23 ; T3 ICB2 MSB reg
0024 IC2AL EQU 0x24 ; T3 ICB2 LSB reg
0025 T3FLCNTR EQU 0x25 ; Temperay T3 overf

0026 FLAG_REG EQU 0x26 ; Register that has

; FLAG_REG bit 7 6 5 4 3 2 1 0
; CAP1 = 0, 1st Capture
; = 1, 2nd Capture
; UFL = 0, No Underflow
; = 1, Underflow during subtract

07FF END_OF_PROG_MEM EQU 0x07FF

0004 ALUSTA EQU 0x04
0006 CPUSTA EQU 0x06
0007 INTSTA EQU 0x07
000A W EQU 0x0A

0012 PORTB EQU 0x12 ; Bank 0

0016 PIR EQU 0x16 ; Bank 1
0017 PIE EQU 0x17

0012 TMR3L EQU 0x12 ; Bank 2
0013 TMR3H EQU 0x13
0016 T3PRL EQU 0x16
0017 T3PRH EQU 0x17

0014 CA2L EQU 0x14 ; Bank 3
0015 CA2H EQU 0x15
0016 TCON1 EQU 0x16
0017 TCON2 EQU 0x17

ORG 0x0000 ; Origin fo

0000 C028 GOTO START ; On reset,
; the program
Capture Module

ORG 0x0008 ; Origin for
; interrupt vector
GOTO EXT_INT ; Goto the
ORG 0x0010 ; on RA0/INT routine
; Origin fo
GOTO RTCCINT ; overflow interrupt vector
; Goto the
ORG 0x0018 ; Origin fo
GOTO RT_INT ; routine
ORG 0x0020 ; Origin fo
GOTO RA1/RT interrupt vector
; Goto the
ORG 0x0020 ; RA1/RT routine
ORG 0x0028 ; Origin fo
GOTO RA1/RT; of any enabled peripheral
; Goto the
ORG 0x0028 ; peripheral routine
ORG 0x0028 ; Origin fo
START BSF CPUSTA,4 ; program memory
; Disable A
; Global Interrupt Disable
; (GLINTD) bit.
MAIN
MOVLB 3 ; Place Main program here
; Select re
CLRF TCON2,0 ; Stop the
MOVWL 0x070 ; Initialize
MOVWF TCON1 ; T1 (8-bit)
; and T3 run off the internal
; system clock. Capture2
; captures on the rising edge.
; Initialize Timer 3, load the timer with the number of cycl
; the device executes (from RESET) before the timer is turn
; Therefore the offset is required due to software overhead
; MOVLB 2 ; Select re
002E 280A
CLRF W,0 ; Clear the
MOVF TMR3H ; Timer3 MS
0030 0113
MOVFW 0x13 ; Timer3 LS
0031 B013
MOVW TMR3L ;
; Load the Timer 3 period register with 0xFFFF, which will
; interrupt on the overflow of Timer3
; MOVWL 0xFF

Capture Module

```
0034 0117        MOVWF T3PRH     ;
0035 0116        MOVWF T3PRL     ;
                     ; the timer should be started and interrupts enabled.
0036 B803        MOVLB 3         ; Select re
0037 8217        BSF TCON2,2     ; Turn on t
0038 8307        BSF INTSTA,3     ; Turn on P
0039 B801        MOVLB 1         ; Select re
003A B048        MOVLW 0x48      ; Enable Ca
003B 0117        MOVWF PIE        ; Interr
                     ; This is where you would do the things you wanted to do.
                     ; this example will only loop waiting for the interrupts.
003C 6C06        WAIT BCF CPUSTA,4 ; Enable AL
003D C03C        GOTO WAIT       ; Loop here
                     ; Interrupt
                     ; The interrupt routine for any peripheral interrupt, This
                     ; only deals with Timer3 (T3) interrupts.
                     ; Time required to execute interrupt routine. Not including
                     ; interrupt latency (time to enter into the interrupt routi
                     ; case1 - only T3 overflow            = 12 cycles
                     ; case2 - 1st capture               = 20 cycles
                     ; case3 - 2nd capture             = 34 cycles
                     ; case4 - T3 overflow and 1st capture = 32 cycles
                     ; case5 - T3 overflow and 2nd capture = 44 cycles
                     ;
003E B801        PER_INT MOVLB 1     ; Select re
003F 9E16        BTFSC PIR,6      ; Did T3 ov
                     ; If not skip
                     ; Inc overf
0040 C05A        GOTO T3OVFL     ; Did the R
                     ; interrupt?
                     ; No RB1/CA
                     ; Return from
                     ; This potion of the code takes the 1st capture and stores
                     ; value in register pair IC2AH:IC2AL. When the 2nd capture
                     ; is take, its value is stored in register pair IC2BH:IC2BL
                     ; A 16-bit subtract is performed, with the final 24-bit res
                     ; being stored in IC2OF:IC2BH:IC2BL. This value will no lon
                     ; be correct after the next capture occurs (IC2BH:IC2BL wil
```
; change), so the main routine must utilize this value before it changes.

0043 B803 CAPTURE MOVLB 3 ; Select register
0044 9826 BTFSC FLAG_REG,0 ; Capture overflow
0045 C04B GOTO FALLING ; It was the last rising edge.
0046 5424 RISING MOVPF CA2L, IC2AL ; Move the
0047 5523 MOVPF CA2H, IC2AH ; temporary
0048 8026 BSF FLAG_REG,0 ; Set flag
0049 8E16 BCF TCON1,6 ; Change edge to falling

004A C055 GOTO FALSE_C ;*** With the last edge, we have

004B 5422 FALLING MOVPF CA2L, IC2BL ; Move the
004C 5521 MOVPF CA2H, IC2BH ; temporary
 ; (to prevent

004D E06B CALL SUB16 ; Call routine
004E 9926 BTFSC FLAG_REG,1 ; Underflow
004F 0725 DECF T3OFLCNTR, 1 ; Since underflow counter.

0050 2926 CLRPF FLAG_REG, 1 ; Clear the

0051 6A25 MOVFP T3OFLCNTR,W ; underflow and
0052 4A20 MOVFP W,IC2OF ; overflow
0053 2825 CLRF T3OFLCNTR,0 ; Clear the
 ; counts how many

0054 8616 BSF TCON1,6 ; Change edge to rising

; Note when you change the edge of the capture, an addition

; is generated. The capture register must be read before the

; flag is cleared.

0055 550A FALSE_C MOVPF CA2H,W ; Dummy read
0056 540A MOVPF CA2L,W ;
0057 B801 MOVLB 1 ; Select register
0058 8B16 BCF PIR,3 ; Clear Capture
0059 0005 RETFIE ; Return from interrupt

© 1993 Microchip Technology Inc.
; T3 overflow or falling edge capture
;
; When Timer 3 has overflowed, the overflow counter only shall be incremented when the overflow occurs after a capture 1 but before the capture 2. The 6 possible cases when entering the T30VFL section of the PER_INT routine are as follows:
;
; Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting for Capture 1 to occur). Do Not increment count
;
; Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting for Capture 2 to occur). Increment counter
;
; Case 3: T3 Overflow, Then Capture 1 happened. Do Not increment overflow counter
;
; Case 4: T3 Overflow, Then Capture 2 happened Increment counter
;
; Case 5: Capture 1, Then T3 Overflow happened Increment counter
;
; Case 6: Capture 2, Then T3 Overflow happened. Do Not Increment counter
;
005A 8E16 T30VFL
005B 9316 BCF PIR,6 ; Clear Overflow in
005C 0068 BTFSS PIR,3 ; Did the RB1/CAP2 overflow?
005D B803 GOTO FR0 ; No, only overflow
005E 280A MOVLB 3 ; Bank 3
005F 9826 CLRF W,0 ; W = 0
0060 C064 BTFSC FLAG_REG,0 ; T3 overflow with
0061 3115 OF_C2 CPFSEQ CA2H ; or Capture 2?
0062 1525 INCF T30FLCNTR,1 ; if CA2H = 0, over first
0063 C043 GOTO CAPTURE ; Increment counter
0064 3115 OF_C1 CPFSEQ CA2H ; first
0065 C043 GOTO CAPTURE ; Capture happened
0066 1525 INCF T30FLCNTR,1 ; Increment overflow counter
0067 C043 GOTO CAPTURE ; and do capture routine
0068 9826 FR0 BTFSC FLAG_REG,0 ; Between Capture 1
0069 1525 INCF T30FLCNTR,1 ; Yes, Inc. the over
Capture Module

006A 0005  RETFIE  ; Return from overf

006B 6A24  SUB16  MOVFP IC2AL,W  ; Do the 16-bit sub

006C 0522  SUBWF IC2BL,l  ;
006D 6A23  MOVFP IC2AH,W  ;
006E 0321  SUBWFB IC2BH,l  ;
006F 9004  BTFSS ALUSTA,0  ; Is the result pos

0070 8126  BSF FLAG_REG,1  ; neg., Set the und

0071 0002  RETURN  ; Return from the s

; Other Interrupt routines. (Not utilized in this example)

0072 0005  EXT_INT  RETFIE  ; RA0/INT interrupt

0073 0005  RTCCINT  RETFIE  ; (NOT used in this program)

0074 0005  RT_INT  RETFIE  ; RTCC overflow int

0075 C028  SRESET  GOTO START  ; (NOT used in this program)

; START and reinitalize.

; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.

ORG END_OF_PROG_MEM ;
07FF C075  GOTO SRESET ; The progr
END ; do a system reset
This is the basic outline for a program that can determine the
frequency of an input, via input capture. The input capture has
been selected to capture on the 16th rising edge. This is useful
for high frequency inputs, where an interrupt on each rising edge
would not be able to be serviced (at that rate). This particular
example can support an input signal with a period of approximately
625 nS. Without the divide by 16 selected, this is approximately
10 us. This period time increases (frequency decreases) as the
overhead in the main routine increases.

This routine uses an 8-bit register to count the times that timer3
overflowed. At the Max crystal frequency of 16 MHz, this gives an
overflow time of \((16(2^{8}+1)(2^{16})(250 \text{ nS}) > 67.37 \text{ sec.})\). If
measurement of longer time intervals is required, the overflow
counter could be extended to 16 (or more) bits.

Timer 3 in this example is a free running timer. The input
capture is generated on the RB1/CAP2 pin. There is a flag
that specifies if this is the 1st or 2nd capture.
The first capture is the start of the period measurement. The
second capture value gives the end of the period. In this type
of measurement If the 2nd capture value < the 1st capture value
then the overflow counter should be decremented.

Do the EQUate table

<table>
<thead>
<tr>
<th>LIST</th>
<th>P=17C42, T=ON, C=80</th>
</tr>
</thead>
<tbody>
<tr>
<td>0020</td>
<td>IC20F EQU 0x20 T3 overflow reg</td>
</tr>
<tr>
<td>0021</td>
<td>IC21H EQU 0x21 T3 ICA2 MSB reg</td>
</tr>
<tr>
<td>0022</td>
<td>IC22L EQU 0x22 T3 ICA2 LSB reg</td>
</tr>
<tr>
<td>0023</td>
<td>IC23H EQU 0x23 T3 ICB2 MSB reg</td>
</tr>
<tr>
<td>0024</td>
<td>IC24L EQU 0x24 T3 ICB2 LSB reg</td>
</tr>
<tr>
<td>0025</td>
<td>T3OFLCNTR EQU 0x25; Temperay T3 overflow</td>
</tr>
<tr>
<td>0026</td>
<td>FLAG_REG EQU 0x26; Register that has</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FLAG_REG bit</th>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAP1 0, 1st Capture</td>
<td></td>
</tr>
<tr>
<td>= 1, 2nd Capture</td>
<td></td>
</tr>
<tr>
<td>UFL 0, No Underflow</td>
<td></td>
</tr>
<tr>
<td>= 1, Underflow during subtract</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>END_OF_PROG_MEM EQU 0x07FF</th>
</tr>
</thead>
</table>

| ALUSTA EQU 0x04 |
| CPUSTA EQU 0x06 |
| INTSTA EQU 0x07 |
| W EQU 0x0A |
| PORTB EQU 0x12; Bank 0 |
; PIR EQU 0x16 ; Bank 1
PIE EQU 0x17
;
TMR3L EQU 0x12 ; Bank 2
TMR3H EQU 0x13
T3PRL EQU 0x16
T3PRH EQU 0x17
;
CA2L EQU 0x14 ; Bank 3
CA2H EQU 0x15
TCON1 EQU 0x16
TCON2 EQU 0x17

ORG 0x0000 ; Origin fo
GOTO START ; On reset,

ORG 0x0008 ; the program

ORG 0x0008 ; Origin fo

ORG 0x0010 ; interrupt vector

ORG 0x0010 ; Goto the

ORG 0x0010 ; on RA0/INT routine

ORG 0x0018 ; overflow interrupt vector

ORG 0x0018 ; Goto the

ORG 0x0018 ; Origin fo

ORG 0x0020 ; RA1/RT interrupt vector

ORG 0x0020 ; Goto the

ORG 0x0020 ; RA1/RT routine

ORG 0x0020 ; Origin fo

ORG 0x0028 ; of any enabled peripheral

ORG 0x0028 ; Goto the

ORG 0x0028 ; peripheral routine

ORG 0x0028 ; Origin fo

START BSF CPUSTA,4 ; program memory

GOTO EXT_INT ; Disable A

GOTO RTCCINT ; Global Interrupt Disable

GOTO RT_INT ; (GLINTD) bit.

GOTO PER_INT ; ; Place Mai

CLRF TCON2,0 ; Select re

MOVLW 0x0F0 ; Stop the

MOVWF TCON1 ; Initialize

MOVLC 3 ; ; T1 (8-b

MOVLB 3 ; and T3 run off the internal

; system clock. Capture2

; captures on the 16th rising edge.

; ; Initialize Timer 3, load the timer with the number of cycl

; the device executes (from RESET) before the timer is turn

; ;
; Therefore the offset is required due to software overhead

; 002D B802
; MOVLB  2           ; Select re
; 002E 280A
; CLRF  W,0          ; Clear the
; 002F 0126
; MOVWF  FLAG_REG    ; Initialize
; 0030 0113
; MOVWF  TMR3H       ; Timer3 MS
; 0031 B000
; MOVWL  0x00        ; Timer3 LS
; 0032 0112
; MOVWF  TMR3L       ;

; Load the Timer 3 period register with 0xFFFF, which will
; interrupt on the overflow of Timer3

; 0033 B0FF
; MOVWL  0xFF
; 0034 0117
; MOVWF  T3PRH       ;
; 0035 0116
; MOVWF  T3PRL       ;

; the timer should be started and interrupts enabled.

; 0036 B803
; MOVLB  3           ; Select re
; 0037 8217
; BSF  TCON2,2       ; Turn on t
; 0038 8307
; BSF  INTSTA,3      ; Turn on P
; 0039 B801
; MOVLB  1           ; Select re
; 003A B048
; MOVWL  0x48        ; Enable Ca
; 003B 0117
; MOVWF  PIE         ; Interr

; This is where you would do the things you wanted to do.
; this example will only loop waiting for the interrupts.

; 003C 8C06
; WAIT  BCF  CPUSTA,4 ; Enable AL
; 003D C03C
; GOTO  WAIT         ; Loop here

; The interrupt routine for any peripheral interrupt, This
; only deals with Timer3 (T3) interrupts.
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routi

; case1 - only T3 overflow = 12 cycles
; case2 - 1st capture = 14 cycles
; case3 - 2nd capture = 30 cycles
; case4 - T3 overflow and 1st capture = 34 cycles
; case5 - T3 overflow and 2nd capture = 50 cycles

; 003E B801
; PER_INT MOVLB  1    ; Select re
; 003F 9E16
; BTFSC  PIR,6       ; Did T3 ov
;                   ; If not skip next In-
Capture Module

This portion of the code takes the 1st capture and stores value in register pair IC2AH:IC2AL. When the 2nd capture is taken, its value is stored in register pair IC2BH:IC2BL. A 16-bit subtract is performed, with the final 24-bit result being stored in IC20F:IC2BH:IC2BL. This value will no longer be correct after the next capture occurs (IC2BH:IC2BL will change), so the main routine must utilize this value before it changes.

0043 8B16 CAPTURE BCF PIR,3 ; Clear Cap
0044 B803 MOVLB 3 ; Select reg
0045 9826 BTFSC FLAG_REG,0 ; 1st or 2nd
0046 C04B GOTO CAP2 ; It was th
0047 5424 CAP1 MOVPF CA2L,IC2AL ; Move the
0048 5523 MOVPF CA2H,IC2AH ; temporary
0049 8026 BSF FLAG_REG,0 ; Have 1st
004A 0005 RETFIE ; Return fr
004B 5422 CAP2 MOVPF CA2L,IC2BL ; Move the
004C 5521 MOVPF CA2H,IC2BH ; temporary
004D E061 CALL SUB16 ; (to prevent
004E 9926 BTFSC FLAG_REG,1 ; Call rout
004F 0725 DECF T30FLCNTR,1 ; 2 16-bit numbers.
0050 2926 CLRIF FLAG_REG,1 ; Underflow
0051 6A25 MOVFP T30FLCNTR,W ; Since und
0052 4A20 MOVFP W,IC2OF ; overflow counter.
0053 2825 CLRIF T30FLCNTR,0 ; Clear the
0054 0005 RETFIE ; underflow and

; counts how many
; overflows.
; Return fr
Capture Module

; When Timer 3 has overflowed, the overflow counter only sh
; be incremented when the overflow occurs after a capture 1
; but before the capture 2. The 4 possible cases when enter
; the T3OVL section of the PER_INT routine are as follows:
; Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting 
; for Capture 1 to occur). Do Not increment count
; Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting 
; for Capture 2 to occur). Increment counter
; Case 3: T3 Overflow happened after Capture. Do Not 
; increment overflow counter
; Case 4: T3 Overflow occurred before Capture 2 and FLAG_R

; (waiting for Capture 2 to occur). Increment cou

0055 8E16 T3OVL BCF PIR,6 ; Clear Ove
0056 9316 BTFSS PIR,3 ; Did the R
0057 C05E GOTO FR0 ; cause an interrupt?
0058 B803 MOVLB 3 ; and 2nd capture
0059 280A CLR W,0 ; Bank 3
005A 3115 CPFSEQ CA2H ; W = 0
005B C05E GOTO FR0 ; if CA2H =
005C B801 MOVLB 1 ; first,
005D C043 GOTO CAPTURE ; bit 0
005E 9826 FR0 BTFSC FLAG_REG,0 ; Back to b
005F 1525 INC T3OVL_CNTR,1 ; Increment overflow
0060 0005 RETFIE ; and do capture routine
0061 6A24 SUB16 MOVFP IC2AL,W ; Between C
0062 0522 SUBWT IC2BL,1 ;
0063 6A23 MOVFP IC2AH,W ;
0064 0321 SUBWF IC2BH,1 ;
0065 9004 BTSS ALUSTA,0 ;
0066 8126 BSF FLAG_REG,1 ;
0067 0002 RETURN ;

; Other Interrupt routines. (Not utilized in this example)

0068 0005 EXT_INT RETFIE ; RA0/INT 1

; (NOT used in this
Capture Module

0069 0005 RTCCINT RETFIE ; RTCC over
006A 0005 RT INT RETFIE ; (NOT used in this
                          ; RA1/RT in
                          ; (NOT used in this program)
                          ; If progra
                          ; START and reinitalize.

006B C028 SRESET GOTO START

; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1’s (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.

07FF C06B ORG END_OF_PROG_MEM
          GOTO SRESET ; The progr
          END

Errors : 0
Warnings : 0
INTRODUCTION

PIC17C42 has an on chip high speed Universal Synchronous Asynchronous Receiver Transmitter (USART). The serial port can be configured to operate either in full-duplex asynchronous mode or half duplex synchronous mode. The serial port has a dedicated 8-bit baud rate generator. Either 8- or 9-bits can be transmitted/received.

This application note provides information on using the serial port, parity generation, serial port expansion, RS-232 interface, I/O port expansion using synchronous mode of serial port.

SERIAL PORT USAGE

Brief code to setup serial port, receive and transmit data is given below. Small sections of code for both asynchronous and synchronous mode is given.

EXAMPLE 1

Asynchronous Mode Setup

Asynchronous mode set-up requires selection of 8/9-bits of data transfer, baud rate, setting the baud rate generator and configuring the TXSTA & RCSTA control registers. The baud rate generator is set by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

\[ SPBRG = \frac{\text{Input}_\text{Clk}_\text{Freq}}{64 \times \text{Baud}_\text{Rate}} - 1 \]

For example, to select a baud rate of 9600 bits/sec with input clock frequency of 16 Mhz, SPBRG is computed from the above equation to be 25. Once the Baud Rate Generator is setup, it is necessary to configure the TXSTA & RCSTA control registers as follows (please refer to the data sheet):

\[ \text{SPBRG} = \frac{16000000}{64 \times 9600} - 1 \]

\[ \text{TXSTA} = \text{INIT} \ 0x20 \]

\[ \text{RCS TA} = \text{INIT} \ 0x90 \]

Sample Code For Asynchronous Mode Serial Port Setup

```c
#include "17C42.h"

Setup_Async_Mode
    movlb 0
    movlw baud(9600) // equals 25 for 9600 baud
    movwf SPBRG // baud rate generator is reset & initialized
    movlw TXSTA_INIT
    movwf TXSTA // 8-bit transmission, async mode
    movlw RCSTA_INIT
    movwf RCSTA // 8-bit reception, enable serial port, enable reception

    return
```

© 1993 Microchip Technology Inc.
Synchronous Mode Setup

Synchronous mode setup requires selection of 8/9-bits of data transfer, bit rate, setting the baud rate generator and configuring the TXSTA & RCSTA control registers. The baud rate generator is set by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

\[
\text{SPBRG} = \frac{\ln(\text{Input Clock Frequency})}{4 \cdot \text{Baud Rate}}.
\]

**EXAMPLE 2**

For example, to select a bit rate of 1Mhz with input clock frequency of 16 Mhz, SPBRG is computed from the above equation to be 3. Once the Baud Rate Generator is setup, it is necessary to configure the TXSTA & RCSTA control registers as follows (please refer to the data sheet):

\[
\text{SPBRG} = \frac{\ln(\text{Input Clock Frequency})}{4 \cdot \text{Baud Rate}}.
\]

<table>
<thead>
<tr>
<th>SPBRG</th>
<th>TXSTA</th>
<th>RCSTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10110000 (B0h)</td>
<td>10010000 (90h)</td>
</tr>
</tbody>
</table>

8-bit transmission, Sync mode (MASTER)
8-bit reception, enable serial port, continuous reception

Sample Code For Synchronous Mode (MASTER) Serial Port Setup

```c
#define ClkFreq 16000000 ; input clock frequency = 16 Mhz
#define baud(X) (((10*ClkFreq/(4*X))+5)/10 - 1
#define TXSTA_INIT OxB0
#define RCTSA_INIT Ox90

#include "17C42.h"

Setup_Sync_Master_Mode

movlb 0 ; SPBRG, TXSTA & RCSTA are in bank 0
movlw baud(1000000) ; equals 3 for 1 Mbits/sec
movwf SPBRG ; baud rate generator is reset & initialized
movlw TXSTA_INIT
movwf TXSTA ; 8-bit transmission, async mode
movlw RCTSTA_INIT
movwf RCSTA ; 8-bit reception, enable serial port,
return ; enable reception
```

SPBRG, TXSTA & RCSTA are in bank 0
SPBRG, TXSTA & RCSTA are in bank 0
Receiving Data (Software Polling)

The sample code provides a way to read the received serial data by software polling (with no serial port interrupts). This applies to both asynchronous and synchronous mode. Software polling is done by checking the RBFL bit (PIR<0>). If this bit is set it means that a word has been received (8 bits are in RCREG and the 9th bit in RCSTA<0>).

EXAMPLE 3

```asm
;*******************************************************************
; Return The 8-bit received Data By Software Polling
; The received data is returned in location SerInData
;*******************************************************************
Get_Serial_Data_Poll
    movlb 1           ; PIR is in bank 1
    PollRcv
        btfss PIR, 0  ; check the RBFL bit
        goto PollRcv ; loop until char received, assume WDT is off
        movlb 0       ; PIR is in bank 1
    movpf RCREG, SerInData
    return          ; Received 8-bits are in SerInData
;*******************************************************************
```

Transmitting Data (Software Polling)

The sample code provides a way to transmit serial data by software polling (no serial port interrupts). Software polling is done by checking the TBMT bit (PIR<0> in bank 1) to be one, indicating the transfer of TXREG to the serial shift register.

EXAMPLE 4

```asm
;*******************************************************************
; Transmit 8-bit Data By Software Polling
; The data to be transmitted is in location SerOutData
;*******************************************************************
Send_Serial_Data_Poll
    movlb 1           ; PIR is in bank 1
    PollTxmt
        btfss PIR, 1  ; check the TBMT bit of PIR register in bank1
        goto PollTxmt ; loop until char received, assume WDT is off
        movlb 0       ; PIR is in bank 1
    movfp SerOutData, TXREG
    return          ; Received 8-bits are in SerInData
;*******************************************************************
```
Transmitting & Receiving A Block Of Data (Interrupt Driven)

A general purpose routine which is interrupt driven that transmits and receives a block of data is provided. The reception or transmission of the block is ended when an end of block character is detected. As an example, the end of block is identified by a 0. The block of data to be transmitted is stored in the program memory and TABLRD instruction is used to transfer this example data to the file registers and serial port. The user may modify this code to a more general purpose routine that suits his application.

EXAMPLE 5

MPASM B0.54

```assembly
; TITLE 'Serial Interface Routines
LIST P=17C42, C=80. I=ON, R=DEC

; This is a short program to demonstrate how to transmit and serial data using the PIC17C42.
; A message will be transmitted and routed right back to the and read. The read information will be saved in an interna
;
; include "pl7reg.h"

ORG 0x0010
goto start

org 0x0010 ;vector for rtcc interrupt

rtcc_int

org 0x0020 ;vector for peripheral inte

perf_int
go to service_perf ;service the interrupts

org 0x0030

;initialize the serial port: baud rate interrupts etc.
init_serial

0030 2922 clrf SERFLAG ;clear all flags
0031 B800 movib 0 ;select bank 0
0032 B007 movlw 0x07 ;select 9600 baud
0033 770A movfp W,SPBRG ;/
0034 B090 movlw 0x90 ;set up serial pins
0035 730A movfp W,RCSTA ;/
```
Serial Port Utilities

0036 2915
0037 B801 clrf TXSTA ;setup transmit status
0038 2916 movlb 1 ;select bank 1
0039 2917 clrf PIR ;clear all interrupts
003A 8017 clrf PIE ;clear all enables
003B B801 bsf PIE,RCIE ;enable receive interrupt
003C 4A20 movlw RX_BUFFER ;set pointer to rx buffer
003D 2917 movfp W,RXPRK ;
/ 003E 2907 clrf INTSTA ;clear all interrupts
003F 8307 movbf INTSTA,FEIE ;enable peripheral ints
retfie

; start transmission of first two bytes
start_xmit
0040 B800 movlb 0 ;select bank 0
0041 8515 bsf TXSTA,TXEN ;enable transmit
0042 A80A tablda 1,1,W ;load latch
0043 A216 tlrd 1,TXREG ;load high byte
0044 B801 movlb 1 ;select bank 1
empty_chk
0045 9116 btfsf PIR,TBMT ;TXBUF empty?
0046 C045 goto empty_chk ;no then keep checking
0047 B800 movlb 0 ;select bank 0
0048 A916 tablda 0,1,TXREG ;load lo byte
0049 B801 movlb 1 ;select bank 1
004A 8117 bsf PIE,TXIE ;enable transmit interrupts
004B 8222 bsf SERFLAG,HILOB ;set up next for high byte
004C 0002 return

; service_perf
;check for transmit or receive interrupts only
004D 9916 btfsf PIR,REIE ;RX buffer full?
004E C062 goto service_recv ;yes then service
004F 9116 btfsf PIR,TBMT ;TX buffer empty?
0050 C060 goto exit_perf ;no, ignore other int.

service_xmt
0051 9822 btfsf SERFLAG,TDONE ;all done?
0052 C060 goto exit_perf ;yes then quit
0053 9A22 btfsf SERFLAG,HILOB ;if clr, do low byte
0054 C057 goto rd_hi ;else read high byte
0055 A30A tablda 0,1,W ;read lo
0056 C058 goto sx_cont ;continue
rd_hi
0057 A20A tlrda 1,W ;read high byte
sx_cont
0058 3A22 btg SERFLAG,HILOB ;toggle flag
0059 B800 movlb 0 ;bar=0
005A 4A16 movfp W,TXREG ;load tx reg
005B 330A tstfsz W ;last byte?
005C C060 goto exit_perf ;no then cont
end_xmt
005D B801 movlb 1 ;select bank 1
005E 8917 bcf PIE,TXIE ;disable tx interrupt
005F 8022 bsf SERFLAG,TDONE ;set done flag
exit_perf
0060 8F07 bcf INTSTA,PEIR ;clear peripheral int
0061 0005 retfie

; service_recv
0062 9922 btfsf SERFLAG,RXDONE ;RX complete?
0063 C060 goto exit_perf ;exit int
0064 6120 movfp RXPTR,FSR0 ;get pointer
0065 B800 movlb 0 ;select bank 0
0066 6014 movfp RCREG,F0 ;load received value
0067 290A clrf W ;clr W
0068 3200 cpfsig F0 ;value = 0?
Serial Port Utilities

0069 C06D     goto end_recv ;yes then end
006A 1501     incf FSR0 ;inc pointer
006B 4120     movpf FSR0, RXPTR ;save pointer
006C C060     goto exit_perf ;return from int

end_recv
006D 8122     bsf SERFLAG, RXDONE ;set flag
006E 2907     clrf INTSTA ;clear all int
006F B801     movlb 1 ;select bank 1
0070 8817     bcf PIE, RCIE ;disable rx interrupts
0071 C060     goto exit_perf ;return

; start
0072 2909     clrf FSR1 ;assign FSR1 as S.P.
0073 0709     decf FSR1 ;
0074 B020     movlw 0x20 ;clear ram space
0075 610A     movfp W, FSR0 ;do indirect addressing

startl
0076 2900     clrf F0 ;clear ram
0077 1F01     incfsz FSR0 ;inc and skip if done
0078 C076     goto startl
0079 E030     call init_serial ;initialize serial port

Warning: MESSAGE not a single byte quantity
007A B000     movlw MESSAGE ;load table pointer
007B 4A00     movpf W, TBLPTRL ;
007C B001     movlw page MESSAGE ;
007D 4A0E     movpf W, TBLPTRH ;
007E E040     call start_xmit ;start transmission

chk_end
007F 9122     btfss SERFLAG, RXDONE ;receive all?
0080 C07F     goto chk_end ;no then keep checking

; loop
go to loop ;spin wheel

ORG 0x100

MESSAGE
0100 5468 6520 636F DATA "The code is: Tea for the Tillerman"
0103 6665 2069 733A
0106 2054 6561 726D
0109 6F72 2074 6865
010C 2054 696C 6C65
010F 726D 616E
0111 0000     DATA 0

END

Errors : 0
Warnings : 0
PARITY GENERATION

Since the serial port of PIC17C42 does not have an on-chip parity generator, parity is generated using software. It takes only 10 program memory words and executes in 10 instruction cycles to generate parity. Since the serial port of PIC17C42 can operate in a 9-bit mode, the parity bit can be generated in software and transmitted as the 9th bit or be compared with the received 9th bit.

In case of transmission, set TX8/9 to 1 (<6> of TXSTA) to enable 9-bit transmission and write the computed parity bit to TXD8 (TXSTA<0>). The 9th bit (parity bit) must be written prior to writing the 8 data bits to TXREG.

In case of reception, first of all enable 9-bit reception by setting RC8/9 to 1 (RCSTA<6>). Upon successful reception, the 9 bit is received in RCD8 (RCSTA<0>). Parity of the 8 bits of received data is computed using the routine listed below and compared with the 9-bit received.

EXAMPLE 6

```
;******************************************************************************
; Generate Parity Bit for the 8 bit register 'data'
; The parity bit is stored in Bit 0 of 'parity'
;
;******************************************************************************

#define ODD_PARITY FALSE

swapf data,w
xorwf data,w
movwf parity
rrncf parity
rrncf parity
xorwf parity,w
andlw Ox03
addlw Ox01
rrncf wreg
movwf parity
#if ODD_PARITY
btg parity, 0
#endif
```
SERIAL PORT UTILITIES

SERIAL PORT EXPANSION

The PIC17C42 has only one serial port. For applications that require the PIC17C42 to communicate with multiple serial ports, a scheme the multiplexes and demultiplexes the RX and TX pins is provided below. This method is suited only if no more than one UART is needed at any one time. This is the case in many applications where the microcontroller drives several outputs devices serially. Figure 1 shown below suggests a way to expand the on-chip serial port to 4 serial ports. To use the scheme as shown in Figure 1, The PIC17C42 must select the desired serial port by appropriately setting the two pins of PORT-B. The same scheme may be used to further expand the serial ports by using more I/O Ports.

FIGURE 1 - MULTIPLEXING THE ON-CHIP UART

RS-232 INTERFACE

Two circuits are provided to interface the CMOS levels of PIC17C42 to RS-232 levels. Figure 2 provides interface to MAX232 (MAXIM's RS-232 Driver/Receiver) with a single +5V power supply. Figure 3 provides a low cost 2 chip solution for RS-232 level translation using a single +5V supply (Note that V- of MC14C88 is connected to DTR of RS-232 Interface. By asserting DTR to low, V- gets the negative voltage from the RS-232 line). An alternative single chip low cost solution is provided in Figure 4. However 3 voltage sources (+5, +12, -12) are necessary.

FIGURE 2 - RS-232 INTERFACE TO MAX232

FIGURE 3 - LOW COST 2 CHIP SOLUTION USING SINGLE POWER SOURCE

FIGURE 4 - LOW COST SINGLE CHIP SOLUTION USING 3 POWER SOURCES
Table 1 provides the summary of RS-232 and V.28 Electrical Specifications.

**TABLE 1 - SUMMARY OF RS-232C AND V.28 ELECTRICAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Driver Output Voltage</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 level</td>
<td>+5V to +15V</td>
<td>With 3-7kΩ load</td>
</tr>
<tr>
<td>1 level</td>
<td>-5V to -15V</td>
<td>with 3-7kΩ load</td>
</tr>
<tr>
<td>Max. output</td>
<td>±25V</td>
<td>No load</td>
</tr>
<tr>
<td><strong>Receiver input Thresholds</strong></td>
<td>(Data and clock signals)</td>
<td></td>
</tr>
<tr>
<td>0 level</td>
<td>+3V to +25V</td>
<td></td>
</tr>
<tr>
<td>1 level</td>
<td>-3V to -25V</td>
<td></td>
</tr>
<tr>
<td><strong>Receiver Thresholds</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS, DSR, DTR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>On level</td>
<td>+3V to +25V</td>
<td>Detects power Off condition at driver</td>
</tr>
<tr>
<td>Off level</td>
<td>Open circuit or -3V to -25V</td>
<td></td>
</tr>
<tr>
<td><strong>Driver output resistance</strong></td>
<td>3kΩ to 7kΩ</td>
<td></td>
</tr>
<tr>
<td><strong>Power off condition</strong></td>
<td>300Ω Min.</td>
<td></td>
</tr>
<tr>
<td><strong>Driver slew rate</strong></td>
<td>30V/µs max.</td>
<td></td>
</tr>
<tr>
<td><strong>Signalling rate</strong></td>
<td>Up to 20k bits/sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Cable length</strong></td>
<td>50/75m.</td>
<td>Longer cables permissable, if CLoad ≤ 2500pF</td>
</tr>
</tbody>
</table>

**EXAMPLE 1**

```assembly
InitSerialPortTxmt
    movlb 0
    clrf SPBRG ; set to highest baud rate = CLKOUT = CLkin/4
    movlw Ox80
    movwf RCSTA ; enable serial port
    movlw OxB0
    movwf TXSTA ; 8 bit synchronous master mode
    bcf DDRB,0 ; set bit 0 of PortB as output, to be used as Latch Clk
    return
SendSerialData ; shift out DataLo & DataHi serially
    movlb 0
    movfp DataLo,TXREG
    nop ; wait for TXREG transfer : if slower baud rate is
    ; used check for TBMT reset
    movfp DataHi,TXREG
    wait btfss TXSTA,TRMT ; wait until all 16 bits shifted out
    goto wait
    bcf PortB,0 ; clock in the serial data to parallel output of HC595
    return
```

**I/O PORT EXPANSION USING SYNCHRONOUS MODE**

Although the PIC17C42 has 33 I/O pins, most of these are multiplexed with other peripheral functions. In case more I/O ports are needed, the scheme provided below expands the I/O port using the synchronous mode of serial port by serially shifting the data. Figure 5 shows a scheme to expand the output ports to 16-bits using 2 standard logic chips (74HC595). The PIC17C42's serial port is configured in synchronous mode and set to be the MASTER. Thus serial data is available on DT (pin 22) and the clock is available on CK (pin 21). The following code will transmit 16-bits serially and clock all the 16-bits at the same time.

**FIGURE 5 - OUTPUT PORT EXPANSION USING SYNCHRONOUS MODE**
A similar scheme as shown above may be implemented in the reverse way to expand Input Ports. For this it is necessary to have a Parallel In and serial out device. Using a standard logic chip (74HC165), the scheme is shown in Figure 6. In order to read the 16-bit input data, the serial port of PIC17C42 is configured to be in Synchronous Mode Reception (MASTER mode). An I/O port (PortB<0>) is used to parallel load the 16 inputs for reading serially. A sample code to read the 16 inputs is shown below.

**EXAMPLE 2**

**InitSerialPortRcv**

```assembly
  movlb 0
  clrf SPBRG ; set to highest baud rate = CLKOUT = CLKN/4
  movlw 0x80
  movwf RCSTA ; enable serial port
  movlw 0x90
  movwf TXSTA ; 8-bit synchronous master mode
  bcf DDRB, 0 ; bit 0 of PortB is output, to be used as Parallel Load
  bsf PortB, 0 ; disable parallel Load
  return
```

**ReadSerialData**

```assembly
  movlb 0 ; shift out DataLo & DataHi serially
  bcf PortB, 0 ; Parallel Load The Inputs into 74HC165
  bsf PortB, 0 ; disable parallel Load
  bsf RCSTA, SREN ; enable single byte reception and wait for data
  movlb 1
  wait1 btfs PIR, RBFL
  goto wait1 ; check until 8-bits are received

  movlb 0
  movwf RCREG, DataLo ; 1st byte is read
  movlb 0
  bsf RCSTA, SREN ; enable another byte of reception and wait for data

  movlb 1
  wait2 btfs PIR, RBFL
  goto wait2 ; check until 8-bits are received

  movlb 0
  movwf RCREG, DataHi ; 2nd byte is read
  return
```

**FIGURE 6 - INPUT PORT EXPANSION USING SYNCHRONOUS MODE**

![Diagram of input port expansion using 74HC165](Image)

**Author:** Amar Palacherla  
**Logic Products Division**
INTRODUCTION

This application note provides some utility math routines for Microchip's second generation of high performance 8-bit microcontroller, the PIC17C42. Three assembly language modules are provided, namely ARITH.ASM, FLOAT.ASM and BCD.ASM. Currently in each file the following subroutines are implemented:

ARITH.ASM
• Single precision 8 x 8 unsigned multiply
• 16 x 16 double precision multiply (signed or unsigned)
• 16 / 16 double precision divide (signed or unsigned)
• 16 x 16 double precision addition
• 16 x 16 double precision subtraction
• double precision square root
• double precision numerical differentiation
• double precision numerical integration
• Pseudo Random number generation
• Gaussian distributed random number generation

FLOAT.ASM
• Binary floating point addition
• Binary floating point subtraction
• Binary floating point multiplication

BCD.ASM
• 8-bit binary to 2 digit BCD conversion
• 16-bit binary to 5 digit BCD conversion
• 5-bit BCD to 16-bit binary conversion
• 2 digit BCD addition

As more routines are available, they will be added to the library. The latest routines may be obtained either through Microchip's bulletin board or by contacting your nearest Microchip sales office for a copy on a MS-DOS 5 1/4" floppy.

These routines have been optimized wherever possible with a compromise between speed, RAM utilization, and code size. Some routines (multiplication and division) are provided in two forms, one optimized for speed and the other optimized for code size.

All the routines have been implemented as callable subroutines and the usage of each routine is explained below. At the end of the application note, the listing files of the above programs are given.

SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 1.

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of two 1-bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic is provided (2’s complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code). These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 2.

TABLE 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpy8x8_F</td>
<td>speed efficient</td>
<td>36</td>
<td>36</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>mpy8x8_S</td>
<td>code efficient</td>
<td>13</td>
<td>69</td>
<td>1</td>
<td>used</td>
</tr>
</tbody>
</table>
Math Routines

### TABLE 2

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_mpyF</td>
<td>Speed Efficient, Signed Arithmetic</td>
<td>204</td>
<td>183</td>
<td>1</td>
<td>used</td>
</tr>
<tr>
<td>D_mpyF</td>
<td>Speed Efficient, Unsigned Arithmetic</td>
<td>179</td>
<td>176</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>D_mpyS</td>
<td>Code Efficient, Signed Arithmetic</td>
<td>52</td>
<td>254</td>
<td>4</td>
<td>used</td>
</tr>
<tr>
<td>D_mpyS</td>
<td>Code Efficient, Unsigned Arithmetic</td>
<td>21</td>
<td>242</td>
<td>3</td>
<td>used</td>
</tr>
</tbody>
</table>

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assemble are used.

The data memory organization is explained in the comment section of the code. Faster execution and code space saving can be achieved by setting "MODE_FAST equ TRUE". However, setting MODE_FAST variable to TRUE restricts that operands and the 32-bit result be in data RAM locations 0x18 and 0x1F (in this mode, MOVFP and MOVPF instructions may be used to transfer data to/from any RAM location to addresses less than 0x1F). If MODE_FAST is set to FALSE, there will be no restriction on the location of the data RAM values used in this subroutine. However, the code will be slightly slower and occupies more program memory.

### TABLE 3

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_divF</td>
<td>Speed Efficient, Unsigned Arithmetic</td>
<td>325</td>
<td>250</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>D_divF</td>
<td>Speed Efficient, Signed Arithmetic</td>
<td>354</td>
<td>260</td>
<td>1</td>
<td>used</td>
</tr>
<tr>
<td>D_divS</td>
<td>Code Efficient, Signed Arithmetic</td>
<td>31</td>
<td>300</td>
<td>1</td>
<td>used</td>
</tr>
<tr>
<td>D_divS</td>
<td>Code Efficient, Unsigned Arithmetic</td>
<td>39</td>
<td>312</td>
<td>2</td>
<td>used</td>
</tr>
</tbody>
</table>

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

### DOUBLE PRECISION DIVISION

This routine performs a 2's complement division of two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code).

These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differenciator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 3.

### DOUBLE PRECISION ADDITION AND SUBTRACTION

Two routines are provided. One performs a 2's complement addition and the other one performs a 2's complement subtraction of two 16-bit binary numbers. These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 4.

### TABLE 4

<table>
<thead>
<tr>
<th>Name</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dadd</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>Dsub</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>used</td>
</tr>
</tbody>
</table>
NEGATE A DOUBLE PRECISION NUMBER

These routines negate a double precision number (16-bit and 32-bit). Two routines and two macros are provided to negate a 16-bit number. The subroutines use indirect addressing mode and the macros use direct addressing scheme. A macro is provided to negate a 32-bit number.

FLOATING POINT ROUTINES

Three binary floating point routines are implemented: addition, subtraction and multiplication. To use these routines, the floating point numbers should be represented as follows:

a. A 16-bit signed Mantissa (in 2's complement form)
b. An 80-bit signed binary exponent. For example, a number "NUM" is represented as follows:

NUM = (<± 16-bit Mantissa>) * (2**<± 8-bit Exponent)

Also, a general purpose Normalization routine is provided and it is recommended that the user call the normalization routine as often as possible to avoid loss of precision. The normalization routine maximizes the number of bits in the mantissa (until there are at least 14-bits).

In case of multiplication, if a 32-bit product is desired (with an 8 bit exponent) the "Mode16" should be set to TRUE. It is recommended that "Mode16" be set to FALSE (this would always produce a 16-bit mantissa, with an 8-bit exponent which is the general floating point format). Only the looped code versions are implemented. In the performance specs, the execution time is not specified, as the time very much depends on the values of the numbers and whether they are normalized or not.

DOUBLE PRECISION SQUARE ROOT

Often in many applications, one needs to find the square root of a number. Of the many numerical methods available to compute the square root of a number, the Newton-Raphson method is one of the most attractive because of its fast convergence rate. In this method, the square root of number, N, is obtained as an approximate solution of

\[ f(Y) = Y^2 - N = 0 \]

The function \( f(Y) \) can be expanded about \( Y_0 \) using the first order Taylor polynomial expansion as:

Equation 1:

\[ f(Y) = f(Y_0) + (Y - Y_0)f'(Y_0) + \frac{(Y - Y_0)^2}{2!}f''(Y_0) + \ldots \]

If \( X \) is a root of \( f(Y) \), then \( f(X) = 0 \): Therefore,

\[ f(X) = f(Y_0) + (X - Y_0)f'(Y_0) + \frac{(X - Y_0)^2}{2!}f''(Y_0) + \ldots = 0 \]

If \( Y_0 \) is an approximate root of \( f(Y) \), then the higher order terms in the above equation are negligible.

Therefore, \( f(Y_0) + (X - Y_0)f'(Y_0) = 0 \)

i.e., \( X = Y_0 + \frac{f(Y_0)}{f'(Y_0)} \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negate</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>unused</td>
</tr>
<tr>
<td>NegateAlt</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>NegMac</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>AltNegMac</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>unused</td>
</tr>
<tr>
<td>NegMac32 (32 bit)</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>used</td>
</tr>
</tbody>
</table>

TABLE 6

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_add</td>
<td>Addition</td>
<td>54</td>
<td>5</td>
<td>used</td>
</tr>
<tr>
<td>F_sub</td>
<td>Subtraction</td>
<td>58</td>
<td>5</td>
<td>used</td>
</tr>
<tr>
<td>F_mpy</td>
<td>Multiplication</td>
<td>114</td>
<td>6</td>
<td>used</td>
</tr>
</tbody>
</table>
Thus X is a better approximation for Y0. From the previous equation, the sequence \(\{X_n\}\) can be generated:

\[
Equation 2: \quad X_n = X_{n+1} - \frac{f(X_{n+1})}{f'(X_{n+1})}, \quad n > 1
\]

For our case, equation 2, reduces to:

\[
Equation 3: \quad X_{n+1} = X_n + \frac{N}{X_n}
\]

The routine "Sqrt" in ARITH.ASM implements the above equation. Equation 3 requires that at first an initial approximation for the root is known. The better the initial approximation, the faster the convergence rate would be. In the "Sqrt" routine, the initial approximation root is set as \(N/2\). This routine calls the double precision division routine (D_divS).

In the code size, the Division routine (Ddiv_S) size is not included.

**BCD ROUTINES**

Three routines are provided for general purpose BCD arithmetic:

a. BCD to binary conversion
b. Binary to BCD conversion
c. BCD addition

The BCD to binary conversion routine converts a 5 digit BCD code to a 16 bit binary number. The BCD addition routine adds two BCD digits directly without converting them at first to binary. Note the usage of the "DAW" instruction. The other two routines convert a binary number to a BCD code. The performance specs for the BCD routines is given in the Table 8 below:

**NUMERICAL DIFFERENTIATION**

This routine performs numerical differentiation of a sequence of data if the input sequence is assumed to be piecewise linear with no discontinuances (this is the case in most of the real world signals). Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. This routine uses the so called 3 Point formula to compute the differential of a sequence of numbers.

Given an equation \(f(t)\), its derivative is given by

\[
f'(t) = \frac{df(t)}{dt}
\]

The above equation can be approximated using the 3-Point formula as given below:

\[
3-Point Formula:
\]

\[
f'(t) = f(t_0 + 3h) - 4f(t_0 + 2h) + 3f(t_0)
\]

where \(t_0\) is the point at which the numerical derivative is desired and "h" is the step size. The smaller the value of the step size (h), the better the approximation. In case of say, PID motor control, the step size is proportional to the time intervals at which the new sample value of the position (or speed) is obtained. Using the above equation to compute the differential, three samples are necessary (present value and the last two past values). The subroutine "Diff" is implemented so that \(1/2h\) factor is stored already in a RAM location (location DiffK) as \(1/2h\) and not as "h" because it is more efficient to multiply than divide.

After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, differentiation may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Diff" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

**TABLE 7**

<table>
<thead>
<tr>
<th>Name</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sqrt</td>
<td>22</td>
<td>3300 (approx.)</td>
<td>6</td>
<td>used</td>
</tr>
</tbody>
</table>

**TABLE 8**

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDtoB</td>
<td>BCD to Binary</td>
<td>30</td>
<td>112</td>
<td>0</td>
<td>used</td>
</tr>
<tr>
<td>B2_BCD_Looped</td>
<td>Binary to BCD (16 bit)</td>
<td>32</td>
<td>750</td>
<td>1</td>
<td>used</td>
</tr>
<tr>
<td>B2_BCD_Straight</td>
<td>Binary to BCD (16 bit)</td>
<td>44</td>
<td>572</td>
<td>1</td>
<td>used</td>
</tr>
<tr>
<td>BinBCD</td>
<td>Binary to BCD (8 bit)</td>
<td>10</td>
<td>62</td>
<td>0</td>
<td>unused</td>
</tr>
<tr>
<td>BCDAdd</td>
<td>BCD addition</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>used</td>
</tr>
</tbody>
</table>
NUMERICAL INTEGRATION

This routine performs numerical integration using Simpson's Three-Eighths Rule. This is a third order approximation for the function, whose integral is to be computed at a given point. Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. Given a function \( f(t) \), its integral over a range \( t_0 \) to \( t_3 \) is represented as:

\[
\int_{t_0}^{t_3} f(t) \, dt
\]

This function is approximated as follows:

Simpson's Three-Eighths Rule:

\[
\int_{t_0}^{t_3} f(t) \, dt = \frac{3h}{8} \left[ f(t_0) + 3f(t_1) + 3f(t_2) + f(t_3) \right]
\]

The constant \( \frac{3h}{8} \) can be computed beforehand and stored in a RAM location (in location IntgKLo and IntgKHi as a 16 bit number). After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, integration may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Integrate" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

TABLE 9

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diff</td>
<td>Numerical Differentiation</td>
<td>34</td>
<td>365</td>
<td>10</td>
<td>used</td>
</tr>
</tbody>
</table>

TABLE 10

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrate</td>
<td>Numerical Integration</td>
<td>39</td>
<td>370</td>
<td>12</td>
<td>used</td>
</tr>
</tbody>
</table>
FIGURE 2 - AUTOCORELATION OF THE DATA POINTS GENERATED BY THE RANDOM NUMBER GENERATOR

Autocorrelation of Data from Random Number Generator

![Autocorrelation Plot]

TABLE 11

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random16</td>
<td>Pseudo Random Number Generator</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>used</td>
</tr>
</tbody>
</table>

TABLE 12

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Program Memory</th>
<th>Instruction Cycles</th>
<th>Scratch RAM</th>
<th>W Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss</td>
<td>Gaussian Random Number Generator</td>
<td>21</td>
<td>452</td>
<td>4</td>
<td>used</td>
</tr>
</tbody>
</table>

PN (pseudo noise) sequences are widely used in digital communication systems for synchronization. These code words can also be used for data scrambling because of their good correlation properties. An interesting application of these sequences is system integrity. For example, these sequences can be regularly transmitted to a processor whole watch dog timer will time out if, say, two consecutive PN sequences do not match.

FIGURE 3 - HISTOGRAM OF THE DATA GENERATED BY THE GAUSSIAN RANDOM NUMBER GENERATOR

Gaussian Distributed Random Number Generator

This routine (subroutine "Gauss" provided in ARITH.ASM) generates a sequence of random numbers with a characteristic of a normal distribution (Gaussian distributed points). This routine calls the pseudo random number generator ("random16") to obtain a near uniformly distributed random points and from these points, the Gaussian distributed points are generated. The method of generating Gaussian points is based on the "Central Limit Theorem", which states that an ensemble of average weighted sum of a sequence of uncorrelated samples tends to have a Gaussian distribution.

As a test, the Gaussian points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC series). A plot of the points captured is shown in Figure 3, which shows that the random points generated have the characteristics of a Gaussian distribution.

Author: Amar Palacherla
Logic Products Division

© 1993 Microchip Technology Incorporated
APPENDIX A: GENERAL PURPOSE MATH ROUTINES
LISTING FILE OF ARITH.ASM

0001  #define PAGE  EJECT
0000  
0001  TITLE  "General Purpose Math Routines For PIC17C42 : Ver 1.0"
0001  LIST  P=17C42, C=120, T=ON, I=0, R=DEC

include "17c42.h"

;*******************************************************************
; Define RAM Locations necessary For the "ARITH.ASM"
; RAM locations should be defined before calling the library math
; routines
;*******************************************************************

if MODE_FAST

CBLOCK 0x18

ACCaLO, ACCaHI, ACCbLO, ACCbHI ; Ram Locations for Arithmetic
ACCcLO, ACCcHI, ACCdLO, ACCdHI ; Routines

ENDC

else

CBLOCK 0x20

ACCaLO, ACCaHI, ACCbLO, ACCbHI
ACCcLO, ACCcHI, ACCdLO, ACCdHI

ENDC

endif

; CBLOCK

tempLo, tempHi, count, sign

ENDC

CBLOCK

NumLo, NumHi

iterCnt

ENDC

; CBLOCK ; RAM locations for "Diff" routine

0027 0003

XnLo, XnHi, Xn_1_Lo

002A 0003

Xn_1_Hi, Xn_2_Lo, Xn_2_Hi

002D 0002

DiffKLo, DiffKHi

002F 0002

DiffLo, DiffHi

ENDC

; CBLOCK ; RAM Locations for "Integrate"

0031 0004

X0Lo, X0Hi, X1Lo, X1Hi

0035 0004

X2Lo, X2Hi, X3Lo, X3Hi

0039 0002

IntgKLo, IntgKHi

003B 0002

IntgLo, IntgHi

; INTEGRATE CONST = 3*h/8
ENDC

;****************************************************************************
mu lend
mulplr equ ACCaHI
L-byte equ ACCbLO
H-byte equ ACCbHI

;_LUPCNT equ 10 ; Set Desired Number of iterations
SqrtLo equ ACCdLO ; for Square Root Routine (NEWTON Iterations)
SqrtHi equ ACCdHI

; Define RAM locations for the Random Number Generators
RandLo equ ACCaLO
RandHi equ ACCaHI ; 16 bit Pseudo Random Number
GaussHi equ ACCbHI
GaussLo equ ACCbLO ; 16 bit Gaussian distributed number
GaussTmp equ tempLo

;****************************************************************************

ORG 0x0000
;****************************************************************************
Math Routines Test Program
;****************************************************************************

; Load constant values to ACCa & ACCb for testing

main

; call loadAB ; result of adding ACCb+ACCa->ACCb
; call D_add ; Here Accb = 81FE
; call loadAB ; result of subtracting ACCb - ACCa->ACCb
; call D_sub ; Here Accb = 7E00
; call loadAB ; result of multiplying ACCb*ACCa-
; call D_mpyS ; Here (ACCd,ACCc) = 00FF 7E01
; call loadAB ; result of multiplying ACCb*ACCa-
; call D_mpyF ; Here (ACCd,ACCc) = 00FF 7E01
; call loadAB ; result of multiplying ACCb/ACCa-
; call D_divS ; Here (ACCd,ACCc) = 0040 003f
; call loadAB ; result of multiplying ACCb/ACCa-
; call D_divF ; Here (ACCd,ACCc) = 0040 003f

movlw 0xf3
movwf NumHi
movlw 0xf6
movwf NumLo
; Set input test number = 62454
; - F3F6h
; result = 00F9h = 249 (in SqrtLo)
; exact sqrt(62454) = 249.9

call loadAB
movlw 0xff
movwf mulplr ; multiplier (in mulplr) = 0FF
movlw 0xff
movwf multiplicand(W Reg) = 0FF

; The result OFF*OFF = FE01 is in locations
; H_byte & L_byte
movlw 0xff
movwf mulplr ; multiplier (in mulplr) = 0FF
movlw 0xff
movwf multiplicand(W Reg) = 0FF
movlw 0xff
movwf multiplicand(W Reg) = 0FF

Math Routines

001A E2B8 call mpy8x8_S ; The result OFF*OFF = FE01 is in _H_byte & _L_byte ; Test The Random Number Generators ; Capture data into trace buffer by TABLE WRITES to a dummy Program Memory location ;

001B B0FF
001C 010D
001D B05F
001E 010E
;
001F B030
0020 0119
0021 B045
0022 011B
;
0023 C028
;
RandPoint

0024 E311
0025 A418
0026 AE19
0027 C024
;
GaussPoint

0028 E31E
0029 A41A
002A AE1B
002B C028
;
self goto self ; End Of Test Routines
;
loadAB

002C C02C
;
*******************************************************************
Double Precision Arithmetic Routines
;
Routines : Addition, Subtraction, Multiplication ,Division
Square Root
;
NOTE : MODE_FAST must first be set to either TRUE or FALSE
;
MODE_FAST determines the RAM address locations of ACCa thru ACCd
;
If MODE_FAST is set TRUE, data transfers can be done efficiently
using "MOVFP" & "MOVPF" instructions instead of indirectly moving
at first to W Reg and then to the desired RAM locations
;
The speed increase using this way of locating ACCa to
ACCd will result in a saving of about 20 Cycles/filter stage
In this case ( a 2 stage filter), it is faster by 40 Cycles
;
If due to other constraints, ACCa thru ACCd cannot be set at
address 0xl8 to Oxff, then the user is required to set
MODE_FAST to FALSE
;
*******************************************************************
Double Precision Addition
Math Routines

; Addition : ACCb (16 bits) + ACCa (16 bits) -> ACCb (16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
; (c) CALL D_add
; (d) The result is in location ACCbLO & ACCbHI (16 bits)

Performance:
Program Memory: 4 (excluding call & return)
Clock Cycles: 4 (excluding call & return)
W Register: Used
Scratch RAM: 0

;***************************************************************************;
D_add
movfp ACCaLO, wreg
addwf ACCbLO ; addwf lsb
movfp ACCaHI, wreg
addwf ACCbHI ; addwf msb with carry
return
;***************************************************************************;

; Double Precision Subtraction
;
Subtraction : ACCb (16 bits) - ACCa (16 bits) -> ACCb (16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
; (c) CALL D_sub
; (d) The result is in location ACCbLO & ACCbHI (16 bits)

Performance:
Program Memory: 4 (excluding call & return)
Clock Cycles: 4 (excluding call & return)
W Register: Used
Scratch RAM: 0

;***************************************************************************;
D_sub
movfp ACCaLO, wreg
subwf ACCbLO
movfp ACCaHI, wreg
subwfb ACCbHI
return
;***************************************************************************;

; Function to negate a 16 bit integer
; The two 8 bit integers are assumed to be in 2 consecutive locations. Before calling this routine, FSR0 should be loaded with the address of the lower byte.
; Assume that ALUSTA register is set for no autoincrement of FSR0.
;***************************************************************************;

negateAlt
movfp indf0, wreg
bcf _fsl
negw indf0
bsf _fsl
movfp indf0, wreg
clrf indf0
subwfb indf0
return

negate
comf indf0
bcf _fsl
incf indf0
bsf _fsl
Math Routines

Double Precision Multiplication

( Optimized for Code : Looped Code )

Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd, ACCc (32 bits )
(a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
(b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
(c) CALL D_mpyS
(d) The 32 bit result is in location ( ACCdHI, ACCdLO, ACCcHI, ACCcLO )

Performance:
Program Memory: 21 (UNSIGNED)
Clock Cycles: 52 (SIGNED)
scratch RAM: 242 (UNSIGNED : excluding CALL & RETURN)
254 (SIGNED : excluding CALL & RETURN)

Note: The above timing is the worst case timing, when the
register ACCb = FFFF. The speed may be improved if
the register ACCb contains a number (out of the two
numbers) with less number of 1s.

Double Precision Multiply (16x16 -> 32)
( ACCb * ACCa -> ACCb, ACCc ) : 32 bit output with high word
in ACCd ( ACCdHI, ACCdLO ) and low word in ACCc ( ACCcHI, ACCcLO ).

********************************************************************

D_mpyS : results in ACCd(16 msb’s) and ACCc(16 lsb’s)

if SIGNED
CALL S_SIGN
endif

0050 2922
0051 8422
0052 5A20
0053 5B21

if MODE_FAST
movpf ACCbLO,tempLo
movpf ACCbHI,tempHi
else
movfp ACCbLO,wreg
movf tempLo
movfp ACCbHI,wreg
movf tempHi
endif

0054 291F
0055 291E

clr ACCdHI
clr ACCdLO

; shift right and addwf 16 times

mpyLoop

0056 1921
0057 1920
0058 9004
0059 C05E
005A 6018
005B 0F1E
005C 6019
005D 111F

rcf tempHi
rcf tempLo
btfss _carry
goto NoAdd ; LSB is 0, so no need to addwf
movfp ACCcLO,wreg
addwf ACCdLO ; addwf lsb
movfp ACCcHI,wreg
addwf ACCcHI ; addwf msb

005E 191F
005F 191E

rcf ACCdHI
rcf ACCdLO

© 1993 Microchip Technology Incorporated
Math Routines

0060 191D
0061 191C
0062 1722
0063 C056

; if SIGNED
; btfss sign, MSB
; return
; comf ACCcLO
; incf ACCcLO
; btfsc _z
; decf ACCcHI
; comf ACCcHI
; btfsc _z
; decf ACCcLO
; comf ACCcLO
; btfsc _z
; decf ACCcHI
; comf ACCcHI
; return
else
; return
endif

; Assemble this section only if Signed Arithmetic Needed
; if SIGNED
; _S_SIGN
movfp ACCaHI, wreg
xorwf ACCbHI, w
movwf sign
; MSB of sign determines whether signed
btfss ACCbHI, MSB
; if MSB set go & negate ACCb
goto chek_A
comf ACCbLO
incf ACCbLO
btfsc _z
; negate ACCb
decf ACCbHI
comf ACCbHI

; chek_A
btfss ACCaHI, MSB
; if MSB set go & negate ACCa
return
comf ACCaLO
incf ACCaLO
btfsc _z
; negate ACCa
decf ACCaHI
comf ACCaHI
return

;***************************************************************************
; Double Precision Multiplication
; ( Optimized for Speed : straight Line Code )
; Multiplication : ACCb (16 bits) * ACCa (16 bits) -> ACCd, ACCc (32 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
; (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
; (c) CALL D_mpy
; (d) The 32 bit result is in location (ACCdHI, ACCdLO, ACCcHI, ACCcLO)
; Performance :
; Program Memory : 179 (UNSIGNED)
; : 204 (SIGNED)
; Clock Cycles : 176 (UNSIGNED :excluding CALL & RETURN)
; : 183 (SIGNED :excluding CALL & RETURN)
Math Routines

; Note: The above timing is the worst case timing, when the
; register ACCb = FFFF. The speed may be improved if
; the register ACCb contains a number (out of the two
; numbers) with less number of 1s.
; The performance specs are for Unsigned arithmetic (i.e,
; with "SIGNED equ FALSE").
; Upon return from subroutine, the input registers
;
;******************************************************************************
; Multiplication Macro
;******************************************************************************

mulMac MACRO
  variable
  i ~ 0
  if SIGNED
    .while i < 15
  else
    .while i < 16
  endif
  .if i < 8
    btfss ACCbLO, i ; test low byte
  .else
    btfss ACCbHI, i-8 ; test high byte
  .fi
  goto NoAddSv(i) ; LSB is 0, so no need to addwf
  movfp ACCaLO, wreg
  addwf ACCaLO
  movfp ACCaHI, wreg
  addwf ACCaHI
  NoAddSv(i)
  rrcf ACCdHI
  rrcf ACCdLO
  rrcf ACCcHI
  rrcf ACCcLO
  bcf _carry
  i = i+1
  .endw
  if SIGNED
    rrcf ACCdHI
    rrcf ACCdLO
    rrcf ACCcHI
    rrcf ACCcLO
    bcf _carry
  endif
ENDM

;******************************************************************************
; Double Precision Negate Macros
;******************************************************************************
AltNegMac MACRO
  fileRegLo, fileRegHi
  movfp fileRegLo, wreg
  negw fileRegLo
  movfp fileRegHi, wreg
  clrf fileRegHi
  subwfb fileRegHi
ENDM

; negMac MACRO
  fileRegLo, fileRegHi
  comf fileRegLo ; negate FileReg ( ~FileReg -> FileReg )
  incf fileRegLo
  btfsc _z
Math Routines

```assembly
; NegMac32 MACRO x3,x2,x1,x0
    movfp x3, wreg
    negw x3
    movfp x2, wreg
    clrf x2
    subwfb x2
    movfp x1, wreg
    clrf x1
    subwfb x1
    movfp x0, wreg
    clrf x0
    subwfb x0
    ENDM

;*******************************************************************~
Double Precision Multiply (16x16 -> 32)
( ACCb*ACCa -> ACCb, ACCc ) : 32 bit output with high word
in ACCd ( ACCdHI, ACCdLO ) and low word in ACCc ( ACCcHI, ACCcLO ).
D_mpyF

if SIGNED
    movfp ACCaHI, wreg
    xorwf ACCbHI, w
    movwf ACCbHI, sign
    btfs ACCbHI, MSB
    goto chek_A_MSB_MPY

negMac ACCbLO, ACCbHI

chek_A_MSB_MPY
    btfs ACCaHI, MSB
    goto continue_MPY
    negMac ACCaLO, ACCaHI

endif
continue_MPY
0065 291F clrf ACCdHI
0066 291E clrf ACCdLO
0067 8804 bcf _carry

; use the mulMac macro 16 times
mulMac
0000 variable i
0000 i = 0
if SIGNED
    .while i < 15
else
    .while i < 16
endif
.if i < 8
    btfs ACCbLO, i ; test low byte
.else
    btfs ACCbHI, i-8 ; test high byte
.fi
    goto NoAdd#v(i) ; LSB is 0, so no need to addwf
    movfp ACCaLO, wreg
```

© 1993 Microchip Technology Incorporated
Math Routines

```assembly
addwf ACCdLO
movf ACCaHI, wreg
addwfc ACCdHI

NoAdd#v(i)
rrcf ACCdHI
rrcf ACCdLO
rrcf ACCcHI
bcf _carry
i = i + 1
.endw

0068 901A
.if i < 8
  btfs ACCbLO, i  ; test low byte
.else
  btfs ACCbHI, i-8  ; test high byte
.fi

0069 C06E
goto NoAdd0  ; LSB is 0, so no need to addwf
006A 601B
movf ACCaLO, wreg
addwf ACCbLO
addwfc ACCbHI

NoAdd0
rrcf ACCdHI
rrcf ACCdLO
rrcf ACCcHI
bcf _carry
i = i + 1

0073 911A
.if i < 8
  btfs ACCbLO, i  ; test low byte
.else
  btfs ACCbHI, i-8  ; test high byte
.fi

0074 C079
goto NoAdd1  ; LSB is 0, so no need to addwf
0075 601B
movf ACCaLO, wreg
addwf ACCbLO
addwfc ACCbHI

NoAdd1
rrcf ACCdHI
rrcf ACCdLO
rrcf ACCcHI
bcf _carry
i = i + 1

007E 921A
.if i < 8
  btfs ACCbLO, i  ; test low byte
.else
  btfs ACCbHI, i-8  ; test high byte
.fi

007F C084
goto NoAdd2  ; LSB is 0, so no need to addwf
0080 601B
movf ACCaLO, wreg
addwf ACCbLO
addwfc ACCbHI

NoAdd2
rrcf ACCdHI
rrcf ACCdLO
rrcf ACCcHI
rrcf ACCcLO
```

© 1993 Microchip Technology Incorporated
Math Routines

0088 8804  bcf  _carry
0089 931A  i = i + 1
          .if i < 8
008A C08F  btfss ACCbLO, i ; test low byte
          .else
008B 6018  btfss ACCbHI, i-8 ; test high byte
          .fi
          goto NoAdd3 ; LSB is 0, so no need to addwf
008C 0F1E  movfp ACCaLO, wreg
008D 6019  addwf ACCdLO ;addwf lsb
008E 111F  movfp ACCaHI, wreg
                  addwfc ACCdHI ;addwf msb
                  NoAdd3
008F 191F  rrcf ACCdHI
0090 191E  rrcf ACCdLO
0091 191D  rrcf ACCcHI
0092 191C  rrcf ACCcLO
0093 8804  bcf  _carry
                  i = i + 1
          .if i < 8
0094 941A  btfss ACCbLO, i ; test low byte
          .else
0095 C09A  btfss ACCbHI, i-8 ; test high byte
          .fi
          goto NoAdd4 ; LSB is 0, so no need to addwf
0096 6018  movfp ACCaLO, wreg
0097 0F1E  addwf ACCdLO ;addwf lsb
0098 6019  movfp ACCaHI, wreg
0099 111F  addwfc ACCdHI ;addwf msb
                  NoAdd4
009A 191F  rrcf ACCdHI
009B 191E  rrcf ACCdLO
009C 191D  rrcf ACCcHI
009D 191C  rrcf ACCcLO
009E 8804  bcf  _carry
                  i = i + 1
          .if i < 8
009F 951A  btfss ACCbLO, i ; test low byte
          .else
00A0 C0A5  btfss ACCbHI, i-8 ; test high byte
          .fi
          goto NoAdd5 ; LSB is 0, so no need to addwf
00A1 6018  movfp ACCaLO, wreg
00A2 0F1E  addwf ACCdLO ;addwf lsb
00A3 6019  movfp ACCaHI, wreg
00A4 111F  addwfc ACCdHI ;addwf msb
                  NoAdd5
00A5 191F  rrcf ACCdHI
00A6 191E  rrcf ACCdLO
00A7 191D  rrcf ACCcHI
00A8 191C  rrcf ACCcLO
00A9 8804  bcf  _carry
                  i = i + 1
          .if i < 8
00AA 961A  btfss ACCbLO, i ; test low byte
          .else
00AB 961A  btfss ACCbHI, i-8 ; test high byte
          .fi
Math Routines

```
00AB C0B0    goto NoAdd6 ; LSB is 0, so no need to addwf
00AC 6018    movf ACCaLO, wreg
00AD 0F1E    addwf ACCaLO ; addwf lsb
00AE 6019    movf ACCaHI, wreg
00AF 111F    addwfc ACCaHI ; addwf msb

00B0 191F    rrcf ACCaHI
00B1 191E    rrcf ACCaLO
00B2 191D    rrcf ACCaHI
00B3 191C    rrcf ACCaLO
00B4 8804    bcf _carry
00B7 971A    if i < 8
             btfss ACCbLO, 1 ; test low byte
             .else
             btfss ACCbHI, i-8 ; test high byte
             .fi

00B8 C0B8    goto NoAdd7 ; LSB is 0, so no need to addwf
00B9 6018    movf ACCaLO, wreg
00BA 0F1E    addwf ACCaLO ; addwf lsb
00BB 6019    movf ACCaHI, wreg
00BC 111F    addwfc ACCaHI ; addwf msb

00BD 191F    rrcf ACCaHI
00BE 191E    rrcf ACCaLO
00BF 191D    rrcf ACCaHI
00C0 191C    rrcf ACCaLO
00C1 8804    bcf _carry
00C4 901B    if i < 8
             btfss ACCbLO, 1 ; test low byte
             .else
             btfss ACCbHI, i-8 ; test high byte
             .fi

00C7 C0C6    goto NoAdd8 ; LSB is 0, so no need to addwf
00C8 6018    movf ACCaLO, wreg
00C9 0F1E    addwf ACCaLO ; addwf lsb
00CC 6019    movf ACCaHI, wreg
00CD 111F    addwfc ACCaHI ; addwf msb

00CE 191F    rrcf ACCaHI
00CF 191E    rrcf ACCaLO
00DD 191D    rrcf ACCaHI
00DE 191C    rrcf ACCaLO
00DF 8804    bcf _carry
0009        if i < 8
             btfss ACCbLO, 1 ; test low byte
             .else
             btfss ACCbHI, i-8 ; test high byte
             .fi
```

© 1993 Microchip Technology Incorporated
Math Routines

0003 191D  rrcf  ACCcHI
0004 191C  rrcf  ACCcLO
0005 8804  bcf  _carry
0006  0=i+1
.if  i  <  8
   btfs  ACCbLO, i  ; test low byte
.else
0006  921B  btfs  ACCbHI, i-8  ; test high byte
 .fi
0007  C0DC  goto  NoAdd10  ; LSB is 0, so no need to addwf
0008  6018  movfp  ACCaLO,wreg
0009  0F1E  addwf  ACCdLO  ;addwf lsb
000A  6019  movfp  ACCaHI,wreg
000B  111F  addwfc  ACCdHI  ;addwf msb
000C  191F  rrcf  ACCdHI
000D  191E  rrcf  ACCdLO
000E  191D  rrcf  ACCcHI
000F  191C  rrcf  ACCcLO
0010  8804  bcf  _carry
0011  0=i+1
.if  i  <  8
   btfs  ACCbLO, i  ; test low byte
.else
0012  931B  btfs  ACCbHI, i-8  ; test high byte
 .fi
0013  C0E7  goto  NoAdd11  ; LSB is 0, so no need to addwf
0014  6018  movfp  ACCaLO,wreg
0015  0F1E  addwf  ACCdLO  ;addwf lsb
0016  6019  movfp  ACCaHI,wreg
0017  111F  addwfc  ACCdHI  ;addwf msb
0018  191F  rrcf  ACCdHI
0019  191E  rrcf  ACCdLO
001A  191D  rrcf  ACCcHI
001B  191C  rrcf  ACCcLO
001C  8804  bcf  _carry
001D  0=i+1
.if  i  <  8
   btfs  ACCbLO, i  ; test low byte
.else
001E  941B  btfs  ACCbHI, i-8  ; test high byte
 .fi
001F  C0F2  goto  NoAdd12  ; LSB is 0, so no need to addwf
0020  6018  movfp  ACCaLO,wreg
0021  0F1E  addwf  ACCdLO  ;addwf lsb
0022  6019  movfp  ACCaHI,wreg
0023  111F  addwfc  ACCdHI  ;addwf msb
0024  191F  rrcf  ACCdHI
0025  191E  rrcf  ACCdLO
0026  191D  rrcf  ACCcHI
0027  191C  rrcf  ACCcLO
0028  8804  bcf  _carry
0029  0=i+1
.if  i  <  8
   btfs  ACCbLO, i  ; test low byte
.else


Math Routines

```
0007 951B
  btfss  ACCbHI,i-8 ; test high byte
  .fi
0008 C0FD
  goto  NoAddl3 ; LSB is 0, so no need to addwf
0009 601B
  movfp ACCaLO,wreg
  addwf ACCdLO ; addwf lsb
000A 0F1B
  movfp ACCaHI,wreg
  addwf ACCcHI ; addwf msb
000B 111F
  NoAddl3
000D 191F
  rrcf ACCdHI
  rrcf ACCdLO
  rrcf ACCcHI
  rrcf ACCcLO
  bcf _carry
  i = i+1
  .if i < 8
  btfss ACCbLO,i ; test low byte
  .else
0012 961B
  btfss  ACCbHI,i-8 ; test high byte
  .fi
0013 C10B
  goto  NoAddl4 ; LSB is 0, so no need to addwf
0014 601B
  movfp ACCaLO,wreg
  addwf ACCdLO ; addwf lsb
0015 0F1B
  movfp ACCaHI,wreg
  addwf ACCcHI ; addwf msb
0016 111F
  NoAddl4
0018 191F
  rrcf ACCdHI
  rrcf ACCdLO
  rrcf ACCcHI
  rrcf ACCcLO
  bcf _carry
  i = i+1
  .if i < 8
  btfss ACCbLO,i ; test low byte
  .else
0020 971B
  btfss  ACCbHI,i-8 ; test high byte
  .fi
0021 C113
  goto  NoAddl5 ; LSB is 0, so no need to addwf
0022 601B
  movfp ACCaLO,wreg
  addwf ACCdLO ; addwf lsb
0023 0F1B
  movfp ACCaHI,wreg
  addwf ACCcHI ; addwf msb
0024 111F
  NoAddl5
0026 191F
  rrcf ACCdHI
  rrcf ACCdLO
  rrcf ACCcHI
  rrcf ACCcLO
  bcf _carry
  i = i+1
  .if SIGNED
  rrcf ACCdHI
  rrcf ACCdLO
  rrcf ACCcHI
  rrcf ACCcLO
  bcf _carry
  endif
```
Math Routines

if SIGNED
    btfss sign,MSB
    negate (ACCc,ACCd)
return
end

Double Precision Division

(a) Load the Denominator in location ACCaHI & ACCaLO (16 bits)
(b) Load the Numerator in location ACCbHI & ACCbLO (16 bits)
(c) CALL D div
(d) The 16 bit result is in location ACCbHI & ACCbLO
(e) The 16 bit remainder is in locations ACCcHI & ACCcLO

Performance:
Program Memory: 31 (UNSIGNED)
               39 (SIGNED)
Clock Cycles:  300 (UNSIGNED: excluding CALL & RETURN)
               312 (SIGNED: excluding CALL & RETURN)

NOTE:
The performance specs are for unsigned arithmetic (i.e.,
with "SIGNED equ FALSE").

Double Precision Divide (16/16 = 16)

(a) ACCb/ACCa = ACCb with remainder in ACCc : 16 bit output
   with Quotient in ACCb (ACCbHI,ACCbLO) and remainder in ACCc

B/A = (Q) + (R)/A
or  B = A*Q + R

where B : Numerator
          A : Denominator
          Q : Quotient (Integer Result)
          R : Remainder

Note: Check for ZERO Denominator or Numerator is not performed
      A ZERO Denominator will produce incorrect results

SIGNED Arithmetic:
In case of signed arithmetic, if either
numerator or denominator is negative, then both Q & R are
represented as negative numbers
-(B/A) = -(Q) + (-R)/A
or  -B = (-Q)*A + (-R)

D_divS

set no auto-increment for fsr0

if SIGNED
CALL s_Sign
endif

011B 2922
011C 8422
011D 291D
011E 291C
011F 291E
0120 291F

; Looped code

0121 8804
0122 1B1A
0123 1B1B
0124 1B1C
0125 1B1D
0126 6019
0127 041D
0128 9204
0129 C12C
012A 6018
012B 041C
012C 9004
012D C133

Looped code

012E 6018
012F 051C
0130 6019
0131 031D
0132 8004
0133 1B1E
0134 1B1F
0135 1722
0136 C121

if SIGNED then

btfs s_sign, MSB
return
movlw ACCcLO
movwf fsr0
call negate
movlw ACCcLO
movwf fsr0
call negate
return
else
return
endif

*******************************************************************
Double Precision Division
*******************************************************************

; Optimized for Speed : straight line Code

Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
Remainder in ACCc ( 16 bits )

; (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
; (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
; (c) CALL D_div
; (d) The 16 bit result is in location ACCbHI & ACCbLO
; (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO

;*******************************************************************
Math Routines

\[
\begin{align*}
B/A &= (Q) + (R)/A \\
\text{or} \quad B &= A*Q + R
\end{align*}
\]

where \( B \) : Numerator
\( A \) : Denominator
\( Q \) : Quotient (Integer Result)
\( R \) : Remainder

Note: Check for ZERO Denominator or Numerator is not performed
A ZERO Denominator will produce incorrect results

SIGNED Arithmetic:
In case of signed arithmetic, if either numerator or denominator is negative, then both \( Q \) & \( R \) are represented as negative numbers

\[
-(B/A) = -(Q) + (-R)/A
\]
\[
-B = (-Q)*A + (-R)
\]

Performance:

<table>
<thead>
<tr>
<th>Program Memory</th>
<th>325 (UNSIGNED)</th>
<th>354 (SIGNED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Cycles</td>
<td>250 (UNSIGNED excluding CALL &amp; RETURN)</td>
<td>260 (SIGNED excluding CALL &amp; RETURN)</td>
</tr>
</tbody>
</table>

*******************************************************************
divMac MACRO
variable i

i = 0
.while i < 16

bcf _carry
rlcf ACCbLO
rlcf ACCbHI
rlcf ACCcLO
rlcf ACCcHI
movfp ACCaHI,wreg
subwf ACCcHI,w ;check if a>c
btfss _z
goto notz#v(i)
movfp ACCaLO,wreg
subwf ACCcLO,w ;if msb equal then check lsb
notz#v(i) btfss _carry ;carry set if c>a
goto nosub#v(i) ;if c < a
subca#v(i) movfp ACCaLO,wreg ;c=a into c
subwf ACCcLO
movfp ACCaHI,wreg
subwfb ACCcHI
bsf _carry ;shift a 1 into d (result)
nosub#v(i) rlcf ACCcLO
rlcf ACCcHI
i=i+1
.endw

ENDM
*******************************************************************

Double Precision Divide ( 16/16 \( \rightarrow \) 16 )

( ACCb/ACCa \( \rightarrow \) ACCb with remainder in ACCc ) : 16 bit output
with Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc

(ACCcHI,ACCcLO).

NOTE: Before calling this routine, the user should make sure that the Numerator(ACCb) is greater than Denominator(ACCa). If the case is not true, the user should scale either Numerator or Denominator or both such that Numerator is greater than
; the Denominator.
;
;*******************************************************************************
;
D_divF
;
if SIGNED
movfp ACCaHI,wreg
xorwf ACCbHI,w
movwf sign
btfss ACCbHI,MSB ; if MSB set go & negate ACCb
goto chek_A_MSB_DIV
;
; negMac ACCbLO,ACCbHI
;
chek_A_MSB_DIV
btfss ACCaHI,MSB ; if MSB set go & negate ACCa
goto continue_DIV
negMac ACCaLO,ACCaHI
endif
;
continue_DIV
clrf ACCcHI
clrf ACCcLO
clrf ACCcHI
clrf ACCcHI
straight line code using the macro divMac
;
divMac
0000
variable i
0000
i = 0
.w while i < 16
;
  bcf _carry
  rlcf ACCbLO
  rlcf ACCbHI
  rlcf ACCcLO
  rlcf ACCcHI
  movfp ACCaHI,wreg
  subwf ACCcHI,w
  btfss _z
  goto notz\#v(i)
notz\#v(i)
  btfss _carry
  ;carry set if c>a
  goto nosub\#v(i)
subca\#v(i)
  movfp ACCaLO,wreg
  ;c-a into c
  subwf ACCcLO
  movfp ACCaHI,wreg
  subwfb ACCcHI
  bcf _carry
  nosub\#v(i)
  rlcf ACCdLO
  rlcf ACCdHI
  i=i+1
.endw
;
013B 291D
0139 291C
013A 291E
013B 291F
013C 8804
013D 1B1A
013E 1B1B
013F 1B1C
0140 1B1D
0141 6019
0142 041D

© 1993 Microchip Technology Incorporated  DS00544B-page 23 4-93
Math Routines

0143 9204  btfsz  _z
0144 9217  goto  notz0
0145 6018  movfp  ACCaLO,wreg
0146 041C  subwf  ACCcLO,w  ;if msb equal then check lsb
0147 9004  notz0  btfsz  _carry
0148 921E  goto  nosub0
0149 6018  movfp  ACCaLO,wreg
014A 051C  subwf  ACCcLO
014B 6019  movfp  ACCAH,wreg
014C 031D  subwfb  ACCcHI
014D 8004  bcf  _carry
014E 1B1E  nosub0  rlf  ACCdLO
014F 1B1F  rlf  ACCdHI
0001  i = i+1

0150 8804  ;
0151 1B1A  bcf  _carry
0152 1B1B  rlf  ACCbLO
0153 1B1C  rlf  ACCbHI
0154 1B1D  rlf  ACCcLO
0155 6019  movfp  ACCAH,wreg
0156 041D  subwf  ACCcHI,w  ;check if a>c
0157 9204  btfsz  _z
0158 C15B  goto  notzl
0159 6018  movfp  ACCaLO,wreg
015A 041C  subwf  ACCcLO,w  ;if msb equal then check lsb
015B 9004  notzl  btfsz  _carry
015C 1C62  goto  nosub1
015D 6018  subca1  movfp  ACCaLO,wreg
015E 051C  subwf  ACCcLO
015F 6019  movfp  ACCAH,wreg
0160 031D  subwfb  ACCcHI
0161 8004  bcf  _carry
0162 1B1E  nosub1  rlf  ACCdLO
0163 1B1F  rlf  ACCdHI
0002  i = i+1

0164 8804  ;
0165 1B1A  bcf  _carry
0166 1B1B  rlf  ACCbLO
0167 1B1C  rlf  ACCbHI
0168 1B1D  rlf  ACCcLO
0169 6019  movfp  ACCAH,wreg
016A 041D  subwf  ACCcHI,w  ;check if a>c
016B 9204  btfsz  _z
016C C16F  goto  notz2
016D 6018  movfp  ACCaLO,wreg
016E 041C  subwf  ACCcLO,w  ;if msb equal then check lsb
016F 9004  notz2  btfsz  _carry
0170 1C76  goto  nosub2
0171 6018  subca2  movfp  ACCaLO,wreg
0172 051C  subwf  ACCcLO
0173 6019  movfp  ACCAH,wreg
0174 031D  subwfb  ACCcHI
0175 8004  bcf  _carry
0176 1B1E  nosub2  rlf  ACCdLO
0177 1B1F  rlf  ACCdHI
0003  i = i+1

0178 8804  ;
0179 1B1A  bcf  _carry
017A 1B1B  rlf  ACCbLO
017B 1B1C  rlf  ACCbHI
017C 1B1D  rlf  ACCcLO
017D 6019  movfp  ACCAH,wreg
017E 041D  subwf  ACCcHI,w  ;check if a>c
Math Routines

017F 9204 btfss _z
0180 C183 goto notz3
0181 6018 movfp ACCaLO,wreg
0182 041C subwf ACCcLO,w ;if msb equal then check lsb
0183 9004 notz3 btfss _carry ;carry set if c>a
0184 C18A goto nosub3 ; if c < a
0185 6018 subca3 movfp ACCaLO,wreg
0186 051C subwf ACCcLO
0187 6019 movfp ACCaHI,wreg
0188 031D subwb ACCcHI
0189 8004 bcf carry ;shift a 1 into d (result)
018A 181E nosub3 rclf ACCdLO
018B 181F rlc ACCdHI
0004 1 = i+1

; 018C 8804 bcf _carry
018D 1B1A rlc ACCbLO
018E 1B1B rlc ACCbHI
018F 1B1C rlc ACCcLO
0190 1B1D rlc ACCcHI
0191 6019 movfp ACCaHI,wreg
0192 041D subwf ACCcHI,w ;check if a>c
0193 9204 btfss _z
0194 C197 goto notz4
0195 6018 movfp ACCaLO,wreg
0196 041C subwf ACCcLO,w ;if msb equal then check lsb
0197 9004 notz4 btfss _carry ;carry set if c>a
0198 C19E goto nosub4 ; if c < a
0199 6018 subca4 movfp ACCaLO,wreg
019A 051C subwf ACCcLO
019B 6019 movfp ACCaHI,wreg
019C 031D subwb ACCcHI
019D 8004 bcf carry ;shift a 1 into d (result)
019E 181E nosub4 rclf ACCdLO
019F 181F rlc ACCdHI
0005 1 = i+1

; 01A0 8804 bcf _carry
01A1 1B1A rlc ACCbLO
01A2 1B1B rlc ACCbHI
01A3 1B1C rlc ACCcLO
01A4 1B1D rlc ACCcHI
01A5 6019 movfp ACCaHI,wreg
01A6 041D subwf ACCcHI,w ;check if a>c
01A7 9204 btfss _z
01A8 C1AB goto notz5
01A9 6018 movfp ACCaLO,wreg
01AA 041C subwf ACCcLO,w ;if msb equal then check lsb
01AB 9004 notz5 btfss _carry ;carry set if c>a
01AC C1B2 goto nosub5 ; if c < a
01AD 6018 subca5 movfp ACCaLO,wreg
01AE 051C subwf ACCcLO
01AF 6019 movfp ACCaHI,wreg
01B0 031D subwb ACCcHI
01B1 8004 bcf carry ;shift a 1 into d (result)
01B2 1B1E nosub5 rclf ACCdLO
01B3 1B1F rlc ACCdHI
0006 1 = i+1

; 01B4 8804 bcf _carry
01B5 1B1A rlc ACCbLO
01B6 1B1B rlc ACCbHI
01B7 1B1C rlc ACCcLO
01B8 1B1D rlc ACCcHI
01B9 6019 movfp ACCaHI,wreg
01BA 041D subwf ACCcHI,w ;check if a>c
Math Routines

01BB 9204  btfs z  
01BC 01BF  goto notz6  
01BD 6018  movfp ACCaLO,wreg  
01BE 01C  subwf ACCcLO,w  
01BF 9004  notz6  btfs _carry  
01C0 01C6  goto nosub6  
01C1 6018  movfp ACCaLO,wreg  
01C2 051C  subwf ACCcLO  
01C3 6019  movfp ACCaHI,wreg  
01C4 031D  subwf ACCcHI  
01C5 8004  bcf _carry  
01C6 1B1E  nosub6  rlcf ACCdLO  
01C7 1B1F  rlcf ACCdHI  
0007  i = i+1  
01C8 8804  
01C9 1B1A  bcf _carry  
01CA 1B1B  rlcf ACCbLO  
01CB 1B1C  rlcf ACCbHI  
01CC 1B1D  rlcf ACCcLO  
01CD 6019  movfp ACCaHI,wreg  
01CE 041D  subwf ACCcHI,w  
01CF 9204  btfs z  
01D0 01D3  goto notz7  
01D1 6018  movfp ACCaLO,wreg  
01D2 041C  subwf ACCcLO,w  
01D3 9004  notz7  btfs _carry  
01D4 1D1A  goto nosub7  
01D5 6018  movfp ACCaLO,wreg  
01D6 051C  subwf ACCcLO  
01D7 6019  movfp ACCaHI,wreg  
01D8 031D  subwf ACCcHI  
01D9 8004  bcf _carry  
01DA 1B1E  nosub7  rlcf ACCdLO  
01DB 1B1F  rlcf ACCdHI  
0008  i = i+1  
01DC 8804  
01DD 1B1A  bcf _carry  
01DE 1B1B  rlcf ACCbLO  
01DF 1B1C  rlcf ACCbHI  
01E0 1B1D  rlcf ACCcLO  
01E1 6019  movfp ACCaHI,wreg  
01E2 041D  subwf ACCcHI,w  
01E3 9204  btfs z  
01E4 1E17  goto notz8  
01E5 6018  movfp ACCaLO,wreg  
01E6 041C  subwf ACCcLO,w  
01E7 9004  notz8  btfs _carry  
01E8 1E1E  goto nosub8  
01E9 6018  movfp ACCaLO,wreg  
01EA 051C  subwf ACCcLO  
01EB 6019  movfp ACCaHI,wreg  
01EC 031D  subwf ACCcHI  
01ED 8004  bcf _carry  
01EE 1B1E  nosub8  rlcf ACCdLO  
01EF 1B1F  rlcf ACCdHI  
0009  i = i+1  
01F0 8804  
01F1 1B1A  bcf _carry  
01F2 1B1B  rlcf ACCbLO  
01F3 1B1C  rlcf ACCbHI  
01F4 1B1D  rlcf ACCcLO  
01F5 6019  movfp ACCaHI,wreg  
01F6 041D  subwf ACCcHI,w  

© 1993 Microchip Technology Incorporated
Math Routines

01F7 9204  btfss  _z
01F8  C1FB  goto  notz9
01F9  6018  movfp  ACCaLO,wreg
01FA  041C  subwf  ACCcLO,w ;if msb equal then check lsb
01FB  9004  notz9  btfss  _carry  ;carry set if c>a
01FC  C202  goto  nosub9  ; if c < a
01FD  6018  subca9  movfp  ACCaLO,wreg  ;c-a into c
01FE  051C  movfp  ACCaHI,wreg
0200  031D  subwfb  ACCcHI
0201  8004  bcf  _carry  ;shift a 1 into d (result)
0202  1B1E  nosub9  rlf ACCdLO
0203  1B1F  rlf ACCdHI
000A  i = i+1

0204  8804  bcf  _carry
0205  1B1A  rlf ACCbLO
0206  1B1B  rlf ACCbHI
0207  1B1C  rlf ACCcLO
0208  1B1D  rlf ACCcHI
0209  6019  movfp  ACCaHI,wreg
020A  041D  subwf  ACCcHI,w ;check if a>c
020B  9204  btfss  _z
020C  C20F  goto  notz10
020D  6018  movfp  ACCaLO,wreg
020E  041C  subca10  movfp  ACCaLO,wreg  ;c-a into c
020F  9004  notz10  btfss  _carry  ;carry set if c>a
0210  C216  goto  nosub10  ; if c < a
0211  6019  subca10  movfp  ACCaLO,wreg
0212  051C  subwf  ACCcHI
0213  6019  movfp  ACCaHI,wreg
0214  031D  subwfb  ACCcHI
0215  8004  bcf  _carry  ;shift a 1 into d (result)
0216  1B1E  nosub10  rlf ACCdLO
0217  1B1F  rlf ACCdHI
000B  i = i+1

0218  8804  bcf  _carry
0219  1B1A  rlf ACCbLO
021A  1B1B  rlf ACCbHI
021B  1B1C  rlf ACCcLO
021C  1B1D  rlf ACCcHI
021D  6019  movfp  ACCaHI,wreg
021E  041D  subwf  ACCcHI,w ;check if a>c
021F  9204  btfss  _z
0220  C223  goto  notz11
0221  6018  movfp  ACCaLO,wreg
0222  041C  subca11  movfp  ACCaLO,wreg  ;c-a into c
0223  9004  notz11  btfss  _carry  ;carry set if c>a
0224  C22A  goto  nosub11  ; if c < a
0225  6018  subca11  movfp  ACCaLO,wreg  ;c-a into c
0226  051C  subwf  ACCcLO
0227  6019  movfp  ACCaHI,wreg
0228  031D  subwfb  ACCcHI
0229  8004  bcf  _carry  ;shift a 1 into d (result)
022A  1B1E  nosub11  rlf ACCdLO
022B  1B1F  rlf ACCdHI
000C  i = i+1

022C  8804  bcf  _carry
022D  1B1A  rlf ACCbLO
022E  1B1B  rlf ACCbHI
022F  1B1C  rlf ACCcLO
0230  1B1D  rlf ACCcHI
0231  6019  movfp  ACCaHI,wreg
0232  041D  subwf  ACCcHI,w ;check if a>c
Math Routines

0233 9204 btfsz _z
0234 C237 goto notz12
0235 6018 movfp ACCaLO,wreg
0236 041C subwf ACCcLO,w ;if msb equal then check lab
0237 9204 notz12 btfsz _carry ;carry set if c>a
0238 C23E goto nosub12 ;if c < a
0239 6018 movfp ACCaLO,wreg ;c-a into c
023A 051C subwf ACCcLO
023B 6019 movfp ACCcHI,wreg
023C 031D subwfb ACCcHI
023D 8004 bsf _carry ;shift a 1 into d (result)
023E 181E nosub12 rlcf ACCdLO
023F 181F rlcf ACCcHI
000D i = i+1

; 0240 8804 bcf _carry
0241 181A rlcf ACCbLO
0242 181B rlcf ACCbHI
0243 181C rlcf ACCcLO
0244 181D rlcf ACCcHI
0245 6019 movfp ACCaHI,wreg ;check if a>c
0246 041D subwf ACCcHI,w ;c-a into c
0247 9204 btfsz _z
0248 C24B goto notz13 ;if msb equal then check lab
0249 6018 movfp ACCaLO,wreg
024A 041C subwf ACCcLO,w ;if c < a
024B 9204 notz13 btfsz _carry ;carry set if c>a
024C C252 goto nosub13 ;if c < a
024D 6018 movfp ACCaLO,wreg ;c-a into c
024E 051C subwf ACCcLO
024F 6019 movfp ACCcHI,wreg
0250 031D subwfb ACCcHI
0251 8004 bsf _carry ;shift a 1 into d (result)
0252 181E nosub13 rlcf ACCcHI
0253 181F rlcf ACCcHI
000E i = i+1

; 0254 8804 bcf _carry
0255 181A rlcf ACCbLO
0256 181B rlcf ACCbHI
0257 181C rlcf ACCcLO
0258 181D rlcf ACCcHI
0259 6019 movfp ACCaHI,wreg ;check if a>c
025A 041D subwf ACCcHI,w ;c-a into c
025B 9204 btfsz _z
025C C25F goto notz14 ;if msb equal then check lab
025D 6018 movfp ACCaLO,wreg
025E 041C subwf ACCcLO,w ;if c < a
025F 9204 notz14 btfsz _carry ;carry set if c>a
0260 C266 goto nosub14 ;if c < a
0261 6018 movfp ACCaLO,wreg ;c-a into c
0262 051C subwf ACCcLO
0263 6019 movfp ACCcHI,wreg
0264 031D subwfb ACCcHI
0265 8004 bsf _carry ;shift a 1 into d (result)
0266 181E nosub14 rlcf ACCcHI
0267 181F rlcf ACCcHI
000F i = i+1

; 0268 8804 bcf _carry
0269 181A rlcf ACCbLO
026A 181B rlcf ACCbHI
026B 181C rlcf ACCcLO
026C 181D rlcf ACCcHI
026D 6019 movfp ACCcHI,wreg ;check if a>c
026E 041D subwf ACCcHI,w ;check if a>c

DS00544B-page 28 © 1993 Microchip Technology Incorporated
Math Routines

026F 9204 btfof _z
0270 C273 goto notz15
0271 6018 movfp ACCaLO,wreg
0272 041C subw ACCcLO,wreg ;if msb equal then check lsb
0273 9004 notz15 btfof _carry ;carry set if c>a
0274 C27A goto nosub15 ; if c < a
0275 6018 subca15 movfp ACCaLO,wreg ;c-a into c
0276 051C subw ACCcLO
0277 6019 movfp ACCaHI,wreg
0278 031D subwfb ACCcHI
0279 8004 bsf 1_carry ;shift a 1 into d (result)
027A 1B1E nosub15 rlof ACCcLO
027B 1B1F rlof ACCcHI
0010 i = i+1

;Newton-Raphson Method
;
;This routine computes the square root of a 16 bit number(with
;low byte in NumLo & high byte in NumHi ). After loading NumLo &
;NumHi with the desired number whose square root is to be computed,
branch to location Sqrt ( by "GOTO Sqrt" ). "CALL Sqrt" cannot
;be issued because the Sqrt function makes calls to Math routines
;and the stack is completely used up.
The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
The total number of iterations is set to ten. If more iterations
are desired, change "LupCnt equ .10" to the desired value. Also,
the initial guess value of the square root is given set as
input/2 ( in subroutine "init" ). The user may modify this scheme
if a better initial approximation value is known. A good initial
guess will help the algorithm converge at a faster rate and thus
less number of iterations required.
Two utility math routines are used by this program : D_divS
and D_add. These two routines are listed as separate routines
under double precision Division and double precision addition
respectively.

Note : If square root of an 8 bit number is desired, it is probably
better to have a table look scheme rather than using numerical
methods.
This method is computationally quite intensive and
slow, but very accurate and the convergence rate is high.

Performance :
Program Memory : 22 (excluding D_divS subroutine)
Clock Cycles : 3000 (approximately, with 10 iterations)
The #of cycles depends on Number of Iterations Selected.
In a lot of cases 5 or less iterations may be sufficient

*******************************************************************
Square Root By Newton Raphson Method
*******************************************************************
Math Routines

Sqrt

```
call SqrtInit          ; compute initial sqrt = Num/2

nextIter

if MODE_FAST
    movfp NumLo,ACCbLO
    movfp NumHi,ACCbHI
else
    movfp NumLo,wreg
    movwf ACCbLO
    movfp NumHi,wreg
    movwf ACCbHI
endif

;call D_divS           ; double precision division
;double precision addition
movfp ACCdLO,wreg
addwf ACCaLO
;addwf lsb
movfp ACCdHI,wreg
addwf ACCaHI
;addwf msb
;now divide by 2
movf p
movwf p
rnovf
bcf _carry
rrcf
rrcf
decf sz
goto return
```

SqrtInit

```
028B 0002                ; End Sqrt

movlw _LUPCNT           ; set number of iterations
movwf iterCnt
if MODE_FAST
    movfp NumHi,ACCaHI
    movfp NumLo,ACCaLO
else
    movfp NumHi,wreg
    movwf ACCaHI
    movfp NumLo,wreg
    movwf ACCaLO
endif

bcf _carry
rrcf ACCaHI
rrcf ACCaLO              ; set initial guess root = NUM/2
return
```

8x8 Software Multiplier

```
{ Fast Version : Straight Line Code }

The 16 bit result is stored in 2 bytes
Before calling the subroutine " mpy ", the multiplier should
be loaded in location " mulplr ", and the multiplicand in
" mulcnd ". The 16 bit result is stored in locations
H_byte & L_byte.

Performance:

Program Memory : 36 words
# of cycles (excluding call & return) : 36
Scratch RAM : 0 locations
W Register : Used

This routine is optimized for speed efficiency ( straight line code )
For code efficiency, refer to "mult8x8S.asm" ( looped code )

Define a macro for adding & right shifting

multiply MACRO
variable i
```
Math Routines

```
i = 0
    .while i < 8
        btfs c mulplr,i
        addwf H_byte
        rrcf H_byte
        rrcf L_byte
        i = i+1
    .endw

ENDM

; End of macro

mpy8x8_F

0293 291B
0294 291A
0295 6018
0296 8804
  clrf H_byte
  clrf L_byte
  movfp mulcanl, wreg
  bcf _carry

; move the multiplicand to W reg.
; Clear the carry bit in the status Reg.

; multiply

  variable i

  i = 0
    .while i < 8
        btfs c mulplr,i
        addwf H_byte
        rrcf H_byte
        rrcf L_byte
        i = i+1
    .endw
```

```
0297 9819
0298 0F1B
0299 191B
029A 191A
0001
    i = i+1
029B 9919
029C 0F1B
029D 191B
029E 191A
0002
    i = i+1
029F 9A19
02A0 0F1B
02A1 191B
02A2 191A
0003
    i = i+1
02A3 9B19
02A4 0F1B
02A5 191B
02A6 191A
0004
    i = i+1
02A7 9C19
02A8 0F1B
02A9 191B
02AA 191A
0005
    i = i+1
02AB 9D19
02AC 0F1B
02AD 191B
02AE 191A
0006
    i = i+1
02AF 9E19
02B0 0F1B
02B1 191B
02B2 191A
0007
    i = i+1
```
Math Routines

02B3 9F19   btfscl mulplr,i
02B4 0F1B   addwf H_byte
02B5 191B   rrcf H_byte
                i = i+1
02B6 191A
0008
02B7 02B7 0002   return
                
;*******************************************************************
; 8x8 Software Multiplier
; ( Code Efficient : Looped Code )
; The 16 bit result is stored in 2 bytes
; Before calling the subroutine " mpy ", the multiplier should
; be loaded in location " mulplr ", and the multiplicand in
; " mulcnd ". The 16 bit result is stored in locations
; H_byte & L_byte.
; Performance :
; Program Memory : 13 words (excluding call & return)
; # of cycles : 69 (excluding call & return)
; Scratch RAM : 1 byte
; W Register : Used
; This routine is optimized for code efficiency ( looped code )
; For time efficiency code refer to "mult8x8F.asm" ( straight line code )
;*******************************************************************

mpy8x8_S

02B8 291B   clrf H_byte
02B9 291A   clrf L_byte
02BA 2922   clrf count
02BB 8322   bsfb count, 3 ; set count = 8
02BC 6018   movfp mulcnd, wreg
02BD 8804   bcf _carry ; Clear the carry bit in the status Reg.

loop

02BE 9B19   btfscl mulplr, 0
02BF 0F1B   addwf H_byte
02C0 191B   rrcf H_byte
02C1 191A   rrcf L_byte
02C2 2119   rrcf mulplr
02C3 1722   decfsz count
02C4 C2BE   goto loop
02C5 0002   return

;*******************************************************************
; Numerical Differentiation
; The so called "Three-Point Formula" is implemented to
differentiate a sequence of points (uniformly sampled).
; The eqn implemented is :
; \[ f'(X_n) = \frac{f(X_n - 2h) - 4f(X_n - h) + 3f(X_n)}{h} \]
; where \( X_n \) is the present sample and \( h \) is the step size.
; The above formula may be rewritten as :
; \[ f'(X_n) = \frac{0.5f(X_n - 2h) - 2f(X_n - h) + 0.5f(X_n)}{DiffK} \]
; where \( DiffK = h = \text{Step Size} \)
; This differenciation routine can be used very effectively
; in the computation of the differential component part in
; a PID Loop calculation in Motor Control Applications
; Double precision arithmetic is used throught
Math Routines

The present sample value is assumed to be in locations (XnHi, XnLo). The past two values are assumed to be in locations (Xn_1_Hi, Xn_1_Lo) & (Xn_2_Hi, Xn_2_Lo).

The output value is located in DiffHi & DiffLo. No overflow checking mechanism is implemented. If the values are limited to 12 bits, then the user need not worry about overflows.

It is user's responsibility to update the past values with the present values before calling this routine.

After computation, the present value Xn is not moved to Xn-1 because the user may want these values to be intact for other computations (say numerical integration). Also it is user's responsibility to set past 2 values (Xn_1 & Xn_2) values to be zero on initialization.

*******************************************************************

Diff

movf p Xn_2_Lo, wreg
addwf XnLo, w
movf ACCbLO

movf p Xn_2_Hi, wreg
addwf XnHi, w
movf ACCbHI ; Y = f(Xn-2) + f(Xn)

movf p XnLo, wreg
addwf ACCbLO

movf p XnHi, wreg
addwf ACCbHI

movf p XnLo, wreg
addwf ACCbLO

movf p XnHi, wreg
addwf ACCbHI ; Y = f(Xn-2) + 3*f(Xn)

bcf _carry
rrcf ACCbHI
rrcf ACCbLO ; Y = 0.5*[ f(Xn-2) + 3*f(Xn) ]

movf p Xn_1_Lo, wreg
subwf ACCbLO

movf p Xn_1_Hi, wreg
subwfb ACCbHI

movf p Xn_1_Lo, wreg
subwf ACCbLO

movf p Xn_1_Hi, wreg
subwfb ACCbHI ; Y = 0.5*[f(Xn-2) + 3*f(Xn)] - 2*f(Xn-1)

movf p DiffKLo, wreg
movf ACCaLO

movf p DiffKH, wreg
movf ACCaHI

movf p DiffKLo, wreg
movf ACCaLO

movf p DiffKH, wreg
movf ACCaHI

call div

movf ACCbLO, wreg
movf ACCbHI, wreg
movf ACCbLO

movf DiffKLo, wreg
movf ACCaLO

movf DiffKH, wreg
movf ACCaHI

movf DiffKLo, wreg
movf ACCaLO

movf DiffKH, wreg
movf ACCaHI ; result = Y/h

return

*******************************************************************

Numerical Integration

Simpson's Three-Eighths Rule is implemented

Y(n) = \left[ f(X0) + 3*f(X1) + 3*f(X2) + f(X3) \right] \times \frac{3}{8} h
where 'h' is the step size and the integral is over the range x₀ to x₃

The above equation can be rewritten as

\[
Y(n) = [ f(x₀) + 3*f(x₁) + 3*f(x₂) + f(x₃)]*IntgK
\]

where IntgK = 3*h/8 (in locations (IntgKHi, IntgKLo))

This Integration routine can be used very effectively in the computation of the integral component part in a PID Loop calculation in Motor Control Applications.

Double precision arithmetic is used throughout. The three input values over which the integral is to be computed are assumed to be in locations (X₀Lo, X₀Hi), (X₁Lo, X₁Hi), (X₂Lo, X₂Hi) and (X₃Lo, X₃Hi).

The output value is located in IntgKHi & IntgKLo. No overflow checking mechanism is implemented. If the values are limited to 12 bits, then the user need not worry about overflows.

It is user's responsibility to update the past values with the present values before calling this routine. After computation, the present value Xₙ is not moved to Xₙ₋₁ because the user may want these values to be intact for other computations (say numerical integration). Also it is user's responsibility to set past 2 values (Xₙ₋₁ & Xₙ₋₂) values to be zero on initialization.

;*******************************************************************
; Integrate
;*******************************************************************

02E9 6031     movfp X0Lo,wreg
02EA 0E37     addwf X3Lo,w    ; Intg = f(x₀) + f(x₃)
02EB 011A     movwf ACCbLO
02EC 6032     movfp X0Hi,wreg
02ED 1038     addwfc X3Hi, w
02EE 011B     movwf ACCbHI

02EF 6033     movfp X1Lo,wreg
02F0 0F1A     addwf ACCbLO
02F1 6034     movfp X1Hi,wreg
02F2 111B     addwfc ACCbHI    ; Intg = f(x₀) + f(x₃) + X₁
02F3 6033     movfp X1Lo,wreg
02F4 0F1A     addwf ACCbLO
02F5 6034     movfp X1Hi,wreg
02F6 111B     addwfc ACCbHI    ; Intg = f(x₀) + f(x₃) + 2*X₁
02F7 6033     movfp X1Lo,wreg
02F8 0F1A     addwf ACCbLO
02F9 6034     movfp X1Hi,wreg
02FA 111B     addwfc ACCbHI    ; Intg = f(x₀) + f(x₃) + 3*X₁

02FB 6035     movfp X2Lo,wreg
02FC 0F1A     addwf ACCbLO
02FD 6036     movfp X2Hi,wreg
02FE 111B     addwfc ACCbHI    ; Intg = f(x₀) + f(x₃) + 3*X₁ + X₂
02FF 6035     movfp X2Lo,wreg
0300 0F1A     addwf ACCbLO
0301 6036     movfp X2Hi,wreg
0302 111B     addwfc ACCbHI    ; Intg = f(x₀) + f(x₃) + 3*X₁ + 2*X₂
0303 6035     movfp X2Lo,wreg
0304 0F1A     addwf ACCbLO
0305 6036     movfp X2Hi,wreg
0306 111B     addwfc ACCbHI    ; Intg = f(x₀) + f(x₃) + 3*X₁ + 3*X₂

0307 6039     movfp IntgKLo,wreg
0308 0118     movwf ACCaLO
0309 603A     movfp IntgKHI,wreg
Math Routines

030A 0119
movf ACCaHI ; ACCa = IntgK (prepare for multiplication)
;
030B 0050
call D_mpyS ; make sure to set for either SIGNED or UNSIGNED
030C 601E
movfp ACCdLO,wreg
030D 013B
movwf IntgLo ; 32 bit result in ACCd & ACCc
030E 601F
movfp ACCdHI,wreg
030F 013C
movwf IntgHI ; upper 16 bits = result
;
0310 0002
return
;
******************************************************************************

Random Number Generator

This routine generates a 16 Bit Pseudo Sequence Random Generator
It is based on Linear shift register feedback. The sequence
is generated by (Q15 xorwf Q14 xorwf Q12 xorwf Q3 )

The 16 bit random number is in location RandHi(high byte)
& RandLo (low byte)

Before calling this routine, make sure the initial values
of RandHi & RandLo are NOT ZERO
A good choice of initial random number is Ox3045
******************************************************************************

Random16

0311 1A19
rlcf RandHi,w
0312 0C19
xorwf RandHi,w
0313 1B00
rlcf wreg ; carry bit = xorwf(Q15,14)
;
0314 1D19
swapf RandHi
0315 1C18
swapf RandLo,w
0316 2300
rlncf wreg
0317 0C19
xorwf RandHi,w ; LSB = xorwf(Q12,Q3)
0318 1D19
swapf RandHi
0319 B501
andlw 0x01
031A 1B18
rlcf RandLo
031B 0D18
xorwf RandLo
031C 1B19
rlcf RandHi
031D 0002
return
;
******************************************************************************

Gaussian Noise Generator

This routine generates a 16 Bit Gaussian distributed random
points. This routine calls the routine “Random16”, which
generates a pseudo random noise sequence. Gaussian noise
is computed using the CENTRAL LIMIT THEOREM.
The Central Limit Theorem states that the average weighted
sum of uncorrelated samples tends to have a Gaussian distribution
For practical purposes, the sum could be over a sample size
of 32 Random numbers. Better results could result if a larger
sample size is desired. For faster results, a sum over 16 samples
would also be adequate (say, for applications like Speech synthesis,
channel simulations, etc).

The 16 bit Gaussian distributed point is in locations
GaussHi & GaussLo

Before calling this routine, the initial seed of Random
number should be NON ZERO (refer to notes on “Random16” routine)
******************************************************************************

Gauss

031E 2922
clf count
Math Routines

031F 8522   bsf    count, 5   ; set Sample size = 32
0320 291A   clrf    GaussLo
0321 291B   clrf    GaussHi
0322 2920   clrf    GaussTmp

; NextGauss
0323 E311   call    Random16   ; get a random value
0324 6018   movfp   RandLo, wreg
0325 OF1A   addwf   GaussLo
0326 6019   movfp   RandHi, wreg
0327 111B   addwf   GaussHi
0328 2900   clrf    wreg
0329 1120   addwf   GaussTmp
032A 1722   decfsz  count
032B C323   goto    NextGauss   ; sum 16 random numbers

; 032C B005   movlw   5
032D 1920   rrcf   GaussTmp
032E 191B   rrcf   GaussHi
032F 191A   rrcf   GaussLo   ; weighted average
0330 1700   decfsz  wreg   ; divide by 32
0331 C32D   goto    GaussDiv16

; 0332 0002   return

END   ; End Of arith.asm

Errors : 0
Warnings : 0
#define PAGE EJECT

;TITLE "BCD Arithmetic Routines : Ver 1.0"

;*******************************************************************
; BCD Arithmetic Routines
;*******************************************************************

LIST P=17C42, C=80, L=0, R=DEC

#include "17c42.h"

CBLOCK 0x20

0020 0002 Lbyte, Hbyte
0022 0003 R2, R1, R0 :must maintain R2, R1, R0 sequence
0025 0001 count
0026 0002 Num1, Num2

ENDC

0026 BCD equ Num1
0026 Htemp equ Num1
0027 Ltemp equ Num2

; ORG 0x0000

;******************************************************

BCD Arithmetic Test Program
;******************************************************

;******************************************************

main setf Hbyte
0000 2B21 setf Lbyte
0001 2B20 ; 16 bit binary num = 0xffff
0002 B01F call B2_BCD_Looped ; after conversion the Deci
0003 2B21 ; in R0, R1, R2 = 06,55,35
0004 2B20 setf Hbyte
0005 E03F setf Lbyte
0006 B006 call B2_BCD_Straight ; same as above, but straig
0007 0124 movlw 0x6
0008 B055 movwf Hbyte
0009 0123 movlw 0x55
000A B035 movwf Lbyte
000B 0122 movlw 0x35
000C E082 movwf R2 ; setf R0R1R2 = 65535

; BCD to B after conversion Hbyte =

; and Lbyte = 0xff
Math Routines

000D B099 movlw 0x99
000E 0126 movwf Num1
000F B099 movlw 0x99
0010 0127 movwf Num2 ; setf Num1 = Num2 = 0x99

0011 E093 call BCDAdd ; after addition, Num2 = 98

0012 B063 movlw 0x63
0013 E015 call BinBCD ; setf Wreg = 63 hex

0014 C014 self goto self

;**********************************************************
; Binary To BCD Conversion Routine (8 bit)

; This routine converts the 8 bit binary number in th
to a 2 digit BCD number in location BCD( compacted BCD Co

; The least significant digit is returned in location
the most significant digit is returned in location MSD.

; Performance:

; Program Memory : 10
; Clock Cycles : 62 (worst case when W =

; i.e. max Decimal nu

;**********************************************************

BinBCD

0015 2926 clrf BCD
0016 B1F6 again
0017 9004 addlw -10
0018 C01B bfss _carry
0019 1526 goto swapBCD
001A C016 incf BCD
001B B10A goto again

swapBCD

001C 1D26 addlw 10
001D 0926 swapf BCD
001E 0002 iorwf BCD

return

;**********************************************************

; Binary To BCD Conversion Routine (16 Bit)

; (LOOPED Version)

; This routine converts a 16 Bit binary Number to a 5
; BCD Number.

; The 16 bit binary number is input in locations Hbyte
; Lbyte with the high byte in Hbyte.
; The 5 digit BCD number is returned in R0, R1 and R2
Math Routines

; containing the MSD in its right most nibble.
;
; Performance :
; Program Memory : 32
; Clock Cycles : 750
;
;**********************************************************

; B2_BCD_Looped
001F 8404
    bnf    _fs0
0020 8504
    bnf    _fs1 ; set fsr0 for no auto inc
;
0021 8804
    bcf    _carry
0022 2925
    clrf    count
0023 8425
    bnf    count,4 ; set count = 16
0024 2924
    clrf    R0
0025 2923
    clrf    R1
0026 2922
    clrf    R2
loop16a
0027 1B20
    rlcf    Lbyte
0028 1B21
    rlcf    Hbyte
0029 1B22
    rlcf    R2
002A 1B23
    rlcf    R1
002B 1B24
    rlcf    R0
;
002C 2725
    dcfsnz    count
return
adjDEC
002D 0002
002E B022
    movlw    R2 ; load R2 as indirect addr
002F 0101
    movwf    fsr0
0030 E036
    call    adjBCD
;
0031 1501
    incf    fsr0
0032 E036
    call    adjBCD
;
0033 1501
    incf    fsr0
0034 E036
    call    adjBCD
;
0035 C027
    goto    loop16a
;
adjBCD
0036 6000
    movfp   indf0, wreg
0037 B103
    addlw   0x03
0038 9B00
    btfsc   wreg, 3 ; test if result > 7
0039 0100
    movwf   indf0
003A 6000
    movfp   indf0, wreg
003B B130
    addlw   0x30
003C 9F00
    btfsc   wreg, 7 ; test if result > 7
003D 0100
    movwf   indf0 ; save as MSD
003E 0002
return
;
;**********************************************************

; Binary To BCD Conversion Routine (16 Bit)
;
; (Partial Straight Line Version)
;
; This routine converts a 16 Bit binary Number to a 5
; BCD Number.
;
; The 16 bit binary number is input in locations Hbyte
; Lbyte with the high byte in Hbyte.
; The 5 digit BCD number is returned in R0, R1 and R2

© 1993 Microchip Technology Incorporated
; containing the MSD in its right most nibble.
;
; Performance :
;  Program Memory : 44
;  Clock Cycles : 572
;
;******************************************************************************

;
B2_BCD_Straight

003F B404
  bsf  _fs0

0040 8504
  bsf  _fs1 ; set fsr0 for no auto inc

;
0041 8804
  bcf  _carry

0042 2925
  clrf  Count

0043 8425
  bsf  count,4 ; set count = 16

0044 2924
  clrf  R0

0045 2923
  clrf  R1

0046 2922
  clrf  R2

loop16b

0047 1B20
  rlcf  Lbyte

0048 1B21
  rlcf  Hbyte

0049 1B22
  rlcf  R2

004A 1B23
  rlcf  R1

004B 1B24
  rlcf  R0

004C 2725
  dcf anz count ; DONE

004D 0002
  return

004E 8022
  movlw  R2 ; load R2 as indirect addr

004F 0101
  movwf  fsr0

; adjustBCD

0050 6000
  movfp  indf0,wreg

0051 B103
  addlw  0x03

0052 9B00
  btfsc  wreg,3 ; test if result > 7

0053 6000
  movwf  indf0

0054 6000
  movfp  indf0,wreg

0055 B130
  addlw  0x30

0056 9F00
  btfsc  wreg,7 ; test if result > 7

0057 0100
  movwf  indf0 ; save as MSD

; incf  fsr0

; adjustBCD

0059 6000
  movfp  indf0,wreg

005A B103
  addlw  0x03

005B 9B00
  btfsc  wreg,3 ; test if result > 7

005C 0100
  movwf  indf0

005D 6000
  movfp  indf0,wreg

005E B130
  addlw  0x30

005F 9F00
  btfsc  wreg,7 ; test if result > 7

0060 0100
  movwf  indf0 ; save as MSD

; incf  fsr0

; adjustBCD

0061 1501
  incf  fsr0

; adjustBCD

0062 6000
  movfp  indf0,wreg

0063 B103
  addlw  0x03

0064 9B00
  btfsc  wreg,3 ; test if result > 7

0065 0100
  movwf  indf0

0066 6000
  movfp  indf0,wreg

0067 B130
  addlw  0x30

0068 9F00
  btfsc  wreg,7 ; test if result > 7

0069 0100
  movwf  indf0 ; save as MSD

; goto  loop16b

;******************************************************************************

; BCD To Binary Conversion
This routine converts a 5 digit BCD number to a 16
number.
The input 5 digit BCD numbers are assumed to be in R0, R1 & R2 with R0 containing the MSD in its right most
The 16 bit binary number is output in registers Hbyte & Lbyte respectively.

The method used for conversion is:
input number X = abcd X 10|10|10|10a+b|c+d|e

Performance:
Program Memory: 30
Clock Cycles: 112

BCDtoB
006B B50F andlw 0x0f
006C 0F20 addwf Lbyte
006D 9804 btfsc _carry
006E 1521 incf Hbyte
006F 8804 bcf _carry
0070 1A20 rlf Lbyte,w
0071 0127 movf Ltemp
0072 1A21 rlf Hbyte,w
0073 0126 movf Htemp
0074 8804 bcf _carry
0075 1B20 rlf Lbyte
0076 1B21 rlf Hbyte
0077 8804 bcf _carry
0078 1B20 rlf Lbyte
0079 1B21 rlf Hbyte
007A 8804 bcf _carry
007B 1B20 rlf Lbyte
007C 1B21 rlf Hbyte
007D 6027 movfp Ltemp,wreg
007E 0F20 addwf Lbyte
007F 6026 movfp Htemp,wreg
0080 1121 addwf Hbyte
0081 0002 return

0082 2921 clrf Hbyte
0083 6024 movfp RO,wreg
0084 B50F andlw 0x0f
0085 0120 movwf Lbyte
0086 E06F call mpy10a
0087 1C23 swaf R1,w
0088 E06B call mpy10b
0089 6023 movfp R1,wreg
008A E06B call mpy10b
008B 1C22 swaf R2,w
Math Routines

008C E06B  call mpy10b ; result = 10[10[10[10a+b]+)

008D 6022  movfp R2, wreg
008E B50F  andlw 0x0f
008F 0F20  addwf lbyte
0090 9804  btfsc _carry
0091 1521  incf Hbyte ; result = 10[10[10[10a+b]
0092 0002  return ; BCD to binary conversion

;******************************************************************************
; Unsigned BCD Addition
;
; This routine performs a 2 Digit Unsigned BCD Addition
;
; It is assumed that the two BCD numbers to be added are in
; locations Num1 & Num2. The result is the sum of Num1+Num2
; and is stored in location Num2 and the overflow carry is
; in location Num1
;
; Performance :
; Program Memory : 5
; Clock Cycles : 5
;******************************************************************************

;**********************************************************
BCDAdd

0093 6026  movfp Num1, wreg
0094 0E27  addwf Num2,w ; perform binary addition
0095 2F27  daw Num2 ; adjust for BCD addition
0096 2926  clr Num1
0097 1926  rlc Num1 ; set Num1 = carry bit
0098 0002  return

;******************************************************************************
;
END

Errors : 0
Warnings : 0
#define PAGE EJECT

TITLE "Binary Floating Arithmetic Routines For PIC17C42 : Ver 1.0"
LIST P=17C42, C=80, L=0, T=ON, R=DEC

;**********************************************************
Binary Floating Point Addition, Subtraction
Multiplication Routines

; Mantissa = 16 bits
Exponent = 8 bits  { exponent is binary and not decimal
i.e a number ABCD EXP(X) = OxABCD

; Before calling any of the following floating point
it is required to set Indirect Register 0 (FSR0) for
No-Autoincrement (i.e. Set bits FS0 & FS1 in ALUSTA to

;**********************************************************

CBLOCK Ox20
ACCaLO, ACCaHI, EXPa
ACCbLO, ACCbHI, EXPb
ACCcLO, ACCcHI
ACCdLO, ACCdHI
temp, sign
ENDC
Model6 equ TRUE ; Change this to FALSE for 32 bit

ORG 0x0000

;**********************************************************
Floating Point Routines Test Program

;**********************************************************
main

bsf _fs0 ; set FSR0 for no autoincr
bsf _fs1

; call loadAB ; result of adding ACCb(EXP

; Here Accb = 403F, EXPb =
Math Routines

0004 E009 call loadAB ; result of subtracting ACC
0005 E016 call F_sub ; Here Accb = 7F7F, EXPb =

0006 E009 call loadAB ; result of multiplying ACC
0007 E03B call F_mpy ; Here ACCb = FF7E, EXPb =

0008 C008 self goto self

; Load constant values to (ACCa, EXPa) & (ACCb, EXPb) for

; loadAB
0009 B001 movlw 0x01
000A 0121 movwf ACCaHI
000B B0FF movlw 0xff ; loads ACCa = 01FF EXP(4
000C 0120 movwf ACCaLO
000D B004 movlw 0x04
000E 0122 movwf EXPa

; movlw 0x7f
000F B07F movwf ACCbHI
0010 0124 movlw 0xff ; loads ACCb = 7fff EXP(6
0011 B0FF movwf ACCbLO
0012 0123 movlw 0x06
0013 B006 movwf EXPb
0014 0125 return

;*******************************************************************************
;
; Floating Point Subtraction ( ACCb - ACCa -> ACCb )
;
; Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16
; (a) Load the 1st operand in location ACCaLO & ACCaHI
; the 8 bit exponent in EXPa .
; (b) Load the 2nd operand in location ACCbLO & ACCbHI
; the 8 bit exponent in EXPb .
; (c) CALL F_sub
; (d) The result is in location ACCbLO & ACCbHI ( 16 b
; the 8 bit exponent in EXPb.
;
;*******************************************************************************

; F_sub
0016 B020 movlw ACCaLO
0017 0101 movwf fsr0
0018 E075 call negate ; At first negate ACCa; Th

;*******************************************************************************
;
; Floating Point Addition ( ACCb + ACCa -> ACCb )
;
; Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bi
; (a) Load the 1st operand in location ACCaLO & ACCaHI
; the 8 bit exponent in EXPa.
Math Routines

(b) Load the 2nd operand in location ACCbLO & ACCbHI

; the 8 bit exponent in EXPb.
; (c) CALL F_add
; (d) The result is in location ACCbLO & ACCbHI (16 b

; the 8 bit exponent in EXPb

;*******************************************************************************

F_add

 0019 6022  movfp   EXPa, wreg
 001A 3125  cpfseq   EXPb
 001B C01D  goto   Lbl1
 001C C026  goto   noAddNorm ; if exponents are equal

 001D 3025  cpfslt   EXPb
 001E E09E  call    F_swap
 0020 0525  movfp   EXPa, wreg
 0021 E030  scloop  subwf   EXPb

 0022 1F25  call    addNorm
 0023 C021  goto    scloop
 0024 6022  movfp   EXPa, wreg
 0025 0125  movwf    EXPb

 0026 6021  movfp   ACCaHI, wreg
 0027 0824  lorwf    ACCbHI, w
 0028 012B  movwf    sign ; save the sign (MSB sta
 0029 E036  call    D_add  ; compute double precision
 002A 972B  btfss   ACCaHI, MSB
 002B 9724  btfss   ACCbHI, MSB
 002C 0002  return
 002D 1525  incf    EXPb
 002E 8804  bcf     _carry
 002F C033  goto    shftR

; addNorm

 0030 8804  bcf     _carry
 0031 9F24  btfsc    ACCbHI, MSB
 0032 8004  bcf     _carry ; set carry if < 0

; shftR

 0033 1924  rrcf    ACCbHI
 0034 1923  rrcf    ACCbLO
 0035 0002  return

;*******************************************************************************

; Double Precision Addition
;
D_add

 0036 6020  movfp   ACCaLO, wreg
 0037 0F23  addwf    ACCbLO ;adddf lab
 0038 6021  movfp   ACCaHI, wreg
 0039 1124  addwfc   ACCbHI ;addwf msb with carry
 003A 0002  return

;*******************************************************************************

; Binary Floating Point Multiplication
;
; Multiplication :
;   ACCb(16 bits)EXP(b) * ACCa(16
where, \( \text{EXP}(x) \) represents an 8 bit exponent.

(a) Load the 1st operand in location \( \text{ACCaLO} \) \& \( \text{ACCaHI} \) an 8 bit exponent in location

(b) Load the 2nd operand in location \( \text{ACCbLO} \) \& \( \text{ACCbHI} \) an 8 bit exponent in location

(c) CALL \_mpy

(d) The 16 bit result overwrites \( \text{ACCbLO} \) \& \( \text{ACCbHI} \) is stored in \( \text{EXPb} \) and the results are normal

NOTE: If one needs to get a 32 bit product \( \& \) an 8 bit exponent, reassemble the program after changing the line " Model6 equ FALSE ". If this option is chosen, then the 32 bit result is stored in \( \text{ACCbHI}, \text{ACCbLO}, \text{ACCcHI}, \text{ACCcLO} \) and the 8 bit exponent. This method (with " Model6 equ FALSE ") is NOT recommended.

If a 32 bit mantissa is desired, set MODE16 equ FALSE

******************************************************

F\_mpy

| 003B 007D | call \_S\_SIGN |
| 003C 0064 | call setup |
| mloop | |
| 003D 8804 | | bcf _carry |
| 003E 1929 | | rrcf ACCdHI |
| 003F 1928 | | rrcf ACCdLO |
| 0040 9804 | | btfsc _carry |
| 0041 E036 | | call D\_add |
| 0042 1924 | | rrcf ACCbHI |
| 0043 1923 | | rrcf ACCbLO |
| 0044 1927 | | rrcf ACCcHI |
| 0045 1926 | | rrcf ACCcLO |
| 0046 172A | | decfsz temp |
| | | ;loop until all bits change |
| 0047 C03D | | goto mloop |
| 0048 6022 | | movfp EXPa, wreg |
| 0049 0F25 | | addwf EXPb |
| | | ; |
| 004A 3324 | | tstfsz ACCbHI |
| 004B C05B | | goto finup |
| 004C 3323 | | tstfsz ACCbLO |
| 004D C055 | | goto Shft08 |
| | | ; |
| 004E 6027 | | movfp ACCcHI, wreg |
| 004F 0124 | | movwf ACCbHI |
| | | ; if ACCb == 0, then move |
| 0050 6026 | | movfp ACCcLO, wreg |
| 0051 0123 | | movwf ACCbLO |
| 0052 B010 | | movlw 16 |
| 0053 0F25 | | addwf EXPb |
| 0054 C05B | | goto finup |
| | | ; |
| 0055 6023 | | movfp ACCbLO, wreg |
| | | |

© 1993 Microchip Technology Incorporated
Math Routines

movwf ACCbHI
movfp ACCcHI,wreg
movwf ACCbLO
movlw 8
addwf EXPb

; endif

movlw ACCcLO
movwf fsrO
call negate
movlw ACCbLO
movwf fsrO
call negate

; normalize floating point

; setup
clr temp = 16
set temp = 16
move ACCb to ACCd

; Double Precision Negate Routines

negateAlt

movfp indfO,wreg
bcf _fsl
neqw indfO
bsf _fsl
movfp indfO,wreg
clrf indfO
subwf indfO

; negate

comf indfO
bcf _fsl
incf IndfO
bsf _fsl
btfsc _z
decf indfO
comf indfO

; Check Sign of the Number, if so negate and set the SIGN

S_SIGN

movfp ACCaHI,wreg
xorwf ACCbHI,w
Math Routines

```
007F 012B    movwf    sign
0080 9724    btfss    ACCbHI,MSB ; if MSB set go & nega
                  goto    chek_A

0081 C085    ;
0082 8023    movlw    ACCbLO
0083 0101    movwf    fsr0
0084 E075    call    negate

0085 9721    ; chek_A
0086 0002    btfss    ACCaHI,MSB ; if MSB set go & nega
                  return
0087 B020    movlw    ACCaLO
0088 0101    movwf    fsr0
0089 E075    call    negate
008A 0002    return

;**********************************************************************

; Normalize Routine
; Normalizes ACCb for use in floating point calculations.
; Call this routine as often as possible to minimize the lost
; of precision. This routine normalizes ACCb so that the
; mantissa is maximized and the exponent minimized.
;**********************************************************************

; F_norm normalize ACCb
008B 3324    tstfsz    ACCbHI
008C C090    goto    C_norm
008D 3323    tstfsz    ACCbLO
008E C090    goto    C_norm
008F 0002    return

; C_norm
0090 9E24    btfss    ACCbHI,MSB-1
0091 0002    return
0092 E095    call    shftSL
0093 0725    decf    EXPb
0094 C090    goto    C_norm

; shftSL
0095 8804    bcf    _carry
                ; if Model6
0096 1B26    rlcf    ACCcLO
0097 1B27    rlcf    ACCcHI
                endif
0098 1B23    rlcf    ACCbLO
0099 1B24    rlcf    ACCbHI
009A 8F24    bcf    ACCbHI,MSB
009B 9E04    btfsc    _carry
009C 8724    bsf    ACCbHI,MSB
009D 0002    return

;**********************************************************************

; Swap ACCa & ACCb [(ACCa,EXPa) <-> (ACCb,EXPb)]
; F_swap
009E 6021    movfp    ACCaHI,wreg
009F 012A    movwf    temp
00A0 6024    movfp    ACCbHI,wreg ; ACCaHI <-> ACCbHI
00A1 0121    movwf    ACCaHI
00A2 602A    movfp    temp,wreg
```
Math Routines

;**********************************************************
; Normalizes A Floating Point Number
; The number is assumed to be (LowByte, HighByte, Ex)

; consecutive locations. Before calling this routine, the a
; of the LowByte should be loaded into FSR0 (indirect regis

;**********************************************************

; Normalize

NextNorm1
inff
movf
_out
movf
_xor
bcf
_shiftNorm

NextNorm2
inff
movf
_out
movf
_xor
bcf
_shiftNorm

END
Implementing IIR Digital Filters

INTRODUCTION

This application note describes the implementation of various digital filters using the PIC17C42, the first member of Microchip's 2nd generation 8-bit microcontrollers. The PIC17C42 is a very high speed 8-bit microcontroller with an instruction cycle time of 250ns (@ 16 MHz input clock). Even though PIC17C42 is an 8-bit machine, its high speed & efficient instruction set allows implementation of digital filters for practical applications. Traditionally digital filters are implemented using expensive Digital Signal Processors (DSPs). In a system the DSP is normally a slave processor being controlled by a either an 8- or 16-bit microcontroller. Where sampling rates are not high (esp. in mechanical control systems), a single chip solution is possible using the PIC17C42.

This application note provides a few examples of implementing digital filters. Example code for 2nd order Infinite Impulse Response (IIR) filters is given. The following type of filters are implemented:

- Low Pass
- High Pass
- Band Pass
- Band Stop (notch) filter

This application note does not explain how to design a filter. Filter design theory is well established and is beyond the scope of this application note. It is assumed that a filter is designed according to the desired specifications. The desired digital filters may be designed using either standard techniques or using commonly available digital filter design software packages.

Finite Impulse Response (FIR) filters have many advantages over IIR filters, but are much more resource intensive (both in terms of execution time and RAM). On the other hand, IIR filters are quite attractive for implementing with the PIC17C42 resources. Especially where, phase information is not so important, IIR filters are a good choice (FIR filters have a linear phase response). Of the various forms used for realizing digital filters (like, Direct form, Direct II form, Cascade form, Parallel, Lattice structure, etc.) the Direct II form is used in this application note. It is easy to understand and simple macros can be built using these structures.

THEORY OF OPERATION

Digital filters in most cases assume the following form of relationship between the output and input sequences.

\[ y(n) = \sum_{i=0}^{M} a_i y(n - i) + \sum_{j=0}^{N} b_j x(n - j) \]

The above equation basically states that the present output is a weighted sum of the past inputs and past outputs. In case of FIR filters, the weighted constants \( a_i = 0 \) and in case of IIR filters, at least one of the \( a_i \) constant is non zero. In case of IIR, the above formula may be re written in terms of Z transform as:

\[ H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} \]

The above equation can further be rewritten in difference equation format as follows:

\[ y(n) = \sum_{i=1}^{M} a_i y(n - i) + \sum_{j=0}^{N} b_j x(n - j) \]

Realization of the above equation is called as the Direct Form II structure. For example, in case of a second order structure, \( M=N=2 \), gives the following difference equations:

\[ \begin{align*}
  d(n) &= x(n) + a_1 d(n-1) + a_2 d(n-2) \\
  y(n) &= b_0 d(n) + b_1 d(n-1) + b_2 d(n-2)
\end{align*} \]

The above difference equations may be represented as shown in Figure 1.

FIGURE 1 - 2ND ORDER DIRECT FORM II STRUCTURE (TRANSPOSED)
Implementing IIR Digital Filters

The structure as shown in Figure 1 may be cascaded to attain a higher order filter. For example, if two stages are cascaded together, a 4th Order IIR Filter is obtained. This way, the output of the 1st stage becomes the input to the second stage. Multiple order filters are thus implemented by cascading a 2nd order filter structure as shown in Figure 1.

IMPLEMENTATION

A 4th order IIR Filter is implemented by cascading two structures shown in Figure 1. The output Y (output of each filter stage) is computed by direct implementation of Equations (1) & (2). Since each stage is similar algorithmically, it is implemented as a Macro using Microchip’s Assembler/Linker for PIC17C42. This Macro (labelled “BIQUAD”) is called twice for implementing a 4th order filter. The output of the 1st stage is directly fed to the input of the second stage without any scaling.

Scaling is required depending on the particular application. The user can modify the code very easily without any penalty on speed. Also saturation arithmetic is not used. Overflows can be avoided by limiting the input sequence amplitude. All numbers are assumed to be 16 bits in Q15 format (15 decimal points, MSB is sign bit). Thus the user must scale & sign extend the input sequence accordingly. For example, if the input is from a 12 bit A/D converter, the user must sign extend the 12 bit input if bit 11 is a one.

The Biquad macro is a generic macro and can be used for all IIR filters whether it is Low Pass, High Pass, Band Pass or Band Stop. A general purpose 16x16 multiplier routine is also provided. This routine is implemented as a straight line code for speed considerations.

The 4th order IIR filter implemented is a Low Pass Filter with the specifications as shown in Table 1.

TABLE 1 - FILTER CONSTANTS

<table>
<thead>
<tr>
<th>Co-efficients</th>
<th>a1</th>
<th>a2</th>
<th>b0</th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>-0.133331</td>
<td>0.167145</td>
<td>0.285431</td>
<td>0.462921</td>
<td>0.285431</td>
</tr>
<tr>
<td>Stage 2</td>
<td>0.147827</td>
<td>0.765900</td>
<td>0.698273</td>
<td>0.499908</td>
<td>0.698273</td>
</tr>
</tbody>
</table>

The above filter co-eficients (5 per stage) are quantized to Q15 format (i.e they are multiplied by 32768) and saved in program memory (starting at label “coeff_lpass”). The constants for both the stages are read into data memory using TLRD & TABLRD instructions in the Initialization Routine (labelled “initFilter”). The user may read the coefficients of only one stage at a time and save some RAM at the expense of speed.

The sample 4th order Low Pass IIR Filter is tested by analyzing the impulse response of the filter. An impulse signal is fed as input to the filter. This is simulated by forcing the input to the filter by a large quantity (say 7F00h) on the first input sample, and the all zeros from the 2nd sample onwards. The output sequence is the filter’s impulse response and is captured into the PICMASTER’s (Microchip’s Universal In-Circuit Emulator) real time trace buffer. This captured data from PICMASTER is saved to file and analyzed. Analysis was done using MathCad for Windows and DSP Analysis program from Burr-Brown (DSPLAY). The Fourier Transform of this impulse response of the filter should display the Filter’s frequency response, in this case being a Low Pass type. The plots of the impulse response and the frequency response is shown in figures below.

TABLE 2 - FILTER COEFFICIENTS

<table>
<thead>
<tr>
<th>Co-efficients</th>
<th>a1</th>
<th>a2</th>
<th>b0</th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>-0.133331</td>
<td>0.167145</td>
<td>0.285431</td>
<td>0.462921</td>
<td>0.285431</td>
</tr>
<tr>
<td>Stage 2</td>
<td>0.147827</td>
<td>0.765900</td>
<td>0.698273</td>
<td>0.499908</td>
<td>0.698273</td>
</tr>
</tbody>
</table>

The Low Pass Filter is designed using a digital filter design package (DFDP by Atlanta Signal Processors Inc.). The filter package produces filter constants of the structure shown in Table 1. Table 2 shows the filter co-efficients that are obtained for the above Low Pass filter specification.

FIGURE 2 - IMPULSE RESPONSE CAPTURED FROM PIC-MASTER
Implementing IIR Digital Filters

FIGURE 3 - SPECTRUM COMPUTED FROM IMPULSE RESPONSE

PERFORMANCE

The resource requirements for filter implementations using PIC17C42 is given below. These numbers can be used to determine whether a higher order filter can be executed in real time. The same information may be used to determine the highest sampling rate possible.

FILTER APPLICATIONS

Digital filters are also needed in Process Control where notch filters and low pass filters are desired because the signals from sensors are transmitted over long lines, especially in a very noisy environment. In this case typically a notch filter (centering 50Hz or 60Hz) is used. In case of eliminating background noise, a band stop filter (e.g. 40Hz to 120Hz) is used. The sample code given in this application note can be used to design a feedback control system's digital compensator. For example, a typical DC Motor's digital compensator (like dead beat compensator) is of second order and has the same filter structure as is implemented in this application note.

TABLE 3 - RESOURCE REQUIREMENTS

<table>
<thead>
<tr>
<th>Timing (Cycles)†</th>
<th>Program Memory (locations)†</th>
<th>RAM (File Registers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#of Filter Stages*775 + 16</td>
<td>#of Filter Stages*68 + 290</td>
<td>#of Filter Stages*16+16</td>
</tr>
</tbody>
</table>

†: The above numbers do not include the initialization routine.

TABLE 4 - RESOURCE REQUIREMENTS

<table>
<thead>
<tr>
<th>Filter Order</th>
<th>Cycles</th>
<th>Real Time (@ 16 MHz)</th>
<th>Maximum Sampling</th>
<th>Program Memory†</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>791</td>
<td>197.75 uSec</td>
<td>5.05 Khz</td>
<td>358</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>1566</td>
<td>391.5 uSec</td>
<td>2.55 Khz</td>
<td>426</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>2341</td>
<td>585.25 uSec</td>
<td>1.7 Khz</td>
<td>494</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>3116</td>
<td>779.0 uSec</td>
<td>1.28 Khz</td>
<td>562</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>3891</td>
<td>972.75 uSec</td>
<td>1.0 Khz</td>
<td>630</td>
<td>96</td>
</tr>
</tbody>
</table>

†: The above numbers do not include the initialization routine.

© 1993 Microchip Technology Inc.

Author: Amar Palacherla
Logic Products Division
Implementing IIR Digital Filters

APPENDIX A: IIR.ASM

MPASM B0.54
Digital IIR Filter Using PIC17C42

TITLE "Digital IIR Filter Using PIC17C42"
LIST P=17C42, C=120, T=ON, R=DEC, N=0

; include "17c42.h"

; include "17c42.mac"

; include "17c42iir.mac"

;*******************************************************************

PIC17C42 MACRO

Macro For A Bi-Quad IIR Filter
2nd order Direct Form (Transposed) Type
Filter co-efficients B0 & B2 are assumed equal
The difference equations for each cascade section is given by:
Y(n) = B0*D(n) + B1*D(n-1) + B2*D(n-2)
D(n) = X(n) - A1*D(n-1) - A2*D(n-2)
where X(n) = input sample, Y(n) = output of filter
and A1, A2, B0, B1, B2 are the Filter Co-efficients
The above difference equations are only for 1 section of a
2nd order Direct Form II Filter structure (IIR)

NOTE:
It is possible to design the above structures
such that the co-efficients B0 = B2. If this is the
case,
Y(n) = B0*[D(n) + D(n-2)] = B2*[D(n) + D(n-2)]
This way, one multiplication can be avoided
If a 4th order filter is to be implemented, the output of
the 1st structure should be input to the 2nd cascade section
Timing (WORST CASE):
59+4*179 = 775 Cycles
(194 uS @ 16 Mhz)
Program Memory:
63 locations

;*******************************************************************

BO_EQUALS_B2 equ TRUE

;*******************************************************************
Parameters to BIQUAD Macro:
Filter Constants A1, A2, B0, B1, B2
& D(n), D(n-1), D(n-2), filter stage #

BIQUAD MACRO Ax1,Ax2,Bx0,Bx1,Dn,Dn_1,Dn_2,stage

; Compute Ax2*D(n-2)

© 1993 Microchip Technology Inc.
Implementing IIR Digital Filters

; MOVFP16 Dn_2,AARG ; D(n-2) = multiplier
MOVFP16 Ax2,BARG ; A2 = multiplicand
call DblMult ; (ACCd,ACCc) = A2*D(n-2)

; Add product to output of 1st section
; Save result in 32 bit Accumulator

ADD32 DPX,ACC

; Compute A1*D(n-1)

MOVFP16 Dn_1,AARG ; AARG = D(n-2) = multiplier
MOVFP16 Ax1,BARG ; BARG = A2 = multiplicand
call DblMult ; (ACCd,ACCc) = A1*D(n-1)

; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
; multiplications already done, so simply perform a 32 bit add
; of previously obtained multiplication results

ADD32 DPX,ACC ; ACC = A1*D(n-1) + A2*D(n-2) + (output of 1st

; save the upper 16 bits of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication

rlcf ACC+B1,w
rlcf ACC+B2,w
movwf Dn
rlcf ACC+B3,w ; decimal adjust (mult by 2)
movwf Dn+B1

; Compute B2 * [D(n) + D(n-2)]

if B0_EQUALS B2

ADD16ACC Dn_2,Dn,AARG ; AARG = Dn + D(n-2) = multiplier
MOVFP16 Bx0,BARG ; BARG = A2 = multiplicand
call DblMult ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
MOVFP32 DPX,ACC

else

MOVFP16 Bx0,BARG
MOVFP16 Dn,AARG
call DblMult ; B0*D(n)
MOVFP32 DPX,ACC

MOVFP16 Bx2,BARG
MOVFP16 Dn_2,AARG
call DblMult ; B2*D(n-2)
ADD32 DPX,ACC
endif

; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
; This way in the next iteration D(n-2) is equal to the present D(n-1)

movfp Dn_1,AARG+B0
movfp AARG+B0,Dn_2 ; Shift down D(n-1)
movfp Dn_1+B1,AARG+B1
movfp AARG+B1,Dn_2+B1 ; AARG = D(n-1) = multiplier

MOVFP16 Bx1,BARG ; BARG = B1 = multiplicand
call DblMult ; (ACCd,ACCc) = B1*D(n-1)

; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
; = B1*D(n-1) + B0*[D(n) + D(n-2)]
Implementing IIR Digital Filters

; Since all multiplications are already done, simply perform a
; 32 bit addition
;
ADD32    DPX,ACC       ; ACC = Bl*D(n-1) + B2*D(n-2) + B0*D(n)
;
; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;
MOV16    Dn,Dn_1       ; Shift down D(n) to D(n-1)
;
ENDM

;***************************************************************************
;  Second Order Direct Form IIR Filter
; 
; In the code given below, a 4th order IIR Elliptic Lowpass Filter
; is implemented. Other order filters may be implemented by
; taking the following example code as a basis.
;
; The specifications of the filter are :
;
; Sampling Frequency = 2.0 KHz
;
; Filter Type = 4th Order Elliptic Lowpass Filter
;
; Band1            Band2
; Lower Band Edge  0.0           600 Hz
; Upper Band Edge  500 Hz         1 KHz
; Nominal Gain     1.0            0.0
; Nominal Ripple   0.01           0.05
; Ripple in dB     0.07830        -26.75
;
; The Filter Co-efficients for the above specifications
; of the filter are computed as follows :
;
; 1st Section :
; A11 = -0.133331
; A12 = 0.167145
; B10 = 0.285431
; B11 = 0.462921
; B12 = 0.285431
;
; 2nd Section
; A21 = 0.147827
; A22 = 0.765900
; B20 = 0.698273
; B21 = 0.499908
; B22 = 0.698273
;
; Performance (WORST Case):
;
; Cycles = #of Filter Stages*775 + 16
; = 2*775+16 = 1566 Cycles
; ( 391 uSec)
; per each sample. Initialization
; time after reset is not counted
; Timing measured with B0_EQUALS_B2
; set to TRUE (see BIQUAD Macro for
; explanation
;
; Program Memory :
; = 16+ # of Filter Stages * (BIQUAD
; + filter co-efficients)
; + multiplier
; = 16+2*(63+5)+274 = 421 locations
; (excluding initialization)
Implementing IIR Digital Filters

; RAM usage = 48 file registers
; RAM usage/each additional stage = 16 file regs
;
; This time is less than 2 Khz (500 uSec),
; which means real time filtering is possible
;
;*******************************************************************
;*******************************************************************

CBLOCK 0

0000 0004
BU,B1,B2,B3
ENDC

;*******************************************************************

CBLOCK 0x18

0018 0004
DPX,DPX1,DPX2,DPX3 ; arithmetic accumulator
001C 0004
AARG,AARG1,BARG,BARG1 ; multiply arguments
ENDC

;*******************************************************************

CBLOCK

0020 0002
Dn1, Dn1_Hi
0022 0002
Dn1_1, Dn1_1_Hi
0024 0002
Dn1_2, Dn1_2_Hi
0026 0000

0026 0002
Dn2, Dn2_Hi
0028 0002
Dn2_1, Dn2_1_Hi
002A 0002
Dn2_2, Dn2_2_Hi
ENDC

;*******************************************************************

CBLOCK

002C 0002
A11, A11_Hi
002E 0002
A12, A12_Hi
0030 0002
B10, B10_Hi
0032 0002
B11, B11_Hi
0034 0002
B12, B12_Hi
0036 0000

0036 0002
A21, A21_Hi
0038 0002
A22, A22_Hi
003A 0002
B20, B20_Hi
003C 0002
B21, B21_Hi
003E 0002
B22, B22_Hi
ENDC

;*******************************************************************

CBLOCK

0040 0002
X, X1 ; 16 bits of input stream
0042 0002
Y, Y1 ; 16 bits of filter output
0044 0000

0044 0004
ACC, ACC1, ACC2, ACC3 ; 32 bit accumulator for computations
ENDC

;*******************************************************************

0002 .set FltStage 2
000A .set NumCoeff (5*FltStage) 5 Co-eff per stage
;
0001 .set LPASS TRUE
0000 .set HPASS FALSE
0000 .set BPASS FALSE
0000 .set BSTOP FALSE
;
0001 .set SIGNED TRUE ; Set This To 'TRUE' for signed multi
; and 'FALSE' for unsigned.
;
;*******************************************************************

Test Program For Low Pass Filter

;*******************************************************************
Implementing IIR Digital Filters

; ORG 0x0000
;
start
   call initFilter
   movlw 0x00
   movwf X
   movlw 0x7f
   movwf X+B1
NextPoint
   call IIR_Filter
   tlwt _LOW,Y
tracePoint
   tablwt _HIGH,0,Y+B1
   nop
   clrf X
   clrf X+B1
   goto NextPoint
self goto self

;****************************************************************************
;
initFilter

; At first read the Filter Co-efficients from Prog. Mem to Data RAM
;
   if LPASS
      movlw _coeff_lpass
      movwf tblptrl
      movlw page _coeff_lpass
      movwf tblptrh
   endif
   if HPASS
      movlw _coeff_hpass
      movwf tblptrl
      movlw page _coeff_hpass
      movwf tblptrh
   endif
   if BPASS
      movlw _coeff_bpass
      movwf tblptrl
      movlw page _coeff_bpass
      movwf tblptrh
   endif
   if BSTOP
      movlw _coeff_bstop
      movwf tblptrl
      movlw page _coeff_bstop
      movwf tblptrh
   endif

;
   movlw All
   movwf fsr0
   bsf _fs0
   bcf _fs1 ; auto increment

; Read Filter Co-efficients from Program Memory
;
   movlw NumCoeff
   tablrd _LOW,INC,All ; garbage
NextCoeff
   tlrd _LOW,indf0
   tablrd _HIGH,INC,indf0
   decfsz wreg
   goto NextCoeff
Implementing IIR Digital Filters

; Initialize "Dn"s to zero

001B B020
001C 0101
001D B07C

NextClr
001E 2900
001F 1700
0020 C01E

; return

;******************************************************************************
; 1st Cascade Section
;******************************************************************************

IIR_Filter

; Compute D(n) = X(n) + A1*D(n-1) + A2*D(n-2)
; Since the filter constants are computed in Q15 format,
; X(n) must be multiplied by 2**15 and then added to the
; other terms.
;
; Move input to accumulator after proper scaling

0022 8804
0023 1941
0024 1940
0025 2900
0026 1900
0027 0145
0028 6040
0029 0146
002A 6041
002B 0147

; 1st Biquad filter section
;
BIQUAD A11,A12,B10,B11,Dn1,Dn1_1,Dn1_2,1

; Compute A12*D(n-2)

002C 7C24
002D 7D25

002E 7E2E
002F 7F2F

0030 E0AF

; Add product to output of 1st section
; Save result in 32 bit Accumulator

0031 6018
0032 0F44

; get lowest byte of DPX into w
; add lowest byte of ACC, save in
Implementing IIR Digital Filters

```
ACC(B0)
0033 6019
0034 1145
ACC(B1)
0035 601A
0036 1146
ACC(B2)
0037 601B
0038 1147
ACC(B3)

MOVFP DPX+B1, WREG ; get 2nd byte of DPX into w
ADDWF ACC+B1 ; add 2nd byte of ACC, save in

MOVFP DPX+B2, WREG ; get 3rd byte of DPX into w
ADDWF ACC+B2 ; add 3rd byte of ACC, save in

MOVFP DPX+B3, WREG ; get 4th byte of DPX into w
ADDWF ACC+B3 ; add 4th byte of ACC, save in

; Compute A1*D(n-1)

MOVFP Dn1_1+B0, AARG+B0 ; move Dn1_1(B0) to AARG(B0)
MOVFP Dn1_1+B1, AARG+B1 ; move Dn1_1(B1) to AARG(B1)

MOVFP All+B0, BARG+B0 ; move All(B0) to BARG(B0)
MOVFP All+B1, BARG+B1 ; move All(B1) to BARG(B1)

call DbLMult ; (ACCd, ACCc) = A1*D(n-1)

; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
; multiplications already done, so simply perform a 32 bit add
; of previously obtained multiplication results

MOVFP DPX+B0, WREG ; get lowest byte of DPX into w
ADDWF ACC+B0 ; add lowest byte of ACC, save in

MOVFP DPX+B1, WREG ; get 2nd byte of DPX into w
ADDWF ACC+B1 ; add 2nd byte of ACC, save in

MOVFP DPX+B2, WREG ; get 3rd byte of DPX into w
ADDWF ACC+B2 ; add 3rd byte of ACC, save in

MOVFP DPX+B3, WREG ; get 4th byte of DPX into w
ADDWF ACC+B3 ; add 4th byte of ACC, save in

; save the upper 16 bits of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication

rlcf ACC+B1, w
rlcf ACC+B2, w
movwf Dn1
rlcf ACC+B3, w ; decimal adjust (mult by 2)

movwf Dn1+B1

; Compute B2 * [D(n) + D(n-2)]

movfp Dn1_2+B0, wreg
addwf Dn1+B0, w
movwf AARG+B0
movfp Dn1_2+B1, wreg
addwfc Dn1+B1, w
movwf AARG+B1

; if B0_EQUALS_B2
```
Implementing IIR Digital Filters

MOVFP B10+B0, BARG+B0 ; move B10(B0) to BARG(B0)
MOVFP B10+B1, BARG+B1 ; move B10(B1) to BARG(B1)
call DblMult ; (ACCd, ACCc) = B2*[D(n)+D(n-2)]

MOVFP DPX+B0, ACC+B0 ; move DPX(B0) to ACC(B0)
MOVFP DPX+B1, ACC+B1 ; move DPX(B1) to ACC(B1)
MOVFP DPX+B2, ACC+B2 ; move DPX(B2) to ACC(B2)
MOVFP DPX+B3, ACC+B3 ; move DPX(B3) to ACC(B3)

else

MOVFP16 B10, BARG
MOVFP16 Dn1, AARG
call DblMult ; B0*D(n)
MOVFP32 DPX, ACC

MOVFP16 Bx2, BARG
MOVFP16 Dn1_2, AARG
call DblMult ; B2*D(n-2)
ADD32 DPX, ACC

endif

; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
; This way in the next iteration D(n-2) is equal to the present D(n-1)

movfp Dn1_1, AARG+B0
movfp AARG+B0, Dn1_2 ; Shift down D(n-1)
movfp Dn1_1+B1, AARG+B1
movfp AARG+B1, Dn1_2+B1 ; AARG - D(n-1) - multiplier

MOVFP B11+B0, BARG+B0 ; move B11(B0) to BARG(B0)
MOVFP B11+B1, BARG+B1 ; move B11(B1) to BARG(B1)
call DblMult ; (ACCd, ACCc) = B1*D(n-1)

; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
; = B1*D(n-1) + B0*[D(n) + D(n-2)]
; Since all multiplications are already done, simply perform a
; 32 bit addition

MOVFP DPX+B0, WREG ; get lowest byte of DPX into w
ADDWF ACC+B0 ; add lowest byte of ACC, save in
ACC(BO)
MOVFP DPX+B1, WREG ; get 2nd byte of DPX into w
ADDWFACC+B1 ; add 2nd byte of ACC, save in
Implementing IIR Digital Filters

ACC(B1)
0063 601A
0064 1146
ACC(B2)
0065 601B
0066 1147
ACC(B3)

MOVFP DFX+B2,WREG ; get 3rd byte of DFX into w
ADDWFC ACC+B2 ; add 3rd byte of ACC, save in

; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;
0067 6020
0068 0122
0069 6021
006A 0123

MOVFP Dnl+B0,WREG ; get byte of Dn1 into w
MOVWF Dnl_1+B0 ; move to Dn1(B0)

MOVFP Dnl+B1,WREG ; get byte of Dn1(B1)
MOVWF Dnl_1+B1 ; move to Dn1(B1)

; 2nd Biquad filter section
;
BIQUAD A21,A22,B20,B21,Dn2,Dn2 2,2 ;
;
; Compute A22*D(n-2)
;
006B 7C2A
006C 7D2B

MOVFP Dn2 2+B0,AARG+B0 ; move Dn2(B0) to AARG(B0)
MOVFP Dn2 2+B1,AARG+B1 ; move Dn2(B1) to AARG(B1)

006D 7E38
006E 7F39

MOVFP A22+B0,BARG+B0 ; move A22(B0) to BARG(B0)
MOVFP A22+B1,BARG+B1 ; move A22(B1) to BARG(B1)

006F E0AF

call DblMult ; (ACCd,ACCc) = A2*D(n-2)
;
; Add product to output of 1st section
; Save result in 32 bit Accumulator
;
0070 6018
0071 0F44
0072 6019
0073 1145

MOVFP DFX+B0,WREG ; get lowest byte of DFX into w
ADDWF ACC+B0 ; add lowest byte of ACC, save in

MOVFP DFX+B1,WREG ; get 2nd byte of DFX into w
ADDWFC ACC+B1 ; add 2nd byte of ACC, save in

MOVFP DFX+B2,WREG ; get 3rd byte of DFX into w
ADDWFC ACC+B2 ; add 3rd byte of ACC, save in

MOVFP DFX+B3,WREG ; get 4th byte of DFX into w
ADDWFC ACC+B3 ; add 4th byte of ACC, save in

; Compute A1*D(n-1)
;
0078 7C28
0079 7D29

MOVFP Dn2 1+B0,AARG+B0 ; move Dn2(B0) to AARG(B0)
MOVFP Dn2 1+B1,AARG+B1 ; move Dn2(B1) to AARG(B1)
Implementing IIR Digital Filters

007A 7E36
MPVF A21+B0,BARG+B0 ; move A21(B0) to BARG(B0)
007B 7F37
MPVF A21+B1,BARG+B1 ; move A21(B1) to BARG(B1)

007C E0AF
call DblMult ; (ACCd,ACCc) = A1*D(n-1)

; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
; multiplications already done, so simply perform a 32 bit add
; of previously obtained multiplication results

007D 6018
MPVF D PXB0,WREG ; get lowest byte of D PX into w
007E 0F44
ADDWF ACC+B0 ; add lowest byte of ACC, save in
ACC(B0)

007F 6019
MPVF D PXB1,WREG ; get 2nd byte of D PX into w
0080 1145
ADDWF ACC+B1 ; add 2nd byte of ACC, save in
ACC(B1)

0081 601A
MPVF D PXB2,WREG ; get 3rd byte of D PX into w
0082 1146
ADDWF ACC+B2 ; add 3rd byte of ACC, save in
ACC(B2)

0083 601B
MPVF D PXB3,WREG ; get 4th byte of D PX into w
0084 1147
ADDWF ACC+B3 ; add 4th byte of ACC, save in
ACC(B3)

; save the upper 16 bits of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication

0085 1A45
rlcf ACC+B1,w
0086 1A46
rlcf ACC+B2,w
0087 0126
movwf Dn2
0088 1A47
rlcf ACC+B3,w ; decimal adjust (mult by 2)
0089 0127
movwf Dn2+B1 ;

; Compute B2 * [D(n) + D(n-2)]

if B0_EQUALS_B2

008A 602A
movfp Dn2_2+B0,wreg
008B 0E26
addwf Dn2+B0,w
008C 011C
movwf AARG+B0
008D 602B
movfp Dn2_2+B1,wreg
008E 1027
addwfc Dn2+B1,w
008F 011D
movwf AARG+B1

0090 7E3A
MOVF B20+B0,BARG+B0 ; move B20(B0) to BARG(B0)
0091 7F3B
MOVF B20+B1,BARG+B1 ; move B20(B1) to BARG(B1)

0092 E0AF
call DblMult ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]

0093 5844
MOVF D PXB0,ACC+B0 ; move D PX(B0) to ACC(B0)
0094 5945
MOVF D PXB1,ACC+B1 ; move D PX(B1) to ACC(B1)
0095 5A46
MOVF D PXB2,ACC+B2 ; move D PX(B2) to ACC(B2)
0096 5B47
MOVF D PXB3,ACC+B3 ; move D PX(B3) to ACC(B3)

else

MOVFP16 B20,BARG
Implementing IIR Digital Filters

MOVFP16 Dn2,AARG
    call DblMult ; B0*D(n)
MOVFP32 DPX,ACC

MOVFP16 Bx2,BARG
MOVFP16 Dn2_2,AARG
    call DblMult ; B2*D(n-2)
ADD32 DPX,ACC

    endif
    ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
    ; This way in the next iteration D(n-2) is equal to the present D(n-1)
    ;
    0097 7C28 movfp Dn2_1,AARG+B0
    0098 5C2A movpf AARG+B0,Dn2_2 ; Shift down D(n-1)
    0099 7D29 movfp Dn2_1+Bl,AARG+B1
    009A 5D2B movpf AARG+B1,Dn2_2+B1 ; AARG = D(n-1) - multiplier

009B 7E3C MOVFP B21+B0,BARG+B0 ; move B21(B0) to BARG(B0)
009C 7F3D MOVFP B21+B1,BARG+B1 ; move B21(B1) to BARG(B1)

009D E0AF call DblMult ; (ACCd, ACCc) = B1*D(n-1)
    ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
    ; = B1*D(n-1) + B0*[D(n) + D(n-2)]
    ; Since all multiplications are already done, simply perform a
    ; 32 bit addition

009E 6018 MOVFP DPX+B0,WREG ; get lowest byte of DPX into w
009F 0F44 ADDWF ACC+B0 ; add lowest byte of ACC, save in
    ACC(B0)
00A0 6019 MOVFP DPX+B1,WREG ; get 2nd byte of DPX into w
00A1 1145 ADDWFC ACC+B1 ; add 2nd byte of ACC, save in
    ACC(B1)
00A2 601A MOVFP DPX+B2,WREG ; get 3rd byte of DPX into w
00A3 1146 ADDWFC ACC+B2 ; add 3rd byte of ACC, save in
    ACC(B2)
00A4 601B MOVFP DPX+B3,WREG ; get 4th byte of DPX into w
00A5 1147 ADDWFC ACC+B3 ; add 4th byte of ACC, save in
    ACC(B3)

    ; Shift down D(n) to D(n-1) so that in the next iteration, the new
    ; D(n-1) is the present D(n)
    ;
    00A6 6026 MOVFP Dn2+B0,WREG ; get byte of Dn2 into w
00A7 0128 MOVWF Dn2_1+B0 ; move to Dn2_1(B0)
00A8 6027 MOVFP Dn2+B1,WREG ; get byte of Dn2 into w
00A9 0129 MOVWF Dn2_1+B1 ; move to Dn2_1(B1)

© 1993 Microchip Technology Inc.

DS00540B-page 14 4-134
Implementing IIR Digital Filters

; The filter output is now computed
; Save the Upper 16 Bits of 32 bit Accumulator into Y after
; adjusting the decimal point
;
MOV16   ACC+B2, Y
00AA  6046
00AB  0142
00AC  6047
00AD  0143

; return           ; Output Y(n) computed
;
;******************************************************************************
; Set SIGNED/UNSIGNED Flag Before Including 17c42MPY.mac
; include "17c42MPY.mac"
;******************************************************************************

;******************************************************************************
; Low Pass Filter Co-efficients
;
;******************************************************************************

ELLIPIC   LOWPASS FILTER
;
FILTER ORDER =  4
Sampling frequency =  2.000 KiloHertz
;
BAND 1  BAND 2
;
;
LOWER BAND EDGE =  .00000  .60000
UPPER BAND EDGE =  .50000  1.00000
NOMINAL GAIN =  1.00000  0.00000
NOMINAL RIPPLE =  .01000  .05000
MAXIMUM RIPPLE =  .00906  .04601
RIPPLE IN dB =  0.07830  -26.74235
;
I     A(I,1)     A(I,2)     B(I,0)     B(I,1)     B(I,2)
;
1    -.333331    .167145    .285431    .462321    .285431
2    .147827     .765900    .698273    .499908    .698273

_coeffs_lpass  ; co-efficients for 1st Cascade
01C1  1111     data  4369  ; -A11
01C2  EA9B     data  -5477  ; -A12
01C3  2489     data  9353  ; B10
01C4  3B41     data  15169  ; B11
01C5  2489     data  9353  ; B12
;
_coeffs_lpass  ; co-efficients for 2nd Cascade
01C6  ED14     data  -4844  ; -A21
01C7  9DF7     data  -25097  ; -A22
01C8  5961     data  22881  ; B20
01C9  3FFD     data  16381  ; B21
01CA  5961     data  22881  ; B22
;
******************************************************************************
;
******************************************************************************

; High Pass Filter Co-efficients
;

Implementing IIR Digital Filters

ELLIPTIC HIGHPASS FILTER

FILTER ORDER = 4
Sampling frequency = 2.000 KiloHertz

<table>
<thead>
<tr>
<th>BAND 1</th>
<th>BAND 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOWER BAND EDGE</td>
<td>.00000</td>
</tr>
<tr>
<td>UPPER BAND EDGE</td>
<td>.40000</td>
</tr>
<tr>
<td>NOMINAL GAIN</td>
<td>.00000</td>
</tr>
<tr>
<td>NOMINAL RIPPLE</td>
<td>.04000</td>
</tr>
<tr>
<td>MAXIMUM RIPPLE</td>
<td>.03368</td>
</tr>
<tr>
<td>RIPPLE IN dB</td>
<td>-29.45355</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I</th>
<th>A(I,1)</th>
<th>A(I,2)</th>
<th>B(I,0)</th>
<th>B(I,1)</th>
<th>B(I,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.276886</td>
<td>.195648</td>
<td>.253677</td>
<td>-.411407</td>
<td>.253677</td>
</tr>
<tr>
<td>2</td>
<td>-.094299</td>
<td>.780396</td>
<td>.678650</td>
<td>-.485840</td>
<td>.678650</td>
</tr>
</tbody>
</table>

_coeff_hpass

co-efficients for 1st Cascade section

| 01CB DC8F | data | -9073 |
| 01CC E6F5 | data | -6411 |
| 01CD 2079 | data | 8313 |
| 01CE CB57 | data | -13481 |
| 01CF 2079 | data | 8313 |

co-efficients for 2nd Cascade section

| 01D0 0C12 | data | 3090 |
| 01D1 9C1C | data | -25572 |
| 01D2 56DE | data | 22238 |
| 01D3 C1D0 | data | -15920 |
| 01D4 56DE | data | 22238 |

*******************************************************************
*******************************************************************

Band Pass Filter Co-efficients

ELLIPTIC BANDPASS FILTER

FILTER ORDER = 4
Sampling frequency = 2.000 KiloHertz

<table>
<thead>
<tr>
<th>BAND 1</th>
<th>BAND 2</th>
<th>BAND 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOWER BAND EDGE</td>
<td>.00000</td>
<td>.30000</td>
</tr>
<tr>
<td>UPPER BAND EDGE</td>
<td>.10000</td>
<td>.70000</td>
</tr>
<tr>
<td>NOMINAL GAIN</td>
<td>.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>NOMINAL RIPPLE</td>
<td>.05000</td>
<td>.05000</td>
</tr>
<tr>
<td>MAXIMUM RIPPLE</td>
<td>.03644</td>
<td>.03867</td>
</tr>
<tr>
<td>RIPPLE IN dB</td>
<td>-28.76779</td>
<td>.32956</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I</th>
<th>A(I,1)</th>
<th>A(I,2)</th>
<th>B(I,0)</th>
<th>B(I,1)</th>
<th>B(I,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-.936676</td>
<td>.550568</td>
<td>.444000</td>
<td>-.865173</td>
<td>.444000</td>
</tr>
<tr>
<td>2</td>
<td>.936707</td>
<td>.550568</td>
<td>.615540</td>
<td>1.199402</td>
<td>.615540</td>
</tr>
</tbody>
</table>

_coeff_bpass

co-efficients for 1st Cascade section

| 01D5 3BF2 | data | 30693/2 |
| 01D6 DCC4 | data | -18041/2 |
| 01D7 1C6A | data | 14549/2 |
| 01D8 C8A1 | data | -28350/2 |
| 01D9 1C6A | data | 14549/2 |

co-efficients for 2nd Cascade section
Implementing IIR Digital Filters

Data -30694/2 ; -A21
Data -18041/2 ; -A22
Data 20170/2 ; B20
Data 39302/2 ; B21
Data 20170/2 ; B22

;*******************************************************************
;*******************************************************************
; Band Stop Filter Co-efficients
;
;
; ELLIPTIC BANDSTOP FILTER
;
; FILTER ORDER = 4
; Sampling frequency = 2.000 KiloHertz
;
; LOWER BAND EDGE
; UPPER BAND EDGE
; NOMINAL GAIN
; NOMINAL RIPPLE
; MAXIMUM RIPPLE
; RIPPLE IN dB
; sampling frequency
; Band
; 0.00000
; 0.30000
; 1.00000
; 0.05000
; 0.03516
; 0.30015
; -29.78523
; 0.05000
; 0.03517
; 0.30027
; 0.05000
; 0.03241
; 0.30015
; -29.78523
; 0.05000
; 0.03517
; 0.30027
; 0.05000
; 0.03241
; 0.30015
; -29.78523
; 0.05000
; 0.03517
; 0.30027
; 0.05000
; 0.03241
; 0.30015
; -29.78523
;
; A(I,1) A(I,2) B(I,0) B(I,1) B(I,2)
;
; 1 .749420 .583282 .392685 .087936 .392685
; 2 -.749390 .583282 1.210022 -.270935 1.210022
;
; _coeff_bstop ; co-efficients for 1st Cascade section
01DF D00A data -24557/2 ; -A11
01E0 DAAC data -19113/2 ; -A12
01E1 1922 data 12868/2 ; B10
01E2 05A0 data 2881/2 ; B11
01E3 1922 data 12868/2 ; B12
;
01E4 2FF6 data 24556/2 ; -A21
01E5 DAAC data -19113/2 ; -A22
01E6 4D71 data 39650/2 ; B20
01E7 EE9 data -8878/2 ; B21
01E8 4D71 data 39650/2 ; B22
;
END

Errors : 0
Warnings : 0

© 1993 Microchip Technology Inc.
INTRODUCTION

Fourier transforms are one of the fundamental operations in signal processing. In digital computations, Discrete Fourier Transforms (DFT) are used to describe, represent and analyze discrete-time signals. However, direct implementation of DFT is computationally very inefficient. Of the various available high speed algorithms to compute DFT, the Cooley-Tukey algorithm is the simplest and most commonly used. These efficient algorithms to compute DFTs are called Fast Fourier Transforms (FFTs).

This application note provides the source code to compute the FFT using PIC17C42. The theory behind the FFT algorithms is well established and described in the literature and hence not described in this application note. A Radix-2 Cooley-Tukey FFT is implemented with no limits on the length of FFT. The length is only limited by the amount of available program memory space. All computations are done using double precision arithmetic.

IMPLEMENTATION

Since the PIC17C42 has only 232 x 8 general purpose RAM (equivalent of 116 x 16), at most a 32 point FFT (16-bit REAL & IMAGINARY data) can be implemented using on chip RAM. To compute higher point FFT, the data is stored in program memory space of PIC17C42. The PIC17C42 has instructions (TABLRD & TABLWR) to transfer data between program memory space and on chip file registers. In extended microcontroller mode, the PIC17C42 has 2K x 16 (0000h:07FFh) on chip program memory space and 62K x 16 (0800h:0FFFFh) of external program memory space. In this mode, the code (in this case, the FFT code) may reside on the on chip EPROM and the data to be analyzed may be stored in external RAMs (up to 62K). A suggested method of connecting external RAMs (appropriate EEPROMs may also be used) is shown in Figure 3.

If PIC17C42 is used in extended microcontroller mode and if all the code resides on chip, then the cost may further be reduced by using only one external SRAM instead of two. The block diagram is shown in Figure 2. The 16-bit data stored in the external RAM is organized as low byte followed by high byte. To achieve this, the code presented in this application note needs minor modifications, especially where TABLRD and TABLWR instructions are used. Address indexing must be incremented by two since 2 reads/writes must be performed to access a 16-bit data.

The FFT is implemented with Decimation In Frequency. Thus the input data before calling the FFT routine ("R2FFT") should be in normal order and the transformed data is in scrambled order. The original data is overwritten by the transformed data to conserve memory. This is achieved by use of in-place calculations. This in-place calculations causes the order of the DFT terms to be permuted. So at the end of the transform, all the data needs to be unscrambled to get the right order of the DFT terms. In some applications the order of terms is not necessary. Keeping this in mind, the unscrambling code is written as a separate subroutine ("Unscramble") and may be called if necessary.

Before implementing the FFT using PIC17C42, a "C" program was written and tested. This high level programming helps in writing the assembly code and the results of both programs can be compared against while debugging the assembly code. The "C" source code for the Radix-2 FFT is shown in Appendix A. The assembly code source file of the FFT program is shown in Appendix B. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note ANw00540.

FIGURE 1 - TEST WAVE FORM

Input Square Wave

<table>
<thead>
<tr>
<th>Sample Number</th>
<th>0</th>
<th>32</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>160</th>
<th>192</th>
<th>224</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplitude</td>
<td>0</td>
<td>4000</td>
<td>8000</td>
<td>12000</td>
<td>16000</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
Implementing FFT

TESTING

The assembly code was developed and debugged using Microchip’s PICMASTER In-Circuit Emulator System. A main program generates a test pattern (like a square wave) and calls the FFT routines “R2FFT” & “Unscramble”. After the DFT terms are computed, the results are captured into PICMASTER’s real time trace buffer by putting a trace point on a dummy TABLRD instruction and capturing only the 2nd cycle (data cycle of TABLRD) of the instruction. The data from Trace buffer was hot linked to Microsoft Excel using DDE and then the graphs were plotted and analyzed.

The code was tested on various wave forms (a rectangular pulse, a triangular wave, square wave and a sine wave) and using FFT lengths of 64, 256 & 1024. The results of a 256 Point FFT on a square wave is shown below. The test wave form is shown in Figure 1 and the frequency spectrum of it computed by PIC17C42 is shown in Figure 2. As expected, the spectra appears at the odd harmonics of the input wave form’s fundamental frequency (At N*256/64, N = 0,1,3,5,..).

PERFORMANCE

The performance of FFT using PIC17C42 is quite impressive noting that for an 8 bit machine with no hardware multiplier. Also note that all computations are performed using double precision arithmetic (16- & 32- bits) which is the case in most of the low end DSPs. Table 1 provides the real time performance in total number of Instruction cycles for both the “R2FFT” & “Unscramble” routines using 64, 256 & 1024 Point FFT. Note that the timings are in a worst case situation and in general will be a lot better than shown in the table. The worst case situation arises because the 16 x 16 software multiplier ("DblMult") does not have a uniform timing and depends on the input data. The worst case timing of the multiplier is used in the computation of performance.

FIGURE 2 - FFT (MAGNITUDE SPECTRUM) OF FIGURE 1 COMPUTED BY PIC17C42

TABLE 1 - WORST CASE PERFORMANCE OF FFT IN INSTRUCTION CYCLES AND REAL TIME @ 16 MHZ

<table>
<thead>
<tr>
<th>N (FFT Length)</th>
<th>64 Point</th>
<th>256 Point</th>
<th>1024 Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2FFT</td>
<td>34116 + 768*Mult = 171588</td>
<td>178024 + 4096*Mult= 911208</td>
<td>878752 + 20480*Mult = 4544672</td>
</tr>
<tr>
<td>Unscramble</td>
<td>5143</td>
<td>22495</td>
<td>93525</td>
</tr>
<tr>
<td>Total</td>
<td>176731 (44.18 mS)</td>
<td>933703 (233.42 mS)</td>
<td>4638197 (1159.55 mS)</td>
</tr>
</tbody>
</table>
Table 2 shows the Program Memory & Data RAM requirements for an N Point FFT. The multiplier routine and other general purpose macro requirements are included in the memory requirements. The speed performance for the square wave test data differs from the Table 1 since worst case timings is not used and reflects a more reasonable data.

**FFT APPLICATIONS**

Although the FFT does not find a place in many microcontroller applications, it is very useful in providing a benchmark of the processor. As can be seen from Table 2, the performance is very satisfactory, considering the fact that PIC17C42 is a Microcontroller and not a DSP. Also it should be borne in mind that all computations are performed in 16/32 bit arithmetic and that PIC17C42 is a low cost 8 bit machine unlike the DSPs which are relatively expensive.

In applications like Instrumentation, where real time FFT computation is not required, PIC17C42 can be used as a single chip solution instead of a Microcontroller and a Digital Signal Processor.

**Suggested Reading**:


**TABLE 2 - REQUIREMENTS FOR RADIX-2 FFT**

<table>
<thead>
<tr>
<th>N (FFT Length)</th>
<th>64 Point</th>
<th>256 Point</th>
<th>1024 Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Space (locations)</td>
<td>603 + 0.75*N = 651</td>
<td>603 + 0.75*N = 795</td>
<td>603 + 0.75*N = 1371</td>
</tr>
<tr>
<td>Data Storage in Program Memory Space</td>
<td>2*N = 128</td>
<td>2*N = 512</td>
<td>2*N = 2048</td>
</tr>
<tr>
<td>Scratch RAM</td>
<td>49</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>Performance (Square Wave Input Data)</td>
<td>122384 (30.6 mS)</td>
<td>644416 (161.1 mS)</td>
<td>3192176 (798 mS)</td>
</tr>
</tbody>
</table>

**FIGURE 3 - EXTERNAL MEMORY CONNECTION**

![External Memory Connection Diagram](image1)

**FIGURE 4 - ALTERNATE EXTERNAL MEMORY CONNECTION**

![Alternate External Memory Connection Diagram](image2)
Implementing FFT

APPENDIX A: FFT ALGORITHM

A Cooley-Tukey Radix-2 DIF FFT

Radix-2 implementation
Decimation In Frequency
Single Butterfly
Table Lookup of Twiddle Factors
Complex Input & Complex Output

All data is assumed to be 16 bits and the intermediate results are stored in 32 bits

Length Of FFT must be a Power Of 2
Max Length Possible is 2**15

The input/output complex data is organized as a single array of real data followed by imaginary data
Data is stored in External Memory and is accessed by TABLRD & TABLWT Instructions

LIST P=17C42, C=120, T=ON, R=DEC, N=0
include "17c42.h"

include "17c42.mac"

-----------------------------------------------------------
RLC16AB

DESCRIPTION:
16 bit rotate left A into B

ARGUMENTS:
2*a => b

TIMING (in cycles):
3

------------------------------

RLC16AB MACRO a,b
BCF _carry
RLCF a=B0,W
MOVWF b=B0
RLCF a=B1,W
MOVWF b=B1
ENDM

-----------------------------------------------------------

TBLADDR

DESCRIPTION:
Load 16 bit table pointer with specified label
Implementing FFT

; TIMING (in cycles):
; 4
;
TBLADDR MACRO label
MOVLW (label) & 0xff
MOVWF tblptrl
MOVLW page (label)
MOVWF tblptrh
ENDM

;***************************************************************************
; ADDLBL
;
; DESCRIPTION:
; Add A Label (16 bit constant) To A File Register (16 bit)
;
; TIMING (in cycles):
; 4
;
;***************************************************************************
ADDLBL MACRO label,f
MOVLW (label) & 0xff
ADDWF f+B0
MOVLW page (label)
ADDWF f+B1
ENDM

;***************************************************************************
FftLen .set 256 FFT Length
Power .set 8 (2**Power = FftLen)
DigitRevCount .set 239 (FftLen-1) - (2**(Power+1)/2))
SCALE_BUTTERFLY .set TRUE intermediate scaling performed
EXT_RAM_START_ADDR .set 0x0800 External Memory Data Storage Start

;***************************************************************************
CBLOCK 0

0000 0004 B0,B1,B2,B3 ; RAM offset constants
ENDC

CBLOCK 0x18

0018 0002 AARG,AARG1 ; 16 bit multiplier A
001A 0002 BARG,BARG1 ; 16 bit multiplicand B
001C 0004 DPX,DPX1,DPX2,DPX3 ; 32 bit multiplier result = A*B
ENDC

CBLOCK

0020 0004 ACC, ACC1, ACC2, ACC3 ; 32 bit accumulator for computa
ENDC

CBLOCK

0024 0002 countl,countll ; N1
0026 0002 count2,count22 ; N2
0028 0002 QuartLen,QuartLenl ; FftLen/4
ENDC

CBLOCK

002A 0002 TF_Offset,TF_Offsetl ; twiddle factor address computa
002C 0002 TF_Addr,TF_Addrl
Implementing FFT

002E 0002  Cos, Cos1,
0030 0002  Sin, Sin1
ENDC

; CBLOCK
0032 0002  VarJloop, VarJloop1
0034 0002  VarJloop, VarJloop1
0036 0001  VarKloop
0037 0002  VarL, VarL1
ENDC

; CBLOCK
0039 0002  XI, X11
003B 0002  Yi, Yi1
003D 0002  XI, X11
003F 0002  Yi, Yi1
ENDC

; CBLOCK
0041 0002  Xt, X1
0043 0002  Yt, Yt1
ENDC

0045 0004  temp, temp1, temp2, temp3
0049 0002  testCount, testCount1
004B 0002  PulseCount, PulseCount1
ENDC

;******************************************************************
; Test Program For FFT Subroutine
;******************************************************************
ORG Ox0000

include "square.asm" ; Generate Test Vector Data

;*******************************************************************
; Test Routine For FFT
;*******************************************************************

 PulswidthFactor .set 8

; testFf
MOVK16 2*PulseWidthFactor, testCount

0000 B010  MOV.L  (2*PulseWidthFactor) & 0xff
0001 0149  MOVWF  testCount+B0
0002 B000  MOV.L  (2*PulseWidthFactor)/256
0003 014A  MOVWF  testCount+B1

CLR16 Yi

0004 293B  CLR.F Yi+B0
0005 293C  CLR.F Yi+B1

TBLADDR ExtRamAddr ; load table pointers with data start addr

0006 B000  MOV.L  (ExtRamAddr) & 0xff
0007 010D  MOVWF  tblptr1
0008 B008  MOV.L  page  (ExtRamAddr)
0009 010E  MOVWF  tblptrh

; nextPulse
Implementing FFT

MOVK FftLen/PulseWidthFactor, PulseCount

000A B020 MOVLW FftLen/PulseWidthFactor
000B 014B MOVWF PulseCount

MOVK FftLen/PulseWidthFactor, PulseCount1

000C B020 MOVLW FftLen/PulseWidthFactor
000D 014C MOVWF PulseCount1

; MOVK16 0x3FFF, Xi

000E B0FF MOVLW (0x3FFF) & 0xff
000F 0139 MOVWF Xi+BO
0010 B03F MOVLW (0x3FFF)/256
0011 013A MOVWF Xi+Bl

LX1
0012 E034 call write
0013 174B decfsz PulseCount
0014 C012 goto LX1

; CLR16 Xi
0015 2939 CLRF Xi+BO
0016 293A CLRF Xi+Bl

LX2
0017 E034 call write
0018 174C decfsz PulseCount1
0019 C017 goto LX2

; DEC16 testCount
001A 2900 CLRF WREG
001B 0749 DECF testCount+BO
001C 034A SUBWF testCount+Bl

TFSZ16 testCount
001D 6049 MOVFP testCount+BO, WREG
001E 084A IORWF testCount+Bl, W
001F 3300 TSTFSZ WREG

0020 C00A goto nextPulse

; MOVLW (FftLen*2) & 0xff
0021 E039 call R2FFT ; Compute Fourier Transform
0022 E116 call Unscramble ; Digit Reverse the scrambled data

; Fourier Transform Completed
;
; capture data to PIC-MASTER Trace Buffer
MOVK16 FftLen*2, temp

0023 B000 MOVLW (FftLen*2) & 0xff
0024 0145 MOVWF temp+BO
0025 B002 MOVLW (FftLen*2)/256
0026 0146 MOVWF temp+Bl
Implementing FFT

TBLADDR ExtRamAddr ; load table pointers with data start addr
0027 B000 MOV LW (ExtRamAddr) & 0xff
0028 010D MOVWF tblptrl
0029 B008 MOV LW page (ExtRamAddr)
002A 010E MOVWF tblptrh

capture
cap 0,1,Sim ; table latch = mem(tblptr)

DEC16 temp
002B A930 CLRF WREG
002C 2900 DECF temp+B0
002D 0745 SUBWF temp+B1

TFSZ16 temp
002E 0346 MOVFP temp+B0,WREG
002F 6045 IORWF temp+B1,W
0030 3300 TSTFSZ WREG

0032 C02B goto capture
0033 C033 self goto self

write
tlw 0,Xi
0034 A439 tablw 1,1,Xi+B1 ; auto increment for Imag Data
tlw 0,Yi
0035 AF3A tablw 1,1,Yi+B1
0036 A43B tlwt 0,Xi
0037 AF3C tlwt 1,1,Yi+B1
0038 0002 return

;*******************************************************************************
;
RADIX-2 FFT
;
; Decimation In Frequency
;
; Input Data should be unscrambled
; Output Data at the end is in scrambled form
; To obtain the unscrambled form, the digit reverse counter
; subroutine, "Unscramble" should be called (see the example)
;
;*******************************************************************************

R2FFT
MOVK16 FftLen,count2 ; count2 = N

0039 B000 MOV LW (FftLen)/4 & 0xff
003A 0126 MOVWF count2+B0
003B B001 MOV LW (FftLen)/256
003C 0127 MOVWF count2+B1

MOVK16 FftLen/4,QuartLen ; QuartLen = FftLen/4

003D B040 MOV LW (FftLen/4)/4 & 0xff
003E 0128 MOVWF QuartLen+B0
003F B000 MOV LW (FftLen/4)/256
0040 0129 MOVWF QuartLen+B1

0041 292B MOVK 1,TF_Offset+Bl ; Init TF_Offset = 1

0042 B001 MOV LW 1
0043 012A MOVWF TF_Offset

© 1993 Microchip Technology Inc.
Implementing FFT

MOVK Power,VarKloop

0044 B008    MOVLW Power
0045 0136    MOVWF VarKloop

; Kloop

Kloop

MOVLW count2,count1

0046 6026    MOVFP count2+BO,WREG
0047 0124    MOVWF count1+BO
0048 6027    MOVFP count2+Bl,WREG
0049 0125    MOVWF count1+Bl

; for K = 1 to Power-1
; count1 = count2

RRC16 count2

004A 1A27    RLCF count2+B1,W
004B 1927    RRCF count2+B1
004C 1926    RRCF count2+B0

; count2 = count2/2

CLR16 VarJloop

004D 2934    CLRJ VarJloop+B0
004E 2935    CLRJ VarJloop+B1

; J = 0

CLR16 TF.Addr

004F 292C    CLRJ TF.Addr+B0
0050 292D    CLRJ TF.Addr+B1

; TF.Addr = 0

Jloop

; Read Twiddle factors from Sine/Cosine Table from Prog Mem

MOVF16 TF.Addr,tblptrl

; load sine table address to table

0051 6D2C    MOVFP TF.Addr+B0,tblptrl+B0
0052 6E2D    MOVFP TF.Addr+B1,tblptrl+B1

ADDLBL SineTable,tblptrl

; add table offset

0053 B094    MOVLW (SineTable) & Oxff
0054 0F0D    ADDWF tblptrl+B0
0055 B002    MOVLW page(SineTable)
0056 110E    ADDWFC tblptrl+B1

0057 A830    tablrd 0,0,Sin
0058 A030    tlrld 0,Sin
0059 A231    tlrld 1,Sin+B1

ADD16 QuartLen,tblptrl

005A 6028    MOVFP QuartLen+B0,WREG
005B 0E0D    ADDWF tblptrl+B0
005C 6029    MOVFP QuartLen+B1,WREG
005D 110E    ADDWFC tblptrl+B1

005E A82E    tablrd 0,0,Cos
005F A02E    tlrld 0,Cos
0060 A22F    tlrld 1,Cos+B1

; Read Sine Value from lookup table

; Read Cosine Value from table

© 1993 Microchip Technology Inc.
Implementing FFT

```
ADD16  TF_Offset,TF_Addr ; TF_Addr = TF_Addr + TF_Offset
0061 602A MOVFP TF_Offset+B0,WREG ; get lowest byte of TF_Offset into w
0062 0F2C ADDWF TF_Addr+B0 ; add lowest byte of TF_Addr, save in
0063 602B MOVFP TF_Offset+B1,WREG ; get 2nd byte of TF_Offset into w
0064 112D ADDWFC TF_Addr+B1 ; add 2nd byte of TF_Addr, save in

; RLC16AB VarJloop,VarIloop ; I = J*2 since Real followed by Imag
0065 8804 BCF carry
0066 1A34 RLCF VarJloop+B0,W
0067 0132 MOVWF VarIloop+B0
0068 1A35 RLCF VarJloop+B1,W
0069 0133 MOVWF VarIloop+B1

; Iloop
RLC16AB count2,VarL ; VarL = count2*2
006A 8804 BCF carry
006B 1A26 RLCF count2+B0,W
006C 0137 MOVWF VarL+B0
006D 1A27 RLCF count2+B1,W
006E 0138 MOVWF VarL+B1

ADD16 VarIloop,VarL ; VarL = (I+count2)*2
006F 602C MOVFP VarIloop+B0,WREG ; get lowest byte of VarIloop into w
0070 0F37 ADDWF VarL+B0 ; add lowest byte of VarL, save in
0071 6033 MOVFP VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
0072 113E ADDWFC VarL+B1 ; add 2nd byte of VarL, save in VarL(B1)

; Get Real & Imag Data from external RAMs (Program Memory)
; load table pointers with data start addr
;
MOVFP16 VarL,tblptrl ; read data(L)
0073 6037 MOVFP VarL+B0,tblptrl+B0 ; move VarL(B0) to tblptrl(B0)
0074 6E38 MOVFP VarL+B1,tblptrl+B1 ; move VarL(B1) to tblptrl(B1)

ADDLBL ExtRamAddr,tblptrl ; add data addr offset
0075 B000 MOVLW (ExtRamAddr) & Oxff
0076 0F0D ADDWF tblptrl+B0
0077 B008 MOVLW page (ExtRamAddr)
0078 110E ADDWFC tblptrl+B1

0079 A93D tabldr 0,1,Xl ; auto increment for Imag Data
007A A93D tldr 0,Xl
007B A23E tldr 1,Xl+B1 ; real data XL
007C A83F tabldr 0,0,Yl
007D A03F tldr 0,Yl
007E A440 tldr 1,Yl+B1 ; imag data YL
MOVFP16 VarIloop,tblptrl ; read data(I)
007F 6032 MOVFP VarIloop+B0,tblptrl+B0 ; move VarIloop(B0) to tblptrl(B0)
0080 6E33 MOVFP VarIloop+B1,tblptrl+B1 ; move VarIloop(B1) to tblptrl(B1)
```
Implementing FFT

ADDLBL ExtRamAddr, tblptrl

; add data addr offset

0081 B000 MOVWL (ExtRamAddr) & 0xff
0082 0F00 ADDWF tblptrl+B0
0083 B008 MOVWL page (ExtRamAddr)
0084 110E ADDWFC tblptrl+B1

0085 A939 tablrd 0,1,Xi ; auto increment for Imag Data
0086 A039 tblrd 0,Xi
0087 A23A tblrd 1,Xi+Bl ; real data XI
0088 A83B tablrd 0,0,Yi
0089 A03B tblrd 0,Yi
008A A23C tblrd 1,Yi+Bl ; imag data YI

; Real & Imag Data is fetched
; Compute Butterfly
;
SUB16ACC Xl,Xi,Xt ; Xt = Xi - Xl

008B 603D MOVFP Xl+BO,WREG ; get lowest byte of Xl into w
008C 0439 SUBWF Xl+BO,W ; sub lowest byte of Xi, save in Xi (B0)
008D 0141 MOVWF Xt+BO
008E 603E MOVFP Xl+Bl, WREG ; get 2nd byte of Xl into w
008F 023A SUBWFB Xl+Bl,W ; sub 2nd byte of Xi, save in Xi (Bl)
0090 0142 MOVWF Xt+Bl

ADD16 Xl,Xi ; Xi = Xi + Xl

0091 603D MOVFP Xl+BO,WREG ; get lowest byte of Xl into w
0092 0F39 ADDWF Xl+BO,W ; add lowest byte of Xi, save in Xi (B0)
0093 603E MOVFP Xl+Bl, WREG ; get 2nd byte of Xl into w
0094 113A ADDWFC Xl+Bl ; sub 2nd byte of Xi, save in Xi (Bl)

SUB16ACC Yl,Yi,Yt ; Yt = Yi - Yl

0095 603F MOVFP Yl+BO, WREG ; get lowest byte of Yl into w
0096 043B SUBWF Yl+BO,W ; sub lowest byte of Yi, save in Yi (B0)
0097 0143 MOVWF Yt+BO
0098 6040 MOVFP Yl+Bl, WREG ; get 2nd byte of Yl into w
0099 023C SUBWFB Yl+Bl, W ; sub 2nd byte of Yi, save in Yi (Bl)
009A 0144 MOVWF Yt+Bl

ADD16 Yl,Yi ; Yi = Yi + Yl

009B 603F MOVFP Yl+BO, WREG ; get lowest byte of Yl into w
009C 0F3B ADDWF Yl+BO, W ; add lowest byte of Yi, save in Yi (B0)
009D 6040 MOVFP Yl+Bl, WREG ; get 2nd byte of Yl into w
009E 113C ADDWFC Yl+Bl ; add 2nd byte of Yi, save in Yi (Bl)

; if SCALE_BUTTERFLY

RRC16 Xi

009F 1A3A RLCF Xl+B1, W ; move sign into carry bit
00A0 193A RRCF Xl+B1
00A1 1939 RRCF Xl+B0

RRC16 Yi

00A2 1A3C RLCF Yi+B1, W ; move sign into carry bit
Implementing FFT

```
00A3 193C  RRCF  Yi+B1
00A4 193B  RRCF  Yi+B0

RRC16  Xt

00A5 1A42  RLCF  Xt+B1,W  ; move sign into carry bit
00A6 1942  RRCF  Xt+B1
00A7 1941  RRCF  Xt+B0

00A8 1A44  RLCF  Yt+B1,W  ; move sign into carry bit
00A9 1944  RRCF  Yt+B1
00AA 1943  RRCF  Yt+B0

endif

; MOVFP16 Cos, AARG
00AB 782E  MOVFP  Cos+B0, AARG+B0  ; move Cos(B0) to AARG(B0)
00AC 792F  MOVFP  Cos+B1, AARG+B1  ; move Cos(B1) to AARG(B1)

; MOVFP16 Yt, BARG
00AD 7A43  MOVFP  Yt+B0, BARG+B0  ; move Yt(B0) to BARG(B0)
00AE 7B44  MOVFP  Yt+B1, BARG+B1  ; move Yt(B1) to BARG(B1)

; MOVFP16 Sin, AARG
00B4 7830  MOVFP  Sin+B0, AARG+B0  ; move Sin(B0) to AARG(B0)
00B5 7931  MOVFP  Sin+B1, AARG+B1  ; move Sin(B1) to AARG(B1)

; MOVFP16 Xt, BARG
00B6 7A41  MOVFP  Xt+B0, BARG+B0  ; move Xt(B0) to BARG(B0)
00B7 7B42  MOVFP  Xt+B1, BARG+B1  ; move Xt(B1) to BARG(B1)

00B8 E182  Call  DblMult ; SIN*Xt, Scale if necessary

; ADD32 ACC, DPX
00B9 6020  MOVFP  ACC+B0, WREG  ; get lowest byte of ACC into w
00BA 601C  ADDWF  DPX+B0  ; add lowest byte of DPX, save in DPX(B0)
00BB 6021  MOVFP  ACC+B1, WREG  ; get 2nd byte of ACC into w
00BC 111D  ADDWF  DPX+B1  ; add 2nd byte of DPX, save in DPX(B1)
00BD 6022  MOVFP  ACC+B2, WREG  ; get 3rd byte of ACC into w
00BE 111E  ADDWF  DPX+B2  ; add 3rd byte of DPX, save in DPX(B2)
00BF 6023  MOVFP  ACC+B3, WREG  ; get 4th byte of ACC into w
00C0 111F  ADDWF  DPX+B3  ; add 4th byte of DPX, save in DPX(B3)
```
Implementing FFT

MOVFP16 DPX+B2, Y1

00C1 5E3F MOVFP DPX+B2+B0, Y1+B0 ; move DPX+B2(B0) to Y1(B0)
00C2 5F40 MOVFP DPX+B2+B1, Y1+B1 ; move DPX+B2(B1) to Y1(B1)

; Y1 = COS*Yt + SIN*Xt, Scale if neces

MOVFP16 Yt, BARG

00C3 7A43 MOVFP Yt+B0, BARG+B0 ; move Yt(B0) to BARG(B0)
00C4 7B44 MOVFP Yt+B1, BARG+B1 ; move Yt(B1) to BARG(B1)

00C5 E182 Call DblMult ; SIN*Yt

MOVFP32 OPX, ACC

00C6 5C20 MOVFP OPX+B2, ACC+B2 ; move OPX(B2) to ACC(B2)
00C7 5D21 MOVFP OPX+B1, ACC+B1 ; move OPX(B1) to ACC(B1)
00C8 5E22 MOVFP OPX+B2, ACC+B2 ; move OPX(B2) to ACC(B2)
00C9 5F23 MOVFP OPX+B3, ACC+B3 ; move OPX(B3) to ACC(B3)

MOVFP16 Cos, AARG

00CA 782E MOVFP Cos+B0, AARG+B0 ; move Cos(B0) to AARG(B0)
00CB 792F MOVFP Cos+B1, AARG+B1 ; move Cos(B1) to AARG(B1)

MOVFP16 Xt, BARG

00CC 7A41 MOVFP Xt+B0, BARG+B0 ; move Xt(B0) to BARG(B0)
00CD 7B42 MOVFP Xt+B1, BARG+B1 ; move Xt(B1) to BARG(B1)

00CE E182 Call DblMult ; COS*Xt, Scale if necessary

SUB32 ACC, DPX

00CF 6020 MOVFP ACC+B0, WREG ; get lowest byte of ACC into w
00D0 051C SUBWF DPX+B0 ; sub lowest byte of DPX, save in DPX(B0)
00D1 6021 MOVFP ACC+B1, WREG ; get 2nd byte of ACC into w
00D2 031D SUBWF DPX+B1 ; sub 2nd byte of DPX, save in DPX(B1)
00D3 6022 MOVFP ACC+B2, WREG ; get 3rd byte of ACC into w
00D4 031E SUBWF DPX+B2 ; sub 3rd byte of DPX, save in DPX(B2)
00D5 6023 MOVFP ACC+B3, WREG ; get 4th byte of ACC into w
00D6 031F SUBWF DPX+B3 ; sub 4th byte of DPX, save in DPX(B3)

MOVFP16 DPX+B2, Xl

00D7 5E2D MOVFP DPX+B2+B0, Xl+B0 ; move DPX+B2(B0) to Xl(B0)
00D8 5F3E MOVFP DPX+B2+B1, Xl+B1 ; move DPX+B2(B1) to Xl(B1)

; Xl = COS*Xt - SIN*Yt, Scale if neces

; Store results of butterfly

DEC16 tblptr1 ; table pointer already loaded with I

00D9 2900 CLRF WREG
00DA 070D DECF tblptr1+B0
00DB 030E SUBWF tblptr1+B1

00DC A439 tlwt 0, Xl
00DD AF3A tablw 1, 1, Xl+B1 ; auto increment for Imag Data

© 1993 Microchip Technology Inc.
Implementing FFT

00DE A43B  tlwt  0,Yi
00DF A33C  tablwt  1,0,Yi+B1 ; Xi & Yi stored

MOVFP16 VarL,tblptrl ; read data(L)
00E0 6D37  MOVFP VarL+B0,tblptrl+B0 ; move VarL(B0) to tblptrl(B0)
00E1 6E38  MOVFP VarL+B1,tblptrl+B1 ; move VarL(B1) to tblptrl(B1)

ADDLBL ExtRamAddr,tblptrl ; add data addr offset
00E2 B000  MOVLW (ExtRamAddr) & 0xff
00E3 0F0D  ADDWF tblptrl+B0
00E4 B008  MOVLW page (ExtRamAddr)
00E5 110E  ADDWFC tblptrl+B1

00E6 A43D  tlwt  0,Xl
00E7 AF3E  tablwt 1,1,Xl+B1 ; auto increment for Imag Data
00E8 A43F  tlwt  0,Yl
00E9 AE40  tablwt 1,0,Yl+B1 ; X(L) & Y(L) stored

; Increment for next Iloop

; RLC16AB countl,temp ; temp = countl*2
00EA 8804  BCF _carry
00EB 1A24  RLCF countl+B0,W
00EC 0145  MOVF W temp+B0
00ED 1A24  RLCF countl+B1,W
00EE 0146  MOVF W temp+B1

ADD16 temp,VarIloop ; I = I + temp
00EF 6045  MOVFP temp+B0,WREG ; get lowest byte of temp into w
00F0 0F32  ADDWF VarIloop+B0 ; add lowest byte of VarIloop, save in VarIloop(B0)

00F1 6046  MOVFP temp+B1,WREG ; get 2nd byte of temp into w
00F2 1133  ADDWFC VarIloop+B1 ; add 2nd byte of VarIloop, save in VarIloop(B1)

MOVK16 (FftLen*2),temp
00F3 B000  MOVLW ((FftLen*2)) & 0xff
00F4 0145  MOVF temp+B0
00F5 B002  MOVLW ((FftLen*2))/256
00F6 0146  MOVF temp+B1

SUB16 VarIloop,temp ; temp = 2*FftLen - I
00F7 6032  MOVFP VarIloop+B0,WREG ; get lowest byte of VarIloop into w
00F8 0545  SUBWF temp+B0 ; sub lowest byte of temp, save in temp(B0)
00F9 6033  MOVFP VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
00FA 0346  SUBWFB temp+B1 ; sub 2nd byte of temp, save in temp(B1)

DEC16 temp
00FB 2900  CLRIF WREG
00FC 0745  DECF temp+B0
00FD 0346  SUBWFB temp+B1

© 1993 Microchip Technology Inc.
Implementing FFT

```
00E 9746           btfss temp+Bl,MSB
00F C06A           goto Iloop        ; while I < 2*FftLen

; I Loop end
;
; increment for next J Loop
;
INC16 VarJloop     ; J = J + 1

0100 2900           CLRIF WREG
0101 1534           INCIF VarJloop+B0
0102 1135           ADDWF VarJloop+B1

MOV16 count2,temp

0103 6026           MOVFF count2+B0,WREG ; get byte of count2 into w
0104 0145           MOVWF temp+B0        ; move to temp(B0)
0105 6027           MOVFF count2+B1,WREG ; get byte of count2 into w
0106 0146           MOVWF temp+B1        ; move to temp(B1)

SUB16 VarJloop,temp  ; temp = count2 - J

0107 6034           MOVFF VarJloop+B0,WREG ; get lowest byte of VarJloop into w
0108 0545           SUBWF temp+B0        ; sub lowest byte of temp, save in temp(B0)
0109 6035           MOVFF VarJloop+B1,WREG ; get 2nd byte of VarJloop into w
010A 0346           SUBWFB temp+B1       ; sub 2nd byte of temp, save in temp(B1)

DEC16 temp

010B 2900           CLRIF WREG
010C 0745           DECIF temp+B0
010D 0346           SUBWFB temp+B1

010E 9746           btfss temp+Bl,MSB
010F C051           goto Jloop         ; while J < count2

; J Loop end
;
; increment for next K Loop
;
RLC16 TF_Offset     ; TF_Offset = 2 * TF_Offset

0110 8804           BCF _carry
0111 1B2A           RLCF TF_Offset+B0
0112 1B2B           RLCF TF_Offset+B1

0113 1736           decfsz VarKloop
0114 C046           goto Kloop        ; while K < Power

; return          ; FFT complete
;
; K Loop End
; FFT Computation Over with data scrambled
; Descramble the data using "Unscramble" Routine
;
;******************************************************************
; Unscramble Data Order Sequence
; A digit reverse counter
;******************************************************************
```
Implementing FFT

include "reverse.asm"

;******************************************************************
; A digit reverse counter 
;
; Unscramble Data Order Sequence Of Radix-2 FFT 
; Length (must be a power of 2) is limited only by 
; the amount of External RAM available and must be 
; a number less than 2**15 
; ;******************************************************************

Unscramble

CLR16 VarJloop ; J = 0
0116 2934 CLRF VarJloop+BO
0117 2935 CLRF VarJloop+Bl

0118 2933 clrF VarIloop+Bl
MOVK 1,VarIloop ; I = 1
0119 B001 MOVLW 1
011A 0132 MOVWF VarIloop

nextI

MOVK16 FftLen/2,VarKloop

011B B080 MOVLW (FftLen/2) & 0xff
011C 0136 MOVWF VarKloop+BO
011D B000 MOVLW (FftLen/2)/256
011E 0137 MOVWF VarKloop+Bl

go to testK

KlessJ

SUB16 VarKloop,VarJloop ; J = J - K

0120 6036 MOVFP VarKloop+BO,WREG ; get lowest byte of VarKloop into w
0121 0534 SUBWF VarJloop+BO ; sub lowest byte of VarJloop, save in VarJloop(B0)

0122 6037 MOVFP VarKloop+Bl,WREG ; get 2nd byte of VarKloop into w
0123 0335 SUBWFB VarJloop+Bl ; sub 2nd byte of VarJloop, save in VarJloop(B1)

RRC16 VarKloop ; K = K/2
0124 1A37 RLCF VarKloop+Bl,W ; move sign into carry bit
0125 1937 RRCF VarKloop+Bl
0126 1936 RRCF VarKloop+BO

testK

MOV16 VarJloop,temp

0127 6034 MOVFP VarJloop+BO,WREG ; get byte of VarJloop into w
0128 0145 MOVWF temp+BO ; move to temp(B0)
0129 6035 MOVFP VarJloop+Bl,WREG ; get byte of VarJloop into w
012A 0146 MOVWF temp+Bl ; move to temp(B1)

SUB16 VarKloop,temp ; temp = J - K
012B 6036 MOVFP VarKloop+BO,WREG ; get lowest byte of VarKloop into w
012C 0545 SUBWFB temp+BO ; sub lowest byte of temp, save in temp(B0)
Implementing FFT

; get 2nd byte of VarKloop into w
MOVFP VarKloop+Bl,WREG

; sub 2nd byte of temp, save in temp(B1)
SUBWFB temp+B1

btfss temp+B1,MSB

; while K < J
goto KlessJ

ADD16 VarKloop,VarJloop

; J = J + K
MOVFP VarKloop+B0,WREG
ADDWF VarJloop+B0

VarJloop(B0)

MOVFP VarKloop+Bl,WREG
ADDWFC VarJloop+Bl

VarJloop(B1)

; if (i < j) then swap data(i) & data(j)

MOV16 VarJloop,temp

MOVFP VarJloop+B0,WREG
MOVFP VarJloop+BO,WREG

MOVWF temp+B0

MOVFP VarJloop+Bl,WREG
MOVFP VarJloop+Bl,WREG

MOVWF temp+B1

MOV16 VarJloop,temp

; temp = J - I

MOVFP VarJloop+Bl,WREG
MOVFP VarJloop+B0,WREG

SUBWF temp+B0

SUBWFB temp+B1

SUB16 VarJloop,temp

temp+Bl,MSB

; temp = J - I

btfsc temp+B1,MSB

; read data(i)

btfsc temp+B1,MSB

; swap data

bcf .carry

; add twice the addr, since Real Data

BCF .carry

RLCF VarKloop+B0,W

MOVWF tblptrl+B0

RLCF VarKloop+B1,W

MOVWF tblptrl+B1

ADDCBL ExtRamAddr,tblptrl

; is followed by Imag Data

MOVLW (ExtRamAddr) & Oxff

ADDWF tblptrl+B0

MOVLW page (ExtRamAddr)

ADDWF tblptrl+B1
Implementing FFT

014B A939	tablrd 0,1,Xi ; auto increment for Imag Data
014C A039
tlrd 0,Xi
014D A23A
tlrd 1,Xi+B1 ; real data XI
014E A83B	tablrd 0,0,Yi
014F A03B
tlrd 0,Yi
0150 A23C
tlrd 1,Yi+B1 ; imag data Yi

; read data{j}

RLC16AB VarJloop, tblptrl ; add twice the addr, since Real Data

0151 8804	BCF _carry
0152 1A34	RLCF VarJloop+B0,W
0153 010D	MOVWF tblptrl+B0
0154 1A35	RLCF VarJloop+B1,W
0155 010E	MOVWF tblptrl+B1

ADDLBL ExtRamAddr, tblptrl ; is followed by Imag Data

0156 B000	MOVLW (ExtRamAddr) & 0xff
0157 0F0D	ADDWF tblptrl+B0
0158 B008	MOVLW page (ExtRamAddr)
0159 110E	ADDWF tblptrl+B1

015A A93D	tablrd 0,1,Xi ; auto increment for Imag Data
015B A03D
tlrd 0,Xi
015C A23E
tlrd 1,Xi+B1 ; real data XL
015D A83F	tablrd 0,0,Yi
015E A03F
tlrd 0,Yi
015F A240
tlrd 1,Yi+B1 ; imag data YL

; Interchange data(I) & data(J)

DEC16 tblptrl

0160 2900	CLR WREG
0161 070D	DECF tblptrl+B0
0162 030E	SUBWF tblptrl+B1

0163 A439	tlwt 0,Xi
0164 AF3A	tablwt 1,1,Xi+B1 ; auto increment for Imag Data
0165 A43B
tlwt 0,Yi
0166 A53C	tablwt 1,0,Yi+B1 ; X(I) & Y(I) stored

RLC16AB VarJloop, tblptrl ; add twice the addr, since Real Data

0167 8804	BCF _carry
0168 1A32	RLCF VarJloop+B0,W
0169 010D	MOVWF tblptrl+B0
016A 1A33	RLCF VarJloop+B1,W
016B 010E	MOVWF tblptrl+B1

ADDLBL ExtRamAddr, tblptrl ; is followed by Imag Data

016C B000	MOVLW (ExtRamAddr) & 0xff
016D 0F0D	ADDWF tblptrl+B0
016E B008	MOVLW page (ExtRamAddr)
016F 110E	ADDWF tblptrl+B1
Implementing FFT

0170 A43D   tlwt 0,Xl
0171 AF3E   tablw 1,1,Xl+Bl ; auto increment for Imag Data
0172 A43F   tlwt 0,Yl
0173 AE40   tablw 1,0,Yl+Bl ; X(L) & Y(L) stored

; increment I
; incl

INC16 VarIloop

0174 2900   CLRF WREG
0175 1532   INCF VarIloop+80
0176 1133   ADDWF VarIloop+81

MOVK16 DigitRevCount,temp

0177 B0EF   MOVLW (DigitRevCount) & Oxff
0178 0145   MOVWF temp+80
0179 B000   MOVLW (DigitRevCount)/256
017A 0146   MOVWF temp+81

SUB16 VarIloop,temp ; temp = DigitRevCount - i

017B 6032   MOVFP VarIloop+80,WREG ; get lowest byte of VarIloop into w
017C 0545   SUBWF temp+80 ; sub lowest byte of temp, save in temp(B0)
017D 6033   MOVFP VarIloop+81,WREG ; get 2nd byte of VarIloop into w
017E 0346   SUBWFB temp+81 ; sub 2nd byte of temp, save in temp(Bl)

017F 9746   btfs temp+81,MSB
0180 C11B   goto nextI ; while i < DigitRevCount
0181 0002   return

; End digit reverse counter
;
;**********************************************************************
;
;**********************************************************************

SIGNED equ TRUE

include "17c42mpy.mac"

;**********************************************************************
;
;**********************************************************************

include "fft256.tbl"
;
; 256 Point FFT Sine Table
; coefficient table (size of table is 3n/4).
; SineTable
;
0294 0000   data 0
0295 0324   data 804
0296 0648   data 1608
0297 096A   data 2410
0298 0C8C   data 3212
0299 OFAB   data 4011

© 1993 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>029A</td>
<td>12C8</td>
<td>035A</td>
<td>15E2</td>
<td>041A</td>
<td>18F9</td>
</tr>
<tr>
<td>029B</td>
<td>140B</td>
<td>035B</td>
<td>1806</td>
<td>041B</td>
<td>1A9F</td>
</tr>
<tr>
<td>029C</td>
<td>16E2</td>
<td>035C</td>
<td>1904</td>
<td>041C</td>
<td>1B7F</td>
</tr>
<tr>
<td>029D</td>
<td>18F9</td>
<td>035D</td>
<td>1A9F</td>
<td>041D</td>
<td>1C7E</td>
</tr>
<tr>
<td>029E</td>
<td>1A9F</td>
<td>035E</td>
<td>1B7F</td>
<td>041E</td>
<td>1D36</td>
</tr>
<tr>
<td>029F</td>
<td>1C7E</td>
<td>035F</td>
<td>1D36</td>
<td>041F</td>
<td>1E63</td>
</tr>
<tr>
<td>02A0</td>
<td>2223</td>
<td>0360</td>
<td>1E63</td>
<td>0420</td>
<td>2528</td>
</tr>
<tr>
<td>02A1</td>
<td>2826</td>
<td>0361</td>
<td>1F1A</td>
<td>0421</td>
<td>2948</td>
</tr>
<tr>
<td>02A2</td>
<td>2B1F</td>
<td>0362</td>
<td>1F1A</td>
<td>0422</td>
<td>2A69</td>
</tr>
<tr>
<td>02A3</td>
<td>2E11</td>
<td>0363</td>
<td>1F1A</td>
<td>0423</td>
<td>2B40</td>
</tr>
<tr>
<td>02A4</td>
<td>30FB</td>
<td>0364</td>
<td>1F1A</td>
<td>0424</td>
<td>2E2B</td>
</tr>
<tr>
<td>02A5</td>
<td>33DF</td>
<td>0365</td>
<td>1F1A</td>
<td>0425</td>
<td>2F0A</td>
</tr>
<tr>
<td>02A6</td>
<td>36BA</td>
<td>0366</td>
<td>1F1A</td>
<td>0426</td>
<td>2F89</td>
</tr>
<tr>
<td>02A7</td>
<td>390C</td>
<td>0367</td>
<td>1F1A</td>
<td>0427</td>
<td>3061</td>
</tr>
<tr>
<td>02A8</td>
<td>3C56</td>
<td>0368</td>
<td>1F1A</td>
<td>0428</td>
<td>3141</td>
</tr>
<tr>
<td>02A9</td>
<td>3F17</td>
<td>0369</td>
<td>1F1A</td>
<td>0429</td>
<td>3227</td>
</tr>
<tr>
<td>02AA</td>
<td>41CE</td>
<td>036A</td>
<td>1F1A</td>
<td>042A</td>
<td>3306</td>
</tr>
<tr>
<td>02AB</td>
<td>447A</td>
<td>036B</td>
<td>1F1A</td>
<td>042B</td>
<td>3382</td>
</tr>
<tr>
<td>02AC</td>
<td>471C</td>
<td>036C</td>
<td>1F1A</td>
<td>042C</td>
<td>3458</td>
</tr>
<tr>
<td>02AD</td>
<td>49B4</td>
<td>036D</td>
<td>1F1A</td>
<td>042D</td>
<td>3534</td>
</tr>
<tr>
<td>02AE</td>
<td>4CBF</td>
<td>036E</td>
<td>1F1A</td>
<td>042E</td>
<td>3610</td>
</tr>
<tr>
<td>02AF</td>
<td>5133</td>
<td>036F</td>
<td>1F1A</td>
<td>042F</td>
<td>3686</td>
</tr>
<tr>
<td>02B0</td>
<td>539B</td>
<td>0370</td>
<td>1F1A</td>
<td>0430</td>
<td>3762</td>
</tr>
<tr>
<td>02B1</td>
<td>55F5</td>
<td>0371</td>
<td>1F1A</td>
<td>0431</td>
<td>3838</td>
</tr>
<tr>
<td>02B2</td>
<td>5842</td>
<td>0372</td>
<td>1F1A</td>
<td>0432</td>
<td>3914</td>
</tr>
<tr>
<td>02B3</td>
<td>5A92</td>
<td>0373</td>
<td>1F1A</td>
<td>0433</td>
<td>398A</td>
</tr>
<tr>
<td>02B4</td>
<td>5C56</td>
<td>0374</td>
<td>1F1A</td>
<td>0434</td>
<td>3A5C</td>
</tr>
<tr>
<td>02B5</td>
<td>5DC3</td>
<td>0375</td>
<td>1F1A</td>
<td>0435</td>
<td>3B36</td>
</tr>
<tr>
<td>02B6</td>
<td>5ED7</td>
<td>0376</td>
<td>1F1A</td>
<td>0436</td>
<td>3C19</td>
</tr>
<tr>
<td>02B7</td>
<td>60EB</td>
<td>0377</td>
<td>1F1A</td>
<td>0437</td>
<td>3C9F</td>
</tr>
<tr>
<td>02B8</td>
<td>62F1</td>
<td>0378</td>
<td>1F1A</td>
<td>0438</td>
<td>3D7B</td>
</tr>
<tr>
<td>02B9</td>
<td>64EB</td>
<td>0379</td>
<td>1F1A</td>
<td>0439</td>
<td>3E5F</td>
</tr>
<tr>
<td>02BA</td>
<td>66C2</td>
<td>037A</td>
<td>1F1A</td>
<td>043A</td>
<td>3F3B</td>
</tr>
<tr>
<td>02BB</td>
<td>68A6</td>
<td>037B</td>
<td>1F1A</td>
<td>043B</td>
<td>3FBB</td>
</tr>
<tr>
<td>02BC</td>
<td>6A6D</td>
<td>037C</td>
<td>1F1A</td>
<td>043C</td>
<td>3F99</td>
</tr>
<tr>
<td>02BD</td>
<td>6C23</td>
<td>037D</td>
<td>1F1A</td>
<td>043D</td>
<td>3F77</td>
</tr>
<tr>
<td>02BE</td>
<td>6E59</td>
<td>037E</td>
<td>1F1A</td>
<td>043E</td>
<td>3F55</td>
</tr>
<tr>
<td>02BF</td>
<td>6F5E</td>
<td>037F</td>
<td>1F1A</td>
<td>043F</td>
<td>3F33</td>
</tr>
<tr>
<td>02C0</td>
<td>70EB</td>
<td>0380</td>
<td>1F1A</td>
<td>0440</td>
<td>3E1B</td>
</tr>
<tr>
<td>02C1</td>
<td>7254</td>
<td>0381</td>
<td>1F1A</td>
<td>0441</td>
<td>3C98</td>
</tr>
<tr>
<td>02C2</td>
<td>73B5</td>
<td>0382</td>
<td>1F1A</td>
<td>0442</td>
<td>3B76</td>
</tr>
<tr>
<td>02C3</td>
<td>7504</td>
<td>0383</td>
<td>1F1A</td>
<td>0443</td>
<td>3A54</td>
</tr>
<tr>
<td>02C4</td>
<td>7641</td>
<td>0384</td>
<td>1F1A</td>
<td>0444</td>
<td>3932</td>
</tr>
<tr>
<td>02C5</td>
<td>776B</td>
<td>0385</td>
<td>1F1A</td>
<td>0445</td>
<td>3810</td>
</tr>
<tr>
<td>02C6</td>
<td>7884</td>
<td>0386</td>
<td>1F1A</td>
<td>0446</td>
<td>3788</td>
</tr>
<tr>
<td>02C7</td>
<td>7989</td>
<td>0387</td>
<td>1F1A</td>
<td>0447</td>
<td>3666</td>
</tr>
<tr>
<td>02C8</td>
<td>7A7C</td>
<td>0388</td>
<td>1F1A</td>
<td>0448</td>
<td>3544</td>
</tr>
<tr>
<td>02C9</td>
<td>7B5C</td>
<td>0389</td>
<td>1F1A</td>
<td>0449</td>
<td>3422</td>
</tr>
<tr>
<td>02CA</td>
<td>7C29</td>
<td>038A</td>
<td>1F1A</td>
<td>044A</td>
<td>3300</td>
</tr>
<tr>
<td>02CB</td>
<td>7CA7</td>
<td>038B</td>
<td>1F1A</td>
<td>044B</td>
<td>3178</td>
</tr>
<tr>
<td>02CC</td>
<td>7D99</td>
<td>038C</td>
<td>1F1A</td>
<td>044C</td>
<td>3056</td>
</tr>
<tr>
<td>02CD</td>
<td>7E1D</td>
<td>038D</td>
<td>1F1A</td>
<td>044D</td>
<td>2934</td>
</tr>
<tr>
<td>02CE</td>
<td>7E9C</td>
<td>038E</td>
<td>1F1A</td>
<td>044E</td>
<td>2812</td>
</tr>
<tr>
<td>02CF</td>
<td>7FF5</td>
<td>038F</td>
<td>1F1A</td>
<td>044F</td>
<td>2690</td>
</tr>
</tbody>
</table>

; 
CosTable 
; 
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>02D4</td>
<td>7FFF</td>
</tr>
<tr>
<td>02D5</td>
<td>7FF5</td>
</tr>
<tr>
<td>02D6</td>
<td>7FD8</td>
</tr>
<tr>
<td>02D7</td>
<td>7FA6</td>
</tr>
<tr>
<td>02D8</td>
<td>7F61</td>
</tr>
<tr>
<td>02D9</td>
<td>7F09</td>
</tr>
<tr>
<td>02DA</td>
<td>7E9C</td>
</tr>
<tr>
<td>02DB</td>
<td>7E1D</td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>02DC</td>
<td>7D89</td>
</tr>
<tr>
<td>02DD</td>
<td>7C31</td>
</tr>
<tr>
<td>02DE</td>
<td>7C29</td>
</tr>
<tr>
<td>02DF</td>
<td>7B5C</td>
</tr>
<tr>
<td>02E0</td>
<td>7A7C</td>
</tr>
<tr>
<td>02E1</td>
<td>7989</td>
</tr>
<tr>
<td>02E2</td>
<td>7884</td>
</tr>
<tr>
<td>02E3</td>
<td>776B</td>
</tr>
<tr>
<td>02E4</td>
<td>7641</td>
</tr>
<tr>
<td>02E5</td>
<td>7504</td>
</tr>
<tr>
<td>02E6</td>
<td>73B5</td>
</tr>
<tr>
<td>02E7</td>
<td>7254</td>
</tr>
<tr>
<td>02E8</td>
<td>70E2</td>
</tr>
<tr>
<td>02E9</td>
<td>6F5E</td>
</tr>
<tr>
<td>02EA</td>
<td>6DC9</td>
</tr>
<tr>
<td>02EB</td>
<td>6C23</td>
</tr>
<tr>
<td>02EC</td>
<td>6A6D</td>
</tr>
<tr>
<td>02ED</td>
<td>68A6</td>
</tr>
<tr>
<td>02EE</td>
<td>66CF</td>
</tr>
<tr>
<td>02EF</td>
<td>64E8</td>
</tr>
<tr>
<td>02F0</td>
<td>62F1</td>
</tr>
<tr>
<td>02F1</td>
<td>60E8</td>
</tr>
<tr>
<td>02F2</td>
<td>5ED7</td>
</tr>
<tr>
<td>02F3</td>
<td>5CB3</td>
</tr>
<tr>
<td>02F4</td>
<td>5A82</td>
</tr>
<tr>
<td>02F5</td>
<td>5842</td>
</tr>
<tr>
<td>02F6</td>
<td>55F5</td>
</tr>
<tr>
<td>02F7</td>
<td>539B</td>
</tr>
<tr>
<td>02F8</td>
<td>5133</td>
</tr>
<tr>
<td>02F9</td>
<td>4EBF</td>
</tr>
<tr>
<td>02FA</td>
<td>4C3F</td>
</tr>
<tr>
<td>02FB</td>
<td>49B4</td>
</tr>
<tr>
<td>02FC</td>
<td>471C</td>
</tr>
<tr>
<td>02FD</td>
<td>447A</td>
</tr>
<tr>
<td>02FE</td>
<td>41CE</td>
</tr>
<tr>
<td>02FF</td>
<td>3F17</td>
</tr>
<tr>
<td>0300</td>
<td>3C56</td>
</tr>
<tr>
<td>0301</td>
<td>398C</td>
</tr>
<tr>
<td>0302</td>
<td>36BA</td>
</tr>
<tr>
<td>0303</td>
<td>33DF</td>
</tr>
<tr>
<td>0304</td>
<td>30FB</td>
</tr>
<tr>
<td>0305</td>
<td>2E11</td>
</tr>
<tr>
<td>0306</td>
<td>2B1F</td>
</tr>
<tr>
<td>0307</td>
<td>2826</td>
</tr>
<tr>
<td>0308</td>
<td>2528</td>
</tr>
<tr>
<td>0309</td>
<td>2223</td>
</tr>
<tr>
<td>030A</td>
<td>1F1A</td>
</tr>
<tr>
<td>030B</td>
<td>1COB</td>
</tr>
<tr>
<td>030C</td>
<td>1BF9</td>
</tr>
<tr>
<td>030D</td>
<td>15E2</td>
</tr>
<tr>
<td>030E</td>
<td>12C8</td>
</tr>
<tr>
<td>030F</td>
<td>0FAB</td>
</tr>
<tr>
<td>0310</td>
<td>0C8C</td>
</tr>
<tr>
<td>0311</td>
<td>096A</td>
</tr>
<tr>
<td>0312</td>
<td>064A</td>
</tr>
<tr>
<td>0313</td>
<td>032A</td>
</tr>
<tr>
<td>0314</td>
<td>0000</td>
</tr>
<tr>
<td>0315</td>
<td>FCDC</td>
</tr>
<tr>
<td>0316</td>
<td>F9B8</td>
</tr>
<tr>
<td>0317</td>
<td>F696</td>
</tr>
<tr>
<td>0318</td>
<td>F374</td>
</tr>
<tr>
<td>0319</td>
<td>F055</td>
</tr>
<tr>
<td>031A</td>
<td>ED38</td>
</tr>
<tr>
<td>031B</td>
<td>EA1E</td>
</tr>
<tr>
<td>031C</td>
<td>E707</td>
</tr>
<tr>
<td>031D</td>
<td>E3F5</td>
</tr>
<tr>
<td>031E</td>
<td>E0E6</td>
</tr>
<tr>
<td>031F</td>
<td>DDDD</td>
</tr>
<tr>
<td>0320</td>
<td>DADD</td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
Implementing FFT

0321 D7DA  data  -10278
0322 D4E1  data  -11039
0323 D1EF  data  -11793
0324 CF05  data  -12539
0325 CC21  data  -13279
0326 C946  data  -14010
0327 C674  data  -14732
0328 C3AA  data  -15446
0329 C0E9  data  -16151
032A BE32  data  -16846
032B BB86  data  -17530
032C BBE4  data  -18204
032D B64C  data  -18868
032E B3C1  data  -19519
032F B141  data  -20159
0330 AECD  data  -20787
0331 AC65  data  -21403
0332 AA0B  data  -22005
0333 A7BE  data  -22594
0334 A57E  data  -23170
0335 A34D  data  -23731
0336 A129  data  -24279
0337 9F15  data  -24811
0338 9D0F  data  -25329
0339 9B18  data  -25832
033A 9931  data  -26319
033B 975A  data  -26790
033C 9593  data  -27245
033D 93DB  data  -27683
033E 9237  data  -28105
033F 90A2  data  -28510
0340 8F1E  data  -28898
0341 8DAC  data  -29268
0342 8C4B  data  -29621
0343 8AFC  data  -29956
0344 89BF  data  -30273
0345 8895  data  -30571
0346 877C  data  -30852
0347 8677  data  -31113
0348 8584  data  -31356
0349 84A4  data  -31580
034A 83D7  data  -31785
034B 831D  data  -31971
034C 8277  data  -32137
034D 81E3  data  -32285
034E 8164  data  -32412
034F 80F7  data  -32521
0350 809F  data  -32609
0351 805A  data  -32678
0352 8028  data  -32728
0353 800B  data  -32757

;*******************************************************************
; FFT Input/Output Data Stored In External RAM
; Operate Processor In Extended Microcontroller Mode
; External Data Starts at Address 0x0800, with 2 bytes of
; Real Data followed by 2 bytes of Imaginary Data.
;*******************************************************************
ORG EXT_RAM_START_ADDR
ExtRamAddr
END

Errors : 0
Warnings : 0
INTRODUCTION

A general purpose resonator routine is implemented using PIC17C42. This routine is used to generate multiple tones. A tone signal is normally generated using extensive table lookup schemes. When a multiple tone signal is desired, each tone must have its own lookup table, thus requiring a large amount of storage space, especially when various frequencies are to be generated. This application note implements a tone generation using recursive techniques. The algorithm for a resonator is developed and implemented using PIC17C42.

THEORY

Generation of a single tone basically implies generating samples of a sine/cosine wave. The Z-Transform of a sine wave is given as follows:

\[
Z[\sin (\omega T)] = \frac{Y(z)}{X(z)} = \frac{z^* \sin (\omega T)}{z^2 - 2z \cos (\omega T) + 1}
\]

The impulse response of the above transform (i.e. for \(X(z) = 1\)) will generate a sine wave of frequency \(\omega\) sampled at a rate of \(T (= 1/fs)\). Thus the above equation is translated to:

\[
Y(z) = \frac{z^{-1} \sin (\omega T)}{1 - 2z^{-1} \cos (\omega T) + z^2}
\]

The above equation can be rewritten in a difference equation form as follows:

\[
y(n) - 2y(n-1) \cos (\omega T) + y(n-2) = x(n-1) \sin (\omega T)
\]

Rearranging the above equation and setting, \(x(n)\) as an impulse sequence, the following recursive equations are obtained:

\[
y(n) = 2K_1 \cdot y(n-1) - y(n-2)
y(n-2) = y(n-1)
y(n-1) = y(n)
\]

with the following conditions:

\[
K_1 = \cos (\omega T)
K_2 = \text{initial } y(n-1) = \sin (\omega T)
K_3 = \text{initial } y(n-2) = 0
\]

IMPLEMENTATION

The above developed algorithm is implemented as a subroutine using PIC17C42. All computations are performed using double precision arithmetic (16/32-bits). The recursive tone generation algorithm is implemented as a subroutine (labelled as "Resonator"). This subroutine generates samples of a single tone. To generate multiple frequencies, simply call this resonator routine for the desired frequencies and sum the outputs. The three tone co-efficients are stored in program memory and are read into the data memory using `TABLRD` instructions.

The fully commented code is listed in Appendix A. The timing and memory requirements are included in the comment sections of the code. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note ANDS00540. This code can be easily modified and used in various applications like DTMF generation, sound generation, etc. The tones generated can easily be output to an on chip PWM channel which in turn can drive a speaker for producing various sounds. If using a PWM channel, it is suggested to set the PWM frequency much higher than the sampling frequency used (in the example code, for 8 KHz sampling frequency, use at least 20 KHz PWM frequency).

As an example, a dual tone is generated and the resulting digital waveform is analyzed. The main program calls the function "Resonator" twice for generating the two desired tones and the two outputs are summed. A sampling frequency of 8 KHz was used to generate a dual tone of 800 Hz and 1,100 KHz. The resulting waveform is shown in Figure 1. The spectrum of the signal shown in Figure 2 shows 2 peaks corresponding to the two desired tones (800 Hz & 1.10 KHz). The assembly code was tested using PICMASTER™ (Microchip's Universal In-Circuit Emulator System).

The generated tones were captured into the PICMASTER's trace buffer and then transferred to Microsoft Excel using Dynamic Data Exchange (DDE). Once the data is in Excel it is analyzed using Excel's FFT utility.
### Tone Generation

#### PERFORMANCE

Table 1 below provides the performance of the resonator (labeled in the source code as "Resonator") in terms of both timing and memory requirements. Since a double precision multiplier is used (software implementation), the multiplier timing is not always constant. Therefore the timings are given for the worst case. Note that irrespective of the frequency and the sampling (resolution) of the tone, program memory requirements is only 54 locations which in case of table lookup could be very large.

**TABLE 1**

<table>
<thead>
<tr>
<th></th>
<th>235 Cycles</th>
<th>58.75 µs</th>
<th>47 µs</th>
<th>9 + 9(# of tones to be generated)</th>
<th>54 locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td></td>
<td>Time @ 16 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time @ 25 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program memory</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### APPLICATIONS

Tone generation is required in many application. The code provided in this application note may be used as a general purpose routine to generate desired tones. It can be used in applications involving secure off-site control, where commands/data in the format of tones are transmitted over a telephone line. The tone generation finds applications involving signal modulations as well. The routine can be used to generate audible tones and output to a speaker connected to an I/O Port or a PWM channel.

---

Author: Amar Palacherla

Logic Products Division

© 1993 Microchip Technology Inc.
APPENDIX A: TONE.LST

MPASM B0.54

;************************************************************************
; Dual Tone Generation
;
; A generic resonator subroutine is implemented to generate
; tones. Samples Of A Sin/Cos Wave are generated using
; recursive techniques. Table Lookups are thus avoided
; This is especially useful in generating multiple tones
; (e.g. DTMF tone generation, tone signalling, etc), or programmable
; tone generation which may vary for each application unit.
;
;************************************************************************

LIST P-17C42, C=80, T=ON, R=DEC, N=0

include "17c42.h"

include "17c42.mac"

;**********************************************************
;
; TBLADDR
;
; DESCRIPTION:
; Load 16 bit table pointer with specified label
;
; TIMING (in cycles):
; 4
;
; TBLADDR MACRO label

MOVLW (label)
MOVF tblptrl
MOVLW page (label)
MOVF tblptrh

ENDM

;**********************************************************
;
; ADDLBL
;
; DESCRIPTION:
; Add A Label (16 bit constant) To A File Register (1
;
; TIMING (in cycles):
; 4
;
; ADDLBL MACRO label,f

MOVLW (label)
ADDWF f+BO
MOVLW page (label)
ADDWFC f+BI

ENDM

© 1993 Microchip Technology Inc.
;**********************************************************
; CB LOCK
; BO,B1,B2,B3 RAM offset constants
; CBLOCK 0x18
; AARG,AARG1 16 bit multiplier A
; BARG,BARG1 16 bit multiplicand B
; DFx,DFx1,DFx2,DFx3 32 bit multiplier result = A*B
; CBLOCK 0
; CBLOCK 0
; CBLOCK 0
; CBLOCK
; CBLOCK
; CBLOCK 0
; CBLOCK 0
; CBLOCK 0
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLOCK
; CBLO
;**********************************************************
;Resonator Co-efficients For Desired Tones
ORG coeff_addr

; Tone 1 Resonator Constants
;Sample Rate = \( f_s = 8 \text{ khz} \), Tone Freq = \( f = 0.800 \text{ Khz} \)

0030 678E DATA 26510 ; \( K_1 = \cos(wT) = \cos(360*f/\pi) \)
0031 259E DATA 9630 ; \( K_2 = \sin(wT) = \sin(360*f/\pi) \)
0032 0000 DATA 0 ; \( K_3 \) is init value of \( y(n-2) \)

; Tone 2 Resonator Constants
;Sample Rate = \( f_s = 8 \text{ khz} \), Tone Freq = \( f = 1.10 \text{ Khz} \)

0033 5321 DATA 21281 ; \( K_1 = \cos(wT) = \cos(360*f/\pi) \)
0034 1855 DATA 6229 ; \( K_2 = \sin(wT) = \sin(360*f/\pi) \)
0035 0000 DATA 0 ; \( K_3 \) is init value of \( y(n-2) \)

;**********************************************************

ORG 0x0040

start call Coeff_Read ; load table pointers with a dummy addr

MOVK16 DummyAddr, tblptrl

0041 B000 MOVWLW (0x0A00) & 0xff
0042 010D MOVWF tblptrl+H0
0043 B00A MOVWLW ((0x0A00) >> 8)
0044 010E MOVWF tblptrl+Bl

NextSample
nop
tlw \( 0,\text{dualTone} \)
capture

0047 AE3C tablw 1,0,\text{dualTone}+Bl ; for PIC-MASTER tr
0048 0000 nop

0049 B029 movlw tone1Kl ; load indirect add
004A 0101 movwf fsr0 ; for Tone 1
004B E06C call Resonator ; Compute next samp

004C B032 movlw tone2Kl ; load indirect add
004D 0101 movwf fsr0 ; for Tone 2
004E E06C call Resonator ; compute next samp

; Compute Tone1 + Tone2 for dual tone

ADD16ACC tonel,tone2, dualTone
; Dual Tone = Tone1

004F 6030  movfp  tone1+B0,wreg
0050 0E39  addwf  tone2+B0,w
0051 013B  movwf  dualTone+B0
0052 6031  movfp  tone1+B1,wreg
0053 103A  addwfc tone2+B1,w
0054 013C  movwf  dualTone+B1

0055 C045  goto  NextSample
0056 C056  self  goto  self

;****************************************************************************************

; Initialization routine :
; Read Tone 1 & Tone 2 Resonator Frequencies from Program M
; Data Memory
; Program Memory : 3 + 8* (# of tones to be gene
; Timing : 4 + 11* (# of tones to be gene

Coeff_Read

TBLADDR coeff_addr

0057 B030  MOVWL  (0x0030)
0058 01D0  MOVWF  tblptrl
0059 B000  MOVWL  page  (0x0030)
005A 010E  MOVWF  tblptrh

005B A929  tablrd  0,1,tone1K1
005C A029  tlr  0,tone1K1
005D AB2A  tablrd  1,1,tone1K1+B1 ; read K1
005E A02E  tlr  0,tone1_1
005F AB2F  tablrd  1,1,tone1_1+B1 ; read K2
0060 A02C  tlr  0,tone1_2
0061 A22D  tlr  1,tone1_2+B1 ; read K3
0062 292B  clr  tone1_2C

0063 A932  tablrd  0,1,tone2K1
0064 A032  tlr  0,tone2K1
0065 AB33  tablrd  1,1,tone2K1+B1 ; read K1
0066 A037  tlr  0,tone2_1
0067 AB38  tablrd  1,1,tone2_1+B1 ; read K2
0068 A035  tlr  0,tone2_2
0069 A236  tlr  1,tone2_2+B1 ; read K3
006A 2934  clr  tone2_2C

006B 0002  return

;****************************************************************************************

; Resonator Subroutine
;
; Before calling this routine, load the indirect register,
; with the starting RAM address of the desired Tone Variable
;
; ( eg. For Tone1 Generation, load FSR0 with "Tone1K1" addr

 © 1993 Microchip Technology Inc.
Timing (worst case):
20 + 36 + (worst case multiplier time)
= 56 + 179 - 235 cycles
= 58.75 uS @ 16Mhz
= 47.00 uS @ 20Mhz
= 37.60 uS @ 25 Mhz

Memory Requirements:
Program Memory : 54 locations
Data Memory : 9 + 9* (# of tones to be gen)

Resonator

Transfer tone variables to resonator’s variables using indirect addressing mode can be used through the code i subroutine, but is less efficient.

MOVFP16 K1, BARG
MOVFP16 SinCos_1, AARG

MOVFP16 K1+B0, BARG+B0; move K1(B0) t
MOVFP16 K1+B1, BARG+B1; move K1(B1) t

MOVFP SinCos_1+B0, AARG+B0; move Si
MOVFP SinCos_1+B1, AARG+B1; move Si

call DblMult; DPX = 2*K1*y(n-1)
BCF _carry
RLCF DPX+B0
007E 1B1D RLCF DPX+B1
007F 1B1E RLCF DPX+B2
0080 1B1F RLCF DPX+B3

; subtract \( y(n-2) \times (2^{15}) \)

0081 2922 clrf SinCos_2C
RRC24 SinCos_2C
0082 1A24 RLCF SinCos_2C+B2,W move sign
0083 1924 RRCF SinCos_2C+B2
0084 1923 RRCF SinCos_2C+B1
0085 1922 RRCF SinCos_2C+B0

SUB24 SinCos_2C,DPX1 ; DPX = 2*K1*y(n-1) - y(n-)

0086 6022 MOVFP SinCos_2C+B0,wreg ; get lowest
0087 051D SUBWF DPX1+B0 ; sub lowest byte
0088 6023 MOVFP SinCos_2C+B1,wreg ; get 2nd b
0089 031E SUBWFB DPX1+B1 ; sub 2nd byte o
008A 6024 MOVFP SinCos_2C+B2,wreg ; get 3rd b
008B 031F SUBWFB DPX1+B2 ; sub 3rd byte o

RLC24 DPX1 ; adjust decimal point
008C 8804 BCF carry
008D 1B1D RLCF DPX1+B0
008E 1B1E RLCF DPX1+B1
008F 1B1F RLCF DPX1+B2

; update past samples with newly computed values

MOVFP16 DPX2,SinCos ; y(n) = 2*K1*y(n-1) - y(n)

0090 5E27 MOVFP DPX2+B0,SinCos+B0 ; move DPX2
0091 5F28 MOVFP DPX2+B1,SinCos+B1 ; move DPX2

MOV16 SinCos_1,SinCos_2 ; y(n-2) = y(n-1)
0092 6025 MOVFP SinCos_1+B0,wreg ; get byte o
0093 0123 MOVWF SinCos_2+B0 ; move to Si
0094 6026 MOVFP SinCos_1+B1,wreg ; get byte o
0095 0124 MOVWF SinCos_2+B1 ; move to Si
Tone Generation

MOVPF16 DPX2, SinCos 1 ; y(n-1) = y(n)

0096 5E25      MOVFP    DPX2+B0, SinCos_1+B0; move DP
0097 5F26

; Generation Of The Next Sample Of The Resonator (sine wave
; The 16 bit result is stored in location "SinCos" (low Byte
; "SinCos+1" (High Byte)
; write back all the computed values to respective tone var

0098 0701      decf    fso
0099 8D04      bcf      _fs1
009A 8C04      bcf      _fs0
009B 6028      movfp   SinCos+Bl, indf0
009C 6027      movfp   SinCos+B0, indf0
009D 6026      movfp   SinCos_1+Bl, indf0
009E 6025      movfp   SinCos_1+B0, indf0
009F 6024      movfp   SinCos_2C+Bl, indf0
00A0 6023      movfp   SinCos_2C+B0, indf0

00A1 0002      ; return

;*******************************************************************
; Include Double Precision Multiplication Routine
;*******************************************************************

0001 SIGNED   equ   TRUE

#include "17c42mpy.mac"

; NOSTD
;*******************************************************************
;
;*******************************************************************

; Double Precision Multiplier For PIC17C42
;
; Dmult
;
; DESCRIPTION:
;
; Multiplication: AARG (16 bits) * BARG (16 bits) -> DPX
;
; (a) Load the 1st operand in locations AARG+B0 & AARG
; (b) Load the 2nd operand in locations BARG+B0 & BARG
; (c) CALL Dmult
; (d) The 32 bit result is in locations ( DPX+B0,DPX+B
;
; In the signed case, a savings of 9 clks can be real
; BARG as the positive factor in the product when pos
;
; TIMING (worst case):
; unsigned: 17
NOTE: Define SIGNED/UNSIGNED To 1/0 before including this file in your program.

Multiplication Macro

; TIMING: unsigned: 11+7*10+8*11 = 169 clks
; (worst case) signed: 11+7*10+7*11+5 = 163 clks

variable i

i = 0
if SIGNED
while i < 15
else
while i < 16
endif
if i < 8 ; test low byte
btfsc BARG+B0,i
else ; test high byte
btfsc BARG+B1,i-8
fi
goto add#v(i)
if i < 8 ; rotate sign into carry bit
rlcf DPX+B3,W
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
else
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0
fi
i = i+1 endw
clrfrf DPX+B0 ; if we get here, BARG = 0 return
add0
movfp AARG+B0,WREG
addwf DFX+B2 ; add lsb
movfp AARG+B1,WREG
addwfc DFX+B3 ; add msb
rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DFX+B3 ; for i < 8, no meaningful bits
rrcf DFX+B2 ; are in DFX+B0
rrcf DFX+B1

i = 1
if SIGNED
  while i < 15
else
  while i < 16
endif
if i < 8
  btfss BARG+B0,i ; test low byte
else
  btfss BARG+B1,i-8 ; test high byte
fi
  goto noadd#v(i)
add#v(i)
movfp AARG+B0,WREG
addwf DFX+B2 ; add lsb
movfp AARG+B1,WREG
addwfc DFX+B3 ; add msb
   noadd#v(i)
if i < 8
  rlcfr AARG+B1,W ; rotate sign into carry bit
  rrcfr DFX+B3 ; for i < 8, no meaningful bits
  rrcfr DFX+B2 ; are in DFX+B0
  rrcfr DFX+B1
else
  rlcfr AARG+B1,W ; rotate sign into carry bit
  rrcfr DFX+B3
  rrcfr DFX+B2
  rrcfr DFX+B1
  rrcfr DFX+B0
fi
i = i+1
endw
if SIGNED
  rlcfr AARG+B1,W ; since BARG is always made posit
  rrcfr DFX+B3 ; the last bit is known to be zero
  rrcfr DFX+B2
  rrcfr DFX+B1
  rrcfr DFX+B0

© 1993 Microchip Technology Inc.
endif
ENDM

; Double Precision Multiply (16x16 -> 32)
; (AARG*BARG -> : 32 bit output in DPX)
; DblMult
if SIGNED

OOA2 971B
btfss BARG+Bl,MSB ; test sign of BARG

OOA3 COAE
NEGI6 AARG+B0
goto argsok ; if positive, ok

OOA4 1318
COMF AARG+B0+B0

OOA5 1319
COMF AARG+B0+B1

OOA6 2900
CLRF wreg

OOA7 1518
INCF AARG+B0+B0

OOA8 1119
ADDWFC AARG+B0+B1

NEGI6 BARG+B0 ; AARG and BARG

OOA9 131A
COMF BARG+B0+B0

OOAA 131B
COMF BARG+B0+B1

OOAB 2900
CLRF wreg

OOAC 151A
INCF BARG+B0+B0

OOAD 111B
ADDWFC BARG+B0+B1

endif

argsok

CLR16 DPX+B2 ; clear initial partial pr

OOAE 291E
CLRF DPX+B2+B0

OOAF 291F
CLRf DPX+B2+B1

MULTMAC ; use macro for multiplication

0000
variable i

0000
i = 0

if SIGNED
while i < 15

else

while i < 16

endif

if i < 8 ; test low byte

btfsc BARG+B0,i

© 1993 Microchip Technology Inc.
else
  btfsc BARG+B1, i-8 ; test high byte
fi

goto addO(v(i))

if i < 8
  rlcf DPX+B3, W ; rotate sign into carry bit
  rrcf DPX+B3 ; for i < 8, no meaningful bits
  rrcf DPX+B2
  rrcf DPX+B1
else
  rlcf DPX+B3, W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0
fi

i = i+1
endw

if i < 8 ; test low byte
  btfsc BARG+B0, i
else ; test high byte
  btfsc BARG+B1, i-8
fi

00B0 981A
  goto add0

if i < 8
  rlcf DPX+B3, W ; rotate sign into
  rrcf DPX+B3 ; for i < 8, no mea
  rrcf DPX+B2
  rrcf DPX+B1
else
  rlcf DPX+B3, W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0
fi

0001
    i = i+1

if i < 8 ; test low byte
    btfsc BARG+B0,i
else ; test high byte
    btfsc BARG+B1,i-8

fi

0002
    i = i+1

if i < 8 ; test low byte
    btfsc BARG+B0,i
else ; test high byte
    btfsc BARG+B1,i-8

fi
Tone Generation

00BD C127

goto add2

if i < 8

00BE 1A1F

rlcf DPX+B3,W ; rotate sign into

00BF 191F

rrcf DPX+B3 ; for i < 8, no mea

00C0 191E
00C1 191D

else

rrcf DPX+B2 ; are in DPX+B0

rrcf DPX+B1

rrcf DPX+B0

fi

0003

i = i+1

if i < 8 ; test low byte

00C2 9B1A

btfsc BARG+B0,i

else ; test high byte

btfsc BARG+B1,i-8

fi

00C3 C131

goto add3

if i < 8

00C4 1A1F

rlcf DPX+B3,W ; rotate sign into

00C5 191F

rrcf DPX+B3 ; for i < 8, no mea

00C6 191E
00C7 191D

else

rlcf DPX+B2 ; are in DPX+B0

rrcf DPX+B1

rrcf DPX+B0

fi
rrcf  DPX+B0

fi

0004  
i = i+1

if i < 8  ; test low byte
00C8 9C1A
    btfsc  BARG+B0,i
    else  ; test high byte
    btfsc  BARG+B1,i-8

fi

00C9 C13B
    goto  add4

if i < 8
00CA 1A1F
    rlcf  DPX+B3,W  ; rotate sign into
00CB 191F
    rrcf  DPX+B3  ; for i < 8, no mea
00CC 191E
00CD 191D
    rrcf  DPX+B2  ; are in DPX+B0
    rrcf  DPX+B1
    rrcf  DPX+B0

else

    rlcf  DPX+B3,W  ; rotate sign into carry bit
    rrcf  DPX+B3
    rrcf  DPX+B2
    rrcf  DPX+B1
    rrcf  DPX+B0

fi

0005  
i = i+1

if i < 8  ; test low byte
00CE 9D1A
    btfsc  BARG+B0,i
    else  ; test high byte
    btfsc  BARG+B1,i-8
Tone Generation

fi

00CF C145
goto add5

if i < 8

00D0 1A1F
rlcf DPX+B3,W ; rotate sign into
00D1 191F
rrcf DPX+B3 ; for i < 8, no mea
00D2 191E
rrcf DPX+B2 ; are in DPX+B0
00D3 191D
rrcf DPX+B1

else

rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

fi

00D6
i = i+1

if i < 8 ; test low byte

00D4 9E1A
btfsc BARG+B0,i
else ; test high byte
btfsc BARG+B1,i-8

fi

00D5 C14F
goto add6

if i < 8

00D6 1A1F
rlcf DPX+B3,W ; rotate sign into
00D7 191F
rrcf DPX+B3 ; for i < 8, no mea
00D8 191E
rrcf DPX+B2 ; are in DPX+B0
00D9 191D
rrcf DPX+B1

else

rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf   DPX+B2
rrcf   DPX+B1
rrcf   DPX+B0

fi

0007
    i = i+1

if i < 8   ; test low byte
    btfsc BARG+B0,i
else   ; test high byte
    btfsc BARG+B1,i=8

fi

0008
    i = i+1

if i < 8   ; test low byte
    btfsc BARG+B0,i
else ; test high byte

00E0 981B
   btfsc   BARG+B1,i-8
fi

00E1 C163
   goto   add8

if i < 8

   rlcf   DPX+B3,W ; rotate sign into carry bit
   rrcf   DPX+B3 ; for i < 8, no meaningful bits
   rrcf   DPX+B2 ; are in DPX+B0
   rrcf   DPX+B1

else

00E2 1A1F
   rlcf   DPX+B3,W ; rotate sign into
00E3 191F
   rrcf   DPX+B3
00E4 191E
   rrcf   DPX+B2
00E5 191D
   rrcf   DPX+B1
00E6 191C
   rrcf   DPX+B0
fi

0009
   i = i+1

if i < 8 ; test low byte

   btfsc   BARG+B0,i

else ; test high byte

00E7 991B
   btfsc   BARG+B1,i-8
fi

00E8 C16E
   goto   add9

if i < 8

   rlcf   DPX+B3,W ; rotate sign into carry bit
   rrcf   DPX+B3 ; for i < 8, no meaningful bits
   rrcf   DPX+B2 ; are in DPX+B0
   rrcf   DPX+B1

else
Tone Generation

```
00E9 1A1F  rlcf DPX+B3,W ; rotate sign into
00EA 191F  rrcf DPX+B3
00EB 191E  rrcf DPX+B2
00EC 191D  rrcf DPX+B1
00ED 191C  rrcf DPX+B0

fi

000A  i = i+1

if i < 8 ; test low byte
  btfs BARG+B0,i

else ; test high byte
  btfs BARG+B1,i-8
fi

00EE 9A1B

00EF C179  goto add10

if i < 8
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B3 ; for i < 8, no meaningful bits
  rrcf DPX+B2 ; are in DPX+B0
  rrcf DPX+B1

else

00F0 1A1F  rlcf DPX+B3,W ; rotate sign into
00F1 191F  rrcf DPX+B3
00F2 191E  rrcf DPX+B2
00F3 191D  rrcf DPX+B1
00F4 191C  rrcf DPX+B0

fi

000B  i = i+1

if i < 8 ; test low byte
  btfs BARG+B0,i

else ; test high byte
```
Tone Generation

00F5 9B1B
   btfusc BARG+B1,i-8
   fi

00F6 C184
   goto add11
   if i < 8
      rlcf DPX+B3,W ; rotate sign into carry bit
      rrcf DPX+B3 ; for i < 8, no meaningful bits
      rrcf DPX+B2 ; are in DPX+B0
      rrcf DPX+B1
   else
      rlcf DPX+B3,W ; rotate sign into

00F7 1A1F
      rlcf DPX+B3
      rrcf DPX+B3
      rrcf DPX+B2
      rrcf DPX+B1
      rrcf DPX+B0
   fi

000C
   i = i+1
   if i < 8 ; test low byte
      btfusc BARG+B0,i
   else ; test high byte
      btfsc BARG+Bl,i-8
   fi

00FD C18F
   goto add12
   if i < 8
      rlcf DPX+B3,W ; rotate sign into carry bit
      rrcf DPX+B3 ; for i < 8, no meaningful bits
      rrcf DPX+B2 ; are in DPX+B0
      rrcf DPX+B1
   else
      rlcf DPX+B3,W ; rotate sign into
Tone Generation

00FF 191F
0100 191E
0101 191D
0102 191C

rrcf   DPX+B3
rrcf   DPX+B2
rrcf   DPX+B1
rrcf   DPX+B0

fi

000D

i = i+1

if i < 8 ; test low byte

btfsc   BARG+B0,i

else ; test high byte

0103 9D1B

btfsc   BARG+B1,i-8

fi

0104 C19A

goto   add13

if i < 8

rlcf   DPX+B3,W ; rotate sign into carry bit
rrcf   DPX+B3 ; for i < 8, no meaningful bits
rrcf   DPX+B2 ; are in DPX+B0
rrcf   DPX+B1

else

0105 1A1F

rlcf   DPX+B3,W ; rotate sign into

0106 191F
0107 191E
0108 191D
0109 191C

rrcf   DPX+B3
rrcf   DPX+B2
rrcf   DPX+B1
rrcf   DPX+B0

fi

000E

i = i+1

if i < 8 ; test low byte

btfsc   BARG+B0,i

else ; test high byte

010A 9E1B

btfsc   BARG+B1,i-8

fi
Tone Generation

010B CIA5

go to add14

if i < 8

rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful bits
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

010C 1A1F

rlcf DPX+B3,W ; rotate sign into
010D 191F
010E 191E
010F 191D
0110 191C

fi

000F
i = i+1

0111 291C
clr if DPX+B0 ; if we get here, B
0112 0002 return

add0

0113 6018
0114 0F1E
0115 6019
0116 111F
0117 1A19

movf AARG+B0,WREG
addwf DPX+B2 ;add lsb
movf AARG+B1,WREG
addwfc DPX+B3 ;add msb
rlcf AARG+B1,W ; rotate sign into

0118 191F
rrcf DPX+B3 ; for i < 8, no mea
0119 191E
011A 191D

i = 1

if SIGNED

while i < 15

else

while i < 16

endif

if i < 8

btfss BARG+B0,i ;test low byte

else
btfss BARG+B1,i-8 ; test high byte

fi

goto noadd\#v(i)
add\#v(i)

movfp AARG+B0,WREG
addwf DPX+B2 ; add lsb
movfp AARG+B1,WREG
addwfc DPX+B3 ; add msb

noadd\#v(i)

if i < 8
  rlcf AARG+B1,W ; rotate sign into carry bit
  rrcf DPX+B3 ; for i < 8, no meaningful bits
  rrcf DPX+B2
  rrcf DPX+B1
else
  rlcf AARG+B1,W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0
fi

i = i+1
endw

if i < 8 ; test low byte
  btfss BARG+B0,i
else ; test high byte
  btfss BARG+B1,i-8
fi

011B 911A

011C C121

add1 goto noadd1

011D 6018
011E 0F1E
011F 6019
0120 111F

noadd1

if i < 8
  rlcf AARG+B1,W ; rotate sign into
  rrcf DPX+B3 ; for i < 8, no mea
  rrcf DPX+B2
  rrcf DPX+B1
else
  rrcf DPX+B2 ; are in DPX+B0
  rrcf DPX+B1

© 1993 Microchip Technology Inc.
Tone Generation

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

fi

0002
i = i+1

if i < 8 ;test low byte
0125 921A
btfss BARG+B0,i
else ; test high byte
btfss BARG+B1,i-8

fi

0126 C12B
goto noadd2
0127 6018
0128 0F1E
0129 6019
012A 111F
add2
movfp AARG+B0,WREG
addwf DPX+B2 ;add lsb
movfp AARG+B1,WREG
addwf DPX+B3 ;add msb
noadd2

if i < 8
012B 1A19
rlcf AARG+B1,W ; rotate sign into
012C 191F
rrcf DPX+B3 ; for i < 8, no mea
012D 191E
rrcf DPX+B2 ; are in DPX+B0
012E 191D
rrcf DPX+B1
else
rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

fi
Tone Generation

0003
i = i+1

if i < 8
   ; test low byte
012F 931A
   btfss  BARG+B0,i
else
   ; test high byte
012F 931A
   btfss  BARG+B1,i-8
fi

0130 Cl35
add3
0131 6018
0132 0F1E
0133 6019
0134 111F
noadd3

0135 1A19
   rlcf  AARG+B1,W ; rotate sign into
0136 191F
   rrcf  DPX+B3 ; for i < 8, no mea
0137 191E
   rrcf  DPX+B2 ; are in DPX+B0
0138 191D
   rrcf  DPX+B1
else
   rlcf  AARG+B1,W ; rotate sign into carry bit
   rrcf  DPX+B3
   rrcf  DPX+B2
   rrcf  DPX+B1
   rrcf  DPX+B0
fi

0004
i = i+1

if i < 8
   ; test low byte
0139 941A
   btfss  BARG+B0,i
else
   ; test high byte
0139 941A
   btfss  BARG+B1,i-8

Tone Generation

```
fi

013A C13F   goto noadd4
013B 6018
013C 0F1E
013D 6019
013E 111F

add4

movfp AARG+B0, WREG
addwf DPX+B2 ; add lsb
movfp AARG+B1, WREG
addwf DPX+B3 ; add msb

noadd4

if i < 8

013F 1A19

rlcf AARG+B1, W ; rotate sign into
0140 191F
rrcf DPX+B3 ; for i < 8, no mea
0141 191E
rrcf DPX+B2 ; are in DPX+B0
0142 191D
rrcf DPX+B1
else

rlcf AARG+B1, W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

fi

0005

i = i+1

if i < 8 ; test low byte
0143 951A
btfss BARG+B0, i
else ; test high byte

btfss BARG+B1, i-8

fi

0144 C149   goto noadd5
0145 6018
0146 0F1E
0147 6019
0148 111F

add5

movfp AARG+B0, WREG
addwf DPX+B2 ; add lsb
movfp AARG+B1, WREG
addwf DPX+B3 ; add msb

noadd5

if i < 8

0149 1A19

rlcf AARG+B1, W ; rotate sign into
014A 191F
rrcf DPX+B3 ; for i < 8, no mea
```
014B 191E
014C 191D

else

rlcf AARG+Bl,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

fi

0006

i = i+1

if i < 8 ;test low byte
014D 961A

btfss BARG+B0,i
else test high byte

btfss BARG+B1,i-8

fi

014E C153

add6 goto noadd6
014F 6018
0150 0F1E
0151 6019
0152 111F

noadd6

if i < 8
0153 1A19

rlcf AARG+Bl,W ; rotate sign into
0154 191F

rrcf DPX+B3 ; for i < 8, no mea
0155 191E
0156 191D

else

rlcf AARG+Bl,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0
fi

0007

    i = i+1

if i < 8 ; test low byte

0157 971A

    btfss BARG+B0,i

else ; test high byte

    btfss BARG+B1,i-8

fi

0158 C15D

add7 goto noadd7

0159 6018 movfp AARG+B0,WREG

015A 0F1E addwf DPX+B2 ; add lsb

015B 6019 movfp AARG+B1,WREG

015C 111F addwf DPX+B3 ; add msb

noadd7

if i < 8

015D 1A19 rlcf AARG+B1,W ; rotate sign into

015E 191F rrcf DPX+B3 ; for i < 8, no mea

015F 191E rrcf DPX+B2 ; are in DPX+B0

0160 191D rrcf DPX+B1

else

    rlcf AARG+B1,W ; rotate sign into carry bit
    rrcf DPX+B3
    rrcf DPX+B2
    rrcf DPX+B1
    rrcf DPX+B0

fi

0008

    i = i+1

if i < 8 ; test low byte

    btfss BARG+B0,i
else ; test high byte

0161 901B
btfss BARG+B1, i-8
fi

0162 C167
goto noadd8

0163 6018
add8
movfp AARG+B0, WREG
0164 0F1E
addwf DPX+B2 ; add lsb
0165 6019
movfp AARG+B1, WREG
0166 111F
addwf DPX+B3 ; add msb

noadd8
if i < 8

rlcf AARG+B1, W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful bits
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

0167 1A19
rlcf AARG+B1, W ; rotate sign into
0168 191F
rrcf DPX+B3
0169 191E
rrcf DPX+B2
016A 1910
rrcf DPX+B1
016B 191C
rrcf DPX+B0
fi

0009
i = i+1

if i < 8 ; test low byte

btfss BARG+B0, i

else ; test high byte

016C 911B
btfss BARG+B1, i-8
fi

016D C172
goto noadd9

016E 6018
add9
movfp AARG+B0, WREG
016F 0F1E
addwf DPX+B2 ; add lsb
0170 6019
movfp AARG+B1, WREG
0171 111F
addwf DPX+B3 ; add msb

noadd9
if i < 8
Tone Generation

```
rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DFX+B3 ; for i < 8, no meaningful bits
rrcf DFX+B2 ; are in DFX+B0
rrcf DFX+B1
else
    rlcf AARG+B1,W ; rotate sign into
    rrcf DFX+B3
    rrcf DFX+B2
    rrcf DFX+B1
    rrcf DFX+B0
    fi
i = i+1
if i < 8 ;test low byte
    btfss BARG+B0,i
else ; test high byte
    btfss BARG+B1,i-8
fi
0178 C17D
add10 goto noadd10
0179 6018
017A 0F1E
017B 6019
017C 111F
noadd10
if i < 8
    rlcf AARG+B1,W ; rotate sign into carry bit
    rrcf DFX+B3 ; for i < 8, no meaningful bits
    rrcf DFX+B2 ; are in DFX+B0
    rrcf DFX+B1
else
    rlcf AARG+B1,W ; rotate sign into
    rrcf DFX+B3
    rrcf DFX+B2
```
Tone Generation

0180 191D
0181 191C

rrcf  DPX+B1
rrcf  DPX+B0

fi

000B

i = i+1

if i < 8
  ;test low byte
  btfss  BARG+B0,i

else
  ; test high byte

0182 931B

btfss  BARG+B1,i-8

fi

0183 C188

add1  goto  noadd1

0184 6018
0185 0F1E
0186 6019
0187 111F

movfp  AARG+B0,WREG
addwf  DPX+B2   ;add lsb
movfp  AARG+B1,WREG
addwfc DPX+B3   ;add msb

noadd1

if i < 8
  rlcf  AARG+B1,W   ; rotate sign into carry bit
  rrcf  DPX+B3    ; for i < 8, no meaningful bits
  rrcf  DPX+B2    ; are in DPX+B0
  rrcf  DPX+B1

else

0188 1A19

rlcf  AARG+B1,W   ; rotate sign into

0189 191F
018A 191E
018B 191D
018C 191C

rrcf  DPX+B3
rrcf  DPX+B2
rrcf  DPX+B1
rrcf  DPX+B0

fi

000C

i = i+1

if i < 8
  ;test low byte
  btfss  BARG+B0,i

else
  ; test high byte
018D 941B

btfss BARG+Bl,i-8

fi

018E C193

add12
goto noadd12

0190 6018
0190 0F1E
0191 6019
0192 111F

noadd12

if i < 8

rlcf AARG+Bl,W ; rotate sign into carry bit

rrcf DPX+B3 ; for i < 8, no meaningful bits

rrcf DPX+B2 ; are in DPX+B0

rrcf DPX+B1

else

0193 1A19

rlcf AARG+Bl,W ; rotate sign into

0194 191F
0195 191E
0196 191D
0197 191C

fi

000D

i = i+1

if i < 8 ; test low byte

btfss BARG+B0,i

else ; test high byte

0198 951B

btfss BARG+Bl,i-8

fi

0199 C19E

add13
goto noadd13

019A 6018
019B 0F1E
019C 6019
019D 111F

noadd13

if i < 8

rlcf AARG+Bl,W ; rotate sign into carry bit
Tone Generation

```assembly
rrcf   DPX+B3 ; for i < 8, no meaningful bits
rrcf   DPX+B2 ; are in DPX+B0
rrcf   DPX+B1
else

019E 1A19
     rlcf   AARG+B1,W ; rotate sign into
019F 191F
01A0 191E
01A1 191D
01A2 191C
     fi
000E
     i = i+1

if i < 8 ; test low byte

     btfss   BARG+B0,i
else ; test high byte

01A3 961B
     btfss   BARG+B1,i-8
fi
01A4 C1A9
     add14  goto   noadd14
01A5 6018
01A6 091E
01A7 6019
01A8 111F
     addwfc  DPX+B3 ; add msb
noadd14
if i < 8

     rlcf   AARG+B1,W ; rotate sign into carry bit
     rrcf   DPX+B3 ; for i < 8, no meaningful bits
     rrcf   DPX+B2 ; are in DPX+B0
     rrcf   DPX+B1
else

01A9 1A19
     rlcf   AARG+B1,W ; rotate sign into
01AA 191F
01AB 191E
01AC 191D
01AD 191C
     rrcf   DPX+B3
     rrcf   DPX+B2
     rrcf   DPX+B1
     rrcf   DPX+B0
```

© 1993 Microchip Technology Inc.
fi

000F    i = i+1

if  SIGNED

01AE 1A19  rlcf  AARG+B1,W  ; since BARG is alw
01AF 191F  rrcf  DPX+B3  ; the last bit is k
01B0 191E  rrcf  DPX+B2
01B1 191D  rrcf  DPX+B1
01B2 191C  rrcf  DPX+B0

endif

01B3 0002  return
;
;***********************************************************************

END

Errors :  0
Warnings :  0
INTRODUCTION

The PIC17C42 microcontroller is an excellent choice for cost-effective servo control in embedded applications. Due to its Harvard architecture and RISC-like features, the PIC17C42 offers excellent computation speed needed for real time closed loop servo control. This application note examines the use of the PIC17C42 as a DC brush motor servo controller. It is shown that a PID (Proportional, Integral, Differential) control calculation can be performed in less than 200 µS (@16 MHz) allowing control loop sample times in the 2 KHz range. Encoder rates up to 3 MHz are easily handled by the PIC17C42's high speed peripherals. Further, the on-chip peripherals of the PIC17C42 allow an absolute minimum cost system to be constructed.

Closed-loop servo motor control is usually handled by 16-bit, high-end microcontrollers and external logic. In an attempt to increase performance many applications are upgrading to DSPs. However, the very high performance of the PIC17C42 makes it possible to implement these servo control applications at a significant reduction in overall system cost.

The servo system uses a PIC17C42 microcontroller, a programmable logic device (PLD), and a single-chip H-bridge driver. Such a system might be used as a positioning controller in a printer, plotter, or scanner. The low cost of implementing a servo control system using the PIC17C42 allows this system to compete favorably with stepper motor systems offering a number of advantages:

- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

SYSTEM OVERVIEW

DC Servo Control

Modern digital servo systems are formed as shown in Figure 1. These systems control a motor with an incremental feedback device known as a sequential encoder. They consist of an encoder counter, a processor, some form of digital-to-analog converter, and a power amplifier, which delivers current or voltage to the motor.

The PIC17C42 implements both the servo compensator algorithm and the trajectory profile (trapezoidal) generation. A trajectory generation algorithm is necessary for optimum motion and its implementation is as important as the servo compensator itself. The servo compensator can be implemented as a traditional digital filter, a fuzzy logic algorithm, or the simple PID algorithm (implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The digital-to-analog conversion can be handled by a conventional DAC or by using pulse-width modulation (PWM). In either case the output signal is fed to a power stage which translates the analog signal(s) into usable voltages and currents to drive the motor.

PWM output can be a duty-cycle signal in combination with a direction signal or a single signal which carries both pieces of information. In the latter case a 50% duty cycle commands a null output, a 0% duty cycle commands maximum negative output, and 100% maximum positive output.

The amplifier can be configured to supply a controlled voltage or current to the motor. Most embedded systems use voltage output because of its simplicity and reduced cost.

Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of D1 and D2 represents an increment of position. The phase of the signals is used to determine direction of rotation.
Servo Control of a DC-Brush Motor

FIGURE 2 - THE PIC17C42 SERVO SYSTEM

FIGURE 3 - THE PIC17C42 BASED SERVO CONTROL BOARD

FIGURE 4 - DIGITAL PID IMPLEMENTATION

Note: The Z-1 operator indicates a one sample time delay
These encoder signals are usually decoded using a small state machine into Count Up and Count Down pulses. These pulses are then routed to an N-bit, up-down counter whose value corresponds to the position of the motor shaft. The decoder/counter may be implemented in hardware, software, or a combination of the two.

The PIC17C42 Based Motor Control Board

The PIC17C42 based servo system described here has a full RS-232 ASCII interface, on-board switching power supply, H-bridge motor drive, over-current protection, limit switch inputs and digital I/O. The entire system measures 5" x 3.5" and is shown in Figure 3. The system can be used to evaluate the PIC17C42 in servo applications. All unused PIC17C42 pins are available at an I/O connector for prototyping.

A PID algorithm is used as a servo compensator and position trajectories are derived from linear velocity ramp segments. This system uses 50%-null PWM as the digital-to-analog conversion technique. The power stage is a high current output switching stage which steps-up the level of the PWM signal. Encoder signal decoding is accomplished using an external PLO. The up/down counter is implemented internally in the PIC17C42 as combination of hardware and software (Figures 5 and 6).

THE COMPENSATOR

PID is the most widely used algorithm for servo motor control. Although it may not be the most optimum controller for all applications, however it is easy to understand and tune.

The standard digital PID algorithm's form is shown in Figure 4. U(k) is the position or velocity error and Y(k) is the output.

This algorithm has been implemented using the PIC17C42 math library. Only 800 instruction cycles are required resulting in a 0.2mS PID execution time at 16 MHz.

Integrator wind-up is a condition which occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator continually builds up during this following error condition even though the output is saturated. The integrator then "unwinds" when the servo system reaches its final destination causing excessive oscillation. The PID implementation shown above avoids this problem by stopping the action of the integrator during output saturation.

MOTOR ACTUATION

The PIC17C42 contains a high-resolution pulse width modulation (PWM) subsystem. This forms a very efficient power D/A converter when coupled to a simple switching power stage. The resolution of the PIC17C42 PWM subsystem is 62.5nS (at 16 MHz). This translates into 10-bit resolution at a 15.6kHz rate or 1 part in 800 (9 1/2-bit) resolution at 20kHz. This allows effective voltage control while still maintaining the modulation frequency at or above the limit of human hearing. This is especially relevant in office automation equipment where minimizing noise is a design goal.

The motor responds to a PWM output stage by time averaging the duty cycle of the output. Most motors react slowly, having an electrical time constant of 0.5mS or more and a mechanical time constant of 20.0mS or more. A 15KHz PWM output is effectively equivalent to that of a linear amplifier.

In the system shown in Figure 2, the H-bridge's direction input is wired directly to the PIC17C42's PWM output. The H-bridge is powered by a DC supply voltage, $V_m$. In this configuration 0 volts is presented to the motor when the PWM signal is at a 50% duty cycle, $-V_m$ volts at 0% duty cycle and $+V_m$ volts at 100% duty cycle.

ENCODER FEEDBACK

Position feedback for the example system is derived from a quadrature encoder mounted on the motor shaft. Both incremental position and direction can be derived from this inexpensive device. The quadrature encoder signals are processed by a 16R8-type PLD device as shown in Figure 2. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 5). These signals are then fed to two 16-bit timers of the PIC17C42 (TMR3 and RTCC). A logic description for the PLD decoder is shown in Appendix A.

The PIC17C42 keeps track of the motor shaft's incremental position by differencing these two 16-bit timers. This operation is performed each servo sample time and the current position is calculated by adding the incremental position to the previous position. Since both timers are 16-bits deep, keeping track of the overflow is unnecessary, unless the encoder signals frequency is greater than 32767 times the sample frequency. For example, at a servo sample time of 1mS, the maximum encoder rate would be 3.2767 MHz.

Counter wrap-around is not a concern because only the difference between the two counters is used. Two's-complement subtraction takes care of this automatically. Position is maintained as a three-byte, 24-bit quantity in the example program shown in Appendix F. However, there is no limit to the size of the internal position register. By adding the 16-bit incremental position each sample time to an N-byte software register, an N-byte position may be maintained.
**Servo Control of a DC-Brush Motor**

**FIGURE 5 - SEQUENTIAL ENCODER SIGNALS**

![Sequential Encoder Signals Diagram]

**FIGURE 6 - ENCODER INTERFACE SCHEME**

![Encoder Interface Scheme Diagram]

**TRAJECTORY GENERATION**

A trajectory generation algorithm is essential for optimum motion control. A linear piecewise velocity trajectory is implemented in this application. For a position move, the velocity is incremented by a constant acceleration value until a specified maximum velocity is reached. The maximum velocity is maintained for a required amount of time and then decremented by the same acceleration (deceleration) value until zero velocity is attained. The velocity trajectory is therefore trapezoidal for a long move and triangular short move where maximum velocity was not reached (Figure 8).

The doPreMove subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The doMove routine is then invoked at every sample time to calculate new “desired” velocity and position values as follows:

\[
V_k = V_{k-1} + A \quad (A = \text{Acceleration})
\]

\[
P_k = P_{k-1} + V_k - 1 + A/2
\]

For more details on trajectory generation, see Appendix E.

**IMPLEMENTATION DETAILS**

The program structure is straightforward: An interrupt service routine (ISR) processes the servo control and trajectory generation calculations, and a foreground loop is used to implement the user interface, serial communication and any exception processing (i.e. limit switches, watchdog timer, etc.).

The ISR has a simple structure. In order to effect servo control we need to read the encoder, calculate the new trajectory point and PID values, and set the output of the PWM at a constant, predefined rate. The ISR is initiated by a hardware timer (TMR2) on the PIC17C42. To make sure that the servo calculation always occurs synchronously with the PWM subsystem, the PWM2 output is wired to the input pin of TMR12 (TMR1 in internally-clocked, 8-bit timer mode; TMR2 in externally-clocked, 8-bit counter mode). N is loaded into the PR2 register. The sample rate then becomes the PWM rate divided by N. In this implementation N=16 (Figure 8).
Servo Control of a DC-Brush Motor

FIGURE 7 - SAMPLING SCHEME

![Sampling Scheme Diagram]

- Timer 1 <8>
- PWM1
- PWM2
- Timer 2 <8>
- Comparator
- Period-reg 2

- Servo-update interrupt 0.9765625 KHz
- 16 PWM cycles = 1.024ms
- Servo-update interrupt = 64 µs

- PWM output 15.625 KHz
- PIC17C42

- Period-reg 2 = 16

FIGURE 8 - VELOCITY RAMP SEGMENTS FOR POSITION MOVES

![Velocity Ramp Diagram]

- Velocity
- Short Move
- Long Move
- Slope = Accel. limit
- Velocity Limit
- Time
Servo Control of a DC-Brush Motor

FIGURE 9 - FLOWCHART FOR FOREGROUND PROCESSING

DCMOTOR.ASM

Program Setup

IdleFunction

GetChk

Got a char? No

Get Command

In list? Yes

No

Next Command

End of list? Yes

Send ERROR message

Send response message

Execute Command Function

FIGURE 10 - FLOWCHART FOR INTERRUPT SERVICE ROUTINE

IntPoll

Save Registers

doMposMvel
  • Read up_count and down_count
  • Measure current velocity, position

doExtStat
  • Monitor ext status inputs

Move Running?

No

New Move?

Yes

doPreMove

Yes

doMove

Trajectory in Progress?

Yes

doError
  • Compute position and velocity error

Servo On?

Yes

doServo
  • Compute PID
  • Compute PWM value
  • Write PWM value

CAPFLAG?

Yes

doCapture Regs
  • For PICMASTER based debug
  • Output position, velocity, etc. info.

No

Restore Registers

RETURN
The following events must occur in the interrupt service routine:

- Read Timers (RTCC & TMR3)
- Calculate the new Reference Position using the Trajectory Generation Routine.
- Calculate Error: \( U(k) = \text{Reference Position} - \text{Current Position} \)
- Calculate \( Y(k) \) using PID
- Set PWM output
- Manage other housekeeping tasks (i.e. service serial characters)

The entire ISR requires only 0.250mS to execute with 16MHz processor clock frequency.

**COMMAND INTERFACE**

The following commands are implemented and recognized by the user interface in the foreground loop.

**Move (Value):** M, [-8,388,608, 10 to 8,388,607, 10]

Commands the axis to move to a new position or velocity. Position data is relative, velocity data is absolute. Position data is in encoder counts. Velocity data is given in encoder counts per sample time multiplied by 256. All moves are performed by the controller such that velocity and acceleration limits set into parameter memory will not be violated.

All move commands are kept in a one deep FIFO buffer. The command in the buffer is executed as soon as the executing command is complete. If no move is currently executing the commanded move will start immediately.

**Mode:** Q, (Type), [P, V, T]

An argument of "P" will cause all subsequent move commands to be incremental position moves. A "V" argument will cause all subsequent moves to be absolute velocity moves. A "T" argument sets a "Torque mode" where all subsequent M commands directly write to the PWM. This is useful for debug purposes.

**Set Parameter:** S. (#, Value) [00h to FFh, -8,388,608, 10 to 8,388,607, 10]

Sets controller parameters to the value given. Parameters are shown in Table 1.

**Read Parameter:** B. (#) [00h to FFh]

Returns the present value of a parameter.

**Shutter:** C

Returns the time (in sample time counts 0 to 65,536, 10) since the start of the present move and captures the commanded and actual values of position and velocity at the time of the command.

**Read commanded position:** P

Returns the commanded position count which was captured during the last Shutter command. Range: -8,388,608, 10 to 8,388,607, 10.

**Read commanded velocity:** Y

Returns the commanded velocity multiplied by 256 which was captured during the last Shutter command. Range: -8,388,608, 10 to 8,388,607, 10.

**Read actual position:** p

Returns the actual position count which was captured during the last Shutter command.

Range: -8,388,608, 10 to 8,388,607, 10.

**Read actual velocity:** y

Returns the actual velocity multiplied by 256 which was captured during the last Shutter command.

Range: -8,388,608, 10 to 8,388,607, 10.

**External Status:** X

Returns a two digit hex number which defines the state of the bits in the external status register. Issuing this command will clear all the bits in the external status register unless the event which set the bit is still true. The bits are defined in Table 2.

**Move Status:** Y

Returns a two-digit hex number which defines the state of the bits in the move status register. Issuing this command will clear all the bits in the move status register unless the event which set the bit is still true. The bits are defined in Table 3.

---

**TABLE 1 - PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>#</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity Limit</td>
<td>00</td>
<td>0 to 8,388,607, 10 *</td>
</tr>
<tr>
<td>Acceleration Limit</td>
<td>01</td>
<td>0 to 8,388,607, 10 **</td>
</tr>
<tr>
<td>Kp: Proportional Gain</td>
<td>02</td>
<td>-32768, 10 to 32767, 10</td>
</tr>
<tr>
<td>Kd: Differential Gain</td>
<td>03</td>
<td>-32768, 10 to 32767, 10</td>
</tr>
<tr>
<td>Ki: Integral Gain</td>
<td>04</td>
<td>-32768, 10 to 32767, 10</td>
</tr>
</tbody>
</table>

* (counts per sample time multiplied by 256)

** (counts per sample time per sample time multiplied by 256)

---

**TABLE 2 - EXTERNAL STATUS REGISTER BITS**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>index marker detected</td>
</tr>
<tr>
<td>6</td>
<td>+limit reached</td>
</tr>
<tr>
<td>5</td>
<td>-limit reached</td>
</tr>
<tr>
<td>4</td>
<td>input true</td>
</tr>
<tr>
<td>7</td>
<td>n/a</td>
</tr>
</tbody>
</table>

---

**TABLE 3 - MOVE STATUS BITS**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>move buffer empty</td>
</tr>
<tr>
<td>6</td>
<td>move complete</td>
</tr>
<tr>
<td>5-0</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Servo Control of a DC-Brush Motor

Read Index position: I
Returns the last index position captured in position counts.

Set Position (Value): H, [-8,388,608,10 to 8,388,607,10]
Sets the actual and commanded positions to the value given. Should not be sent unless the move FIFO buffer is empty.
Reset: Z
Performs a software reset.

Capture Servo-Response: c (#Count)
The c command will set a flag inside indicating that starting with the next M (servo move) command, velocity and position information will be sent out (by invoking the doCaptureRegs procedure) during every servo-loop for #count times. At the end of the #count, the processor will halt (see doCaptureRegs procedure). This is useful for debug purposes.

Disable Servo: s
This command disables servo actuation. The servo will activate again with the execution of the next M (move) command. This is useful for debug purposes.
Examples:
Z ;Reset software (No <CR> required)
OV ;Set velocity servo mode (No <CR> ;required)
M 1000<CR> ;Set velocity to 1000
M-1000<CR> ;Set velocity to 1000 in reverse ;direction

OPTIMIZING THE SYSTEM
Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER In-Circuit Emulator for the PIC17C42.
The PICMASTER is a highly sophisticated real-time in-circuit emulator with unlimited break-point capability, 8K deep trace buffer and external logic probes. It's user interface software runs under Windows™ 3.1 with pull-down menus and on-line help. The PICMASTER software also support dynamic data exchange (DDE) through which it possible to send its trace buffer information to a spreadsheet, such as EXCEL, also running under windows.

To tune the PID, first a small amount of diagnostics code is added in the servo routine (doCaptureRegs). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using TABLWT instruction. These are captured in the trace buffer of the emulator. The 'trace' condition is set up to only trace the data cycles of the 2-cycle TABLWT instructions. Next, the trace buffer is transferred to EXCEL and the various

FIGURE 11 - TYPICAL SERVO RESPONSE:

DESIRED/ACTUAL POSITION

Position Response

Position

Kp = 2048
Kd = 20480
Ki = 1024
Actual Desired

Time (mSec)

Kp = 2048
Kd = 20480
Ki = 1024

DESIRED/ACTUAL VELOCITY

Velocity Response

Velocity

Kp = 2048
Kd = 20480
Ki = 1024
Actual Desired

Time (mSec)

Kp = 2048
Kd = 20480
Ki = 1024

VELOCITY ERROR

Velocity Error

Error

Kp = 2048
Kd = 20480
Ki = 1024

Time (mSec)

Kp = 2048
Kd = 20480
Ki = 1024

Velocity = counts/sample

Velocity = counts/sample

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

parameters are plotted. The plots graphically show the amounts of overshoot, ripple and response time. By altering \( K_p \), \( K_i \) and \( K_d \), and plotting the results, the system can be fine tuned.

Under Windows multitasking environment, using PICMASTER emulator this can be done in real time as described below.

Three sessions are set up under Windows:
1. A terminal emulator session to send commands to the motor control board. The "terminal" program provided with Windows is used, although any communications software such as PROCOMM will work.
2. Second, a PICMASTER emulation session is invoked. The actual PIC17C42 is replaced in-circuit by the emulator probe. Within the emulator, trace points are set up to capture the actual and desired position and velocity values on appropriate bus cycles.
3. Third, a session of EXCEL is started and dynamically linked to the PICMASTER sessions such that whenever the trace buffer is full, the data is sent over to EXCEL. A few simple filtering commands in EXCEL are used to separate the various data types, i.e. actual position data from desired position from actual velocity etc. Next, various plot windows are set up within EXCEL to plot these information.

Once these setup have been done, for every servo move, the responses are automatically plotted. It is then a simple matter of varying the PID coefficients and observing the responses to achieve the desired system response. At any point, the responses can be stored in files and/or printed out.

Except for very long "move" commands, most position and velocity commands are executed (i.e. system settled) in less than 500 samples, making it possible to capture all variables (actual and desired position and velocity, and position errors and servo output) in PICMASTER's 8K trace buffer.

CONCLUSIONS

Using a high-performance 8-bit microcontroller as the heart of a servo control system is a cost-effective solution which requires very few external components. A comparison with a popular dedicated servo-control chip, is presented in Table 4.

<table>
<thead>
<tr>
<th>Max Encoder Rate</th>
<th>LM629 @8MHz</th>
<th>PIC17C42 @16MHz</th>
<th>PIC17C42 @25MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Servo Update Time</td>
<td>-</td>
<td>0.25 ms</td>
<td>0.16 ms</td>
</tr>
<tr>
<td>Max Sampling Frequency</td>
<td>4 KHz</td>
<td>2-3 KHz</td>
<td>4-5 KHz</td>
</tr>
</tbody>
</table>

Also apparent in the comparison table is the additional processing power available when using the microcontroller. This processing can be used to provide a user interface, handle other I/O, etc. Alternatively, the additional processing time might be used to improve the performance of compensator and trajectory generation algorithms. A further advantage is that for many embedded applications using motor control the microcontroller proves to be a complete, minimum cost solution.

Credit

This application note and a working demo board has been developed by Teknic Inc. Teknic (Rochester, N.Y.) specializes in Motor Control Systems.

References

Windows is a trademark of Microsoft Corporation.

© 1993 Microchip Technology Inc.
APPENDIX A (CONT.): SCHEMATIC DIAGRAM

PIC17C42 Based DC Servo Control System
Sheet 3 of 3
Power Supply and Output

Microchip Technology Incorporated
APPENDIX B: ENCODER PLD EQUATIONS

Combination quadrature decoder and input synchronizer. This design allows 1x decoding or 4x decoding based on the X4 pin.

* Ver 1.0 - November 8, 1991

MODULE QuadDivider;
TITLE QuadDivider V1.0;
COMMENT Device: 16R8;

TYPE M9I 16R8;

INPUTS;
RESET NODE[PIN2] INVERTED;
X4 NODE[PIN3];
P0 NODE[PIN4];
P90 NODE[PIN5];
INDEX NODE[PIN6];
S2 NODE[PIN12];
S4 NODE[PIN13];
P90D NODE[PIN14];
P90D NODE[PIN15];
CntUp NODE[PIN18];
CntDn NODE[PIN19];
UP NODE[PIN16];
COUNT NODE[PIN17] INVERTED;

OUTPUTS;
S2 NODE[PIN12];
S4 NODE[PIN13];
P90D NODE[PIN14];
P90D NODE[PIN15];
CntUp NODE[PIN18];
CntDn NODE[PIN19];
UP NODE[PIN16];
COUNT NODE[PIN17] INVERTED;

TABLE;
S2 := P90D & !RESET;
S4 := P90D & !RESET;
P90D := P90 & !RESET;
P90D := P90 & !RESET;

CntUp := COUNT & UP;
CntDn := COUNT & !UP;

COUNT :=
( P90D & S2 & !P90D & S4 & X4 { C1 }
+!POD & !S2 & P90D & !S4 { C2 }
+!POD & S2 & !P90D & !S4 & X4 { C3 }
+ P90D & !S2 & P90D & S4 & X4 { C4 }
+ P90D & S2 & P90D & !S4 & X4 { C5 }
+!POD & !S2 & P90D & S4 { C6 }
+!POD & !S2 & P90D & S4 & X4 { C7 }
+ P90D & !S2 & !P90D & S4 & X4 { C8 }
 ) & !RESET;

UP :=
( !POD & S2 & !POD & S4
+!POD & S2 & P90D & S4
+!POD & S2 & P90D & !S4
+ P90D & S2 & P90D & !S4
+ P90D & !S2 & P90D & !S4
+ P90D & !S2 & !P90D & !S4
+ P90D & !S2 & !P90D & S4
+!POD & !S2 & !P90D & S4
 ) & !RESET;

END QuadDivider;
APPENDIX C: (PART 1): PID ALGORITHM FLOWCHART

- Enter doServo
  - Compute Error Value → U0
  - Compute proportional
    \[ Y = K_p \cdot U_0 \]
  - Is Y saturated?
    - No
      - Compute integral of Error
        \[ \Sigma U = \Sigma U + U_0 \]
        Compute integral portion
        \[ Y = Y + K_i \cdot \Sigma U \]
      - Compute differential
        \[ Y = Y + K_d \cdot (U_0 - U_1) \]
      - Scale Y
        \[ Y = Y \cdot 2^{SHIFTNUM} \]
      - Positive overflow in Y
        Yes → Set Y to max pos value = 007FFFFFFh
      - Negative overflow in Y
        Yes → Set Y to max neg value = FF800000h
    - Yes → Bypass integral "Anti-wind-up"
  - Extract YPWM from YK
    \[ Y_{PWM} = Y <23:8> \]
  - Check external position min./max. limit inputs

- Limit exceeded
  - No
    - Set YPWM = 0
  - Yes
    - Convert YPWM from unipolar to bipolar
      - LMD18200 requires that PWM duty cycle is not 0% or 100%
      - YPWM = 0? Yes → Set YPWM = 1%
      - YPWM = 100% Yes → Set YPWM = 99%
      - Zer0 lowest 6 bits of YPWM
        - Write YPWM high byte → PW1DCH
        - Write YPWM high byte → PW1DCL
      - Save current error value as previous error \( U_1 = U_0 \)
    - Return
APPENDIX C: (PART 2): PID ALGORITHM CODE LISTING

;***********************************************************************
; NAME:       doServo
; DESCRIPTION: Performs the servo loop calculations.
;***********************************************************************

doServo

MOV16  POSError,U0          ; save new position error in U0
LOADAB U0,KP
CALL Dmult
MVFP32 DPX,Y

CLRF WREG                 ; if previous output saturated, do
CPFSGT SATFLAG
CALL doIntegral

LOADAB INTEGRAL,KI       ; compute KI*INTEGRAL
CALL Dmult
ADD32 DPX,Y

MVFP16 U0,AARG
SUB16 U1,AARG
MVFP16 KV,BARG
CALL Dmult
ADD32 DPX,Y

CLRF WREG
CPFSGT SHIFTNUM
GOTO grabok

MOVFP SHIFTNUM,TMP

4-211
Servo Control of a DC-Brush Motor

zero6bits

```assembly
MOV24 Y+B1,YPWM+B0 ; move Y to YPWM and zero 6 bits
doTorque
MOVLW 0xC0
ANDWF YPWM+B0
BTFSC YPWM+B1,MSB
GOTO tmlimit
tplimit
BTFSS EXSTAT,BIT6
GOTO mplimitok
CLR32 YPWM
GOTO mplimitok
tmlimit
BTFSS EXSTAT,BIT5
GOTO mplimitok
CLR32 YPWM
mplimitok
MOVLW PWIDCH_INIT ; adjustment from bipolar to unipolar
MOVFP WREG,TMP+B1
MOVLI PWIDCL_INIT
MOVFP WREG,TMP+B0
ADD16 TMP,YPWM
CLRF TMP+B1 ; correct by 1 LSB
MOVLW 0x40
MOVPF WREG,TMP+B0 ; add one to bit5 of PWIDCL
ADD16 TMP,YPWM
testmax
CLRF TMP+B2 ; check pwm maximum limit
CLRF YPWM+B2 ; LMD18200 must have a minimum pulse
CLRF YPWM+B3
MOVFP16 YPWMAX,TMP
SUB24 YPWM,TMP
BTFSS TMP+B2,MSB
GOTO testmin
MOV16 YPWMAX,YPWM
GOTO limitok
testmin
CLRF TMP+B2 ; check pwm minimum limit
CLRF YPWM+B2
CLRF YPWM+B3
MOVFP16 YPWMIN,TMP
SUB24 YPWM,TMP
BTFSC TMP+B2,MSB
GOTO limitok
MOV16 YPWMIN,YPWM ; saturate to min
limitok
MOVLI BANK3 ; set new duty cycle
MOVFP YPWM+B0,PWIDCL
MOVFP YPWM+B1,PWIDCH
MOV16 U0,U1 ; push errors into U(k-1)
RETURN
```

If external position limits have been reached then zero PWM output.

Convert PWM from unipolar to bipolar

PWM cycle must not be 0% of 100%

Write PWM values to PWM registers

•*****************************************************************************
DS00532B-page 16 © 1993 Microchip Technology Inc.
4-212
APPENDIX D: ENCODER INTERFACE ROUTINE

;*****************************************************************************
; NAME: doMPosMVel
; DESCRIPTION: Calculates current position from UpCount and DownCount
;
; doMPosMVel
;
; Do UpCounter first

readUp

MVFP16 UPCOUNT, TMP+B0 ; save old upcount
MOVFP RTCCH, WREG
MOVFP RTCCCL, UPCOUNT+B0
CPPSEQ RTCCH ; Skip next if HI hasn't changed
GOTO readUp ; HI changed, re-read LO
MOVFP WREG, UPCOUNT+B1 ; OK to store HI now
CLRF MVELOCITY+B0 ; clear bits below binary point
MOV16 UPCOUNT, MVELOCITY+B1 ; compute upcount increment
SUB16 TMP+B0, MVELOCITY+B1

; Now do DownCounter

readDown

MVFP16 DOWNCOUNT, TMP+B0 ; save old downcount
MOV16 BANK2 ; timers in Bank 2
MOVFP TMR3H, WREG
MOVFP TMR3L, DOWNCOUNT+B0
CPPSEQ TMR3H ; Skip next if HI hasn't changed
GOTO readDown ; HI changed, re-read LO
MOVFP WREG, DOWNCOUNT+B1 ; OK to store HI now
MVFP16 DOWNCOUNT+B0, TMP+B2 ; compute downcount increment
SUB16 TMP+B0, TMP+B2
SUB16 TMP+B2, MVELOCITY+B1 ; compute new measured velocity
CLRF MVELOCITY+B3 ; sign extend measured velocity for position
STFSC MVELOCITY+B2, MSB ; 24 bit addition to measured position
SETF MVELOCITY+B3

ADD24 MVELOCITY+B1, MPOSITION ; compute new measured position
; delta position = measured velocity

RETURN

;*****************************************************************************

© 1993 Microchip Technology Inc.
APPENDIX E: IMPLEMENTATION DETAILS OF TRAJECTORY GENERATION

**doPreMove:**

This routine is executed only once at the beginning of each move. First, various buffers and flags are initialized and a test for modetype is performed. In position mode, the minimum move is triangular and consists of two steps. Therefore, if abs(MOVVAL) > 2, an immediate move is performed. Otherwise, normal move generation is possible with the sign of the move in MOVSIGN and the appropriate signed velocity and acceleration limits in V and A, and MOVVAL/2 in HMOVVAL.

In velocity mode, the sign of the move is calculated in MOVSIGN and the appropriate signed velocity and acceleration limits are placed in V and A. Finally, at modeready, MOVVAL is sign extended for higher precision arithmetic and the servo is enabled.

In torque mode, MOVVAL is output directly to the PWM and the servo is disabled, and doMove is not executed.

**doMove:**

Move generation is based on a piecewise constant acceleration model. During constant acceleration, this results in the standard equations for position and velocity given by

\[ x(t) = x_0 + v_0 t + a(t^2)/2, \quad v(t) = v_0 + a t \]

With the units for t in sample times, the time increment between subsequent sample times is 1, yielding the iterative equations for updating position and velocity implemented in doPosVel and given by

\[ P(k) = P(k-1) + V(k-1) + A/2, \quad V(k) = V(k-1) + A, \]

where A is the signed acceleration limit calculated in doPreMove. The inverse equations of this iteration, necessary for undoing an unwanted step, are contained in undoPosVel and given by

\[ P(k-1) = P(k) - V(K-1) - A/2, \quad V(K-1) = V(k) - A. \]

In position mode, the actual shape of the velocity profile depends on the values of V, A and the size of the move. Either the velocity limit is reached before half the move is completed, resulting in a trapezoidal velocity profile, or half the move is completed before the velocity limit is realized, resulting in a triangular velocity profile.

In the algorithm employed here, the velocity limit is treated as a bound on the actual velocity limit, thereby permitting exactly the same number of steps during the speedup and speeddown sections of the move. Phase1 is defined as the section of the move where the commanded position is less than half the move, and phase2 is the remaining portion of the move. T1 is time when the actual velocity limit is reached and T2 is the time at the end of phase 1.

**FIGURE A - SPEED PROFILE FOR TRAPEZOIDAL MOVES**

![Figure A](image)

**FIGURE B - SPEED PROFILE FOR TRIANGULAR MOVES**

![Figure B](image)

**FIGURE C - SPEED PROFILE FOR VELOCITY MOVES**

![Figure C](image)
Furthermore, let $x$ be the amount of undershoot and $y$ the amount of overshoot of half the move at $T_2$. Discretization error is minimized by using the values of $x$ and $y$ whether one more step will reduce the size of the final immediate move during the last step of the move. For a triangular move, the discretization error is given by $\min(2x, 2y)$, resulting in the condition that if $2x > 2y$, then take one more speedup step. In the case of a trapezoidal move, the discretization error is given by $\min(x, y-x)$, yielding the condition that if $3x > y$, take one more step during the flat section of phase 2.

At the beginning of doMove, MOVTIME is incremented and doPosVel is called to evaluate the next proposed values of commanded position and velocity under the current value of $A$. In position mode, phase 1, the original position plus half the move minus the new proposed commanded position is calculated and placed in MOVDEL, with the previous MOVDEL saved in MOVTMP. As half the move would be passed, MOVTMP = $-x$ and MOVDEL = $y$, with $y>0$ for the first time indicating that phase 1 is about to be completed. Therefore, if $y<0$, we continue in phase 1, where if maximum velocity has not been reached, the new proposed commanded position is executed. On the other hand, if the proposed move would exceed the maximum velocity, we undo the proposed move, set the current acceleration to zero, regenerate the iterative equations with the new acceleration, set $T_1 = MOVTIME - 1$, and execute the move.

Since $T_1$ is cleared in doPreMove, it is used as a flag to indicate if this corner in the velocity profile has been reached. Once we find that $y>0$, we drop into code that is executed only one time, with phase 2 beginning on the next step. If $T_1=0$, maximum velocity has not yet been reached, so $T_1=T_2$ and the velocity profile is triangular. In this case, $A$ is negated for speeddown, and if $x-y$, one more step is needed to minimize the discretization error. So $A$ is negated, the proposed step undone, $A$ is again negated for speeddown and the step recalculated and executed, with $T_2=T_1=MOVTIME-1$.

If $T_1$ is not zero, indicating that we are in the flat section of phase 1, then go to t-line 1, where $T_2=MOVTIME-1$, and if $3x>y$, then one more phase 2 flat step is necessary to minimize the discretization error. PH2FLAT is defined as the number of steps in the flat section of phase 2, and is used as a counter during its completion. If $3x>y$, then PH2FLAT = $T_2-T_1$, otherwise PH2FLAT = $T_2-T_1-1$ and phase 1 is finally complete. All subsequent steps will proceed through phase 2, first deciding if the flat section is finished by checking if PH2FLAT has reached zero. If not, go to flat where PH2FLAT is decremented, and tested if zero. If so, the speeddown section is begun by calculating the appropriate signed acceleration limit $A$, and executing the last of the flat section moves. For all following steps, PH2FLAT = 0, leaving only the final test for zero commanded velocity to indicate the end of the move. This will always occur since the actual maximum velocity, bounded above by the user supplied limit, is always an integer multiple of the user supplied acceleration limit, with exactly the same number of steps taken during speedup and speeddown.

The velocity mode is much more straightforward, with the velocity profile in the form of a ramp. If the final velocity has not been reached, the move continues at maximum acceleration. If the final velocity has been reached, the acceleration is set to zero and the move generation of commanded position and velocity continues unless the final velocity is zero.
### APPENDIX F: COMPLETE CODE LISTING (DCMOTOR.LST)

**Master Clock**
- `set MASTER_CLOCK = 16000000`;
- Input Clock Freq in Hz

**Sample Rate**
- `set SAMPLE_RATE = 1000`;
- Sample rate in Hz

**Encoder Rate**
- `set _ENCODER_RATE = 2000`;
- 2000 Pulses/rev

**Rated Speed**
- `set _RATED_SPEED = 6000`;
- in RPM

**Baud Rate**
- `set _BAUDRATE_ = 9600`

---

**Hardware Constants**

**Set Baud Rate**
- `#define _SET_BAUD_RATE(bps) ((10*MASTER_CLOCK/(64*bps)+5)/10 - 1)`

---

**Miscellaneous**

**Include**
- include "17c42.h"
- include "17c42.mac";
  - General Purpose Macros

---

**Assembly Code**

---

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

000F   PR2_INIT set ((10*MASTER_CLOCK/(4*(PR1_INIT+1)*SAMPLE_RATE))+5)/
       rwldcH_INIT set (((PR1_INIT+1) << 8) >> 9
       rwldcL_INIT set (((PR1_INIT+1) << 8) >> 1) & 0xff)

007F   PW1DCH_INIT set 0x7F
00C0   PW1DCL_INIT set 0xC0
0080   RTCSTA_INIT set 0x80
0090   RCSTA_INIT set 0x90
0020   TXSTAAI_INIT set 0x20
0019   SPBRG_INIT set _SET_BAUD_RATE(_BAUDRATE_)

00F3   DDRB_INIT set 0xF3
0000   DDRD_INIT set 0x00
       ; max and min pwm values
       ;
0040   PW1MINL set 0x40
0001   PW1MINH set 0x01 ; 0x0000 + 0x0140 (min 10 bit
0080   PW1MAXL set 0x80
00FE   PW1MAXH set 0xFE ; 0xFFCO - 0x0140 (max 10 bit
       ;
       ;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       ; global variables
       ;
       CBLOCK 0x18
0018   0004   DFX,DFX1,DFX2,DFX3 ; arithmetic accumula
001C   0004   AARG,AARG1,BARG,BARG1
       ;
       ; CBLOCK 0x18
0018   0004   TMP,TMP1,TMP2,TMP3 ; temporary variables
001C   0004   MOV TMP1,M0V TMPL,M0V TMP2,M0V TMP2
       ;
       ; CBLOCK 0x20
0020   0003   VL,VL1,VL2 ; velocity limit
0023   0003   AL,AL1,AL2 ; acceleration limit
0026   0000   KP,KP1 ; proportional gain
0028   0002   KV,KV1 ; velocity gain
002A   0002   K1,K11 ; integral gain
002C   0000   IM ; integrator mode
002C   0001   FV,FV1 ; velocity feedforward
002F   0002   FA,FA1 ; acceleration
0031   0000   VALBUF,VALBUF1,VALBUF2 ; i/o value buffer
0034   0000   DVALBUF,DVALBUF1,DVALBUF2 ; i/o value buffer
0037   0002   ISRBSR,ISRWREG ; i/o save storage
0039   0004   CMDCHAR,CMDTEMP,CMDPTRH,CMDPTLR ; command interface
003D   0003   XARPEN,PARLEN,PARFTR ; parameter variables
0040   0000   ;
0040   0003   CPOSITION,CPOSITION1,CPOSITION2 ; shutter commanded
0043   0003   CVELOCITY,CVELOCITY1,CVELOCITY2 ; shutter commanded
0046   0003   CMPOSITION,CMPOSITION1,CMPOSITION2 ; shutter measured
0049   0003   CMVELOCITY,CMVELOCITY1,CMVELOCITY2 ; shutter measured
004C   0000   ;
004C   0002   STRVALH,STRVALL ; string i/o variables
004E   0003   HEXVAL,HEXTMP,HEXSTAT ; hex i/o variables
0051   0000   ;
0051   0002   OPOSITION,OPOSITION1 ; original commanded
0053   0002   OPOSITION2,OPOSITION3 ;
0055   0003   POSITION,POSITION1,POSITION2 ; commanded position
0058   0003   VELOCITY,VELOCITY1,VELOCITY2 ; commanded velocity
005B   0000   ;

© 1993 Microchip Technology Inc. DS00532B-page 21
Servo Control of a DC-Brush Motor

005B 0004 NMOVVAL, NMOVVAL1, NMOVVAL2, NMOVVAL3 ; move value
005F 0004 MOVVAL, MOVVAL1, MOVVAL2, MOVVAL3 ; move value
0063 0004 HMOVVAL, HMOVVAL1, HMOVVAL2, HMOVVAL3 ; half move value
0067 0002 MOVTIME, MOVTIME1 ; move time in sample
0069 0000
0069 0001
0080 for
006A 0002
006C 0002
006E 0002
0070 0001
0071 0001
0072 0000
0072 0003
0075 0002
0077 0002
0079 0003
007C 0003
007F 0000
007F 0001
0080 0000
0088 0000
0088 0004
0090 0000
0090 0001
0091 0001
0092 0001
0093 0001
0094 0001
0095 0001
0096 0002
0098 0000
0098 0004
0099 0000
009C 0000
009C 0004
00A0 0004
00A4 0004
00A8 0004
00AC 0000
00AC 0002
00AE 0002
00B0 0000
00B0 0004
00B4 0002
00B6 0003
00B9 0000
00B9 0001
00BA 0000
00BA 0001
00BB 0000
00BB 0000
00BB 0001
00BC 0002
00BE 0002
00CC 0000
00CC 0000

if _PICMASTER_DEBUG
;*********************************************************************************************
; For PICMASTER Debug/servo tuning Purposes Only
;*****************************************************************************
CAPFLAG
; trace capture flag
CAPCOUNT, CAPCOUNT1
; PICMASTER trace
CAPTMP, CAPTMP1
; trace capture
;*********************************************************************************************

DS00532B-page 22 © 1993 Microchip Technology Inc.

4-218
Servo Control of a DC-Brush Motor

00C0 0001    endif
00C1 0000
00C1 0002
00C3 0002
00C5 0002
00C7 0001
00C8 0000

ENDC

;*************************************************************************
; NAME: AUTONO
;
; DESCRIPTION: Sets no auto increment or decrement
;
; TIMING (cycles): 4
;
AUTONO MACRO

BSF _fs0
BSF _fs1
BSF _fs2
BSF _fs3

ENDM

;*************************************************************************
; NAME: AUTOINC
;
; DESCRIPTION: Set auto increment
;
; TIMING (cycles): 4
;
AUTOINC MACRO

BSF _fs0
BCF _fs1
BSF _fs2
BCF _fs3

ENDM

;*************************************************************************
; NAME: AUTODEC
;
; DESCRIPTION: Sets auto decrement
;
; TIMING (cycles): 4
;
AUTODEC MACRO

BCF _fs0
BCF _fs1
BCF _fs2
BCF _fs3

ENDM

;*************************************************************************
; NAME: LOADAB
; DESCRIPTION: Loads extended math library AARG and BARG
; ARGUMENTS: A -> AARG
; B -> BARG
; TIMING (cycles): 4

LOADAB MACRO A,B

MOVFP A+B0, AARG+B0 ; load lo byte of A to AARG
MOVFP A+B1, AARG+B1 ; load hi byte of A to AARG
MOVFP B+B0, BARG+B0 ; load lo byte of B to BARG
MOVFP B+B1, BARG+B1 ; load hi byte of B to BARG

ENDM

;*********************************************************
; ascii constants
;
000D   CR set 0x0D
0018   CAN set 0x18
0008   BS set 0x08
0020   SP set 0x20
000A   LF set 0x0A
002D   MN set '-'

;*********************************************************
; cmds constants and macros
;
0001   CHARREADY set 0x01

0008   NUMPAR set 0x08

; Response characters

0021   CMD_OK set '!'
003F   CMD_BAD set '?'

; Exit values

0000   HEX_SP set 0x00
0001   HEX_MN set 0x01
0002   HEX_CR set 0x02
0003   HEX_CAN set 0x03

0000   DEC_SP set 0x00
0001   DEC_MN set 0x01
0002   DEC_CR set 0x02
0003   DEC_CAN set 0x03

; Command characters

000D   DO_NULL set CR
004D   DO_MOVE set 'M'; M
004F   DO_MODE set 'O'; O
0053   DO_SETPARAMETER set 'S'; S
0052   DO_READPARAMETER set 'R'; R
0043   DO_SHUTTER set 'C'; C
0050   DO_READCOMPOSITION set 'P'; P
0056   DO_READCOMVELOCITY set 'V'; V
Servo Control of a DC-Brush Motor

0070 DO_READACTPOSITION set ‘p’ ; p
0076 DO_READACTVELOCITY set ‘v’ ; v
0058 DO_EXTERNALSTATUS set ‘X’ ; X
0059 DO_MOVESTATUS set ‘Y’ ; Y
0049 DO_READINDPOSITION set ‘I’ ; I
0048 DO_SETPOSITION set ‘H’ ; H
005A DO_RESET set ‘Z’ ; Z
0073 DO_STOP set ‘s’ ; s
0063 DO_CAPTURE set ‘c’ ; c

;*************************************************************************
; NAME: CMD_DEF
; DESCRIPTION: Creates all the definitions for a command table data struc-
; ture. The first word is at the command character used, and
; the second word is a pointer to the function that handles
; this command function.
; ENTRY CONDITIONS: Must be contiguous with the other entries for the
; function to work.
; ARGUMENTS: FUNC command execution function
; ROOT NAME ROOT
;
CMD_DEF MACRO FUNC,ROOT
DATA ROOT
DATA FUNC
ENDM

0002 CMD_ENTRY_LENGTH set 2

;*************************************************************************
; NAME: CMD_START
; DESCRIPTION: Labels the start of the command table.
;
CMD_START MACRO LABEL
LABEL
ENDM

;*************************************************************************
; NAME: CMD_END
; DESCRIPTION: Marks the end of the command table with an entry of 0x00
;
CMD_END MACRO
DATA 0x00
ENDM

;*************************************************************************

;*************************************************************************

PIO Constants

; define PIV parameters, computation based on errors only. Does not involve
Servo Control of a DC-Brush Motor

Kp=3600, Ki=112, Kd= 28800, Shiftcount = 3
Kp=2300 Ki= 41, Kd= 32200, Shiftcount = 4
Kp=1024, Ki=8, Kd=31405 , ShiftCount = 5
Kp=5400, Ki=310, Kd=23400 , ShiftCount = 2
Kp=3600, Ki=192, Kd=16800, ShiftNum = 3
Kp=1800, Ki=52, Kd=15600, ShiftNum = 4

Adjust Shiftcount by maximizing Kd (for a 16 bit signed num)

#define _KP
#define _KI
#define _KV
_vel_limit set (_RATED_SPEED*ENCODER_RATE)/2000
_accl_limit set 0x2000

if _SEVO_PID == TRUE
  _SHIFTNUM equ 4
endif

ORG 0x0
goto Startup

ORG 0x20
goto InterruptPoll

; NAME: Startup
; DESCRIPTION: This routine is called on the hardware reset or when the
; program wishes to restore initial conditions. Initialization
; of run-time constants takes place here.
; RETURNS: restart to safe and initial state
; STACK UTILIZATION: none
; TIMING (in cycles): X

Startup

bsf _glintd disable all
 ; no auto

BSF _fs0
BSF _fs1
BSF _fs2
BSF _fs3

movlw 0x18
movpf wreg,fsrO

memloop

clr _indfO
incfsz _fsrO
goto memloop

incf ONE
Servo Control of a DC-Brush Motor

movlb bank3 ; bank3 ini
moviw TCON2_INIT
movfp wreg,tcon2
movlw PWIDCH_INIT ; set duty
movfp wreg,pw1dch
movfp wreg,pw2dch
moviw PWIDCL_INIT
movfp wreg,pw1dcl
movfp wreg,pw2dcl
movlw TCON1_INIT ; set organization of timers
movfp wreg,tcon1
movlb bank2 ; bank2 initialization
moviw PRL_INIT
movfp wreg,prl ; initialize timer1 period
movlw PR2_INIT
movfp wreg,pr2 ; initialize timer2 period
movlw TCON1_INIT
movfp wreg,tcon1
movlb bank0 ; bank0 initialization
moviw RTCSTA_INIT
movfp wreg,rtcsta ; sets RTC for external input
movlw Ox3F
movwf porta ; RA2 connected to BREAK Input of LMD18200
motor

if _SERIAL_IO
moviw RCSTA_INIT
movfp wreg,rcsta ; set receive status
moviw TXSTA_INIT
movfp wreg,txsta ; set transmit status
moviw SPBRG_INIT
movfp wreg,spbrg ; set baud rate
endif
moviw DDRB_INIT
movfp wreg,ddrb ; set port B for whatever
movlb bank1 ; bank1 initialization

if (_SERVO_PID == TRUE)
MOVK16 _KP,KP
MOVILW (1800) & 0xff
MOVWF KP+B0
MOVILW (((1800) >> 8)
MOVWF KP+B1
MOVK16 _KI,KI
MOVILW (52) & 0xff
MOVWF KI+B0
MOVILW (((52) >> 8)
MOVWF KI+B1
MOVK16 _KV,KV
MOVLW (15600) & 0xff
MOVF KV+B0
MOVLW ((15600) >> 8)
MOVF KV+B1

MOVK16 _ACCL_LIMIT,AL
MOVLW (_ACCL_LIMIT) & 0xff
MOVF AL+B0
MOVLW ((_ACCL_LIMIT) >> 8)
MOVF AL+B1

MOVK16 _VEL_LIMIT,VL
MOVLW (_VEL_LIMIT) & 0xff
MOVF VL+B0
MOVLW ((_VEL_LIMIT) >> 8)
MOVF VL+B1

movlw _SHIFTNUM
movwf SHIFTNUM
movpf pwldch,YPWM+B1

movlw PWMAXL ; initialize pwm limits
movpf wreg,YPWMAX+B0
movlw PWMAXH
movpf wreg,YPWMAX+B1
movlw PWMINL
movpf wreg,YPWMIN+B0
movlw PWMINH
movpf wreg,YPWMIN+B1

clrf pir ; clear flags, set individual interrupts
clrf intsta
bsf _tm2ie
bsf _peie
bcf glintd ; enable interrupts
movlb bank2

crflr rtcc1 ; clear up counter
crflr rtcc2

crflr tmr3l ; clear down counter
crflr tmr3h

movlw 0xFF
delay decfsz wreg
goto delay

crflr rtcc1,wreg
iorwf rtcc2,w
iorwf tmr3l,w
iorwf tmr3h,w

movlw TSTFSZ wreg
goto zeroctrs ; motor still moving
goto PollingLoop

;************************************************************************

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

;*************************************************************************
; NAME: InterruptPoll

InterruptPoll

007D 0138 movwf ISRWREG ; save W Reg
007E 6A04 movfp alusta,wreg ; save alusta
007F 01C7 movwf alustatemp ; save Table Pointers
0080 4F37 movfp bsr,ISRBSR
0081 4DC3 movfp tblptrl,tblptrltemp
0082 4EC4 movfp tblptrh,tblptrhtemp
0083 A0C5 tlrd 0,TblLatLo
0084 A2C6 tlrd 1,TblLatHi ; save Table Latch
0085 B801 movlb bank1
0086 E8E5 call doMPosMVel ; calculate measured position and velocity
0087 E111 call doExtstat ; evaluate external status
0088 2293 rlncf MOVSTAT,W ; if MOVFLAG=0 and MOVSTAT,bit7=1
0089 B501 andlw 0x01 ; then do premove. This is only
008A 0494 subwf MOVFLAG,W ; executed once at the beginning of
008B 9F0A btfsr wreg,MSB ; each move
008C E23D call doPreMove
008D B993 btfsr MOVSTAT,bit6 ; is motion continuing?
008E E30F call doMove ; if so, do move
008F E09E call doError ; calculate position and velocity
0090 3390 tstfsz SERVOFLAG ; error
0091 E4AC call doServo ; do servo
0092 33BB _pimsidebug ; PIC-MASTER Trace Capture, demo
0093 879A tstfsz CAPFLAG
0094 B801 movlb bank1 ; for PIC-MASTER Trace Capture, demo
0095 2916 clrf pir ; clear all interrupt request flags
0096 A4C5 tlwt 0,TblLatLo ; restored table latch
0097 6C6 tlwt 1,TblLatHi ; restored table pointers
0098 6DC3 movfp tblptrltemp,tblptrl
0099 6EC4 movfp tblptrhtemp,tblptrh
009A 6F37 movfp ISRBSR,bsr
009B 64C7 movfp alustatemp,alusta
009C 6A38 movfp ISRWREG,wreg
009D 0005 retfie

;*************************************************************************
;*************************************************************************
; NAME: doError
;
; DESCRIPTION: Calculates the position and velocity error.
;
doError

MOV24 POSITION,POSERROR ; calculate position error
Servo Control of a DC-Brush Motor

009E 6A55 MOVFP POSITION+B0,wreg ; get byte of POSITION into w
009F 4A79 MOVFP wreg,POSErrOR+B0 ; move to POSErrOR(B0)
00A0 6A56 MOVFP POSITION+B1,wreg ; get byte of POSITION into w
00A1 4A7A MOVFP wreg,POSErrOR+B1 ; move to POSErrOR(B1)
00A2 6A57 MOVFP POSITION+B2,wreg ; get byte of POSITION into w
00A3 4A7B MOVFP wreg,POSErrOR+B2 ; move to POSErrOR(B2)

SUB24 MPOSITION,POSError

00A4 6A72 MOVFP MPOSITION+B0,wreg ; get lowest byte of MPOSITION into
00A5 0579 SUBWF POSErrOR+B0 ; sub lowest byte of POSErrOR,
00A6 6A73 MOVFP MPOSITION+B1,wreg ; get 2nd byte of MPOSITION into
00A7 037A SUBWF POSErrOR+B1 ; sub 2nd byte of POSErrOR, save
00A8 6A74 MOVFP MPOSITION+B2,wreg ; get 3rd byte of MPOSITION into
00A9 037B SUBWF POSErrOR+B2 ; sub 3rd byte of POSErrOR, save

00AA 9F7B btfsC POSErrOR+B2,MSB ; saturate error to lowest 16 bits
00AB COB7 goto pneg

SUB24 MVELOCITY,VELERROR

00AC 6A7A movfp VELOCITY+B0,wreg ; get byte of VELOCITY into w
00AD B580 movfp VELOCITY+B1,wreg ; move to VELOCITY(B0)
00AE 097B movfp VELOCITY+B2,wreg ; move to VELOCITY(B1)
00AF 290A MOVFP wreg,VELERROR+B0 ; get byte of VELOCITY into w
00B0 327B MOVFP wreg,VELERROR+B1 ; get 2nd byte of VELOCITY into w
00B1 COCL MOVFP wreg,VELERROR+B2 ; get 3rd byte of VELOCITY into w

DS00532B-page 30 © 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

```
00CF 6A7D movfp VELERROR+B1,wreg
00DD B580 andlw 0x80
00D1 097E lorwf VELERROR+B2
00D2 290A clrf wreg
00D3 327E cplfsgt VELERROR+B2
goto vsatok
00D4 C084 clrf VELERROR+B2
00D5 297E movlw 0x7F
00D6 B07F movpf wreg,VELERROR+B1
00D7 4A7D setf VELERROR
00D8 2B7C goto vsatok
00D9 C024

vreg movfp VELERROR+B1,wreg
00DA 6A7D iorlw 0x7F
00DB B37F andwf VELERROR+B2
00DC 0B7E setf wreg
00DD 2B0A cplfslt VELERROR+B2
goto vsatok
00DE C0E4 setf VELERROR+B2
00EF 2B7E clrf VELERROR+B1
00F0 8770 bsf VELERROR+B1,MSB
00F1 297C clrf VELERROR

vsatok return

;******************************************************************************
;******************************************************************************
; NAME: doMPosMVel
;
; DESCRIPTION: Calculates current position from UpCount and DownCount
;
doMPosMVel

; Do UpCounter first
MOVFP16 UPCOUNT,TMP+BO ; save old upcount

00E5 78AC MOVFP UPCOUNT+BO,TMP+BO+BO ; move UPCOUNT(BO) to TMP(BO)
00E6 79AD MOVFP UPCOUNT+Bl,TMP+BO+Bl ; move UPCOUNT(Bl) to TMP(Bl)

readUp

00E7 4CA movpf rtccl,UPCOUNT+BO
00E8 49AC movpf rtcch,UPCOUNT+Bl
00E9 310C cpfsseq rtcch
00EA C0E7 goto readUp
00EB 4AAD movpf wreg,UPCOUNT+Bl

00EC 2975 clrf MVELOCITY+BO

MOV16 UPCOUNT,MVELOCITY+Bl

00ED 6A4 MOVFP UPCOUNT+BO,wreg
00EE 0176 MOVWF MVELOCITY+Bl+BO
00EF 6A4 MOVFP UPCOUNT+Bl,wreg
00F0 0177 MOVWF MVELOCITY+Bl+Bl

SUB16 TMP+BO,MVELOCITY+Bl

00F1 6A18 MOVFP TMP+BO+BO,wreg
00F2 0576 SUBWF MVELOCITY+Bl+BO
00F3 6A19 MOVFP TMP+BO+Bl,wreg
00F4 0377 SUBWF MVELOCITY+Bl+Bl

;******************************************************************************
;******************************************************************************
```

© 1993 Microchip Technology Inc.
; Now do DownCounter
MOVFP16 DOWNCOUNT,TMP+B0 ; save old downcount
00F5 78AE MOVFP DOWNCOUNT+B0,TMP+B0+B0 ; move DOWNCOUNT+B0 to
00F6 79AF MOVFP DOWNCOUNT+B1,TMP+B0+B1 ; move DOWNCOUNT+B1 to

readDown
00F7 B802 movlb bank2 ; timers in Bank 2
00F8 530A movpf tmr3h,wreg
00F9 52AE movpf tmr3l,DOWNCOUNT+B0
00FA 3113 cpfseq tmr3h
00FB 0F7F goto readDown ; Skip next if HI hasn’t changed
00FC 4AAF movpf wreg,DOWNCOUNT+B1 ; HI changed, re-read LO

MOVFP16 DOWNCOUNT+B0,TMP+B2 ; compute downcount increment
00FD 7A9E MOVFP DOWNCOUNT+B0+B0,TMP+B0+B0 ; move DOWNCOUNT+B0(B0) to
00FE 7BAF MOVFP DOWNCOUNT+B0+B1,TMP+B0+B1 ; move DOWNCOUNT+B0(B1) to

SUB16 TMP+B0,TMP+B2
00FF 6A18 MOVFP TMP+B0+B0,wreg ; get lowest byte of TMP+B0 into
0100 051A SUBWF TMP+B2+B0 ; sub lowest byte of TMP+B2, save
0101 6A19 MOVFP TMP+B0+B1,wreg ; get 2nd byte of TMP+B0 into w
0102 031B SUBWFB TMP+B2+B1 ; sub 2nd byte of TMP+B2, save in

SUB16 TMP+B2,MVELOCITY+B1 ; compute new measured velocity
0103 6A1A MOVFP TMP+B2+B0,wreg ; get lowest byte of TMP+B2 into
0104 0576 SUBWF MVELOCITY+B1+B0 ; sub lowest byte of
0105 6A1B MOVFP TMP+B2+B1,wreg ; get 2nd byte of TMP+B2 into w
0106 0377 SUBWFB MVELOCITY+B1+B1 ; sub 2nd byte of MVELOCITY+B1,

0107 2978 clrf MVELOCITY+B3 ; sign extend measured velocity
0108 9F77 btfsc MVELOCITY+B2,MSB ; 24 bit addition to measured
0109 2B78 setf MVELOCITY+B3

ADD24 MVELOCITY+B1,MPOSITION ; compute new measured position
010A 6A76 MOVFP MVELOCITY+B1+B0,wreg ; get lowest byte of MVELOCITY+B1
010B 0F72 ADDWF MPOSITION+B0 ; add lowest byte of MPOSITION,
010C 6A77 MOVFP MVELOCITY+B1+B1,wreg ; get 2nd byte of MVELOCITY+B1
010D 1173 ADDWF MPOSITION+B1 ; add 2nd byte of MPOSITION, save
010E 6A78 MOVFP MVELOCITY+B1+B2,wreg ; get 3rd byte of MVELOCITY+B1
010F 1174 ADDWF MPOSITION+B2 ; add 3rd byte of MPOSITION, save

; delta position = measured
0110 0002 return

;*************************************************************************
; NAME: doExtstat
; DESCRIPTION; Get +limit,-limit,GPI from PORTB and set in EXTSTAT
; doExtstat
0111 9407 btfs _intir
0112 C11B goto otherbits
MOV24 MPOSITION,INDEXPOS

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

MOVFP MPOSITION+B0,wreg ; get byte of MPOSITION into w
MOVFP wreg,INDEXPOS+B0 ; move to INDEXPOS(B0)
MOVFP MPOSITION+B1,wreg ; get byte of MPOSITION into w
MOVFP wreg,INDEXPOS+B1 ; move to INDEXPOS(B1)
MOVFP MPOSITION+B2,wreg ; get byte of MPOSITION into w
MOVFP wreg,INDEXPOS+B2 ; move to INDEXPOS(B2)

bcf _intir
bsf EXTSTAT,MSB

movlb bank0 ; get +limit,-limit and GPI
movfp portb,wreg ; arrange in correct bit posi
andlw 0x61
swapf TMP
iorwf TMP,W
iorwf EXT STAT ; set in EXTSTAT

return

PollingLoop

if _SERIAL_IO
    call IdleFunction
    call GetChk ; GetChk, is receive buffer full?
    cpfseq ONE
    goto PollingLoop
else
    clrwdt
    goto PollingLoop ; wait for Interrupt
endif

;*************************************************************************
;*************************************************************************
include "mult.asm" ; Double Precision

Double Precision Multiplication Routine
;*************************************************************************
; Double Precision Multiplication Routine
; ; NAME: Dmult
; ; DESCRIPTION: Mult: AARG (16 bits) * BARG (16 bits) -> DPX (32 bits)
; ; (a) Load the 1st operand in locations AARG+B0 & AARG+B1 (16 bits)
; ; (b) Load the 2nd operand in locations BARG+B0 & BARG+B1 (16 bits)
; ; (c) call Dmult
; ; (d) The 32 bit result is in locations (DPX+B0,DPX+B1,DPX+B2,DPX+B3)
; ; In the signed case, a savings of 9 clks can be realized by choosing
Servo Control of a DC-Brush Motor

; BARG as the positive factor in the product when possible.
;
; TIMING (worst case): unsigned: 173 clks
; signed: if BARG positive: 170 clks
; if BARG negative: 179 clks
;
;*************************************************************************
SIGNED equ TRUE ; Set This To 'TRUE' for signed multiply
; 'FALSE' for unsigned.
;*************************************************************************

MULTMAC MACRO
variable i

i = 0
if SIGNED
  while i < 15
else
  while i < 16
endif
if i < 8
  btsc BARG+BO,i ; test low byte
else
  btsc BARG+Bl,i-8 ; test high byte
endif
if i < 8
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B2 ; for i < 8, no meaningful
  rrcf DPX+B1
else
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
endif
i = i+1
endw

clr DPX+B0 ; if we get here, BARG = 0

return

add0
movf AARG+BO,wreg
addwf DPX+B2 ;add lsb
movf AARG+Bl,wreg ;add msb
addwf DPX+B3
rlcf AARG+Bl,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2
rrcf DPX+B1

i = 1
if SIGNED
  while i < 15

0001
Servo Control of a DC-Brush Motor

else
    while i < 16
        endif
        if i < 8
            btfss BARG+B0,i ; test low byte
        else
            btfss BARG+B1,i-8 ; test high byte
        endif
        goto noadd#v(i) ; BARG+BO, wreg
        movfp DPX+B2 ; add low bit
        movfp AARG+B1,wreg ; add high bit
        addwfc DPX+B3 ; add msb
    endif
    noadd#v(i)
    if i < 8
        rlcf AARG+B1,W ; rotate sign into carry bit
        rrcf DPX+B3 ; for i < 8, no meaningful
        rrcf DPX+B2 ; are in DPX+B0
        rrcf DPX+B1
        rrcf DPX+B0
    else
        rlcf AARG+B1,W ; rotate sign into carry bit
        rrcf DPX+B3
        rrcf DPX+B2
        rrcf DPX+B1
        rrcf DPX+B0
    endif
    i = i+1
endw
if SIGNED
    rlcf AARG+B1,W ; since BARG is always made
    rrcf DPX+B3 ; the last bit is known to be
    rrcf DPX+B2
    rrcf DPX+B1
    rrcf DPX+B0
endif
ENDM

; Double Precision Multiply
; ( AARG*BARG -> : 32 bit

; Dmult
if SIGNED

012B 971F btfss BARG+B1,MSB ; test sign of BARG
012C C137 goto argsok ; if positive, ok
                NEG16 AARG+B0 ; if negative, then negate

012D 131C COMF AARG+B0+B0
012E 131D COMF AARG+B0+B1
012F 290A CLRF wreg
0130 151C INCF AARG+B0+B0
0131 111D ADDWFC AARG+B0+B1

                NEG16 BARG+B0 ; AARG and BARG

0132 131E COMF BARG+B0+B0
0133 131F COMF BARG+B0+B1
Servo Control of a DC-Brush Motor

0134 290A  CLRF  wreg
0135 151E  INCF  BARG+B0+B0
0136 111F  ADDWFC  BARG+B0+B1

endif

argsok

0137 291B  clrf  DPX+B3  ; clear initial partial product
0138 291A  clrf  DPX+B2

MULTMAC  variable i

0000  i = 0
0000  if SIGNED
    while i < 15
      if
        while i < 16
          endif
          goto  add#v(i)
          endif
          goto  add#v(i)
        endif
        goto  add#v(i)
    endif
    if i < 8
      btfsc  BARG+B0,i  ; test low byte
      else
        btfsc  BARG+B1,i-8  ; test high byte
        endif
        goto  add#v(i)
    endif
    if i < 8
      rlcf  DPX+B3,W  ; rotate sign into carry bit
      rrcf  DPX+B3  ; for i < 8, no meaningful bits
      rrcf  DPX+B2
      rrcf  DPX+B1
      else
        rlcf  DPX+B3,W  ; rotate sign into carry bit
        rrcf  DPX+B3  ; are in DPX+B0
        rrcf  DPX+B2
        rrcf  DPX+B1
        rrcf  DPX+B0
      endif
      i = i+1
      endw

0139 981E  if i < 8
          btfsc  BARG+B0,i  ; test low byte
          else
            btfsc  BARG+B1,i-8  ; test high byte
          endif
          goto  add0
          if i < 8
            rlcf  DPX+B3,W  ; rotate sign into carry bit
            rrcf  DPX+B3  ; for i < 8, no meaningful
            rrcf  DPX+B2  ; are in DPX+B0
            rrcf  DPX+B1
            else
              rlcf  DPX+B3,W  ; rotate sign into carry bit
              rrcf  DPX+B3
              rrcf  DPX+B2
              rrcf  DPX+B1
              rrcf  DPX+B0

DS00532B-page 36  © 1993 Microchip Technology Inc.
endif

i = i+1

if i < 8
  btfs BARG+B0,i ; test low byte
else
  btfs BARG+B1,i-8 ; test high byte
endif

goto add1

if i < 8
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B3 ; for i < 8, no meaningful
  rrcf DPX+B2 ; are in DPX+B0
  rrcf DPX+B1
else
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0
endif

i = i+1

if i < 8
  btfs BARG+B0,i ; test low byte
else
  btfs BARG+B1,i-8 ; test high byte
endif

goto add2

if i < 8
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B3 ; for i < 8, no meaningful
  rrcf DPX+B2 ; are in DPX+B0
  rrcf DPX+B1
else
  rlcf DPX+B3,W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0
endif

i = i+1

if i < 8
  btfs BARG+B0,i ; test low byte
else
  btfs BARG+B1,i-8 ; test high byte
endif

goto add3

if i < 8
Servo Control of a DC-Brush Motor

014D 1A1B
014E 191B
014F 191A
0150 1919
else
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
endif

0004

i = i+1

if i < 8
btfsc BARG+B0,i ; test low byte
else
btfsc BARG+B1,i-8 ; test high byte
endif

0152 C1C4
goto add4

if i < 8
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
else
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0
endif

0005

i = i+1

if i < 8
btfsc BARG+B0,i ; test low byte
else
btfsc BARG+B1,i-8 ; test high byte
endif

0158 C1CE
goto add5

if i < 8
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
else
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0
Servo Control of a DC-Brush Motor

```assembly
  rrcf      DPX+B0
  endif

  0006
  i  =  i+1

  if i  <  8
    btfscl BARG+B0,i  ; test low byte
    else
    btfscl BARG+B1,i-8  ; test high byte
  endif

  015D 9E1E
  endif

  goto      add6
  if i  <  8
    rlcfl DPX+B3,W  ; rotate sign into carry bit
    rrcfl DPX+B3     ; for i  <  8, no meaningful
    rrcfl DPX+B2     ; are in DPX+B0
    rrcfl DPX+B1
    rrcfl DPX+B0
  else
    rlcfl DPX+B3,W  ; rotate sign into carry bit
    rrcfl DPX+B3
    rrcfl DPX+B2
    rrcfl DPX+B1
    rrcfl DPX+B0
  endif

  0007
  i  =  i+1

  if i  <  8
    btfscl BARG+B0,i  ; test low byte
    else
    btfscl BARG+B1,i-8  ; test high byte
  endif

  0164 C1E2
  goto      add7
  if i  <  8
    rlcfl DPX+B3,W  ; rotate sign into carry bit
    rrcfl DPX+B3     ; for i  <  8, no meaningful
    rrcfl DPX+B2     ; are in DPX+B0
    rrcfl DPX+B1
    rrcfl DPX+B0
  else
    rlcfl DPX+B3,W  ; rotate sign into carry bit
    rrcfl DPX+B3
    rrcfl DPX+B2
    rrcfl DPX+B1
    rrcfl DPX+B0
  endif

  0008
  i  =  i+1

  if i  <  8
    btfscl BARG+B0,i  ; test low byte
    else
    btfscl BARG+B1,i-8  ; test high byte
  endif
```

---

© 1993 Microchip Technology Inc.  DS00532B-page 39
Servo Control of a DC-Brush Motor

016A C1EC  goto add8
    if i < 8
        rlcf DPX+B3,W  ; rotate sign into carry bit
        rrcf DPX+B3  ; for i < 8, no meaningful
        rrcf DPX+B2  ; are in DPX+B0
        rrcf DPX+B1
    endif

0009  i = i+1
    if i < 8
        btf sc BARG+BO, i  ; test low byte
    else
        btfsc BARG+Bl, i-8  ; test high byte
    endif

0170 991F  goto add9
    if i < 8
        rlcf DPX+B3,W  ; rotate sign into carry bit
        rrcf DPX+B3  ; for i < 8, no meaningful
        rrcf DPX+B2  ; are in DPX+B0
        rrcf DPX+B1
    endif

000A  i = i+1
    if i < 8
        btf sc BARG+BO, i  ; test low byte
    else
        btfsc BARG+Bl, i-8  ; test high byte
    endif

0177 9A1F  goto add10
    if i < 8
        rlcf DPX+B3,W  ; rotate sign into carry bit
        rrcf DPX+B3  ; for i < 8, no meaningful
        rrcf DPX+B2  ; are in DPX+B0
        rrcf DPX+B1
    endif

0179 1A1B  goto add11
    if i < 8
        rlcf DPX+B3,W  ; rotate sign into carry bit
        rrcf DPX+B3  ; for i < 8, no meaningful
        rrcf DPX+B2  ; are in DPX+B0
        rrcf DPX+B1
    endif

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

```
017C 1919
017D 1918
endif
    i = i + 1
if i < 8
    btfsc BARG+B0, i ; test low byte
else
017E 9B1F
    btfsc BARG+B1, i-8 ; test high byte
017F C20D
    goto addll
endif
    if i < 8
        rlcf DPX+B3, W ; rotate sign into carry bit
        rrcf DPX+B3
        rrcf DPX+B2
        rrcf DPX+B1
    else
        rlcf DPX+B3, W ; rotate sign into carry bit
        rrcf DPX+B3
        rrcf DPX+B2
        rrcf DPX+B1
    endif
0180 1A1B
0181 191B
0182 191A
0183 1919
0184 1918
endif
    i = i + 1
if i < 8
    btfsc BARG+B0, i ; test low byte
else
0185 9D1F
    btfsc BARG+B1, i-8 ; test high byte
0186 C218
    goto addl2
endif
    if i < 8
        rlcf DPX+B3, W ; rotate sign into carry bit
        rrcf DPX+B3
        rrcf DPX+B2
        rrcf DPX+B1
    else
        rlcf DPX+B3, W ; rotate sign into carry bit
        rrcf DPX+B3
        rrcf DPX+B2
        rrcf DPX+B1
    endif
0187 1A1B
0188 191B
0189 191A
018A 1919
018B 1918
endif
    i = i + 1
if i < 8
    btfsc BARG+B0, i ; test low byte
else
018C 9D1F
    btfsc BARG+B1, i-8 ; test high byte
018D C223
    goto addl3
endif
    if i < 8
        rlcf DPX+B3, W ; rotate sign into carry bit
```

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

```assembly
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0
endif

i = i+1

if i < 8
btfsc BARG+B0,i ; test low byte
else
endif

0193 9E1F
btfsc BARG+B1,i-8 ; test high byte
endif
goto addl4

if i < 8
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2
rrcf DPX+B1

else
rlcf DPX+B3,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0
endif

i = i+1

019A 2918
clrfd DPX+B0 ; if we get here, BARG = 0
019B 0002
return

add0

movfp AARG+B0,wreg
addwf DPX+B2
movfp AARG+B1,wreg
addwf DPX+B3 ;add msb
rlcf AARG+B1,W ; for i < 8, no meaningful
rrcf DPX+B3 ; are in DPX+B0
rrcf DPX+B2
rrcf DPX+B1

i = i-1

if SIGNED
    while i < 15
else
    while i < 16
endif
```
Servo Control of a DC-Brush Motor

if i < 8
    btfss BARG+BO,i ; test low byte
else
    btfss BARG+BI,i-8 ; test high byte
endif
    goto noadd#v(i)
add#v(i)
    movfp AARG+BO,wreg
    addwf DFX+B2 ;add lsb
    movfp AARG+BI,wreg
    addwfc DFX+B3 ;add msb

noadd#v(i)
    if i < 8
        rlcf AARG+BI,W ; rotate sign into carry bit
        rrcf DFX+B3 ; for i < 8, no meaningful bits
        rrcf DFX+B2
        rrcf DFX+B1
    else
        rlcf AARG+BI,W ; rotate sign into carry bit
        rrcf DFX+B3
        rrcf DFX+B2
        rrcf DFX+B1
        rrcf DFX+B0
    endif
    i = i+1
endw

if i < 8
    btfss BARG+BO,i ; test low byte
else
    btfss BARG+BI,i-8 ; test high byte
endif

addl
goto noaddl

movfp AARG+BO,wreg
addwf DFX+B2 ;add lsb
movfp AARG+BI,wreg
addwfc DFX+B3 ;add msb

noaddl
    if i < 8
        rlcf AARG+BI,W ; rotate sign into carry bit
        rrcf DFX+B3 ; for i < 8, no meaningful bits
        rrcf DFX+B2
        rrcf DFX+B1
    else
        rlcf AARG+BI,W ; rotate sign into carry bit
        rrcf DFX+B3
        rrcf DFX+B2
        rrcf DFX+B1
        rrcf DFX+B0
Servo Control of a DC-Brush Motor

```asm
endif

0002  
i = i+1
01AE 921E
else
endif

01AF C1B4
add2
01B0 6A1C
movfp AARG+0, wreg
01B1 0F1A
addwf DPX+2
01B2 6A1D
movfp AARG+1, wreg
01B3 111B
addwf DPX+3;

noadd2
if i < 8
01B4 1A1D
rlcf AARG+1,W
01B5 191B
rrcf DPX+3
01B6 191A
rrcf DPX+2
01B7 1919
rrcf DPX+1
else
rlcf AARG+1,W
rrcf DPX+3
rrcf DPX+2
rrcf DPX+1
rrcf DPX+0
endif

0003  
i = i+1
01B8 931E
else
endif

01B9 C1BE
add3
goto noadd3
01BA 6A1C
movfp AARG+0, wreg
01BB 0F1A
addwf DPX+2
01BC 6A1D
movfp AARG+1, wreg
01BD 111B
addwf DPX+3;

noadd3
if i < 8
01BE 1A1D
rlcf AARG+1,W
01BF 191B
rrcf DPX+3
01C0 191A
rrcf DPX+2
else
rlcf AARG+1,W
rrcf DPX+3
rrcf DPX+2
rrcf DPX+1
rrcf DPX+0
endif
```

© 1993 Microchip Technology Inc.

4-240
Servo Control of a DC-Brush Motor

```
01C1 1919
    rrcf   DPX+B1
    else
        rlcf   AARG+B1,W   ; rotate sign into carry bit
        rrcf   DPX+B3
        rrcf   DPX+B2
        rrcf   DPX+B1
        rrcf   DPX+B0
    endif

0004
    i = i+1

01C2 941E
    if i < 8
        btfss   BARG+B0,i   ; test low byte
    else
        btfss   BARG+B1,i-8  ; test high byte
    endif

01C3 C1C8
    add4
    goto   noadd4

01C4 6A1C
    movfp   AARG+B0,wreg
01C5 0F1A
    addwf   DPX+B2
01C6 6A1D
    movfp   AARG+B1,wreg
01C7 111B
    addwf   DPX+B3

    noadd4
    if i < 8
        rlcf   AARG+B1,W   ; rotate sign into carry bit
        rrcf   DPX+B3
        rrcf   DPX+B2
        rrcf   DPX+B1
    else
        rlcf   AARG+B1,W   ; rotate sign into carry bit
        rrcf   DPX+B3
        rrcf   DPX+B2
        rrcf   DPX+B1
        rrcf   DPX+B0
    endif

0005
    i = i+1

01CC 951E
    if i < 8
        btfss   BARG+B0,i   ; test low byte
    else
        btfss   BARG+B1,i-8  ; test high byte
    endif
```
Servo Control of a DC-Brush Motor

01CD C1D2  add5  goto  noadd5
01CE 6A1C  movfp  AARG+B0,wreg
01CF 0F1A  addwf  DPX+B2 ;add lsb
01D0 6A1D  movfp  AARG+B1,wreg
01D1 111B  addwfc  DPX+B3 ;add msb

noadd5

if i < 8

01D2 1A1D  rlcf  AARG+B1,W
01D3 191B  rrcf  DPX+B3
01D4 191A  rrcf  DPX+B2
01D5 1919  rrcf  DPX+B1

else

rlcf  AARG+B1,W   ; rotate sign into carry bit
rrcf  DPX+B3
rrcf  DPX+B2
rrcf  DPX+B1
rrcf  DPX+B0

endif

0006

i = i+1

01D6 961E  if i < 8
btfss  BARG+B0,i   ; test low byte
else
btfss  BARG+B1,i-8  ; test high byte
endif

01D7 C1DC  goto  noadd6
01D8 6A1C  movfp  AARG+B0,wreg
01D9 0F1A  addwf  DPX+B2 ;add lsb
01DA 6A1D  movfp  AARG+B1,wreg
01DB 111B  addwfc  DPX+B3 ;add msb

noadd6

if i < 8

01DC 1A1D  rlcf  AARG+B1,W
01DD 191B  rrcf  DPX+B3
01DE 191A  rrcf  DPX+B2
01DF 1919  rrcf  DPX+B1

else

rlcf  AARG+B1,W
rrcf  DPX+B3
rrcf  DPX+B2
rrcf  DPX+B1

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit

for i < 8, no meaningful
are in DPX+B0

rotate sign into carry bit
Servo Control of a DC-Brush Motor

```assembly
rrcf DPX+B0

endif

0007
i = i+1

01E0 971E
if i < 8
  btfss BARG+B0,i ; test low byte
else
  btfss BARG+B1,i-8 ; test high byte
endif

01E1 C1E6
add7
  goto noadd7

01E2 6A1C
01E3 0F1A
01E4 6A1D
01E5 111B

noadd7
if i < 8
01E6 1A1D
  rlcf AARG+B1,W ; rotate sign into carry bit
  rrcf DPX+B3 ; for i < 8, no meaningful
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0 ; are in DPX+B0
else
  rlcf AARG+B1,W ; rotate sign into carry bit
  rrcf DPX+B3
  rrcf DPX+B2
  rrcf DPX+B1
  rrcf DPX+B0
endif

0008
i = i+1

01EA 901F
if i < 8
  btfss BARG+B0,i ; test low byte
else
  btfss BARG+B1,i-8 ; test high byte
endif

01EB C1F0
add8
  goto noadd8

01EC 6A1C
01ED 0F1A
01EE 6A1D
01EF 111B

noadd8
if i < 8
```

© 1993 Microchip Technology Inc. DS00532B-page 47
Servo Control of a DC-Brush Motor

rlcf AARG+Bl,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

01F0 1A1D
01F1 191B
01F2 191A
01F3 1919
01F4 1918

rlcf AARG+Bl,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

endif

i = i+1

if i < 8
btfss BARG+BO,i ; test low byte
else

01F5 911F

btfss BARG+B1,i-8 ; test high byte
endif

01F6 C1FB

movfp AARG+B0,wreg
addwf DPX+B2 ; add lsb
movfp AARG+B1,wreg
addwf DPX+B3 ; add msb

noadd9

if i < 8

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

01FB 1A1D
01FC 191B
01FD 191A
01FE 1919
01FF 1918

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

endif

i = i+1

if i < 8
btfss BARG+BO,i ; test low byte
else
Servo Control of a DC-Brush Motor

0200 921F btffs BARG+B1,i-8 ; test high byte
0201 C206 endif
goto noadd10
0202 6A1C add10
0203 0F1A movfp AARG+B0,wreg
0204 6A1D addwf DPX+B2 ;add lsb
0205 111B movfp AARG+B1,wreg
addwf DPX+B3 ;add msb

noadd10

if i < 8

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

0206 1A1D rlcf AARG+B1,W ; rotate sign into carry bit
0207 191A rrcf DPX+B3
0208 191A rrcf DPX+B2
0209 1919 rrcf DPX+B1
020A 1918 rrcf DPX+B0

endif

0000B

i = i+1

if i < 8

btffs BARG+B0,i ; test low byte
else

020B 931F btffs BARG+B1,i-8 ; test high byte
020C C211 endif
goto noadd11
020D 6A1C add11
020E 0F1A movfp AARG+B0,wreg
020F 6A1D addwf DPX+B2 ;add lsb
0210 111B movfp AARG+B1,wreg
addwf DPX+B3 ;add msb

noadd11

if i < 8

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

0211 1A1D rlcf AARG+B1,W ; rotate sign into carry bit
0212 191B rrcf DPX+B3

© 1993 Microchip Technology Inc.

DS00532B-page 49
Servo Control of a DC-Brush Motor

0213 191A  rrcf  DPX+B2
0214 1919  rrcf  DPX+B1
0215 1918  rrcf  DPX+B0

endif

000C     i = i+1

if i < 8  btffss  BARG+B0,i ; test low byte
else

0216 941F  btffss  BARG+B1,i-8 ; test high byte
0217 C21C  goto  noadd12

add12

0218 6A1C  movfp  AARG+B0,wreg
0219 0F1A  addwf  DPX+B2 ; add lsb
021A 6A1D  movfp  AARG+B1,wreg
021B 111B  addwfc  DPX+B3 ; add msb

noadd12

if i < 8  rlcf  AARG+B1,W ; rotate sign into carry bit
            rrcf  DPX+B3 ; for i < 8, no meaningful
            rrcf  DPX+B2 ; are in DPX+B0
            rrcf  DPX+B1

else

021C 1A1D  rlcf  AARG+B1,W ; rotate sign into carry bit
021D 191B  rrcf  DPX+B3
021E 191A  rrcf  DPX+B2
021F 1919  rrcf  DPX+B1
0220 1918  rrcf  DPX+B0

endif

000D     i = i+1

if i < 8  btffss  BARG+B0,i ; test low byte
else

0221 951F  btffss  BARG+B1,i-8 ; test high byte
0222 C227  goto  noadd13

add13

0223 6A1C  movfp  AARG+B0,wreg
0224 0F1A  addwf  DPX+B2 ; add lsb
0225 6A1D  movfp  AARG+B1,wreg
0226 111B  addwfc  DPX+B3 ; add msb

noadd13

if i < 8  rlcf  AARG+B1,W ; rotate sign into carry bit
Servo Control of a DC-Brush Motor

```assembly
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

0227 1A1D
0228 191B
0229 191A
022A 1919
022B 1918

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

endif

i = i+1

if i < 8
btfss BARG+B0,i ; test low byte

else

btfss BARG+B1,i-8 ; test high byte

endif

goto noadd14

movfp AARG+B0,wreg
addwf DPX+B2 ;add lsb
movfp AARG+B1,wreg
addwfc DPX+B3 ;add msb

noadd14

if i < 8

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

0232 1A1D
0233 191B
0234 191A
0235 1919
0236 1918

rlcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0

endif

i = i+1

if SIGNED

0237 1A1D
0238 191B
0239 191A
023A 1919
023B 1918

rlcf AARG+B1,W ; since BARG is always made
rrcf DPX+B3 ; the last bit is known to be
rrcf DPX+B2
rrcf DPX+B1
rrcf DPX+B0
```

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

endif

return

 inclusion “traject.asm”;

Trajectory Generation

;*************************************************************************

; Trajectory Generation Routines

;*************************************************************************

;*************************************************************************

; NAME: doPreMove

; DESCRIPTION:

doPreMove:

CLR16 INTEGRAL

023D 2996
023E 2997

MOV24 NM0VVL,MOVVAL ; move buffer to MOVVAL

023F 6A5B
0240 4A5F
0241 6A5C
0242 4A60
0243 6A5D
0244 4A61

MOVFP NM0VVL+0,wreg ; get byte of NM0VVL into w
MOVFP NM0VVL+1,wreg ; move to MOVVAL(0)
MOVFP NM0VVL+2,wreg ; get byte of NM0VVL into w
MOVFP NM0VVL+3,wreg ; move to MOVVAL(1)
MOVFP NM0VVL+4,wreg ; get byte of NM0VVL into w
MOVFP NM0VVL+5,wreg ; move to MOVVAL(2)

0245 8F93
0246 8693
0247 8593
0248 6AC2
0249 4A94

bcf MOVSTAT,bit7 ; clear buffer flag
bsf MOVSTAT,bit6 ; set motion status flag
bsf MOVSTAT,bit5 ; set move in progress flag
movfp ONE,wreg
movfp wreg,MOVFLAG ; initialize MOVEFLAG to 1

024A 2951

CLR16 OPOSITION+B0 ; initialize buffers

MOV24 POSITION,OPOSITION+1

024B 6A55
024C 4A52
024D 6A56
024E 4A53
024F 6A57
0250 4A54

MOVFP POSITION+0,wreg ; get byte of POSITION into w
MOVFP wreg,OPOSITION+1+B0 ; move to OPOSITION+1(B0)
MOVFP POSITION+1,wreg ; get byte of POSITION into w
MOVFP wreg,OPOSITION+1+B1 ; move to OPOSITION+1(B1)
MOVFP POSITION+2,wreg ; get byte of POSITION into w
MOVFP wreg,OPOSITION+1+B2 ; move to OPOSITION+1(B2)

MOV32 OPOSITION,M0VPBUF

0251 6A51
0252 4A44
0253 6A52

MOVFP OPOSITION+0,wreg ; get byte of OPOSITION into w
MOVFP wreg,M0VPBUF+0 ; move to M0VPBUF(0)
MOVFP OPOSITION+1,wreg ; get byte of OPOSITION into w
Servo Control of a DC-Brush Motor

0254 4AA5 MOVFP wreg,MOVBUF+B1 ; move to MOVBUF(B1)
0255 6A53 MOVFP OPOSITION+B2,wreg ; get byte of OPOSITION into w
0256 4AA6 MOVFP wreg,MOVBUF+B2 ; move to MOVBUF(B2)
0257 6A54 MOVFP OPOSITION+B3,wreg ; get byte of OPOSITION into w
0258 4AA7 MOVFP wreg,MOVBUF+B3 ; move to MOVBUF(B3)

0259 2995 clrf SATFLAG
CLR16 MOVTIME ; clear move times
025A 29E7 CLRF MOVTIME+B0
025B 29E8 CLRF MOVTIME+B1

CLR16 T1 ; 0 used as flag for no maximum
025C 296A CLRF T1+B0
025D 296B CLRF T1+B1

CLR16 T2
025E 296C CLRF T2+B0
025F 296D CLRF T2+B1

CLR16 TAU
0260 296E CLRF TAU+B0
0261 296F CLRF TAU+B1

CLR32 MOVDEL ; clear move discretization error
0262 2990 CLRF MOVDEL+B0
0263 2991 CLRF MOVDEL+B1
0264 2992 CLRF MOVDEL+B2
0265 2993 CLRF MOVDEL+B3

CLR16 PH2FLAT ; clear phase 2 flat counter
0266 29B4 CLRF PH2FLAT+B0
0267 29B5 CLRF PH2FLAT+B1

0268 3391 tstfsz MODETYPE
0269 C2C5 goto vmode

pmode MOVFP24 MOVVAL,TMP
026A 785F MOVFP MOVVAL+B0,TMP+B0 ; move MOVVAL(B0) to TMP(B0)
026B 7960 MOVFP MOVVAL+B1,TMP+B1 ; move MOVVAL(B1) to TMP(B1)
026C 7A61 MOVFP MOVVAL+B2,TMP+B2 ; move MOVVAL(B2) to TMP(B2)

026D 971A btfsz TMP+B2,MSB
026E C276 goto mvpos

NEG24 TMP
026F 1318 COMF TMP+B0
0270 1319 COMF TMP+B1
0271 131A COMF TMP+B2
0272 290A CLRF wreg
0273 1518 INCF TMP+B0
0274 1119 ADDWF TMP+B1
0275 111A ADDWF TMP+B2
Servo Control of a DC-Brush Motor

; calculate abs(MOVVAL) - 3
; do immediate move if negative
clrf MOVTMP+BO
clrf MOVTMP+B1
clrf MOVTMP+B2

; get lowest byte of MOVTMP
SUBWF TMP+B0
; sub lowest byte of TMP, save
SUBWF TMP+B1
; get 2nd byte of MOVTMP into
SUBWF TMP+B2
; sub 2nd byte of TMP, save in
SUBWF TMP+B3
; get 3rd byte of MOVTMP into
SUBWF TMP+B4
; sub 3rd byte of TMP, save in
SUBWF TMP+B5

; check for zero move
btfs TMP+B2,MSB
goto nonzero
; set servoflag to restore
setf SERVOFLAG
clr MOVTMP
bcf MOVTMP+BIT5
bcf MOVTMP+BIT6

ADD + MOVVAL, POSITION
; get lowest byte of MOVVAL
ADD + MOVVAL+BIT5
; get 2nd byte of MOVVAL into
ADD + MOVVAL+BIT6
; get 3rd byte of MOVVAL into
ADD + MOVVAL+BIT7
; get 4th byte of MOVVAL into
ADD + MOVVAL+BIT8
; get 5th byte of MOVVAL into
ADD + MOVVAL+BIT9
; get 6th byte of MOVVAL into
ADD + MOVVAL+BIT10
; get 7th byte of MOVVAL into
ADD + MOVVAL+BIT11
; get 8th byte of MOVVAL into
ADD + MOVVAL+BIT12
; get 9th byte of MOVVAL into
ADD + MOVVAL+BIT13
; get 10th byte of MOVVAL into
ADD + MOVVAL+BIT14
; get 11th byte of MOVVAL into
ADD + MOVVAL+BIT15
; get 12th byte of MOVVAL into
ADD + MOVVAL+BIT16
; get 13th byte of MOVVAL into
ADD + MOVVAL+BIT17
; get 14th byte of MOVVAL into
ADD + MOVVAL+BIT18
; get 15th byte of MOVVAL into
ADD + MOVVAL+BIT19
; get 16th byte of MOVVAL into
ADD + MOVVAL+BIT20
; get 17th byte of MOVVAL into
ADD + MOVVAL+BIT21
; get 18th byte of MOVVAL into
ADD + MOVVAL+BIT22
; get 19th byte of MOVVAL into
ADD + MOVVAL+BIT23
; get 20th byte of MOVVAL into
ADD + MOVVAL+BIT24
; get 21st byte of MOVVAL into
ADD + MOVVAL+BIT25
; get 22nd byte of MOVVAL into
ADD + MOVVAL+BIT26
; get 23rd byte of MOVVAL into
ADD + MOVVAL+BIT27
; get 24th byte of MOVVAL into
ADD + MOVVAL+BIT28
; get 25th byte of MOVVAL into
ADD + MOVVAL+BIT29
; get 26th byte of MOVVAL into
ADD + MOVVAL+BIT30
; get 27th byte of MOVVAL into
ADD + MOVVAL+BIT31
; get 28th byte of MOVVAL into
ADD + MOVVAL+BIT32
; get 29th byte of MOVVAL into
ADD + MOVVAL+BIT33
; get 30th byte of MOVVAL into
ADD + MOVVAL+BIT34
; get 31th byte of MOVVAL into
ADD + MOVVAL+BIT35
; get 32nd byte of MOVVAL into
ADD + MOVVAL+BIT36
; get 33rd byte of MOVVAL into
ADD + MOVVAL+BIT37
; get 34th byte of MOVVAL into
ADD + MOVVAL+BIT38
; get 35th byte of MOVVAL into
ADD + MOVVAL+BIT39
; get 36th byte of MOVVAL into
ADD + MOVVAL+BIT40
; get 37th byte of MOVVAL into
ADD + MOVVAL+BIT41
; get 38th byte of MOVVAL into
ADD + MOVVAL+BIT42
; get 39th byte of MOVVAL into
ADD + MOVVAL+BIT43
; get 40th byte of MOVVAL into
ADD + MOVVAL+BIT44
; get 41st byte of MOVVAL into
ADD + MOVVAL+BIT45
; get 42nd byte of MOVVAL into
ADD + MOVVAL+BIT46
; get 43rd byte of MOVVAL into
ADD + MOVVAL+BIT47
; get 44th byte of MOVVAL into
ADD + MOVVAL+BIT48
; get 45th byte of MOVVAL into
ADD + MOVVAL+BIT49
; get 46th byte of MOVVAL into
ADD + MOVVAL+BIT50
; get 47th byte of MOVVAL into
ADD + MOVVAL+BIT51
; get 48th byte of MOVVAL into
ADD + MOVVAL+BIT52
; get 49th byte of MOVVAL into
ADD + MOVVAL+BIT53
; get 50th byte of MOVVAL into
ADD + MOVVAL+BIT54
; get 51st byte of MOVVAL into
ADD + MOVVAL+BIT55
; get 52nd byte of MOVVAL into
ADD + MOVVAL+BIT56
; get 53rd byte of MOVVAL into
ADD + MOVVAL+BIT57
; get 54th byte of MOVVAL into
ADD + MOVVAL+BIT58
; get 55th byte of MOVVAL into
ADD + MOVVAL+BIT59
; get 56th byte of MOVVAL into
ADD + MOVVAL+BIT60
; get 57th byte of MOVVAL into
ADD + MOVVAL+BIT61
; get 58th byte of MOVVAL into
ADD + MOVVAL+BIT62
; get 59th byte of MOVVAL into
ADD + MOVVAL+BIT63
Servo Control of a DC-Brush Motor

02A2 4A9E  MOVFF wreg,A+B2 ; move to A(B2)
02A3 290A  clrf wreg
02A4 3269  cpfgst MOVSIGN
02A5 C2B8  goto minc

NEG32 V

02A6 13A0  COMF V+B0
02A7 13A1  COMF V+B1
02A8 13A2  COMF V+B2
02A9 13A3  COMF V+B3
02AA 290A  CLRF wreg
02AB 15A0  INCF V+B0
02AC 11A1  ADDWFC V+B1
02AD 11A2  ADDWFC V+B2
02AE 11A3  ADDWFC V+B3

NEG32 A

02AF 139C  COMF A+B0
02B0 139D  COMF A+B1
02B1 139E  COMF A+B2
02B2 139F  COMF A+B3
02B3 290A  CLRF wreg
02B4 159C  INCF A+B0
02B5 119D  ADDWFC A+B1
02B6 119E  ADDWFC A+B2
02B7 119F  ADDWFC A+B3

minc clrf HMOWAL+B0 ; evaluate MOVVAL/2
MOV24 MOVVAL,HMOVVAL+B1

02B8 2963  ; get byte of MOVVAL into w
MOVFP MOVVAL+B0,wreg ; move to HMOVVAL+B1(B0)
02B9 6A5F  MOVFP wreg,HMOVVAL+B1+B0
02BA 4A64  MOVFP MOVVAL+B1,wreg ; get byte of MOVVAL into w
02BB 6A60  MOVFP wreg,HMOVVAL+B1+B1
02BC 4A65  MOVFP MOVVAL+B2,wreg ; move to HMOVVAL+B1(B1)
02BD 6A61  MOVFP MOVVAL+B2,wreg
02BE 4A66  MOVFP wreg,HMOVVAL+B1+B2

RRC32 HMOVVAL ; half move in QB

02BF 1A66  RLCF HMOVVAL+B3,W ; move sign into carry bit
02C0 1966  RRCF HMOVVAL+B3
02C1 1965  RRCF HMOVVAL+B2
02C2 1964  RRCF HMOVVAL+B1
02C3 1963  RRCF HMOVVAL+B0
02C4 C2FE  goto modeready

vmode

02C5 9F91  btfs MOCTYPE,MSB ; is it torque move?
02C6 C306  goto tmode
02C7 2966  clrf HMOVVAL+B3 ; compute final minus initial
MOV24 MOVVAL,HMOVVAL

02C8 6A5F  MOVFP MOVVAL+B0,wreg ; get byte of MOVVAL into w
02C9 4A63  MOVFP wreg,HMOVVAL+B0 ; move to HMOVVAL(B0)
02CA 6A60  MOVFP MOVVAL+B1,wreg ; get byte of MOVVAL into w
02CB 4A64  MOVFP wreg,HMOVVAL+B1 ; move to HMOVVAL(B1)
02CC 6A61  MOVFP MOVVAL+B2,wreg ; get byte of MOVVAL into w
02CD 4A65  MOVFP wreg,HMOVVAL+B2 ; move to HMOVVAL(B2)
Servo Control of a DC-Brush Motor

02CE 9F61 btfsc MOVVAL+B2, MSB
02CF 2B66 setf MOVVAL+B3

SUB3 MOVVBUF, MOVVAL

0200 6A68 MOVFP MOVVBUF+B0, wreg
0201 0563 SUBWF MOVVAL+B0
0202 6A9A MOVFP MOVVBUF+B1, wreg
0203 0364 SUBWF MOVVAL+B1
0204 6AAB MOVFP MOVVBUF+B2, wreg
0205 0365 SUBWF MOVVAL+B2
0206 6AEC MOVFP MOVVBUF+B3, wreg
0207 0366 SUBWF MOVVAL+B3

get lowest byte of MOVVBUF into w
get 2nd byte of MOVVBUF into w
get 3rd byte of MOVVBUF into w
get 4th byte of MOVVBUF into w
get lowest byte of MOVVAL, save in
get 2nd byte of MOVVAL, save in
get 3rd byte of MOVVAL, save in
get 4th byte of MOVVAL, save in

0208 6A66 movf MOVVAL+B3, wreg
0209 B580 andlw 0x80
020A 4A69 movpf wreg, MOVSIGN

020B 29A3 clrf V+B3
MOV24 VL, V

create appropriate velocity and
acceleration limits from move sign

020C 6A20 MOVFP VL+B0, wreg
020D 4AA0 MOVFP wreg, V+B0
020E 6A21 MOVFP VL+B1, wreg
020F 4A9C MOVFP wreg, V+B1
0210 6A22 MOVFP VL+B2, wreg
0211 4A90 MOVFP wreg, V+B2
0212 6A23 MOVFP VL+B3, wreg
0213 4A9E MOVFP wreg, V+B3

get byte of VL into w
move to V(B0)
get byte of VL into w
move to V(B1)
get byte of VL into w
move to V(B2)

0214 299F clrf A+B3
MOV24 AL, A

get byte of AL into w
move to A(B0)
get byte of AL into w
move to A(B1)
get byte of AL into w
move to A(B2)

0215 6A23 MOVFP AL+B0, wreg
0216 4A99 MOVFP wreg, A+B0
0217 6A24 MOVFP AL+B1, wreg
0218 4A95 MOVFP wreg, A+B1
0219 6A25 MOVFP AL+B2, wreg
021A 4A9E MOVFP wreg, A+B2

get byte of AL into w
move to A(B0)
get byte of AL into w
move to A(B1)
get byte of AL into w
move to A(B2)

021B 290A clrf wreg
021C 3269 cpfsgt MOVSIGN
021D C2FE goto modeready

NEG32 V

021E 13A0 COMP V+B0
021F 13A1 COMP V+B1
0220 13A2 COMP V+B2
0221 13A3 COMP V+B3
0222 290A CLRF wreg
0223 15A0 INCFC V+B0
0224 11A1 ADDWF V+B1
0225 11A2 ADDWF V+B2
0226 11A3 ADDWF V+B3

NEG32 A

0227 139C COMP A+B0
0228 139D COMP A+B1
0229 139E COMP A+B2
022A 139F COMP A+B3
022B 290A CLRF wreg
022C 159C INCFC A+B0
022D 119D ADDWF A+B1
022E 119E ADDWF A+B2
022F 119F ADDWF A+B3

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

modo

02FE 2962
02FF 9F61
0300 2B62
0301 2B90

clrf MOVVAL+B3
btfs MOVVAL+B2,MSB
setf SERVOFLAG

if _PICMASTER_DEBUG

;****************************************************************

; For PICMASTER Debug/servo tuning purposes only Purposes Only
;

;****************************************************************

testCapCount
movfp CAPCOUNT+BO,wreg
iorwf CAPCOUNT+B1,\n
movfp wreg,CAPFLAG

;*************************************************************************

endif

0305 0002

return

; torque/voltage mode

MOV16 MOVVAL+B1,YPWM

0306 6A60
0307 0188
0308 6A61
0309 0189

MOVFP MOVVAL+B1+BO,wreg
MOVWF YPWM+BO
MOVFP MOVVAL+B1+Bl,wreg
MOVWF YPWM+Bl

clrf SERVOFLAG

call doTorque

clrf MOVFLAG

bcf MOVSTAT,bit5

if _PICMASTER_DEBUG

goto testCapCount

else

return

endif

;**************************************************************************

NAME: doMove

DESCRIPTION: In position mode, trapezoidal moves are performed. Phase1
and phase2 respectively, are the periods for the first and
second halves of the move. The move time is defined as zero
at the beginning of the move,T2 is the time at half the
move, T1 is the time when c
begins,(the region of constant velocity reduces to a point
in the case where maximum speed is not realized, and the
trapezoidal move degenerates into a triangular move,

together with T1=T2), and TAU is the total time of the move.
The accelerations are +-AL or 0.

© 1993 Microchip Technology Inc. DS00532B-page 57
triangle speed                              trapezoidal speed

\[
\begin{array}{cccccc}
\text{\(0\)} & \text{\(T_1=T_2\)} & \tau & \text{\(0\)} & \text{\(T_1\)} & \text{\(T_2\)} & \tau
\end{array}
\]\n
Let \(x\) denote the undershoot and \(y\) the overshoot commanded at adjacent sample times as half the move is crossed.

In the case of a triangular move, the discretization error is given by

\[
\text{error} = \min (2x,2y)
\]

For a trapezoidal move, the discretization error is

\[
\text{error} = \min (2x,y-x) \leq 0.5 \times (\text{maximum commanded speed})
\]

This discretization error is resolved in the final sample time of the move by executing a step to the final position at zero speed. The method employed here the best possible performance with regard to discretization error without dynamically modifying velocity and acceleration limits.

In velocity mode, ramp moves are performed.

/ final velocity
/ / initial velocity /
/ / / /
/ / / /
/ / / /

In velocity mode, ramp moves are performed.

\[
\begin{array}{cccc}
\text{\(0\)} & \tau
\end{array}
\]\n
doMove

INC16 MOVTIME ; increment move time

030F 290A CLRF wreg
0310 1567 INCF (MOVTIME)+B0
0311 1168 ADDWFC (MOVTIME)+B1

0312 E468 call doPosVel ; evaluate iterative equations
0313 3391 tstfsz MODETYPE
0314 C429 goto vmove
0315 6AC2 movfp ONE,wreg ; test if in phasel
0316 3194 cpfseq MVFPAG
0317 C3DB goto phase2
0318 326C MOVFP32 MOVDEL,MVTMP ; save previous discretization error

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

```
0318 7CB0 MOVFP MOVDEL+B0,MOVTMP+B0 ; move MOVDEL(B0) to MOVTMP(B0)
0319 7DB1 MOVFP MOVDEL+B1,MOVTMP+B1 ; move MOVDEL(B1) to MOVTMP(B1)
031A 7EB2 MOVFP MOVDEL+B2,MOVTMP+B2 ; move MOVDEL(B2) to MOVTMP(B2)
031B 7FB3 MOVFP MOVDEL+B3,MOVTMP+B3 ; move MOVDEL(B3) to MOVTMP(B3)

MOV32 OPOSITION,MOVDEL ; test if half move
031C 6A51 MOVFP OPOSITION+B0,wreg ; get byte of OPOSITION into w
031D 4AB0 MOVPF wreg,MOVDEL+B0 ; move to MOVDEL(B0)
031E 6A52 MOVFP OPOSITION+B1,wreg ; get byte of OPOSITION into w
031F 4AB1 MOVPF wreg,MOVDEL+B1 ; move to MOVDEL(B1)
0320 6A53 MOVFP OPOSITION+B2,wreg ; get byte of OPOSITION into w
0321 4AB2 MOVPF wreg,MOVDEL+B2 ; move to MOVDEL(B2)
0322 6A54 MOVFP OPOSITION+B3,wreg ; get byte of OPOSITION into w
0323 4AB3 MOVPF wreg,MOVDEL+B3 ; move to MOVDEL(B3)

ADD32 HMOVVAL,MOVDEL
0324 6A63 MOVFP HMOVVAL+B0,wreg ; get lowest byte of HMOVVAL into w
0325 0F80 ADDWF MOVDEL+B0 ; add lowest byte of MOVDEL, save in
0326 6A64 MOVFP HMOVVAL+B1,wreg ; get 2nd byte of HMOVVAL into w
0327 11B1 ADDWF MOVDEL+B1 ; add 2nd byte of MOVDEL, save in
0328 6A65 MOVFP HMOVVAL+B2,wreg ; get 3rd byte of HMOVVAL into w
0329 11B2 ADDWF MOVDEL+B2 ; add 3rd byte of MOVDEL, save in
032A 6A66 MOVFP HMOVVAL+B3,wreg ; get 4th byte of HMOVVAL into w
032B 11B3 ADDWF MOVDEL+B3 ; add 4th byte of MOVDEL, save in

SUB32 MOVP8UF,MOVDEL
032C 6AA4 MOVFP MOVP8UF+B0,wreg ; get lowest byte of MOVPBUF into w
032D 0F80 SUBWF MOVDEL+B0 ; sub lowest byte of MOVDEL, save in
032E 6AA5 MOVFP MOVP8UF+B1,wreg ; get 2nd byte of MOVPBUF into w
032F 03B1 SUBWF MOVDEL+B1 ; sub 2nd byte of MOVDEL, save in
0330 6AA6 MOVFP MOVP8UF+B2,wreg ; get 3rd byte of MOVPBUF into w
0331 03B2 SUBWF MOVDEL+B2 ; sub 3rd byte of MOVDEL, save in
0332 6AA7 MOVFP MOVP8UF+B3,wreg ; get 4th byte of MOVPBUF into w
0333 03B3 SUBWF MOVDEL+B3 ; sub 4th byte of MOVDEL, save in

0334 9769 btfs MOVSIGN,MSB
0335 C33F goto mpos1
NEG32 MOVDEL

0336 13B0 COMP MOVDEL+B0
0337 13B1 COMP MOVDEL+B1
0338 13B2 COMP MOVDEL+B2
0339 13B3 COMP MOVDEL+B3
033A 290A CLRWF wreg
033B 15B0 INCWF MOVDEL+B0
033C 11B1 ADDWF MOVDEL+B1
033D 11B2 ADDWF MOVDEL+B2
033E 11B3 ADDWF MOVDEL+B3

mpos1 btfs MOVDEL+B3,MSB
0340 C3A5 goto speedup ; continue to speed up if in

TFSZ16 T1 ; if T1=0, maximum velocity not
0341 6A6A MOVFP T1+B0,wreg
0342 086B IORWF T1+B1,W
0343 330A TSTFSZ wreg

; reached,
```
Servo Control of a DC-Brush Motor

0344 C378  goto t2net1 ; has been set in speedup
0345 139C  NEG32 A ; negate A for speeddown
0346 139D  COMP A+B0
0347 139E  COMP A+B1
0348 139F  COMP A+B2
0349 290A  CLRF wreg
034A 159C  INCF A+B0
034B 119D  ADDWF A+B1
034C 119E  ADDWF A+B2
034D 119F  ADDWF A+B3

ADD32 MOVDEL,MOVTMP ; test x-y < 0
034E 6AB0  MOVFP MOVDEL+B0,wreg ; get lowest byte of MOVDEL into w
034F 0F1C  ADDWF MOVTMP+B0 ; add lowest byte of MOVTMP, save in w
0350 6AB1  MOVFP MOVDEL+B1,wreg ; get 2nd byte of MOVDEL into w
0351 111D  ADDWF MOVTMP+B1 ; add 2nd byte of MOVTMP, save in w
0352 6AB2  MOVFP MOVDEL+B2,wreg ; get 3rd byte of MOVDEL into w
0353 111E  ADDWF MOVTMP+B2 ; add 3rd byte of MOVTMP, save in w
0354 6AB3  MOVFP MOVDEL+B3,wreg ; get 4th byte of MOVDEL into w
0355 111F  ADDWF MOVTMP+B3 ; add 4th byte of MOVTMP, save in w

0356 971F  btfss MOVTMP+B3,MSB ; if new discretization error larger,
0357 C36E  goto trick ; backup to define T2, otherwise ok
0358 2B6C  setf T2+B0 ; set T2=-1 for backup
0359 2B6D  setf T2+B1
035A 139C  NEG32 A again for speeddown
035B 139D  COMP A+B0
035C 139E  COMP A+B1
035D 139F  COMP A+B2
035E 290A  CLRF wreg
035F 159C  INCF A+B0
0360 119D  ADDWF A+B1
0361 119E  ADDWF A+B2
0362 119F  ADDWF A+B3

0363 E48A  call undoPosVel ; negate A to undo
0364 139C  COMP A+B0
0365 139D  COMP A+B1
0366 139E  COMP A+B2
0367 139F  COMP A+B3
0368 290A  CLRF wreg
0369 159C  INCF A+B0
036A 119D  ADDWF A+B1
036B 119E  ADDWF A+B2
036C 119F  ADDWF A+B3

036D E468  call doPosVel and reevaluate iterative equations
036E 6A67  ADD16 MOVTIME, T2 add time to T2
036F 0F6C  MOVFP MOVTIME+B0,wreg ; get lowest byte of MOVTIME into w
0370 6A68  ADDWF T2+B0 ; add lowest byte of T2, save in w
0371 116D  MOVFP MOVTIME+B1,wreg ; get 2nd byte of MOVTIME into w
0372 116E  ADDWF T2+B1 ; add 2nd byte of T2, save in T2(B1)
Servo Control of a DC-Brush Motor

MOV16 T2, T1

0372 6A6C MOVFP T2+BO, wreg ; get byte of T2 into w
0373 016A MOVWF T1+BO ; move to T1(BO)
0374 6A6D MOVFP T2+B1, wreg ; get byte of T2 into w
0375 016B MOVWF T1+B1 ; move to T1(B1)

0376 1594 incf MOVFLAG ; increment move flag for
0377 C3CE goto mvok ; execute last phase1 move

; t2net1
0378 2B6C seft T2+BO ; set T2=1 for backup
0379 2B6D seft T2+B1 ; set time to T2

ADD16 MOVTIME, T2

037A 6A67 MOVFP MOVTIME+BO, wreg ; get lowest byte of MOVTIME
037B 0F6C ADDWF T2+BO ; add lowest byte of T2, save
037C 6A68 MOVFP MOVTIME+B1, wreg ; get 2nd byte of MOVTIME into
037D 116D ADDWFC T2+B1 ; add 2nd byte of T2, save in

MOVFP32 MOVTMP, TMP ; test if 3x-y < 0

037E 781C MOVFP MOVTMP+BO, TMP+BO ; move MOVTMP(BO) to TMP(BO)
037F 791D MOVFP MOVTMP+B1, TMP+B1 ; move MOVTMP(B1) to TMP(B1)
0380 7A1E MOVFP MOVTMP+B2, TMP+B2 ; move MOVTMP(B2) to TMP(B2)
0381 7B1F MOVFP MOVTMP+B3, TMP+B3 ; move MOVTMP(B3) to TMP(B3)

RLC32 MOVTMP

0382 8804 BCF _carry
0383 1B1C RLCF MOVTMP+B0
0384 1B1D RLCF MOVTMP+B1
0385 1B1E RLCF MOVTMP+B2
0386 1B1F RLCF MOVTMP+B3

ADD32 TMP, MOVTMPS

0387 6A18 MOVFP TMP+BO, wreg ; get lowest byte of TMP into
0388 0F1C ADDWF MOVTMPS+B0 ; add lowest byte of MOVTMPS,
0389 6A19 MOVFP TMP+B1, wreg ; get 2nd byte of TMP into w
038A 111D ADDWFC MOVTMPS+B1 ; add 2nd byte of MOVTMPS, save
038B 6A1A MOVFP TMP+B2, wreg ; get 3rd byte of TMP into w
038C 111E ADDWFC MOVTMPS+B2 ; add 3rd byte of MOVTMPS, save
038D 6A1B MOVFP TMP+B3, wreg ; get 4th byte of TMP into w
038E 111F ADDWFC MOVTMPS+B3 ; add 4th byte of MOVTMPS, save

ADD32 MOVDEL, MOVTMPS

038F 6A80 MOVFP MOVDEL+BO, wreg ; get lowest byte of MOVDEL
0390 0F1C ADDWF MOVTMPS+B0 ; add lowest byte of MOVTMPS,
0391 6A81 MOVFP MOVDEL+B1, wreg ; get 2nd byte of MOVDEL into
0392 111D ADDWFC MOVTMPS+B1 ; add 2nd byte of MOVTMPS, save
0393 6A82 MOVFP MOVDEL+B2, wreg ; get 3rd byte of MOVDEL into
0394 111E ADDWFC MOVTMPS+B2 ; add 3rd byte of MOVTMPS, save
0395 6A83 MOVFP MOVDEL+B3, wreg ; get 4th byte of MOVDEL into
0396 111F ADDWFC MOVTMPS+B3 ; add 4th byte of MOVTMPS, save
Servo Control of a DC-Brush Motor

0397 971F  btfss  MOVTMP+B3,MSB  ; if new discretization error
0398  C39B  goto  trapok  ; take one more flat step
0399  2BB4  setf  PH2FLAT+B0
039A  2BB5  setf  PH2FLAT+B1

trapok
ADD16  T2,PH2FLAT

039B  6A6C  MOVFP  T2+B0,wreg  ; get lowest byte of T2 into w
039C  0FB4  ADDWF  PH2FLAT+B0  ; add lowest byte of PH2FLAT,
039D  6A6D  MOVFP  T2+B1,wreg  ; get 2nd byte of T2 into w
039E  0FB5  ADDWF  PH2FLAT+B1  ; add 2nd byte of PH2FLAT,

SUB16  T1,PH2FLAT

039F  6A6A  MOVFP  T1+B0,wreg  ; get lowest byte of T1 into w
03A0  0FB4  SUBWF  PH2FLAT+B0  ; sub lowest byte of PH2FLAT,
03A1  6A6B  MOVFP  T1+B1,wreg  ; get 2nd byte of T1 into w
03A2  0FB5  SUBWF  PH2FLAT+B1  ; sub 2nd byte of PH2FLAT,

03A3  1594  incf  MOVFLAG  ; increment move flag for
03A4  C3CE  goto  mvok  ; execute last phase move

speedup
MOVFP32  V,MOVTP

03A5  7CA0  MOVFP  V+B0,MOVTMP+B0  ; move V(B0) to MOVTMP(B0)
03A6  7DA1  MOVFP  V+B1,MOVTMP+B1  ; move V(B1) to MOVTMP(B1)
03A7  7EA2  MOVFP  V+B2,MOVTMP+B2  ; move V(B2) to MOVTMP(B2)
03A8  7FA3  MOVFP  V+B3,MOVTMP+B3  ; move V(B3) to MOVTMP(B3)

SUB32  MOVVBUF,MOVTP

03A9  6AA8  MOVFP  MOVVBUF+B0,wreg  ; get lowest byte of MOVVBUF
03AA  051C  SUBWF  MOVTMP+B0  ; sub lowest byte of MOVTMP,
03AB  6AA9  MOVFP  MOVVBUF+B1,wreg  ; get 2nd byte of MOVVBUF into
03AC  031D  SUBWF  MOVTMP+B1  ; sub 2nd byte of MOVTMP, save
03AD  6AAA  MOVFP  MOVVBUF+B2,wreg  ; get 3rd byte of MOVVBUF into
03AE  031E  SUBWF  MOVTMP+B2  ; sub 3rd byte of MOVTMP, save
03AF  6AB8  MOVFP  MOVVBUF+B3,wreg  ; get 4th byte of MOVVBUF into
03B0  031F  SUBWF  MOVTMP+B3  ; sub 4th byte of MOVTMP, save

in
03B1  9769  btfs  MOVSIGN,MSB
03B2  C38C  goto  mpos

NEG32  MOVTP

03B3  131C  COMPF  MOVTMP+B0
03B4  131D  COMPF  MOVTMP+B1
03B5  131E  COMPF  MOVTMP+B2
03B6  131F  COMPF  MOVTMP+B3
03B7  290A  CCLR  wreg
03B8  151C  INCNF  MOVTMP+B0
03B9  111D  ADWFPC  MOVTMP+B1
03BA  111E  ADWFPC  MOVTMP+B2
03BB  111F  ADWFPC  MOVTMP+B3

mpos
03BC  971F  btfs  MOVTMP+B3,MSB  ; if not, execute move
03BD  C3CE  goto  mvok

TFS216  T1  ; if so, check to see if T1

03BE  6A6A  MOVFP  T1+B0,wreg
03BF  086B  IORWF  T1+B1,W
Servo Control of a DC-Brush Motor

03C0 330A
	TSTFSZ wreg

03C1 C3CE
go to mvok
03C2 E48A
call undoPosVel

CLR32 A

03C3 299C
CLR A+B0
03C4 299D
CLR A+B1
03C5 299E
CLR A+B2
03C6 299F
CLR A+B3

03C7 E468
call doPosVel
03C8 2B6A
setf T1+B0
03C9 2B6B
setf T1+B1

ADD16 MOVTIE, T1

03CA 6A67
MOVFP MOVTIME+B0, wreg

03CB OF6A
ADVF T1+B0
03CC 6A68
MOVFP MOVTIME+B1, wreg

03CD 116B
ADVFC T1+B1

03CE 6AA5
MOVFP MOVBUFF+B1+B0, wreg
03CF 4A55
MOVFP wreg,POSITION+B0
03D0 6AA6
MOVFP MOVBUFF+B1+B1, wreg
03D1 4A56
MOVFP wreg, POSITION+B1
03D2 6AA7
MOVFP MOVBUFF+B1+B2, wreg
03D3 4A57
MOVFP wreg, POSITION+B2

MOV24 MOVBUFF+B0, VELOCITY

03D4 6AA8
MOVFP MOVBUFF+B0+B0, wreg
03D5 4A58
MOVFP wreg, VELOCITY+B0
03D6 6AA9
MOVFP MOVBUFF+B0+B1, wreg
03D7 4A59
MOVFP wreg, VELOCITY+B1
03D8 6AAA
MOVFP MOVBUFF+B0+B2, wreg
03D9 4A5A
MOVFP wreg, VELOCITY+B2

MOV24 MOVVBUF, POSITION

03DA 0002
return

phase2

TFSZ16 PH2FLAT

03DB 6AB4
MOVFP PH2FLAT+B0, wreg
03DC 08B5
IORWF PH2FLAT+B1, W
03DD 330A
TSTFSZ wreg

03DE C3FF
go to flat

TFSZ32 MOVVBUF

03DF 6AA8
MOVFP MOVVBUF+B0, wreg
03E0 08A9
IORWF MOVVBUF+B1, W
03E1 08AA
IORWF MOVVBUF+B2, W
03E2 08AB
IORWF MOVVBUF+B3, W
03E3 330A
TSTFSZ wreg

03E4 C41C
go to mready

; already been set
; if not, backup and redo
; equations, resulting in an

; maximum speed <= VL
; evaluate T1

; get lowest byte of MOVTIME
; add lowest byte of T1, save
; get 2nd byte of MOVTIME into
; add 2nd byte of T1, save in

; move Q8 calculated position
; get byte of MOVBUFF+B1 into
; move to POSITION (BO)
; get byte of MOVBUFF+B1 into
; move to POSITION (B1)
; get byte of MOVBUFF+B1 into
; move to POSITION (B2)

; move Q0 calculated velocity
; get byte of MOVBUFF+B0 into
; move to VELOCITY (B0)
; get byte of MOVBUFF+B0 into
; move to VELOCITY (B1)
; get byte of MOVBUFF+B0 into
; move to VELOCITY (B2)

; is flat section finished?

; is velocity zero?

; if not, execute move

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

03E5 2994        clrfr MOVFLAG        ; if so, clear MOVFLAG
03E6 8E93        bcf    MOVSTAT,bit6     ; clear motion status flag
03E7 8D93        bcf    MOVSTAT,bit5     ; clear move in progress flag
CLR32 A
03E8 299C        CLRF A+B0
03E9 299D        CLRF A+B1
03EA 299E        CLRF A+B2
03EB 299F        CLRF A+B3

MOV16 MOVTIME,TAU
03EC 6A67        MOVFP MOVTIME+BO,wreg    ; get byte of MOVTIME into w
03ED 016E        MOVWF TAU+BO
03EE 6A68        MOVFP MOVTIME+Bl,wreg    ; get byte of MOVTIME into w
03EF 016F        MOVWF TAU+Bl

MOV32 OPOSITION,MOVPBUF
03F0 6A51        MOVFP OPOSITION+BO,wreg  ; execute last move to P(O)+MOVPVAL
03F1 4A4A        MOVFP wreg,MOVPBUF+BO    ; move to MOVPBUF(B0)
03F2 6A52        MOVFP OPOSITION+Bl,wreg  ; get byte of OPOSITION into w
03F3 4A45        MOVFP wreg,MOVPBUF+Bl    ; move to MOVPBUF(B1)
03F4 6A53        MOVFP OPOSITION+B2,wreg  ; get byte of OPOSITION into w
03F5 4A46        MOVFP wreg,MOVPBUF+B2    ; move to MOVPBUF(B2)
03F6 6A54        MOVFP OPOSITION+B3,wreg  ; get byte of OPOSITION into w
03F7 4A47        MOVFP wreg,MOVPBUF+B3    ; move to MOVPBUF(B3)

ADD24 MOVPVAL,MOVPBUF+Bl
03F8 6A5F        MOVFP MOVPVAL+BO,wreg   ; get lowest byte of MOVPVAL into w
03F9 0F5A        ADDWF MOVPBUF+Bl+BO    ; add lowest byte of MOVPBUF+Bl, save
03FA 6A60        MOVFP MOVPVAL+Bl,wreg   ; get 2nd byte of MOVPVAL into w
03FB 11A6        ADDWF MOVPBUF+Bl+Bl    ; add 2nd byte of MOVPBUF+Bl, save in
03FC 6A61        MOVFP MOVPVAL+B2,wreg   ; get 3rd byte of MOVPVAL into w
03FD 11A7        ADDWF MOVPBUF+B2+B2    ; add 3rd byte of MOVPBUF+B2, save in

03FE C41C        goto mready

ADD16 MOVTMP,PH2FLAT
0400 2B1C        setf MOVTMP+BO          ; decrement by one use DEC16
0401 2B1D        setf MOVTMP+Bl

TFSZ16 PH2FLAT
0402 0F84        ADDWF PH2FLAT+BO
0403 6A1D        MOVFP MOVTMP+Bl,wreg    ; get lowest byte of MOVTMP into w
0404 11B5        ADDWF PH2FLAT+B1

0405 6A84        MOVFP PH2FLAT+BO,wreg  ; add lowest byte of PH2FLAT, save in
0406 0885        IORWF PH2FLAT+B1,w
0407 330A        TSTFSZ wreg

0408 C41C        goto mready

0409 299F        clrfr A+B3             ; begin speed down section
                 MOV24 AL,A
040A 6A23        MOVFP AL+BO,wreg      ; get byte of AL into w
Servo Control of a DC-Brush Motor

040B 4A9C MOVFP wreg,A+B0 ; move to A(B0)
040C 6A24 MOVFP AL+B1,wreg ; get byte of AL into w
040D 4A9D MOVFP wreg,A+B1 ; move to A(B1)
040E 6A25 MOVFP AL+B2,wreg ; get byte of AL into w
040F 4A9E MOVFP wreg,A+B2 ; move to A(B2)

0410 290A clrf wreg
0411 3169 cpfseq MOVSIGN
0412 C41C goto mready

NEG32 A

0413 139C COMF A+B0
0414 139D COMF A+B1
0415 139E COMF A+B2
0416 139F COMF A+B3
0417 290A CLRF wreg
0418 159C INCF A+B0
0419 119D ADDWFC A+B1
041A 119E ADDWFC A+B2
041B 119F ADDWFC A+B3

mready

MOV24 MOVPBUF+B1,POSITION

041C 6AA5 MOVFP MOVPBUF+B1+B0,wreg ; get byte of MOVPBUF+B1 into w
041D 4A55 MOVFP wreg,POSITION+B0 ; move to POSITION(B0)
041E 6AA6 MOVFP MOVPBUF+B1+B1,wreg ; get byte of MOVPBUF+B1 into w
041F 4A56 MOVFP wreg,POSITION+B1 ; move to POSITION(B1)
0420 6AA7 MOVFP MOVPBUF+B1+B2,wreg ; get byte of MOVPBUF+B1 into w
0421 4A57 MOVFP wreg,POSITION+B2 ; move to POSITION(B2)

MOV24 MOVPBUF+B0,VELOCITY

0422 6AA8 MOVFP MOVPBUF+B0+B0,wreg ; get byte of MOVPBUF+B0 into w
0423 4A58 MOVFP wreg,VELOCITY+B0 ; move to VELOCITY(B0)
0424 6AA9 MOVFP MOVPBUF+B0+B1,wreg ; get byte of MOVPBUF+B0 into w
0425 4A59 MOVFP wreg,VELOCITY+B1 ; move to VELOCITY(B1)
0426 6AAA MOVFP MOVPBUF+B0+B2,wreg ; get byte of MOVPBUF+B0 into w
0427 4A5A MOVFP wreg,VELOCITY+B2 ; move to VELOCITY(B2)

0428 0002 return

tmove

MOVFP32 MOVVAL,MOVTMP ; test if final velocity reached

0429 7C5F MOVFP MOVVAL+B0,MOVTMP+B0 ; move MOVVAL(B0) to MOVTMP(B0)
042A 7D60 MOVFP MOVVAL+B1,MOVTMP+B1 ; move MOVVAL(B1) to MOVTMP(B1)
042B 7E61 MOVFP MOVVAL+B2,MOVTMP+B2 ; move MOVVAL(B2) to MOVTMP(B2)
042C 7F62 MOVFP MOVVAL+B3,MOVTMP+B3 ; move MOVVAL(B3) to MOVTMP(B3)

SUB32 MOVPBUF,MOVTMP

042D 6AA8 MOVFP MOVPBUF+B0,wreg ; get lowest byte of MOVPBUF into w
042E 051C SUBWF MOVTMP+B0 ; sub lowest byte of MOVTMP, save in
042F 6AA9 MOVFP MOVPBUF+B1,wreg ; get 2nd byte of MOVPBUF into w
0430 031D SUBWF MOVTMP+B1 ; sub 2nd byte of MOVTMP, save in
0431 6AAA MOVFP MOVPBUF+B2,wreg ; get 3rd byte of MOVPBUF into w
0432 031E SUBWF MOVTMP+B2 ; sub 3rd byte of MOVTMP, save in
0433 6AAB MOVFP MOVPBUF+B3,wreg ; get 4th byte of MOVPBUF into w
0434 031F SUBWF MOVTMP+B3 ; sub 4th byte of MOVTMP, save in

0435 9769 btfss MOVSIGN,MSB
0436 C440 goto vmpos
Servo Control of a DC-Brush Motor

NEG32 MOVTMP

0437 131C COMF MOVTMP+B0
0438 131D COMF MOVTMP+B1
0439 131E COMF MOVTMP+B2
043A 131F COMF MOVTMP+B3
043B 290A CLRF wreg
043C 151C INCF MOVTMP+B0
043D 111D ADDRFC MOVTMP+B1
043E 111E ADDRFC MOVTMP+B2
043F 111F ADDRFC MOVTMP+B3

0440 971F btfs MOVTMP+B3,MSB
0441 C45B goto vmpos ; if not, continue

CLR32 A

0442 299C CLRF A+B0
0443 299D CLRF A+B1
0444 299E CLRF A+B2
0445 299F CLRF A+B3

0446 6A5F MOVFP MOWAL+B0, wreg
0447 4AAB MOVFP MOVTMP+B0, wreg
0448 6A60 MOVFP MOVTMP+B1, wreg
0449 4A9A MOVFP MOVTMP+B2, wreg
044A 6A61 MOVFP MOVTMP+B3, wreg
044B 4AAA MOVFP MOVVBUF+B0
044C 6A62 MOVFP MOVVBUF+B1
044D 4AAB MOVFP MOVVBUF+B2
044E 299A ; move unless the final velocity
044F 8093 bcf MOVSTAT, bit 5

0450 6A67 MOVFP MOVTIM+BO, wreg
0451 016E MOVFP TAU+B0
0452 6A68 MOVFP MOVTIM+B1, wreg
0453 016F MOVFP TAU+B1

TFS32 MOWAL

0454 6A5F MOVFP MOWAL+B0, wreg
0455 0860 IORWF MOVVAL+B1, W
0456 0861 IORWF MOVVAL+B2, W
0457 0862 IORWF MOVVAL+B3, W
0458 330A TSTFSZ wreg

0459 C45B goto vmpos ; if so, set A=0 and continue with

045A 8S93 bcf MOVSTAT, bit 6 ; is zero.

vmpos

0460 971F btfs MOVTMP+B3, MSB
0461 C45B goto vmmoveok ; if not continue

CLR32 A

0462 299C CLRF A+B0
0463 299D CLRF A+B1
0464 299E CLRF A+B2
0465 299F CLRF A+B3

0466 6A5F MOVFP MOWAL+B0, wreg ; get byte of MOWAL into w
0467 4AAB MOVFP MOVTMP+B0, wreg ; move to MOVVBUF (B0)
0468 6A60 MOVFP MOVTMP+B1, wreg ; get byte of MOVTMP into w
0469 4A9A MOVFP MOVTMP+B2, wreg ; move to MOVVBUF (B1)
046A 6A61 MOVFP MOVTMP+B3, wreg ; get byte of MOVTMP into w
046B 4AAA MOVFP MOVVBUF+B0
046C 6A62 MOVFP MOVVBUF+B1
046D 4AAB MOVFP MOVVBUF+B2
046E 299A ; move unless the final velocity
046F 8093 bcf MOVSTAT, bit 5 ; clear MOVFLAG

0470 6A67 MOVFP MOVTIM+BO, wreg ; get byte of MOVTIM into w
0471 016E MOVFP TAU+B0 ; move to TAU (B0)
0472 6A68 MOVFP MOVTIM+B1, wreg ; get byte of MOVTIM into w
0473 016F MOVFP TAU+B1 ; move to TAU (B1)

0474 6A5F MOVFP MOWAL+B0, wreg ; is zero.
0475 0860 IORWF MOVVAL+B1, W
0476 0861 IORWF MOVVAL+B2, W
0477 0862 IORWF MOVVAL+B3, W
0478 330A TSTFSZ wreg

0479 C45B goto vmmoveok ; if final velocity is zero, clear

047A 8S93 bcf MOVSTAT, bit 6 ; motion status flag

vmmoveok

047B 6A5F MOVFP MOVBPBUF+B1, wreg ; get byte of MOVBPBUF+B1 into w
047C 4A55 MOVFP wreg, POSITION+B0 ; move to POSITION (B0)
047D 6A66 MOVFP MOVBPBUF+B1, wreg ; get byte of MOVBPBUF+B1, wreg
047E 4A56 MOVFP wreg, POSITION+B1 ; get byte of MOVBPBUF+B1 into w
047F 6A67 MOVFP MOVBPBUF+B1, wreg
Servo Control of a DC-Brush Motor

0460 4A57
MOVFP wreg,POSITION+82 ; move to POSITION(B2)

MOV24 MOVVFBUF+B0,VELOCITY

0461 6AA8
MOVFP MOVVFBUF+B0+B0, wreg ; get byte of MOVVFBUF+B0 into w
0462 4A58
MOVFP wreg,VELOCITY+B0 ; move to VELOCITY(B0)
0463 6AA9
MOVFP MOVVFBUF+B0+B1, wreg ; get byte of MOVVFBUF+B0 into w
0464 4A59
MOVFP wreg,VELOCITY+B1 ; move to VELOCITY(B1)
0465 6AAA
MOVFP MOVVFBUF+B0+B2, wreg ; get byte of MOVVFBUF+B0 into w
0466 4A5A
MOVFP wreg,VELOCITY+B2 ; move to VELOCITY(B2)

0467 0002
return

;******************************************************************
;******************************************************************
; NAME: doPosVel
; DESCRIPTION: Evaluates the iterative equations for trapezoidal
generation
;
; V(k)=V(k-1)+A,
; P(k)=P(k-1)+V(k-1)+A/2,
;
; where abs(A)={AL,O} depending on the region of the
being executed.
;
; doPosVel

ADD32 MOVVFBUF,MOVVFBUF
; P(k-1)+V(k-1)

0468 6AA8
MOVFP MOVVFBUF+B0, wreg ; get lowest byte of MOVVFBUF into w
0469 0FA4
ADDWF MOVVFBUF+B0 ; add lowest byte of MOVVFBUF, save in
046A 6AA9
MOVFP MOVVFBUF+B1, wreg ; get 2nd byte of MOVVFBUF into w
046B 11A5
ADDWF MOVVFBUF+B1 ; add 2nd byte of MOVVFBUF, save in
046C 6AAA
MOVFP MOVVFBUF+B2, wreg ; get 3rd byte of MOVVFBUF into w
046D 11A6
ADDWF MOVVFBUF+B2 ; add 3rd byte of MOVVFBUF, save in
046E 6AA8
MOVFP MOVVFBUF+B3, wreg ; get 4th byte of MOVVFBUF into w
046F 11A7
ADDWF MOVVFBUF+B3 ; add 4th byte of MOVVFBUF, save in

ADD32 A,MOVVFBUF
; V(k)=V(k-1)+A

0470 6A9C
MOVFP A+B0, wreg ; get lowest byte of A into w
0471 0FA8
ADDWF MOVVFBUF+B0 ; add lowest byte of MOVVFBUF, save in
0472 6A9D
MOVFP A+B1, wreg ; get 2nd byte of A into w
0473 11A9
ADDWF MOVVFBUF+B1 ; add 2nd byte of MOVVFBUF, save in
0474 6A9E
MOVFP A+B2, wreg ; get 3rd byte of A into w
0475 11AA
ADDWF MOVVFBUF+B2 ; add 3rd byte of MOVVFBUF, save in
0476 6A9F
MOVFP A+B3, wreg ; get 4th byte of A into w
0477 11AB
ADDWF MOVVFBUF+B3 ; add 4th byte of MOVVFBUF, save in

MOVFP32 A,MOVTFMP
; compute A/2

0478 7C9C
MOVFP A+B0, MOVTFMP+B0 ; move A(B0) to MOVTFMP(B0)
0479 7D9D
MOVFP A+B1, MOVTFMP+B1 ; move A(B1) to MOVTFMP(B1)
047A 7E9E
MOVFP A+B2, MOVTFMP+B2 ; move A(B2) to MOVTFMP(B2)
047B 7F9F
MOVFP A+B3, MOVTFMP+B3 ; move A(B3) to MOVTFMP(B3)

RRC32 MOVTFMP

047C 1A1F
RLCF MOVTFMP+B3, W ; move sign into carry bit
Servo Control of a DC-Brush Motor

ADD32 MOVTMP, MOVPBUF

0481 6A1C MOVFP MOVTMP+B0, wreg; get lowest byte of MOVTMP into w
0482 0FA4 ADDWF MOVPBUF+B0; add lowest byte of MOVPBUF, save in
0483 6A1D MOVFP MOVTMP+B1, wreg; get 2nd byte of MOVTMP into w
0484 11A5 ADDWF MOVPBUF+B1; add 2nd byte of MOVPBUF, save in
0485 6A1E MOVFP MOVTMP+B2, wreg; get 3rd byte of MOVTMP into w
0486 11A6 ADDWF MOVPBUF+B2; add 3rd byte of MOVPBUF, save in
0487 6A1F MOVFP MOVTMP+B3, wreg; get 4th byte of MOVTMP into w
0488 11A7 ADDWF MOVPBUF+B3; add 4th byte of MOVPBUF, save in

0489 0002 return

;*******************************************************************
;*******************************************************************
; NAME: undoPosVel

; DESCRIPTION: Backward iteration of the equations for trapezoidal
generation

; V(k-1)=V(k)-A, P(k-1)=P(k)-V(k-1)-A/2,

; where abs(A)={AL,0} depending on the region of the
being executed. This routine is used to reverse a

; to be made beyond a decision point.

;*******************************************************************

undoPosVel

SUB32 A, MOVVBUF

048A 6A9C MOVFP A+B0, wreg; get lowest byte of A into w
048B 05A8 SUBWF MOVVBUF+B0; sub lowest byte of MOVVBUF, save in
048C 6A9D MOVFP A+B1, wreg; get 2nd byte of A into w
048D 03A9 SUBWF MOVVBUF+B1; sub 2nd byte of MOVVBUF, save in
048E 6A9E MOVFP A+B2, wreg; get 3rd byte of A into w
048F 03AA SUBWF MOVVBUF+B2; sub 3rd byte of MOVVBUF, save in
0490 6A9F MOVFP A+B3, wreg; get 4th byte of A into w
0491 03AB SUBWF MOVVBUF+B3; sub 4th byte of MOVVBUF, save in

SUB32 MOVVBUF, MOVPBUF

0492 6A85 MOVFP MOVVBUF+B0, wreg; get lowest byte of MOVVBUF into w
0493 05A4 SUBWF MOVPBUF+B0; sub lowest byte of MOVPBUF, save in
0494 6A89 MOVFP MOVVBUF+B1, wreg; get 2nd byte of MOVVBUF into w
0495 03A5 SUBWF MOVPBUF+B1; sub 2nd byte of MOVPBUF, save in
0496 6A8A MOVFP MOVVBUF+B2, wreg; get 3rd byte of MOVVBUF into w
0497 03A6 SUBWF MOVPBUF+B2; sub 3rd byte of MOVPBUF, save in
0498 6A8B MOVFP MOVVBUF+B3, wreg; get 4th byte of MOVVBUF into w
0499 03A7 SUBWF MOVPBUF+B3; sub 4th byte of MOVPBUF, save in

MOVFP32 A, MOVTMP; compute A/2

049A 7C9C MOVFP A+B0, MOVTMP+B0; move A(B0) to MOVTMP(B0)
Servo Control of a DC-Brush Motor

Serveo Control of a DC-Brush Motor

049B 7D9D MOVFP A+B1,MOVTEMP+B1 ; move A(B1) to MOVTEMP(B1)
049C 7E9E MOVFP A+B2,MOVTEMP+B2 ; move A(B2) to MOVTEMP(B2)
049D 7F9F MOVFP A+B3,MOVTEMP+B3 ; move A(B3) to MOVTEMP(B3)

RRC3 MOVTEMP

049E 1A1F RLCF MOVTEMP+B3,W ; move sign into carry bit
049F 191F RRCF MOVTEMP+B3
04A0 191E RRCF MOVTEMP+B2
04A1 191D RRCF MOVTEMP+B1
04A2 191C RRCF MOVTEMP+B0

SUB32 MOVTEMP,MOVBF8 ; P(k-1)=P(k)-V(k-1)-A/2,

04A3 6A1C MOVFP MOVTEMP+B0,wreg ; get lowest byte of MOVTEMP into w
04A4 05A4 SUBWF MOVBF8+B0 ; sub lowest byte of MOVBF8, save in
04A5 6A1D MOVFP MOVTEMP+B1,wreg ; get 2nd byte of MOVTEMP into w
04A6 03A5 SUBWF MOVBF8+B1 ; sub 2nd byte of MOVBF8, save in
04A7 6A1E MOVFP MOVTEMP+B2,wreg ; get 3rd byte of MOVTEMP into w
04A8 03A6 SUBWF MOVBF8+B2 ; sub 3rd byte of MOVBF8, save in
04A9 6A1F MOVFP MOVTEMP+B3,wreg ; get 4th byte of MOVTEMP into w
04AA 03A7 SUBWF MOVBF8+B3 ; sub 4th byte of MOVBF8, save in

04AB 0002 return

;******************************************************************************

if _SERVO_PID include "pid.asm" ; PID Algorithm
;******************************************************************************

;******************************************************************************

NAME: doServo
; DESCRIPTION: Performs the servo loop calculations.
;

doServos:

MOV16 POSERROR,U0 ; save new position error in

04AC 6A79 MOVFP POSERROR+B0,wreg ; get byte of POSERROR into w
04AD 0184 MOVWF U0+B0 ;move to U0(B0)
04AE 6A7A MOVFP POSERROR+B1,wreg ; get byte of POSERROR into w
04AF 0185 MOVWF U0+B1 ;move to U0(B1)

LOADAB U0,KP ; compute KP*U0

04B0 7C84 MOVFP U0+B0,AARG+B0 ; load lo byte of U0 to AARG
04B1 7D85 MOVFP U0+B1,AARG+B1 ; load hi byte of U0 to AARG
04B2 7E26 MOVFP KP+B0,BARG+B0 ; load lo byte of KP to BARG
04B3 7F27 MOVFP KP+B1,BARG+B1 ; load hi byte of KP to BARG

04B4 E12B call Dmul
Servo Control of a DC-Brush Motor

MOVPF32 DPX,Y ; Y=KP*UO
04B5 5880 MOVPF DPX+3,Y+3 ; move DPX(BO) to Y(BO)
04B6 5981 MOVPF DPX+1,Y+1 ; move DPX(Bl) to Y(Bl)
04B7 5A82 MOVPF DPX+2,Y+2 ; move DPX(B2) to Y(B2)
04B8 5B83 MOVPF DPX+3,Y+3 ; move DPX(B3) to Y(B3)

04B9 290A clrf wreg ; if previous output saturated, do
04BA 3295 cpfsgt SATFLAG not
04BB E552 call do Integral

LOADAS INTEGRAL,KI
04BC 7C96 MOVPF INTEGRAL+B0,AARG+B0 ; load lo byte of INTEGRAL to AARG
04BD 7E97 MOVPF INTEGRAL+B1,AARG+B1 ; load hi byte of INTEGRAL to AARG
04BE 7E2A MOVPF KI+B0,BARG+B0 ; load lo byte of KI to BARG
04BF 7F28 MOVPF KI+B1,BARG+B1 ; load hi byte of KI to BARG

LOADAS UO,AARG
04C0 E12B call Dmult ; Y=KP*UO+KI*INTEGRAL

ADD32 DPX,Y ; Y=KP*UO+KI*INTEGRAL
04C1 6A18 MOVPF DPX+3,wreg ; get lowest byte of DPX into w
04C2 0F80 ADDWF Y+0 ; add lowest byte of Y, save in Y(BO)
04C3 6A19 MOVPF DPX+1,wreg ; get 2nd byte of DPX into w
04C4 1181 ADDWF Y+B1 ; add 2nd byte of Y, save in Y(Bl)
04C5 6A1A MOVPF DPX+2,wreg ; get 3rd byte of DPX into w
04C6 1182 ADDWF Y+B2 ; add 3rd byte of Y, save in Y(B2)
04C7 6A1B MOVPF DPX+3,wreg ; get 4th byte of DPX into w
04C8 1183 ADDWF Y+B3 ; add 4th byte of Y, save in Y(B3)

MOVFP16 UO,AARG ; compute KV*(UO-U1)
04C9 7C84 MOVPF UO+0,AARG+B0 ; move UO(BO) to AARG(BO)
04CA 7D85 MOVPF UO+1,AARG+B1 ; move UO(Bl) to AARG(B1)

SUB16 U1,AARG
04CB 6A86 MOVPF U1+B0,wreg ; get lowest byte of U1 into w
04CC 051C SUBWF AARG+B0 ; sub lowest byte of AARG, save in
04CD 6A87 MOVPF U1+B1,wreg ; get 2nd byte of U1 into w
04CE 031D SUBWF B AARG+B1 ; sub 2nd byte of AARG, save in

MOVFP16 KV,BARG
04CF 7E28 MOVPF KV+B0,BARG+B0 ; move KV(BO) to BARG(BO)
04D0 7F29 MOVPF KV+B1,BARG+B1 ; move KV(Bl) to BARG(B1)

ADD32 DPX,Y ; Y=KP*UO+KI*INTEGRAL+KV*(UO-U1)
04D1 E12B call Dmult

MOVFP16 KV,BARG
04D2 6A18 MOVPF DPX+3,wreg ; get lowest byte of DPX into w
04D3 0F80 ADDWF Y+B0 ; add lowest byte of Y, save in Y(BO)
04D4 6A19 MOVPF DPX+1,wreg ; get 2nd byte of DPX into w
04D5 1181 ADDWF Y+B1 ; add 2nd byte of Y, save in Y(Bl)
04D6 6A1A MOVPF DPX+2,wreg ; get 3rd byte of DPX into w
04D7 1182 ADDWF Y+B2 ; add 3rd byte of Y, save in Y(B2)
04D8 6A1B MOVPF DPX+3,wreg ; get 4th byte of DPX into w
04D9 1183 ADDWF Y+B3 ; add 4th byte of Y, save in Y(B3)
Servo Control of a DC-Brush Motor

MOV16 U0, U1 ; push errors into U(k-1)
MOVFP U0+BO, wreg ; get byte of U0 into w
MOVWF UI+BO ; move to UI(BO)
MOVFP U0+B1, wreg ; get byte of U0 into w
MOVWF UI+B1 ; move to UI(B1)

clrf wreg
CPFSTG SHIFTNUM
GOTO grabok

MOVFP SHIFTNUM, TMP
goto grabloop

RLC32 Y

BCF _carry

RCF Y+BO

RCF Y+B1

RCF Y+B2

RCF Y+B3

DECFSZ TMP
GOTO grabloop

CLRF SATFLAG ; saturate to middle 16 bits,
GOTO NEG ; keeping top 10 bits for pwldcH
; and pwldcL

MOVFP Y+B2, wreg ; check if Y >= 2**23
ANDlw 0x80
IORWF Y+B3
CLRF wreg

CPFSTG Y+B3

GOTO zero6bits ; if not, zero 6 bits

INCF SATFLAG ; if so, set Y=0x007FFFFF

CLRF Y+B3 ; clear for debug purposes

MOVlw 0x7F
MOVFP wreg, Y+B2
SETF Y+B1
SETF Y+B0
GOTO zero6bits

MOVFP Y+B2, wreg ; check if Y <= -2**23
IORlw 0x7F
ANDWF Y+B3
SETF wreg

CPFSTC Y+B3

GOTO zero6bits ; if not, zero 6 bits

SETF SATFLAG ; if so, set Y = 0xFF800000

SETF Y+B3

CLRF Y+B2

BSF Y+B2, MSB

CLRF Y+B1

CLRF Y+B0

GOTO zero6bits

MOV24 Y+B1, YPWM+BO ; move Y to YPWM and zero 6 bits

MOVFP Y+B1+BO, wreg ; get byte of Y+B1 into w
MOVFP wreg, YPWM+B0+BO ; move to YPWM+BO(B0)
MOVFP Y+B1+Bl, wreg ; get byte of Y+B1 into w
MOVFP wreg, YPWM+B0+B1 ; move to YPWM+B0(B1)
MOVFP Y+B1+B2, wreg ; get byte of Y+B1 into w
MOVFP wreg, YPWM+B0+B2 ; move to YPWM+B0(B2)
Servo Control of a DC-Brush Motor

doTorque ; entry point for torque mode

050B B0C0 movlw 0x0C
050C 0B88 andwf YPWM+BO
050D 9F89 btfsC YPWM+BI,MSB
tplimit
050E C516 goto tmlimit
050F 9692 btfsY EXSTAT,bit6
goto mplimitok
CLR32 YPWM

0510 C51C clrF YPWM+BO
0511 2988 clrF YPWM+BI
0512 2989 clrF YPWM+BS
0513 298A clrF YPWM+BT
0514 298B clrF YPWM+BS

0515 C51C goto mplimitok
tmlimit
0516 9592 btfsY EXSTAT,bit5
goto mplimitok
CLR32 YPWM

0518 2988 clrF YPWM+BO
0519 2989 clrF YPWM+BI
051A 298A clrF YPWM+BS
051B 298B clrF YPWM+BT

mplimitok

051C B07F movlw PWIDCH_INIT ; adjustment from bipolar to unipolar
051D 4A19 movpf wreg,TMP+BI ; for 50% duty cycle
051E B0C0 movlw PWIDCL_INIT
051F 4A18 movpf wreg,TMP+B0
ADD16 TMP,YPWM

0520 6A18 MOVFP TMP+B0,wreg ; get lowest byte of TMP into w
0521 0F88 ADDWF YPWM+B0 ; add lowest byte of YPWM, save in YPWM(B0)
0522 6A19 MOVFP TMP+BI,wreg ; get 2nd byte of TMP into w
0523 1189 ADDWF YPWM+BI ; add 2nd byte of YPWM, save in YPWM(B1)

0524 2919 clrf TMP+B1 ; correct by 1 LSB
0525 B040 movlw 0x40 ; add one to bit5 of pwidcl
0526 4A18 movpf wreg,TMP+B0
ADD16 TMP,YPWM

0527 6A18 MOVFP TMP+B0,wreg ; get lowest byte of TMP into w
0528 0F88 ADDWF YPWM+B0 ; add lowest byte of YPWM, save in YPWM(B0)
0529 6A19 MOVFP TMP+BI,wreg ; get 2nd byte of TMP into w
052A 1189 ADDWF YPWM+BI ; add 2nd byte of YPWM, save in YPWM(B1)

testmax

052B 291A clrf TMP+B2 ; check pwm maximum limit
052C 298A clrf YPWM+B2 ; IMD18200 must have a minimum pulse
052D 298B clrf YPWM+B3 ; so duty cycle must not be 0 or 100%
MOVFP16 YPWMAX,TMP

052E 788E MOVFP YPWMAX+B0, TMP+B0; move YPWMAX(B0) to TMP(B0)
052F 798F MOVFP YPWMAX+B1, TMP+B1; move YPWMAX(B1) to TMP(B1)
Servo Control of a DC-Brush Motor

SUB24 YPWM, TMP

0530 6A88
0531 0518
0532 6A89
0533 0319
0534 6A8A
0535 031A
0536 971A
0537 C53D
0538 6A8E
0539 0188
053A 6A8F
053B 0189
053C C54E
053D 291A
053E 29A8
053F 298B
0540 788C
0541 798D
0542 6A88
0543 0518
0544 6A89
0545 0319
0546 6A8A
0547 031A
0548 9F1A
0549 C54E
054A 6A8C
054B 0188
054C 6A8D
054D 0189
054E B803
054F 7088
0550 7289
0551 0002

;***************************************************************************************************
;***************************************************************************************************
NAME: dointegral
; DESCRIPTION: Evaluates the integral for the servo calculations.
; doIntegral
ADD16 U0, INTEGRAL
; do integral
0552 6A84

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

ADDWF INTEGRAL+B0 ; add lowest byte of INTEGRAL, save in
MOVFP U0+B1,wreg ; get 2nd byte of U0 into w
ADDWF INTEGRAL+B1 ; add 2nd byte of INTEGRAL, save in

return

endif

if _SERIAL_IO
include "serial.asm" ; Serial I/O Routines

;****************************************************************
end

#include "serial.asm" ; Serial I/O Routines

;****************************************************************

; NAME: IdleFunction

; DESCRIPTION: This routine will perform work while doing waits in
serial I/O functions.

; IdleFunction

CLRWDT
return

;******************************************************************************

; NAME: DoCommand

; DESCRIPTION: Search command table for command and execute it.

; DoCommand

movlw (CMD_TABLE & 0xff) CMD_TABLE LS8
movpf wreg,tblptrl
movlw page CMD_TABLE CMD_TABLE MS8
movpf wreg, tblptrh

tablrd 1,1,CMDTEMP

tryNextCmd

tablrd 0,1,CMDTEMP ; read entry from table
tlrd 1,CMDPTRH
tablrd 0,1,CMDPTRL

movfp CMDTEMP,wreg
cpfslt ZERO
goto noCommand ; error if end of table
goto tryNextCmd

call PutChar ; echo command

goto tryNextCmd ; indirect jump to command routine

Servo Control of a DC-Brush Motor

056A E679
  call PutChar    ; send response character from
  movlw CR       ; command routine followed by CR
056B B00D
056C E679
056D C124
  goto PollingLoop
  noCommand
056E B03F
056F C56A
  movlw CMD_BAD  ; send error character
  goto cmdFinish

;***************************************************************************
; NAME: do_null
; DESCRIPTION: The do nothing command used to determine if the chip is
; working. Initiated by a carriage return.

0570 B021
0571 C56A
  movlw CMD_OK
  goto cmdFinish

;***************************************************************************
; NAME: do_move
; DESCRIPTION: Commands the axis to move to a new position or velocity.
; Position data is relative, and in encoder counts. Velocity
; data is absolute, and in encoder counts/sample time multi-
; plied by 256. All moves are performed by the controller such
; that velocity and acceleration limits set into parameter
; memory will not be violated. All move commands are kept in a
; one deep FIFO buffer. The command in the buffer is executed
; as soon as the currently executed command is complete.

ARGUMENTS: M [800000,7FFFFF]

do_move
  if DECIO
    call GetDecVal
  else
    call GetVal
  endif
0572 E6CC
0573 9F93
0574 C57E
  btfsC MOVSTAT,bit7   ; test if buffer available
  goto bufoverflow
  MOV24 VALBUF,NMOVVAL ; if so, accept value into NMOVVAL
0575 6A31
0576 4A5B
0577 6A32
0578 4A5C
0579 6A33
057A 4A5D
057B 8793
057C B021
  movlw CMD_OK
Servo Control of a DC-Brush Motor

057D C56A

goto cmdFinish

bufoverflow

057E B03F
movlw CMD_BAD ; else, return error

goto cmdFinish

;**************************************************************************
;**************************************************************************

; NAME: do_mode
;
; DESCRIPTION: An argument of "P" will cause all subsequent move commands
; to be incremental position moves. A "V" argument will cause
; all subsequent moves to be absolute velocity moves.
;
; ARGUMENTS: 0 [P,V]
;

do_mode

call IdleFunction ; get single character loop

call GetChk

cpfsseq ONE

goto do_mode

call GetChar

movpf wreg,STRVALL

clrf MODETYPE MODETYPE=O for position moves

testP

movlw 'P' position moves for type P

cpfsseq STRVALL

goto testV

goto modeok

testV

movlw 'V' velocity moves for type V

cpfsseq STRVALL

goto testT

incf MODETYPE ; MODETYPE=1 for velocity moves

goto modeok

testT

movlw 'T' TORQUE Moves for type 'T'

cpfsseq STRVALL

goto modeerror

setf MODETYPE ; MODETYPE=-1 for torque moves

clrf SERVOFLAG ; disable servo

goto modeok

modeerror

movlw CMD_BAD ; mode error

goto cmdFinish

modeok

movfp STRVALL,wreg ; echo type character

call PutChar

059A B021

movlw CMD_OK

goto cmdFinish

;**************************************************************************
;**************************************************************************

; NAME: do_setparameter
;
; DESCRIPTION: Sets controller parameters to the value given.
;
; Parameter         #       Range
;
; VL=velocity limit 0        [0,7FFFFF]
; AL=acceleration limit 1     [0,7FFFFF]
; KP=proportional gain 2     [8000,7FFF]
; KP=velocity gain 3          [8000,7FFF]

DS00532B-page 76 © 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

; KP=integral gain 4 [8000,7FFF]
; IM=integrator mode 5 [0,3]
; FV=velocity FF 6 [8000,7FFF] : Not Imple
; FA=acceleration FF 7 [8000,7FFF] : Not Imple

; ARGUMENTS: S [0,FF] [800000,7FFFFF]

do_setparameter
    call GetPar ; get parameter number
    movlw NUMPAR
    cpfslt VAL8UF+80
    goto Serror
    movlw IPAR_TA8LE
    movpf wreg, tblptr
    movlw page PAR_TA8LE
    movpf wreg, tblptrh
    table d 1,1,PARTEMP

setNextPar
    tblrd 1,1,PARTEMP ; read entry from table
    tblrd 0,1,PARLEN
    tblrd 0,1,PARPTR
    movlw NUMPAR
    cpfslt PAR_TEMP ; error if end of table
    movf p PARTEMP,wreg
    cpfseq VAL8UF+80
    goto setNextPar
    movfp PARPTR,wreg
    movfp wreg,fsrl
    if DECIO ; pointer to parameter in fsrl
        call GetDecVal
    else
        call GetVal
    endif
    movlw VALBUF ; get new value in VALBUF
    call GetDecVal
    movf p PARLEN
    movfp wreg,fsrO
    autoINC
    movfp _fsO
    movfp _fs1
    movfp _fs2
    movfp _fs3
    setGetMore
    movfp indfO,indf1 ; move new value to parameter
    decf PARLEN
    tstfz PARLEN

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

```
05BA C5B7    goto  setGetMore
              ; no autoincrement

05BB 8404    BSF  _fs0
05BC 8504    BSF  _fs1
05BD 8604    BSF  _fs2
05BE 8704    BSF  _fs3

05BF B021    movlw  CMD_OK
05C0 C56A    goto  cmdFinish

;error
05C1 B03F    movlw  CMD_BAD
05C2 C56A    goto  cmdFinish

;**************************************************************************
;**************************************************************************
; NAME: do_readparameter
; DESCRIPTION: Returns the present value of a parameter.
; ARGUMENTS: R [0,FF]  
; RETURNS: The present value of the requested parameter is returned.

do_readparameter

05C3 E669    call  GetPar  ; get parameter number

05C4 B608    movlw  NUMPAR  ; check if in range [0,NUMPAR]
05C5 3631    cpfslt  VALBUF+BO  ;PAR_TABLE LSB
05C6 C5EB    goto  Rerror

05C7 B07C    movlw  (PAR_TABLE & Oxff)  ;PAR_TABLE LSB
05C8 4AO0    movpf  wreg,tblptrl
05C9 B007    movlw  page PAR_TABLE  ;PAR_TABLE MSB
05CA 4AOE    movpf  wreg,tblptrh
05CB AB3D    tablrd  l,1,PARTEMP  ;readNextPar

05CC A23D    tablrd  1,PARTEMP  ;read entry from table
05CD A93E    tablrd  0,1,PARLEN
05CE A93F    tablrd  0,1,PARPTR

05CF B008    movlw  NUMPAR  ;error if end of table
05D0 303D    cpfslt  PARTEMP
05D1 C5EB    goto  Rerror

05D2 6A3D    movfp  PARTEMP, wreg
05D3 3131    cpfseq  VALBUF+BO
05D4 C5CC    goto  readNextPar

05D5 6A3F    movfp  PARPTR, wreg  ;pointer to parameter in fsrl
05D6 690A    movfp  wreg, fsrl

05D7 B031    movlw  VALBUF  ;pointer to VALBUF in fsrl
05D8 610A    movfp  wreg, far0  ;set autoincrement

AUTO INC
05D9 8404    BSF  _fs0
05DA 8D04    BCF  _fs1
05DB 8604    BSF  _fs2
```
Servo Control of a DC-Brush Motor

05DC 8F04

BCF _fs3

CLR24 VALBUF ; clear old VALBUF
05DD 2931
05DE 2932
05DF 2933
CLRF VALBUF+B0
CLRF VALBUF+B1
CLRF VALBUF+B2

readGetMore
05E0 6008
movfp indfl,indf0 ; read parameter into VALBUF
05E1 073E
decf PARLEN
05E2 333E
tstfsz PARLEN
05E3 C56A
goto readGetMore

AUTON0 ; no autoincrement
05E4 8404
05E5 8504
05E6 8604
05E7 8704
BSF _fs0
BSF _fs1
BSF _fs2
BSF _fs3

if DECIO ; send parameter value
05E8 B728
call PutDecVal
else
    call PutVal
endif

05E9 B021
05EA C56A
movlw CMD_OK
goto cmdFinish

Rerror
05EB B03F
05EC C56A
movlw CMD_BAD
goto cmdFinish

;***************************************************************************
;***********~************************************************************

NAME: do_shutter

DESCRIPTION: Returns the time (in sample time counts [0,FFFF]) since the start of the present move and captures the commanded and measured values of position and velocity at the time of the command.

ARGUMENTS:

RETURNS:
The time since the start of the present move is returned.

do_shutter

MOV24 POSITION,CPOSITION ; capture commanded position
05ED 6A55
05EE 4A40
05EF 6A56
05F0 4A41
05F1 6A57
MOVFP POSITION+B0,wreg ; get byte of POSITION into w
MOVFP wreg,CPOSITION+B0 ; move to CPOSITION(B0)
MOVFP POSITION+B1,wreg ; get byte of POSITION into w
MOVFP wreg,CPOSITION+B1 ; move to CPOSITION(B1)
MOVFP POSITION+B2,wreg ; get byte of POSITION into w
MOV24 VELOCITY,CVELOCITY ; capture commanded velocity

MOV24 MPOSITION,CMPOSITION ; capture measured position

MOV24 MVELOCITY,CMVELOCITY ; capture measured velocity

if DECIO
    call PutDecVal
else
    call PutVal
endif

movlw CMD_OK
goto cmdFinish

******************************************************************************

NAME: do_readcomposition
DESCRIPTION: Returns the commanded position count which was captured during the last shutter command.

ARGUMENTS: p

RETURNS: The last captured position count is returned. [800000,7FFFFF]
do_readcomposition

MOV24 CPOSITION, VALBUF ; move CPOSITION to VALBUF

if

MOVFP CPOSITION+0, wreg ; get byte of CPOSITION into w
MOVFP wreg, VALBUF+0 ; move to VALBUF(B0)

else

MOVFP CPOSITION+1, wreg ; get byte of CPOSITION into w
MOVFP wreg, VALBUF+1 ; move to VALBUF(B1)

endif

movlw CMD_OK
goto cmdFinish

;***************************************************************************
;***************************************************************************
; NAME: do_readcomvelocity
;
; DESCRIPTION: Returns the commanded velocity multiplied by 256 which was
; captured during the last shutter command.
;
; ARGUMENTS: V
;
; RETURNS: The last captured commanded velocity times 256 is returned.
; [#00000, 7FFFFF]
;
;***************************************************************************
;***************************************************************************

do_readcomvelocity

MOV24 CVELOCITY, VALBUF ; move commanded velocity to VALBUF

if

MOVFP CVELOCITY+0, wreg ; get byte of CVELOCITY into w
MOVFP wreg, VALBUF+0 ; move to VALBUF(B0)

else

MOVFP CVELOCITY+1, wreg ; get byte of CVELOCITY into w
MOVFP wreg, VALBUF+1 ; move to VALBUF(B1)

endif

movlw CMD_OK
goto cmdFinish

;***************************************************************************
;***************************************************************************
Servo Control of a DC-Brush Motor

; NAME: do_readactposition
; DESCRIPTION: Returns the measured position count which was captured during the last shutter command.
; ARGUMENTS: p
; RETURNS: The last captured measured position count is returned. [800000,7FFFFF]

;move measured position to

 MOV24 CMPOSITION,VALBUF
 MOVFP CMPOSITION+80,wreg
 MOVFP wreg,VALBUF+B0
 MOVFP CMPOSITION+81,wreg
 MOVFP wreg,VALBUF+B1
 MOVFP CMPOSITION+82,wreg
 MOVFP wreg,VALBUF+B2

if DECIO
    call PutDecVal
else
    call PutVal
endif

 movlw CMD_OK
goto cmdFinish

;**************************************************************************
;**************************************************************************
; NAME: do_readactvelocity
; DESCRIPTION: Returns the measured velocity multiplied by 256 which was captured during the last shutter command.
; ARGUMENTS: v
; RETURNS: The last captured measured velocity times 256 is returned. [800000,7FFFFF]

;move measured velocity to

 MOV24 CMVELOCITY,VALBUF
 MOVFP CMVELOCITY+80,wreg
 MOVFP wreg,VALBUF+B0
 MOVFP CMVELOCITY+81,wreg
 MOVFP wreg,VALBUF+B1
 MOVFP CMVELOCITY+82,wreg
 MOVFP wreg,VALBUF+B2

if DECIO
    call PutDecVal
else


call    PutVal
endif

movlw    CMD_OK
goto    cmdFinish

;***********************************************************************
; NAME: do_externalstatus
;
; DESCRIPTION: Returns a two digit hex number which defines the state of
; the bits in the external status register. Issuing this
; command will clear all the bits in the external status
; register unless the event which set the bit is still true.
;
; ARGUMENTS: X
;
; RETURNS: The external status register is returned.
;
do_externalstatus

bsf    _glintd
movfp    EXTSTAT,wreg
clrif    EXTSTAT
bcf    _glintd
call    PutHex

movlw    CMD_OK
goto    cmdFinish

;***********************************************************************
; NAME: do_movestatus
;
; DESCRIPTION: Returns a two digit hex number which defines the state of
; the bits in the move status register. Issuing this command
; will clear all the bits in the move status register unless
; the event which set the bit is still true.
;
; ARGUMENTS: Y
;
; RETURNS: The move status register is returned.
;
do_movestatus

movfp    MOVSTAT,wreg
call    PutHex

movlw    CMD_OK
goto    cmdFinish

;***********************************************************************
; NAME: do_readindposition
;
; DESCRIPTION: Returns the last index position captured in position counts.
;
; ARGUMENTS: I
;
; RETURNS: The last captured index position is returned.
;
do_readindposition
Servo Control of a DC-Brush Motor

MOV24 INDEXPOS, VALBUF ; move measured velocity to VALBUF

063C 6AB6  MOVFP INDEXPOS+BO, wreg ; get byte of INDEXPOS into w
063D 4A31  MOVFP wreg, VALBUF+BO ; move to VALBUF (BO)
063E 6AB7  MOVFP INDEXPOS+BI, wreg ; get byte of INDEXPOS into w
063F 4A32  MOVFP wreg, VALBUF+BI ; move to VALBUF (BI)
0640 6AB8  MOVFP INDEXPOS+B2, wreg ; get byte of INDEXPOS into w
0641 4A33  MOVFP wreg, VALBUF+B2 ; move to VALBUF (B2)

if DECIO
    call PutDecVal
else
    call PutVal
endif

MOV24 VALBUF, POSITION

0642 E728  MOVFP VALBUF+BO, wreg ; get byte of VALBUF into w
0643 B021  MOVFP wreg, POSITION+BO ; move to POSITION (BO)
0644 C56A  MOVFP VALBUF+Bl, wreg ; get byte of VALBUF into w
0645 E6CC  MOVFP wreg, POSITION+Bl ; move to POSITION (Bl)
0646 6A31  MOVFP VALBUF+B2, wreg ; get byte of VALBUF into w
0647 4A55  MOVFP wreg, POSITION+B2 ; move to POSITION (B2)
0648 6A32  MOVFP VALBUF+BO, wreg ; get byte of VALBUF into w
0649 4A56  MOVFP wreg, POSITION+BO ; move to POSITION (BO)
064A 6A33  MOVFP VALBUF+BI, wreg ; get byte of VALBUF into w
064B 4A57  MOVFP wreg, POSITION+BI ; move to POSITION (BI)

MOV24 VALBUF, MPOSITION

064C 6A31  MOVFP VALBUF+BO, wreg ; get byte of VALBUF into w
064D 4A72  MOVFP wreg, MPOSITION+BO ; move to MPOSITION (BO)
064E 6A32  MOVFP VALBUF+BI, wreg ; get byte of VALBUF into w
064F 4A73  MOVFP wreg, MPOSITION+BI ; move to MPOSITION (BI)
0650 6A33  MOVFP VALBUF+B2, wreg ; get byte of VALBUF into w
0651 4A74  MOVFP wreg, MPOSITION+B2 ; move to MPOSITION (B2)
CLR32  Y
CLR3Y  Y+B0
CLR3Y  Y+B1
CLR3Y  Y+B2
CLR3Y  Y+B3

movlw  CMD_OK
goto  cmdFinish

;***********************************************************************
;***********************************************************************
; NAME:    do_reset
;
; DESCRIPTION: Performs a software reset.
;
; ARGUMENTS:  Z
;
;***********************************************************************

do_reset

movlw  CMD_OK
call  PutChar
goto  Startup

;***********************************************************************
;***********************************************************************
; NAME:    do_stop
;
; DESCRIPTION: Stops servo by clearing SERVOFLAG.
;
;***********************************************************************

do_stop

clrf  SERVOFLAG
movlw  CMD_OK
goto  cmdFinish

;***********************************************************************
;***********************************************************************
; NAME:    do_capture
;
;***********************************************************************

do_capture

if  ((_PICMASTER_DEBUG == 1) && (DECIO == 1))
call  GetDecVal
endif

if  ((_PICMASTER_DEBUG == 1) && (DECIO == 0))
call  GetVal
endif

if  _PICMASTER_DEBUG == 1

MOV16  VALBUF,CAPCOUNT

MOVFP  VALBUF+B0,wreg  ; get byte of VALBUF into w
MOVWF  CAPCOUNT+B0  ; move to CAPCOUNT(B0)
MOVFP  VALBUF+B1,wreg  ; get byte of VALBUF into w
MOVWF  CAPCOUNT+B1  ; move to CAPCOUNT(B1)

© 1993 Microchip Technology Inc. DS00532B-page 85
MOV16   VALBUF,CAPTMP

0663 6A31
0664 01BE
0665 6A32
0666 01BF

movl w  CMD_OK
goto  cmdFinish
endif

;***************************************************************************
; NAME: GetPar
;
; DESCRIPTION: Get a parameter number [0,FF] from the serial port and place
; it in VALBUF+80.
;
GetPar

CLR24   VALBUF

0669 2931
066A 2932
066B 2933

066C B6B2
066D 6A4E
066E B50F
066F 4A31
0670 1D31

call  GetHex
movfp  HEXVAL,wreg
andlw  OX0F
movpf  wreg,VALBUF+80
swapf  VALBUF+80

0671 B6B2
0672 6A31

0673 0E4E
0674 4A31

addwf  HEXVAL,w
movpf  wreg,VALBUF+80

0675 0002

return

;***************************************************************************
;***************************************************************************
; NAME: GetChar
;
; DESCRIPTION: Get character from receive buffer.

GetChar

movlb  bank0 ; set bank0
movpf  rcreg,wreg ; receive character

0676 B800
0677 540A
0678 0002

return

;***************************************************************************
;***************************************************************************
; NAME: PutChar
;
; DESCRIPTION: send character out the serial port
;
PutChar

movl w  bank0 ; set bank0
movpf  rcreg,wreg ; receive character

0676 B800
0677 540A
0678 0002

return
Servo Control of a DC-Brush Motor

PutChar

0679 B081 movlb bank1 ; set bank1
bufwait
067A 9116 btfss _tmt ; is transmit buffer empty?
067B C67A goto bufwait
067C B800 movlb bank0 ; set bank0
shfwait
067D 9115 btfss _rmt ; is transmit shift register empty?
067E C67D goto shfwait
067F 4A16 movpf wreg,txreg ; if so, send character
0680 0002 return

***************************************************************************
***************************************************************************
NAME: GetChk
DESCRIPTION: Check if character is in receive buffer.
***************************************************************************
***************************************************************************
GetChk

0681 B801 movlb bank1 ; set bank1
0682 560A movpf pir, wreg
0683 B501 andlw CHARREADY ; return status in wreg
0684 0002 return

***************************************************************************
***************************************************************************
NAME: PutDec
DESCRIPTION: Converts a hex value [0,F] in wreg to its ASCII equivalent.
The upper nibble of wreg is assumed to be zero.
ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII decimal
if DECIO

PutDec

0685 B130 addlw 0x30 ; convert to ASCII
0686 B679 call PutChar
0687 0002 return
endif

***************************************************************************
***************************************************************************
NAME: PutHex
DESCRIPTION: Convert the wreg value to ASCII hexadecimal. The output
format is two digits with the A-F parts in upper case and
leading zeros. The result is sent out the serial port with
PutChar.
ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII hex

PutHex

0688 4A4E movpf wreg,HEXVAL
0689 1D0A swapf wreg
068A B50F andlw 0x0F
068B 4A4F movpf wreg,HEXTMP
Servo Control of a DC-Brush Motor

```assembly
068C 2D0A negw wreg
068D B109 addlw 0x09
068E 970A btfss wreg, MSB
068F C693 goto putH20
0690 B037 movlw 'A' - 0x0A
0691 0E4F addwf HEXTMP, W
0692 C695 goto putH25

putH20
0693 B030 movlw '0'
0694 0E4F addwf HEXTMP, W

putH25
0695 E679 call PutChar

0696 6A4E movfp HEXVAL, wreg
0697 B50F andlw 0x0F
0698 4A4F movpf wreg, HEXTMP
0699 2D0A negw wreg
069A B109 addlw 0x09
069B 970A btfss wreg, MSB
069C C6A0 goto putL120
069D B037 movlw 'A' - 0x0A
069E 0E4F addwf HEXTMP, W
069F C6A2 goto putL125

putL120 movlw '0'
06A0 0E4F addwf HEXTMP, W

putL125 06A2 E679 call PutChar
06A3 0002 return

;************************************************************************************
; NAME: PutStr
;
; DESCRIPTION: Sends a character string out the serial port.
;
PutStr
06A4 AB4C tabird 1,1, STRVALH
GetNextPair
06A5 A24C tlrld 1, STRVALH
06A6 A94D tabird 0,1, STRVALL
06A7 6A4C movfp STRVALH, wreg
06A8 31C1 cpfseq ZERO
06A9 C6AB goto putH
06AA 0002 return

putH 06AB E679 call PutChar
06AD 6A4D movfp STRVALH, wreg
06AE 31C1 cpfseq ZERO
06AF 0002 goto putL
06B0 E679 return
06B1 C6A5 goto GetNextPair

;***********************************************************************************
;
; NAME: GetHex
;
; DESCRIPTION: Receive an ASCII hex character from the serial port and convert to numerical value.
```

---

DS00532B-page 88 © 1993 Microchip Technology Inc.

4-284
Servo Control of a DC-Brush Motor

; RETURNS: numerical value in HEXVAL

GetHex

getnxt

06B2 E557        call IdleFunction
06B3 E681        call GetChk
06B4 31C2        cpfseq ONE
06B5 C6B2        goto getnxt

06B6 2950        clr HEXSTAT
06B7 E676        call GetChar
06B8 4A4E        movpf wreg,HEXVAL
06B9 E679        call PutChar
06BA B00D        movlw CR
06BB 044E        subwf HEXVAL,W
06BC 330A        tstfz wreg
06BD C6BF        goto gth10
06BE C6C9        goto gchCR

gth10

06BF 6A4E        movfp HEXVAL,wreg
06C0 B239        sublw '9'
06C1 970A        btfss wreg,MSB
06C2 C6C5        goto gth20

06C3 B009        movlw 0x09
06C4 0F4E        addwf HEXVAL

gth20

06C5 B00F        movlw 0x0F
06C6 0B4E        andwf HEXVAL
06C7 2950        clr HEXSTAT
06C8 0002        return

gchCR

06C9 B001        movlw 0x01
06CA 4A50        movpf wreg,HEXSTAT
06CB 0002        return

;***************************************************************************
;***************************************************************************
; NAME:      getval
; DESCRIPTION: Get a value [800000,7FFFFF] from the serial port and place
; it in VALBUF.
; if DECIO
; else

GetVal

CLR24 VALBUF

getnxt

call GetHex

movlw 0x01
cpfseq HEXSTAT
return

shift

swapf VALBUF+B2
movfp VALBUF+B2,wreg
andlw 0xF0
movpf wreg,VALBUF+B2
swapf VALBUF+B1
movfp VALBUF+B1,wreg
andlw 0xF0

© 1993 Microchip Technology Inc. DS00532B-page 89
Servo Control of a DC-Brush Motor

```assembly
addwf VALBUF+B2
movfp VALBUF+B1, wreg
andlw 0xF0
movpf wreg, VALBUF+B1
swapf VALBUF+B0
movfp VALBUF+B0, wreg
andlw 0xF0
addwf VALBUF+B1
movfp VALBUF+B0, wreg
andlw 0xF0
addwf HEXVAL, w
movpf wreg, VALBUF+B0

goto getnext
endif

;***************************************************************************
;***************************************************************************
; NAME: GetDecVal
;
; DESCRIPTION: Get a value [-8388608, 8388607] from the serial port and
; place it in VALBUF
;
; RETURNS: numerical value is returned in VALBUF

if DECIO
GetDecVal
CLR24 VALBUF

06CC 2931
06CD 2932
06CE 2933
06CF E708
06D0 299B
06D1 B001
06D2 3199
06D3 299B

call GetDec
setf DEC SIGN
movlw DEC MN
cpfseq DEC STAT
clrf DEC SIGN
getdecnext
call GetDec
movlw DEC CR
cpfseq DEC STAT
goto mul10
goto fixsign

mul10
RLC24 VALBUF
BCF carry
RLCF VALBUF+B0
RLCF VALBUF+B1
RLCF VALBUF+B2

MOV24 VALBUF, DVALBUF

06DD 6A31
06DE 6A32
06DF 6A35
06E0 6A33
06E2 4A36

MOVFP VALBUF+B0, wreg
MOVFP VALBUF+B1, wreg
MOVFP VALBUF+B2, wreg
MOVFP wreg, DVALBUF+B0
MOVFP wreg, DVALBUF+B1
MOVFP wreg, DVALBUF+B2

RLC24 VALBUF
```

© 1993 Microchip Technology Inc.
Servo Control of a DC-Brush Motor

06E3 8804  BCF _carry
06E4 1B31  RLCF VALBUF+B0
06E5 1B32  RLCF VALBUF+B1
06E6 1B33  RLCF VALBUF+B2

RLC24 VALBUF    ; VALBUF now multiplied by eight

06E7 8804  BCF _carry
06E8 1B31  RLCF VALBUF+B0
06E9 1B32  RLCF VALBUF+B1
06EA 1B33  RLCF VALBUF+B2

ADD24 DVALBUF,VALBUF    ; VALBUF now multiplied by ten

06EB 6A34  MOVFP DVALBUF+B0,wreg
06EC 6A35  MOVFP DVALBUF+B1,wreg
06EE 1132  ADDWFC VALBUF+B1
06EF 6A36  MOVFP DVALBUF+B2,wreg
06F0 1133  ADDWFC VALBUF+B2

CLR24 DVALBUF

06F1 2934  CLR F DVALBUF+B0
06F2 2935  CLR F DVALBUF+B1
06F3 2936  CLR F DVALBUF+B2

06F4 6A98  movfp DECVAL,wreg
06F5 4A34  movpf wreg,DVALBUF+B0

ADD24 DVALBUF,VALBUF

06F6 6A34  MOVFP DVALBUF+B0,wreg    ; get lowest byte of DVALBUF into w
06F7 6A35  MOVFP DVALBUF+B1,wreg    ; get 2nd byte of DVALBUF into w
06F8 1132  ADDWFC VALBUF+B1    ; add 2nd byte of VALBUF, save in VALBUF (B1)
06F9 6A36  MOVFP DVALBUF+B2,wreg    ; get 3rd byte of DVALBUF into w
06FA 1133  ADDWFC VALBUF+B2    ; add 3rd byte of VALBUF, save in VALBUF (B2)

CLR24 DVALBUF

06FC C6D4  goto getdecnext

06FD 290A  clrf wreg
06FE 329B  cpfsgt DECSIGN
06FF 0002  return

NEG24 VALBUF

0700 1331  COM F VALBUF+B0
0700 1332  COM F VALBUF+B1
0700 1333  COM F VALBUF+B2
0700 290A  CLRF wreg
0704 1531  INC F VALBUF+B0
0705 1132  ADDWFC VALBUF+B1
0706 1133  ADDWFC VALBUF+B2

0707 0002  , return

endif
Servo Control of a DC-Brush Motor

;***********************************************************************
;***********************************************************************
; NAME: GetDec
; DESCRIPTION: Receive an ASCII decimal character from the serial port and
; convert to its numerical value.
; ARGUMENTS: numerical value is returned in DECVAL
;
; if DECIO

GetDec

call IdleFunction

call GetChk

cpfseq ONE

goto getdecnxt

0708 E557
0709 E681
070A 31C2
070B C708
070C E676
070D 4A98
070E E679
070F B00D
0710 0498
0711 30C1
0712 C71F
0713 B02D
0714 0498
0715 30C1
0716 C722
0717 B020
0718 0498
0719 30C1
071A C725
071B B00F
071C 0B98
071D 2999
071E 0002

getdecnxt

call GetChar

movpf wreg,DECVAL

call PutChar

movlw CR

subwf DECVAL,W

cpfsclt ZERO

goto gtdCR

movlw MN

subwf DECVAL,W

cpfsclt ZERO

goto gtdMN

movlw SP

subwf DECVAL,W

cpfsclt ZERO

goto gtdSP

qtd09

movlw 0x0F

andwf DECVAL

crff DECSTAT

return

qtdCR

movlw DEC_CR

movpf wreg,DECSTAT

return

qtdMN

movlw DEC_MN

movpf wreg,DECSTAT

return

qtdSP

movlw DEC_SP

movpf wreg,DECSTAT

return

endif

;***********************************************************************
;***********************************************************************
; NAME: PutVal
; DESCRIPTION: Sends the value in VALBUF [800000,7FFFFF] out the serial
;
; if DECIO

PutVal

else
Servo Control of a DC-Brush Motor

PutVal

movfp VALBUF+82,wreg
call PutHex
movfp VALBUF+81,wreg
call PutHex
movfp VALBUF+80,wreg
call PutHex

return

endif

***************************************************************************

***************************************************************************

NAME: PutDecVal

DESCRIPTION: Send the value in VALBUF [-8388608,8388607] out the serial

if DECIO

PutDecVal

0728 9733 btfss VALBUF+82,MSB
0729 C734 goto pdpos

NEG24 VALBUF

072A 1331 COMF VALBUF+80
072B 1332 COMF VALBUF+81
072C 1333 COMF VALBUF+82
072D 290A CLRF wreg
072E 1531 INCF VALBUF+80
072F 1132 ADDWF VALBUF+81
0730 1133 ADDWF VALBUF+82

0731 8020 movlw MN
call PutChar
0732 E679 goto pddigits
0733 C736 pdpos

call PutChar

0734 B020 movlw SP
call PutChar
0735 E679 pddigits

0736 B08D movlw (DEC_TABLE & 0xff) ; DEC_TABLE LSB
0737 4A0D movpf wreg, tblptrl
0738 B007 movlw page DEC_TABLE
0739 4A0E movpf wreg, tblptrh

073A A934 tablrd 0,1,DVALBUF+80
readNextDec

073B A034 tird 0,DVALBUF+80 ; read entry from table
073C AB35 tablrd 1,1,DVALBUF+81
073D A936 tablrd 0,1,DVALBUF+82

073E 2B0A setf wreg ; unitsposition if end of table
073F 3134 cplseq DVALBUF+80
0740 C742 goto getdigit
0741 C756 goto unitsposition

getdigit

0742 1534 incf DVALBUF+80 ; restore to power of 10
0743 2B98 setf DECVAL ; set DECVAL to -1
0744 1598 incf DECVAL ; increment DECVAL
0745 SUB24 DVALBUF,VALBUF ; check if in range

© 1993 Microchip Technology Inc.

DS00532B-page 93
Servo Control of a DC-Brush Motor

0745 6A34  MOVFP DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
0746 0531  SUBWF VALBUF+B0 ; sub lowest byte of VALBUF, save in
0747 6A35  MOVFP DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0748 0332  SUBWF VALBUF+B1 ; sub 2nd byte of VALBUF, save in VALBUF(B1)
0749 6A36  MOVFP DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
074A 0333  SUBWF VALBUF+B2 ; sub 3rd byte of VALBUF, save in VALBUF(B2)

074B 9733  btfss VALBUF+B2,MSB
074C C744  goto inc

ADD24 DVALBUF,VALBUF ; if so, correct VALBUF for next digit
074D 6A34  MOVFP DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
074E 0F31  ADDWF VALBUF+B0 ; add lowest byte of VALBUF, save in
074F 6A35  MOVFP DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0750 1132  ADDWF VALBUF+B1 ; add 2nd byte of VALBUF, save in VALBUF(B1)
0751 6A36  MOVFP DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
0752 1133  ADDWF VALBUF+B2 ; add 3rd byte of VALBUF, save in VALBUF(B2)

0753 6A98  movfp DECVAL,wreg ; send DECVAL
0754 E685  call PutDec
0755 C73B  goto readNextDec ; get next table entry

unitsposition
0756 6A31  movfp VALBUF+B0,wreg ; units position value now in VALBUF
0757 E685  call PutDec
0758 0002  return

endf

***************************************************************************

TABLES:

CMD_START CMD_TABLE

CMD_TABLE

CMD_DEF do_null,DO_NULL
0759 000D  DATA DO_NULL
075A 0570  DATA do_null

CMD_DEF do_move,DO_MOVE
075B 004D  DATA DO_MOVE
075C 0572  DATA do_move

CMD_DEF do_mode,DO_MODE
075D 004F  DATA DO_MODE
075E 0580  DATA do_mode

CMD_DEF do_setparameter,DO_SETPARAMETER
075F 0053  DATA DO_SETPARAMETER
0760 059C  DATA do_setparameter

CMD_DEF do_readparameter,DO_READPARAMETER

***************************************************************************
Servo Control of a DC-Brush Motor

0761 0052  DATA  DO_READPARAMETER
0762 05C3  DATA  do_readparameter

CMD_DEF  do_shutter,DO_SHUTTER

0763 0043  DATA  DO_SHUTTER
0764 05ED  DATA  do_shutter

CMD_DEF  do_readcomposition,DO_READCOMPOSITION

0765 0050  DATA  DO_READCOMPOSITION
0766 060D  DATA  do_readcomposition

CMD_DEF  do_readcomvelocity,DO_READCOMVELOCITY

0767 0056  DATA  DO_READCOMVELOCITY
0768 0616  DATA  do_readcomvelocity

CMD_DEF  do_readactposition,DO_READACTPOSITION

0769 0070  DATA  DO_READACTPOSITION
076A 061F  DATA  do_readactposition

CMD_DEF  do_readactvelocity,DO_READACTVELOCITY

076B 0076  DATA  DO_READACTVELOCITY
076C 0628  DATA  do_readactvelocity

CMD_DEF  do_externalstatus,DO_EXTERNALSTATUS

076D 0058  DATA  DO_EXTERNALSTATUS
076E 0631  DATA  do_externalstatus

CMD_DEF  do_movestatus,DO_MOVESTATUS

076F 0059  DATA  DO_MOVESTATUS
0770 0638  DATA  do_movestatus

CMD_DEF  do_readindposition,DO_READINDPOSITION

0771 0049  DATA  DO_READINDPOSITION
0772 063C  DATA  do_readindposition

CMD_DEF  do_setposition,DO_SETPOSITION

0773 0048  DATA  DO_SETPOSITION
0774 0645  DATA  do_setposition

CMD_DEF  do_reset,DO_RESET

0775 005A  DATA  DO_RESET
0776 0658  DATA  do_reset

CMD_DEF  do_stop,DO_STOP

0777 0073  DATA  DO_STOP
0778 065B  DATA  do_stop

if  _PICMASTER_DEBUG
    CMD_DEF  do_capture,DO_CAPTURE
endif

0779 0063  DATA  DO_CAPTURE
077A 065E  DATA  do_capture

CMD_END

077B 0000  DATA  0x00
In PAR_TABLE, the code word is as follows:
Low Byte is # of bytes, Hi Byte is function code

077C 0003  PAR_TABLE  DATA  0x0003
077D 0020  DATA  VL
077E 0103  DATA  0x0103
077F 0023  DATA  AL
0780 0202  DATA  0x0202
0781 0026  DATA  KP
0782 0302  DATA  0x0302
0783 0028  DATA  KV
0784 0402  DATA  0x0402
0785 002A  DATA  KI
0786 0501  DATA  0x0501
0787 002C  DATA  IM
0788 0602  DATA  0x0602
0789 002D  DATA  FV
078A 0702  DATA  0x0702
078B 002F  DATA  FA
078C 0008  DATA  NUMPAR

078D 0423F  DEC_TABLE  DATA  0x423F
078E 000F  DATA  0x000F
078F 869F  DATA  0x869F
0790 0001  DATA  0x0001
0791 270F  DATA  0x270F
0792 0000  DATA  0x0000
0793 03E7  DATA  0x03E7
0794 0000  DATA  0x0000
0795 0063  DATA  0x0063
0796 0000  DATA  0x0000
0797 0009  DATA  0x0009
0798 0000  DATA  0x0000
0799 FFFF  DATA  0xFFFF

if  DECIO

078D 423F  DEC_TABLE  DATA  0x0423F
078E 000F  DATA  0x000F
078F 869F  DATA  0x869F
0790 0001  DATA  0x0001
0791 270F  DATA  0x270F
0792 0000  DATA  0x0000
0793 03E7  DATA  0x03E7
0794 0000  DATA  0x0000
0795 0063  DATA  0x0063
0796 0000  DATA  0x0000
0797 0009  DATA  0x0009
0798 0000  DATA  0x0000
0799 FFFF  DATA  0xFFFF
endif

if _PICMASTER_DEBUG
    include "picmastr.asm" ; PIC-MASTER Debug (data capture) File
endif

#define CaptureAddr  0x8000 ; addr for dummy Table Writes (to TRACE

;********************************************************************************
; NAME:    doCaptureRegs
; DESCRIPTION: Captures Desired Register Values To PIC-MASTER Trace Buffer
; Intended for PICMASTER Demo/debug/servo tuning Purposes Only
; Capture The following registers to Trace Buffer by putting
; A Trace point on a TABLE instruction. Trace only 2nd Cycle
; (a) POSERROR (position error : 16 bits)
; (b) VELERROR (velocity error : 16 bits)
; (c) MPOSITION (measured position value : 24 bits)
; (d) MVELOCITY (measured velocity value : 24 bits)
; (e) POSITION (commanded position : 24 bits)
; (f) VELOCITY (commanded velocity : 24 bits)
; (g) Y (output of servo loop : 32 bits)
; (h) YPWM (output value written to PWM : 10 bits)
Servo Control of a DC-Brush Motor

doCaptureRegs
    movlw (CaptureAddr & 0xff)
    movwf tblptrl
    movlw CaptureAddr/256
    movwf tblptrh
    ; setup table pointer address
    tablwt 0,0,POSError+BO
    tlwt 1,POSError+BI
    ; now table latch = 16 bits contents of POSError
    capPerr
    tablwt 0,0,POSError+BO
    ; perform actual table write of
    capVerr
    tablwt 0,0,VELError+BO
    tlwt 1,VELError+BI
    ; capture Velocity error
    capMpos
    tablwt 0,0,MPosition+BO
    tlwt 1,MPosition+BI
    ; capture measured position
    capPos
    tablwt 0,0,POSITION+BO
    tablwt 0,0,MVelocity+BO
    tlwt 1,MVelocity+BI
    ; capture measured velocity
    capVel
    tablwt 0,0,VELOCITY+B0
    tlwt 1,VELOCITY+BI
    ; capture commanded velocity
    capPwrn
    tablwt 0,0,YPWM+B0
    tlwt 1,YPWM+BI
    ; capture commanded velocity

DEC16 CAPTMP
    CLRw rreg
    DECF CAPTMP+B0
    SUBFB CAPTMP+B1

// TLSZ16 CAPTMP
    // MOVFP CAPTMP+B0, wreg
    // IORMF CAPTMP+B1, W
    // TSTFSZ wreg

    return

    DATA 0x0001
    ; HALT instruction (avail only in
    ; HaltTrace

    CLR CAPFLAG
    MOV16 CAPCOUNT,CAPTMP

    MOVFP CAPCOUNT+B0,wreg
    ; get byte of CAPCOUNT into w
    MOVFP CAPCOUNT+B1,wreg
    ; get byte of CAPCOUNT into w
07BF 01BF

MOVWF CAPTMP+B1 ; move to CAPTMP(B1)

07CO 0002

return

;*****************************************************************************************************************

endif

END

Errors : 0
Warnings : 0
SECTION 5
INTERFACING PIC16/17 WITH SERIAL EEPROMS

Communicating with IC Bus Using PIC16C5X - AN515 ......................................................... 5- 1
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X - AN530 .............................................. 5- 11
Logic Powered Serial EEPROMs - AN535 ............................................................................. 5- 29
INTRODUCTION

The Microchip Technology Inc.'s 24CXX and 85CXX serial EEPROMs feature a two wire serial interface bus. The bus protocol is I²C compatible. Interface to a serial port with I²C bus protocol in a microcontroller is trivial. This application note is intended for design engineers who want to develop their software programs to communicate a microcontroller with a 2-wire bus serial EEPROM through a general purpose I/O port.

Unlike the 3-wire bus serial EEPROMs, the 24CXX/85CXX communicate with any microcontroller only by a serial data I/O line (SDA) and a serial clock (SCL). Chip select is not required. Data transfer may be initiated only when the bus is not busy. During such transfer, the data line (SDA) must remain stable whenever the clock line (SCL) is high. Changes in the data line while the clock line is high are interpreted as a START or STOP condition. A typical transfer format is shown in Figure 1.

After the START condition, a slave address is sent. This address is 7-bits long, the eighth bit is a data direction bit. (R/W - a logical '0' indicates a transmission WRITE, a logical '1' represents a request for data READ. A data transfer is always terminated by a STOP condition generated by the master controller. However, if a master still wishes to communicate on the bus, it can generate another START condition and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such transfer.

![FIGURE 1 - TRANSFER FORMAT](image1)

![FIGURE 2 - A SIMPLE HARDWARE CONNECTION](image2)

NOTE 2: NC PIN FOR 85CXX,
NF PIN FOR 24C01A AND
WP PIN FOR 24C02A/24C04A

NOTE 2: CHIP ADDRESS INPUTS, MUST BE
TIED TO Vcc OR Vss.
Communicating with I²C Bus

An example program has been provided in Appendix A containing all PIC16C54 routines needed to exercise a 24CXX or 85CXX device. A simple hardware connection is illustrated in Figure 2. A maximum of eight 24C01A/24C02A/85C72/85C82's, or four 24C04A/85C92's can be addressed by a microcontroller on the same two wire bus without additional interfaces. Each device is identified by its Chip Address and will only respond to the correct slave address. A detailed bus flow is shown in Figure 3.

Figure 3 as shown below describes how the bit stream is set up for READ and WRITE mode in the microcomputer programming software prior to sending it on the two wire serial bus.

The stop condition, after the write sequence, starts the internal self-timed write cycle which may last up to 6 milliseconds (.7 ms per byte). Acknowledge signal should be monitored during this period.

CONTACT: Bruce Negley
Memory Products Division

**FIGURE 3A - SETTING THE INTERNAL WORD ADDRESS OF THE 24CXX/85CXX**

1. START CONDITION
2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX
3. DEVICE ADDRESS A0=A1, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92
4. 0=WRITE TO ADDRESS REGISTER
5. EEPROM DRIVES BUS LOW WITH ACKNOWLEDGEMENT (TIMEOUT IF NO RESPONSE)
6. ADDRESS FOR DATA, A7-A0 (MSB-LSB)
7. BUS LOW FOR ACKNOWLEDGEMENT

**FIGURE 3B - BYTE WRITE SEQUENCE**

1. START CONDITION
2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX
3. DEVICE ADDRESS A0=A1, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92
4. 1=READ MODE
5. ACKNOWLEDGE FROM EEPROM
6. FIRST DATA BYTE
7. SECOND DATA BYTE, ETC.
8. STOP CONDITION

**FIGURE 3C - READ MODE SEQUENCE**

1. START CONDITION
2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX
3. DEVICE ADDRESS A2=A3,A1,A0, A0=PA, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92
4. 1=READ MODE
5. ACKNOWLEDGE FROM EEPROM
6. FIRST DATA BYTE
7. SECOND DATA BYTE, ETC.
8. STOP CONDITION

READ UP TO 128 BYTES (24C01A, 85C72)
READ UP TO 256 BYTES (24C02A/85A, 85C82/92)
Communicating with I²C Bus

Appendix A:

MPALC CROSS ASSEMBLER 2.00 d:\seeprom\appnotes\i2cbus.asm
Apr 11 15:36:02 1990 PAGE 1

TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

0001 TITLE "TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x"
0002 ;
0003 LIST P=16C54
0004 ;
0005 ;*******************************
0006 ; ** Two wire/I2C Bus READ/WRITE Sample Routines
0007 ; of Microchip's 24CXX/85CXX serial CMOS
0008 ; ** EEPROM interfacing to a PIC16C54 8-bit CMOS
0009 ; ** single chip microcomputer
0010 ; ** Part use = PIC16C54-XT/JW
0011 ; ** Note: 1) All timings are based on a
0012 ; reference crystal frequency of 2 MHz which
0013 ; is equivalent to an instruction cycle
0014 ; ** time of 2 usec.
0015 ; ** 2) Address and literal values are read
0016 ; in octal unless otherwise specified.
0017 ; 3) The following sample program is
0018 ; intended to interface a two wire/I2C
0019 ; serial EEPROM with a PIC16C54 on a
0020 ; stand-alone application only.
0021 ; In the case where the two wire bus is
0022 ; multiplexing with other circuitry, it is
0023 ; recommended to check the 24CXX/85CXX in
0024 ; standby mode to avoid bus contention.

0025 ;
0026 ;*******************************
0027 ; Files Assignment
0028 ;------------------------------------
0029 ;------------------------------------
0030 ;
0031 0021 0002 PC EQU 2 ; Program counter
0032 0022 0004 FSR EQU 4 ; File Select Register
0033 0023 0005 RA EQU 5 ; Port A use to select
0034 ; device address
0035 0024 0006 RB EQU 6 ; RB7 = SDA, RB6 = SCL
0036 ;
0037 0025 ;
0038 0026 0010 STATUS EQU 10 ; Status register
0039 0027 0011 FLAG EQU 11 ; Common flag bits
0040 ; register
0041 0028 0012 EEPROM EQU 12 ; Bit buffer
0042 0029 0013 ERCODE EQU 13 ; Error code (to indicate
0043 ; bus status)
0044 0030 0020 ADDR EQU 20 ; Address register

© 1993 Microchip Technology Inc.

5-3
Communicating with I²C Bus

0031 0021  DATAI  EQU  21 ; Stored data input 
       ; register
0032 0022  DATAO  EQU  22 ; Stored data output 
       ; register
0033 0023  SLAVE  EQU  23 ; Device address 
       ; (1010xxx0)
0034 0024  TXBUF  EQU  24 ; TX buffer
0035 0025  RXBUF  EQU  25 ; RX buffer
0036 0026  COUNT  EQU  26 ; Bit counter
0037  
0038 0030  TIMER0  EQU  30 ; Delay timer0
0039 0031  TIMER1  EQU  31 ; Delay timer1
0040  
0041  
0042  ; --------------------------------------------------------------------------
0043  ; Bit Assignments
0044  ; --------------------------------------------------------------------------
0045  
0046  ;   FLAG Bits
0047  
0048 0000  ERROR   EQU  0   ; Error flag
0049  
0050  ;   EEPROM Bits
0051  
0052 0007  DI     EQU  7   ; EEPROM input
0053 0006  DO     EQU  6   ; EEPROM output
0054  
0055  ;   I2C Device Bits
0056  
0057 0007  SDA    EQU  7   ; RB7, data in/out
0058 0006  SCL    EQU  6   ; RB6, serial clock
0059  
0060  ; END FILES/BITS EQUATE
0061  
0062  
0063  ;--------------------------------------------------------------------------
0064  ; Two wire/I2C - CPU communication error status table 
       ; and subroutine
0065  ;--------------------------------------------------------------------------
0066  ; input : W-reg   = error code
0067  ; output : ERCODE = error code
0068  ;   FLAG(ERROR)  = 1
0069  
0070  ; code   error status mode
0071  ; ------------------------------------------
0072  ;   1 :     SCL locked low by device (bus is still busy)
Communicating with I²C Bus

Subroutine to identify the status of the serial clock (SCL) and serial data (SDA) condition according to the error status table. Codes generated are useful for bus/device diagnosis.

```assembly
0082 ERR
0083 0000 3411 BTFSS FLAG,ERROR ; Remain as first error encountered
0084 0001 0053 MOVWF ERCODE ; Save error code
0085 0002 2411 BSF FLAG,ERROR ; Set error flag
0086 0003 4000 RETLW 0

START bus communication routine

START

 MOVLW B' 00111111' ; Put SCL, SDA line in output state
 TRIS RB
 BSF RB, SCL ; Set clock high
 MOVLW 1 ; Ready error status
 BCF RB, SDA ; SDA goes low during SCL high
 CALL ERR ; SCL locked low by device
 BCF RB, SCL ; Start clock train
 MOVLW B'00111111' ; Put SCL, SDA line in output state
```
Communicating with I²C Bus

; STOP bus communication routine
; Input: None
; Output: Bus communication, STOP condition

; Generate STOP bit (SDA goes from low to high during SCL high state)
; and check bus conditions.

;------------------------------------------------------
STOP bus communication routine
;------------------------------------------------------
Input: None
Output Bus communication, STOP condition

;------------------------------------------------------
·------------------------------------------------------
; Input: None
; Output To (DI) of serial EEPROM device

;------------------------------------------------------
Serial data send from PIC16CXX to serial EEPROM,
; bit-by-bit subroutine

;------------------------------------------------------
; Input: None
; Output : To (DI) of serial EEPROM device

;------------------------------------------------------
BITIN

;------------------------------------------------------
;------------------------------------------------------
;------------------------------------------------------
;------------------------------------------------------
;------------------------------------------------------
;------------------------------------------------------
;------------------------------------------------------
Communicating with I\textsuperscript{2}C Bus

BIT1

BITOUT

BITOUT
Communicating with I²C Bus

0202 0102 2411 BSF FLAG,ERROR ; Set error flag
0203 0103 2110 BIT2
0204 0103 0000 NOP
0205 0104 0000 NOP
0206 0105 2306 BCF RB,SCL ; Return SCL to low
0207 0106 4000 RETLW 0
0208 ;
0209 ;END SUB
0211 ;
0212 ;
0213 ;------------------------------------------------------------------
0214 ; RECEIVE DATA subroutine
0215 ;------------------------------------------------------------------
0216 ; Input : None
0217 ; Output : RXBUF = Receive 8-bit data
0218 ;------------------------------------------------------------------
0219 ;
0220 RX
0221 0107 6010 MOVLW .8 ; 8 bits of data
0222 0110 0066 MOVWF COUNT
0223 0111 0165 CLRF RXBUF
0224 RXLP
0225 1565 RLF RXBUF ; Shift data to buffer
0226 0112 0113 SKPC
0227 0113 3403 + BTFSS 3,0
0228 0114 2025 BCF RXBUF,0 ; carry → f(0)
0229 0115 SKPNC
0230 0115 3003 + BTFSC 3,0
0230 0116 2425 BSF RXBUF,0
0231 0117 4434 CALL BITIN
0232 0120 3352 BTFSC EEPROM,DI
0233 0121 2425 BSF RXBUF,0 ; Input bit = 1
0234 0122 1366 DECFSZ COUNT ; 8 bits?
0235 0123 5112 GOTO RXLP
0236 0124 2712 BSF EEPROM,DO ; Set acknowledge bit = 1
0237 0125 4454 CALL BITOUT ; to STOP further input
0238 0126 4000 RETLW 0
0239 ;
0240 ;END SUB
0241 ;
0242 ; TRANSFER DATA subroutine
0243 ;
0244 ;------------------------------------------------------------------
0245 ; Input : TXBUF
0246 ; Output : Data X'mitted to EEPROM device
0247 ;------------------------------------------------------------------
0248 ;
0249 TX
0250 0127 6010 MOVLW .8
MOVWF COUNT
TXLP
BCF EEPROM, DO ; Shift data bit out.
BTFSC TXBUF, 7 ; If shifted bit = 0, data
; bit = 0
BSF EEPROM, DO ; Otherwise data bit = 1
CALL BITOUT ; Serial data out
RLF TXBUF ; Rotate TXBUF left
SKPC
; f(6) -> f(7)
BTFSS 3, 0
BCF TXBUF, 0
SKPNC
; f(7) -> carry
; carry -> f(0)
BSF TXBUF, 0
DECFSZ COUNT
GOTO CALL TXLP
BITIN
MOVLW 3
; 8 bits done?
BTFSC EEPROM, DI ; Check for
; acknowledgement
CALL ERR ; No acknowledge from
; device
RETLW
END SUB

BYTE-WRITE, write one byte to EEPROM device
----------------------------------------------------
Get SLAVE address
to TX buffer
Generate START bit
Get WORD address
intobuffer
Output WORD address
Move DATA
intobuffer
Communicating with I^2C Bus

0295 0211 4527 CALL TX ; Output DATA and detect 
; acknowledgement
0296 0212 4420 CALL BSTOP ; Generate STOP bit
0297 
0298 
0299 
0300 ;----------------------------------------------------
0301 ; BYTE-READ, read one byte from serial EEPROM 
; device
0302 ;----------------------------------------------------
0303 ; Input : ADDR = source address 
0304 ; SLAVE = device address (1010xxx0) 
0305 ; Output : DATAI = data read from serial 
; EEPROM
0306 ;----------------------------------------------------
0307 
0308 0300 ORG 300 ; The location for BYTE- 
; READ routine can be 
; assigned anywhere 
; between (377-777) octal.
0309 
0310 RDBYTE
0311 0300 1023 MOVF SLAVE,W ; Move SLAVE address 
0312 0301 0064 MOVWF TXBUF ; into buffer (R/W = 0) 
0313 0302 4404 CALL BSTART ; Generate START bit 
0314 0303 4527 CALL TX ; Output SLAVE address. 
; Check ACK.
0315 0304 1020 MOVF ADDR,W ; Get WORD address 
0316 0305 0064 MOVWF TXBUF 
0317 0306 4527 CALL TX ; Output WORD address. 
; Check ACK.
0318 0307 4404 CALL BSTART ; START READ (if only one 
; device 
0319 0310 1023 MOVF SLAVE,W ; is connected to the I^2C 
; bus)
0320 0311 0064 MOVWF TXBUF
0321 0312 2424 BSF TXBUF,0 ; Specify READ mode 
; (R/W = 1)
0322 0313 4527 CALL TX ; Output SLAVE address 
0323 0314 4507 CALL RX ; READ in data and 
; acknowledge
0324 0315 4420 CALL BSTOP ; Generate STOP bit 
0325 0316 1065 MOVF RXBUF ; Save data from buffer 
0326 0317 0061 MOVWF DATAI ; to DATAI file.
0327 
0328 
0329 
0330 END

%ASM-I, No Errors, No Warnings
Microchip Technology Inc.'s popular 93C46/56/66 and 93LC46/56/66 Serial EEPROMs feature a three/four wire serial interface bus. The attractive price and simple interface make it the ideal device for additional memory space. This application note is intended for design engineers who wish to incorporate a pre-packaged serial EEPROM interface driver into their application.

**INTRODUCTION**

The hardware connection

A typical 4-wire hardware connection is illustrated in Figure 1a and a typical 3-wire connection is illustrated in Figure 1b. Since all I/O ports on the PIC16C5X are configurable as input and/or output, a 3-wire interface makes optimum utilization of the I/O pins by having a common connection for the DI and DO lines of the serial EEPROM. The port pin on the PIC16C5X connected to these pins, has a default setting as an output and is configured, when needed, as an input during program execution.

**THE SOFTWARE CONNECTION**

An example interface driver is listed in Appendix A. A flow diagram is given in Figure 2. The interface driver is written to minimize both overhead to the calling program as well as the program space necessary for its inclusion into the user's code. The driver has been written as a generic driver to service all 93 Series serial EEPROMs made by Microchip. Processor resources which must be made available to the driver prior to being called are: 1) Two levels of processor stack. 2) Six register locations (four for command/data passing and two for software counters). 3) The File Select Register (FSR), which is used to pass a command/data string pointer to the driver.

**Note:** The four command/data passing registers have to be defined consecutively in order for the FSR to access them successfully in the program execution.

The user should take the following steps when using the routines provided in Appendix A.

A) Specify and define a 3-/4-wire interface by defining the common connection to the DI/DO lines and setting the equate 'wire3' TRUE or FALSE (4-wire is automatically assumed if 3-wire is false).

B) Specify and define if 16-bit or 8-bit data organization is used, by setting equate 'org8' TRUE or FALSE.

C) The user should assemble the source file by specifying which type of serial EEPROM is being used.

This is done by defining the equate S93C46, S93LC46 etc. as TRUE. Only one device can be TRUE, the rest have to be defined FALSE.

D) The user would invoke the driver as follows:

1. Load the register defined as 'cmd' with the 93CX6 Command Opcode (only the four upper bits are used in this register; see Figure 3).

2. If necessary, load the register defined 'addr' with the lower 8/7/6 bit address of the location.

3. If necessary, load the 9th bit of the address as bit 3 of the register defined as 'cmd' (see Figure 3).

**Note:** READ, WRITE and ERASE commands need to have an address associated with the command and the 9th bit of the address is only
FIGURE 2 - FLOW CHART

- Start
  - Transfer location of register 'cmd' to FSR
  - Select device, clock out start bit
  - Call doutxx, clock out xx bits of cmd/addr
  - Write or erase?
    - Y: Inc. FSR to Register Containing 1st write byte
    - Call dout8, output 1st byte
    - 8 bit mode?
      - Y: Inc. FSR to Register Containing 2nd write byte
      - N: Call dout8, output 2nd byte
      - Erase?
        - Y: Call rdychk wait for rdy status within allocated time
        - N: De-select device
    - N: Read?
      - Y: Inc. FSR to Register Receiving 1st byte
      - Call din8, input 1st byte
      - 8 bit mode?
        - Y: Inc. FSR to Register Receiving 2nd byte
        - N: Call din8, input 2nd byte
      - Erase?
        - Y: Compute completion status (error/no_error) and return
        - N: De-select device
      - N: EWEN?
        - Y: (erase enable)
        - N: WRAL?
          - Y: (write all)
          - N: ERAL?
            - Y: (erase all)
            - N: De-select device
            - Write or erase?
              - Y: Inc. FSR to Register Containing 1st write byte
              - Call dout8, output 1st byte
              - 8 bit mode?
                - Y: Inc. FSR to Register Containing 2nd write byte
                - N: Call dout8, output 2nd byte
                - Erase?
                  - Y: Call rdychk wait for rdy status within allocated time
                  - N: De-select device
                  - Read?
                    - Y: Inc. FSR to Register Receiving 1st byte
                    - Call din8, input 1st byte
                    - 8 bit mode?
                      - Y: Inc. FSR to Register Receiving 2nd byte
                      - N: Call din8, input 2nd byte
                      - Erase?
                        - Y: Compute completion status (error/no_error) and return
                        - N: De-select device
                        - Write or erase?
                          - Y: Inc. FSR to Register Containing 1st write byte
                          - Call dout8, output 1st byte
                          - 8 bit mode?
                            - Y: Inc. FSR to Register Containing 2nd write byte
                            - N: Call dout8, output 2nd byte
                            - Erase?
                              - Y: Call rdychk wait for rdy status within allocated time
                              - N: De-select device
                              - Read?
                                - Y: Inc. FSR to Register Receiving 1st byte
                                - Call din8, input 1st byte
                                - 8 bit mode?
                                  - Y: Inc. FSR to Register Receiving 2nd byte
                                  - N: Call din8, input 2nd byte
                                  - Erase?
                                    - Y: Compute completion status (error/no_error) and return
                                    - N: De-select device
                                    - Write or erase?
                                      - Y: Inc. FSR to Register Containing 1st write byte
                                      - Call dout8, output 1st byte
                                      - 8 bit mode?
                                        - Y: Inc. FSR to Register Containing 2nd write byte
                                        - N: Call dout8, output 2nd byte
                                        - Erase?
                                          - Y: Call rdychk wait for rdy status within allocated time
                                          - N: De-select device
                                          - Read?
                                            - Y: Inc. FSR to Register Receiving 1st byte
                                            - Call din8, input 1st byte
                                            - 8 bit mode?
                                              - Y: Inc. FSR to Register Receiving 2nd byte
                                              - N: Call din8, input 2nd byte
                                              - Erase?
                                                - Y: Compute completion status (error/no_error) and return
                                                - N: De-select device
                                                - Write or erase?
                                                  - Y: Inc. FSR to Register Containing 1st write byte
                                                  - Call dout8, output 1st byte
                                                  - 8 bit mode?
                                                    - Y: Inc. FSR to Register Containing 2nd write byte
                                                    - N: Call dout8, output 2nd byte
                                                    - Erase?
                                                      - Y: Call rdychk wait for rdy status within allocated time
                                                      - N: De-select device
                                                      - Read?
                                                        - Y: Inc. FSR to Register Receiving 1st byte
                                                        - Call din8, input 1st byte
                                                        - 8 bit mode?
                                                          - Y: Inc. FSR to Register Receiving 2nd byte
                                                          - N: Call din8, input 2nd byte
                                                          - Erase?
                                                            - Y: Compute completion status (error/no_error) and return
                                                            - N: De-select device
                                                            - Write or erase?
                                                              - Y: Inc. FSR to Register Containing 1st write byte
                                                              - Call dout8, output 1st byte
                                                              - 8 bit mode?
                                                                - Y: Inc. FSR to Register Containing 2nd write byte
                                                                - N: Call dout8, output 2nd byte
                                                                - Erase?
                                                                  - Y: Call rdychk wait for rdy status within allocated time
                                                                  - N: De-select device
                                                                  - Read?
                                                                    - Y: Inc. FSR to Register Receiving 1st byte
                                                                    - Call din8, input 1st byte
                                                                    - 8 bit mode?
                                                                      - Y: Inc. FSR to Register Receiving 2nd byte
                                                                      - N: Call din8, input 2nd byte
                                                                      - Erase?
                                                                        - Y: Compute completion status (error/no_error) and return
                                                                        - N: De-select device
                                                                        - Write or erase?
                                                                          - Y: Inc. FSR to Register Containing 1st write byte
                                                                          - Call dout8, output 1st byte
                                                                          - 8 bit mode?
                                                                            - Y: Inc. FSR to Register Containing 2nd write byte
                                                                            - N: Call dout8, output 2nd byte
                                                                            - Erase?
                                                                              - Y: Call rdychk wait for rdy status within allocated time
                                                                              - N: De-select device
                                                                              - Read?
                                                                                - Y: Inc. FSR to Register Receiving 1st byte
                                                                                - Call din8, input 1st byte
                                                                                - 8 bit mode?
                                                                                  - Y: Inc. FSR to Register Receiving 2nd byte
                                                                                  - N: Call din8, input 2nd byte
                                                                                  - Erase?
                                                                                    - Y: Compute completion status (error/no_error) and return
                                                                                    - N: De-select device
                                                                                    - Write or erase?
                                                                                      - Y: Inc. FSR to Register Containing 1st write byte
                                                                                      - Call dout8, output 1st byte
                                                                                      - 8 bit mode?
                                                                                        - Y: Inc. FSR to Register Containing 2nd write byte
                                                                                        - N: Call dout8, output 2nd byte
                                                                                        - Erase?
                                                                                          - Y: Call rdychk wait for rdy status within allocated time
                                                                                          - N: De-select device
                                                                                          - Read?
                                                                                              - Y: Inc. FSR to Register Receiving 1st byte
                                                                                              - Call din8, input 1st byte
                                                                                              - 8 bit mode?
                                                                                                 - Y: Inc. FSR to Register Receiving 2nd byte
                                                                                                 - N: Call din8, input 2nd byte
                                                                                                 - Erase?
                                                                                                   - Y: Compute completion status (error/no_error) and return
                                                                                                   - N: De-select device
                                                                                                   - Write or erase?
                                                                                                     - Y: Inc. FSR to Register Containing 1st write byte
                                                                                                     - Call dout8, output 1st byte
                                                                                                     - 8 bit mode?
                                                                                                       - Y: Inc. FSR to Register Containing 2nd write byte
                                                                                                       - N: Call dout8, output 2nd byte
                                                                                                       - Erase?
                                                                                                         - Y: Call rdychk wait for rdy status within allocated time
                                                                                                         - N: De-select device
                                                                                                         - Read?
                                                                                                              - Y: Inc. FSR to Register Receiving 1st byte
                                                                                                              - Call din8, input 1st byte
                                                                                                              - 8 bit mode?
                                                                                                                - Y: Inc. FSR to Register Receiving 2nd byte
                                                                                                                - N: Call din8, input 2nd byte
                                                                                                                - Erase?
                                                                                                                  - Y: Compute completion status (error/no_error) and return
                                                                                                                  - N: De-select device
                                                                                                                  - Write or erase?
                                                                                                                    - Y: Inc. FSR to Register Containing 1st write byte
                                                                                                                    - Call dout8, output 1st byte
                                                                                                                    - 8 bit mode?
                                                                                                                      - Y: Inc. FSR to Register Containing 2nd write byte
                                                                                                                      - N: Call dout8, output 2nd byte
                                                                                                                      - Erase?
                                                                                                                        - Y: Call rdychk wait for rdy status within allocated time
                                                                                                                        - N: De-select device
                                                                                                                        - Read?
                                                                                                                          - Y: Inc. FSR to Register Receiving 1st byte
                                                                                                                          - Call din8, input 1st byte
                                                                                                                          - 8 bit mode?
                                                                                                                            - Y: Inc. FSR to Register Receiving 2nd byte
                                                                                                                            - N: Call din8, input 2nd byte
                                                                                                                            - Erase?
                                                                                                                              - Y: Compute completion status (error/no_error) and return
                                                                                                                              - N: De-select device
                                                              - N: (write all)
    - N: (erase all)
  - N: Read?
    - Y: EWEN?
      - Y: (erase enable)
      - N: ERAL?
        - Y: (erase all)
        - N: WRAL?
          - Y: (write all)
          - N: De-select device
          - Return with no error
required when using the 93C56/56 or 93LC56/66 devices in the 8 bit mode (ORG tied to GND).

4. If necessary, load the register defined as 'highb' and 'lowb' with the 16 bits of data, the most significant byte loaded in 'highb'. In 8 bit mode, 'highb' should be loaded with the 8 bit data.

Note: Only the WRITE and WRAL commands require data to be loaded in the 'highb', 'lowb' locations.

5. Call the driver sub-routine 'see'.

6. Upon completion, the driver will return a completion status in the W register (error/no error). Only commands requiring a status check are capable of returning a valid error/no error status, in all other cases a no error is returned.

7. If the READ command is executed, the 16/8 bit data will be loaded in the 'highb' and 'lowb' registers, where 'highb' contains the MSB in the 16 bit mode and 8 bit data in the 8-bit mode.

The Example interface assumes a 4 MHz oscillator clock which gives us a 1 µS instruction cycle. If a higher clock speed is used, additional NOPs have to be included in the code in order to meet the minimum clock speed requirements of the 93 Series serial EEPROMs (see data sheet for further details).

Listing in Appendix B is for an interface to 93C46 Serial EEPROM only.

SUMMARY

The 93 Series serial EEPROMs are a simple and versatile method of increasing read/write memory capability in a PIC16C5X application. The 'generic' code in Appendix A makes it easy to incorporate in any PIC16C5X application. Any of the 93 series serial EEPROM can be applied, while at the same time using a minimal amount of I/O, code and RAM resources.

Code size: 6 bytes of RAM.

Appendix A listing: 127 bytes program code (max.)

Appendix B listing: 86 bytes program code

AUTHORS: Stan D'Souza
Logic Products Division
Bob Ward
Field Applications/Sales Organization

FIGURE 3 - 'CMD', 'ADD R', DATA BYTE DEFINITION

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Register (cmd)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB of address in 8 bit mode for 93C56/66 (if necessary)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSB of 4 bit opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd bit of opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd bit of opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB of 4 bit opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>A7</td>
<td>A6</td>
<td>A5</td>
<td>A4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address Register (addr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lower 6 bit address for 93C46/56/66 93LC46/56/66</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB of address in 8 bit mode for 93LC46</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB of address in 16 bit mode for 93C56/66</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16 Bit Mode

<table>
<thead>
<tr>
<th>high b</th>
</tr>
</thead>
<tbody>
<tr>
<td>D15</td>
</tr>
<tr>
<td>D8</td>
</tr>
</tbody>
</table>

8 Bit Mode

<table>
<thead>
<tr>
<th>high b</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
</tr>
<tr>
<td>D0</td>
</tr>
</tbody>
</table>

DS00530C-page 3 5-13
Interfacing 93 Series Serial EEPROMs

Appendix A

Microchip Technology 16c5X Assembler 3.03A Wed Feb 26 11:09:07 1992 Page 1

R/W EEPROM

Line PC Opcode

0001 LIST P = 16C54
0002 ;Define Equates:
0003 01FF PIC54 EQU 1FFH
0004 ;
0005 0000 ORG 0
0006 ;
0007 START
0008 0000 OA7B goto main ;run test program
0009 ;
0010 ;
0011 ;
0012 ;
0013 0001 TRUE EQU 1
0014 0000 FALSE EQU 0
0015 0000 S93C46 EQU FALSE
0016 0000 S93LC46 EQU FALSE
0017 0000 S93C56 EQU FALSE
0018 0000 S93LC56 EQU FALSE
0019 0001 S93C66 EQU TRUE
0020 0000 S93LC66 EQU FALSE
0021 0001 wire3 equ TRUE ;for four-wire setup equate to FALSE
0022 0000 org8 EQU FALSE
0023 ;
0024 0001 H16 EQU TRUE
0025 0000 H8 EQU FALSE
0026 0000 LC46 EQU FALSE
0027 0000 XC46 EQU FALSE
0028 ;
0029 ;
0030 ;
0031 ;***********************************************************************
0032 ;* Register Assignments
0033 ;***********************************************************************
0034 ;
0035 0000 indir equ 0 ;Use this register as source/destination
0036 ;for indirect addressing.
0037 0002 pc equ 2 ;PIC Program Counter.
0038 0003 status equ 3 ;PIC Status Register.
0039 0004 fsr equ 4 ;File Select Register.
0040 0005 serial equ 5 ;Port used for 93CX6 control.
0041 ;The following four registers must be
0042 ;located consecutively in memory.
0043 001A cmd equ la ;This register contains the 4 bit
0044 ;command op code for 93CX6 as follows:
0045 ;bit 7 msb of command op code
0046 ;bit 6 next bit of op code
0047 ;bit 5 next bit of op code
0048 ;bit 4 lsb of op code
0049 ;bit 3 A8 of address in case of
0050 ;56/66 in 8 bit mode.
0051 001B addr equ 1b ;memory address of lower 7/8 bits
0052 001C highb equ 1c ;Used in read/write routines to store the
0053 ;upper byte of a 16 bit data word,
0054 ;or the data in a 8 bit data word
0055 001D lowb equ 1d ;Used in read/write routines to store the
0056 ;lower byte of a 16 bit data word,
0057 ;or not used in 8 bit data word.
0058 001E cnthi equ le ;Used as the upper byte of a sixteen bit
0059 ;loop counter in RYCHSK routine.
0060 ;
Interfacing 93 Series Serial EEPROMs

```assembly
0061 001F  cnt   equ    1f ;Used as the lower byte of a sixteen bit
0062 001E  temp_cmd   equ    1e ;loop counter in RDYCHK routine, and
0063    001F  temp_addr equ    1f ;elsewhere as an eight bit counter.
0064
0065 001E  ;doubles as a temp register for cmd
0066
0067  ;*******************
0068 /* Bit Assignments
0069 /* The following assignments are for 3-wire. For 4-wire please assign
0070 /* din and dout to two separate pins.
0071
0072 0000  carry   equ    0 ;Carry Flag of Status Register.
0073 0002  zflag    equ    2 ;Zero Flag of Status Register.
0074
0075 0002  cs       equ    2 ;Port pin tied to CS on 93CX6.
0076 0001  din      equ    1 ;Port pin tied to DI on 93CX6. 3-wire setup
0077 0001  dout     equ    1 ;Port pin tied to DO on 93CX6. 3-wire setup
0078 0003  clock    equ    3 ;Port pin tied to CLK on 93CX6.
0079
0080  ;*******************
0081 /* General Assignments
0082
0083 0000  no_err   equ    0
0084 0001  error    equ    1 ;After issuing a WRITE, ERASE, ERAL, or WRAL
0085 0020  tries    equ    20 ;command, the approximate number of machine
0086 0086  ;cycles X 256 to wait for the RDY status.
0087 0088  ;This value must be adjusted for operating
0088 0089  ;frequencies other than 4 MHz.
0090
0091 0080  read     equ    b'10000000';read command op code
0092 0040  write    equ    b'01000000';write command op code
0093 00C0  erase     equ    b'11000000';erase command op code
0094 0030  ewen      equ    b'00110000';erase enable command opcode
0095 0000  ewds      equ    b'00000000';erase disable command opcode
0096 0020  eral      equ    b'00100000';erase all command op code
0097 0010  wral      equ    b'00010000';write all command op code
0098
0099  ;*******************
0100 /* Macro Definitions
0101
0102 0103  sel       MACRO ;Selects the 93CX6 device
0104 0105  bsf       serial,cs ;Chip Select (CS) = '1' to select
0106 0107  ENDM
0108 0109  dsel      MACRO ;De-select the 93CX6 device.
0110 0111  bcf       serial,cs ;Chip Select (CS) = '0' to de-select
0112 0113  ENDM
0114
0115 0116  strtbt     MACRO ;Issue the Start Bit to the 93CX6.
0117 0118  bsf       serial,din ;Start Bit = '1'.
0119 0120  clkkit     MACRO ;Clock it out.
0121 0122  ENDM
0123 0124  clkit      MACRO ;Clocks a serial data bit into or out
0125 0126  ;of the 93CX6 device.
0127 0128  bsf       serial,clk ;Clock (CLK) = '1'.
0129 0130  nop        ;Adjust the number of nop instructions
0131 0132  ;between the assertion and de-assertion of
0133 0134  ;CLK in proportion to the PIC operating
0135 0136  ;frequency. Refer to the 93CX6 data for the
0137 0138  ;minimum CLK period.
```
Interfacing 93 Series Serial EEPROMs

0127  bcf    serial,clock    ;Clock (CLK) = '0'.
0128  ENDM
0129
0130  ;*******************************************************************************
0131  ;* DOUTx
0132  ;*******************************************************************************
0133  ;doutxx, outputs up to 11 bits of op code/data, depending on whether
0134  ;a 46/56/66 serial eeprom is being used. The number of bits over 8 are
0135  ;saved in the cmd register and the rest in the addr register. Before
0136  ;calling this routine the cmd and the addr registers should be loaded
0137  ;as follows:
0138  ;cmd reg. bits 7|6|5|4|3|2|1|0
0139  ;------------------|--|--|--|--|--|---
0140  ;           X|X|X|X|X|X|X|Y| -> not used
0141  ;           X|X|X|X|X|X|Y|X| -> not used
0142  ;           X|X|X|X|X|X|Y|X|X| -> 9th bit of address if necessary
0143  ;           X|X|X|X|X|X|X|X|X| -> lsb of command op code
0144  ;           X|X|X|X|X|X|X|X|X| -> 3rd bit of command opcode
0145  ;           X|X|X|X|X|X|X|X|X|X| -> 2nd bit of command opcode
0146  ;           X|X|X|X|X|X|X|X|X|X|X| -> mab of command op code
0147  ;
0148  ;addr reg. 6/7/8 bits of address if necessary.
0149
0150  dout10
0151
0152  0001 0360  rlf    indir    ;rotate thru carry
0153  0002 0425  bcf    serial,din    ;set output to 0
0154  0003 0603  btfsc status,carry    ;set?
0155  0004 0525  bcf    serial,din    ;else set output to 1
0156  0005 0000  clkit    ;clk data
0157
0158  m
0159  0006 0000  m
0160  m
0161  0005 0565  m    bcf    serial,clock    ;Clock (CLK) = '1'.
0162
0163  m
0164  m
0165  m
0166  m
0167  0007 0465  bcf    serial,clock    ;Clock (CLK) = '0'.
0168  dout9
0169  0008 0360  rlf    indir    ;rotate thru carry
0170  0009 0425  bcf    serial,din    ;set output to 0
0171  000A 0603  btfsc status,carry    ;set?
0172  000B 0525  bcf    serial,din    ;else set output to 1
0173  000C 0565  clkit    ;clk data
0174
0175  m
0176  m
0177  m
0178  000D 0000  m    bcf    serial,clock    ;Clock (CLK) = '1'.
0179
0180  m
0181  m
0182  m
0183  m
0184  000E 0465  bcf    serial,clock    ;Clock (CLK) = '0'.
0185  000F 02A4  incf    far    ;inc pointer
0186  dout8
0187  0010 0C08  movlw 8    ;Initialize loop counter.
0188  0011 003F  movwf    cnt
0189
0190  d.o.8
0191  0012 0425  bcf    serial,din    ;Assume that the bit to be transfered is a
0192  0013 0360  rlf    indir    ;'0'. Hence, de-assert DI.
0193
0194  0014 0603  btfsc status,carry    ;Test the carry, if our assumption was
0195  correct, skip the next instruction.
Interfacing 93 Series Serial EEPROMs

0263 rdychk
0264                ; is not present after this set period, the
0265                ; routine will return with an error status.
0266                ;
0267                ;
0268 002E 0C02      movlw b'00000010'
0269 002F 0005      ; set up RA1 as a input before proceeding
0270 0030 0C20      tris serial ;
0271 0031 003E      movlw tries ;Initialize time-out counter
0272 0032 007F      clf cnt ;
0273 0033 0445      m bcf serial, cs ; Chip Select (CS) = '0' to de-select
0274 0034 0545      m ; the device.
0275                ;
0276                ;
0277                ;
0278                ;
0279                ;
0280                ;
0281                ;
0282                ;
0283                ;
0284 0035 0625      m bsf serial, cs ; Chip Select (CS) = '1' to select
0285 0036 0A3E      notrdy btsc serial, dout ; If DO is a '0', 93CX6 has yet to completed
0286 0037 02FF      ; the last operation (still busy).
0287 0038 0A35      goto rdynoerr ; skip to no error
0288 0039 02FF      decfsz cnt ; No, not yet ready. Decrement the LSB of our
0289 003A 0A35      goto notrdy ; Still some time left. Try again.
0290 003B 0A35      decfsz cnthi ; Least significant byte expired - decrement
0291 003C 0005      goto notrdy ; and check for expiration of the MSB.
0292 003D 0801      retlw error ; Still some time left. Try again.
0293 003E 0800      ;
0294                ;
0295                ;
0296                ;
0297 003F 0C00      movlw 0 ; set RA1 as output
0298 003F 0005      tris serial ; /
0299 003F 0001      retlw no_err ; RDY status was not present in the allotted
0300                ; time, return with error status.
0301 rdynoerr
0302                ;
0303                ;
0304                ;
0305                ;
0306                ;
0307                ;
0308                ;
0309                ;
0310                ;
0311                ;
0312                ;
0313                ;
0314                ;
0315                ;
0316                ;
0317                ;
0318                ;
0319                ;
0320                ;
0321                ;
0322                ;
0323                ;
0324                ;
0325                ;
0326 0041 021A      movf cmd, w ; save cmd
0327 0042 003E      movwf temp_cmd ;
0328 0043 021B      movf addr, w ; save addr
0329 0044 003F      movwf temp_addr ;
0330 0045 0C1A      movlw cmd ; Load W with the location of the cmd
0331                ; register.

********************************************************************
* See see will control the entire operation of a
* 93CX6 device. Prior to calling the routine,
* load a valid command/memory address into
* the location cmd, and for WRITE or WRAL
* commands, load registers highb and lowb with
* 16 bits of write data. Upon exit, the W
* register will contain the completion status.
* Only 93CX6 instructions which require a
* status check can return with an error as the
* completion status. The values that denote
* the completion status are defined as
* 'error' and 'no err' in the
* general assignments section.
********************************************************************
Interfacing 93 Series Serial EEPROMs

0332 0046 0024 movwf fsr ; Transfer that information into the File
0333 0047 0545 ; Select Register. The fsr now points to
0334 0048 0525 ; location cmd.
0335 0049 0565 ; Select the 93CX6.
0336 004A 0000
0337 004B 0465 ; Chip Select (CS) = '1' to select
0338 004C 06FA ; Send a start bit.
0339 004D 077A ; Start Bit = '1'.
0340 004E 07DA ; Clock it out.
0341 004F OA73 ; Clock (CLK) = '1'.
0342 0050 06BA ; of the 93CX6 device.
0343 0051 069A ; Adjust the number of nop instructions
0344 0052 005C ; between the assertion and de-assertion of
0345 0053 0565 ; CLK in proportion to the PIC operating
0346 0054 0050 ; frequency. Refer to the 93CX6 data for the
0347 0055 021E ; minimum CLK period.
0348 0056 003A ;
0349 0057 0545 ;
0350 0058 003B ;
0351 0059 0546 ;
0352 005A 0525 ;
0353 005B 0565 ;
0354 005C 0525 ;
0355 005D 0545 ;
0356 005E 0525 ;
0357 005F 0565 ;
0358 0060 0525 ;
0359 0061 0545 ;
0360 0062 0565 ;
0361 0063 0545 ;
0362 0064 0565 ;
0363 0065 0545 ;
0364 0066 0565 ;
0365 0067 0545 ;
0366 0068 0545 ;
0367 0069 0545 ;
0368 006A 0545 ;
0369 006B 0545 ;
0370 006C 0545 ;
0371 006D 0545 ;
0372 006E 0545 ;
0373 006F 0545 ;
0374 0070 0545 ;
0375 0071 0545 ;
0376 0072 0545 ;
0377 0073 0545 ;
0378 0074 0545 ;
0379 0075 0545 ;
0380 0076 0545 ;
0381 0077 0545 ;
0382 0078 0545 ;
0383 0079 0545 ;
0384 007A 0545 ;
0385 007B 0545 ;
0386 007C 0545 ;
0387 007D 0545 ;
0388 007E 0545 ;
0389 007F 0545 ;
0390 0080 0545 ;
0391 0081 0545 ;
0392 0082 0545 ;
0393 0083 0545 ;
0394 0084 0545 ;
0395 0085 0545 ;
0396 0086 0545 ;
0397 0087 0545 ;
0398 0088 0545 ;
0399 0089 0545 ;
0400 008A 0545 ;
0401 008B 0545 ;

© 1993 Microchip Technology Incorporated

DS00530C-page 9
Interfacing 93 Series Serial EEPROMs

; Check.
; Increment the File Select Register to point
; to the register receiving the upper byte of
; the incoming 93CX6 data word.
; Input the upper byte.
; Increment the File Select Register to point
; to the register receiving the lower byte.
; Input 8 more bits.
; No further processing required, exit now.
; Increment the File Select Register to point
; to the upper byte of the 16 bit 93CX6 data
; word to be transmitted.
; Output that byte.
; Increment the File Select Register to point
; to the lower byte.
; Output the lower byte of the 16 bit 93CX6
; data word.
; Exit with a status check.

; ************ *** *********** **** * *** * ********* * * **** * * ****************
; * Test Program
; •
; ****** **** *** * *** * ****** **** ************* ** *********** ******** ***** *

; We’ve include a sample program to exercise
; the PIC to 93C66 interface using a simple
; erase, write and verify routine.
; 8 bit organization has been used
; with a 3 wire interface.

; Clear the port tied to the 93C66 device.
; Initialize the data direction register for
; that port.
; Load W with the Erase/Write Enable command.
; Transfer W into cmd register.
; Erase the 93C66.
; Check completion status.
; Test for error condition.
; Yes, bad completion status, error-out.
; Write loop:
; Define the test pattern to be written.
; Load W with the Write command.
; Transfer W into cmd register.
Interfacing 93 Series Serial EEPROMs

0471 0089 0CAA  movlw  tstptrn  ;Initialize the 93C66 data registers with 
0472 008A 003C  movwf  highb  ;write data.
0473 008B 007B  clrf  addr  ;load in high byte only
0474 008C 0941  call  see  ;since 8 bit low byte is ignored
0475 008D 0F01  xorlw  error  ;start at addr 0
0476 008E 0643  btfsc  status, zflag  ;Write data word into 93C66 device.
0477 008F 008A 003C  movwf  highb  ;Check completion status.
0478 008B 007B  clrf  addr  ;Test for error condition.
0479 008C 0A97  goto  errloop  ;Yes, bad completion status, error-out.
0480 0090 03FB  incfsz  addr  ;No, increment the 8 bit memory address 
0481 0091 0A97  goto  testl  ;field.
0482 0092 0AA6  goto  wrt_nxt_pg  ;write another location
0483 0093 003A  mosw  cmd  ;see if all done
0484 0094 0A95  goto  wrt_nxt_pg  ;no then write next page
0485 0095 009C  subwf  highb, w  ;read written data
0486 0096 0957  goto  testl  ;Written.
0487 0097 057A  bsf  addr  ;set page bit
0488 0098 0A97  goto  testl  ;No, write another location.

0489
0490
0491  read_tst  movlw  read  ;Read loop:
0492 0097 0C80  movwf  cmd  ;Load W with the Read command.
0493 0098 003A  mosw  cmd  ;Transfer W into cmd register.
0494 0099 0A97  goto  testl  ;Read addressed data word from 93C66 device.
0495 009A 0CAA  movlw  tstptrn  ;Load W with the pattern written.
0496 009B 000C  subwf  highb, w  ;Verify the data read against what was 
0497 009C 0743  btfss  status, zflag  ;written.
0498 009D 0AA6  goto  errloop  ;Same?
0499 009E 03FB  incfsz  addr  ;No, error-out.
0500 009F 0A99  goto  wrt_nxt_pg  ;Yes, both byte correct, increment the 8 bit 
0501 00A0 003A  mosw  cmd  ;memory address field.
0502 00A1 009C  subwf  highb, w  ;do next byte
0503 00A2 0A99  goto  rd_nxt_pg  ;check page bit
0504 00A3 077A  btfss  cmd, 3  ;all done!!!
0505 00A4 0AA5  goto  allok  ;no then chk next page
0506 00A5 0AA5  goto  wrt_nxt_pg  ;all done!!!
0507 00A6 057A  bsf  cmd, 3  ;set page bit
0508 00A7 0A99  goto  test2  ;No, read another location.
0509
0510 00A8 0AA5  allok  goto  allok  ;Home safe!
0511  ;
0512  ;
0513  errloop  goto  errloop  
0514 00A9 003A  goto  test2  
0515  ;
0516  ;
0517  ;
0518  ;KEY DEFINITIONS
0519  
0520 0000  GPRG  PIC54  
0521 01FF  SYS_RESET  GOTO  START  
0522 0000  ;
0523 0000  END  

0524

© 1993 Microchip Technology Incorporated
Appendix B

Microchip Technology PIC16C5X Assembler 3.03A Wed Feb 26 11:11:15 1992 Page 1

Line PC Opcode

0001 0001
0002 0002
0003 0003
0004 0004
0005 0005
0006 0006
0007 0007
0008 0008
0009 0009
0010 0010
0011 0011
0012 0012
0013 0013
0014 0014
0015 0015
0016 0016
0017 0017
0018 0018
0019 0019
0020 0020
0021 0021
0022 0022
0023 0023
0024 0024
0025 0025
0026 0026
0027 0027
0028 0028
0029 0029
0030 0030
0031 0031
0032 0032
0033 0033
0034 0034
0035 0035
0036 0036
0037 0037
0038 0038
0039 0039
0040 0040
0041 0041
0042 0042
0043 0043
0044 0044
0045 0045
0046 0046
0047 0047
0048 0048
0049 0049
0050 0050
0051 0051
0052 0052
0053 0053
0054 0054
0055 0055
0056 0056
0057 0057
0058 0058
0059 0059
0060 0060
0061 0061
0062 0062
0063 0063

;********************************************************************
;* MPALC Directives Section
;********************************************************************

0000 indir equ 0x00 ;Use this register as source/destination for indirect addressing.
0002 pc equ 0x02 ;PIC16/17 Program Counter.
0003 status equ 0x03 ;PIC16/17 Status Register.
0004 fsr equ 0x04 ;File Select Register.
0005 serial equ 0x05 ;Port used for 93C46 control. Since Port A is 4 bits wide, we'll use 3 or 4 pins of Port A.
0010 cmd equ 0x10 ;The following three registers must be located consecutively in memory.
0011 highb equ 0x11 ;This register contains the 2 bit 93C46 command is the upper 2 bit positions and memory address in the lower 6.
0012 lowb equ 0x12 ;Used in read/write routines to store the upper byte of a 16 bit 93C46 data word.
0013 cnthi equ 0x13 ;Used in read/write routines to store the lower byte of a 16 bit 93C46 data word.
0014 cnt equ 0x14 ;Used as the upper byte of a sixteen bit loop counter in RDYCHK routine.
0018 cs equ 0x18 ;Port pin tied to CS on 93C46.
0019 din equ 0x19 ;Port pin tied to DI on 93C46, 3-wire setup.
001A dout equ 0x1A ;Port pin tied to DO on 93C46, 3-wire setup.
001B clock equ 0x1B ;Port pin tied to CLK on 93C46.

;********************************************************************
;* Register Assignments
;********************************************************************

;Use this register as source/destination for indirect addressing.
;PIC16/17 Program Counter.
;PIC16/17 Status Register.
;File Select Register.
;Port used for 93C46 control. Since Port A is 4 bits wide, we'll use 3 or 4 pins of Port A.

;The following three registers must be located consecutively in memory.
;This register contains the 2 bit 93C46 command is the upper 2 bit positions and memory address in the lower 6.
;Used in read/write routines to store the upper byte of a 16 bit 93C46 data word.
;Used in read/write routines to store the lower byte of a 16 bit 93C46 data word.
;Used as the upper byte of a sixteen bit loop counter in RDYCHK routine.
;Used as the lower byte of a sixteen bit loop counter in RDYCHK routine, and elsewhere as an eight bit counter.

;********************************************************************
;* Bit Assignments: The following assignment is for 3-wire setup.
;********************************************************************

0000 carry equ 0 ;Carry Flag of Status Register.
0002 zflag equ 2 ;Zero Flag of Status Register.
0000 cs equ 0 ;Port pin tied to CS on 93C46.
0001 din equ 1 ;Port pin tied to DI on 93C46, 3-wire setup.
0001 dout equ 1 ;Port pin tied to DO on 93C46, 3-wire setup.
0003 clock equ 3 ;Port pin tied to CLK on 93C46.

;********************************************************************
;* General Assignments
;********************************************************************

0000 no_err equ 0 ;
0001 error equ 1 ;
0004 tries equ 0x04 ;After issuing a WRITE, ERASE, ERAL, or WRAL command, the approximate number of machine cycles X 256 to wait for the RDY status.
0005 tries equ 0x05 ;This value must be adjusted for operating frequencies other than 4 MHz.
0000 read equ 0x80 ;93C46 Read command.
0000 write equ 0x40 ;93C46 Write command.
0000 erase equ 0xC0 ;93C46 Erase command.
0000 ewen equ 0x30 ;93C46 Erase/Write Enable command.
Interfacing 93 Series Serial EEPROMs

;93C46 Erase/Write Disable command.
;92CXX Erase All command.
;92CXX Write All command.

;*****************************************************************************
; Macro Definitions
;*****************************************************************************

MACRO
; Selects the 93C46 device.
bsf serial, cs ; Chip Select (CS) = '1' to select the device.
ENDM

MACRO
; De-select the 93C46 device.
bcf serial, cs ; Chip Select (CS) = '0' to de-select the device.
ENDM

MACRO
; Issue the Start Bit to the 93C46.
bsf serial, din ; Start Bit = '1'.
clkit ; Clock it out.
ENDM

MACRO
; Clocks a serial data bit into or out of the 93C46 device.
bsf serial, clock ; Clock (CLK) = '1'.
ENDM

MACRO
; Adjust the number of nop instructions between the assertion and de-assertion of CLK in proportion to the PIC operating frequency. Refer to the 93C46 data for the minimum CLK period.
bsf serial, clock
nap
bcf serial, clock ; Clock (CLK) = '0'.
ENDM

;*****************************************************************************
; Power-On/Reset Entry Point
;*****************************************************************************

org Ox000
goto main

;*****************************************************************************
; 93C46 Routines
;*****************************************************************************

org Ox000 ; Locate all subroutines in the lower half of a Program Memory Page.

;*****************************************************************************
; DOUT8
;*****************************************************************************

; Dout8 will output 8 bits of data to the 93C46. Before calling this routine, the FSR must point to the byte being transmitted.
movlw Ox08
movwf cnt ;

bcf serial, din ; Assume that the bit to be transferred is a '0'. Hence, de-assert DI.
rlf indir ; Rotate the actual bit to be transferred into the carry bit.
btfsc status, carry ; Test the carry, if our assumption was correct, skip the next instruction.
bsf serial, din ; No, actual bit was a '1'. Assert DI.
clkit ; Clock the 93C46.

;*****************************************************************************
Interfacing 93 Series Serial EEPROMs

: ; between the assertion and de-assertion of 
: ; CLK in proportion to the PIC operating 
: ; frequency. Refer to the 93C46 data for the 
: ; minimum CLK period.

bcf serial,clock ; Clock (CLK) = '0'.
declfz cnt ; Repeat until cnt = 0.
goto d o _ 8 ; Cnt still > 0.
rlf indir ; Restore register to its original condition.
bcf serial,din ; make din low before return
retlw no_err ; Exit with good status.

jeqz ; **********************************************
DIN8 */
jeqz ; **********************************************
din8
jeqz ; **********************************************
movlw b'00000010' ; set up PORTA
tris serial ;
jeqz ; **********************************************
movlw 0x08 ; Initialize loop counter.
movwf cnt ;

reset register as input before proceeding
movlw b'00000010' ; set up PORTA
tris serial ;
reset register as input before proceeding
movlw 0x08 ; Initialize loop counter.
movwf cnt ;

; Clock a bit out of the 93C46.
; 93C46 device.
; Adjust the number of nop instructions
; between the assertion and de-assertion of
; CLK in proportion to the PIC operating
; frequency. Refer to the 93C46 data for the
; minimum CLK period.

bcf serial,clock ; Clock (CLK) = '0'.
rlf indir ; make room for the incoming bit in the
; destination register.
bcf indir,0 ; Assume that the incoming bit is a '0' and
btfsc serial,dout ; test the incoming bit, if our assumption
was correct, skip the next instruction.
bsf indir,0 ; No, actual bit is a '1'. Set the LSB of the
; destination register.

decfsz cnt ; Repeat until cnt = 0.
retlw no_err ; Cnt still > 0

; Restore RA1 back as output
movlw 0 ; set RA1 as output
tris serial ;

; Exit with good status.

; Rdychk will read the 93C46 READY/BUSY status
; and wait for Rdy status within the allotted
; number of processor cycles. If Rdy status
; is not present after this set period, the
; routine will return with an error status.

rdychk ; set up RA1 as a input before proceeding
movlw b'00000010' ; set up PORTA
Interfacing 93 Series Serial EEPROMs

0205 001F 0005 tris serial ; /
0206 movlw tries ; Initialize time-out counter.
0207 0020 0C04 movwf cnthi ;
0208 0021 0033 clrf cnt ;
0210 dsel ; De-select the 93C46.
0211 m
0212 0023 0405 m bcf serial, cs ; Chip Select (CS) = '0' to de-select the
0213 ; device.
0215 ;
0216 ;
0217 ;
0218 ;
0219 ;
0220 ;
0221 ;
0222 m
0223 0024 0505 m bcf serial, cs ; Chip Select (CS) = '1' to select the device.
0224 0025 0625 notrdy btfsc serial, dout ; If DO is a '0', 93C46 has yet to completed
0225 ; the last operation (still busy).
0226 0026 0A2E decfsz cnt ; Skip to no error
0227 0027 02F4 ;
0228 0028 0A25 ;
0229 0029 02F3 ;
0230 002A 0A25 ;
0231 ;
0232 ;
0233 ;
0234 ;
0235 002B 0C00 ;
0236 002C 0005 ;
0237 ;
0238 002D 0801 ;
0239 ;
0240 rdynoerr ;
0241 ;
0242 ;
0243 002E 0C00 ;
0244 002F 0005 ;
0245 ;
0246 0030 0800 ;
0247 ;
0248 ;
0249 ;
0250 ;
0251 ;
0252 ;
0253 ;
0254 ;
0255 ;
0256 ;
0257 ;
0258 ;
0259 ;
0260 ;
0261 ;
0262 ;
0263 ;
0264 ;
0265 ;
0266 ;
0267 0031 0C10 see movlw cmd ; Load W with the location of the cmd
0268 ; register.
0269 0032 0024 movwf fsr ; Transfer that information into the File
0270 ; Select Register. The fsr now points to
0271 ; location cmd.
0272 ; Select the 93C46.
0273 m
0274 0033 0505 bcf serial, cs ; Chip Select (CS) = '1' to select the device.
0275 ; Send a start bit.

© 1993 Microchip Technology Incorporated
DS00530C-page 15
5-25
Interfacing 93 Series Serial EEPROMs

0276
0277 0034 0525 m bsf serial,din ;Start Bit = '1'.
0278 m clkit ;Clock it out.
0279 m ;93C46 device.
0280 0035 0565 m bsf serial,clock ;Clock (CLK) = '1'.
0281 m m
0282 m m
0283 0036 0000 m nop ;Adjust the number of nop instructions
0284 m ;between the assertion and de-assertion of
0285 m ;CLK in proportion to the PIC operating
0286 m ;frequency. Refer to the 93C46 data for the
0287 m ;minimum CLK period.
0288 m
0289 0037 0465 m bcf serial,clock ;Clock (CLK) = '0'.
0290 0038 0900 call dout8 ;Transmit the 2 bit command and six bit
0291 ;address.
0292 0039 0600 btfsc cmd,6 ;Check for a WRITE or ERASE command.
0293 003A 0A43 goto cmd,6 ;Yes, parse the command further.
0294 003B 06F0 btfsc cmd,7 ;Check for a READ command.
0295 003C 0A4D goto read_ ;Yes, process READ command.
0296 003D 06B0 btfsc cmd,5 ;Check for a EWREN or ERAL command.
0297 003E 0A4A goto see3 ;Yes, parse the command further.
0298 003F 0690 btfsc cmd,4 ;Check for a WRAL command.
0299 0040 0A52 goto write_ ;Yes, process WRITE/WRAL command.
0300 0301 exit_dsel ;No further processing required; 93C46
0302 m
0303 0041 0405 m bcf serial,cs ;Chip Select (CS) = '0' to de-select the
0304 ;device.
0305 0042 0800 retlw no_err ;command completed.
0306 0043 0B70 see2 btfss cmd,7 ;Check for a ERASE command.
0307 0044 0A52 goto write_ ;No, process WRITE command.
0308 0045 091E exit2_ call rdychk ;ERASE command requires a status check.
0309 0046 09B2 dsel ;De-select the 93C46.
0310 m
0311 0047 0405 m bcf serial,cs ;Chip Select (CS) = '0' to de-select the
0312 ;device.
0313 0048 01E2 addwf pc ;Compute completion status from results of
0314 ;status check.
0315 0049 0800 retlw no_err ;Return with good completion status.
0316 004A 0801 retlw error ;Return with bad completion status.
0317 0318 0319
0320 004B 0900 see3 btfsc cmd,4 ;Check for a EWREN command.
0321 004C 0A41 goto exit_ ;Yes, no further processing required, exit
0322 ;now.
0323 004D 0A45 goto exit2_ ;No, ERAL command which requires a status
0324 ;check.
0325 0326 02A4 read_ incf fsr ;Increment the File Select Register to point
0327 ;to the register receiving the upper byte of
0328 ;the incoming 93C46 data word.
0329 0330 090E call din8 ;Input the upper byte.
0331 0332 02A4 incf fsr ;Increment the File Select Register to point
0333 ;to the register receiving the lower byte.
0334 0335 090E call din8 ;Input 8 more bits.
0336 0337 0A41 goto exit_ ;No further processing required, exit now.
0338 0339 02A4 write_ incf fsr ;Increment the File Select Register to point
0340 ;to the upper byte of the 16 bit 93C46 data
0341 ;word to be transmitted.
0342 0343 0900 call dout8 ;Output that byte.
0344 0345 02A4 incf fsr ;Increment the File Select Register to point
0346 ;to the lower byte.
0347 0348 0900 call dout8 ;Output the lower byte of the 16 bit 93C46
0349 ;data word.
0350 0351 0A45 goto exit2_ ;Exit with a status check.
Interfacing 93 Series Serial EEPROMs

Test Program

; We've include a sample program to exercise the PIC to 93C46 interface using a simple erase, write and verify routine.

main

clrf

movlw b'11111010' ; Initialize the data direction register for that port.

movlw 0005

tris serial

movlw OC30

call see

movlw b'0030' ; Load W with the Erase/Write Enable command.

movwf cmd

call see

movlw b'0931' ; Enable the 93C46 device.

movlw b'0030' ; Load W with the Erase All command.

movwf cmd

incf loopcnt

movlw b'0030' ; Test for error condition.

btfsc error, OxlF

goto err loop

movlw b'0AA' ; Yes, bad completion status, error-out.

movwf cmd

movlw 0006 ; Define an unused location for our test program loop counter.

loopcnt

equ 0x3F

tstptrn

equ 0xAA

movlw b'0643' ; Define the test pattern to be written.

movlw 001F

tstptrn

equ 0x3F

movwf cmd

movlw b'0030' ; Initialize that counter.

movwf loopcnt

movlw b'004F' ; Load W with the Write command.

movwf cmd

movlw b'0931' ; Initialize the 93C46 data registers with write data.

testl

call see

movlw b'0931' ; Write data word into 93C46 device.

movlw .64

movlw b'003F' ; Check completion status.

movwf loopcnt

movlw b'04F0' ; Test for error condition.

btfsc status, zflag

goto error

movlw b'0082' ; Yes, bad completion status, error-out.

movlw b'0030' ; No, increment the 6 bit memory address field.

movlw b'0060' ; Have we written all 64 locations?

movlw 0006 ; Write another location.

movlw b'02FF' ; Read loop:

tstptrn

equ 0x3F

movwf cmd

movlw b'0030' ; Initialize loop counter.

movwf loopcnt

movlw b'003F' ; Load W with the Read command.

movwf cmd

incf cmd

btfss status, zflag

goto error

movlw b'02B0' ; Yes, both byte correct, increment the 6 bit memory address field.

incf cmd

movlw b'0643' ; Repeat with the lower byte read.

movlw tstptrn

movlw b'0091' ; No, error-out.

subwf highb, 0

movlw b'0743' ; Verify the data read against what was written.

btfss status, zflag

goto error

movlw b'08A2' ; Same?

movlw cmd

subwf lowb, 0

movlw b'0743' ; Repeat with the lower byte read.

movlw status, zflag

goto error

movlw b'0091' ; No, error-out.

movlw b'007A' ; Yes, both byte correct, increment the 6 bit memory address field.

movlw cmd

subwf lowb, 0

movlw b'0743' ; Have we read all 64 locations?

movlw status, zflag

goto error

movlw b'007A' ; No, read another location.

movlw b'0743' ; Home safe!

movlw b'007A' ; Bad news!

movlw b'0000' ; Thats all folks!

END
Embedded applications increasingly want more integration and power, in less space for less cost. Using low power Serial EEPROMs (SEE) for application firmware, lookup tables, and microcode coupled with small footprints makes for permanent storage at respectable savings. One additional method of saving on the power budget is selectively powering off components when not needed, a basic for embedded power management. The low-power SEEs offered by Microchip Technology Inc., offer an additional benefit, powering the SEE from a microcontroller port. This allows the controller to manipulate not only when the SEE reads and writes, but also when it is powered on. Satellite communications use this technique to save power and total dose accumulation. We call this technique POWER PORT™. The microcontroller port must have sufficient loh (source current) to sustain the voltage and current for all memory functions, READ, ERASE, and WRITE. Obviously, not all memory or peripheral devices could be powered thusly, but Microchip’s SEE devices will function in this environment.

The microcontroller, using its internal software and hardware decision functions, determines when it needs to communicate with the memory device, then acts accordingly. Any standard wake-up sequence will accomplish this task. The wake-up code needs only power up the memory and wait for the power to become stable before doing a read or write by driving the POWER PORT high. Then all serial communication executes normally. The SEEs are powered off for additional power savings and the data or code is utilized from RAM. Obviously, the port output must be allowed to settle, but normal operation of the output structures would guarantee that this would be met. The I/O port Tpd for the Microchip Technology Inc. PIC16C5X, is specified at 40ns maximum.

The 24LCXX and 93LCXX CMOS SEE series parts from Microchip were designed to achieve low current consumption across all ranges of operation.

The four primary Icc parameters for these products are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icc STANDBY</td>
<td>Not in an active operation while Vcc is supplied.</td>
</tr>
<tr>
<td>Icc READ</td>
<td>The part is in a READ operation.</td>
</tr>
</tbody>
</table>

| Icc PEAK WRITE | The BYTE / PAGE WRITE and ERASE operations have self timed cycles of 10 ms. A typical of 4 ms is the actual time of the operation. This is the amount of time when the Icc requires the most current (PEAK WRITE). The part is drawing STANDBY Icc during the remaining 6ms of the cycle. |
| Icc AVG WRITE | The avg of the PEAK WRITE Icc and STANDBY Icc during the self-timed 10ms write cycle. |

The attached characteristic curves (Figures A and B) indicate that Icc PEAK WRITE current consumes the most current. The worst case condition is at 6.0V and -40°C. The 24LCXX series parts draw a typical 3.2 mA and the 93LCXX series parts draw a typical of 2.0 mA. These low Icc characteristics offer a unique current saving benefit for battery applications. Figure C and D illustrate the sink and source current capabilities of the PIC16C5X family of controllers. It is clear from these characterization curves that the microcontroller can deliver sufficient current across all temperature ranges to power a SEE using the POWER PORT technique.

Figure E shows the connection scheme for the Microchip Technology Inc. PIC16C54. It should be noted that not all versions of competitive microcontrollers are capable of powering a device in this manner and the specific data sheets for the controller being considered must be consulted for maximum source current. The microcontroller port must be capable of sourcing sufficient current for the duration of the write cycle or 10ms, worse case. The peak write requirement for the 24LCXX product family is 3.2 mA at 5.5 Vdc (-40°C).

Listing A demonstrates the appropriate code sequences when using the PIC16C54 microcontroller. The sequences included are power control, start bit, stop bit, send and receive bit, Tx and Rx, and a general addressing routine.
FIGURE A - TYPICAL Icc FOR 24LCXX

24LCXX Typical Icc PEAK WRITE (mA)

Vcc vs. Icc

24LCXX Typical Icc AVG WRITE (uA)

Vcc vs. Icc

24LCXX Typical Icc READ (uA)

Vcc vs. Icc

24LCXX Typical Icc STANDBY (uA)

Vcc vs. Icc

FIGURE B - TYPICAL Icc FOR 93CXX

93LCXX Typical Icc PEAK WRITE (mA)

Vcc vs. Icc

93LCXX Typical Icc AVG WRITE (uA)

Vcc vs. Icc

93LCXX Typical Icc READ (uA)

Vcc vs. Icc

93LCXX Typical Icc STANDBY (uA)

Vcc vs. Icc
The primary benefits of this application are:

- The SEE is completely powered down to save power when the SEE is not executing an operation. This will directly effect the total system power consumption. This means that the SEE is in a total quiescent state and even the standby current savings is realized, greatly increasing usable battery life, and consequently allowing for a more sophisticated design on the same power budget.
- The very fast 5 µs power-up time minimizes power-up delay.
- Since the serial operation is gated by a stable microcontroller Voh, risk of data being corrupted by a glitch is minimized. This, in effect, is a regulated Vcc supply and provides a reliable power source to ensure data integrity.

Several cautions need to be noted:

1. Gang powering multiple devices must not exceed the IO port Ioh or capacitive load specifications.
2. The total power requirements vs. power budget must be considered, including the extra drain on the microcontroller.
3. The microcontroller Icc max must not be exceeded.
4. Normal decoupling methods must be employed.
5. The microcontroller Ioh for the port in use must not be exceeded.

Figure F shows a typical power on to start bit sequence. Notice that the device is available to receive a clock at 5 µs after Vcc has become stable.

Many applications, especially remote or handheld data acquisition applications, where power consumption is at a premium or battery life is critical can use the POWER PORT technique with the PIC16/17 microcontrollers and possibly other microcontrollers. Remote metering applications where the controller must wake up and report previously stored data or periodically sample inputs, such as gas, electrical, or water monitoring systems are good examples where POWER PORT would be beneficial. Underground monitoring equipment for fuel storage and environmental monitoring systems are also suitable applications.

AUTHORS: Richard J. Fisher
Memory Products Division
Bruce Negley
Memory Products Division
LISTING A

LIST P-16C54

; Sample test program to power up serial EEPROM
; using PIC16/17 port A, then write one byte and read same byte, then repeat forever.

;*******************************************************************************************

port_a equ Sh ; port 5 used for device
    ; address select
port_b equ 6h ; port 6 used for data and
    ; clock lines
eeprorn equ 0ah ; buffer
addr equ 0ch ; address register
datai equ 0dh ; stored data input reg.
datao equ 0eh ; stored data output reg.
slave equ 0fh ; device address
    ; (1010xxxx0)
txbuf equ 10h ; tx buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
rxbuf equ 13h ; receive buffer
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter 2

; Bit Assignments

d equ 7 ; eeprorn input
do equ 6 ; eeprorn output
data equ 7 ; data line (port b)
sclk equ 6 ; clock line (port b)
vcc equ 3 ; vcc for dut (port a)

org Olffh
begin goto PWRUP
org 000h
goto PWRUP

;*******************************************************************************************

DELAY ROUTINE
; this routine takes the value in loops and loops that many times. Every
; increase in 'loops' yields approx 1 more millisecond.
; i.e., if 'loops' is 10 then the wait period is approx 10 milliseconds.

;-------------------------------------------------------------------------

WAIT
;

top2 movlw .110
    movwf loops2
top
    nop ; sit and wait
    nop
    nop
    nop
    nop
    decfsz loops2 ; inner loop done?
goto top ; no, go again
    decfsz loops ; outer loop done?
goto top2 ; no, go again
    retlw 0 ; yes, return from sub
Logic Powered Serial EEPROMs

;******************************************************************************
;  Start Bit Subroutine
;  this routine generates a start bit
;******************************************************************************

; BSTART

movlw b'00111111'
trs port_b ; port_b for output
bsf port_b, sdata ; set clock high
nop
nop
bsf port_b, sclk ; set clock high
nop
nop
nop
nop
nop
nop
nop
bcf port_b, sdata ; data line goes low during high clock for start bit
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
bcf port_b, sclk ; start clock train
nop
nop
nop
retlw 0

; End of Subroutine

;******************************************************************************
;  Stop Bit Subroutine
;  this routine generates a stop bit
;******************************************************************************

; BSTOP

movlw b'00111111' ;
trs port_b ; set data/clock lines as outputs
bcf port_b, sdata ; make sure data line is low
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
bcf port_b, sclk ; set clock high
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
retlw 0
Logic Powered Serial EEPROMs

; End of Subroutine
;******************************************************
; Serial data send 1 bit from PIC16/17 to dut
;--------------------------------------------------------
; BITOUT
movlw b'00111111'; set data, clock as outputs
tris port_b
btfss eeprom,do
goto BIT0
bsf port_b,sdata; output bit 0
goto CLK1; data line clocked low by device

; BIT0
bcf port_b,sdata; output bit 0

CLK1
nop
nop
bsf sclk; set clock line high

BIT2
nop
nop
nop
bcf sclk; return clock line low
retlw 0

; End of Subroutine
;
;******************************************************
; Bit in routine
; this routine gets a bit of data from the part
; into the 'eeprom' register, bit 'di'
;--------------------------------------------------------

BITIN
movlw b'10111111'; make sdata an input line
tris port_b
bcf eeprom,di; assume input bit low
bsf port_b,sclk; set clock line high
nop; just sit here a sec
nop
nop
nop
nop
btfsc port_b,sdata; read data line
bsf eeprom,di; set input bit if needed
bcf port_b,sclk; set clock line low
retlw 0; hit the road

;******************************************************
; Transmit Data Subroutine
;--------------------------------------------------------

TX
movlw .8
movwf count; set the #bits to 8

TXLP
bcf eeprom,do
btfsc txbuf,7
bsf eeprom,do; otherwise data bit -1
call BITOUT; serial data out
set txbuf; rotate txbuf left
decfsz count; 8 bits done?
Logic Powered Serial EEPROMs

goto TXLP ; no - go again
all BITIN ; read ack bit
;
retlw 0
; End of Subroutine
;*******************************************************************************************

Receive data Routine
; this routine gets a byte of data from the part into 'rxbuf'
RX
movlw .8 ; set # bits to 8
movwf count

dtf
RXLP
rlE
RX
bcf rxbuf, 0 ; assume bit is zero
aill BITIN ; read bit
btfsc eeprom, di ; input bit high?
bsf rxbuf, 0 ; yes, set buffer bit high
decfsz count ; 8 bits done?
goto RXLP ; no, do another
bcf eeprom, do ; set ack bit = 0
aill BITOUT ; to finish transmission
retlw 0
;
;*******************************************************************************************

Power up routine
; this routine blinks the lights
PWRUP
movlw b'00000001'
tris port_a ; set RA0 as input, rest output
bhf port_a, vcc ; turn on power to dut
nop
nop
nop
nop
nop

;*******************************************************************************************

Byte Write Routine
; this writes the data in "55h" to the first byte
in the serial EEPROM.

; WRBYTE

movlw b'10100000' ; set slave address and write mode
movwf slave
movlw b'01010101' ; set data to 55h
movwf datao
;
dtf
addr ; set address to 00h
;
all BSTART ; generate start bit
movf slave,w ; get slave address
movwf txbuf ; into transmit buffer
all TX ; and send it
movf addr,w ; get word address
movwf txbuf ; into transmit buffer
all TX ; and send it
movf datao,w ; move data
movwf txbuf ; to transmit buffer
all TX ; and transmit it
all BSTOP ; generate stop bit
;
movlw .10

© 1993 Microchip Technology Incorporated
movwf loops ; set delay time to give
call WAIT ; 10 ms wait after every byte
;
; now drop through and do the read

;******************************************************************************
;
; READ (read routine)
; this routine reads the first address
; of the dut
;
READ
;
movlw b'10100000' ; set slave address and write mode
movwf slave
;
chf addr ; set address to 00h
;
call BSTART ; generate start bit
nop
nop
movf slave,w ; get slave address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; get word address
movwf txbuf ; into transmit buffer
call TX ; and send it
nop
nop
call BSTART ; generate start bit
nop
nop
movlw b'10100001' ; set slave address and read mode
movwf txbuf ; into transmit buffer
call TX ; and transmit it
nop
call RX ; get 8 bits of data
call BITOUT ; send high ack bit and then a

call BSTOP ; stop bit to end transmission from dut
nop
nop
nop
nop
bcf port_a,vcc ; turn power to dut off
movlw .100
movwf loops
call WAIT ; wait awhile
goto PWRUP ; go do the whole thing over again
;
END
SECTION 6
SERIAL EEPROMS
TUTORIALS AND APPLICATION NOTES

Basic Serial EEPROM Operation - AN536 ................................................................. 6- 1
Everything a System Engineer Needs to Know About Serial EEPROM Endurance - AN537 ................................................................. 6- 15
Using the Microchip Endurance Predictive Software - AN562 .................................. 6- 23
Interfacing 24LCXX Serial EEPROMs to the PIC16C54 - AN567 .............................. 6- 27
Using the 24C65 and 24C32 with Stand-alone PIC16/17 Code - AN558 ................... 6- 51
24C01A Compatibility Issues and Its Mobility for Memory Upgrade - AN517 ........... 6- 91
Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559 .............. 6- 93
Using the 93LC56 and 93LC66 - AN560 ............................................................... 6- 97
ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues - AN516 ............ 6- 115
1.8 Volt Technology - Benefits .............................................................................. 6- 117
Serial EEPROM Solutions vs. Parallel Solutions .................................................. 6- 119
Basic Serial EEPROM Operation

BASIC SERIAL EEPROM OPERATION

Looking for the optimum nonvolatile memory product for your system that requires a small footprint, byte level flexibility, low power, and is highly cost effective? Serial EEPROM technology is one of the nonvolatile memory technologies that has emerged as a leading embedded control solution. Unfortunately, most system designers are not aware of the Serial EEPROM benefits. Also, the supporting documentation in data books is most often not adequate due to incomplete or ambiguous information. As a result, the system designer often selects a nonvolatile solution that does not meet their requirements or the designer must face a more complicated design-in with a Serial EEPROM.

This article is to address two issues that exist today for designers considering Serial EEPROM products:

First, to provide awareness of the application benefits.

Second, to provide a primer on the operating principles and instructions. These items are often buried in data book text or not adequately addressed. Also included are common default conditions to significantly reduce the system designers learning curve.

TABLE OF CONTENTS

Serial EEPROM Applications
Overview of the Primary Protocol Benefits
3 Wire Bus Operation Primer
2 Wire Bus Operation Primer
Microchip 2 Wire Default Conditions
Timing Diagram Attachments

SERIAL EEPROM APPLICATIONS

Serial EEPROMS are ideal nonvolatile cost effective memory solutions in applications that require:

Small footprint and board space as in cellular phone applications;

BYTE level ERASE, WRITE, and READ of data as in a TV tuner;

Low voltage and current for hand held battery applications as in a keyless entry transmitter;

Multiple nonvolatile functions in the same application such as a VCR;

Low availability of microcontroller I/O lines.

The common applications for Serial EEPROMS are shown below:

<table>
<thead>
<tr>
<th>Market</th>
<th>Common Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer</td>
<td>TV tuners, VCRs, CD players, cameras, radios, and remote controls</td>
</tr>
<tr>
<td>Automotive</td>
<td>Airbags, anti-lock brakes, odometers, radios, and keyless entry</td>
</tr>
<tr>
<td>Office Automation</td>
<td>Printers, copiers, PC's, and portable PC's</td>
</tr>
<tr>
<td>Telecom</td>
<td>Cellular, cordless and full feature phones, Faxes, modems, pagers, and satellite receivers</td>
</tr>
<tr>
<td>Industrial</td>
<td>Bar code readers, point of sale terminals, smart cards, lock boxes, garage door openers, and test measurement equipment</td>
</tr>
</tbody>
</table>

The typical functions that Serial EEPROMs are utilized for are:

- Memory storage of channel selectors or analog controls (volume, tone, etc,) in consumer electronics products.
- Power down storage and retrieval of events such as fault detection or error diagnostics in automotive products.
- Electronic real time event or maintenance logs such as page counting in office automation products. Also, configuration or dip switch storage in office automation products.
- Last number redial storage and speed dial number storage in telecom products.
- User in-circuit reprogrammable look up tables such as bar code readers, point of sale terminals, environmental controls and other industrial products.

Other application examples include:

- Data storage from a learn function as in a remote control transmitter.
- ID number storage for security or remote access for electronic keys and entry databases.
- Reprogrammable calibration data for test equipment or analog interface products.

© 1993 Microchip Technology Inc.
Basic Serial EEPROM Operation

As a result of density and architectural evolution, Serial EEPROMs offer significant benefits in some applications that previously could only utilize Parallel EEPROM products. The diagram below illustrates the footprint and board space differences.

The Serial EEPROM requires only 10% of the board space that a Parallel EEPROM requires. Also, the Serial EEPROM requires fewer I/O lines from the microcontroller which significantly reduces the overall system cost and board space.

A very fast READ speed is the only significant limitation of a Serial EEPROM for a decision between a Serial and a Parallel EEPROM. It is very interesting to note that the Serial EEPROM READ speed is restricted more by the protocol than the process technology. The 2-wire I²C (Inter-Integrated Circuit) products must add large internal delays to slow down the part to meet the 100 kHz protocol requirements, which will be reviewed later. Characterization of 3-wire bus Serial EEPROMs have indicated clock frequencies in excess of 6 MHz.

OVERVIEW OF THE PRIMARY PROTOCOL BENEFITS

After a designer decides to use a Serial EEPROM solution, the next step is to select one of the two primary Serial EEPROM protocols. Unfortunately, most system designers select the type of Serial EEPROM (2 or 3-wire) that they are most familiar with, regardless of the benefits associated with each type.
The benefits of each protocol are shown below:

<table>
<thead>
<tr>
<th>3 Wire Bus Serial EEPROMs</th>
<th>2 Wire Bus Serial EEPROMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single VDD supply of &lt;2V to 5.5V</td>
<td>Single VDD supply of &lt;2V to 5.5V</td>
</tr>
<tr>
<td>Very low current consumption</td>
<td>Very low current consumption</td>
</tr>
<tr>
<td>Reduced overall component cost</td>
<td>Reduced overall component cost</td>
</tr>
<tr>
<td>Four pins (other than Vcc &amp; GND) are required or operation</td>
<td>Two pins (other than Vcc &amp; GND) are required for operation</td>
</tr>
<tr>
<td>X16 bit and X8 bit data widths</td>
<td>X8 data bit width</td>
</tr>
<tr>
<td>Software WRITE Protection</td>
<td>Hardware WRITE Protection</td>
</tr>
<tr>
<td>Edge triggered clocks and signals</td>
<td>Level triggered clocks and signals and input glitch filters for high noise immunity</td>
</tr>
<tr>
<td>2Mhz+ operation</td>
<td>I²C standard 100 KHz and 400 KHz protocols with a 1 MHz option</td>
</tr>
<tr>
<td>Ready/Busy data polling</td>
<td>Page WRITE capability to 16 bytes</td>
</tr>
<tr>
<td>Security options available</td>
<td>Software and hardware compatible from 2k to 16K densities</td>
</tr>
<tr>
<td>Less complex protocol</td>
<td></td>
</tr>
</tbody>
</table>

A 2 wire product is utilized in applications that require an I²C bus, noise immunity, limited microcontroller I/O pin availability, or a WRITE buffer for multiple bytes to be stored with one instruction. A 3 wire product is utilized in applications that have limited protocol requirements, an SPI protocol, higher clock frequency requirements, or an x16 data width applications.

The next two sections describe the basic operation and Microchip's default conditions for the 3-wire and 2-wire Serial EEPROMs to allow the system designer to utilize the benefits of Serial EEPROMs.

3 WIRE BUS OPERATION PRIMER

Many Serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are unfortunately either vaguely embedded in the data sheet text or not included. Serial EEPROMs are not conventional memories due to the Serial communication protocols involved. This section is a PRIMER for the data sheet to familiarize the system designer with the basic principles of the 3 wire bus operation.

Basic Principles

Common device nomenclature is 93XXXX.

The 93XX06 is a 256 bit product.
The 93XX46 is a 1k bit product.
The 93XX56 is a 2k bit product.
The 93XX66 is a 4k bit product.

Four pins are required:
- CS (Chip Select)
- DI (data in)
- CLK (Clock)
- DO (data out)

All 93XXXX parts are hardware compatible for these four pins. However, there may be compatibility issues for the other pins.

SOFTWARE compatibility is a key issue since there may be subtle differences in each manufacturer's protocol. These subtle differences referred to as default conditions are addressed later in the attached 3 Wire Timing Diagram examples for Microchip's products. Software compatibility for density migration should also be reviewed by the designer on a case by case basis. There is no software industry compatibility for a 256 bit part to a 4k part.

Data is available in x8 or x16 organizations. This selection is determined either by the ORG pin or by purchasing a standard x16 organization.

Units will power-up in an EWDS (ERASE/WRITE Disable State). All ERASE and WRITE functions are disabled until the EWEN (ERASE/WRITE Enable) instruction is performed. This is to prevent accidental data corruption.

An Auto-ERASE (logical “1”) cycle is performed during each WRITE Cycle.

The 7 instructions are shown in the attached instruction set table. These instructions are for Microchip's 93LCXX family products.

After an instruction is loaded, the CLK and DI pins are in a DON'T CARE state until the next START bit.
Basic Serial EEPROM Operation

The following is required for each instruction set (all input bits are triggered by the positive clock edges):

| Start Bit | The first Data-in high signal clocked in after CS is high. |
| Opcode | Two Bits to identify the instruction |
| Address | Refer to the Instruction Set table for the number of bits required. |
| Data | Separate data-in and data-out pins. However, these two pins may be tied together for true 3-wire operation. Please refer to the attached 3-wire Bus READ timing diagram example. |

READ, WRITE, and ERASE

The attached 93LC66 timing diagrams illustrate the key concepts and timing parameters for each of these operations. Please refer to the instruction set tables and the AC parameters in the databook for supplemental information.

**ERASE ALL (ERAL)**

An ERASE ALL (ERAL) operation is identified by a "00" opcode. The ERAL instruction requires the next two bits to be clocked in as "10" in the address block of the instruction set. All bits in the array will be set to a logic "1" state by one command in typically less than 10ms.

**WRITE ALL (WRAL)**

A WRITE ALL (WRAL) operation is also identified by a "00" opcode. The WRAL requires the next two bits to be clocked in as "01" in the address block of the instruction set. The data-in block will contain the data for a SINGLE BYTE which is to be repeated throughout the entire array. For example, if a 4F5A is loaded in the 16 data-in bits of the instruction set, a 4F5A will be written into every word in the array.

**EWEN and EWDS**

As stated before, all units will power up in to an ERASE/WRITE DISABLE (EWDS) state to prevent data corruption. All future ERASE/WRITE operations must execute an ERASE/WRITE ENABLE (EWEN) op-code until the next power down is detected or until other EWDS opcodes are executed. Please refer to the instruction set table.
Basic Serial EEPROM Operation

### INSTRUCTION SET FOR 93LC46: ORG = 1 (x 16 organization)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>SB</th>
<th>Opcode</th>
<th>Address</th>
<th>Data In</th>
<th>Data Out</th>
<th>Req. CLK Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1</td>
<td>10</td>
<td>A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>D15 - D0</td>
<td>25</td>
</tr>
<tr>
<td>EWEN</td>
<td>1</td>
<td>00</td>
<td>1 1 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>9</td>
</tr>
<tr>
<td>ERASE</td>
<td>1</td>
<td>11</td>
<td>A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>9</td>
</tr>
<tr>
<td>ERAL</td>
<td>1</td>
<td>00</td>
<td>1 0 X X X X</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>9</td>
</tr>
<tr>
<td>WRITE</td>
<td>1</td>
<td>01</td>
<td>A5 A4 A3 A2 A1 A0</td>
<td>D15 - D0</td>
<td>(RDY/BSY)</td>
<td>25</td>
</tr>
<tr>
<td>WRAL</td>
<td>1</td>
<td>00</td>
<td>0 1 X X X X</td>
<td>D15 - D0</td>
<td>(RDY/BSY)</td>
<td>25</td>
</tr>
<tr>
<td>EWDS</td>
<td>1</td>
<td>00</td>
<td>0 0 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>9</td>
</tr>
</tbody>
</table>

### INSTRUCTION SET FOR 93LC46: ORG = 0 (x 8 organization)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>SB</th>
<th>Opcode</th>
<th>Address</th>
<th>Data In</th>
<th>Data Out</th>
<th>Req. CLK Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1</td>
<td>10</td>
<td>A6 A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>D7 - D0</td>
<td>18</td>
</tr>
<tr>
<td>EWEN</td>
<td>1</td>
<td>00</td>
<td>1 1 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>10</td>
</tr>
<tr>
<td>ERASE</td>
<td>1</td>
<td>11</td>
<td>A6 A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>10</td>
</tr>
<tr>
<td>ERAL</td>
<td>1</td>
<td>00</td>
<td>1 0 X X X X</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>10</td>
</tr>
<tr>
<td>WRITE</td>
<td>1</td>
<td>01</td>
<td>A6 A5 A4 A3 A2 A1 A0</td>
<td>D7 - D0</td>
<td>(RDY/BSY)</td>
<td>18</td>
</tr>
<tr>
<td>WRAL</td>
<td>1</td>
<td>00</td>
<td>0 1 X X X X</td>
<td>D7 - D0</td>
<td>(RDY/BSY)</td>
<td>18</td>
</tr>
<tr>
<td>EWDS</td>
<td>1</td>
<td>00</td>
<td>0 0 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>10</td>
</tr>
</tbody>
</table>

### INSTRUCTION SET FOR 93LC56: ORG = 1 (x 16 organization)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>SB</th>
<th>Opcode</th>
<th>Address</th>
<th>Data In</th>
<th>Data Out</th>
<th>Req. CLK Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1</td>
<td>10</td>
<td>X A6 A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>D15 - D0</td>
<td>27</td>
</tr>
<tr>
<td>EWEN</td>
<td>1</td>
<td>00</td>
<td>1 1 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>11</td>
</tr>
<tr>
<td>ERASE</td>
<td>1</td>
<td>11</td>
<td>X A6 A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>11</td>
</tr>
<tr>
<td>ERAL</td>
<td>1</td>
<td>00</td>
<td>1 0 X X X X</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>11</td>
</tr>
<tr>
<td>WRITE</td>
<td>1</td>
<td>01</td>
<td>X A6 A5 A4 A3 A2 A1 A0</td>
<td>D15 - D0</td>
<td>(RDY/BSY)</td>
<td>27</td>
</tr>
<tr>
<td>WRAL</td>
<td>1</td>
<td>00</td>
<td>0 1 X X X X</td>
<td>D15 - D0</td>
<td>(RDY/BSY)</td>
<td>27</td>
</tr>
<tr>
<td>EWDS</td>
<td>1</td>
<td>00</td>
<td>0 0 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>11</td>
</tr>
</tbody>
</table>

### INSTRUCTION SET FOR 93LC56: ORG = 0 (x 8 organization)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>SB</th>
<th>Opcode</th>
<th>Address</th>
<th>Data In</th>
<th>Data Out</th>
<th>Req. CLK Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1</td>
<td>10</td>
<td>X A7 A6 A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>D7 - D0</td>
<td>20</td>
</tr>
<tr>
<td>EWEN</td>
<td>1</td>
<td>00</td>
<td>1 1 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>12</td>
</tr>
<tr>
<td>ERASE</td>
<td>1</td>
<td>11</td>
<td>X A7 A6 A5 A4 A3 A2 A1 A0</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>12</td>
</tr>
<tr>
<td>ERAL</td>
<td>1</td>
<td>00</td>
<td>1 0 X X X X</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>12</td>
</tr>
<tr>
<td>WRITE</td>
<td>1</td>
<td>01</td>
<td>X A7 A6 A5 A4 A3 A2 A1 A0</td>
<td>D7 - D0</td>
<td>(RDY/BSY)</td>
<td>20</td>
</tr>
<tr>
<td>WRAL</td>
<td>1</td>
<td>00</td>
<td>0 1 X X X X</td>
<td>D7 - D0</td>
<td>(RDY/BSY)</td>
<td>20</td>
</tr>
<tr>
<td>EWDS</td>
<td>1</td>
<td>00</td>
<td>0 0 X X X X</td>
<td>—</td>
<td>High-Z</td>
<td>12</td>
</tr>
</tbody>
</table>

### INSTRUCTION SET FOR 93LC66: ORG = 1 (x 16 organization)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>SB</th>
<th>Opcode</th>
<th>Address</th>
<th>Data In</th>
<th>Data Out</th>
<th>Req. CLK Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1</td>
<td>10</td>
<td>A7 - A0</td>
<td>—</td>
<td>D15 - D0</td>
<td>27</td>
</tr>
<tr>
<td>EWEN</td>
<td>1</td>
<td>00</td>
<td>11XXXXXX</td>
<td>—</td>
<td>High-Z</td>
<td>11</td>
</tr>
<tr>
<td>ERASE</td>
<td>1</td>
<td>11</td>
<td>A7 - A0</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>11</td>
</tr>
<tr>
<td>ERAL</td>
<td>1</td>
<td>00</td>
<td>10XXXXXX</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>11</td>
</tr>
<tr>
<td>WRITE</td>
<td>1</td>
<td>01</td>
<td>A7 - A0</td>
<td>D15 - D0</td>
<td>(RDY/BSY)</td>
<td>27</td>
</tr>
<tr>
<td>WRAL</td>
<td>1</td>
<td>00</td>
<td>01XXXXXX</td>
<td>D15 - D0</td>
<td>(RDY/BSY)</td>
<td>27</td>
</tr>
<tr>
<td>EWDS</td>
<td>1</td>
<td>00</td>
<td>00XXXXXX</td>
<td>—</td>
<td>High-Z</td>
<td>11</td>
</tr>
</tbody>
</table>

### INSTRUCTION SET FOR 93LC66: ORG = 0 (x 8 organization)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>SB</th>
<th>Opcode</th>
<th>Address</th>
<th>Data In</th>
<th>Data Out</th>
<th>Req. CLK Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1</td>
<td>10</td>
<td>A8 - A0</td>
<td>—</td>
<td>D7 - D0</td>
<td>20</td>
</tr>
<tr>
<td>EWEN</td>
<td>1</td>
<td>00</td>
<td>11XXXXXX</td>
<td>—</td>
<td>High-Z</td>
<td>12</td>
</tr>
<tr>
<td>ERASE</td>
<td>1</td>
<td>11</td>
<td>A8 - A0</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>12</td>
</tr>
<tr>
<td>ERAL</td>
<td>1</td>
<td>00</td>
<td>10XXXXXX</td>
<td>—</td>
<td>(RDY/BSY)</td>
<td>12</td>
</tr>
<tr>
<td>WRITE</td>
<td>1</td>
<td>01</td>
<td>A8 - A0</td>
<td>D7 - D0</td>
<td>(RDY/BSY)</td>
<td>20</td>
</tr>
<tr>
<td>WRAL</td>
<td>1</td>
<td>00</td>
<td>01XXXXXX</td>
<td>D7 - D0</td>
<td>(RDY/BSY)</td>
<td>20</td>
</tr>
<tr>
<td>EWDS</td>
<td>1</td>
<td>00</td>
<td>00XXXXXX</td>
<td>—</td>
<td>High-Z</td>
<td>12</td>
</tr>
</tbody>
</table>
Basic Serial EEPROM Operation

2 WIRE BUS OPERATION PRIMER

As indicated in the 3 wire bus section, many Serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are unfortunately either vaguely embedded in the data sheet text or not included. This section is a PRIMER for the data sheet to familiarize the system designer with the basic 2 wire Serial EEPROM operation principles.

Basic Principles

The common device nomenclature is 24XXXX and 85XXXX.

Only the SCL and SDA pins are essential for bus operation. The other pins are supplementary:

- SCL (Serial clock)
- SDA (Serial Data)
- WP (Active High WRITE Protection)
- A0, A1, and A2 (Chip or block select)

SDA's open-drain requires a pull-up resistor to VDD.

The data is organized as XS.

Signals are level triggered, not edge triggered. Also, there are filters on the inputs that will filter noise glitches <100ns wide.

An Auto-ERASE (logical "1") cycle is performed during each WRITE cycle.

The I²C protocol utilizes master / slave bi-directional communication. A device that sends data onto the bus is defined as the transmitter, and a device that is receiving data is the receiver. Both the master and the slave can operate as the transmitter or receiver. The bus must be controlled by a master device (most often a microcontroller), which generates the serial clock (SCL), controls the bus direction, and generates the START and the STOP conditions.

The Serial EEPROM is the slave. The Serial EEPROM will be the bus transmitter during READ operations and when the Serial EEPROM must acknowledge data transmitted by the master.

START and STOP bits control the bus activity. Operations must begin with a START bit and end with a STOP bit.

A START bit is when SDA transitions LOW while SCL is HIGH while observing the START set-up and hold time specifications.

A STOP bit is when SDA transitions HIGH while SCL is HIGH while observing the STOP set-up and hold time specifications.

Data is recognized as valid while SCL is high. The data on SDA must observe data in set-up and hold specifications before and after SCL is pulsed. There is only one bit of data for each SCL pulse.

Control Byte Requirements

After a START bit, each command begins with an 8 bit control byte sent by the master. This control byte has the following three primary functions before the data and/or word address information is loaded for all commands:

- Identify the Serial EEPROM as the slave addressed on the bus.
- To select the specific Serial EEPROM or the internal memory block on the bus. There may be up to 8 Serial EEPROMs on the bus.
- Select the READ or WRITE function for the next command transmitted by the master.

The diagram of a control byte (not including the START bit) is shown below:

1 0 1 0 A2 A1 A0 X

| I²C Slave Address | Chip or Block Select | Read or Write bit |

Control Bits 1-4 are the Slave Address Bits (Must be 1010 for Memory)

Since there is not a chip select pin, the part is selected by a four bit code in the control byte to identify the type of product. The four bit code was established by Philips for the I²C protocol. A 1010 code identifies the slave device as a Serial EEPROM. The Serial EEPROM will remain in stand-by until the 1010 code is transmitted on the bus. Other non Serial EEPROM slave devices will not respond to the 1010 code on the bus.

Control Bits 5-7 are the 1 of 8 Chip or Block Address Select Bits

The next three control bits are utilized for the chip selection or internal block selection. The standard I²C protocol was developed to allow up to 16k bits of memory to be selected. This could be accomplished by accessing a combination of devices or blocks within a device, as shown in the table on the following page:
Basic Serial EEPROM Operation

<table>
<thead>
<tr>
<th>Device</th>
<th>K bits Density</th>
<th>Internal Blocks</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>Bus Access Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>24LC01B, 24C01,85C72</td>
<td>1</td>
<td>1</td>
<td>H or L</td>
<td>H or L</td>
<td>H or L</td>
<td>Up to 8 devices</td>
</tr>
<tr>
<td>24LC02B, 24C02,85C82</td>
<td>2</td>
<td>1</td>
<td>H or L</td>
<td>H or L</td>
<td>H or L</td>
<td>Up to 8 devices</td>
</tr>
<tr>
<td>24LC04B, 24C04,85C92</td>
<td>4</td>
<td>2</td>
<td>X</td>
<td>H or L</td>
<td>H or L</td>
<td>Up to 4 devices</td>
</tr>
<tr>
<td>24LC08B</td>
<td>8</td>
<td>4</td>
<td>X</td>
<td>X</td>
<td>H or L</td>
<td>Up to 2 devices</td>
</tr>
<tr>
<td>24LC16B</td>
<td>16</td>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Only 1 device</td>
</tr>
</tbody>
</table>

X= NOT USED. This pin must be tied to Vss or VDD. It is not recommended to FLOAT these pins since there may be test modes accessed to these pins via a high voltage signal.

These three bits for this select must match the hardware conditions (if ANY ARE USED) of the external A0, A1, and A2 pins or the internal block selects.

With this selection scheme, devices from 2k to 16k are software compatible. For example, four 2k devices or one 8k device could be connected to the bus with the same software.

The A0, A1, and A2 signals are the same for the 1k and 2k products. The A7 bit for the 1k product is a DON'T CARE.

The A0, A1, and A2 pins are not commonly used today in the industry with the advent of the density evolution up to the I2C protocol limit of 16K bits.

**Control Bit 8 Operation Code**

If this bit is a "1" then the operation will be a READ
If this bit is a "0" then the operation will be a WRITE

After the control byte acknowledge bit is generated by the Serial EEPROM, the master will send the appropriate word address and data information.

**Acknowledge Requirements**

The Serial EEPROM must generate an acknowledge bit after receiving each byte segment in a command. The Serial EEPROM will generate the acknowledge bit automatically after the master has transmitted all of the data for the segment.

To acknowledge the master, the Serial EEPROM must pull the SDA line LOW during the entire HIGH period of the next clock generated by the master. During the READ operations, the master must acknowledge each data byte or the Serial EEPROM will abort the READ operation and return to a stand-by mode waiting for the next START bit.

The attached 24LC16 timing diagrams illustrate the READ and WRITE operations.

**MICROCHIP 2 WIRE DEFAULT CONDITIONS**

As stated before, data sheets do not provide adequate information on basic operation. This lack of information forces each reader of the databook to make interpretations of the operating conditions. These readers have included other semiconductor circuit designers, which unfortunately leads to subtle compatibility problems. The part is designed to operate to the default of the circuit designers interpretation. This next section details Microchip's default conditions to help the system engineer to minimize "Trial and Error" prototyping and to increase the awareness of these default conditions.

Also, to improve corporate wide compatibility, Microchip is standardizing their circuits on various product versions. Unless indicated otherwise, all references to default conditions are for the 24LCXX products, not the 24CXXXX products.

**Power Up**

READ, WRITE, and ERASE operations are valid 5 us after VDD has ramped to the specified operating range.

**PAGE WRITE by Product for Multiple BYTE WRITE Operation**

The 24C01 and 24C02 have a 2 byte buffer.
The 24C04 has an 8 byte buffer.

The 24LC01 and 24LC02 have an 8 byte page.
The 24LC04, 24LC08, and 24LC16 have a 16 byte page.

The buffer will load bytes identically as the page loads bytes. The difference in the two modes is that the buffer will execute a WRITE of one byte per WRITE cycle in sequence. The page mode will execute all bytes loaded in one WRITE cycle in parallel.
Basic Serial EEPROM Operation

There are pages within blocks. For a 16 byte page product, the most significant 4 bits of the word address point to the page address and the least significant 4 bits point to the byte address within a page. For an 8 byte page product, the most significant 5 bits of the word address point to the page address and the least significant 3 bits point to the byte address within a page.

The number of bytes loaded in to the page is from one byte up to the page size. For example, three bytes can be loaded into the 16 byte page of the 24LC16. If during the loading of the fourth byte a STOP bit is received, the page will WRITE three bytes. The fourth byte will not be written since loading the fourth byte was not complete.

NOTE: New versions released in March 1993 will default to ABORTING the entire operation if a STOP bit is received in the middle of a byte while loading a page.

If more than 16 bytes are loaded in the page of a 16 byte page product, then the 17th byte will override the data loaded into the original first byte (the page data will wrap around within a page). Therefore, the system designer must take precautions to not WRITE over a page boundary during a multiple byte WRITE operation.

Bytes not changed in the page will NOT result in data corruption in the array. For example, if two bytes are loaded in to the 24LC16 page with the least significant word address bits of 0000 and then a STOP bit is transmitted. Bytes 1 and 2 in the array will have the data changed to the new page contents. Bytes 3 through 16 WILL NOT change.

The WRITE operation will not be executed until a STOP bit is transmitted.

At this point, the Serial EEPROM is free from the bus since the actual WRITE function is self-timed. Therefore, the microcontroller interfacing to the Serial EEPROM may perform other functions not associated to communication with the Serial EEPROM during the self-timed WRITE operations.

Once the part is in the auto-ERASE mode, it will complete the ERASE/WRITE operation unless power is removed. STOP and START bits will be ignored.

READ

Once the Serial is in a RANDOM READ operation, it can be placed into the sequential READ operation. If the master issues an acknowledge bit instead of a STOP bit, the Serial will READ the next sequential 8 bits. The Serial will wait for the next bit command from the master. The sequential READ will continue as long as the master issues an acknowledge bit on the next clock cycle after the last bit is READ. The READ will continue from block to block and will wrap around if the last bit in the array is addressed. Again, this will continue until the master issues a STOP bit instead of an acknowledge bit.

While reading zeroes the master cannot pull SDA high to generate a STOP bit, since the Serial EEPROM SDA pin is outputting a low. To recover from a fault during a READ, repeat 9 clocks with data floating high. Therefore, the acknowledge bit will not occur and the part will reset and return to stand-by.

A START bit during an operation will cease the current operation and begin the next operation.

AUTHOR: Steve Drehobl
Memory Products Division
3 WIRE BUS EXAMPLE
MICROCHIP 93LC66 READ CYCLE TIMING DIAGRAM

NOTE: THIS EXAMPLE IS FOR X8 OPERATION OF MICROCHIP'S 93LC66.
INSTRUCTION LENGTHS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS.

Data must conform to specified set-up and hold times (Tdis and Tdih) relative to the RISING clock edge. Each parameter is typically 100ns.

A Read operation is identified by the "1 0" op-code following the start bit.

Next, the address location bits are loaded.

Then the address contents are clocked out on the rising clock pulse edge. Data will become valid on the DOUT pin per the specified Tpd time (typically 400ns) relative to the rising edge of the clock on the DOUT pin. Note, the first bit output will be a "dummy bit" with a logical zero state. This event is triggered by the clock rising edge of the last address bit for a duration of one clock pulse.

If the data from the current address is complete and the clock pulses continue, the data from the next address will be READ automatically as long as CS remains high. This is the SEQUENTIAL READ FUNCTION. READ operations will continue while clock pulses continue or until CS is brought low.

It is possible to tie the DOUT pin and the DI pin together to save on I/O requirements from the microcontroller. Caution must be exercised to avoid bus contention for an A0 high condition, because of the dummy bit. It recommended that a resistor between the microcontroller port connected to the DI pin and DOUT pin be added for isolation. This example is shown below:

To the microcontroller

DI

DOUT

Serial EEPROM
3 WIRE BUS EXAMPLE
MICROCHIP'S 93LC66 WRITE CYCLE TIMING DIAGRAM

CLK | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 | #11 | #12 | #13 | #14 | #15 | #16 | #17 | #18 | #19 | #20 | #21 | #22
---|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----

CS to clock rising edge set-up time (Tcss) > 50ns min

Data SET-UP to clock rising edge time (Tds) > 100ns min

DI | 1 | 0 | #6 | #7 | #8 | #9 | #A | #B | #C | #D | #E | #F | #G | #H | #I | #J | #K | #L | #M | #N | #O | #P | #Q
---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----

START BIT

Data HOLD to clock rising edge time (Tdhi) > 100ns min

DOUT | TRI STATE | TRI STATE
---|----------|----------

NOTE: THIS EXAMPLE IS FOR THE X8 OPERATION OF MICROCHIP'S 93LC66.
INSTRUCTION LENGTHS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS.

Data must conform to specific set-up and hold times (Tdis and Tdih) relative to the clock edge. Each parameter is typically 100ns.

A WRITE operation is identified by a the "0 1 " op code following the start bit

Next, the address location bits are loaded on the DI pin.

Then, the data bits to be written are loaded on the DI pin.

CS must be brought low after the last DI bit is loaded. When CS is brought low for the Tcs1 period, a self timed WRITE is executed. If too many bits are loaded during ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

The DOUT pins only function during a WRITE is to indicate the status of the write with the READY/BUSY function. While DOUT is low, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When DOUT is high, the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the Tcs1 time is complete to initiate the READY/ BUSY function. Microchip's 93LCXX products can be polled multiple times for the same cycle.

Up through clock pulse 20, data for the instruction is being LOADED. When the CS goes low, the instruction is being EXECUTED. If there are not enough bits loaded during ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the Serial EEPROM will return to stand-by.
3 WIRE BUS EXAMPLE

MICROCHIP 93LC66 ERASE CYCLE TIMING DIAGRAM

NOTE: This example is for X8 operation of the Microchip's 93LC66. Instruction lengths may vary with array size and data widths.

Data must conform to specified set-up and hold times (Tdis and Tdih) relative to the clock edge. Each parameter is typically 100ns.

AN ERASE operation is identified by a "11" two bit code that follows the start bit

Next, the address location bits are loaded on the DI pin.

THERE ARE NO DATA BITS TO LOAD. THE ADDRESS LOCATION LOADED WILL BE SET TO AN ERASE STATE OF "1".

CS must be brought low after the last DI bit is loaded. When CS is brought low for the Tcsl period, a self timed ERASE is executed. If too many bits are loaded during the ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

The DOUT pin's only function during a ERASE is to indicate the status of the write with the READY/BUSY function. While DOUT is low, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When Do is high, the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the Tcsl time is complete to initiate the READY/BUSY function.

Up through clock pulse 12, the address for the instruction is being LOADED. When CS goes low, the instruction is being EXECUTED. If there are not enough bits loaded during the ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the serial EEPROM will return to stand-by.
2 WIRE BUS EXAMPLE
MICROCHIP 24LC16 BYTE WRITE CYCLE TIMING DIAGRAM

NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE.

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL DATA SET-UP TIME OF 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (THD: DAT) FOR THE FALLING CLOCK EDGE.

THE STOP BIT ON CLOCK PULSE #29 WILL INITIATE A SELF TIMED WRITE. THE SERIAL EEPROM CAN NOT EXECUTE ADDITIONAL INSTRUCTIONS UNTIL THE CYCLE IS COMPLETE.

TO EXECUTE A PAGE WRITE, CONTINUE TO LOAD 8 BITS OF DATA AT CYCLE 29 INSTEAD OF ISSUING A STOP BIT (FROM THE MASTER). REMEMBER, A CLOCK PULSE MUST BE ALLOCATED AFTER EACH SUBSEQUENT 8 BITS FOR THE SERIAL EEPROM TO ISSUE AN ACKNOWLEDGE SIGNAL (LOW). AFTER THE DESIRED NUMBER OF BYTES HAVE BEEN LOADED, UPTO THEIR PAGE SIZE, THE MASTER MUST ISSUE A STOP BIT TO EXECUTE THE INSTRUCTION.

THE SEQUENCE OF EACH WRITE COMMAND AND THE MASTER/SERIAL DIRECTION OF THE COMMUNICATION IS SHOWN BELOW:

BYTE WRITE
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
STOP BIT FROM THE MASTER

CONTROL BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
STOP BIT FROM THE MASTER
2 WIRE BUS EXAMPLE

MICROCHIP 24LC16 READ CYCLE TIMING DIAGRAM

START BIT SLAVE ADDRESS BLOCK SELECT WRITE START BIT SLAVE ADDRESS BLOCK SELECT READ DATA FROM THE SERIAL STOP BIT

NOTE: THE FIRST 19 CLOCK PULSES OF THE WRITE COMMAND ARE IDENTICAL TO THE FIRST 19 CLOCK PULSES IN THE READ COMMAND. EVEN THE 9TH CLOCK PULSE IS A WRITE BIT '0' TO TRANSMIT TO THE SERIAL EEPROM THE DESIRED WORD ADDRESS.

THE READ COMMAND HAS TWO START BITS. THIS IS FOR THE RANDOM READ COMMAND.

AS SHOWN ON THE PREVIOUS PAGES, IF A READ IS DESIRED FROM A CURRENT ADDRESS THEN THE FIRST 19 CLOCK PULSES ARE NOT REQUIRED. THEREFORE, THE FIRST START BIT IS AT CLOCK PULSE #20. THIS IS ONLY FOR THE CURRENT ADDRESS READ COMMAND.

ANOTHER USEFUL READ COMMAND IS THE SEQUENTIAL READ COMMAND. THE SEQUENTIAL READ COMMAND IS THE SAME AS THE RANDOM READ COMMAND; HOWEVER, THE MASTER MUST ISSUE AN ACKNOWLEDGE BIT INSTEAD OF THE STOP BIT AS SHOWN FOR CLOCK PULSE #38. THIS SIGNALS THE SERIAL TO READ THE DATA FROM THE NEXT SEQUENTIAL ADDRESS. THE MASTER MUST CONTINUE TO ACKNOWLEDGE EACH BYTE RECEIVED UNTIL THE MASTER ISSUES A STOP BIT.

NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE.

READ (FROM A RANDOM ADDRESS)
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (FROM THE CURRENT ADDRESS)
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (SEQUENTIAL READ of 3 bytes)
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK I2C PROTOCOL DATA SET-UP TIME OF 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (THD: DAT) FOR THE FALLING CLOCK EDGE.
The term "endurance" has become a confusing parameter for both users and manufacturers of EEPROM products. This is largely because many semiconductor vendors treat this important application-dependent reliability parameter as a vague specmanship topic. As a result, the system engineer often designs without proper reliability information or underutilizes the EEPROM as an effective solution.

Endurance (the number of times an EEPROM cell can be erased and rewritten without corrupting data) is a measure of the device's reliability, not its parametric performance. As such, endurance is not achieved by somehow making EEPROM devices more durable or robust to extend the life of the intrinsic erase/write cycle, but rather by reducing their defect-density failure rates. This has a direct impact on the design engineer characterizing EEPROM memory needs for an application and evaluating components from various manufacturers. The system design engineer needs to understand not only the relationship between the application, expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

This tutorial volume is intended to clarify some of the issues in the industry and provide a tool for the system design engineer, the system reliability engineer, and the component engineer to determine EEPROM reliability and understanding how to apply it to actual application requirements. It will examine four main areas:

- CMOS floating gate memory cell operation and characteristics
- Significant process and design interactions and endurance characterization variables
- Common mis-interpretations of endurance
- Determining some real world application reliability requirements

**EEPROM MEMORY CELL OPERATION AND CHARACTERISTICS**

In discussing endurance characteristics of EEPROMs, it's important to review how these components operate, and why and how they fail. Figure 1 illustrates a CMOS floating gate EEPROM cell, including voltage conditions for READ, ERASE, and WRITE operations. To erase or write, the row select transistor must have the relatively high potential of 20V. This voltage is internally generated on chip by a charge pump, with the only external voltage required being Vdd. The only difference between an ERASE and a WRITE is the direction of the applied field potential relative to the polysilicon floating gate.

When 20V is applied to the polysilicon memory cell gate and 0V is applied to the bit line drain (column), electrons tunnel from the substrate through the 90-angstrom Tunnel Dielectric (TD) oxide to the polysilicon floating gate until the polysilicon floating gate is saturated with charge. The cell is now at an ERASE state of "1". When 0V is applied to the polysilicon memory cell gate and 20V is applied to the bit line drain (column), electrons tunnel from the polysilicon floating gate through the TD oxide to the substrate. The cell then is at a WRITE state of "0". This sequence of the transfer of charge onto the floating gate (ERASE) and the electrical removal of that charge from the floating gate (WRITE) is one ERASE/ WRITE cycle, or "E/W cycle."

The field (applied voltage to an oxide thickness) across the tunneling path created by the 20V potential is extremely high in order to transfer the electrons. Over the cell's "application time," as measured by E/W cycles, the EEPROM cell begins to wear out due to the field stress. The EEPROM cell wears out as the number of cycles increase resulting in the voltage margin between the ERASE and WRITE states decreasing until finally there is not enough margin for the EEPROM sense amp to detect a difference in the two states during a READ. Failure is defined as when the sense amp can no longer reliably differentiate logic state changes.

Figure 2 (single cell EEPROM endurance characteristics) illustrates that the intrinsic wear out point for a normal cell with specified dimensions and electrical characteristics is very acceptable, in excess of 2 million E/W cycles. Failures at lower cycles are due mostly to very small defects or imperfections in the oxide or silicon-to-oxide interface.
A key point to remember is that most failures occurring at less than 2 million E/W cycles are due to the number of defects per a given area (defect density dependent.) Thus high EEPROM endurance reliability is achieved by reducing the number of intrinsic cycles in the cell's operational design.

Error correction circuits are design techniques commonly used by EEPROM manufacturers to increase endurance by reducing the failure rate caused by single bit failures. These circuits are transparent to the user. One typical scheme is using 4 bits of error correction for every 8 “real” bits (one byte). In this scheme, one bit failure in the byte is correctable, while if two bits within the byte fail, the byte is not correctable.

Another error correction scheme is to use one “parity” bit for every “cell.” Here both EEPROM cells must fail to result in a bit fail.

**FIGURE 1 - CMOS FLOATING GATE EEPROM CELL**

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Read</th>
<th>Write</th>
<th>Erase</th>
<th>Standby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Line</td>
<td>1.6 volts</td>
<td>18.0 volts</td>
<td>0.0 volts</td>
<td>0.0 volts</td>
</tr>
<tr>
<td>Row Select Gate</td>
<td>5.0 volts</td>
<td>20.0 volts</td>
<td>20.0 volts</td>
<td>0.0 volts</td>
</tr>
<tr>
<td>Memory Cell Gate</td>
<td>5.0 volts</td>
<td>0.0 volts</td>
<td>20.0 volts</td>
<td>0.0 volts</td>
</tr>
<tr>
<td>Common Source</td>
<td>0.0 volts</td>
<td>Float</td>
<td>0.0 volts</td>
<td>0.0 volts</td>
</tr>
</tbody>
</table>
Figure 2 - Single EEPROM Cell Endurance Characteristics

Process and Design Variables Affecting Endurance

There are many subtle process and design variables that have a strong impact on endurance. These interacting variables will play very different roles depending on the different process technologies of various semiconductor manufacturers.

The primary interaction is the amount of time at the high voltages that is ultimately applied to the cell. A finite amount of time at finite voltages are required to achieve "optimal" erase and write thresholds. If the time is too short and the voltage is too low, the EEPROM will not program to the proper threshold. Also, if the programming ramp time is too fast and the voltage is too high, the EEPROM’s endurance will be reduced. Unfortunately, there is most often a trade-off between fast reliable programming performance or high endurance reliability. Some of the significant process and design variables are shown below and their impact on programming performance and endurance performance.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Programming</th>
<th>Endurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal High Voltages</td>
<td>HIGHER = Faster Programming</td>
<td>LOWER = Increased Endurance</td>
</tr>
<tr>
<td>Internal High Voltage Ramp Rate</td>
<td>FASTER = Faster Programming</td>
<td>SLOWER = Increased Endurance</td>
</tr>
<tr>
<td>Programming Time</td>
<td>LONGER = Improved Voltage Margin on the Cell</td>
<td>SHORTER = Less Oxide Stress for Increased Reliability</td>
</tr>
<tr>
<td>TD Oxide thickness</td>
<td>THICKER = Slower Programming</td>
<td>THINNER = Reduced Endurance</td>
</tr>
<tr>
<td>Temperature</td>
<td>LOWER = Faster Programming</td>
<td>LOWER = Increased Endurance</td>
</tr>
</tbody>
</table>
COMMON MISINTERPRETATIONS

In examining industry EEPROM literature on the topic of endurance, it’s easy to misunderstand or misinterpret endurance concepts due to incomplete databook statements. The following are clarifications to some of the common misinterpretations:

**Endurance and Read Cycles**

READ operations are unlimited since they impose virtually no stress on the cell. Endurance data apply only to E/W cycles.

**Erase/Write Ratings**

E/W ratings are based on each byte in the application, not on the number of op codes or control byte commands utilized. For example, if a part is rated to 100K E/W cycles, then each individual byte can be erased and written 100K times. The part is NOT limited to only a total of 100K E/W op-codes or control bytes. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement.

**Cycles/Day**

In many cases, a serial EEPROM is used for widely varying functions in an application. These functions have different E/W usage requirements (cycles/day), resulting in different endurance requirements and, usually, different reliability results for each function.

For example, assume a given end-product application will have a 10-year life. For each function within that application, an assumption must be made for the expected E/W cycles per day for a given segment of bytes. If a function has a segment of bytes cycled 1 time per day, then this segment of bytes will have 3,650 cycles in its lifetime (365 days per year for 10 years at 1 cycle per day and 7 days per week operation). Any given segment of bytes would have to cycle 274 times per day everyday to reach 1,000,000 E/W cycles in its 10-year application lifetime. Such a frequency is, of course, very rare in actual applications.

For reference purposes, Figure 3 indicates typical cycles per day for some common applications. Although many manufacturers routinely discuss very high numbers of E/W cycles, the amount of applications actually utilizing 1 million cycles is very small.

A further and very important incorrect assumption often made is that ALL bits in an application need the same number of cycles and endurance ratings. In most applications, however, functions that require a high

![Figure 3 - Typical Serial EEPROM E/W Cycles/Day by Application](image-url)
number of E/W cycles per day require only a small number of bits. Last number redial in a telephone, for example, consumes many E/W cycles per day, but utilizes only a few bytes for this function. By contrast, speed dial storage in that same telephone consumes only a fraction of E/W cycles per day, but requires a relatively large segment of bits to accommodate the many speed dial options. In such an application, the same serial EEPROM normally performs both functions at different address locations.

ENDURANCE DATA FROM THE CUSTOMER’S PERSPECTIVE

Unfortunately, an industry standard for an endurance test method has yet to be adopted. Since endurance data is not baselined, the process of evaluating endurance becomes that much more complicated for the system designer and reliability engineer.

It is not uncommon for customers to request endurance data from many semiconductor vendors. All vendors would be expected to comment that they experience a low failure rate through 100K E/W cycles. While this can be a true statement, it can also be a very incomplete statement. It is extremely doubtful that all vendors test their components to the same conditions. Yet the variables within endurance testing are extremely significant. Small differences in text protocol can have enormous differences. Pattern, cycling mode, temperature and array size, for example, are the most significant testing variables.

First, memory cell failure rates are defect density driven up to the intrinsic wear out point. Existing defects in a cell, while not causing failure initially, are stressed during every transfer of electrons through the TD oxide until they eventually cause cell failure. Worst case testing would be to erase and write each bit, which is what a write all "0"s pattern with an auto-erase of "1" routine will perform. Indeed, this write all "0"s test pattern will produce very different results than a checkerboard test pattern of alternating "1"s and "0"s within a byte, since cells are changed more often writing all "0"s than in an alternating "1" and "0" write pattern. The resultant failure rate differences are indicated on the pattern effect graph in Figure 4.

In actual use, however, a system will experience a random pattern much more like the alternating "1"s and "0"s pattern than the more stressful all "0"s pattern. The key point for system designers is to determine how accurate a test routine has been used to determine a particular manufacturer's endurance data, and make the appropriate judgement on that part's expected endurance in the application.

FIGURE 4 - PATTERN EFFECT ON ENDURANCE TESTING

© 1992 Microchip Technology Inc.
Serial EEPROM Endurance

Second, the cycling mode graph in Figure 5 indicates that significantly different results can be achieved in endurance testing using a block cycle mode than using a byte cycle mode. The block mode is commonly used by manufacturers to "speed up" the endurance characterization process. However, endurance results usually will appear much better for the block mode than the byte mode, due to the high voltage variables discussed earlier. The reason is that the voltage ramp rate is significantly SLOWER, the high voltages are slightly lower, thus less stressful for block cycling since the capacitive load of the entire array is on the high voltage charge pump. The capacitive load is much lower with a single row or byte, which thus has a significantly faster ramp rate. Also, there is not a polysilicon to polysilicon stress for adjacent cells since all cells are at the same potential. Most often, these factors combine to yield lower failure rates for block cycling than for byte cycling. Again, the test conditions must match the system conditions.

Finally, increasing temperature also increases stress on the cell. Microchip's endurance characterization data indicates that increasing temperature adds an activation energy (Ea) of 0.12eV on the cell. From a 25°C to 85°C ambient the acceleration factor is approximately 2.1. Therefore, the higher the temperature, the higher the stress. These results will vary significantly with each EEPROM manufacturer.

RECOMMENDED EEPROM ENDURANCE TESTING

Microchip believes that EEPROM components should be endurance tested to reflect system conditions. Therefore, units are cycled to an alternating ONE and ZERO pattern (checkerboard), then to an alternating ZERO and ONE pattern (inverse checkerboard).

Again, since endurance characterization data indicates that a random single bit fail is the primary first order failure mode, endurance is defect density (def/cm2 or segment of bit size) dependent. Therefore an expected failure rate range by density can be established.

DETERMINING THE RELIABILITY CALCULATIONS

There are three primary components for the system design engineers to use in determining the endurance reliability required for a defect density limited application. These three components are:

- Erase/write cycles/day estimated for the function.
- The number of bits in the function (or segment size).
- Case operating temperature of the Serial EEPROM.

Let's look at three typical examples utilizing the above information to predict a cumulative failure rate at different points in a system lifetime. Please note that Industry endurance perceptions have improved from a very high (>2%) failure rate expectation to a very low actual PPM level failure rate in the past few years.

FIGURE 5 - CYCLING MODE EFFECT ON ENDURANCE TESTING
EXAMPLE #1 - AUTOMOBILE ELECTRONIC COMPASS

A 16-byte (128-bits) segment from a 2K array stores data every time the power is turned off in an automobile electronic compass. The design engineers expect the system to power down an average of 5 times/day over a 10-year life in an 85°C case temperature environment. The projected Microchip cumulative failure rate under these conditions at the 10 year point is less than 8 PPM or 0.0008% failures in 10 years. The cumulative PPM failure rate graph is referenced on Figure 6.

EXAMPLE #2 FULL-FEATURED TELEPHONE

An 8-byte (64-bit) segment is used to store the last number redial on a stationary full feature phone operating in a room temperature environment. A 12K bit (12,288-bits) segment is used for storage of speed dial numbers on the same phone. This application has two major functions and therefore it will have two separate failure rate calculations.

The last number redial function is expected to be utilized an average of 20 times/day over a 10 year life. Each speed dial is expected to be updated an average of 0.1 times/day over a 10 year life.

Figure 7, the cumulative failure rate graph for these two conditions indicates an extremely low failure rate in the projected 10 year lifetime. The failure rate begins beyond 20 years as shown on the attached cumulative PPM failure rate graph.

EXAMPLE #3 - LASER PRINTER

A serial EEPROM could have many functions in a laser printer. A function that would likely require the most cycles/day is the maintenance log storage of the pages printed (estimated to be 100 cycles/day). Three bytes are utilized to store this number. The case temperature environment is estimated to be between 55°C and 85°C.

The high number of cycles at an extreme temperature of 85°C indicate a failure rate of less than 1200 PPM thru the first 5 years and 2600 PPM thru the first 10 years.

This failure rate can be dramatically reduced if the operating temperature is reduced to 55°C. The same 5 and 10 year PPM levels are reduced to 450 PPM and 1600 PPM at 55°C.

These failure rates are illustrated on Figure 8.

SUMMARY

Microchip Technology Inc. has recognized that increasing reliability in serial EEPROMs through increased endurance is not a function of extending the life of the intrinsic erase/write cycle, but depends on reducing defect-density failure rates. Design engineers characterizing EEPROM memory needs for an application and evaluating EEPROM components from various manufacturers need to understand not only the relationship between the application and expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

FIGURE 6 - EXAMPLE #1: THE AUTOMOBILE COMPASS

<table>
<thead>
<tr>
<th>24LC02, 85C, 5V, 16bytes, 5cycles/day, RANDOM, BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>PPM = 82</td>
</tr>
<tr>
<td>0.000</td>
</tr>
<tr>
<td>18,250 Cycles, 10.0 Yrs</td>
</tr>
</tbody>
</table>
Microchip has developed a program to calculate the cumulative PPM failure rates for specific system conditions, as shown on the previous pages. This program is being transferred onto menu driven DOS floppy disks. These disks will be available in December 1992 to the system designers, reliability engineers, and component engineers to project endurance failure rates for specific system conditions. These disks will be periodically updated to reflect Microchip's most recent endurance data.

FIGURE 7 - EXAMPLE #2: THE FULL FEATURED PHONE

<table>
<thead>
<tr>
<th>PPM</th>
<th>250</th>
<th>300</th>
<th>350</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.77E-05</td>
<td>8.91E-05</td>
<td>1.20E-04</td>
<td>1.50E-04</td>
<td>1.51E-04</td>
</tr>
<tr>
<td>1.00E-05</td>
<td>1.64E-04</td>
<td>2.42E-04</td>
<td>3.02E-04</td>
<td>3.33E-04</td>
</tr>
</tbody>
</table>

24LC16, 22C, 5V, K Cycles, 10 Yrs.

FIGURE 8 - EXAMPLE #3: THE LASER PRINTER

Laser Printer Cumulative 365k Cycles 10 Yrs.

- 55C
- 85C

AUTHOR: Steve Drehobl
Memory Products Division
INTRODUCTION

Endurance, as it applies to non-volatile memory, refers to the number of times an individual memory cell can be erased and or written (some architectures do not erase before every write). Advances in process technology have made it possible to increase these limits and for Microchip Technology Inc. to offer new concepts - Total Endurance™ and a split architectural design for variable endurance. These concepts lead to more reliable products with more bits per dice, such as the 24C32 and 24C65.

TOTAL ENDURANCE™

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate (for each byte in the application not on the number of op codes or control byte commands) and is never based on any read functions whether they be a data read or configuration read. If a part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. These studies led to the concept of using the computer to predict the theoretical wear out of the floating gate and ultimately to project the point in time of a products life cycle when the first non-volatile memories bit or periphery failure should occur. After many man years of data collecting, predicting and verifying the results, Microchip feels confident in publishing and offering for the general technical community this predictive model in the form of IBM® PC-compatible software. Microchip has a patent pending on this predictive mathematical model.

TOTAL ENDURANCE PREDICTIVE SOFTWARE

The predictive software described here originally was being developed as a tool for determining endurance levels of Microchip non-volatile devices. Upon seeing the potential as a design aid, it was decided to develop a software tool that could be purchased by the engineering community and used as to predict non-volatile endurance parameters for architectural and operating parameter trade-offs before the design was completed and without subjecting the devices to prolonged incoming test for endurance levels. It should be noted that this predictive model applies only to Microchip Technology Inc. non-volatile devices.

The program uses a iterative statistical model developed by Microchip Technology Inc. physicists. The model was first used in a DOS-based text program as a proof of concept and for developing the exhaustive database needed for such a tool (included on the program disk as enddos.exe). This model was then imported to a Windows™-based software package with full GUI capabilities and all the normal cut, paste, print, viewing properties. The model actually operates as a mathematical function which is called from within the Windows Visual Basic shell and is passed all of the pertinent operational, process, and device information. The model then, after calculating the essential data points, returns this information to the main program to formatted and displayed both textually and graphically.

Applying the predictive data to the high endurance block of the 24C65, using the 24LC04 which has similar characteristics, and assuming the following:

• a five-year life
• an expected E/W cycles of 10 times per day
• a function of 11 bytes
Using Endurance Predictive Software

Operational specifications:

<table>
<thead>
<tr>
<th>Device</th>
<th>24C65 (24LC04B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>5</td>
</tr>
<tr>
<td>Temperature</td>
<td>25°C</td>
</tr>
<tr>
<td>Bytes/Cycle</td>
<td>11</td>
</tr>
<tr>
<td>E/W/Day</td>
<td>10</td>
</tr>
<tr>
<td>App. Life (Yr.)</td>
<td>5</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>BYTE</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>RANDOM</td>
</tr>
</tbody>
</table>

The 4K HE block with 1 M E/W cycles typical, in this application, should yield the following results:

- FIT: 1.0 PPM
- Time: 5.0 Write cycles
- 18,250 cycles

The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance, see the related application notes AN536 and AN537 contained elsewhere in this volume. All operation parameters, along with the process technology, effect the effective endurance of a non-volatile device. The voltage, temperature, cycles per, bytes per cycle, and even the number of times written per day (time between write cycles) all have an effect on the oxide breakdown or periphery failure rate of a particular non-volatile process.

Endurance is not a well-defined concept within the semiconductor industry. The number of erase/write cycles which a particular EEPROM can endure is dependent not only upon the design of the device but also upon the application environment in which it is used. Therefore, blanket claims such as “1 million erase/write cycles typical” can only be validated based upon the specific parameters of each application. Yet until now, there has been no tool available for predicting the endurance of a particular EEPROM device within a set of application parameters. Trade-off analysis can be painfully time-consuming and only marginally accurate without specific knowledge of the behavior of the device under different conditions of use.

The Microchip Total Endurance Software allows the designer to trade off voltage, temperature, write cycles, number of bytes written, number of writes per day, PPM and FIT rates, and years of use in order to optimize the system and accurately predict product lifetime and reliability.

The following is an example using the Endurance Software to aid in the design of an electronic phone book/auto-dialer:

The auto-dialer may have new numbers added or changed several times per day; but how can the manufacturer specify the life of the unit, and at what rate of update of the phone numbers? First, the designer must make some assumptions. If we assume that the average user will change or add 50 phone numbers per day, and the manufacturer is willing to live with a 0.1% failure rate (1,000 PPM) after 10 years of use, then we have almost enough information to verify whether we are in the ballpark given the physics of the EEPROM device which will store the numbers. We also need to know the operating voltage and temperature of the application; we will say that a 3.3V lithium button battery is powering the unit and the temperature range is limited to that for which the LCD display will function: 0°C to 70°C. End-of-life voltage for the battery is approximately 2.0V; assuming that the ASIC or microcontroller in the application will operate down to 2.5V, the EEPROM also has a 2.5V requirement. The designer would like to be able to store 100 phone numbers of 16 bytes each, which results in a 1.6K byte requirement for the Serial EEPROM. Because 1.6K bytes is equal to 12.8K bits, a 16K bit 2-wire Serial EEPROM will more than suffice. Specifically, Microchip’s 24LC16B will operate down to 2.5V and even includes a write-protect feature which can be used to block inadvertent writes in a noisy environment.

Here is a summary of the application:

- Device: 24LC16B
- Voltage: 2.5V - 3.3V
- Temperature: 0°C to 70°C (55°C typical)
- Cycles per day: 50
- Bytes per cycle: 16
- Application life: 10 years

The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance, see the related application notes AN536 and AN537 contained elsewhere in this volume.
Once these values are entered into the Total Endurance program, it outputs the following:

### Device Data: Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>24LC16B</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3</td>
</tr>
<tr>
<td>Temperature</td>
<td>55</td>
</tr>
<tr>
<td>Bytes/Cycle</td>
<td>16</td>
</tr>
<tr>
<td>E/W</td>
<td>50</td>
</tr>
<tr>
<td>App. Life (Yrs)</td>
<td>10</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>BYTE</td>
</tr>
<tr>
<td>Pulse Width (Ms)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>RANDOM</td>
</tr>
</tbody>
</table>

### Device Data: Output Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT</td>
<td>21.0</td>
</tr>
<tr>
<td>PPM</td>
<td>1,842</td>
</tr>
<tr>
<td>Time</td>
<td>10.0</td>
</tr>
<tr>
<td>Write cycles</td>
<td>182,500</td>
</tr>
</tbody>
</table>

Both of the lists above were copied directly from the Total Endurance program output to the Microsoft® Windows clipboard and pasted into this document (the Total Endurance program has a handy menu click to make this easy).

Unfortunately for our designer, the desired 0.1% failure rate has almost doubled to 0.18% (1842 PPM). But fortunately for the designer, the Total Endurance program makes trade off analysis very simple and fast. At this point there are at least three options: (1) live with almost 2000 PPM, or (2) look at the endurance plot and check whether there is a reasonable number of E/W cycles which will provide a 1000 PPM failure rate, or (3) specify a PPM rate to the Total Endurance program and let it crank out the number of cycles it will take.

Below is the endurance plot, again pasted directly from the Total Endurance program:

You can see that by reducing the number of cycles from the 182,500 which resulted from our first trial to about 100,000, we can achieve a PPM rate of about 1000 (0.1%). But how does 100,000 cycles translate into application life or cycles per day?

By switching the Total Endurance program mode to a PPM request mode instead of application life mode, we can query the program for this information. Let's ask it for the application life of the product given a 1000 PPM failure rate. Here are the results:

### Device Data: Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>24LC16B</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3</td>
</tr>
<tr>
<td>Temperature</td>
<td>55</td>
</tr>
<tr>
<td>Bytes/Cycle</td>
<td>16</td>
</tr>
<tr>
<td>E/W</td>
<td>50</td>
</tr>
<tr>
<td>PPM Level (Yrs)</td>
<td>1000</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>BYTE</td>
</tr>
<tr>
<td>Pulse Width (Ms)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>RANDOM</td>
</tr>
</tbody>
</table>

### Device Data: Output Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPM</td>
<td>1,000</td>
</tr>
<tr>
<td>Time</td>
<td>5.97</td>
</tr>
<tr>
<td>Write cycles</td>
<td>109,000</td>
</tr>
</tbody>
</table>

Now we have some more options: (1) specify the product life at 5 years or (2) trade off other parameters of the application such as voltage or temperature, or (3) decide which is more important - a 10-year product lifetime, or the ability to change 50 numbers every single day. Maybe this analysis has caused our designer to re-evaluate the 50 cycle-per-day requirement. Will the user really change or add that many numbers per day - half of the unit's total capacity? Maybe 20 or even 10 is a more practical figure. Realistically, a user may enter or change quite a few numbers the first week or two of the application, and after that the unit will be used mostly for reading and dialing numbers.

Changing the number of erase/write cycles to 20 per day gives us the following results:

### Device Data: Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>24LC16B</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3</td>
</tr>
<tr>
<td>Temperature</td>
<td>55</td>
</tr>
<tr>
<td>Bytes/Cycle</td>
<td>16</td>
</tr>
<tr>
<td>E/W</td>
<td>20</td>
</tr>
<tr>
<td>PPM Level (Yrs)</td>
<td>1000</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>BYTE</td>
</tr>
<tr>
<td>Pulse Width (Ms)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>RANDOM</td>
</tr>
</tbody>
</table>
Using Endurance Predictive Software

<table>
<thead>
<tr>
<th>Device Data:</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPM</td>
<td>1,000</td>
</tr>
<tr>
<td>Time</td>
<td>14.93</td>
</tr>
<tr>
<td>Write cycles</td>
<td>109,000</td>
</tr>
</tbody>
</table>

Wow! Reducing the number of cycles per day not only brought us back to a 10-year life, it gave us some margin on that, too. Keeping all the other parameters the same and forcing a 10-year lifetime gives us the following final results:

<table>
<thead>
<tr>
<th>Device Data:</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT</td>
<td>7.1</td>
</tr>
<tr>
<td>PPM</td>
<td>625</td>
</tr>
<tr>
<td>Time</td>
<td>10.0</td>
</tr>
<tr>
<td>Write cycles</td>
<td>73,000</td>
</tr>
</tbody>
</table>

The new PPM rate of 625 gives our triumphant designer more than 30% margin on his PPM target of 1000.

This example shows the significant reduction in time for design trade off analysis and time-to-market which can be achieved with a useful tool like the Microchip Total Endurance Disk. In addition, it demonstrates the increase in robustness of the system design by providing known quantities and readily accessible handles to modify those quantities in the trade-off analysis. This tool can literally reduce weeks of effort into a few minutes of point and click.

AUTHORS: Peter Sorrells
Memory Products Division

Richard J. Fisher
Memory Products Division
INTRODUCTION

The 24LCXXB Serial EEPROMs from Microchip Technology are I²C™ compatible, will operate in the standard 100 kHz and the 400 kHz Fast Mode. When using any serial protocol with a general microcontroller that does not have a dedicated protocol specific serial port, the designer must generate the specific code routines to accomplish the several memory access functions that the microcontroller will perform. The PIC16CXX products from Microchip are both versatile and efficient to program. This application note is a series of stand-alone programs to perform the basic I²C interface functions on a PIC16C54 running at 4 MHz in XT mode. This configuration will provide a 60 kHz serial bus rate. At this speed, the interface does not meet the 100 kHz specification. If a higher clock rate is desired, the HS crystal oscillator must be used, which has a maximum frequency of 20 MHz. NOTE THAT THE TIMING ROUTINES MUST BE REWRITTEN TO RUN AT THESE FASTER CLOCK RATES. The user must consult all applicable data sheets for design details. These programs have been fully tested and are being made available for general use. Figure 1 demonstrates the tested configuration of PIC16C54 and the 24LCXXB device.

This application note includes the following programs.

- 2-Wire Byte Read
- 2-Wire Byte Write
- 2-Wire Byte Write with Data Polling
- 2-Wire Page Write
- 2-Wire Sequential Read

FIGURE 1 - TESTED CONFIGURATION OF THE PIC16C54 AND THE 24LCXXB DEVICE

AUTHOR: Bruce Negley
Memory Products Division
LIST P=16C54

;******************************************************************************
; 2-Wire Byte Read Program (118 bytes)
;******************************************************************************

This program demonstrates how to interface a Microchip PIC16C54 to a 24LCXXB Serial EE device and perform a random read operation. This program will read 8 consecutive addresses in the 'random read' mode. This entails setting the address pointer before doing read command for every address.

Another, more efficient method of reading consecutive addresses is the 'sequential read' mode. This involves sending the control byte and address for the first byte to read, then continuing to provide clocks for the next addresses. The device will automatically increment the address. An example of the sequential read mode is provided in the '2wseqr.asm' file.

Timing is based on using the PIC16C5XX in 'XT' mode using a 4MHz crystal. Clock speeds to the serial EE will be approximately 60KHz for this setup.

As an option, the user may connect a LED to pin 18 on the PIC16C5XX (use about a 1K resistor in series) as an acknowledge fail indicator. This LED will come on if the serial EE fails to acknowledge correctly.

PIC16C5XX to Serial EE Connections:

- PIC16C5XX Serial EE
  - Pin 12 (RB6) -> SCLK
  - Pin 13 (RB7) -> SDATA
  - PIN 18 (RAll) -> Acknowledge fail LED (Optional)

******************************************************************************

Register Definitions
******************************************************************************

port_a equ Sh ; port 5 (port a) used for LED display
port_b equ 6h ; port 6 (port b) used for data and clock lines
eeprom equ Oah ; bit buffer
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address 1010xxxx0
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter

Bit Definitions
******************************************************************************

d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b, pin 13)
sclk equ 6 ; serial EE clock line (port_b, pin 12)
ackf equ 1 ; acknowledge fail LED (port_a)

org 01ffh ; set reset vector
goto PWRUP
org 000h

******************************************************************************

© 1993 Microchip Technology Inc.
Interfacing the 24LCXXB Serial EEPROMs

;*******************************************************
; Start Bit Subroutine
; This routine generates a start bit
; (Low going data line while clock is high)
;*******************************************************

BSTART
  bcf port_b,sdata ; make sure data is high
  movlw b'00111111' ; set data and clock lines for output
  tris port_b
  bcf port_b,sclk ; make sure clock is low
  bcf port_b,sclk ; set clock high
  nop
  nop
  nop
  bcf port_b,sdata ; timing adjustment
  bcf port_b,sclk ; start clock train
  nop
  retlw 0

;*******************************************************
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;*******************************************************

BSTOP
  bcf port_b,sdata ; make sure data line is low
  movlw b'00111111' ; set data/clock lines as outputs
  tris port_b
  bcf port_b,sdata ; make sure data line is low
  nop
  nop
  nop
  bcf port_b,sclk ; set clock high
  nop
  nop
  bcf port_b,sdata ; data goes high while clock high
  ; for stopbit
  nop
  nop
  bcf port_b,sclk ; set clock low again
  nop
  nop
  retlw 0

;*******************************************************
; BITOUT routine takes the bit of data in 'do' and
; transmits it to the serial EE device
;*******************************************************

BITOUT
  movlw b'00111111' ; set data, clock as outputs
  tris port_b
  btfss eeprom,do ; check for state of data bit to xmit
  goto bitlow ; low? go set data line low
  bcf port_b,sdata ; high? set data line high
  goto clkout ; go toggle the clock
Interfacing the 24LCXXB Serial EEPROMs

```assembly
; Bitlow
bcf port_b, sdata ; output a low bit
clokout
bcf port_b, sclk ; set clock line high
nop
nop
nop
bcf port_b, sclk ; return clock line low

; *****************************************************
; BITIN routine reads one bit of data from the serial EE device and stores it in the bit 'di'
; *****************************************************
BITIN
bcf eeprom, di ; assume input bit is high
movlw b'10111111' ; make sdata an input line
tris port_b
bcf port_b, sdata ; set sdata line for input
bcf port_b, sclk ; set clock line high
nop
nop
nop
nop
btfss Jxf
retlw
port_b, sdata
eeprom, di
port_b, sclk

; Assume input bit is high
; Make sdata an input line
; Set sdata line for input
; Set clock line high
; Just sit here a sec
; Read the data bit
; Input bit was low, set 'di' accordingly
; Set clock line low

; *****************************************************
; Transmitter Subroutine
; This routine takes the byte of data stored in the 'datao' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then the device did not send a proper ack bit and the ack
; fail LED will be turned on.
; *****************************************************
TX
movlw .8
movwf count ; set the #bits to 8

TXLP
bcf eeprom, do ; assume bit out is low
btfsc txbuf, 7 ; is bit out really low?
bef eeprom, do ; no, set it high
call BITOUT ; send the bit to serial EE
sf txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no - go again
call BITIN ; read ack bit
btfsc eeprom, di ; check ack bit
bcf port_a, ackf ; set acknowledge fail LED if the device did not pull data low

; *****************************************************
; RX
clr datai ; clear input buffer
movlw .8 ; set # bits to 8
```
Interfacing the 24LCXXB Serial EEPROMs

RXLP

 movwf count  ; rotate data left
 call BITIN  ; read a bit
 btfsc eeprom,di  ; set bit 0 if necessary
 bcf datai,0  ; 8 bits done?
goto RXLP  ; no, do another
 bcf eeprom,do  ; set ack bit = 1
 call BITOUT  ; to finish transmission
 retlw 0

; ****************************************************************
; Power up routine
; This is the program entry point. I/O line status is
; set for port A here.
; ****************************************************************

PWRUP

 movlw b'00000000'
 tris port_a  ; set port A as all output
 clrf port_a  ; all output lines low
 goto port_a

; ***************
; READ (read routine)
; This routine reads 8 consecutive addresses of the
; serial EE device starting at address 00 in the
; random access mode (non-sequential read). Reading
; the device using the random access mode
; requires that the address pointer be set for every
; byte before the read takes place. The address pointer
; is set by sending a 'write mode' control byte with the
; address of the location to read.
; ***************

READ

 bcf port_a,ackf  ; clear the ack fail LED if on
 movlw .8  ; set number of bytes to read as 8
 movwf bcount

 clrf addr

 rbyte call BSTART  ; generate start bit

 movlw b'10100000'  ; get slave address (write mode)
 movwf txbuf  ; into transmit buffer
 call TX  ; and send it
 movf addr,w  ; get word address
 movwf txbuf  ; into transmit buffer
 call TX  ; and send it

 movlw b'10100001'  ; now read one byte from the part

 call BSTART  ; generate start bit
 movlw b'10100001'  ; get slave address and read mode
 movwf txbuf  ; into transmit buffer
 call TX  ; and transmit it
 call RX  ; read 1 byte from serial EE
 call BSTOP  ; send stop bit to end transmission
 clrf addr  ; add 1 to address counter
 decfsz bcount  ; yes, are all 8 bytes read?
goto rbyte  ; no, do another byte
 goto READ  ; yes, start all over

END
Interfacing the 24LCXXB Serial EEPROMs

LIST P=16C54

;****************************
; 2-Wire Byte Write Program (123 bytes)
;
; This program demonstrates how to interlace a
Microchip PIC16C54 to a 24LCXX Serial EE device
and perform a byte write operation on 8 consecutive
addresses.

; This routine waits approximately 10mS for the write
cycle time, which is enough time for any of the
24LCxx devices to complete a write. A more efficient
method of determining when the write cycle is complete
is called "data polling." The data polling method is
explained in the program "2wpoll.asm." That
particular program uses data polling in a byte write
mode but it can be used in exactly the same way for
a page mode write.

As an option, the user may connect a LED to pin 18
on the PIC16CXX (use about a 1K resistor in series) as an
acknowledge fail indicator. This LED will come on if
the serial EE fails to acknowledge correctly.

Timing is based on using the PIC16CXX in 'XT' mode
using a 4Mhz crystal. Clock speeds to the serial EE
will be approximately 60 KHz for this setup.

PIC16CXX to Serial EE Connections:

PIC16CXX Serial EE
-- -- -- -- -- --
Pin 12 (RB6) --> SCLK
Pin 13 (RB7) --> SDATA

PIN 18 (RA1) --> Acknowledge fail LED (Optional)

;****************************
; Register Definitions
;****************************

port_a equ Sh ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port b) used for data and
clock lines
eeprom equ Oah ;bitbuffer
bycnt equ Obh ;byte counter for read mode
addr equ 0ch ;address counter
datai equ 0dh ;data input register
datao equ 0eh ;data output register
slave equ 0fh ;device address
(1010xxxx)
taxbuf equ 10h ;transmit buffer
count equ 11h ;bit counter
bcount equ 12h ;byte counter
loops equ 15h ;delay loop counter
loops2 equ 16h ;delay loop counter

; BitDefinitions
;****************************

d equ 7 ;eeprom input bit
do equ 6 ;eeprom output bit
sdata equ 7 ;serial EE data line (port_b,pin 13)
sclk equ 6 ;serial EE clock line (port_b,pin 12)
ackf equ 1 ;acknowledge fail LED (port_a,pin 18)

; org 01ffh ; set reset vector
org 0000h
org 0000h

DS00567A-page 6 © 1993 Microchip Technology Inc.

6-32
Interfacing the 24LCXXB Serial EEPROMs

; ;************************************************************************************
; ;       DELAY ROUTINE
; ; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
; ;************************************************************************************

WAIT
;
top movlw .110 ; timing adjustment variable
movwf loops2
top2
nop ; sit and wait
nop
nop
nop
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again
;
decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub

; ;************************************************************************************
; ;     Start Bit Subroutine
; ; this routine generates a start bit
; (Low going data line while clock is high)
; ;************************************************************************************

BSTART
bsf port_b,sdata ; make sure data is high
movlw b'00111111'
tris port_b ; set data and clock lines for output
bcf port_b,sclk ; make sure clock is low
nop
bsf port_b,sclk ; set clock high
nop
nop
nop
nop
nop
nop
bcf port_b,sdata ; data line goes low during
; high clock for start bit
nop
nop
nop
nop
nop
nop
nop
nop
; timing adjustment
bcf port_b,sclk ; start clock train
nop
nop
retlw 0 ;
;     End of Subroutine

; ;************************************************************************************
; ;       Stop Bit Subroutine
; ; This routine generates a stop bit
; (High going data line while clock is high)
; ;************************************************************************************

BSTOP
movlw b'00111111'
tris port_b ; set data/clock lines as outputs
bcf port_b,sdata ; make sure data line is low
nop
nop
nop
bsf port_b,sclk ; set clock high
nop
nop

© 1993 Microchip Technology Inc.
Interfacing the 24LCXXB Serial EEPROMs

BITOUT routine takes one bit of data in 'do' and transmits it to the serial EE device.

BITOUT
movlw b'00111111' ; set data, clock as outputs
tris port_b
bsf eeprom, do ; check for state of data bit to xmit
goto bitlow ;
bsf port_b, sdata ; set data line high
goto clkout ; go toggle the clock
bitlow
bsf port_b, sdata ; output a low bit
nop
nop
nop
nop
bcf port_b, sclk ; set clock line high
retlw 0

BITIN routine reads one bit of data from the serial EE device and stores it in 'di'.

BITIN
bsf eeprom, di ; assume input bit is high
movlw b'10111111' ; make sdata an input line
tris port_b
bsf port_b, sdata ; set sdata line for input
bsf port_b, sclk ; set clock line high
nop ; just sit here a sec
nop
nop
nop
nop
nop
nop
btfss ; read the data bit
bsf eeprom, di ; input bit was low
bcf port_b, sclk ; set clock line low

; Transmitter Data Subroutine
; This routine takes the byte of data stored in the 'datao' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then the device did not send a proper ack bit and the ack fail LED will be turned on.

TX
movlw .8
movwf count ; set the #bits to 8

nop
bsf port_b, sdata ; data goes high while clock high
; for stop bit
nop
nop
bcf port_b, sclk ; set clock low again
nop
nop
nop
retlw 0

; End of Subroutine
; BITOUT routine takes one bit of data in 'do' and transmits it to the serial EE device
;*******************************************************************************

End of Subroutine
Interfacing the 24LCXXB Serial EEPROMs

TXLP

bxr eeprom, do ; assume bit out is low
btfsc txbuf, 7 ; is bit out really low?
bsf eeprom, do ; otherwise data bit = 1
all BITOUT ; serial data out
or txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no - go again
gall BITIN ; readack bit
btfsc eeprom, di ; check ack bit
bsf port_a, ackf ; set acknowledge fail LED if the

retlw 0 ;

;******************************************************************************************

; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; byte write routine.
;******************************************************************************************

PWRUP

movlw b'00000000'
tris port_a ; set port A as all output
drf port_a goto wrbyte

;******************************************************************************************

Byte Write Routine
; This routine writes the data in "datao" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. This routine waits 10ms after every
; byte to give the device time to do the write. This
; program repeats forever.
;******************************************************************************************

WRBYTE

; clear all LEDs
movlw b'01010101' ; set slave address and write mode
movwf slave
movlw b'01010101' ; set data to write as 55h
movwf datao

; set number of bytes
movlw .8
movwf bcount ; to write as to 8
movw addr ; set starting address to 00

byte
gall BSTART ; generate start bit
movf slave, w ; move slave address
movwf txbuf ; into transmit buffer
gall TX ; and send it
movf addr, w ; move word address
movwf txbuf ; into transmit buffer
gall TX ; and send it
movf datao, w ; move data byte
movwf txbuf ; to transmit buffer
gall TX ; and transmit it
gall BSTOP ; generate stop bit

movlw .10
movwf loops ; set delay time to give
all WAIT ; 10 ms wait after every byte
inr addr ; add 1 to address counter
decfsz bcount ; all 8 bytes written?
goto byte ; no, do another byte

; start over

END
Interfacing the 24LCXXB Serial EEPROMs

LIST P=16C54
;***************************************************************
2-Wire Byte Write With Data Polling Program (129 bytes)
;
This program demonstrates how to interface a
Microchip PIC16C54 to a 24LCXX Serial EE device.
This program performs a byte write operation on 8
consecutive addresses using the data polling method

to determine when the write cycle is complete.
;
When writing to a serial E\(^\text{M}\) device, there are 2
ways to handle the internal timed write cycle
time of the device. The simplest method is
simply to wait until the maximum cycle time
is exceeded before attempting another command.
The other, more efficient method is known as "data
polling." Data polling is done by sending a start
bit and control byte to the part after the write
cycle has been initiated by a stop bit. If the ackbit
is low, then the device is through writing, otherwise
the sequence is repeated. If no low acknowledge
is found within 40 attempts (about 10 milliseconds)
than the routine times out and sets the timeout
LED (pin 1) high.
;
As an option, the user can connect a LED to pin 1
on the PIC16CXX (use about a 1K resistor in series) as a
timeout indicator. This LED will come on if the
data polling is unsuccessful and the device being
programmed does not respond. This can be tested by
simply running the program with no part in the socket.
;
Timing is based on using the PIC16CXX in 'XT' mode
using a 4Mhz crystal. Clock speeds to the serial EE
will be approximately 40 KHz for this setup.
;
PIC16CXX to Serial EE Connections:

; PIC16CXX    Serial EE
- - - - - - - - - - - - - -
Pin 12 (RB6) -> SCLK
Pin 13 (RB7) -> SDATA
;
Pin 1 (RA2) -> Write cycle timeout fail LED (optional)
;
***************************************************************
Register Definitions
***************************************************************
port_a equ Sh ; port 5 (port_a) used LEDs
port_b equ 6h ; port 6 (port_b) used for data and
                ; clock lines
EEPROM equ 0ah ; EEPROM
BYTENT equ 0bh ; byte counter for read mode
ADDR equ 0ch ; address counter
datai equ Odh ; data input register
datao equ Oeh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcounr equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
pollcnt equ 17h ; data polling counter
***************************************************************
Bit Definitions
***************************************************************
deq 7 ; EEPROM input bit
deq 6 ; EEPROM output bit
sdata eq 7 ; Serial EE data line (port_b, pin 13)
Interfacing the 24LCXXB Serial EEPROMs

sclk        equ  6 ; serial EE clock line (port_b, pin 12)
timeout     equ  2 ; write cycle timeout fail LED, port_a (pin 1)
ackf        equ  1 ; acknowledge fail LED, port_a (pin 18)

;org Olffh ; set reset vector
goto PWRUP
org 000h    ;
goto PWRUP

;***********************************************************************

; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;***********************************************************************

WAIT
;
top        movlw .110  ; timing adjustment variable
           movwf loops2
        top2    nop        ; sit and wait
           nop
           nop
           nop
           nop
           nop
           decfsz loops2     ; inner loops complete?
           goto top2        ; no, go again
           ;
           decfsz loops      ; outer loops complete?
           goto top          ; no, go again
           retlw 0          ; yes, return from sub

;***********************************************************************

; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;***********************************************************************

BSTART
      bcf    port_b,sdata  ; make sure data is high
      movlw  b'001111111'
      tris   port_b        ; set data and clock lines for output
      bcf    port_b,sclk    ; make sure clock is low
      nop
      bcf    port_b,sclk    ; set clock high
      nop
      nop
      nop
      nop
      nop
      bcf    port_b,sdata    ; data line goes low during
      ; high clock for start bit
      nop
      nop
      nop
      bcf    port_b,sclk    ; start clock train
      nop
      nop
      retlw  0
      
;***********************************************************************

; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;***********************************************************************

BSTOP
Interfacing the 24LCXXB Serial EEPROMs

```
movlw b'00111111';
tris port_b; set data/clock lines as outputs
bcf port_b,sdata; make sure data line is low
nop
nop
nop
bsf port_b,sclk; set clock high
nop
nop
nop
bsf port_b,sdata; data goes high while clock high
for stop bit
nop
nop
bcf port_b,sclk; set clock low again
nop
nop
retlw 0

;******************************************************************************
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;******************************************************************************
BITOUT
movlw b'00111111'; set data, clock as outputs
tris port_b
btfss eeprom,do; check for state of data bit to emit
goto bitlow
bcf port_b,sdata; set data line high
goto clkout; go toggle the clock

bitlow bcf port_b,sdata; output a low bit
nop
nop
nop
nop
bcf port_b,sclk; return clock line low
retlw 0

; End of Subroutine
;
;******************************************************************************
; BITIN routine reads one bit of data from the serial EE device and stores it in 'di'
;******************************************************************************
BITIN
bsf eeprom,di; assume input bit is high
movlw b'10111111'; make sdata an input line
tris port_b
bsf port_b,sdata; set sdata line for input
bsf port_b,sclk; set clock line high
nop
nop
nop

btfss port_b,sdata; read the data bit
bcf eeprom,di; input bit was low
bcf port_b,sclk; set clock line low

; End of Subroutine

; Transmit Data Subroutine
; This routine takes the byte of data stored in the 'datao' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the acknowledge bit. If the ack bit from the part was low
```
Interfacing the 24LCXXB Serial EEPROMs

then the transmission was successful. If it is high, then
the device did not send a proper ack bit and the ack
fail will be turned on.

TX

movlw .8
movwf count ; set the #bits to 8

TXLP

bcf eeprom,do ; assume bit out is low
btfsc txbuf, 7 ; is bit out really low?
rlf
txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no - go again
call BITIN ; read ack bit

retlw 0

Power up routine

This is the program entry point, which in this case simply
sets the port A I/O lines and directs control to the
write routine.

PWRUP

movlw b'00000000'
tis
rnf goto port_a

wr BYTE

movwf slave
movlw b'10101010'

movf addr,w ; set starting address to 00

call BSTART ; generate start bit
movf slave,w ; move slave address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; move word address
movf txbuf ; into transmit buffer
call TX ; and send it
movf datao.w ; move data byte
movwf txbuf ; to transmit buffer

© 1993 Microchip Technology Inc.
Interfacing the 24LCXXB Serial EEPROMs

```assembly
; and transmit it
CALL TX
; generate stop bit
CALL BSTOP
; now start polling for a low ack bit
CALL poll
; set max number of times to poll as 40
MOVlw .40
MOVWF pollcnt
; generate start bit
CALL BSTART
; move slave address (write mode)
MOVlw b'10100000'
MOVWF txbuf
; into transmit buffer
CALL TX
; and send it
BTFSS eeprom,di
; was the ack bit low?
GOTO exit
DECFSZ pollcnt
; is poll counter down to zero?
GOTO poll
; no, poll again. Otherwise the part is
BSF port_a,timeout
; not responding in time so set timeout
; LED and continue on

; add 1 to address counter
DECFSZ bcount
; all 8 bytes written?
GOTO byte
; no, do another byte
GOTO WRBYTE
; yes, start over

END
```
LIST P-16C54

;*******************************************************
; 2-Wire Page Write Program (122 bytes)
;
; This program demonstrates how to interface a
; Microchip PIC 16C54 to a 24LCXX Serial EE device
; and perform a page write operation on 8 consecutive
; addresses.
;
; All of Microchip's 2-wire serial EE devices have a
; buffer or 'page' that can be used as a more efficient
; method of writing to consecutive addresses. Some
; devices have page lengths of 2 bytes, others have
; page lengths of 4, 8, 16 or 64 bytes. Please consult
; the databook for the page length of the device you
; are using. THIS ROUTINE IS WRITTEN FOR A DEVICE
; WITH A PAGE LENGTH OF 8 OR MORE BYTES.
;
; When using page mode, the control byte and address
; are sent for the first address only. After the data
; byte for the first address is sent, the data for the
; next consecutive address is clocked in. This is
; repeated as many times as needed (as long as the page
; length is not exceeded) and then a stop bit is sent.
; The device will still acknowledge between every byte of
; data. After the stop bit is sent, the part will
; initiate the self timed write cycle. For all of
; Microchip's 24LCxx devices, the cycle time for
; a byte write and a page write is the same. Therefore,
; writing 8 bytes in byte mode with a typical 5ms write
; cycle consumes 40ms of wait time, while the same data
; written in page mode would consume only 5ms of wait time.
;
; This routine waits approximately 10ms for the write
; cycle time, which is enough time for any of the
; 24LCxx devices to complete a write. A more efficient
; method of determining when the write cycle is complete
; is called "data polling." The data polling method is
; explained in the program "2wpoll.asm." That
; particular program uses data polling in a byte write
; mode but it can be used in exactly the same way for
; a page mode write.
;
; As an option, the user can connect a LED to pin 18
; on the PIC (use about a 1K resistor in series) as a
; acknowledge fail indicator. This LED will come on
; if the device being programmed does not send a low
; acknowledge bit at the proper times. This can be
; tested by simply running the program with no part
; in the socket.
;
; Timing is based on using the PIC in 'XT' mode
; using a 4Mhz crystal. Clock speeds to the serial EE
; will be approximately 40 Khz for this setup.
;
; PIC to Serial EE Connections:
;
; PIC            Serial EE
; ----------     ----------
; Pin 12 (RB6)  ->  SCLK
; Pin 13 (RB7)  ->  SDATA
; PIN 18 (RA1)  ->  Acknowledge Fail LED (Optional)
;
;*******************************************************

Register Definitions
;*******************************************************

© 1993 Microchip Technology Inc.
Interfacing the 24LCXXB Serial EEPROMs

port_a equ 5h ; port 5 (port a) used for LED outputs
port_b equ 6h ; port 6 (port b) used for data and
 ; clock lines
eeprom equ 0ah ; bit buffer
daddr equ 0ch ; address counter
dataeq equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter

Bit Definitions

d  equ 7 ; eeprom input bit
do  equ 6 ; eeprom output bit
data eq 7 ; serial EE data line (port_b, pin 13)
sclk equ 6 ; serial EE clock line (port_b, pin 12)
ackf equ 1 ; acknowledge fail LED, port_a (pin 18)

; ;******************************************************************************
; org 01ffh ; set reset vector
; goto PWRUP
; org 000h ;
; goto PWRUP

; ;******************************************************************************
; DELAY ROUTINE
; This routine takes the value in 'loops' 
; and multiplies it times 1 millisecond to 
; determine delay time.
;******************************************************************************

WAIT

; top movlw .110 ; timing adjustment variable
; movwf loops2

;top2  nop ; sit and wait
; nop
; nop
; nop
; nop
; nop
; decfsz loops2 ; inner loops complete?
; goto top2 ; no, go again
; decfsz loops ; outer loops complete?
; goto top ; no, go again
; retlw 0 ; yes, return from sub

; ;******************************************************************************
; Start Bit Subroutine
; this routine generates a start bit
; (low going data line while clock is high)
;******************************************************************************

; START
; bsf port_b,sdata ; make sure data is high
; movlw b'01111111'
; tri port_b ; set data and clock lines for output
; bcf port_b,sclk ; make sure clock is low
; bcf
; bcf port_b,sclk ; set clock high
; bcf
; bcf
; bcf
; bcf

DS00567A-page 16 © 1993 Microchip Technology Inc.
Interfacing the 24LCXXB Serial EEPROMs

```assembly
bcf port_b, sdata ; data line goes low during
nop ; high clock for start bit
nop
nop
nop
nop ; timing adjustment
bcf port_b, sclk ; start clock train
nop
nop
retlw 0
;
; End of Subroutine

;******************************************************************************

; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;******************************************************************************

BSTOP
movlw b'00111111'
tris port_b ; set data/clock lines as outputs
bcf port_b, sdata ; make sure data line is low
nop
nop
nop
bsf port_b, sclk ; set clock high
nop
nop
bsf port_b, sdata ; data goes high while clock high
nop
nop
bcf port_b, sclk ; set clock low again
nop
nop
nop
retlw 0
;
; End of Subroutine

;******************************************************************************

; BITOUT routine
; This routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;******************************************************************************

BITOUT
movlw b'00111111' ; set data,clock as outputs
tris port_b ; check for state of data bit to xmit
goto bitlow ;
bsf port_b, sdata ; set data line high
goto clkout ;

bitlow bcf port_b, sdata ; output a low bit

clkout bcf port_b, sclk ; set clock line high
nop
nop
nop
bcf port_b, sclk ; return clock line low
retlw 0
;
; End of Subroutine

;******************************************************************************

; BITIN routine reads one bit of data from the
; serial EE device and stores it in 'di'
;******************************************************************************

BITIN
```
Interfacing the 24LCXXB Serial EEPROMs

bsf eeprom, di  ; assume input bit is high
movlw b'10111111'  ; make sdata an input line
tris port_b
bsf port_b, sdata  ; set sdata line for input
bsf port_b, sclk  ; set clock line high

nop  ; just sit here a sec
nop
nop
nop

btfss port_b, sdata  ; read the data bit
bcf eeprom, di  ; input bit was low
bcf port_b, Selk  ; set clock line low

retlw 0

; TX
movlw .8
movwf count  ; set the #bits to 8

TXLP
bcf eeprom, do  ; assume bit out is low
btfsc txbuf, 7  ; is bit out really low?
bsf eeprom, do  ; otherwise data bit =1
call BITOUT  ; serial data out
ff txbuf  ; rotate txbuf left
decfsz count  ; 8 bits done?
goto TXLP  ; no - go again
call BITIN  ; read ack bit
btfsc eeprom, di  ; check ack bit
bsf port_a, ackf  ; set acknowledge fail LED if the
                  ; device did not pull data low

retlw 0

; PWRUP
movlw b'00000000'
tris port_a  ; set port A as all output
clr port_a  ; all output lines low

; WRPAGE

; Page Write Routine
; This routine uses page mode to write the data in "datao" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. This routine waits 10ms after every
; page to give the device time to do the write. This
; routine executes forever
;
Interfacing the 24LCXXB Serial EEPROMs

clr port_a ; clear all LEDs
movlw b'10100000' ; set slave address and write mode
movwf slave
movlw b'01010101' ; set data to write as 55h
movwf datao

movlw .8 ; set number of bytes
movwf bcount ; to write as to 8
clr addr ; set starting address to 00
;
call BSTART ; generate start bit
movf slave,w ; move slave address
movwf txbuf ; into transmit buffer
call TX ; and send it

movf addr,w ; move word address
movwf txbuf ; into transmit buffer
call TX ; and send it

byte movf datao,w ; move data byte
movwf txbuf ; to transmit buffer
call TX ; and transmit it
decfsz bcount ; all 8 bytes written?
goto byte ; no, do another
call BSTOP ; yes, generate stop bit
;
movlw .10 ; set delay time to give
movwf loops ; 10 ms wait after every byte
;
goto wrpage ; start over
;
END
LIST P=16C54

; 2-Wire Sequential Read Program (120 bytes)
;
; This program demonstrates how to interface a
; Microchip PIC16C54 to a 24LCXX Serial EE device
; and perform a sequential read operation. A sequential
; read involves setting the address pointer once and then
; using the auto-increment ability of the part to read
; consecutive addresses by simply providing more clocks.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4MHz crystal. Clock speeds to the serial EE
; will be approximately 60 KHz for this setup.
;
; As an option, the user may connect a LED to pin 18
; on the PIC16CXX (use about a 1K resistor in series) as an
; acknowledge fail indicator. This LED will come on if
; the serial EE fails to acknowledge correctly.
;
; PIC16CXX to Serial EE Connections:
; PIC16CXX Serial EE
; -- -- -- -- -- -- -- --
; Pin 12 (RB6) --> SCLK
; Pin 13 (RB7) --> SDATA
; PIN 18 (RA1) -> Acknowledge fail LED (Optional)
;
; Register Definitions

port_a equ 5h ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port_b) used for data and
; clock lines
eeprom equ 0ah ; bitbuffer
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txhlf equ lOh ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter

; BitDefinitions

d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b,pin13)
sclk equ 6 ; serial EE clock line (port_b,pin12)
ackf equ 1 ; acknowledge fail LED, port_a

org 01ffh ; set reset vector
goto PWRUP
org 000h ;
goto PWRUP

; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)

BSTART
Interfacing the 24LCXXB Serial EEPROMs

bsf port_b, sdata ; make sure data is high
movlw b'00111111'
tris port_b ; set data and clock lines for output
bcf port_b, sclk ; make sure clock is low
nop
bsf port_b, sclk ; set clock high
nop
nop
nop
nop
nop
bcf port_b, sdata ; data line goes low during ; high clock for start bit
nop
nop
nop
nop
nop
nop
bcf port_b, sclk ; timing adjustment
nop
nop
nop
nop
nop
nop
retlw 0
; End of Subroutine

; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;*******************************************************************************
BSTOP
movlw b'00111111' ;
tris port_b ; set data/clock lines as outputs
bcf port_b, sdata ; make sure data line is low
nop
nop
nop
bcf port_b, sclk ; set clock high
nop
nop
nop
nop
bsf port_b, sdata ; data goes high while clock high ; for stopbit
nop
nop
bcf port_b, sclk ; set clock low again
nop
nop
retlw 0
;
; End of Subroutine
;*******************************************************************************
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;*******************************************************************************
BITOUT
movlw b'00111111' ; set data, clock as outputs
tris port_b ; set data/clock lines as outputs
btssel eeprorn, do ; check for state of data bit to transmit
goto bitlow ;
bcf port_b, sdata ; set data line high
goto clkout ; go toggle the clock
bitlow bcf port_b, sdata ; output a low bit
nop
nop
clkout bcf port_b, sclk ; set clock line high
nop
nop
nop
nop
nop

Interfacing the 24LCXXB Serial EEPROMs

```assembly
; bcf    port_b,sclk ; return clock line low
retlw   0
;
; End of Subroutine
;
;****************************************************************

BITIN routine reads one bit of data from the
serial EE device and stores it in 'di'
;****************************************************************

BITIN

bsf    eeprom,di ; assume input bit is high
movlw  b'10111111' ; make sdata an input line
tris   port_b
bsf    port_b,sdata ; set sdata line for input
bsf    port_b,sclk ; set clock line high
nop
nop
nop
nop
btfss  port_b,sdata ; read the data bit
bcf    eeprom,di ; input bit was low, set 'di' accordingly
bcf    port_b,sclk ; set clock line low

retlw  0
;
;****************************************************************

Transmit Data Subroutine
This routine takes the byte of data stored in the
'vedata' register and transmits it to the serial EE device.
It will then send one more clock to the serial EE for the
acknowledge bit. If the ack bit from the part was low
then the transmission was successful. If it is high, then
the device did not send a proper ack bit and the ack
fail LED will be turned on.
;****************************************************************

TX

movlw .8
movwf  count ; set the # bits to 8

TXLP

bcf    eeprom,do ; assume bit out is low
btfsc  txbuf,7 ; is bit out really low?
bcf    eeprom,do ; no, set it high
call   BITOUT ; send the bit to serial EE
if    txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto   TXLP ; no - go again
call   BITIN ; read ack bit
btfsc  eeprom,di ; check ack bit
bsf    port_a,ackf ; set acknowledge fail LED if the
device did not pull data low
;
retlw  0
;
;****************************************************************

Receive Data Routine
This routine reads one byte of data from the part
into the 'datai' register. It then sends a high
ack bit to indicate that no more data is to be read
;****************************************************************

RX

movlw .8 ; set # bits to 8
movwf  count

RXLP

if    datai ; rotate datai 1 bit left
call   BITIN ; read a bit
btfsc  eeprom,di
```
Interfacing the 24LCXXB Serial EEPROMs

```assembly
bsf datai,0 ; set bit 0 if necessary
decfsz count ; 8 bits done?
goto RXLP ; no, do another
retlw 0

;****************************************************************
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; Read routine.
;****************************************************************
PWRUP
    movlw b'00000000'
    tris port_a ; set port A as all output
    clrf port_a ; all output lines low
    goto READ

;****************************************************************
; READ (sequential read routine)
; This routine reads 8 consecutive addresses of the
; serial EEPROM device starting at address 00 in the
; sequential read mode. Reading in this mode is more
; efficient than the random read mode as the control byte
; and address have to be sent only once at the beginning
; of the sequence. As many consecutive addresses as
; needed can then be read from the part until a stop bit is
; sent. In the read mode, the PIC16CXX must send the acknowledge
; bit after every 8 data bits from the device. When the
; last byte needed has been read, then the controller will
; send a high acknowledge bit and then a stop bit to halt
; transmission from the device.
;****************************************************************
READ
    bcf port_a,ackf ; clear the ack fail LED if on
    movlw .8
    movwf bcount ; set number of bytes to read as 8
    movlw b'10100000' ; set slave address and write mode
    clrf addr ; set starting address to 00
;    call BSTART ; generate start bit
    movw slave,w ; get slave address
    movw txbuf ; into transmit buffer
    call TX ; and send it
    movf addr,w ; get word address
    movw txbuf ; into transmit buffer
    call TX ; and send it
    movlw b'10100001' ; get slave address and read mode
    movw txbuf ; into transmit buffer
    call TX ; and send it
    rbyte call RX ; read 1 byte from device
    decfsz bcount ; are all 8 bytes read?
    goto lowack ; no, send low ack and do another
    bsf eeprom,do ; yes, send high ack bit
    call BITOUT ; to stop transmission
    call BSTOP ; and send a stop bit
    goto READ ; start all over
    lowack bcf eeprom,do ; send low ack bit
    call BITOUT ; to continue transmission
    goto rbyte ; and read another byte

END
```

© 1993 Microchip Technology Inc.
USING THE SMART SERIAL™ SERIES

With the advent of CMOS silicon devices came the battery powered application. The battery powered application required more functionality and thus more power from the microcontroller which required more power from the batteries which cost more. Just as nature has a definite cycle, so it seems, does the portable application. Nowhere has this been more evident than in the areas of personal communication and data acquisition. Never before in the history of the industrial revolution have the type of applications seen emerging today been possible. Every man-made object that requires human or machine interface, has the potential to be controlled by embedded circuits and to be powered by the advanced technology batteries available today. Along with this increased functionality comes the requirement for MORE MEMORY. More in a smaller package, at a better price requiring less power. The majority of the hand-held embedded applications either use what little non-volatile memory that is available on the controllers or they use external devices. Parallel EEPROMs require too many I/O connects, which is the major source of device active current. Serial EEPROMs are typically the answer for these applications and the industry standard I'C™ is the industry leader (>70 % worldwide). The I'C protocol assigns a slave address for each unique device or device family. Microchip Technology, and most non-volatile suppliers, use the slave address of 1010 for serial electrical erasable programmable read only memory (Serial EEPROM). The protocol also facilitates up to a maximum of 16K bytes of memory on the bus via the 8-bit address and the three device or memory block select pins A0, A1, and A2. Here lies the dilemma: with the advent of the more sophisticated personal communication devices such as cellular and full featured phones, personal digital assistants and palm-top computers, 16K bytes is not enough!

Smart Serial Products

The Smart Serial concept grew from the industry need for increased memory requirements in I2C embedded applications, smarter endurance performance, security needs, and the need for more functionality at lower power demands. Currently, Microchip Technology Inc. has the 24C65 and 24C32 devices available. All comments in this application note pertain primarily to the 24C65 but the routines and general architectural comments apply to the 24C32 as well.

The user should reference the individual data sheets for specific differences.

The Microchip Technology Inc. 24C65 is a serial memory device with 64K bits (8Kx8) capacity and additional patent pending unique features not found anywhere else. First let's look at the current I'C addressing scheme, the cascadable solution, and finally the Microchip total embedded systems solution.

I'C ADDRESSING

The I'C protocol utilizes a master/slave bi-directional communication bus. The master, usually a microcontroller, which controls the bus, generates the serial clock (SCL) originates the start and stop conditions. A Serial EEPROM is considered a slave device and is defined as a transmitter during read operations and generates acknowledges when receiving data from the master. The start and stop bits are utilized to control the bus. Normal operation begins with a start bit and ends with a stop bit. Following a start, commands begin with an 8 bit 'control' byte originated by the master. The control byte identifies the slave device to be addressed and defines the operation to take place. A typical control byte for a Serial EEPROM (slave address = 1010) is shown in figure 1. The control byte, therefore, consists of a start bit, a four-bit slave address, a read/write bit and an acknowledge. The slave address consists of the 1010 identifying address plus the three block or chip select bits.
Using the 24C65 and 24C32

All memory and peripheral devices on the I²C bus conform to this sequence for identification and selection. There are 128 assigned slave addresses in the standard protocol. There is a 10-bit extension to the protocol for 1024 additional slave addresses.

THE CASCADABLE SOLUTION

Cascading is an addressing scheme used in the 24164 (2Kx8 EEPROM) to enable using more than the 16K limit set forth by the standard. In this method the A0, A1, and A2 pins are mapped into bits 2, 3, and 4 of the 8-bit slave address. This approach allows the system designer to use the non-standard 24164 to increase the total memory of a given memory subsystem. This solution is definitely workable, but does not offer the user the system design flexibility needed for the newer and more complex systems. Most system architects would prefer to have a linear address space for program and data memory. Programmers also find the linear solution more attractive. The overhead required for bank switching and chip selection usually requires additional overhead and hardware.

THE ULTIMATE SOLUTION BY MICROCHIP

Microchip Technology Inc. has designed an addressing scheme based on the standard I²C protocol and device addresses but incorporating an additional address byte for enabling the designer to use up to 256K bits per device and add from 1 to 8 devices on the system bus. This flexibility allows for future memory expansion and more advanced features in a smaller, more cost effective, design. This enhanced addressing combined with the many advanced and patent pending features of the 24C65 make the 24C65 an exciting and innovative device. It is the first in a family of sophisticated Smart Serials™ EEPROMs from Microchip Technology Inc.

24C65 Features

The 24C65 has an advanced architecture with the following features:

- 15-bit, 2-byte address field (14 bit for the 24C32)
- 4K-bit High Endurance Block - 1 Million E/W cycles typical (Fixed at the last 4k bit block in the array)
- Programmable write protect security features with up to a 15 blocks of 4K bits
- 8 byte by 8 line input write cache for I²C Fast Mode, burst mode capability, and use as a capture buffer

24C65 Addressing

For the first byte or control byte, the 24C65 adheres to the I²C protocol (reference figure 2). This is the first byte received following the start condition from the master device. The control byte consists of a four-bit control code for the 24C65 (this is assigned as 1010 binary for read and write operations). The next three bits of the control byte are the device select bits (A2, A1, A0). They are used by the master device to select which of the eight devices are to be accessed. These bits are in effect the three most significant bits of the word address. The last bit of the control byte defines the operation to be performed. When set to one a read operation is selected, when set to zero a write operation is selected. The least significant 13 bits of the next 2 bytes define the address of the first byte within the 8K block. The most significant byte is transferred first. Following the start condition, the 24C65 monitors the SDA bus, checking the device type identifier being transmitted, upon receiving a 1010 code and appropriate device select bits, the slave device outputs an acknowledge signal on the SDA line. Depending on the state of the R/W bit, the 24C65 will select a read or write operation.

The addressing scheme uses the standard first byte slave address format of the I²C standard with the Microchip Technology-assigned 1010 slave address. The internal bus controller scans the next byte for bit 7 to be asserted indicating that a security operation is to take place. This will be further explained later. The remainder of the byte is composed of the most significant address bits. The next byte is the least significant address byte. See figure 2 for graphical representation of this sequence. After receiving a valid 2 byte address and a stop bit, the 24C65 will process this address according to bit zero of the control byte and either wait for data to be written, if in a write sequence, or present the requested data if in a read sequence.

ADVANCED FEATURES

Programmable security

The 24C65 has a sophisticated security mechanism by which selected blocks of memory may be write protected by the user. The write sequence includes a bit for enabling the security protection scheme, bit 7 of the first address byte. When this bit is set to one, the first byte
following the address during a write sequence defines the security block. This includes a pointer to the starting 4K block to be protected (the write address), a write/erase flag, and a non-zero four bit code for determining the number of 4K blocks to protect up to the maximum of 15 (60K). The 4K blocks must be contiguous. The high endurance block cannot be protected. In a normal application the security block or blocks would be set after all code or look-up table data has been finalized. THIS OPERATION CAN ONLY BE PERFORMED ONCE. (See Figures 2 and 3.)

**Total Endurance™**

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate, for each byte in the application, not on the number of op codes or control byte commands, and is never based on any read functions whether they be a data read or configuration read. If a
d part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. Applying the predictive data, from the 24LC04 which has similar characteristics, to the 4K high endurance block and assuming the following:

- a five-year life for a personal communication device
- an expected E/W cycles of 10 times per day
- a last number redial function of 11 bytes

Operational specifications:

<table>
<thead>
<tr>
<th>Device</th>
<th>24C65</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>5</td>
</tr>
<tr>
<td>Temperature</td>
<td>25 C</td>
</tr>
<tr>
<td>Bytes/Cycle</td>
<td>11</td>
</tr>
<tr>
<td>E/W/Day</td>
<td>10</td>
</tr>
<tr>
<td>App. Life (Yr.)</td>
<td>5</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>BYTE</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>RANDOM</td>
</tr>
</tbody>
</table>

The 4k HE block with 1 M E/W cycles typical, in this application, should yield the following results:

| FIT      | 1.0 |
| PPM      | 6   |
| Time     | 5.0 |
| Write cycles | 18,250 |

---

**FIGURE 2 - CONTROL BYTE ALLOCATION**

START READ/WRITE

SLAVE ADDRESS R/W A

1 0 1 0 A2 A1 A0

**FIGURE 3 - BYTE WRITE**

BUS ACTIVITY: MASTER CONTROL WORD ADDRESS (1) WORD ADDRESS (0) DATA

SDA LINE

STOP

**BUS ACTIVITY**

A A A A

C C C C

K K K K

© 1993 Microchip Technology Inc.
Using the 24C65 and 24C32

The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance see the related application notes AN537 and AN562 contained in this volume. (See Figure 4.)

64 Byte Write Cache

The cache is arranged in 8-byte pages by 8 lines each. This yields a total of 64 bytes. The interface to the 24C65 supports both the standard 100 kHz mode and FAST mode at 400 kHz. The input cache can therefore support a burst write option of up to 64 bytes. When using the I²C protocol, an end of a page is defined by the transmission of a stop bit by the master. This sequence in the 24C65 could be used to define pages from 8 to 64 bytes in length. The 8 byte by 8 line cache will roll over if a write is attempted past byte 8 of line 8, thus it can be used as a 64 byte capture or snapshot buffer. Each line is overwritten during a subsequent write to the same line. Faster memory and controller interfaces will become increasingly important in applications incorporating the ACCESS.bus interface standard.

Power Management

Increasingly, power management is becoming a predominant requirement for hand-held devices where a limited amount of power is available from the total power budget for a particular function. The 24C65 has built-in power saving features such that the entire device is put into standby mode upon receiving a stop bit or an abort when in a read sequence, and after the completion of writing the data in the cache lines to the array when in an erase/write sequence. When the device is in standby mode, the only active circuit is the input circuit for the I²C clock. This yields a standby current that is the current consumption of this lone input and the normal leakage current of the silicon and typically will be less than 2 µA.

The 24C65 is the first device available in the Smart Serial product line. The features and capabilities initiated with this device will become standard on all future Microchip Technology Inc. serials and some features will be enhanced, such as the security options.

Appendix A of this application note contains the required PIC16C54 assembly routines for setting the security function, addressing the Smart Serial devices and using the most common addressing, read, write, and data polling functions for the I²C bus.

All of the list programs are complete stand-alone programs.

AUTHORS: Richard J. Fisher
Memory Products Division

Bruce Negley
Memory Products Division

© 1993 Microchip Technology Inc.
FIGURE 5 - CACHE WRITE

DATA STARTING AT BYTE ZERO ON A PAGE BOUNDARY AND WRITING 64 BYTES

Page Cache 8 Byte x 8

Page 0 bytes 0-7
Page 1 bytes 8-15
Page 2 bytes 16-23
Page 3 bytes 24-31
Page 4 bytes 32-39
Page 5 bytes 40-47
Page 6 bytes 48-55
Page 7 bytes 56-63

DATA STARTING AT BYTE THREE OF A PAGE BOUNDARY AND WRITING 64 BYTES (3 LSB=0011)

**

NOTE: IF THE CACHE IS WRITTEN TO PAST THE 64TH BYTE, IT WILL ROLL OVER TO BYTE 0 OF THE FIRST PAGE WRITTEN.

```
Using the 24C65 and 24C32

© 1993 Microchip Technology Inc.
```
APPENDIX A

LIST P-16C54

64K Byte Read Program (138 bytes)

This program demonstrates how to interface a Microchip PIC16C54 to a 24LC65 Serial EEPROM device and perform a random read operation. This program will read 8 consecutive addresses in the 'random read' mode. This entails setting the address pointer before doing the read command for each address.

Another, more efficient method of reading consecutive addresses is the 'sequential read' mode. This involves sending the control byte and address for the first byte to read, then continuing to provide clocks for the next addresses. The device will automatically increment the address. An example of the sequential read mode is provided in the '64kseq.asm' file.

Timing is based on using the PIC16CXX in 'XT' mode using a 4Mhz crystal. Clock speeds to the serial EE will be approximately 60 KHz for this setup.

As an option, the user may connect a LED to pin 18 on the PIC16CXX (use about a 1K resistor in series) as an acknowledge fail indicator. This LED will come on if the serial EE fails to acknowledge correctly.

PIC16CXX to Serial EE Connections:

PIC16CXX Serial EE
-- -- -- -- -- --
Pin 12 (RB6) -> SCLK
Pin 13 (RB7) -> SDATA

PIN 18 (RA1) -> Acknowledge fail LED (Optional)

Register Definitions

port_a equ 5h ; port 5 (port a) used for LED display
port_b equ 6h ; port 6 (port b) used for data and clock lines

eeprom equ 0ah ; byte buffer
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; word 0 address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address 1010xx00)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter

Bit Definitions

d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b,pin 13)
sclk equ 6 ; serial EE clock line (port_b,pin 12)
ackf equ 1 ; acknowledge fail LED (port_a)
Using the 24C65 and 24C32

org 01ffh ; set reset vector
goto PWRUP
org 000h
goto PWRUP

;******************************************************************************
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
determine delay time.
;******************************************************************************

DELAY ROUTINE

wait loops
nop
nop
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again

decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub

;******************************************************************************
; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;******************************************************************************

BSTART

bsf port_b, sdata ; make sure data is high
movlw b'00111111'
tlrs port_b ; set data and clock lines for output
bcf port_b, sclk ; make sure clock is low
bsf port_b, sclk ; set clock high
bcf port_b, sdata ; data line goes low during
; high clock for start bit
nop
cbcf
; timing adjustment

bsf port_b, sclk ; start clock train
nop
nop
retlw 0

;******************************************************************************
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;******************************************************************************

BSTOP

bcf port_b, sdata ; make sure data line is low
movlw b'00111111'
tlrs port_b ; set data/clock lines as outputs
bcf port_b, sdata ; make sure data line is low
nop

Using the 24C65 and 24C32

BITOUT routine takes the bit of data in 'do' and transmits it to the serial EE device

```
BITOUT
    movlw b'00111111'
    tris port_b
    btfss eeprom,do
    goto bitlow
    bsf port_b,sdata
    high? set data line high
    goto clkout
    go toggle the clock

bitlow
    bcf port_b,sdata
    output a low bit

clkout
    bcf port_b,sclk
    set clock line high

```

BITIN routine reads one bit of data from the serial EE device and stores it in the bit 'di'

```
BITIN
    bsf eeprom,di
    ; assume input bit is high
    movlw b'10111111'
    tris port_b
    bsf port_b,sdata
    ; set sdata line for input
    bsf port_b,sclk
    ; set clock line high
    nop
    ; just sit here a sec
    btfss port_b,sdata
    ; read the data bit
    bcf eeprom,di
    ; input bit was low, set 'di' accordingly
    bcf port_b,sclk
    ; set clock line low
    retlw 0
```

Transmit Data Subroutine

This routine takes the byte of data stored in the 'datao' register and transmits it to the serial EE device.
It will then send 1 more clock to the serial EE for the acknowledge bit. If the ack bit from the part was low then the transmission was successful. If it is high, then the device did not send a proper ack bit and the ack fail LED will be turned on.

```
TX
    movlw .8
    movwf count ; set the #bits to 8
```
Using the 24C65 and 24C32

TXLP

bcf eeprom,do ; assume bit out is low
btfsc txbuf,7 ; is bit out really low?
bsf eeprom,do ; no, set it high
call BITOUT ; send the bit to serial EE
sr txbuf ; rotate txbuf left
decfz count ; 8 bits done?
goto TXLP ; no - go again
call BITIN ; read ack bit
btfsc eeprom,di ; check ack bit
bsf port_a,ackf ; set acknowledge fail LED if the
device did not pull data low
retlw 0 ;

;******************************************************************************
; Receive data routine
; This routine reads one byte of data from the part
; into the 'datai' register. It then sends a high
; ack bit to indicate that no more data is to be read
;******************************************************************************
RX

movlw .8 ; set # bits to 8
movwf count
rlf cbtai ; rotate datai 1 bit left
call BITIN ; read bit
btfsc eeprom,di ; set bit 0 if necessary
decfz count ; 8 bits done?
goto RXLP ; no, do another
bsf eeprom,do ; set ack bit = 1
call BITOUT ; to finish transmission
retlw 0 ;

;******************************************************************************
; Power up routine
; This is the program entry point. I/O line status is
; set for port A here.
;******************************************************************************
PWRUP

movlw b'00000000'
tris port_a ; set port A as all output
clrf port_a ; all output lines low
goto port_a

;******************************************************************************
; READ (read routine)
; This routine reads 8 consecutive addresses of the
; serial EE device starting at address 00 in the
; random access mode (non-sequential read). Reading
; the device using the random access mode
; requires that the address pointer be set for every
; byte before the read takes place. The address pointer
; is set by sending a 'write mode' control byte with the
; address of the location to read.
;******************************************************************************
READ

bcf port_a,ackf ; clear the ack fail LED if on
movlw .8 ; set number of bytes to read as 8
movwf bcount ;
clrf addrl ; set starting high address byte to 00
clrf addr ; set starting low address byte to 00
rbyte call BSTART ; generate start bit
Using the 24C65 and 24C32

; now send the write control byte and
; address to set the pointer

movlw b'10100000' ; get slave address (write mode)
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addrl,w ; get word 1 address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; get word 0 address
movwf txbuf ; into transmit buffer
call TX ; and send it

; now read one byte from the part

call BSTART ; generate start bit
movlw b'10100001' ; get slave address and read mode
movwf txbuf ; into transmit buffer
call TX ; and transmit it
call RX ; read 1 byte from serial EE
call BSTOP ; send stop bit to end transmission
incf addr ; add 1 to address counter
declfsz bcount ; yes, are all 8 bytes read?
goto rbyte ; no, do another byte

movlw .255 ; long delay for scope
movwf loops ; trigger purposes only
call wait
goto READ ; yes, start all over

END
Using the 24C65 and 24C32

LIST P=16C54

; 64K Byte Write Program (127 bytes)
;
; This program demonstrates how to interface a
Microchip PIC16C54 to the 24C65/32 Serial E2 device
and perform a byte write operation on 8 consecutive
addresses.
;
; After each byte is written, time must be given to the
device for it to complete the write cycle before
the next command can be sent. The easiest solution
is to consult the data book for the maximum write
cycle time and just wait that long before the next
command is sent (10ms for this device). This program
demonstrates that solution.
;
; Another, more efficient method of determining when the
write cycle is complete is called 'data polling.' This
method is demonstrated in the program "64kdpoll."
;
; As an option, the user may connect a LED to pin 18
on the PIC16CXX (use about a 1K resistor in series) as an
acknowledge fail indicator. This LED will come on if
the serial EE fails to acknowledge correctly.
;
; Timing is based on using the PIC16CXX in 'XT' mode
using a 4MHz crystal. Clock speeds to the serial EE
will be approximately 60Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX       Serial EE
; ---------------   -----------
; Pin 12 (RB6)  --> SCLK
; Pin 13 (RB7)  --> SDATA
;
; PIN 18 (RA1)  --> Acknowledge fail LED (Optional)
;
;******************************************************************************
;
; Register Definitions
;******************************************************************************

port_a equ Sh ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port_b) used for data and
clock lines
eeeprom equ 0ah ; EEPROM input bit
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; word 0 address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter

;******************************************************************************
;
; Bit Definitions
;******************************************************************************

d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b,pin 13)
sclk equ 6 ; serial EE clock line (port_b,pin 12)
ackf equ 1 ; acknowledge fail LED (port_a,pin 18)

;******************************************************************************
Using the 24C65 and 24C32

```assembly
org 01ffh ; set reset vector
goto PWRUP
org 000h ;
goto PWRUP

;******************************************************************************
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;******************************************************************************
WAIT

; top movlw .110 ; timing adjustment variable
movwf loops2

; sit and wait
nop
nop
nop
nop
nop

; inner loops complete?
decfsz loops2 goto top2

; no, go again

; outer loops complete?
goto top

; no, go again

decfsz loops goto top2

; yes, return from sub
retlw 0

;******************************************************************************
; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;******************************************************************************

; make sure data is high
bsf port_b, sdata

; set data and clock lines for output
movlw b'00111111'
tris port_b

; make sure clock is low
bcf port_b, sclk

; set clock high
bsf port_b, sclk

; data line goes low during
nop

; high clock for start bit

; timing adjustment
bcf port_b, sclk

; start clock train

retlw 0
```
Using the 24C65 and 24C32

;***************************************************************
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;***************************************************************

BSTOP

movlw b'00111111'
tris port_b
bcf port_b, sdata
nop
nop
nop
bsf port_b, sclk
nop
nop
nop
bsf port_b, sdata
nop
nop
bcf port_b, sclk

retlw 0

;***************************************************************
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;***************************************************************

BITOUT

movlw b'00111111'
tris port_b
bsf port_b, sdata
goto bitlow
btfss bitlow
bsf port_b, sclk
goto clkout

bitlow bcf port_b, sdata

clkout bcf port_b, sclk

nop
nop
nop
nop
bcf port_b, sclk

retlw 0

;***************************************************************
; BITIN routine reads one bit of data from the
; serial EE device and stores it in 'di'
;***************************************************************

BITIN

bsf eeprom, di

movlw b'10111111'
tris port_b
bsf port_b, sdata
bsf port_b, sclk
nop
nop
nop
nop
nop
nop
nop
btfss port_b, sdata
bcf eeprom, di
bcf port_b, sclk

retlw 0

;
Using the 24C65 and 24C32

;************************************************************************************************************
; Transmit Data Subroutine
; This routine takes the byte of data stored in the 'datao' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack fail LED will be turned on.
;************************************************************************************************************

TX
  movlw .8 ; set the #bits to 8

TXLP
  bcf eeprom,do ; assume bit out is low
  btfsc txbuf, 7 ; is bit out really low?
  bsf eeprom,do ; otherwise data bit = 1
  call BITOUT ; serial data out
  rlf txbuf ; rotate txbuf left
  decfsz count ; 8 bits done?
  goto TXLP ; no - go again
  call BIT IN ; read ack bit
  btfsc eeprom,di ; check ack bit
  bsf port_a, ackf ; set acknowledge fail LED if the

retlw 0

;************************************************************************************************************

Power up routine
This is the program entry point, which in this case simply
sets the port_a I/O lines and directs control to the
byte write routine.

PWRUP
  movlw b'00000000'
  tris dif goto port_a
  port_a
  WRBYTE

;************************************************************************************************************

Byte Write Routine
This routine writes the data in "datao" to
8 consecutive bytes in the serial EE device starting
at address 00. This routine waits 100ns after every
byte to give the device time to do the write. This
program repeats forever.

WRBYTE

  clrf port_a ; clear all LEDs
  movlw b'10100000' ; set slave address and write mode
  movwf slave
  movlw b'01010101' ; set data to write as 55h
  movwf datao
  movlw .8 ; set number of bytes
  movwf bcount ; to write as to 8
  clrf addr1 ; set high address byte to 00
  clrf addr ; set low address byte to 00
  byte call BSTART ; generate start bit
  movf slave,w ; move slave address
  movwf txbuf ; into transmit buffer
  call TX ; and send it
  movf addr1,w ; move word 1 address
Using the 24C65 and 24C32

```assembly
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; move word 0 address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf datao,w ; move data byte
movwf txbuf ; to transmit buffer
call TX ; and transmit it
call BSTOP ; generate stop bit

movlw .10
movwf loops ; set delay time to give
call WAIT ; 10 ms wait after every byte
incf addr ; add 1 to low address counter
decfsz bcount ; all 8 bytes written?
goto byte ; no, do another byte
               
goto wrbyte ; start over
               
END
```
Using the 24C65 and 24C32

LIST P=16C54

; 64K Byte Write With Data Polling Program (133 bytes)
;
; This program demonstrates how to interface a
; Microchip PIC16C54 to a 24LC65 Serial E2 device.
; This program performs byte write operation on 8
; consecutive addresses using the data polling method
; to determine when the write cycle is complete.
;
; When writing to a serial E2 device, there are 2
; ways to handle the internal timed write cycle
; time of the device. The simplest method is
; simply to wait until the maximum cycle time
; is exceeded before attempting another command.
; The other, more efficient method is known as "data
; polling." Data polling is done by sending a start
; bit and control byte to the part after the write
; cycle has been initiated by a stop bit. If the ack bit
; is low, then the device is through writing, otherwise
; the sequence is repeated. If no low acknowledge
; is found within 40 attempts (about 10 milliseconds)
; then the routine times out and sets the timeout
; LED (pin 1) high.
;
; As an option, the user can connect a LED to pin 1
; on the PIC16CXX (use about a 1K resistor in series) as a
; timeout indicator. This LED will come on if the
; data polling is unsuccessful and the device being
; programmed does not respond. This can be tested by
; simply running the program with no part in the socket.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4MHz crystal. Clock speeds to the serial EE
; will be approximately 40 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX     Serial EE
; -----       -------
; Pin 12 (RB6) -> SCLK
; Pin 13 (RB7) -> SDATA
; Pin 1 (RA2)   -> Write cycle timeout fail LED (optional)

; Register Definitions

port_a equ Sh ; port 5 (port_a) used LEDs
port_b equ 6h ; port 6 (port_b) used for data and
; clock lines
eeprom equ 0ah ; eeprom
b cynct equ 0bh ; byte counter for read mode
addr equ 0ch ; address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxxx)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
b count equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
pollcnt equ 17h ; data polling counter
addr1 equ 18h ; word 1 address counter
;
; Bit Definitions

; Register Definitions

d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
Using the 24C65 and 24C32

sdata equ 7 ; serial EE data line (port_b, pin 13)
ackf equ 1 ; acknowledge fail LED, port_a (pin 18)
sclk equ 6 ; serial EE clock line (port_b, pin 12)
timeout equ 2 ; write cycle timeout fail LED, port_a (pin 1)

;*******************************
org 0ffh ; set reset vector
goto PWRUP
org 000h

;*******************************
DELAY ROUTINE
This routine takes the value in 'loops'
and multiplies it times 1 millisecond
to determine delay time.

WAIT

; timing adjustment variable
; sit and wait
; inner loops complete?
; no, go again
; outer loops complete?
; no, go again
; yes, return from sub

;*******************************
Start Bit Subroutine
this routine generates a start bit
(Low going data line while clock is high)

;*******************************
BSTART

bsf port_b, sdata ; make sure data is high
movlw b'00111111'
tris port_b
bsf port_b, sclk

bsf port_b, sclk ; set clock high

bcf port_b, sdata ; data line goes low during

; high clock for start bit

nop
nop
nop

; timing adjustment

bcf port_b, sclk ; start clock train

nop

retlw 0

;*******************************
Stop Bit Subroutine
This routine generates a stop bit
(High going data line while clock is high)

;*******************************
Using the 24C65 and 24C32

BSSTOP
    movlw b'00111111'  ; set data/clock lines as outputs
    tris port_b       ; make sure data line is low
    nop               
    nop               
    bsf port_b,sdata ; set data high
    nop               
    nop               
    bsf port_b,sclk  ; set clock high
    nop               
    nop               
    bsf port_b,sdata ; data goes high while clock high
    nop               
    bcf port_b,sdata ; data goes high while clock high
    ; for stopbit
    nop               
    nop               
    bcf port_b,sclk  ; set clock low again
    nop               
    bsf port_b,sdata ; set clock low again
    ;
    retlw 0
    ;

; ****************************************************************

BITOUT routine takes one bit of data in 'do' and
transmits it to the serial EE device
; ****************************************************************

BITOUT
    movlw b'00111111'  ; set data, clock as outputs
    tris port_b       ; check for state of data bit to xmit
    btfss eeprom,do   ; bitlow
    goto bitlow       ;
    bsf port_b,sdata ; set data line high
    goto clkout       ; go toggle the clock
    ;
    bitlow bcf port_b,sdata ; output a low bit
    clkout bsf port_b,sclk ; set clock line high
    nop               
    nop               
    nop               
    bcf port_b,sclk  ; return clock line low
    retlw 0
    ;
    ; End of Subroutine
    ;

; ****************************************************************

BITIN routine reads one bit of data from the
serial EE device and stores it in 'di'
; ****************************************************************

BITIN
    bsf eeprom,di     ; assume input bit is high
    movlw b'10111111' ; make sdata an input line
    tris port_b       ;
    bsf port_b,sdata ; set sdata line for input
    bsf port_b,sclk  ; set clock line high
    ; just sit here a sec
    nop               
    nop               
    bcf port_b,sdata ; read the data bit
    bcf eeprom,di     ; input bit was low
    bcf port_b,sclk  ; set clock line low
    ;
    retlw 0
    ;

; ****************************************************************

Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'datao' register and transmits it to the serial EE device.

Using the 24C65 and 24C32

It will then send 1 more clock to the serial EE for the
acknowledge bit. If the ack bit from the part was low
then the transmission was successful. If it is high, then
the device did not send a proper ack bit and the ack
fail LED will be turned on.

TX
movlw .8
movwf count ; set the #bits to 8

TXLP
bcf eeprom, do ; assume bit out is low
btfsc txbuf, 7 ; is bit out really low?
bsf eeprom, do ; otherwise data bit =1
movw BITOUT ; serial data out
movw txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no - go again
movw BITIN ; read ack bit

retlw 0

Power up routine
This is the program entry point, which in this case simply
sets the port_a I/O lines and directs control to the
write routine.

PWRUP
movlw b'00000000'
clrE port_a
movlw b'10101010' ; set data to write as AAh
movwf datao
movlw .8 ; set number of bytes
movwf bcount ; to write as 8
movwf addr1 ; set starting high address to 00
movf addr ; set starting low address to 00

byte call BSTART ; generate start bit
movf slave,w ; move slave address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr1,w ; move word 1 address
Using the 24C65 and 24C32

```assembly
; into transmit buffer
movwf txbuf
; and send it
call TX
; move word 0 address
movf addr,w
; into transmit buffer
movwf txbuf
; and send it
movf datao,w
movwf txbuf
; to transmit buffer
call TX
; generate stop bit
; now start polling for a low ack bit
; set max number of times to poll as 40
pill movlw .40
; generate start bit
; move slave address (write mode)
movlw b'10100000'
movwf txbuf
; and send it
btfss eeprom,d.i
; was the ack bit low?
goto exitpoll
; yes, do another byte
decfsz pillmt
; is poll counter down to zero?
goto poll
; no, poll again. Otherwise the part is
bsf port_ a, timeout
; not responding in time so set timeout
; LED and continue on

exitpoll incf addr
; add 1 to address counter
decfsz bcount
; all 8 bytes written?
goto byte
; no, do another byte
goto WRBYTE
; yes, start over

END```

Using the 24C65 and 24C32

LIST P=16C54
;******************************************************************************
; | 64K Page Write Program (126 bytes)
; |
; | The 24LC65 has a page length of 8 bytes. This page
; | can be used to write up to 8 bytes of data into the
; | part before initiating the write cycle. Since the
; | write cycle is timed the same for 1 byte or 8 bytes
; | it is more efficient to use the page mode when
; | consecutive addresses are being written to.
; |
; | When using page mode, the control byte, upper and lower
; | addresses are sent for the first address only. After the
; | data byte for the first address is sent, the data for the
; | next consecutive address is clocked in. This is
; | repeated as many times as needed (as long as the page
; | length is not exceeded) and then a stop bit is sent.
; | The device will still acknowledge between every byte of
; | data. After the stop bit is sent, the part will
; | initiate the self timed write cycle.
; |
; | This routine waits approximately 10ms for the write
; | cycle to complete for each page. A more efficient
; | method of determining when the write cycle is complete
; | is called "data polling." The data polling method is
; | explained in the program "64kdpoll.asm." That
; | particular program uses data polling in a byte write
; | mode but it can be used in exactly the same way for
; | a page mode write.
; |
; | As an option, the user can connect a LED to pin 18
; | on the PIC16CXX (use about a 1K resistor in series) as a
; | acknowledge fail indicator. This LED will come on
; | if the device being programmed does not send a low
; | acknowledge bit at the proper times. This can be
; | tested by simply running the program with no part
; | in the socket.
; |
; | Timing is based on using the PIC16CXX in 'XT' mode
; | using a 4Mhz crystal. Clock speeds to the serial EE
; | will be approximately 40 Khz for this setup.
; |
; |
; | PIC16CXX to Serial EE Connections:
; |
; | PIC16CXX          Serial EE
; | P12 (RB6)         SCLK
; | P13 (RB7)         SDATA
; |
; | PIN 18 (RA1)      Acknowledge fail LED (Optional)
; |
; |******************************************************************************

; Register Definitions
;******************************************************************************

port_a equ 5h        ; port 5 (port_a) used for LEDs
port_b equ 6h        ; port 6 (port_b) used for data and
                     ; clock lines
eeprom equ 0ah        ; bit buffer
bycnt equ 0bh         ; byte counter for read mode
addr equ 0ch          ; word 0 address counter
datai equ 0dh          ; data input register
datao equ 0eh          ; data output register
slave equ 0fh          ; device address (1010xxx0)
tbuf equ 10h           ; transmit buffer
count equ 13h          ; bit counter
bcount equ 12h         ; byte counter
loops equ 15h          ; delay loop counter

© 1993 Microchip Technology Inc.
Using the 24C65 and 24C32

loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter

; Bit Definitions

\( \text{d} \) equ 7 ; eeprom input bit
\( \text{do} \) equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port b, pin 13)
sclk equ 6 ; serial EE clock line (port b, pin 12)
ackf equ 1 ; acknowledge fail LED (port a, pin 18)

; org 01ffh ; set reset vector
goto PWRUP
org 000h
goto PWRUP

; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.

!=-

WAIT

\( \text{top} \) movlw .110
\( \text{top}2 \) movwf loops2
\( \text{top}2 \) nop ; sit and wait
\( \text{top} \) nop
\( \text{top} \) nop
\( \text{top} \) nop
\( \text{top} \) nop
\( \text{top} \) decfsz loops2 ; inner loops complete?
goto top2 ; no, go again
\( \text{top} \) decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub

; Start Bit Subroutine
; this routine generates a start bit
; (low-going data line while clock is high)

!=

BSTART
\( \text{bsf} \) port b, sdata ; make sure data is high
\( \text{movlw} \) b'00111111'
\( \text{tris} \) port b ; set data and clock lines for output
\( \text{bcf} \) port b, sclk ; make sure clock is low
\( \text{bsf} \) port b, sclk ; set clock high
\( \text{bcf} \) port b, sdata ; data line goes low during
\( \text{bcf} \) port b, sclk ; high clock for start bit
\( \text{bcf} \) port b, sclk ; timing adjustment
\( \text{bcf} \) port b, sclk ; start clock train
\( \text{retlw} \) 0

© 1993 Microchip Technology Inc.
Using the 24C65 and 24C32

;**************************************************************************
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;**************************************************************************

BSSTOP
movlw b'00111111'
tris port_b ; set data/clock lines as outputs
bcf port_b, sdata ; make sure data line is low
nop
nop
nop
bcf port_b, sclk ; set clock high
nop
nop
nop
bcf port_b, sdata ; data goes high while clock high
nop
nop
bcf port_b, sclk ; set clock low again
nop
nop
nop
retlw 0 ;

;**************************************************************************
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;**************************************************************************

BITOUT
movlw b'00111111'
tris port_b ; set data, clock as outputs
tfss eeprom, do ; check for state of data bit to xmit
goto bitlow ;
bcf port_b, sdata ; set data line high
bcf port_b, sclk ; set clock line high

bitlow bcf port_b, sdata ; output a low bit
goto clkout ; go toggle the clock

clkout bcf port_b, sclk ; set clock line high
nop
nop
nop
bcf port_b, sclk ; return clock line low
retlw 0 ;

;**************************************************************************
; BITIN routine reads one bit of data from the
; serial EE device and stores it in 'di'
;**************************************************************************

BITIN
bfs eeprom, di ; assume input bit is high
movlw b'10111111' ; make sdata an input line
tris port_b
bcf port_b, sdata ; set sdata line for input
bcf port_b, sclk ; set clock line high
nop
nop
nop
nop
bfs port_b, sdata ; read the data bit
bcf eeprom, di ; input bit was low
bcf port_b, sclk ; set clock line low
retlw 0 ;
Using the 24C65 and 24C32

;****************************************************************
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'datao' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;****************************************************************

TX
movlw .8
movwf count ; set the #bits to 8

TXLP
bcf eeprom,do ; assume bit out is low
btfsc txbuf, 7 ; is bit out really low?
bsf eeprom,do ; otherwise data bit =1

aall
BITOUT ; serial data out
s# txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no - go again

aall
BITIN ; read ack bit
btfsc eeprom,di ; check ack bit
bsf port_a, ackf ; set acknowledge fail LED if the

retlw 0

;****************************************************************
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_A I/O lines and directs control to the
; byte write routine.
;****************************************************************

PWRUP
movlw b'00000000'
tris port_a ; set port A as all output
drf port_a ; all output lines low
goto pwrite ; go do the page write

;****************************************************************
; Page Write Routine
; This routine writes the data in "datao" to 8 consecutive
; bytes using page write mode starting at address 00.
; This routine waits 10ms after every page to give the device
; time to do the write. This program repeats forever.
;****************************************************************
pwrite

drf port_a ; clear all LEDs
movlw b'10100000' ; set slave address and write mode
movwf slave
movlw b'01010101' ; set data to write as 55h
movwf datao

movlw .8 ; set number of bytes
movwf bcount ; to write as to 8

aall addr1 ; set high address byte to 00
aall addr ; set low address byte to 00

call BSTART ; generate start bit
movf slave,w ; move slave address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr1,w ; move word 1 address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; move word 0 address
movwf txbuf ; into transmit buffer
Using the 24C65 and 24C32

```
call TX ; and send it
byte movf datao,w ; move data byte
      movwf txbuf ; to transmit buffer
call TX ; and transmit it
decfsz bcount ; all 8 bytes written?
goto byte ; no, do another byte
      call BSSTOP ; generate stop bit
      movlw .10 ; set delay time to give
      movwf loops ; 10 ms wait after every byte
      goto pwrite ; start over
; ;
END
```
Using the 24C65 and 24C32

Listing P=16C54

;**************************************************************
; 64K Sequential Read Program (142 bytes)
;**************************************************************

; This program demonstrates how to interface a
; Microchip PIC16C54 to a 24LC65 Serial E2 device
; and perform a sequential read operation. This program
; will read 6 consecutive addresses in the 'sequential
; read' mode. This entails setting the address pointer
; for the first address only and then clocking out as many
; bytes of data as needed. The device will automatically
; increment the address.

; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4MHz crystal. Clock speeds to the serial EE
; will be approximately 60KHz for this setup.

; As an option, the user may connect a LED to pin 18
; on the PIC16CXX (use about a 1K resistor in series) as an
; acknowledge fail indicator. This LED will come on if
; the serial EE fails to acknowledge correctly.

; PIC16CXX to Serial EE Connections:
; PIC16CXX Serial EE
; Pin 12 (RB6) -> SCLK
; Pin 13 (RB7) -> SDATA
; PIN 18 (RA1) -> Acknowledge fail LED (Optional)

;**************************************************************

Register Definitions

port_a equ $h ; port 5 (port a) used for LED display
port_b equ $6h ; port 6 (port b) used for data and
; clock lines
eprom equ $0ah ; bit buffer
bcount equ $0bh ; byte counter for read mode
addr equ $0ch ; word 0 address counter
datai equ $0dh ; data input register
datao equ $0eh ; data output register
slave equ $0fh ; device address 1010xxxx0
txbuf equ $10h ; transmit buffer
count equ $11h ; bit counter
bcount equ $12h ; byte counter
loops equ $15h ; delay loop counter
loops2 equ $16h ; delay loop counter
addr1 equ $17h ; word 1 address counter

Bit Definitions

d equ $7 ; eeprom input bit
do equ $6 ; eeprom output bit
sdata equ $7 ; serial EE data line (port b, pin 13)
sclk equ $6 ; serial EE clock line (port b, pin 12)
ackf equ $1 ; acknowledge fail LED (port a)

org 01ffh ; set reset vector
goto PWRUP
org 000h ;
goto PWRUP

;**************************************************************

DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
Using the 24C65 and 24C32

;*******************************************************************************
WAIT
;
top  movlw .110 ; timing adjustment variable
      movwf loops2
      nop
      nop
      nop
      nop
      nop
      decfsz loops2 ; inner loops complete?
      goto top2 ; no, go again
      decfsz loops ; outer loops complete?
      goto top ; no, go again
      -lw 0 ; yes, return from sub

;*******************************************************************************
  Start Bit Subroutine
  this routine generates a start bit
  (Low going data line while clock is high)
;*******************************************************************************

BSTART
  bcf    port_b, sdata    ; make sure data is high
  movlw  b'00111111'
  tris   port_b           ; set data and clock lines for output
  bcf    port_b, sclk     ; make sure clock is low
  nop
  bcf    port_b, sclk     ; set clock high
  nop
  nop
  nop
  nop
  bcf    port_b, sdata    ; data line goes low during
          ; high clock for start bit
  nop
  nop
  nop
  nop
  nop
  ; timing adjustment
  bcf    port_b, sclk     ; start clock train
  nop
  nop
  retlw  0
          ;

;*******************************************************************************
  Stop Bit Subroutine
  This routine generates a stop bit
  (High going data line while clock is high)
;*******************************************************************************

BSTOP
  bcf    port_b, sdata    ; make sure data line is low
  movlw  b'00111111'
  tris   port_b           ; set data/clock lines as outputs
  bcf    port_b, sdata    ; make sure data line is low
  nop
  nop
  nop
  bcf    port_b, sclk     ; set clock high
  nop
  nop
Using the 24C65 and 24C32

BITOUT routine takes the bit of data in ‘do’ and
transmits it to the serial EE device.

BITOUT

movlw b'00111111'
bsf port_b
btfse eeprom,do
goto bitlow
bsf port_b,sdata
go clout
bsf port_b,sdata
bsf port_b,sclk
retlw 0

BITIN routine reads one bit of data from the
serial EE device and stores it in the bit ‘di’.

BITIN
bsf eeprom,di
movlw b'10111111'
bsf port_b
bsf port_b,sdata
goto bitlow
bsf port_b,sdata
tfss bcflbcf
retlw 0

Transmit Data Subroutine
This routine takes the byte of data stored in the
‘datao’ register and transmits it to the serial EE device.
It will then send 1 more clock to the serial EE for the
acknowledge bit. If the ack bit from the part was low
then the transmission was successful. If it is high, then
the device did not send a proper ack bit and the ack
fail LED will be turned on.

TX
movlw .8
movwf count

TXLP
bsf eeprom,do

DS00558A-page 28 © 1993 Microchip Technology Inc.
Using the 24C65 and 24C32

```assembly
btfsc txbuf, 7 ; is bit out really low?
bsf eeprom, do ; no, set it high
call BITOUT ; send the bit to serial EE
bff txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no, go again
call BITIN ; read ack bit
btfsc eeprom, di ; check ack bit
bsf port_a, ackf ; set acknowledge fail LED if the
device did not pull data low
retlw 0

;******************************************************************************
; Receive data routine
; This routine reads one byte of data from the part
; into the 'datai' register. It then sends a high
; ack bit to indicate that no more data is to be read.
;******************************************************************************
RX
clr datai ; clear input buffer
movlw 8 ; set # bits to 8
movwf count
RXLP
bff datai ; rotate datai 1 bit left
call BITIN ; read a bit
btfsc eeprom, di
bsf datai, 0 ; set bit 0 if necessary
decfsz count ; 8 bits done?
goto RXLP ; no, do another
retlw 0

;******************************************************************************
; Power up routine
; This is the program entry point. I/O line status is
; set for port A here.
;******************************************************************************
PWRUP
movlw b'00000000' ; set port A as all output
bcf port_a, ackf ; clear the ack fail LED if on
movlw 8 ; set number of bytes to read as 8
movwf bcount
movlw b'10100000' ; set slave address and write mode
movwf slave
clrf addr1 ; set starting high address to 00
clf addr ; set starting low address to 00
; call BSTART ; generate start bit
movf slave, w ; get slave address
movwf txbuf ; into transmit buffer
```

Receive data routine
- This routine reads one byte of data from the part into the 'datai' register. It then sends a high ack bit to indicate that no more data is to be read.

Power up routine
- This is the program entry point. I/O line status is set for port A here.

READ (read routine)
- This routine reads 8 consecutive addresses of the serial EE device starting at address 00 in the sequential read mode. Reading in this mode is more efficient than the random read mode as the control byte and address have to be sent only once at the beginning of the sequence. As many consecutive addresses as needed can then be read from the part until a stop bit is sent. In the read mode, the PIC16CXX must send the acknowledge bit after every 8 data bits from the device. When the last byte needed has been read, then the controller will send a high acknowledge bit and then a stop bit to halt transmission from the device.
Using the 24C65 and 24C32

call TX ; and send it
movf addr1,w ; get word 1 address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; get word 0 address
movwf txbuf ; into transmit buffer
call TX ; and send it
call BSTART ; generate start bit
movlw b'10100001' ; get slave address and read mode
movwf txbuf ; into transmit buffer
call TX ; and transmit it

rbyte call RX ; read 1 byte from device
decfsz bcount ; are all 8 bytes read?
goto lowack ; no, send low ack and do another
btf eeprom,do ; yes, send high ack bit
call BITOUT ; to stop transmission
call BSTOP ; and send a stop bit
movlw .255 ; long delay for scope
movwf loops ; trigger purposes only
call wait ;
goto READ ; start all over

lowack bcf eeprom,do ; send low ack bit
call BITOUT ; to continue transmission
goto rbyte ; and read another byte

END
Using the 24C65 and 24C32

LIST P=16C54
;************************************************************************************
; Program to set the high endurance block on the 64K device (24C65). (122 bytes)
; The 24LC65 is a 64K device, 4K of which is error corrected to increase the endurance of that portion of the array. The location of this high endurance 'block' can be set by the user. When the device comes from the factory, the high endurance block will be set as the uppermost block in the array.
; This program sets the high endurance block as the first block in the array (address range 00h to 1FFh).
; The high endurance block is set by sending the following sequence to the device:
; SB 10100000 lXXAAAAX XXXXXXXX OOXXOOOO STB
; Where SB=start bit
; 1=data high
; 0=data low
; X=don't care
; AAA=4 bit high endurance block address
; STB=stop bit
; As an option, the user may connect a LED to pin 18 on the PIC16CXX (use about a 1K resistor in series) as an acknowledge fail indicator. This LED will come on if the serial EE fails to acknowledge correctly.
; Timing is based on using the PIC16CXX in 'XT' mode using a 4MHz crystal. Clock speeds to the serial EE will be approximately 60Khz for this setup.
; PIC16CXX to Serial EE Connections:
; PIC16CXX Serial EE
; Pin 12 (RB6) -> SCLK
; Pin 13 (RB7) -> SDATA
; PIN 18 (RA1) -> Acknowledge fail LED (Optional);
;************************************************************************************
; Register Definitions
;************************************************************************************
port_a equ Sh ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port b) used for data and
; clock lines
eprom  equ 0ah ; eeprom buffer
bycnt  equ 0bh ; byte counter for read mode
addr   equ 0ch ; word 0 address counter
datai  equ 0dh ; data input register
datao  equ 0eh ; data output register
slave  equ 0fh ; device address (1010xxx0)
txdifi  equ 10h ; transmit buffer
count  equ 11h ; bit counter
bcount  equ 12h ; byte counter
loops  equ 13h ; delay loop counter
loops2 equ 16h ; delay loop counter
addr1  equ 17h ; word 1 address counter
he_blk equ 18h ; high endurance block address
;************************************************************************************
; Bit Definitions
;************************************************************************************
d  equ 7 ; eeprom input bit
do  equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b, pin 13)
Using the 24C65 and 24C32

sclk equ 6 ; serial EE clock line (port_b, pin 12)
ackf equ 1 ; acknowledge fail LED (port_a, pin 18)

;******************************************************************************************

org 0ffh ; set reset vector
goto PWRUP
org 000h

goto PWRUP

;******************************************************************************************

DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;******************************************************************************************

WAIT
;
top movlw .ll0
movwf loops

nop ; sit and wait

nop

nop

nop

nop

nop

decfsz loops2 ; inner loops complete?
goto top2 ; no, go again

;

decfsz loops ; outer loops complete?
goto top ; no, go again

retlw 0 ; yes, return from sub

;******************************************************************************************

Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;******************************************************************************************

BSTART

bsf port_b, sdata ; make sure data is high

movlw b'00111111'
tms port_b ; set data and clock lines for output

bcf port_b, sclk ; make sure clock is low

nop

bsf port_b, sclk ; set clock high

nop

nop

nop

nop

nop

bcf port_b, sdata ; data line goes low during
; high clock for start bit

nop

nop

nop

nop

nop

nop

nop

nop

retlw 0

;******************************************************************************************

Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;******************************************************************************************

BSTOP

movlw b'00111111';
tms port_b ; set data-clock lines as outputs
Using the 24C65 and 24C32

BITOUT routine takes one bit of data in ‘do’ and
transmits it to the serial EE device

BITOUT

```assembly
bcf port_b,sdata ; make sure data line is low
nop
nop
nop
bsf port_b,sclk ; set clock high
nop
nop
nop
bsf port_b,sdata ; data goes high while clock high
; for stop bit
nop
nop
bcf port_b,sclk ; set clock low again
nop
nop
retlw 0
```

BITIN routine reads one bit of data from the
serial EE device and stores it in ‘di’:

BITIN

```assembly
bsf eeprom,di ; assume input bit is high
movlw b'00111111' ; set data,clock as outputs
tris
bsf
bsf
nop
nop
nop
nop
btfss
bcf
bcf
retlw 0
```

Transmit Data Subroutine

This routine takes the byte of data stored in the
datao’ register and transmits it to the serial EE device.
It will then send 1 more clock to the serial EE for the
acknowledge bit. If the ack bit from the part was low
then the transmission was successful. If it is high, then
the device did not send a proper ack bit and the ack
fail LED will be turned on.

TX
Using the 24C65 and 24C32

```
movlw .8
movwf count ; set the #bits to 8
;
TXLP
bcf eeprom,do ; assume bit out is low
btfsc txbuf,7 ; is bit out really low?
bcf eeprom,do ; otherwise data bit = 1
call BITOUT ; serial data out
rf txbuf ; rotate txbuf left
decfsz count ; 8 bits done?
goto TXLP ; no - go again
call BITIN ; read ack bit
btfsc eeprom,di ; check ack bit
bsf port_a,ackf ; set acknowledge fail LED if the

retlw 0
;
;****************************************************************
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; byte write routine.
;****************************************************************
PWRUP
movlw b'00000000'
tris port_a ; set port A as all output
df port_a ; all output lines low
goto setheblk ; set go set the high endurance block

;****************************************************************
; Set High Endurance Block Routine
; This routine sets the high endurance block to the first
; block in the array and then delays about a half second. This
; routine will repeat forever.
;****************************************************************
setheblk
;
df port_a ; clear all LEDs
movwf slave
movlw .0
movwf he_blk ; set the endurance block as first
; block in array (block 0)
rf he_blk ; rotate starting block left 1 bit
; to arrange it correctly
bsf he_blk,7 ; set msb bit in block address byte
call BSTART ; generate start bit
movlw b'10100000' ; get slave address and write mode
movwf txbuf ; into transmit buffer
call TX ; and send it
movf he_blk,w ; get the block address byte
movwf txbuf ; into transmit buffer
call TX ; and send it
df txbuf ; clear buffer
call TX ; send 8 don't care bits
df txbuf
call TX ; send config byte (all O's)
call BSTOP ; send stop bit

movlw .255 ; long delay
movwf loops
call wait
movlw .255
movwf loops
call wait

goto setheblk ; go again
;
END
```
LIST P=16C54

************** CAUTIONS TO THE USER !!!! **************

1) THE SECURITY OPTION CAN ONLY BE SET ONCE!!
2) THE HIGH ENDURANCE BLOCK CANNOT BE CHANGED
   AFTER THE SECURITY OPTION IS SET.
3) THE HIGH ENDURANCE BLOCK CANNOT BE PROTECTED.

The security option is set by sending the following
sequence to the device:

SB 10100000 lXXAAAAX XXXXXXXX lOXXNNNN STB

Where SB=start bit
1=data high
0=data low
X=don’t care
AAAA=4 bit number to indicate first secure block (0-15)
NNNN=4 bit number to indicate how many secure blocks (1-15)
STB=stop bit

As an option, the user may connect a LED to pin 18
on the PIC16CXX (use about a 1K resistor in series) as an
acknowledge fail indicator. This LED will come on if
the serial EE fails to acknowledge correctly.

Timing is based on using the PIC16CXX in ‘XT’ mode
using a 4Mhz crystal. Clock speeds to the serial EE
will be approximately 60 Khz for this setup.

PIC16CXX to Serial EE Connections:

PIC16CXX Serial EE
------- -------
Pin 12 (RB6) -> SCLK
Pin 13 (RB7) -> SDATA
PIN 18 (RA1) -> Acknowledge fail LED (Optional);

RegisterDefinitions

;***************************************************************************
port_a equ Sh
; port 5 (port_a) used for LEDs
port_b equ 6h
; port 6 (port_b) used for data and
; clock lines
eeprom equ 0ah
; bit buffer
bycnt equ 0bh
; byte counter for read mode
addr equ 0ch
; word 0 address counter
data1 equ 0dh
; data input register
data0 equ 0eh
; data output register
slave equ 0fh
; device address (1010xxxx)
txbuf equ 10h
; transmit buffer
count equ 11h
; bit counter
bcnt equ 12h
; byte counter
loops equ 15h
; delay loop counter
loops2 equ 16h
; delay loop counter
adrr1 equ 17h
; word 1 address counter
strt_blk equ 18h
; starting block number for secure portion
sec_blks equ 19h
; number of secure blocks
Using the 24C65 and 24C32

Bit Definitions

d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b,pin13)
sclk equ 6 ; serial EE clock line (port_b,pin12)
ackf equ 1 ; acknowledge fail LED (port_a,pin18)

org 01ffh ; set reset vector
goto PWRUP
org 000h ;
goto PWRUP

; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.

WAIT
; top movlw 110 ; timing adjustment variable
movwf loops2
top2 nop ; sit and wait
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again

; decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub

; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)

; BSTART
bsf port_b,sdata ; make sure data is high
movlw b'00111111'
tisz
bsf port_b
bsf port_b,sclk ; make sure clock is low
nop
bsf port_b,sclk ; set clock high
nop
nop
nop
nop
nop
bsf port_b,sdata ; data line goes low during
; high clock for start bit
nop
nop
nop
nop
nop
nop
nop
nop
nop
bcf port_b,sclk ; start clock train
nop
nop
nop
nop
retlw 0
Using the 24C65 and 24C32

;**********************************************************************
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;**********************************************************************

BSSTOP
movlw b'00111111'
tris port_b ; set data/clock lines as outputs
bcf port_b, sdata ; make sure data line is low
nop
nop
nop

bsf port_b, sclk ; set clock high
nop
nop

bsf port_b, sdata ; data goes high while clock high
for stopbit

nop
nop

bsf port_b, sclk ; set clock low again
nop
nop

retlw 0

;**********************************************************************
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;**********************************************************************

BITOUT
movlw b'00111111'
tris port_b
btfss eeprom, do ; check for state of data bit to transmit
goto bitlow
bf if port_b, sdata ; set data line high
goto clkout ; go toggle the clock

bitlow bcf port_b, sdata ; output a low bit

clkout bcf port_b, sclk ; set clock line high
nop
nop

bcf port_b, sclk ; return clock line low
retlw 0

;**********************************************************************
; BITIN routine reads one bit of data from the
; serial EE device and stores it in 'di'
;**********************************************************************

BITIN
bsf eeprom, di ; assume input bit is high
movlw b'10111111' ; make sdata an input line
tris port_b
bsf port_b, sdata ; set sdata line for input
bsf port_b, sclk ; set clock line high
nop
nop
nop

nop

btfss port_b, sdata ; read the data bit
bcf eeprom, di ; input bit was low
bcf port_b, sclk ; set clock line low

retlw 0


Using the 24C65 and 24C32

;******************************************************************************
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'dataO' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;******************************************************************************

TX

   movlw .8
   movwf count ; set the #bits to 8
   ;
   TXLP

   bcf eeprom, do ; assume bit out is low
   btfsc txbuf, 7 ; is bit out really low?
   bsf eeprom, do ; otherwise data bit =1
   call BITOUT ; serial data out
   # txbuf ; rotate txbuf left
   decfsz count ; 8 bits done?
   goto TXLP ; no - go again
   call BITIN ; read ack bit
   btfsc eeprom, di ; check ack bit
   bsf port_a, ackf ; set acknowledge fail LED if the
   retlw 0
   ;******************************************************************************

; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; byte write routine.
;******************************************************************************

PWRUP

   movlw b'00000000'
   tris port_a ; set port A as all output
   clr port_a ; all output lines low
   goto set_sec ; set go set the high endurance block
   ;******************************************************************************

; Set Security Routine
; This routine sets the secure portion of the array to the
; lower 4 blocks (2K bytes) in the array and then delays
; about a half second. This routine will repeat forever.
;******************************************************************************

set_sec

   clr port_a ; clear all LEDs
   movlw .0
   movwf strt_blk ; set the first protected block as
                    ; block 0
   movlw .4
   movwf sec_blks ; set the number of protected blocks
                   ; as 4 (2048 bytes)
   # strt_blk ; rotate starting block left 1 bit
   bsf strt_blk,7 ; to arrange it correctly
   bsf sec_blks,7 ; set msb bit in starting block address
   call BSTART ; set msb block count byte
   movlw b'10100000' ; get slave address and write mode
   movwf txbuf ; into transmit buffer
   call TX ; and send it
   movf strt_blk, w ; get the starting block address
   movwf txbuf ; into transmit buffer
   call TX ; and send it
   clr txbuf ; clear buffer
call TX ; send 8 don't care bits
movf sec_blks,w ; get secure blocks byte
movwf txbuf ; into transmit buffer
call TX ; and send it
call BSTOP ; send stop bit

movlw .255 ; long delay
movwf loops
call wait
movlw .255
movwf loops
call wait

goto set_sec ; go again

END
INTRODUCTION
The 24C01 is a 1K (128 x 8) serial EEPROM which is currently offered by Microchip and Xicor. There are several important differences between the two devices which are discussed in this report. This report refers to the Microchip part as the 24C01A and the Xicor part as the X24C01. It is intended to assist in designing a memory subsystem which is compatible with either device.

COMPATIBILITY ISSUES
There are three major differences between Microchip's 24C01A and Xicor's X24C01 as detailed below.

2.1 PAGE MODE DIFFERENCES
The 24C01A was originally designed to work in the same socket as the PCD8572 which has a two-byte page mode. Therefore, its page buffer is two bytes deep. The X24C01 has a page mode of four bytes depth.

If more than two bytes are transmitted to the 24C01A during a page programming cycle, the 24C01A will terminate the write cycle.

In many applications where serial EEPROMs are used and speed is not a key issue, the byte write mode can be used without any loss of system performance. If only the byte write mode is used, there is no compatibility problem (other than the slave address software differences discussed in 2.2).

If the page write feature must be used, two different page mode algorithms can be transmitted by the master depending upon whose device is being used. The master will have to first do a polling routine to determine if it is interfacing with a 24C01A or X24C01. This polling technique is discussed in 2.2.

Interestingly, the 24C01A actually updates faster in the page mode even though it has one-half the page depth of the X24C01. This is due to the faster write cycle time of the 24C01A. The two devices are compared in Figure 1.

2.2 SOFTWARE DIFFERENCES
Microchip's 24C01A is designed to share a 2-wire bus on which it resides with other devices. To support this, the first byte of each command sequence from the master to the 24C01A must be a slave address. The 24C01A monitors the 2-wire bus for its slave address and "wake-up" from standby mode if the address transmitted matches its address as defined by the voltage level (Vss or Vcc) on pins 1, 2 and 3. X24C01 does not support a multiple device bus and will always "wake-up" if a start condition is detected.

A slave address must be transmitted to the 24C01A at certain points during reading and writing. This slave address is not required by the X24C01. Transmitting a slave address to X24C01 will result in erroneous operation. This problem can be solved by having the master transmit the proper serial bit pattern to the slave, but first the master has to ascertain with which 24C01 it is communicating.

The master can do a simple polling routine before beginning serial communication with 24C01A or X24C01 to determine with which device it is working. The proper serial protocol for both devices must be contained in the master controller's firmware. Once the master knows which 24C01 is on the bus, it can execute the proper serial commands.

<table>
<thead>
<tr>
<th></th>
<th>Microchip</th>
<th>Xicor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max byte program time</td>
<td>1 ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>Max page program time</td>
<td>2 ms (2 bytes)</td>
<td>10 ms (4 bytes)</td>
</tr>
<tr>
<td>Max time to program 4 bytes</td>
<td>4 ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>Max time to rewrite device</td>
<td>128 ms</td>
<td>320 ms</td>
</tr>
</tbody>
</table>
24C01A Compatibility Issue

The polling consists of the pattern like the one shown below:

SDA LINE: | START BIT | 00000001 | ACKNOWLEDGE BIT | DATA 7...0 |

If an X24C01 is used on the 2-wire bus, an acknowledge bit and eight data bits will be returned whereas 24C01A will issue no response and will ignore the command.

2.3 HARDWARE DIFFERENCES

Unlike the X24C01, the 24C01A is designed to share a 2-wire bus with other devices. Chip address bits are included in the slave address for the 24C01A, and are incorporated into pins 1, 2 and 3 of the device. They must be connected to Vcc or Vss for proper operation. Since pins 1, 2 and 3 of the Xicor part are NC (no connect) pins and they are not internally connected, they can be tied high or low.

Another hardware difference involves pin 7 which MUST be connected to Vss on the X24C01. The 24C01A can have pin 7 connected to Vss or Vcc.

If only one device is planned for the 2-wire bus, the board can be made compatible for either device by connecting pins 1, 2 and 3 to either Vss or Vcc and tying pin 7 to Vss.

Mobility For Memory Upgrade And Expansion

In system applications where the master device needs to address more than one serial EEPROM on a 2-wire bus, the Microchip 24C01A offers the flexibility. Up to eight 24C01A's can be connected to the 2-wire bus. More than one Xicor X24C01 connected to the bus may result in bus contention.

If memory upgrade is required, the Microchip 24C01A can be upgraded to the 24C02A (256 x 8) or the 24C04A (512 x 8) in the same IC socket with NO change in hardware. Using the Xicor X24C01, both software and hardware would have to be reconfigured to accomodate the changes.

CONTACT: Bruce Negley
Memory Products Division

© 1992 Microchip Technology Inc.
SERIAL EEPROM WRITE TIME REQUIREMENTS

Elements of the Write Cycle Time

The total write operation time for a Serial EEPROM is determined by three main elements:

- Number of bytes to load for each write operation
- Bus clock speed at which the write operation is loaded
- Fixed internal write cycle timer required for the programming operation

The load component of the write command consists of the control byte, address, and the data of up to 16 bytes. The time required to load the operation depends on the number of bytes to load at one time and the bus clock speed. After this load is complete, the part commences the internally controlled write cycle and the bus and system are free to perform other tasks. The internal write cycle timer is a fixed time delay which is required to program the EEPROM memory cells. Table 1 gives examples of total write time for 1 and 16 bytes for various parts at fast and normal clock speeds.

- **Load time (time at bus free)** is the time the part is being loaded with the instruction, address, and data. The bus is free after this time interval, and the part commences the internally controlled write cycle sequence.

**TABLE 1 - WRITE OPERATION TIME COMPARISON**

<table>
<thead>
<tr>
<th>Product</th>
<th>Page Width</th>
<th># Bytes to load</th>
<th>Speed (KHz)</th>
<th>Load time (ms)</th>
<th>Write timer (ms)</th>
<th>Write timer (ms)</th>
<th>Typical total write time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24C01</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>0.28</td>
<td>1</td>
<td>0.4</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>8 (4 x 2)</td>
<td>100</td>
<td>1.12</td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>24LC01</td>
<td>8</td>
<td>1</td>
<td>100</td>
<td>0.28</td>
<td>10</td>
<td>3</td>
<td>3.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.91</td>
<td></td>
<td>10</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.07</td>
<td></td>
<td>10</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.23</td>
<td></td>
<td>10</td>
<td>3</td>
<td>3.23</td>
</tr>
<tr>
<td>24C04</td>
<td>8</td>
<td>1</td>
<td>100</td>
<td>0.28</td>
<td>1</td>
<td>0.4</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.82</td>
<td></td>
<td>1</td>
<td>0.4</td>
<td>8.22</td>
</tr>
<tr>
<td></td>
<td>16 (2 x 8)</td>
<td>100</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>100</td>
<td>1.63</td>
<td></td>
<td>10</td>
<td>3</td>
<td>3.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>10</td>
<td>3</td>
<td>4.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.07</td>
<td></td>
<td>10</td>
<td>3</td>
<td>3.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.41</td>
<td></td>
<td>10</td>
<td>3</td>
<td>3.41</td>
</tr>
<tr>
<td>24LC04</td>
<td>16</td>
<td>1</td>
<td>100</td>
<td>0.28</td>
<td>10</td>
<td>3</td>
<td>3.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- **Write timer (worse case)** indicates the time the part is in the internally controlled write cycle allowing for the maximum specified datasheet requirements.
- **Write timer (typical)** indicates the time the part is in the internally controlled write cycle assuming nominal conditions and utilizing write cycle polling.
- **Total write** time is the combined load time and typical internal write cycle time.

MINIMIZING SERIAL BUS COMMUNICATION TIME IN A SYSTEM

Utilizing the Page Write Option

The original Microchip Serial EEPROM products, though utilizing a page buffer, only write bytes sequentially. This means the time required to write 8 bytes is 8 ms, worse case. The new 24LCXX products incorporate a page mode that allows simultaneous writes of up to 16 bytes. This allows the programming of up to 16 bytes in one write cycle (10 ms) compared to 16 write cycles (16 ms) for the original 24CXX products. Microchip uses an 8 byte page in the 24LC01 and 24LC02, and a 16 byte page the 24LC04 and higher densities. What this means to a system designer is a write of 8 bytes in a 24LC01 or 24LC02 would take 10 ms worse case versus 8 ms.
Minimizing Serial Bus Communication Time

worse case in either a 24C01 or 24C02. Loading the entire memory of an original 24C04 takes 512 ms, but the same operation on a new 24LC04 is reduced to only 320 ms, assuming worse case. As table 1 shows, the difference is even greater using typical numbers.

The 93LCXX products do not utilize a page mode. However, in 16 bit mode, all 16 bits are written simultaneously. The original 93CXX products write the 16 bits in 2 write cycles, so the write cycle time increase of the new products is only 5 X (3.5 X typical).

Utilizing Write Cycle Polling

One powerful method of increasing programming efficiency is by periodically polling the part to determine if the write cycle has completed. To poll the 24LCXX series products, a control byte is sent and the acknowledge bit from the part is read. If the part acknowledges (pulls SDA low), it is ready to accept a new command. The part will not acknowledge while in the internally timed write cycle.

To poll the 93LCXX series products, the chip select is pulled high after the write cycle commences, and the data out line is read for the ready/busy status. If the part is still busy, it will pull the data line low. When the internally timed write cycle is complete, the part will pull the data line high, indicating it is ready for a new command.

Serial EEPROM System Optimization

Serial EEPROMs are used in systems for two purposes: storing data and reading back data. Read operations are at full clock speed, so the only methods for optimization are to run the clock at maximum frequency and to utilize sequential read whenever possible. Sequential read allows a continuous output data stream on one command.

The write operation, with its internal write timer component, needs special consideration when designing the control software. Efficient operation can be accomplished using both the page mode and write cycle polling. The following example shows a typical fetch-store operation in a system and how optimization can be incorporated. In the example system, the microcontroller must fetch bytes of data from a sensor and send the bytes to a EEPROM for storage. Two cases are shown in Figure 2: case 1 uses a 24C01A with no optimization; case 2 uses a 24LC01B with the available page mode and write cycle polling.

By utilizing the available page write mode and by polling for the write cycle completion, nearly four times as many bytes can be initially loaded to the serial EEPROM in the same time interval. Continuous operation for the optimized case 2 takes only 0.49 ms per byte compared to 1.28 ms per byte for the non-optimized case 1.
## FIGURE 2 - CASE COMPARISON

<table>
<thead>
<tr>
<th>Operation</th>
<th>Serial Bus Time</th>
<th>Operation</th>
<th>Serial Bus Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch byte</td>
<td>0.28 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>load to serial</td>
<td></td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>fetch byte</td>
<td></td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>wait for write timer</td>
<td>1 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>load to serial</td>
<td>0.28 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>fetch byte</td>
<td></td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>wait for write timer</td>
<td>1 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>load to serial</td>
<td>0.28 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>fetch byte</td>
<td></td>
<td>load 8 bytes to serial</td>
<td>0.91 ms</td>
</tr>
<tr>
<td>wait for write timer</td>
<td>1 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>load to serial</td>
<td>0.28 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>fetch byte</td>
<td></td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>wait for write timer</td>
<td>1 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>load to serial</td>
<td>0.28 ms</td>
<td>fetch byte</td>
<td></td>
</tr>
<tr>
<td>fetch byte</td>
<td></td>
<td>poll for write timer</td>
<td>&lt;3 ms typical</td>
</tr>
<tr>
<td>wait for write timer</td>
<td>1 ms</td>
<td>load 8 bytes to serial</td>
<td>0.91 ms</td>
</tr>
<tr>
<td>load to serial</td>
<td>0.28 ms</td>
<td>load 8 bytes to serial</td>
<td>0.91 ms</td>
</tr>
</tbody>
</table>

First **8 bytes** loaded in 9.24 ms

First **16 bytes** loaded in 4.82 ms

**AUTHOR:** Lenny French  
Memory Products Division
INTRODUCTION

The Microchip Technology Inc. 93LC56/66 low-power 3-/4-wire non-volatile memories and are suitable for many embedded system code and data storage. These devices are easily interfaced to most microcontrollers in today's market place, but Microchip's 8-bit RISC series PIC16CXX offer the best code density of any microcontroller on the market today. Using the PIC16C54, the assembly programs contained in this application note have been fully tested and provide the correct timing and 3-wire sequences to fully operate the 93LC56/66 in a PIC16CXX-based embedded application. The PIC16C54 was clocked at a 10 MHz frequency. This application note is intended to provide the engineer with readily available stand alone code modules to accomplish all of the necessary functions to utilize these devices in a low power application using the efficient PIC16C54 microcontroller.

The 93 series of devices have essentially four I/O pins:

- CS Chip Select
- CLK Clock
- DI Data In
- DO Data Out

This series of devices use a series of commands to accomplish the normal memory functions. These are READ, WRITE, EWEN, ERASE, ERAL, WRAL, EWDS. For a more detailed discussion of the function of these devices reference the appropriate data sheet and AN536, also published by Microchip Technology.

The following programs are included in this application note and are fully functional stand alone modules.

3-wire Byte Read Program

- Start Bit Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

3-wire Byte Write Program

- Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine (EWEN)
- Byte Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Byte Write with Data Polling Program

- Data Polling Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine
- Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Sequential Read Program

- Delay Routine
- Start Bit Routine
- Bit In Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

AUTHOR: Bruce Negley
Memory Products Division
Using the 93LC56 and 93LC66

LIST P=16C54
;********************************************************************
; 3-Wire Byte Read Program (80 bytes)
; This program demonstrates how to interface a Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE device. This program will read 8 consecutive addresses in the 'random read' mode. This means that the opcode and address for each byte will be sent to the device.
; This program will repeat forever.
;
; Another, more efficient method of reading consecutive addresses is called the 'sequential read' mode. This involves sending the opcode and address for the first byte to read, then continuing to provide clocks for the next addresses. The device will automatically increment the address. An example of the segmental read mode is provided in the '3wirecr.asm' file.
;
; This program communicates to the serial EE in the x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN ON THE DEVICE TO Vcc.
;
; Timing is based on using the PIC16CXX in 'XT' mode using a 4Mhz crystal. Clock speeds to the serial EE will be approximately 50 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX Serial EE
; ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----elli
;**********************************************************************
; 3-Wire Byte Read Program (80 bytes)
; This program demonstrates how to interface a Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE device. This program will read 8 consecutive addresses in the 'random read' mode. This means that the opcode and address for each byte will be sent to the device.
; This program will repeat forever.
;
; Another, more efficient method of reading consecutive addresses is called the 'sequential read' mode. This involves sending the opcode and address for the first byte to read, then continuing to provide clocks for the next addresses. The device will automatically increment the address. An example of the segmental read mode is provided in the '3wirecr.asm' file.
;
; This program communicates to the serial EE in the x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN ON THE DEVICE TO Vcc.
;
; Timing is based on using the PIC16CXX in 'XT' mode using a 4Mhz crystal. Clock speeds to the serial EE will be approximately 50 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX Serial EE
; ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ****
Using the 93LC56 and 93LC66

;******************************************************************************
; Start Bit Subroutine
; this routine generates a start bit
; (Chip select and DI high when clock goes high)
;******************************************************************************

;******************************************************************************

;******************************************************************************

;******************************************************************************

;******************************************************************************

;******************************************************************************

;******************************************************************************
Using the 93LC56 and 93LC66

;******************************************************************************
; Transmit Data Subroutine
; This routine takes the byte of data stored in the 
datao' register and transmits it to the serial EE device.
;******************************************************************************
TX
movf  bits,w ; set the number of bits to xmit
movwf  count
;
TXLP
bcf   eeprom,do ; assume bit 7 is low
btfsc  txbuf, 7 ; is bit 7 clear?
bsf   eeprom,do ; no, set data bit =1
call   BITOUT ; transmit 1 bit to serial EE
sf   txbuf ; rotate txbuf left
decfsz  count ; all bits done?
goto   TXLP ; no, do another bit
retlw 0 ; yes, jump out
;
;******************************************************************************
; POWER-UP ROUTINE
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; read routine.
;******************************************************************************
PWRUP

movlw b'00000000'
tris  port_a ; set port_a as all output
cnf   port_a ; all lines low
movlw b'10000000'
tris  port_b ; set RB7 as input, rest output
;
; Fall through and do the read
;
;******************************************************************************
; READ ROUTINE
; This routine reads 8 consecutive addresses in 
; random mode starting at address 0. This is done in
; x16 mode and will repeat forever.
;******************************************************************************
READ

movlw .0 ; set starting address to 00
movwf  addr
movlw .8 ; set number of addresses to
movwf  bytcnt ; read as 8

rbyte  call   BSTART ; generate the start bit
movlw .2 ; set #bits to 2
movwf  bits
movlw b'10000000' ; get opcode (10b)
movwf  txbuf ; into output buffer
call   TX ; and transmit it
movlw .8
movwf  bits ; set number of bits to 8
movf   addr,w ; get the address
movwf  txbuf ; into the output buffer
call   TX ; and transmit it

all   RX ; read the high byte
movf   datai,w ; move input data to w
movwf  hbyte ; xfer it to high byte

all   RX ; read the low byte
movf   datai,w ; move input data to w
movwf  hbyte ; xfer it to low byte
Using the 93LC56 and 93LC66

```assembly
bcf port_b, chpsel ; clear the chip select line
incf addr ; add 1 to the address
decfsz bytcnt ; have all bytes been read?
goto rbyte ; no, read another byte
goto READ ; yes, start over
```

END
LIST P=16C54

;---------------------------------------------------------------------
; 3-Wire Byte Write Program (106 bytes)
;
; This program demonstrates how to interface a Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE device. This program will execute the erase/write enable command, write to 8 consecutive addresses, and then execute the erase/write disable command. This sequence will repeat forever.
;
; After each byte is written, time must be given to the device for it to complete the write cycle before the next command can be sent. The easiest solution is to consult the data book for the maximum write cycle time and just wait that long before the next command is sent. This program demonstrates that solution.
;
; Another, more efficient method of determining when the write cycle is complete is called 'data polling.' This method is demonstrated in the program "3wpdpoll."
;
; This program communicates to the serial EE in the x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN ON THE DEVICE TO Vcc.
;
; Timing is based on using the PIC16CXX in 'XT' mode using a 4Mhz crystal. Clock speeds to the serial EE will be approximately 40 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX Serial EE
; -------------------
; Pin 10 (RB4) -> Chip Select
; Pin 11 (RB5) -> Clock
; Pin 12 (RB6) -> Data In
; Pin 13 (RB7) -> Data Out
; ORG=Vcc
;
;******************************************************************************
;
; Register Assignments
;******************************************************************************

port_a equ Sh ; port 5 (port_a)
port_b equ 6h ; port 6 (port b) comm lines to serial EE
eeprom equ 0ah ; bit buffer
addr equ 0ch ; address register
datai equ Odh ; stored data input reg.
datao equ 0eh ; stored data output reg.
txbuf equ 10h ; transmit buffer
count equ 11h ; bits transmitted so far
bits equ 12h ; bits to transmit
bytcnt equ 13h ; byte counter for write routine
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
;
;******************************************************************************
;
; Bit Assignments
;******************************************************************************

d equ 7 ; eeprom input
do equ 6 ; eeprom output
datout equ 7 ; data out line (port_b)
datin equ 6 ; data in line (port_b)
sclk equ 5 ; clock line (port_b)
chpsel equ 4 ; chip select line (port_b)
Using the 93LC56 and 93LC66

```
org 01ffh
begin goto PWRUP ; set the reset vector
org 000h
goto PWRUP
;
;****************************************************************************
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
determine delay time.
;******************************************************************************
DELAY ROUTINE

DELAY ROUTINE

WAIT

; top movlw .110 ; timing adjustment variable
movwf loops2

; top2 nop ; sit and wait
nop
nop
nop
nop
nop

decfsz loops2 ; inner loops complete?
goto top2 ; no, go again

; decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub
;
;****************************************************************************

Start Bit Subroutine
; this routine generates a start bit
; (Chip select and DI high when clock goes high)
;******************************************************************************
BSTART

movlw b'10001111'
tms port_b
; set port b for output
; except for the data out line
bcf port_b,datin
; set datain and chipselect lines
bcf port_b, chpssel
; low just to check operation
bcf port_b,sclk
; make sure clock starts low too.

; bef port_b, chpssel ; set chip select line high
bef port_b,datin ; set data in line high

; bef port_b.sclk ; set the clock line high to
; generate the start bit

; retlw 0 ; set clock low again
;
;****************************************************************************

BITOUT routine
; This routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;******************************************************************************
BITOUT

btfss eeprom,do ; check state of data bit
goto bitlow ; low, goto bitlow
bef port_b,datin ; high, set datain high
goto clkout ; and clock it

; bitlow
bcf port_b,datin ; output a logic low
clkout

; nop
bcf port_b,sclk ; set clock line high

; retlw 0 ; return clock line low

;****************************************************************************
Using the 93LC56 and 93LC66

;***********************************************
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
;***********************************************
TX
    movf bits, w          ; set the number of bits to xmit
    movwf count

    TS
        bcf eeprom, do    ; assume bit 7 is low
        btfsc txbuf, 7     ; is bit 7 clear?
        J::ef
            eeprom, do    ; no, set data bit =1
            mil
            BI TOUT
        mil
            TXLP        ; transmit 1 bit to serial EE
            Decfsz count ; all bits done?
        goto TXLP     ; no, do another bit
        mil
            Retlw 0     ; yes, jump out

    ;***********************************************
    ; POWER-UP ROUTINE
    ; This is the program entry point, which in this case simply
    ; sets the port_a I/O lines and directs control to the
    ; erase/write enable routine.
    ;***********************************************
PWRUP

    movlw b'00000000'    ; set port_a as all output
    tris port_a
    clrf port_a          ; all lines low

    movlw b'10000000'    ; set FB7 as input, rest output
    tris port_b

    ; Fall through and do erase/write enable

    ;***********************************************
    ; EWEN (Erase/Write ENable Routine)
    ; this routine enables the device for erasing and
    ; writing. This must be done prior to any erase, write
    ; eral, wral instructions.
    ;***********************************************
EWEN

    call BSTART          ; generate a start bit

    movlw .2             ; set #bits to 2
    movwf bits

    movlw b'00000000'    ; get the opcode (00b)
    movwf txbuf          ; into the output buffer
    call TX               ; and transmit it

    movlw .8             ; set #bits to 8
    movwf bits

    movlw b'11000000'    ; get opcode and address
                           ; (11XXXXXX)
    movwf txbuf          ; into output buffer
    call TX               ; and transmit it
    bcf port_b, chpsel   ; set chip select line low
    nop

    ; Now continue on to the write command

    ;***********************************************
    ; Byte Write Routine
    ; This routine writes an AA55h pattern into
    ; 8 consecutive addresses starting at address 00.
    ; A delay of about 10ms is given after each byte
    ; for the write cycle to complete.
    ; The write is done in the x16 mode: the user must
    ; have the ORG pin tied to Vcc on the device
WRITE
;
movlw .0 ; set starting address to 00
movwf addr ;
movlw .8 ; set number of bytes to write as 8
movwf bytcnt ;

; topwr call BSTART ; generate the start bit
; movlw .2 ; set # bits to 2 for the opcode
movwf bits ;
movlw b'01000000' ; get opcode (0lb)
movwf txbuf ; into the transmit buffer
call TX ; and send it
;
movlw .8 ; set # of bits to 8 for the
movwf bits ; address
movf addr,w ; get address counter
movwf txbuf ; into output buffer
call TX ; and send it
movlw b'10101010' ; get upper byte of data (AAh)
movwf txbuf ; into the transmit buffer
call TX ; and send it
movlw b'01010101' ; get lower byte of data (55h)
movwf txbuf ; into transmit buffer
call TX ; and send it
bcf port_b, chpsel ; clear the chip select line
; to initiate write cycle
;
movlw .10 ;
movwf loops ; set delay time to 10mS
call WAIT ; and wait
;
incf addr ; increment address counter
decfsz bytcnt ; all bytes written?
goto topwr ; no, do another
; yes, go on
;
; Now continue on to the erase/write disable command
;
;******************************************************************************
; EWDS (Erase/Write Disable Routine)
; This routine executes the erase/write disable command
; which prevents the contents of the array from being
; written to.
;
;******************************************************************************

EWDS
;
; call BSTART ; generate a start bit
;
movlw .2 ; set # bits to 2
movwf bits ;
movlw b'00000000' ; get the opcode (00b)
movwf txbuf ; into the output buffer
call TX ; and transmit it
movlw .8 ; set # bits to 8
movwf bits ;
movlw b'00000000' ; get opcode and address
 movlw b'00000000' ; get opcode and address
 movfw txbuf ; into output buffer
call TX ; and transmit it
bcf port_b, chpsel ; set chip select line low
nop
goto EWEN ; start all over
; END
LIST P=16C54

;******************************************************************************
;
; 3-Wire Byte Write with Data Poll Program (107 bytes)
;
; This program demonstrates how to interface a
; Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
; device. This program will execute the erase/write enable
; command, write to 8 consecutive addresses, and then
; use the 'data polling' method to determine when the
; write cycle is complete. The program then executes
; the erase/write disable command. This sequence will
; repeat forever.
;
; When writing to a 3-wire serial EE device, the
; internally timed write cycle will begin after
; the opcode, address and data are sent to the
; device. There are two ways to make sure that the
; cycle is complete before you send it another command.
; The simplest method is to simply wait for the maximum
; write cycle time, which can be obtained from the data book.
; A more efficient method is "Data polling." This is
; done by toggling the chip select line low and then
; back high after the opcode, address and data are sent.
; The chip select line must be low for at least 250ns
; before bringing it high again. The device will pull
; the data out line low at that point, and set it high
; when the write cycle is complete. The user can then check
; or "poll" the dataout line until it goes high before
; sending the next command.
;
; As an option, the user can connect a LED to pin 18
; on the PIC16CXX (use about a 1K resistor in series) as a
; timeout indicator. This LED will come on if the
; data polling is unsuccessful and the device being
; programmed does not respond. This can be tested by
; simply running the program with no part in the socket.
;
; This program communicates to the serial EE in the
; x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
; ON THE DEVICE TO Vcc.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4MHz crystal. Clock speeds to the serial EE
; will be approximately 50Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX Serial EE
; ------- -------
; Pin 10 (R84) -> Chip Select
; Pin 11 (R85) -> Clock
; Pin 12 (R86) -> Data In
; Pin 13 (R87) -> Data Out
; ORG=Vcc
;
;******************************************************************************

Register Assignments
******************************************************************************

port_a equ Sh ; port S (port_a)
port_b equ 6h ; port 6 (port_b) used comm lines to serial EE
eeprom equ 0ah ; bit buffer
addr equ 0ch ; address register
datai equ 0dh ; stored data input reg.
datao equ 0eh ; stored data output reg.
txdif equ 10h ; transmit buffer
count equ 11h ; bits transmitted so far
bits equ 12h ; bits to transmit
bytcnt equ 13h ; byte counter for write routine
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
Using the 93LC56 and 93LC66

; Bit Assignments

d equ 7 ; eeprom input
do equ 6 ; eeprom output
datout equ 7 ; data out line (port_b)
datin equ 6 ; data in line (port_b)
sclk equ 5 ; clock line (port_b)
chpsel equ 4 ; chip select line (port_b)
timeout equ 1 ; write cycle timeout warning (port_a)

;******************************************************************************
org 01ffh
begin goto PWRUP ; set the reset vector
org 000h
goto PWRUP

;******************************************************************************
DATA POLL DELAY ROUTINE
; This routine delays 100 us is
; used for delay between polls
;******************************************************************************
dpayload
movlw .25 ; timing adjustment variable
movwf loops2
top
nop
decfsz loops2 ; loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub

;******************************************************************************
Start Bit Subroutine
; This routine generates a start bit
; (Chip select and DI high when clock goes high)
;******************************************************************************

BSTART
bcf port_b,datin ; set datain and chipselect lines
bcf port_b,chpsel ; low just to check operation
bcf port_b,sclk ; make sure clock starts low too.
nop

; bcf port_b,chpsel ; set chip select line high
bfa port_b,datin ; set data in line high
nop
bfa port_b,sclk ; set the clock line high to
goto clkout ; generate the start bit
nop
nop
bfa port_b,sclk ; set clock low again
retlw 0

;******************************************************************************
BITOUT routine
; This routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;******************************************************************************

BITOUT
btfss eeprom,do ; check state of data bit
goto bitlow ; low, goto bitlow
bfa port_b,datin ; high, set datain high
goto clkout ; and clock it

bitlow
bfa port_b,datin ; output a logic low
clkout

bfa port_b,sclk ; set clock line high
nop
bfa port_b,sclk ; return clock line low
retlw 0

;
Transmit Data Subroutine

This routine takes the byte of data stored in the ‘datao’ register and transmits it to the serial EE device.

```
TX
movf bits,w ; set the number of bits to emit
movwf count

TXLP
bcf eeprom,do ; assume bit 7 is low
btfsc txbuf, 7 ; is bit 7 clear?
hsf eeprom,do ; no, set data bit =1
afil
BITOUT ; transmit 1 bit to serial EE
rlf txbuf ; rotate txbuf left
decfsz count ; all bits done?
goto TXLP ; no, do another bit
retlw 0 ; yes, jump out
```

POWER-UP ROUTINE

This is the program entry point, which in this case simply sets the port_a I/O lines and directs control to the erase/write enable routine.

```
PWRUP
movlw b'OOOOOOOO'
bsis port_a ; set port_a as all output
drf port_a ; all lines low
movlw b'10000000'
bsis port_b ; set RB7 as input, rest output

; Fall through and do erase/write enable
```

EWEEN (Erase/Write ENable Routine)

This routine enables the device for erasing and writing. This must be done prior to any erase, write, erase, write instructions.

```
EWEN
call BSTART ; generate a start bit
movlw .2 ; set # bits to 2
movwf bits
movlw b'00000000' ; get the opcode (00b)
movwf txbuf ; into the output buffer
call TX ; and transmit it
movlw .8 ; set # bits to 8
movwf bits
movlw b'11000000' ; get opcode and address
movwf txbuf ; into output buffer
call TX ; and transmit it
bcf port_b, chpsel ; set chip select line low
nop

; Now continue on to the write command
```

WRITE

This routine writes a AA55h pattern into 8 consecutive addresses starting at address 00. It will then ‘poll’ the data out line to determine when the write cycle is complete. If the cycle has not completed within 20 ms, then it will continue.
Using the 93LC56 and 93LC66

Anyway and turn the timeout fail LED on.
The write is done in the x16 mode: the user must have the ORG pin tied to Vcc on the device. This program will repeat forever.

WRITE

movlw .0 ; set starting address to 00
movwf addr ;
movlw .8 ; set number of bytes to write as 8
movwf bytcnt ;
topwr call BSTART ; generate the start bit ;
movlw .2 ; set # bits to 2 for the opcode
movwf bits ;
movlw b'01000000' ; get opcode (01b)
movwf txbuf ; into the transmit buffer
call TX ; and send it ;
movlw .8 ; set # of bits to 8 for the
movwf bits ; address
movf addr,w ; get address counter
movwf txbuf ; into output buffer
call TX ; and send it
movlw b'10101010' ; get first half of data (AAh)
movf addr,w ; get address counter
movlw .2 ; set # bits to 2 for the
movwf bits ; address
movf addr, w ; get address counter
movlw b'01010101' ; get second half of data (55h)
movf addr,w ; get address counter
movlw .200
movwf loops ; poll 100 times before timeout (about 20ms)
polltop call dpdelay ; wait 100us
btfsc port_b, datout ; is the data out line high?
goto okpoll ; yes-cycle complete: do another
decfsz loops ; has it timed out?
goto polltop ; no, check again
bsf port_a, timeout ; yes, set timeout LED and go on
goto badpoll ;
dkpoll bcf port_a, timeout ; clear the timeout LED
badpoll bcf port_b, chpsel ; chip select line back low
incf addr ; increment address counter
decfsz bytcnt ; all bytes written?
goto topcnt ; no, do another
; yes, go on

; Now continue on to the erase/write disable command

; ****************************************************************

EWDS (Erase/Write Disable Routine)
This routine executes the erase/write disable command
which prevents the contents of the array from being written to.

; ****************************************************************
Using the 93LC56 and 93LC66

EWDS
;
call BSTART ; generate a start bit
;
movlw .2 ; set # bits to 2
movwf bits ;
movlw b'00000000' ; get the opcode (00b)
movwf txbuf ; into the output buffer
call TX ; and transmit it
movlw .8 ; set # bits to 8
movwf bits ;
movlw b'00000000' ; get opcode and address
; (00XXXXXX)
movwf txbuf ; into output buffer
call TX ; and transmit it
bcf port_b, chpsel ; set chip select line low
nop
goto EWEN ; start all over
;
;
END
Using the 93LC56 and 93LC66

LIST P=16C54

; 3-Wire Sequential Read Program (93 bytes)
;
; This program demonstrates how to interface a
; Microchip PIC16C54 to a 93LCS6 or 93LC66 Serial EE
; device. This program will read 8 consecutive addresses
; in the 'sequential read' mode. This means that the opcode
; and address are sent for the first address only. After
; the first address is read, the Chip Select line is left
; high and clocks for the remaining 7 bytes are sent to the
; device. The device will automatically increment the address
; pointer in this mode, allowing the user to read as many
; consecutive addresses as needed. This program will repeat
; forever.
;
; This program communicates to the serial EE in the
; x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
; ON THE DEVICE TO Vcc.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4MHz crystal. Clock speeds to the serial EE
; will be approximately 50 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; Pic16cxx Serial EE
; -- ----
; Pin 10 (RB4) -> Chip Select
; Pin 11 (RB5) -> Clock
; Pin 12 (RB6) -> Data In
; Pin 13 (RB7) -> Data Out
; ORG = Vcc
;
;*****************************************************************************
;
; Register Assignments
;*****************************************************************************

port_a equ Sh  ; port 5 (port_a)
port_b equ 6h  ; port 6 (port_b) used comm lines to serial EE
eeprom equ 0ah ; bit buffer
addr equ 0ch ; address register
datain equ 0dh  ; stored data input reg.
dataout equ 0eh ; stored data output reg.
txbuf equ 10h ; transmit buffer
count equ 11h ; bits transmitted so far
bits equ 12h ; bits to transmit
bytcnt equ 13h ; byte counter for read routine
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
hbyte equ 17h ; high byte for input data
lbyte equ 18h ; low byte for input data

;*****************************************************************************
;
; Bit Assignments
;*****************************************************************************

d equ 7 ; eeprom input
do equ 6 ; eeprom output
dataout equ 7 ; dataout line (port_b)
datin equ 6 ; data in line (port_b)
sclk equ 5 ; clock line (port_b)
chpssel equ 4 ; chip select line (port_b)

;*****************************************************************************

org 01ffh
begin goto PWRUP  ; set the reset vector
org 000h
goto PWRUP
;*****************************************************************************
Using the 93LC56 and 93LC66

; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;********************************************************************
WAIT
;
top movlw .110 ; timing adjustment variable
            movwf loops2

            nop ; sit and wait

            nop
            nop
            nop
            nop
            nop
            decfsz loops2 ; inner loops complete?
            goto top2 ; no, go again

            decfsz loops ; outer loops complete?
            goto top ; no, go again
            retlw 0 ; yes, return from sub
;********************************************************************

; Start Bit Subroutine
; this routine generates a start bit
; (Chip select and DI high when clock goes high)
;********************************************************************
START
        bcf port_b,datin ; set datain and chipselect lines
        bcf port_b,chpsel ; low just to check operation
        bcf port_b,slck  ; make sure clock starts low too.

        nop ;

        bcf port_b,chpsel ; set chip select line high
        bcf port_b,datin ; set data in line high

        nop

        bcf port_b,slck  ; set the clock line high to
                          ; generate the start bit

        nop

        bcf port_b,slck  ; set clock low again
        retlw 0 ;
;********************************************************************

; BITIN routine reads one bit of data from the
; serial EE device and stores it
; in 'di'
;********************************************************************
BITIN
        bcf eeprom,di   ; assume input bit is high
        bcf port_b,slck ; set clock line high

       nop;

        bfss port_b,datout ; read the data bit
        bcf eeprom,di   ; input bit was low
        bcf port_b,slck ; set clock line low

        retlw 0 ;
;********************************************************************

; Receive data routine
; This routine reads one byte of data from the part
; into the 'datai' register.
;********************************************************************
RX
        clrf datai      ; clear input buffer
        movlw .8        ; set # bits to 8
        movwf count

RXLP  fist datai   ; rotate the buffer left 1 bit
        call BITIN      ; read 1 bit
        bcf datai,0     ; assume the input bit was low

© 1993 Microchip Technology Inc.
Using the 93LC56 and 93LC66

BITOUT routine
This routine takes one bit of data in 'do' and
transmits it to the serial EE device

BITOUT
btfss eeprom,do ; check state of data bit
goto bitlCM ; low, goto bitlow
bsf port_b,datin ; high, set datain high
goto clkout ; and clock it;
bitlow bcf port_b,datin ; output a logic low
clkout bcf port_b,sclk ; set clock line high
nop
bcf port_b,sclk ; return clock line low
retlw 0

Transmit Data Subroutine
This routine takes the byte of data stored in the
'datao' register and transmits it to the serial EE device.

TX
movf bits,w ; set the number of bits to xmit
movwf count

TXLP
bcf eeprom,do ; assume bit 7 is low
btfsc txbuf,7 ; is bit 7 clear?
bsf eeprom,do ; no, set data bit =1
call BITOUT ; transmit 1 bit to serial EE
rfl txbuf ; rotate txbuf left
decfsz count ; all bits done?
goto TXLP ; no, do another bit
retlw 0 ; yes, jump out

POWER-UP ROUTINE
This is the program entry point, which in this case simply
sets the port_a I/O lines and directs control to the
erase/write enable routine.

PWRUP

READ ROUTINE (Sequential read mode)
This routine reads 8 consecutive addresses in
sequential mode starting at address 0. This
program will repeat forever

READ

movlw .0 ; set starting address to 00
movwf addr
movlw .8 ; set number of addresses to
Using the 93LC56 and 93LC66

movwf bytcnt ; read as 8

call BSTART ; generate start bit
movlw .2 ; set #bits to 2
movwf bits ;
movlw b'10000000' ; get opcode (10b)
movwf txbuf ; into output buffer
call TX ; and transmit it
movlw .8 ;
movwf bits ; set number of bits to 8
movf addr,w ; get the address
movwf txbuf ; into the output buffer
call TX ; and transmit it

rbyte

movf datai,w ; move input data to w
movwf hbyte ; xfer it to high byte

call RX ; read the high byte
movf datai,w ; move input data to w
movwf hbyte ; xfer it to low byte

inc addr ; add 1 to the address
decfsz bytcnt ; have all bytes been read?
goto rbyte ; no, read another byte

bcf port_b,chpsel ; yes, clear the chip select line
goto READ ; and start over

END
INTRODUCTION

The ER59256 and 93C06 are 256 bit (16 x 16) serial EEPROMs currently offered by Microchip. The ER59256 is fabricated in N-channel SNOS technology and the 93C06 in advanced CMOS. There are some uncertainties in the field regarding the compatibility between Microchip’s 256 bit serial EEPROM and National’s device (NMC9306). This report highlights the differences.

SOFTWARE DIFFERENCES

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Microchip 93C06</th>
<th>National NMC9306/NMC93C06</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>1 1 0 0 0 A3A2A1A0</td>
<td>1 1 0 0 0 A3A2A1A0</td>
</tr>
<tr>
<td>WRITE</td>
<td>1 0 1 0 0 A3A2A1A0</td>
<td>1 0 1 0 0 A3A2A1A0</td>
</tr>
<tr>
<td>ERASE</td>
<td>1 1 1 0 0 A3A2A1A0</td>
<td>1 1 1 0 0 A3A2A1A0</td>
</tr>
<tr>
<td>EWEN</td>
<td>1 0 0 1 1 0 0 0 0</td>
<td>1 0 0 1 1 0 0 0 0</td>
</tr>
<tr>
<td>EWDS</td>
<td>1 0 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>ERAL</td>
<td>1 0 0 1 0 0 0 0</td>
<td>1 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>WRAL</td>
<td>NOT SPECIFIED</td>
<td>NOT SPECIFIED</td>
</tr>
</tbody>
</table>

Note: EWEN .... Erase/Write Enable
EWAL .... Erase All
EWAL .... Erase All

From the instruction sets shown, the address bits for the commandsEWEN, EWDS, ERAL and WRAL are “don’t care” for 9306/93C06 and all logical 0’s for the ER59256. The WRAL instruction is not specified for the ER59256. The two LSBs of the opcode are 0’s for Microchip’s ER59256/93C06 and “don’t care” for National’s NMC9306/NMC93C06. In order to make the software fully compatible to all parts, it is recommended to design programs with all logical 0’s in place of the “don’t care” bits.

Polling is available on the 93C06 and NMC93C06. The soft polling can be done by checking the status signal on the DO line (pin 4). A low busy signal (BSY) indicates the device is still in the programming mode and a high level (RDY) represents the device is ready to receive a new instruction. The ER59256 and the NMC9306 however, do not provide this feature.
HARDWARE DIFFERENCES

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>Microchip</th>
<th>National</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN 6</td>
<td>ER59256</td>
<td>93C06</td>
</tr>
<tr>
<td>PIN 7</td>
<td>TEST</td>
<td>NC</td>
</tr>
<tr>
<td>CLOCK FREQ</td>
<td>250 kHz</td>
<td>NC</td>
</tr>
<tr>
<td>CLK DUTY CYCLE</td>
<td>25% - 75%</td>
<td>1 MHz</td>
</tr>
<tr>
<td>CLK HIGH TIME min.</td>
<td>not specified</td>
<td>not specified</td>
</tr>
<tr>
<td>CLK LOW TIME min.</td>
<td>not specified</td>
<td>500 ns</td>
</tr>
<tr>
<td>CS LOW TIME min.</td>
<td>not specified</td>
<td>250 ns</td>
</tr>
<tr>
<td>ENDURANCE</td>
<td>10K</td>
<td>100K</td>
</tr>
<tr>
<td>ESD RATING</td>
<td>1.0 KV</td>
<td>4.0 KV</td>
</tr>
<tr>
<td>E/W TIME</td>
<td>10 - 30 ms</td>
<td>2 ms</td>
</tr>
<tr>
<td>ACTIVE CURRENT</td>
<td>10 mA</td>
<td>4 mA</td>
</tr>
<tr>
<td>STANDBY CURRENT</td>
<td>3 mA</td>
<td>100 mA</td>
</tr>
</tbody>
</table>

For slower clock frequencies, the ER59256 specification would restrict the user regarding clock low and high times:

A clock frequency of 100 kHz or 10 µs cycle time would result in a clock low of 75% of 10 µs or 7.5 µs, and a clock high of 25% of 10 µs or 2.5 µs.

In reality, Microchip's ER59256 can meet National's NMC9306 specification regarding SK low and SK high of 1 µsec over all operating frequencies as 250 kHz is the maximum allowable frequency and is, therefore, the worst case.

**Chip select low time**

All parts require a chip select (CS) input low between any two instructions. Programming of the parts (ER59256, NMC9306, NMC93C06) begins at the falling edge of the CS. Microchip's 93C06, however, starts self-programming at the rising edge of the last data bit. CS going low and high with a minimum of chip select low time (Tcsl) during programming can be used for polling purposes as described in 2.

Pin 6 and 7 configuration -

Microchip uses pins 6 and 7 of the ER59256 for factory internal test purposes. Pin 6 is used to monitor the on-chip charge pump which generates the required internal high programming voltage (>20 volts) during the ERASE and WRITE cycles. Any circuitry connected to this pin will reduce the data retention of the device or even inhibit programming. Signals on pin 7 can force the device into its internal test modes resulting in overprogramming all memory locations. It is, therefore, recommended that pin 6 be left open and pin 7 tied to Vss for normal operation.

Microchip's 93C06 and National's NMC9306/NMC93C06 do not use pins 6 and 7. To make the ER59256 compatible, the TEST pin (pin 7) can be left floating but must be noise-free.

**Clock high time, clock low time**

For a clock frequency of 250 kHz, both the ER59256 and the NMC9306 have the same electrical parameters:

250 kHz or 4 µs clock cycle time

25% of the clock period (4 µs) = 1 µs (clock high time)

75% of the clock period (4 µs) = 3 µs (clock low time = 4 - 3 µs = 1 µs)

CONTACT: Bruce Negley
Memory Products Division
Portability is and will continue to be one of the key features driving the design and development of new electronic equipment. Existing products such as palmtop computers, cellular phones and data acquisition devices are becoming increasingly smaller and more powerful. Handheld markets will demand that product feature sets be enhanced and that battery operating life be increased. These market requirements, in turn, will put more demands on battery and integrated circuit technology.

In order to optimize battery technology ICs must be able to operate at voltages well below the 5V range that most ICs operate at today. In fact, even integrated circuits operating at 3 volts do not make the most of the available battery technology. Thus the need for devices that operate below 3 volts.

**BATTERY TECHNOLOGY OVERVIEW**

**Primary versus Secondary**

In portable or handheld equipment there are two major battery types employed; primary and secondary. Primary cells are disposable, and alkaline and lithium cells are the main primary batteries in use today. Alkaline batteries are extremely common in consumer applications such as electronic games, cordless phones and palmtop computers. While lithium batteries are used as a primary power source in other consumer products such as cameras and as a back-up supply in palmtops.

Secondary cells are rechargeable batteries. Among the many secondary cells available are lead-acid, Ni-cad and Nickel Metal Hydride (NMH) are the most popular.

**Voltage Ratings**

Batteries for portable applications, whether primary or secondary, have three different voltage ratings; Rated or nominal, Operating, and End-of-life. The rated voltage is usually the open circuit voltage, which in the case of alkaline "AA" cells is 1.5V. Under normal load the "operating voltage" is realized and in all cases is less than the nominal voltage. The final battery voltage rating is the end-of-life voltage, which is defined as the voltage at which 100% of the usable power of the battery is consumed or as 75% of the operating voltage. In the case of the same alkaline battery the end-of-life voltage is 0.9 volts. Table 1 lists the operating and end-of-life voltages for a single "AA" or equivalent cell of the five battery types mentioned above.

**TABLE 1 - BATTERY VOLTAGE SUMMARY**

<table>
<thead>
<tr>
<th>BATTERY TYPE</th>
<th>OPERATING VOLTAGE</th>
<th>END-OF-LIFE VOLTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALKALINE</td>
<td>1.2V</td>
<td>0.9V</td>
</tr>
<tr>
<td>LITHIUM</td>
<td>2.7V</td>
<td>2.0V</td>
</tr>
<tr>
<td>LEAD-ACID</td>
<td>2.0V</td>
<td>1.75V</td>
</tr>
<tr>
<td>NI-CAD</td>
<td>1.2V</td>
<td>0.9V</td>
</tr>
<tr>
<td>NMH</td>
<td>1.2V</td>
<td>1.0V</td>
</tr>
</tbody>
</table>

With applications employing primary cells this means that batteries are replaced less often, thus reducing operating cost as well as having a positive effect on the environment due to the use and disposal of fewer batteries. In systems with secondary cells time between recharging increased, thus enhancing the performance of the product.

**LOW VOLTAGE APPLICATIONS**

As mentioned previously, the need for increased portability will drive the employment of low voltage technology. Portable and handheld applications requiring low voltage (sub 3V) technology will be numerous and will in many cases be high volume. These include a variety of personal communications devices such as cellular and cordless phones, computing devices like palmtops and portable PCs, and a number of data acquisition devices for a variety of medical, industrial and commercial applications.
MICROCHIP'S LOW VOLTAGE PRODUCT POSITION

Prior to the announcement of a family of 1.8V Serial EEPROMs in November of 1992, Microchip had already established itself as a leader in low voltage semiconductor products. Microchip's current product portfolio consists of: 1) Serial EEPROMs that operate, both READ and WRITE, down to the following voltage levels: 1.8V, 2.0V, 2.5V and 5V; 2) ROM based microcontrollers that operate down to 2V, and 3) 3V One Time Programmable EPROMs.

The 1.8V Serial EEPROM family operates down to 1.8V without the relaxation of any specification. This includes all AC, DC retention and endurance parameters. Currently the 1.8V family consists of 3 three-wire devices, in densities ranging from 1K to 4K bits. Packaging options include 8 pin SOIC and 8 pin PDIP devices. Typical operating currents for these devices, at 1.8V, are in the 70ua range, resulting in power consumption levels of less than 150uW. This compares to 1 to 5 mW for comparable devices operating at 5V. In the near future, the 1.8V family of Serial EEPROMs will be expanded to include all 2-wire devices.

SUMMARY

With the introduction of 1.8V Serial EEPROMs, Microchip has made available silicon technology that optimizes existing battery technology. This over time will result in the development, production, marketing and application of more portable and higher performance handheld and portable computing, telecommunications, and data acquisition products.
In searching for solutions to their system non-volatile memory requirements, equipment, systems and product designers are faced with a plethora of design related issues and trade-offs. The non-volatile memory options available to them offer a variety of different device features including performance, ease of design, power consumption, operating voltage, programmability, density, and physical size. For the most part these non-volatile memory options can be grouped into two major categories: serial solutions and parallel solutions. This paper discusses the attributes of each, conducts a comparative analysis and in the process identifies the benefits and advantages of Serial EEPROMs.

SERIAL EEPROMS

The main feature that makes a device a "Serial" and sets it apart from parallel devices is, as its name implies, the ability to communicate through a serial interface. This ability has numerous benefits. First, serial communication is accomplished with a minimum number of I/O's. Serial EEPROMs require only two to four lines (depending on the hardware and software protocol) for complete communication; memory addressing, data input and output, and device control. Thus, the hardware interface requirements for Serial EEPROMs are kept at a minimum. The most common Serial EEPROMs in use today are devices that utilize a 2-wire protocol.

Another benefit of serial communication is package size. Ranging from densities of 256 to 16K bits, most Serial EEPROMs today are available in space-saving 8 pin PDIP and 150mil wide SOIC packaging. This obviously is very beneficial for applications where product size and weight is a key design factor. The final benefit is low current consumption. Due to a limited number of I/O ports and therefore on-chip support requirements, operating currents for Serial EEPROMs are usually well below 3 milli-amperes.

Other features of Serial EEPROMs include: 1) Byte programmability—The ability to erase and program one byte at a time without affecting the contents of the other memory locations in the array; 2) Clock rates of up to 6 Mhz—2-wire devices are rated at 100K hz and 400K hz per the standard I2C® protocol, while 3-wire devices can be operated at 6M hz rates; 3) Low voltage operation—Microchip has introduced a family of devices that operate, both read and write, down to 1.8V. This family complements other 2V and 2.5V low voltage Serial EEPROM families available.

PARALLEL NON-VOLATILE MEMORIES

There are a number of memory devices that fall into this category. The major ones include Parallel EEPROMs, Flash memory products, EPROMs, and SRAMs with battery back-up.

The main common feature of all of these devices is that communication with the device is done through a parallel interface, which results in a high system clock rate. Each type of device has separate data, address and control lines. Thus pin counts are in the 24 to 40 pin range. This also results in relatively large and costly packages and large footprints, even with the most advanced surface mount packages like TSOP. SRAMs with on-board batteries require DIP package heights that are significantly higher than those of standard DIP packages, adding to its package size and cost disadvantage.

Parallel EEPROM and battery backed-up SRAMs are the only two of the four major types of parallel non-volatile memories that have the capability to erase and program one byte at a time. EPROM and Flash devices require the whole array or at least large sectors to be erased prior to reprogramming.

SERIAL VERSUS PARALLEL

Serial EEPROMs have five major advantages over parallel non-volatile memories.

1) Lower Current Consumption - The maximum operating current (at 5 volts operating voltage) of a 16K serial EEPROM device is approximately an order of magnitude less than that of an equivalent density parallel EEPROM. Operating currents for 16K Serials are specified at 3mA, while 16K parallel devices are specified at 30mA and above. This relationship will continue as 64K serial devices are introduced. Since power consumption is directly proportional to current consumption, the lower the current the lower the power consumption.

2) Lower Voltage - Serial EEPROMs have been available in single supply low voltage options for some time. As mentioned above, Microchip has low voltage Serial EEPROMs that operate down to 1.8V, as well as other low voltage Serials that function down to 2.0V or 2.5V volts. 3V EPROMs...
and parallel EEPROMs and single voltage 5V flash devices are just being introduced to the market. (Most flash devices on the market today require 12V for programming in addition to the 5 volts required for normal operation). Low voltage operation also has a positive effect on power consumption. A reduction in the operating voltage from 5 volts to 1.8 volts will result in a power consumption reduction of almost 90% and almost a 65% reduction in power if the operating voltage is reduced from 3V to 1.8V.

3) Programmability - Neither currently available Flash devices nor EPROMs have the ability to program one byte at a time. Erasing is an array or sector function. Therefore, whenever one byte needs to be reprogrammed the entire array or sector must be reprogrammed. This procedure takes a relatively long amount of time to complete, time which may not be available, as is the case when storing critical parameters or data during inadvertent and unexpected system power loss. This procedure also requires software overhead to manage the retrieval and reprogramming operation.

4) Physical Size - Again, when comparing a 16K Serial EEPROM to a 16K parallel device the serial has a significant advantage. The area of the 150mil 8 pin SOIC footprint is less than 50K square mils. This compares to an area of more than 250K square mils for a 24 pin SOIC and almost 800K square mils for a 24 pin 500mil DIP package footprint.

5) I/O Requirements - Serials only require 2 to 4 input or output lines for complete communications. Most parallel devices require at least 22 lines, depending on the memory density. This results in increased microcontroller/microprocessor overhead and additional real estate to accommodate the numerous hardware lines.

The advantages that parallel devices currently have over serial EEPROMs is memory density and AC performance. However, in most microcontroller based applications for which Serial EEPROMs are intended, high density and AC are not the most critical design issues or most needed product features.

The key benefits of Serial EEPROM solutions as a result of the advantages outlined above, are reduced system costs, enhanced feature sets, and improved system performance. System size and weight is reduced and power sourcing requirements are kept at a minimum. The following graph compares some of the main attributes of a 16K Serial EEPROM device to a 16K Parallel device.
USES AND APPLICATIONS OF SERIAL EEPROMS

Uses of Serial EEPROMs

The days of simply being a DIP switch replacement for Serial EEPROMs is over. Here is a list of the functions that Serial EEPROMs perform in a variety of computer, industrial, telecommunication, automotive and consumer applications:

1) Memory storage of channel selectors or analog controls (volume, tone, etc.).
2) Power down storage and retrieval of events such as fault detection or error diagnostics.
3) Electronic real time event or maintenance log such as page counting.
4) Configuration storage.
5) Last number redial and speed dial storage.
6) User in-circuit look-up tables.

Serial EEPROM Applications

Serial EEPROMs have found homes in hundreds of embedded control applications in all major application markets. The following list demonstrates the number and variety of applications for serial EEPROMs.

<table>
<thead>
<tr>
<th>Market</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSUMER</td>
<td>TV tuners, VCRs, CD players, cameras, radios, and remote controls</td>
</tr>
<tr>
<td>COMPUTER/OA</td>
<td>Printers, copiers, PCs, palmtop and portable computers, disk drives and organizers</td>
</tr>
<tr>
<td>INDUSTRIAL</td>
<td>Bar code readers, point of sale terminals, smart cards, lock boxes, garage door openers, test measurement equipment and medical equipment</td>
</tr>
<tr>
<td>TELECOMM</td>
<td>Cellular, cordless and full feature phones, faxes, modems, pagers, and satellite receivers</td>
</tr>
<tr>
<td>AUTOMOTIVE</td>
<td>Air bags, anti-lock brakes, odometers, radios and keyless entry</td>
</tr>
</tbody>
</table>

Using Serial EEPROMs for critical data and configuration storage has only recently become a reality. The current offerings of 2 and 3 wire serial devices offers the systems designer interesting alternatives to the standard parallel EEPROM devices. The Serial EEPROM is basically a standard EEPROM array without the normal parallel data and address I/O. These functions are handled via serial I/O ports coupled with internal self-timed state machines. Not only will the serial device...
Serial EEPROM Solutions vs. Parallel Solutions

save power, board space, and cost, but they also offer the advantage of fewer I/O and consequently power in the embedded microcontroller because less I/O are needed to control the same functions. A typical embedded application is shown in FIGURE A, depicting a controller and several functions used in a personal communications device, such as a mobile or portable phone. The EEPROM stores speed dial and last number redial numbers, credit card numbers, ID numbers, and configuration parameters.

FIGURE B shows these same functions using a controller with fewer I/O and a Serial EEPROM. There is no loss of functionality but a significant savings in current, board space, I/O pads, and cost. The serial solution employs 8 to 16 less I/O on the microcontroller, freeing up much needed functionality, and possibly allowing for a much smaller device package and downsized circuit boards.

SUMMARY

Serial EEPROMs are ideal cost effective solutions to all non-volatile memory embedded control applications that require: 1) A small footprint space saving format, 2) The ability and ease of programming one byte at a time, 3) Low current consumption and low operating voltage, 4) Low microcontroller overhead and support and, 5) The best price performance non-volatile memory solution available.

Their size, ease of programmability, low power consumption, and low cost make Serial EEPROMs extremely suitable for all the fast growing handheld and portable battery powered computer, personal communications, medical and industrial markets.

Author: Tom Tyson
Memory Products Division
SECTION 7
DEVELOPMENT TOOLS

Introduction to Microchip Development Tools .............................................................................. 7-1
Application-Specific Standard Products Division:
   PICSEE™ Tools Product Brief ................................................................................................. 7-3
Logic Product Division:
   MPALC Cross Assembler Product Brief ................................................................................ 7-5
   MPASM Universal Assembler/Linker Product Brief ............................................................... 7-7
   MPSIM Simulator Product Brief ............................................................................................ 7-9
   PICMASTER™-16X System Product Brief .............................................................................. 7-11
   PICMASTER-17 System Product Brief ................................................................................... 7-15
   PRO MASTER™ Product Brief .............................................................................................. 7-19
   PICSTART™-16B Product Brief .............................................................................................. 7-21
Memory Products Division:
   Total Endurance™ Predictive Software Model ...................................................................... 7-23
   Serial EEPROM Evaluation Board ........................................................................................ 7-25
   Microchip Bulletin Board Service (BBS) ............................................................................... 7-27
PIC16/17 MICROCONTROLLER DEVELOPMENT SUPPORT

Look to Microchip to bring you the most complete set of development tools available for the entire PIC16/17 Microcontroller product line. There are a host of configurations available to suit all of your development needs. All of our products are PC-hosted systems allowing ease of use and maximum flexibility. Our products are available as bundled systems or individual components, which ever fits your development needs.

PIC16C5X SUPPORT PRODUCTS

PICMASTER-16X
PIC16CXX PC-Based High Performance In-Circuit Emulator System

PICMASTER Universal Emulator Pod, PIC16CXX Emulator Probe, PC (ISA) Interface Board, Emulation Control Software, Demonstration Board, PRO MASTER, MPALC Assembler, MPSIM Simulator, Power Supply, Cables, and Documentation.

PICSTART-16B
Development Programmer for the PIC16C5X, PIC16C71, and PIC16C84
Includes MPALC Assembler, MPSIM Simulator, and Development Programmer Board.

MPALC
PIC16C5X/PIC16CXX PC-Based Assembler Program
Assembler Software on 3.5" diskette and User's Manual.

MPASM
PIC16C5X/XX/17CXX PC-Based Assembler Program
Assembler Software on 3.5" diskette and User's Manual.

MPSIM
PIC16C5X PC-Based Simulator Program

PIC17CXX SUPPORT PRODUCTS

PICMASTER-17
PIC17CXX PC-Based High Performance In-Circuit Emulator System

PICMASTER Universal Emulator Pod, PIC17CXX Emulator Probe, PC (ISA) Interface Board, Emulation Control Software, PRO MASTER Programmer, Power Supply, Cables, and Documentation.

UNIVERSAL SUPPORT PRODUCTS

PRO MASTER Universal Programmer
INTRODUCTION

The PICSEE Development Systems provide the product development engineer with cost effective and timely design tool solutions for the MTA8XXX family of multichip modules. They are designed specifically for the MTA8XXX family. These tools work in conjunction with existing hardware and software design tools for the PIC16/17 microcontroller family. This allows the development engineer to efficiently implement systems utilizing these multichip modules with a minimal learning curve and capital investment.

PICSEEKIT — P/N AC812001

• Supports MTA81010
• Programming Adapters for PDIP and SOIC packages
• Daughter card for PICPROBE-16A
• I²C™ bus Serial Communication Application Software

This kit is supports the MTA81010 multichip module. It contains programming adapters, a PICMASTER™ emulator daughter board and MTA81010 product samples in 28-lead PDIP. Also included is an MSDOS, PC-compatible 3.5-inch software diskette that contains example source code for implementing the I²C serial bus protocol to communicate with a Serial EEPROM. Documentation is provided for all of the included hardware and software.

Programming Support

Two programming adapters are provided to allow the MTA81010's internal program EPROM as well as it's data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip’s PIC16C54 can program the MTA81010's internal EPROM. Also, any programmer that supports Microchip’s 24LC01B Serial EEPROM can program the MTA81010's internal Serial EEPROM. There is one adapter for MTA81010's in DIP packages and another for SOIC packages. Both DIP and SOIC programming adapters interface to programmers via a 300 mil DIP header.

Emulation Support

The emulator daughter board allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA81010 Microcontroller with Serial EEPROM. This daughter board replaces Microchip's PIC16C5X Emulator Probe Header (P/N AC162009) emulator probe to support the MTA81010. The daughter board provides the required translation from a PIC16C54 pin out to the MTA81010 pin out. It also contains a discrete 24LC01B Serial EEPROM to provide the same functions as the MTA81010's internal EEPROM. This provides a cost-effective emulation solution to customers who may wish to purchase a PICMASTER in-circuit emulator or those that already have a PICMASTER.

Software Support

Example source code for I²C Bus communication with a serial EEPROM is included in the PICSEE. This pre-tested code can be used directly or modified by the developer to meet their specific needs. This example code is provided royalty free and license free.

PICSEESTART — P/N DV813001

• Complete Low Cost Development Solution for MTA81010
• Combines PICSEEKIT AC812001 and PICSTART DV163001
• MPALC Assembler
• MPSIM Simulator
• Low-Cost Programmer
• Programming Adapter Sockets
• I²C Bus Applications Software

This kit combines the PICSEEKIT (P/N AC812001) with a PICSTART™ (P/N DV163001) to form a complete low-cost development system for the MTA81010 multichip module. It is designed to support the MTA81010 during the software development and initial prototype phases of new product development. It contains tools for software development and debugging, as well as programmer for programming the MTA81010's internal EPROM program memory. For a more detailed description, please refer to the PICSEEKIT P/N AC812001 and PICSTART P/N DV163001 product descriptions.
SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

<table>
<thead>
<tr>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC812001</td>
<td>PICSEEKIT FOR MTA81010</td>
</tr>
<tr>
<td>DV813001</td>
<td>PICSEESTART FOR MTA81010</td>
</tr>
</tbody>
</table>
This product brief describes the technical aspects of the Microchip Assembler (MPALC) developed by Microchip Technology Incorporated.

The MPALC Cross Assembler is a PC hosted symbolic assembler. It supports the PIC16C5X and PIC16CXX CMOS microcontroller series.

MPALC offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

**MPALC REQUIREMENTS**

MPALC will run on any IBM PC/XT™, AT™ or compatible computer, running DOS 3.31 or later. The distribution media is 3 1/2", low density (720k) floppy. It is distributed at the root level, and may be executed directly from the floppy.

No special display or ancillary devices are required.

**MPALC ASSEMBLER FEATURES**

MPALC supports the 12 bit PIC16C5X and the 14 bit PIC16CXX cores. The 12 bit core has 33 instructions, the 14 bit core has 35.

All instructions in both instruction sets are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPALC has the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for PIC16C5X and PIC16CXX Microchip micro-controllers.
- Macro Assembly capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Output formats: INHX6S, INHX8M, INHX16, and relocatable objects.
- Supports Hex (default), Decimal and Octal source and listing formats.

**MPALC DIRECTIVE LANGUAGE**

MPALC provides a full featured directive language represented by the following four classes of directives:

- **Data Directives** are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- **Listing Directives** control the MPALC listing format. They allow the specification of titles, subtitles, page ejects and other listing controls.
- **Control Directives** permit sections of conditionally assembled code.
- **Macro Directives** control the execution and data allocation within macro body definitions.

**MPALC INSTRUCTION SET**

MPALC supports the entire instruction set of the PIC16C5X and PIC16CXX micro-controllers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Control Operations

The Microchip microcontroller instruction set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- Two data transfer operations
- Six arithmetic operations (the PIC16CXX series provides two more)
- Six logical operations
- Three rotate operation

MPALC provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The PIC16C5X and PIC16CXX instruction sets allow read and write of special function registers such as the PC and status registers.
This product brief describes the technical aspects of the PIC16/17 Assembler developed by Byte Craft Limited and distributed by Microchip Technology. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC16C5X CMOS, PIC16CXX, and PIC17CXX families.

MPASM offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from the Microchip Universal Emulator System (PICMASTER).

**MPASM REQUIREMENTS**

MPASM will run on any IBM PC/XT™, AT™ or compatible computer running DOS 4.1 or later. The distribution media is 3 1/2", low density (720K) floppies. It is distributed at the root level, and may be executed directly from the floppy.

No special display or ancillary devices are required.

**MPASM ASSEMBLER FEATURES**

MPASM supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions in both instruction sets are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPASM have the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro Assembly Capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.

**MPASM DIRECTIVE LANGUAGE**

MPASM provides a full featured directive language represented by four basic classes of directives:

- **Data Directives** are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- **Listing Directives** control the MPASM listing display. They allow the specification of titles and subtitles, page ejects and other listing control.
- **Control Directives** permit sections of conditionally assembled code.
- **Macro Directives** control the execution and data allocation within macro body definitions.

**MPASM INSTRUCTION SET**

MPASM supports the entire instruction set of the PIC16C5X, PIC16CXX and PIC17CXX microcontrollers, as represented in the following four classes of instructions:

- **Data Move Operations**
- **Arithmetic and Logical Operations**
- **Bit Manipulation Operations**
- **Special Control Operations**

The Microchip microcontroller set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- **Data Transfer Operations**
- **Logical Operations**
- **Rotate Operations**

MPASM provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The Microchip microcontroller instruction sets allow read and write of special function registers such as the PC and status registers.
MPLINK

MPLINK supports the linking of multiple relocatable objects created by MPASM into a single absolute hex file, suitable for simulating, emulating, and programming. MPLINK supports the hex outputs supported by MPASM and absolute listing file.

MPLIB

MPLIB provides the ability to group several relocatable objects into a logical collection, in one file. Libraries created by MPLIB can be referenced by MPASM. Only those objects required by the linking phase are included in the resulting hex output.
MPSIM is a discrete event simulator software application designed to imitate operation of the PIC16C5X and PIC16CXX microcontrollers. It allows the user to debug software that will use any of these micro-controllers.

At any instruction boundary, you may examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. MPSIM gives you a solid, low cost, source-level debug tool to help you through the early design verification stages of your project.

**MPSIM REQUIREMENTS**

MPSIM requires an IBM PC/XT™, AT™ or compatible computer running DOS version 3.31 or later. The PC needs a 3 1/2" floppy disk drive and at least 256K of main memory; MPSIM.EXE occupies roughly 150K. Recommended is a hard disk drive with 5 Mb of available storage.

**MPSIM SIMULATOR**

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging Microchip microcontroller assembler code.

The simulator is meant for use with smaller projects not requiring precise, more extensive development equipment. Since the PIC16C5X architecture is essentially a single tasking micro-controller without interrupts, many applications can be developed by using a simulator program alone.

The PIC16CXX family supports various peripherals and interrupt strategies. These interrupts can be simulated but certain peripheral functions (such as A/D conversions) are not.

The MPSIM Simulator has the following features to assist in the debugging of software / firmware for the user:

**Program Load / Save**

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

**Display and Alter**

Provisions are made to display and alter Program Memory, Register Files and status register bits. Also, simulator information such as cycle times, elapsed time, and step count can be displayed.

**Disassembler**

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

**Utility Functions**

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use and ASCII data.

**Symbolic Debugging**

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

**Execution and Trace**

During program execution, a number of items can be traced. Address ranges, registers and register contents and others.

**Breakpoints**

The user may specify up to 512 breakpoints at any one time.

**Assembler Support**

MPSIM supports both the Microchip MPALC and the MPASM Universal Assembler.
SYSTEM FEATURES

General:

- Complete Hi-Performance PC-based Microcontroller Development System for the PIC16CXX family.
- For use on PC-compatible 286, 386, and 486 machines under Microsoft Windows® 3.X environment.
- Assembler Software, Emulator System, and EPROM Programmer unit, sample kit, and demonstration hardware and software provide a complete microcontroller product development environment.

Emulator System:

- Hi-Performance In-Circuit Emulation of Microchip Microcontrollers.
- Real-time instruction emulation.
- Single and Multiple instruction step execution.
- Program Memory emulation and memory mapping capability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory.
- Real-time trace memory capture of 40 bits of information for each instruction cycle in an 8Kx40 trace buffer. Trace region can range from 0 to 64K in any address combinations.
- Real-time trace data can be captured and displayed without halting emulation.
- Unlimited number of hardware breakpoints can be set anywhere in the program memory.
- External Break with “AND”/”OR” capability with internal breakpoints.
- Multiprocessor emulation capability. Up to eight PICMASTER emulators can be synchronized on a single PC, for multi-processor development.
- Extended 48-bit cycle counter.
- Trigger Output available on any range of addresses.
- Full Symbolic Debug Capability. Symbolic display and alter of all register files, special purpose registers, stack registers, and bank registers.
- Selectable Internal Emulator Clock or User Target (Prototype) System Clock.
- User selectable internal or external Power Supply (provided).
PICMASTER-16X Development System

EPROM Programmer System:
- PRO MASTERTM Device Programmer unit for all current PIC16C5X and PIC16CXX products.
- Operates as a Stand-alone Unit or in Conjunction with a PC-compatible host system.
- Performs READ, PROGRAM, and VERIFY functions in Stand-alone mode.
- PC Host Software provides file display and editing, file transfer to and from programmer unit, device serialization, and program voltage calibration.

Macro Assembler:
- Provides translation of Assembler source code to object code for the PIC16C5X and PIC16CXX family of microcontrollers.
- Macro-assembly and conditional assembly capability.
- Produces Object files, Listing files, Symbol files, and special files required for symbolic debug with the PICMASTER Emulator System.
- Binary / Hex output formats: INHX8S, INHX8M, INHX16, and PICMASTER.

SYSTEM DESCRIPTION

The PICMASTER Universal In-Circuit Emulator System is intended to provide the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PIC16CXX. The PICMASTER system currently supports the PIC16C54, PIC16C55, PIC16C56 and PIC16C57 at clock frequencies of 4 MHz and PIC16C71, PIC16C84 to 10 MHz.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

The Emulator System is designed to operate on low-cost PC-compatible machines ranging from 286-AT class ISA-bus systems through the new 486 EISA-bus machines. The development software runs in the Microsoft Windows 3.X environment, allowing the operator access to a wide range of supporting software and accessories.

Provided with the PICMASTER System is a high performance real-time In-Circuit Emulator, a microcontroller programmer unit, a macro assembler program, and a simulator program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC16CXX microcontroller line.

Coupled with the user's choice of text editor, the system is ready for development of products containing any of Microchip's microcontroller products.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is included for additional convenience.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS system.

Host System Requirements:
The PICMASTER has been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The AT platform and Windows 3.X environment was chosen to best make these features available to you the end user. To properly take advantage of these features, PICMASTER requires installation on a system having the following minimum configuration:
- PC/AT-compatible machine: 286, 386SX, 386DX, or 486 with ISA or EISA Bus.
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MSDOS / PCDOS version 3.1 or greater.
- Microsoft Windows version 3.0 or greater operating in either standard or 386 enhanced mode).
- 1 Mbyte RAM (2 Mbytes recommended).
- One 5.25" floppy disk drive.
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)
- Microsoft mouse or compatible (highly recommended).

© 1993 Microchip Technology Inc.
Emulator System Components:

The PICMASTER Emulator Universal System consists primarily of four major components:

- **Host-Interface Card:** The PC Host Interface Card connects the emulator system to a PC compatible system. This high-speed parallel interface requires a single half-size standard AT/ISA slot in the host system. A 37-conductor cable connects the interface card to the external Emulator Control Pod.

- **Emulator Control Pod:** The Emulator Control Pod contains all emulation and control logic common to all microcontroller devices. Emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic are contained here. The Pod controls and interfaces to an interchangeable target-specific emulator probe via a 14" precision ribbon cable.

- **Target-specific Emulator Probe:** A probe specific to microcontroller family to be emulated is installed on the ribbon cable coming from the control pod. This probe configures the universal system for emulation of a specific microcontroller. Currently, the PIC16C5X family, PIC16C71, PIC16C84, and the PIC17C42 microcontrollers are supported. Future microcontroller probes will be available as they are released.

- **PC Host Emulation Control Software:** Host software necessary to control and provide a working user interface is the last major component of the system. The emulation software runs in the Windows 3.X environment, and provides the user with full display, alter, and control of the system under emulation. The Control Software is also universal to all microcontroller families.

The Windows 3.X System is a multitasking operating system which will allow the developer to take full advantage of the many powerful features and functions of the PICMASTER system.

PICMASTER emulation can operate in one window, while a text editor is running in a second window. Dynamic Data Exchange (DDE), a feature of Windows 3.X, will be available in this and future versions of the software. DDE allows data to be dynamically transferred between two or more Windows programs. With this feature, data collected with PICMASTER can be automatically transferred to a spreadsheet or database program for further analysis.

Under Windows 3.X, up to eight PICMASTER emulators can run simultaneously on the same PC making development of multi-microcontroller systems possible (e.g., a system containing a PIC16CXX processor and a PIC17CXX processor).

The PRO MASTER Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.

**STAND-ALONE MODE**

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

**VERIFY**

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

**PROGRAM**

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

**READ**

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

**PC HOST CONNECT MODE**

When the PRO MASTER is connected to a host PC system, many more options and conveniences are available to the user. Host mode allows full interactive control over the PRO MASTER unit. A full screen, user-friendly software program is provided to fully assist the user.

As in stand-alone mode, parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified. In addition, other features available in host-mode are:
PICMASTER-16X Development System

Editing
A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program and fuse data can be loaded and saved to DOS disk files. Files generated by the Assembler program are directly loadable into programmer memory.

VDD and VPP Adjust
The programming environment voltage settings of VDD max, VDD min, and VPP can be set and altered only on PC host mode. The voltage settings allow the user to program the part in the environment that the part will be used. The part will be programmed at VDD max and verified at VDD min. VPP is the programming voltage.

SALES AND SUPPORT
To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the factory or the listed sales offices.

<table>
<thead>
<tr>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM167007</td>
<td>Complete PICMASTER-16A System for PIC16C5X</td>
</tr>
<tr>
<td>EM167010</td>
<td>Complete PICMASTER-16A System for PIC16C5X without Programmer</td>
</tr>
<tr>
<td>EM167011</td>
<td>Complete PICMASTER-16B System for PIC16C71</td>
</tr>
<tr>
<td>EM167012</td>
<td>Complete PICMASTER-16B System for PIC16C71 without Programmer</td>
</tr>
<tr>
<td>EM167013</td>
<td>Complete PICMASTER-16C System for PIC16C84</td>
</tr>
<tr>
<td>EM167014</td>
<td>Complete PICMASTER-16C System for PIC16C84 without Programmer</td>
</tr>
<tr>
<td>EM167017</td>
<td>Complete PICMASTER-16E System for PIC16C64</td>
</tr>
<tr>
<td>EM167018</td>
<td>Complete PICMASTER-16E System for PIC16C64 without Programmer</td>
</tr>
</tbody>
</table>
SYSTEM FEATURES

General:
- The PICMASTER-17 Development System is designed by Microchip Technology Inc. and manufactured in the U.S.A.
- Complete Hi-Performance PC-based Microcontroller Development System for the PIC17CXX family.
- For use on PC-compatible 286, 386, and 486 machines under the Windows® 3.X environment.
- Assembler Software, Emulator System, and Programmer unit, sample kit, and software provide a complete microcontroller product development environment.

Emulator System:
- Universal In-Circuit Emulation pod supports emulation of PIC17CXX family. It can easily support other Microchip microcontroller products with the purchase of a low cost personality probe kit.
- Real-time emulation to 16 MHz.
- Single and Multiple instruction step execution.
- Program Memory emulation and memory mapping capability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory.
- Real-time trace memory capture of 40 bits of information for each instruction cycle in an 8Kx40 trace buffer. Trace region can range from 0 to 64K in any address combinations.
- Real-time trace data can be captured and displayed without halting emulation.
- Unlimited number of hardware breakpoints can be set anywhere in the program memory.
- External Break with "AND"/"OR" capability with internal breakpoints.
- Multiprocessor emulation capability. Two or more PICMASTER emulators can be synchronized on a single PC for multi-processor development.
- Extended 48-bit cycle counter.
- Trigger Output available on any range of addresses.
- Full Symbolic Debug Capability. Symbolic display and alter of all register files, special purpose registers, stack, and bank registers.
- Selectable Internal Emulator Clock or User Target (Prototype) System Clock.
- User selectable internal or external Power Supply (provided).
PICMASTER-17 PIC17CXX In-Circuit Emulator

EPROM Programmer System:
- PRO MASTER™ EPROM Programmer unit for all Microchip PIC17CXX CMOS microcontrollers.
- Operates as a Stand-alone Unit or in Conjunction with a PC-compatible host system.
- Performs READ, PROGRAM, and VERIFY functions in Stand-alone mode.
- PC Host Software provides file display and editing, file transfer to and from programmer unit, device serialization, and program voltage calibration.

Macro Assembler:
- ASM-17 provides macro-assembly and conditional assembly capability.
- Provides translation of Assembler source code to object code for the PIC17CXX family of microcontrollers.
- Produces Object files, Listing files, Symbol files, and special files required for symbolic debug with the PICMASTER Emulator System.
- Binary / Hex output formats: INHX8S, INHX8M, INHX32.

SYSTEM DESCRIPTION

The PICMASTER Universal In-Circuit Emulator System provides the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PIC16CXX and PIC17CXX families.

The PICMASTER-17 System is configured to support the PIC17C42 and related PIC17CXX family members.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

The Emulator System is designed to operate on low-cost PC-compatible machines ranging from 286-AT class ISA-bus systems through the new 486 EISA-bus machines. The development software runs in the Microsoft Windows 3.X environment, allowing the operator access to a wide range of supporting software and accessories.

Provided with the PICMASTER System is a high performance real-time In-Circuit Emulator, a microcontroller programmer unit, and a macro assembler program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC17CXX microcontroller line.

Coupled with the user's choice of text editor, the system is ready for development of products containing any of Microchip's microcontroller products.

A "Quick Start" PIC17CXX Product Sample Pak containing user programmable parts is included for additional convenience.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing Microchip Technology's Bulletin Board System (BBS).
Host System Requirements:

The PICMASTER has been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The AT* platform and Windows 3.X environment was chosen to best make these features available to you, the end user. To properly take advantage of these features, PICMASTER requires installation on a system having the following minimum configuration:

- PC/AT-compatible machine: 286, 386SX, 386DX, or 486 with ISA or EISA Bus
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MSDOS / PCDOS version 3.1 or greater.
- Microsoft Windows version 3.0 or greater operating in either standard or 386 enhanced mode.
- 1 Mbyte RAM (2 Mbytes recommended).
- One 5.25" floppy disk drive.
- Approximately 10 Mbytes of hard disk (1 Mbyte required for PICMASTER, remainder for Windows 3.0 system)
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)
- Microsoft mouse or compatible (highly recommended).

Emulator System Components:

The PICMASTER Emulator Universal System consists primarily of four major components:

- **Host-Interface Card**: The PC Host Interface Card connects the emulator system to a PC compatible system. This high-speed parallel interface requires a single half-size standard AT / ISA slot in the host system. A 37-conductor cable connects the interface card to the external Emulator Control Pod.

- **Emulator Control Pod**: The Emulator Control Pod contains all emulation and control logic common to all Microchip CMOS microcontroller devices. Emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic are contained here. The Pod controls and interfaces to an interchangeable target-specific emulator probe via a 14" precision ribbon cable.

- **Target-Specific Emulator Probe**: A probe specific to microcontroller family to be emulated is installed on the ribbon cable coming from the control pod. This probe configures the universal system for emulation of a specific microcontroller. Currently, the PIC16CXX family, and the PIC17C42 microcontrollers are supported. Future microcontroller probes will be available as they are released.

- **PC Host Emulation Control Software**: Host software necessary to control and provide a working user interface is the last major component of the system. The emulation software runs in the Windows 3.X environment, and provides the user with full display, alter, and control of the system under emulation. The Control Software is also universal to all microcontroller families.

The Windows 3.X System is a multitasking operating system which allows the developer to take full advantage of the many powerful features and functions of the PICMASTER system.

PICMASTER emulation can operate in one window, while a text editor is running in a second window. PICMASTER supports the window feature Dynamic data Exchange (DDE). DDE allows data and commands to be dynamically transferred between two or more Windows programs. With this feature, data collected with PICMASTER can be automatically transferred to a spreadsheet or database program for further analysis.

Under Windows 3.X, two or more PICMASTER emulators can run simultaneously on the same PC making development of multi-microcontroller systems possible (e.g., a system containing a PIC16C5X controller and a PIC17CXX controller) or two or more of the same controller family.

This allows data collected by PICMASTER to be automatically transferred to spreadsheets, data bases, or other analytic tools for further evaluation. DDE also allows automated control of PICMASTER which in turn allows development of automated test suites, life testing and production testers.

**PRO MASTER EPROM Programmer:**

The PRO MASTER Programmer system included in the PICMASTER Development System provides the product developer with the ability to program (transfer) the developer's software into PIC17CXX microcontrollers.

The programmer unit comes complete with accessories for use with a PC host computer. Supplied are interface cables and connectors to a standard PC serial port COM 1-4, power supply cable, and host operating software.

The PRO MASTER Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user as explained below.
PICMASTER-17 PIC17CXX In-Circuit Emulator

STAND-ALONE MODE
Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY
VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM
In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ
A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE
When the PRO MASTER is connected to a host PC system, many more options and conveniences are available to the user such as serialized code programming. Host mode allows full interactive control over the PRO MASTER unit. A full screen, user-friendly software program is provided to assist the user.

As in stand-alone mode, parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified. Other features available in host-mode are:

Editing
A large screen buffer editing facility allows the user to change and program location in hexadecimal mode. Complete program and fuse data can be loaded and saved to DOS disk files. Files generated by the Assembler program are directly loadable into programmer memory.

VDD and VPP Adjust
The programming environment voltage settings of VDD max, VDD min, and VPP can be set and altered only on PC host mode. The voltage settings allow the user to program the part in the environment that the part will be used. The part will be programmed at VDD max and verified at VDD min. VPP is the programming voltage.

SALES AND SUPPORT - To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the factory or the listed sales offices.

<table>
<thead>
<tr>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM177001</td>
<td>PICMASTER PIC17CXX In-Circuit Emulator System</td>
</tr>
<tr>
<td>EM177004</td>
<td>PICMASTER-17 System without PRO MASTER Programmer</td>
</tr>
</tbody>
</table>
SYSTEM FEATURES

EPROM Programmer System:

- PRO MASTER Programmer unit for the PIC16C5X, PIC16CXX, and PIC17CXX Microcontroller family.
- Operates as a Stand-alone Unit or in Conjunction with a PC Compatible host system.
- READS, PROGRAMS, and VERIFIES in Stand-alone mode.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit
- Communication Via RS-232

SYSTEM DESCRIPTION

PRO MASTER Programmer:

The PRO MASTER Programmer system provides the product developer with the ability to program user software into PIC16C5X, PIC16CXX, and PIC17CXX CMOS microcontrollers.

The programmer unit comes complete with accessories to be used with the PC host computer. Supplied are interface cables and connectors to a standard PC serial port, a universal input power supply unit, and host operating software.

The PRO MASTER Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.
STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

The PRO MASTER provides a very user friendly user interface which allows complete control over the programming session.

The PRO MASTER host software is a DOS windowed environment with full mouse support to allow the user to point and click when entering commands.

The Host Software communicates with the PRO MASTER via the serial port of the PC. Any of the four (COM 1-4) ports may be used. The communication is done at 19200 baud to insure fast throughput. Communication will be established with the PRO MASTER Device Programmer prior to any transfers taking place.

Serialization is done by generating a serialization file, and then using that file to serialize locations in the PIC microcontroller. Once a serialization file is generated, it may be used over different programming sessions. Serial numbers are automatically marked as used when a PIC is programmed successfully with that serial number.

Complete control over the programming environment is also provided. Control over the programming and verify voltage of Vdd insures that the Microcontroller will perform in the desired environment. Programming (Vpp) voltage is also adjustable to insure complete compatibility with future programming algorithms.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

<table>
<thead>
<tr>
<th>PROGRAMMER PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG007001</td>
<td>Programmer Kit as described above</td>
</tr>
<tr>
<td>PG007002</td>
<td>Programmer Kit without power supply</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOCKET PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC164001</td>
<td>PIC16C54 thru C57 18 &amp; 28 Lead PDIP Socket Module</td>
</tr>
<tr>
<td>AC164002</td>
<td>PIC16C54 thru C57 18 &amp; 28 Lead SOIC Socket Module</td>
</tr>
<tr>
<td>AC164003</td>
<td>PIC16C54/56 20 lead SSOP Socket Adapter</td>
</tr>
<tr>
<td>AC164010</td>
<td>PIC16C71/84 18 Lead PDIP/SOIC Socket Module</td>
</tr>
<tr>
<td>AC164011</td>
<td>PIC16C55/57 28 Lead SSOP Socket Adapter</td>
</tr>
<tr>
<td>AC164012</td>
<td>PIC16C64 40 PIN DIP Socket Module</td>
</tr>
<tr>
<td>AC164013</td>
<td>PIC16C64 44 PIN PLCC Socket Module</td>
</tr>
<tr>
<td>AC164014</td>
<td>PIC16C64 44 PIN PQFP Socket Adapter</td>
</tr>
<tr>
<td>AC174001</td>
<td>PIC17C42 40 Lead PDIP Socket Module</td>
</tr>
<tr>
<td>AC174002</td>
<td>PIC17C42 44 Lead PLCC Socket Module</td>
</tr>
<tr>
<td>AC174003</td>
<td>PIC17C42 44 Lead QFP Socket Module (future release)</td>
</tr>
</tbody>
</table>

Socket modules are sold separately.
Microchip

PICSTART™-16B

PIC16CXX Low-Cost Microcontroller Development System

SYSTEM FEATURES

EPROM Programmer System:
• Operates with a PC-compatible host system.
• READS, PROGRAMS, and VERIFIES EPROM Memory.
• PC Host Software provides file display and editing, and transfer to and from Programmer unit.

Macro Assembler:
• Provides translation of Assembler source code to object code for all PIC16C5X and PIC16CXX microcontroller product family.
• Macro-Assembly capability.

• Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
• Output formats: INHX8S, INHX8M and INHX16.

Simulator:
• Instruction-level Simulator of the PIC16CXX microcontroller product family.
• For PC-compatible systems running the MSDOS operating system.
• Full screen simulation user interface.
• Symbolic debugging capability.
• I/O stimulus input capability.

"Quick Start" Sample Kit:
• Provides the User / Developer with a sample kit of PIC16CXX parts for initial prototype use.
SYSTEM DESCRIPTION

The PICSTART-16B Development System provides the product development engineer with an alternative low-cost introductory microcontroller design tool set for the PIC16CXX family where full real-time emulation is not required. The equipment in the PICSTART-16B system operates on any PC compatible machine running the MSDOS/PCDOS operating system.

Provided in the System is an MSDOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPALC).

Sample software programs to be run on the simulator are provided to help the user to quickly become familiar with the development system and the PIC16CXX microcontroller line.

The user need only provide his or her own preferred text editor and the system is ready for development of end products using the PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, or PIC16C84 microcontrollers.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is also included.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS.

PICSTART-16B Development Programmer:

The Microchip device programmer system included in the PICSTART-16B Development System provides the product developer with the ability to program user software into PIC16CXX EPROM microcontrollers. It is designed to be a development programmer and not recommended for use in a production environment.

The programmer unit connects to a standard PC serial port.

A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified.

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program data can be loaded and saved to DOS disk files. Files generated by the MPALC Assembler program are directly loadable into programmer memory.

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16CXX assembler code.

The simulator is meant for use with smaller projects not requiring precise more extensive development equipment. Since the PIC16CXX architecture is essentially a single tasking microcontroller without interrupts, many applications can be developed by using a simulator program alone.

The MPSIM Simulator has the following features to assist in the debugging of software/firmware for the user.

Program Load/Save
Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display & Alter
Provisions are made to display and alter Program Memory, Register Files, and status register bits. Also simulator information such as cycle times, elapsed time, and step count can be displayed.

Utility Functions
Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use, and ASCII data.

Disassembler
Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Symbolic Debugging
The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace
During program execution, address ranges, registers, register contents, and others can be traced.

Breakpoints
The user may specify up to 512 breakpoints at any one time.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

<table>
<thead>
<tr>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DV163001</td>
<td>PICSTART-16B DEVELOPMENT SYSTEM</td>
</tr>
</tbody>
</table>
INTRODUCTION

Microchip's Total Endurance Disk provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. Now designers can describe their system to an advanced mathematical model (with a standard Windows™ interface) which will then predict the performance and reliability of the Serial EEPROM within that environment. Design trade-off analysis that formerly consumed hours, days or weeks can now be accomplished in minutes - with a level of accuracy that delivers a truly robust design.

Users may control the following parameters:
- Serial EEPROM device type
- Bytes per cycle
- Cycling mode - byte or block
- Data patterns type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application life or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level versus the number of cycles. The model is available in both DOS and Windows™ versions and offers the following additional features.

FEATURES

- IBM® PC compatibility
- Automatic or manual recalculation
- Full-screen or windowed graph view
- Hypertext on-screen help
- Key entry or slide-bar entry of parameters
- Support of Microchip's 2- and 3-wire Serial EEPROMs
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk endurance tutorial
- Real-time update of data

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- 386 or 486 processor recommended
- Math coprocessor recommended
- Windows 3.1
- 1 MB memory

Eliminate time and guesswork from your next Serial EEPROM design. Contact your local Microchip representative today and ask for the Total Endurance Disk.
INTRODUCTION

The Microchip Serial EEPROM Evaluation Board (SEEVAL™) provides system design engineers with a very efficient and economical way to evaluate and program Microchip Serial EEPROMs. The board is designed for serial (RS232) connection to most IBM®-compatible PCs, and its powerful software interface runs under Microsoft® Windows™ 3.1. Graphical representation of the EEPROM array provides fast, easy access and control of data.

Application software provided with the Serial EEPROM Evaluation Board allows the user to program special features of advanced devices such as the 24C65 Smart Serial™ and 93LCS66 write-protectable serial. It also allows the user to fill the EEPROM array using any of several methods:

1. Download data from a binary file.
2. Fill with any user-defined repeating pattern.
3. Fill with “checkerboard” or “inverse checkerboard” patterns.
4. Read or write any portion of the array down to 1 byte in size.
5. Test header for scope connections.
6. Full Read/Write cycling functions

EEProm data can also be saved to a file on disk.

Software and system debug time and overall time-to-market can be reduced by using the Serial EEPROM Evaluation Board to write known data into the array in advance or to verify data which was stored by the application itself. The package lends itself very well to quick-turn changes and rapid verification during software development and system integration.

Instructions for use and a power-supply cable are included in the Microchip Serial EEPROM Evaluation Board Kit.
Get current information and help on Microchip's Bulletin Board Service (BBS)! Microchip wants to provide you with the best responsive service possible. To accomplish this, the systems team monitors the BBS, posting the latest component data and software tool updates, providing technical help and embedded systems insights, and discussing how Microchip products provide project solutions. Extend your technical groups staff with microcontroller and memory experts through Microchip's BBS communication channel.

CONNECTING TO MICROCHIP

Connect worldwide to the Microchip BBS using the Compuserve communications network. In most cases, a local call is your only expense. The Microchip BBS connection does not use Compuserve membership services, therefore you do not need Compuserve membership to join Microchip's BBS.

The procedure to connect will vary slightly from country to country. Please check with your local Compuserve agent for details if you have a problem. Compuserve services allows multiple users at baud rates up to 9600.

To connect:
1. Set your modem to 8 bit, No parity, and One stop (8N1). This is not the normal Compuserve setting which is 7E1.
2. Dial your local Compuserve phone number.
3. Type <RETURN> and a garbage string will appear because Compuserve is expecting a 7E1 setting.
4. Type +<RETURN> and Host Name: will appear.
5. Type MCHIPBBS<RETURN> and you will be connect to the Microchip BBS.

To learn Compuserve's phone number closest to you, set your modem to 7E1, and dial (800) 848 8980, and follow Compuserve's directions. If you are dialing from overseas, you may call 614-457-1550 for voice information.

Connect without charge to the bulletin board. However, you are responsible for your phone charges. Access is available to all, but users are required to register the first time they "log in." No registration fees are required at this time.

USING THE BULLETIN BOARD

The Microchip Bulletin Board is a multi-faceted tool. Topic information includes:

- Special Interest Groups
- Files
- Mail
- Bug lists
- Technical assistance
- Consultant Directory

Special Interest Groups

Special Interest Groups, or SIGs, offer you the opportunity to discuss technical issues and topics with other users. Take advantage of the Microchip user community's broad background to glean information not available by any other method.

SIGs exists for most Microchip systems, including:
- PIC-SW
- PICMASTER
- PRO MASTER
- UTILITIES
- BUGS
- APP NOTE

These groups are moderated by Microchip staff.

Files

The Microchip Systems BBS is used regularly to distribute bug reports, history files, and interim patches for Microchip software products.

Users can contribute files for distribution on the BBS. These files will be monitored, scanned, and approved or disapproved by the SIG moderator. No executable files are accepted from the user community in general.

Mail

The BBS can be used to distribute mail to other users of the service.

This is an excellent way to get questions answered by the Microchip staff, as well as to keep in touch with fellow Microchip product users worldwide.

The BBS is an evolving product intended to serve your needs. We welcome your ideas and input. Consider mailing a message to your SIG moderator, or to the SYSOP, if you have ideas or questions about particular Microchip products, or BBS operation.
## SECTION 8

### ARTICLE REPRINTS FOR PIC16/17 MICROCONTROLLERS AND SERIAL EEPROMS

<table>
<thead>
<tr>
<th>PIC Model</th>
<th>Reprint Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16C5X</td>
<td>Lean and Mean PIC Machines</td>
<td>8-1</td>
</tr>
<tr>
<td>16C71</td>
<td>Enhanced PIC16Cxx Gets ADC and Interrupts</td>
<td>8-7</td>
</tr>
<tr>
<td>16C84</td>
<td>Microchip PIC Adds EEPROM Data Memory</td>
<td>8-8</td>
</tr>
<tr>
<td>17C42</td>
<td>Using the PIC Micro</td>
<td>8-9</td>
</tr>
<tr>
<td>17C42</td>
<td>PIC17C42 Based DC Motor Control (Japanese)</td>
<td>8-11</td>
</tr>
<tr>
<td>PICSTART-16B</td>
<td>Take your Pick with Controllers</td>
<td>8-17</td>
</tr>
<tr>
<td>Serial EEPROM</td>
<td>Microchip Technology Rewrites the EEPROM</td>
<td>8-18</td>
</tr>
<tr>
<td>Serial EEPROM</td>
<td>Serial EEPROM for Embedded Applications</td>
<td>8-19</td>
</tr>
<tr>
<td>Endurance</td>
<td>A Tool for Calculating EEROM Endurance</td>
<td>8-23</td>
</tr>
</tbody>
</table>
Lean and Mean 
PIC Machines

One of the first things an aspiring marketer learns is to choose one’s words carefully. As a junior chip peddler (er... sales consultant), I remember being told that “cheap” was a word to be avoided. Period. Pick your euphemism (“cost sensitive,” “high volume,” “cost over performance”) but for heaven’s sake, not “cheap”!

Along the same lines, simply stating your product’s features is considered amateurish. You must instead relate the “benefits” that will accrue to those customers wise enough to choose you over competitors. With apologies to Woody Allen, the ultimate sales pitch is something like:

FEATURE: Super-duper See-Thru Look-ahead Pipe-dream

BENEFIT: a) will be loved and b) will never die.

Gee, where do I sign.

Maybe I’m burned-out from the ever-escalating hype permeating the high-tech biz, but now is time for a new era of glasnost marketing. I want to see a data sheet like:

- Faster than a bat out of hell!
- Easier to use than a politician!
- Cheap!
- Cheaper!
- Cheapest!

Here are some chips from Microchip Technology that fill the bill.

PIC AND CHOOSE

Let me just say up front, continuing with my no-bull approach, the PIC 16C5x series has what are probably the cheapest micros around. I’m talking as little as $2 in volume for a complete OTP (One-Time Program; i.e., EPROM in a no-window package) micro with RAM, EPROM and I/O.

Many times, I have been faced with the need for a little piece of logic to perform some fairly mundane task many times. Usually, the choices boiled down to either wiring up a few 74xx TTLs or using one of the popular 80xx or 68xx micros.

The problem is often neither approach is ideal. The TTL-gates approach may be best for high-volume applications (or for those who agree with our illustrious leader, Steve Ciarcia, when he says, “my favorite programming language is solder’), but wire-wrapping and debugging more than a few boards quickly becomes tiresome. Furthermore, you’re faced with diving back into the rat’s nest when you change your mind (inevitably) about how it should work.

The classic 8-bit micro may be the easiest approach, but it can be expensive. A “single-chip” micro was traditionally either ROM based (i.e., minimum feasible order size is thousands) or windowed EPROM (expensive; e.g., $10 for an 8748). Another alternative is using; a “non-single-chip” micro, with low-cost external RAM and EPROM, but then I’m back to the wire-wrapping blues (or more often than I care to admit, just throwing a $100 single-board computer at what should be a $5 application).

Admittedly, recent technologies have closed the gap. On the gate side, PALs can easily consolidate 5-10 74xxs. Meanwhile, the classic 8-bit to EPROM-based single-chip micros are beginning to appear in OTP versions at lower cost.

Despite this convergence on the needs of low cost and volume applications, these approaches always leave me feeling a little uncomfortable about the waste involved because I like to squeeze every bit of functionality out of a given technology. The gates approach is often speed overkill (i.e., you don’t need a 50-MHz PAL to toggle an LED) while the micro approach is complexity overkill (e.g., an 8K-byte, 32-I/O-bit micro with 7.5K bytes of NOPs and 24 no connects). Read on to see how the PICs uniquely fill the gap between gates and regular 8-bit micros.

CHEAP RISC

Figure 1 shows the pin-outs for the two basic versions (1 8-pin and 28-pin) of the PIC. Right away the dual personality of the chip becomes apparent; it is an 8-bit micro in small gate-like package (especially the 18-pin DIP version). Why pay for more if you don’t need to?

With so few pins, explaining their function becomes blessedly easy. Note the only 1/0 difference between the 18-pin and 28-pin versions is the latter has an additional 8-bit I/O port (RC0-RC7). In addition to the general-purpose I/0 lines, MCLR* is a reset input while OSC1 and OSC2 are the CPU clock lines. The oscillator is unique because it can accept a simple RC (Resistor/Capacitor) as well as traditional crystal or TTL clock

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.
source. Save money by using the RC option if you're willing to accept less speed and accuracy as shown in Figure 2. RTCC is an input that can clock an on-chip 8-bit counter (with optional 1:2, 1:4, ..., 1:128 prescaler). Alternatively, the on-chip counter and prescaler can be driven by the internal clock divided by four (for example, 5 MHz for a 20-MHz PIC).

The RISCy nature of the PIC becomes clearer moving on-chip (Figure 3). Let me check off some of the traditional RISC criteria against the PIC.

- **Fixed Length Instructions:** Yep. Every PIC instruction is 12 bits, and different models offer 512 (16C54/55), 1K (16C56), or 2K (16C57) instruction capacity. Programming is easy because there are only 38 different instructions, which is about as reduced as possible.
- **Pipelined, Single-Cycle Execution:** OK, with the understanding that a single cycle is four clock periods (as shown in Figure 4). With a two-level pipeline (fetch and execute overlapped), the PIC's "5-MIPS" performance (at 20 MHz) leaves most other 8-bit micros eating dust.
- **Load/Store:** The PIC fulfills this most fundamental of RISC precepts, subject to a little handwaving. All PIC instructions reference "W," a File Register (FR), or both. In one interpretation, W is a type of "accumulator" while FRs (the '54, '55, and '56 offer 32 while the '57 offers 80 FRs) are "RAM." In this light, the PIC isn't Load/Store because instructions can operate directly on "memory" (the FRs). However, if you consider W as a "temporary register" and FRs as "regular registers," the instructions only work on "registers" per the Load/Store criteria. Actually, this limitation doesn’t matter because Load/Store, which addresses the issue of "slow" external memory accesses versus "fast" on-chip register becomes a rather meaningless concept when all "memory" is on-chip.
- **Harvard Architecture:** this criterion refers to the use of separate bus and memories for instructions and data, which the PIC exploits. Though not really a RISC tenet, Harvard architecture is hyped as such on complex RISC chips like the AMD 29000 and Motorola 88k.

Of course, the PIC does not have cache (another meaningless concept when all memory is on-chip), delay slots, branch prediction, speculative execution, superscalar, superpipeline, or superanything. These omissions are not surprising because the PIC doesn’t even have interrupts and it barely has a stack (two levels

<table>
<thead>
<tr>
<th>Cext</th>
<th>Next</th>
<th>Fosc @ 5.0V, 25°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>20pF</td>
<td>5.ak</td>
<td>3.33 MHz ±22%</td>
</tr>
<tr>
<td>10k</td>
<td>1.85 MHz ±22%</td>
<td></td>
</tr>
<tr>
<td>100k</td>
<td>189 kHz ±38%</td>
<td></td>
</tr>
<tr>
<td>100pF</td>
<td>5.1k</td>
<td>1.18 MHz ±15%</td>
</tr>
<tr>
<td>10k</td>
<td>668 kHz ±15%</td>
<td></td>
</tr>
<tr>
<td>100k</td>
<td>67.8 kHz ±25%</td>
<td></td>
</tr>
<tr>
<td>300pF</td>
<td>5.1k</td>
<td>46 kHz ±10%</td>
</tr>
<tr>
<td>10k</td>
<td>254 kHz ±13%</td>
<td></td>
</tr>
<tr>
<td>100k</td>
<td>25.1 kHz ±21%</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2—If lower speeds and less accuracy are acceptable tradeoffs for lower cost, a simple RC circuit may be used in place of a crystal.
Figure 3—RISC-like features of the PIC processor include fixed-length instructions; pipelined, single-cycle execution; load/store; and Harvard architecture. Other useful features include a power-on-reset timer, a watchdog timer which operates off a clock independent of the processor’s, and a sleep mode.

The Computer Applications Journal, Feb/Mar 1992, Tom Cantrell

only). That’s OK, because not only do you get what you pay for but you pay solely for what you need.

You do get some handy features like a power-on-reset timer that eliminates the external RC usually required, something that should appeal to designers who are really cheap... oops!... who are concerned about minimizing system cost. You also get a watchdog timer with a clever feature: it operates off a clock circuit separate from the CPU. This aspect is handy because the watchdog remains vigilant even if the CPU clock is stopped. Stopping the clock is something you might want to do because the PIC is “static” and includes a sleep mode. These features make very low speed, voltage, and power (i.e., battery) operation possible. For example, a PIC consumes a miserly 3 volts at 32 microamps with 32-kHz clock—wow!

CHEAP TOOLS

I tend to take up-front tool cost for granted because I have all kinds of assemblers, simulators, emulators, and programmers gathering dust. Thus, I haven’t been in the market recently, but I guess the minimum tools setup for a PAL or 8-bit micro runs at least $1000, and I know it can get much higher.

Cheap chips need cheap tools, and here Parallax Inc. fills the bill. They offer a complete PC-based development package, including assembler, “emulator,” and device programmer for only $449. You can also buy small quantities of the PIC at very decent prices (as low as $3) directly from Parallax.

I say “emulator” because it isn’t the classic emulator with expensive features like real-time trace or source-level debug. Rather, it more closely corresponds to an EPROM emulator you might use with a regular micro. The Parallax setup basically implements a downloadable RAM version of the PIC with a few extras, like power supply, switch-selectable clock source, and so forth. These features allow quick code change/debug iterations without the cost and time to burn an actual EPROM version of the chip.

The emulator/programmer setup (combined: a tidy 20 sq- in.) connects to the PC printer port on one side and
to an 18- or 28-pin DIP header on the other using a 6" ribbon cable, which plugs into the target PIC socket. Each board is powered by a small wall-mount transformer, so they can be used together or separately.

Entering a program on the PC using your favorite editor, and then assembling that program using PASM.EXE completes the developmental procedure. Next, you can use PEP.EXE to download the object code to the emulator, run the program, and after everything is working, burn a PIC using the device programmer. The latter includes LIF (low insertion force) sockets for both the 18- and 28-pin PICs.

A key point is the Parallax PASM redefines the instruction syntax, so it differs from the "official" definition by Microchip. Normally, such a move would be considered taboo, but I’m letting it pass for two reasons. First, I don’t have to worry about maintaining compatibility with vast libraries of existing official PIC code (I’ll bet you don’t either). Second, and most important, in my opinion the Parallax scheme is superior.

The Parallax improvements generally fall into two classes: more consistent instruction names and formats, and macroinstructions in which a single PASM instruction generates multiple official PIC instructions. These improvements effectively decrease the already reduced instruction set. For example, Parallax turns ten different official instructions into one generic MOV instruction. Furthermore, the Parallax macroinstructions can replace up to four official instructions.

Remember, brain cells start dying when you’re in your twenties, and the population of programmers is rapidly aging. The Parallax approach postpones the day when we’ll be forced to watch commercials with elderly hackers whimpering, "I’ve fallen—and I can’t remember all the darn instructions!"

**CHEAP THRILLS**

The "no-bull" approach calls for less talk and more action, so let me discuss an example PIC application. Figure 5 shows the schematic for a lean, mean, four-channel digital oscilloscope based on the 28-pin 16C55-HS PIC, a 64K x 4 DRAM, and a resistor network configured as a voltage-dividing DAC. Thanks to the low price of the OTP PIC, the total parts cost for the gizmo is less than $20—refreshing isn’t it?

When the Record button is pressed, the PIC captures and stores 64K samples from the four input channels in the 64K x 4 DRAM. For display, the PIC drives the resistor ladder DAC in a manner that causes an attached oscilloscope to magically show the four input traces plus a "guide" trace reflecting the position of the scope screen (64 samples) in the trace buffer (1024 screens). The DAC implementation exploits the fact that PIC I/ O lines are true CMOS, unlike those of most TTL-compatible micros (i.e., high impedance when configured as inputs and sink/source rail-to-rail when configured as outputs). The Left and Right buttons scroll the scope "screen" back and forth either a sample at a time (position designated by the small guide) or 16 screens (i.e., 1024 samples) at a time (position designated by the large guide).

The display loop starts with a sync" pulse generated by driving the DAC from 0 to 255 (i.e., 5 volts). Next, the PIC runs through a loop 64 times, outputting the guide trace voltage and then a voltage for each of the four data traces. The latter are determined by adding a voltage representing each sample value (0 or 1) and a position offset voltage corresponding to the channel number.

Three sampling modes are provided for Record. If Record is pressed by itself, the PIC simply captures 64K
samples as fast as possible. If Record is pressed and held down, + or - are pressed and released, and then Record is released, the PIC waits for the specified edge on the Trigger input channel and then records 64K samples at full speed. Finally, if Record is pressed and held down, + or - are pressed and held down, and then Record is released followed by + or - release, the PIC records a single sample each time the specified edge is detected on the Trigger input. A look at the Record portion of the code shows how the PIC determines the triggering mode and performs the data capture. Listing 1 also serves as a good introduction to PIC programming. The Record routine starts by clearing the DRAM address and a bit designating full-speed or triggered sample mode. Next, the sequence starting at: debounce (the ".." makes debounce a local label; the same label name can be used elsewhere in the program) is a loop that waits for and debounces the Record switch input. This classical PIC code uses the SETB-and CLRB instructions to set and clear bits and the SB and SNB to skip the next instruction depending on the state of a bit. It looks a little strange, but you can step through it yourself to see how it sets and clears bit (trigger mode/full-speed mode) and direction (rising or falling edge if trigger mode).

The PIC option register is written to configure the RTCC trigger input as rising or falling edge. Note again how the skip instruction makes the common chore of loading a register with a bit-dependent value easy. Similar code sets up a looping address (which is saved in temp) depending on the state of the + or - keys (regular_loop or trigger_loop).

In trigger mode, the PIC spins on the three instructions at trigger_loop waiting for the LSB of the RTCC to change from 0 to 1 (i.e., when the RTCC increments in response to the appropriate edge on the trigger input). Meanwhile, the second instruction checks whether the Record key is pressed providing a way to cancel sampling should the trigger input fail to make an appearance. The core data capture routine at regular_loop does the DRAM boogie by asserting the row address/ RAS and the column address/ CAS at which point whatever data is sitting on the four DRAM inputs is stored. The last steps
increment the address and exit the loop when 64K samples are captured. Note how the final jump instruction loops back to either trigger_loop (for sample-per-trigger mode) or regular_loop, depending on the address in W (which is saved/restored from variable temp). The core loop takes only 2.8 µs to execute (14 cycles x 4

clocks/cycle x 80 ns/clock) thanks to the fact the PIC is running at 20 MHz, thus, it acquires the data at over 350 kHz. Not bad at all. In fact, it acquires data much faster than any number of more expensive micros that shall remain nameless (lucky for them).

Oh well, the overpriced competitors have one big advantage over the PIC—at least they aren’t cheap.

Tom Cantrell holds a B.S. and an M.B.A. from UCLA. He owns and operates Microfuture Inc., and has been in Silicon Valley for ten years working on chip, board, and system design and marketing.
Enhanced PIC16Cxx gets ADC and interrupts

Eight-bit microcontrollers (µCs) are taking over the low-cost, down-in-the-dirt, embedded-system world. Low-cost small pinouts, critical peripherals, and easy design-in are requirements in this tough design environment. Microchip Technology’s upgraded versions of its PIC 16Cxx 8-bit µC line, the PIC 16C71, now has additional functions that include interrupt processing, an A/D converter, and high-current I/O for driving LEDs directly.

PIC µCs combine high processor-throughput rates with small pinouts (18 to 28 pins) and simple register-based architecture. PIC µCs evolved from an I/O controller for GE mainframes and have taken on distinct RISC-like features: a small number of fixed-length instructions and a pipelined operation (2-stage). Unlike RISC, PICs have an accumulator (working register) and handle dynamic data in registers, not RAM. The 16C71 adds significant power to the PIC 16Cxx family. The instruction word is lengthened from 12 to 14 bits, which makes control easier; interrupts have been added (earlier chips required continual polling); and a 4-channel A/D converter (20 µsec) has been added, which eliminates the need for an off-chip converter.

The PIC 16C71 runs at 20 MHz and executes most instructions in one pipeline cycle-200 nsec. Branches take two cycles, or 400 nsec.

The PIC 16C71 has four interrupt sources and an 8-level hardware stack. Interrupts or subroutine calls automatically push the 13-bit program counter onto the stack and pop it off on return. (Earlier PICs had a 2-level stack that could easily overflow.)

SLEEP mode cuts power by turning off the oscillator; the A/D converter can be turned off, too. The converter does conversions while the CPU is in SLEEP mode, waking up the CPU with a conversion-complete interrupt.

A more powerful version of the PIC family, the 17C42, adds interrupts, a larger instruction set, external-memory capability, and a wider, 16-bit instruction word.

Microchip PIC 16C71 8-bit µC

- 8-bit µC, 20-MHz clock
- 35 14-bit instructions
- 200-nsec instruction execution, 2-stage pipeline
- 36-byte RAM
- 1k-word (14 bits) ROM
- 4-channel, 8-bit A/D converter (20 µsec)
- Sleep mode with A/D wake-up
- 3 to 5V (2 mA at 3V, 4 MHz)
- $3.25 (10,000) in 18-pin DIP or SOIC

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.
Microchip PIC adds EEPROM data memory

For some embedded applications, RAM and ROM don’t cut it. These applications need small chunks of writable, nonvolatile memory to hold critical information. Microchip’s PIC 16C84 extends the 8-bit µC architecture by adding 64 bytes of 10msec write EEPROM data memory.

You can use this EEPROM to store changeable key data, such as security codes, data-input values, and interim data points. You access the EEPROM as a memory-mapped peripheral through the RAM register set. Both EEPROM address and data registers are set for a write; or, the address registers are set for a read. You control operations on the EEPROM via bits set on a hardware-control register.

The EEPROM is perfect to holding slowly changing data, and it provides data at the same read rate as the register file. EEPROM reads take 10 msec; reads take a single CPU cycle (400 nsec at a 10-MHz clock rate). The 16C84 uses a 10-MHz internal clock, which is half the rate of faster PICs.

A second-generation extensions of the PIC 16C5x 8-bit µC family, the 16C71 and 16C84 µCs have additional interrupts and wider ROM instruction word. Earlier PICs, with 2-stage execution (200 nsec/cycle, 2 cycles/instruction), lacked interrupt facilities and had a shallow hardware stack facility. The new µCs extend the ROM instruction word from 12 to 14 bits and add Sleep mode; one external and three internal (including wakeups) interrupt; and a deeper stack (from two to eight levels).

PICs µCs are popular for low-end embedded applications. Engineers use PIC µC’s fast execution cycle to compensate for minimal peripheral set. PIC’s are available with ROM, EPROM, and OTP (one-time-programmable) memory.

Microchip also sells PICMaster-16C, a development system for PIC chips, and the PIC 16C84 µC. The system functions as a real-time ICE (in-circuit emulator), monitoring a CPU probe, emulating ROM memory, and collecting execution trace data. Users run the development tool form an IBM PC under Microsoft Windows. PICMaster-16C sells for $3450.--Ray Weiss

To hold key data, the PIC 16C84 integrates a 64-byte EEPROM data memory in the low-end 8-bit µC.
Using the PIC Micro

Automotive engine carburetors and points have gone the way of vacuum tube radios and starter cranks, they have been made obsolete by advances in technology. Thanks in large part to recent advances made in cost effective single-chip microcontrollers, functions traditionally addressed by mechanical and electromechanical methods continue to be replaced by sophisticated embedded control systems. Increasing use of distributed intelligence has resulted in automobiles that provide better performance, more convenience features, a high degree of safety, better serviceability, and better long-term reliability. This article explores the advantages offered by Microchip’s PIC microcontrollers used in this field.

KEYLESS ENTRY

Although the small rf transmitter for an automotive Keyless Entry system may appear simple from the outside, it actually requires the microcontroller inside to meet specific and rather stringent requirements. These requirements include very small physical size, extremely low current drain (for long multi-year battery life), and low system cost.

One microcontroller device that has proven very successful in Keyless Entry systems is Microchip Technology’s 8-bit PIC16C54. It is available in the industry’s smallest microcontroller package, the 18 lead ssop package. The operating current drain is less than 15µA at low operating frequency, and the standby current is less than 3µA. And in high volume, the price of the 8-bit PIC16C54 is designed to offer the lowest possible system cost.

An additional capability that further sets the PIC16C54 apart from competitive devices is called Serialised Quick Turn Programming (SQTP). SQTP is a unique programming service available from Microchip for the PIC16C54, and other members of the product family. With SQTP, the eprom program memory is programmed by Microchip with the same identical software algorithm plus a security code unique to each specific device.

SQTP programming provides several important benefits to Keyless Entry systems and other security applications:

1) Due to the unique programmed security code each transmitter will work with only the specific automobile whose receiver has the identical security code. A SQTP programmed PIC16C54 (or larger member of the family) is normally used in the automobile receiver as well.

2) System cost is reduced since no external circuitry or switches are required to specify the security code.

3) Due to the excellent software code protection of the eprom based microcontroller, potential car thieves would find it virtually impossible to read out the software algorithm or unique security code.

A Low Cost ABS System Implemented on a High-End 8-Bit Microcontroller

---

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.

© 1993 Microchip Technology Inc.
In addition to the Keyless Entry systems SQTP would be useful in other automotive security and Vehicle Identification systems as well. For example, high-end auto theft deterrent systems will use SQTP to program certain critical information that would be useful to help find and identify a stolen vehicle. Other future application plans being discussed include an Automatic Vehicle Identification system that would allow an automobile to transmit its vehicle ID number automatically to roadside tool booths and security check points. In both applications the automobile would transmit information over a radio channel.

IN-CIRCUIT PROGRAMMABILITY

In some automotive applications it would be very useful if microcontrollers could be programmed with the software algorithm in the actual application circuit at the very last possible moment. The system designers of such an application would have maximum flexibility to make last minute changes to the software algorithm. Fortunately some new microcontrollers are moving in this direction.

Typically these unique microcontroller devices use eeprom (Electrically Erasable Programmable Read Only Memory) for both the program and data memories. Eeprom memories do not require special programming voltages, they can be programmed in the actual application circuit. A good example of such a device is Microchip’s recently announced low cost 8-bit PIC16C84 microcontroller. It contains 1k words of eeprom program memory and 64 bytes of data memory. The program memory can be programmed serially which also contributes to the ease of in-circuit application programming.

In-circuit programming of eeprom based microcontrollers offers several potential advantages for automotive applications:

1) As mentioned already, last minute software code changes can be made painlessly in-circuit.

2) Electronic modules can be upgraded by simply reprogramming the software algorithm. In many cases this would save the time and effort required to replace the entire electronic module or sub-system.

3) Over time, equipment parameters may change and the new parameters may need to be saved. For example, in an active suspension system the shocks may begin to wear out. To compensate, new parameters may need to be saved, under software control to the eeprom to maintain proper ride and road handling characteristics.

4) In an automobile local area network (lan) in-circuit programming could be useful in establishing addresses on the lan. Standard modules could placed as lan nodes throughout an automobile. Each node’s microcontroller could learn and permanently store its individual lan address in-circuit.

ABS

ABS designs have traditionally used 16-bit microprocessors or microcontrollers. However new high end 8-bit microcontrollers such as Microchip’s PIC17C42 now offers the high execution performance necessary while offering a considerable saving in system cost. Below is a block diagram of low an ABS brake system could be implemented with the PIC17C42.

And ABS systems must be ultra reliable and thus so must the ABS microcontroller. New high-end devices go to considerable lengths to ensure ultra reliable execution of the software algorithm. In the case of the PIC17C42, for example, these steps include:

1) A very reliable on-chip Watchdog Timer with its own independent on-chip RC timebase. The purpose of the Watchdog Timer is to reset the microcontroller should it detect improper execution of the software algorithm. The Watchdog Timer cannot be disabled through software execution.

2) Due to a rather unique Harvard Architecture, there are separate busses for Program and Data flow. There is no chance that the PIC17C42 could somehow execute Data information as if it were part of the program.

In summary a high end 8-bit microcontroller can now offer the performance necessary for most ABS applications at a considerable saving in system cost. Also the newer devices take great pains to ensure ultra reliable execution of the software algorithm.

In this article we have explored several different automobile applications. We have pointed out how some new developments in microcontroller devices have helped make possible better automobiles with low design and manufacturing costs.
要約

PIC17C42プロセッサ（以下、PIC17C42と登記する）は、埋込み型アプリケーションにおけるコストノバフォーマンスがよいサーボ・コントローラの設計に非常に適しています。PIC17C42のハーバード、RISCライク・アーキテクチャにより、サーボ・ループを高速でクローズできます。本アプリケーション・ノートは、DCモータのサーボ・コントローラとしてのPIC17C42の使い方について述べます。また、PID制御の計算は16MHzの場合360μs以内で行なえるので、制御ループのサンプル・タイムは2KHzの範囲内であることを示します。さらに、PIC17C42はベリフェラルを内蔵しているので、最小限のコストで完全なシステムを構成できます。

序文

PIC17C42は、オプティカル・エンコーダのフィードバック機能をもつ1軸のDCサーボ・コントローラの中心部として使われます。以下のサーボ・システムの例では、PIC17C42によって生成される高分解能のPWMをHブリッジ・モータ・ドライバーへの入力として使います。このシステムで、最大3MHzのエンコーダ・フィードバック信号を処理できます。図1に示すように、この制御システム全体は、PIC17C42、PLDが1個およびHブリッジのドライバー・チップで構成されます。

PICモータ制御

このようなシステムは、プリンタ、プロッタまたはスキャナのポジショニング・コントローラとして使われます。PIC17C42を使うと低いコストでコントローラを設計でき、このシステムはステッパ・モータ・システムに対して有利にでき、また、以下のように多くの利点を得られます：

![PIC17C42のブロック図](image-url)
エンコーダ・フィードバック

上記のシステムの例のポジション・フィードバックは、モータ・シャフトに取り付けられた直交エンコーダによって行ないます。位置と方向の両方の増分は、この低価格のデバイスによって符号化されます。図1に示すように、直交エンコーダの信号は、16R8系のPLDデバイスで処理されます。PLDは、直交パルスを、カウンタ・アップとカウンタ・ダウンの2つのパルス・ストリームに変換します。次に、図1に示すように、これらの信号はPIC17C42のRTCCとTMR 3の入力に送られます。

PIC17C42は、内蔵の16ビットのタイマーであるRTCCとTMR 3の2つの差分を計算することによって、モータの増分の位置に追従制御します。サーボ・サンプル・タイムごとにこの処理を行ない、増分を前の位置に加えることによってカレントの位置を計算します。タイムは両方とも16ビット幅なので、エンコーダの信号の周波数がサンプルの周波数の32,767倍より高くないかぎり、オーバーフローに注意を払う必要はありません。たとえば、サーボ・サンプル・タイムが1 mSであれば、エンコーダの最高速度は3.2767MHzになります。

2つのカウンタ間の差分のみを使うので、カウンタのラップアラウンドについては問題となりません。
2の補数の減算により、これを自動的に処理します。上記の例は、2 バイト、つまり16 ビット精度で位置を追従制御します。しかし、内部の位置レジスタの大きさには限界があります。したがって、サンプル・タイムごとに16 ビットの増分の位置をN バイトのソフトウェア・レジスタに加えることによって、N バイトの位置を保持できます。

モータの駆動

PIC17C42には、すくれたPWM（Pulse Width Modulation）サブシステムの機能があります。これを簡単なスイッチング・パワー段と組み合わせた場合、効率のよいパワーD/Aコンバータを構成できます。

PIC17C42のPWMサブシステムの分解能は62.5 nSです。これは、15.6 KHzのサンプルレートで10ビット、または20 KHzで800分の1（9.12 ビット）の分解能に変換します。これにより、変調周波数を人間に誤覚の限界以上に保ちながら、すくれた電圧制御が行えます。これは、最小限のノイズを設計の目標とするオフセット・オートメーション機能に特に適しています。

モータは、PWM出力のデューティ・サイクルを時間平均するようにPWMの出力に応答します。モータには電気的時定数があるので、ほとんどのモータはゆっくり反応し、PWM出力のデューティ・サイクルを時間平均するようにPWMの出力に応答します。また、ほとんどのモータは、0.5 mS以上の電気的時定数と20.0 mS以上の機械的時定数でゆっくり反応します。15 KHzのPWM出力は、実際はリニア・アンプ
PIC17C42でのDCサーボ・コントロール

の出力に相当します。
図1に示すシステムにおいて、HブリッジのDirection入力端子はPIC17C42のPWM出力端子に直接接続されています。HブリッジにはDC供給電圧V_mがかかっています。この構成では、PWM信号が50%のデューティサイクルのときモータには0ボルトがかかり、0%のデューティサイクルのときV_mボルト、および100%のデューティサイクルのとき+V_mボルトがかかります。

PIDアルゴリズム

PIDは、最も広く使われているサーボ・モータ・コントロールのアルゴリズムです。このコントローラがすべてのアプリケーションに最適であるとはかぎりませんが、設計しやすく、また、調整が容易です。

図2に、標準のデジタルPIDアルゴリズムの形を示します。U(k)はポジション・エラー、また、Y(k)は出力です。

図2 デジタルPIDの仕組み

このアルゴリズムは、計算の効率と数値安定度のために、標準の差分方程式としてつくられています：

\[ Y(k) = Y(k-1) + C_0 U(k) + C_1 U(k-1) + C_2 U(k-2) \]

ここで、以下のことが容易にわかります：

\[ C_0 = P+4+D \]
\[ C_1 = - (P+2D) \]
\[ C_2 = D \]

C1, C2およびC3は、オーバヘッドをできるかぎり小さくするために、サンプル・タイムの割り込みルーチン外で計算されます。
図3 アンチ・ワイド・アップ機能をもつ、計算によって改良されたPID

図3に、この差分方程式のとおりに設計されたPIDを示します。PIC17C42に必要なサイクル数は571サイクルのみで、その結果、PIDの実行時間は0.36mSになります。
積分器のワイド・アップとは、システムの中で大きなエラーが発生する場合、たとえば、大きなステップ・レスポンスが命令されたことによってPIDコントローラにおいて起こる状態です。積分器は、たとえこの出力が飽和しても、このエラーの状態の間積分し続けます。その後、積分器は、サーボ・システムが最後のデスティネーションに達し過大発振を起こすと「アンチワイド」します。この問題は、出力が飽和した場合に積分器の動作を止めることで避けられます。ここに示すPIDの設計には、単にY(k-1)を最後のサイクルでPWMタイマに実際に出力される値まで飽和させることによって、積分器の「アンチ・ワイド・アップ」を行なえるという利点があります。

サーボ・システムの構成

上記のコード例で、PIC17C42を使ってサーボ・システムを容易に構成できます。このプログラムの構成は非常に簡単です。すなわち、割り込みサービス・ルーチンがサーボ・コントロールの計算を処理し、また、フォアグラウンド・ループを使ってユーザ・インタフェース、シリアル・コミュニケーションおよびすべての例外処理（すなわち、リミット・スイッチ、ウオッチドッグ・タイマなど）を行ないます。フォアグラウンド・ループは本アプリケーション・ノートの範囲外なので、ここでは説明しません。
割り込みサービス・ルーチンの構成は簡単です。サーボ・コントロールを行うためには、すべて一定のあらかじめ設定されたレートでエンコーダを読み込み、PID制御を計算し、また、PWMの出力を設定することが必要です。これを行うために簡単な方法は、割り込みサービス・ルーチンを、PIC17C42のハードウエアのタイマのうちの1つによって開始させることです。サーボの計算が常にPWMサブシステムと同期して実行されるように、PWM2出力をTMR1とTMR2の入力ピン（TMR1を内部でクロック同期がとられる8ビットのタイマ/モードで、また、TMR2を外部でクロック同期がとられる8ビットのカウンタ/モードで）接続します。PR2レジスタにNビットをロードすると、サンプル・レートはNで割
ったPWMレートになります。
以下は、割り込みサービス・ルーチンの中で行なわれなければならない事項です:
・ タイマ (RTCCとTMR) の読み込み
・ 参照位置の更新
・ エラーの計算 U(k) = 参照位置 - カレントの位置
・ PIDを使った Y(k) の計算
・ PWM出力の設定
・ 他のハウスキピング・タスクの管理（すなわち、シリアル文字の処理）

速度を制御するために参照位置を計算する場合、サンプル・タイムごとに、単に、サンプル・タイムあたりのカウンタ数で表す速度を最後の参照位置に加えます。また、位置を制御する場合は、サンプル・タイムごとに、目的とする位置の軌道に基づいて参照位置を計算します。
サーボ・システムの構成は、上記のように非常に簡単です。

完全なシステム

PIC17C42を使って、デモンストレーションのサーボ・システムを構成しました。このシステムには、完全なRS-232インタフェース、内蔵のスイッチング電源、Hブリッジのモータ・ドライプ、オーバーカレン
ト・プロテクション、リミット・スイッチ入力およびデジタルI/Oがあります。システム全体が、5×3.5
インチのプリント回路板上に収まっています。このシステムによって、サーボ・アプリケーションにおけるPIC17C42を評価できます。
デモンストレーションのサーボ・システムにおけるすべての未使用のPIC17C42のピンは、プロトタイプ
製作時にI/Oコネクタで使えます。まもなく、完全に当社のPIC17C42のサーボ・デモンストレーション・
システムが供給されます。
NOTES:
TAKE YOUR PIC WITH CONTROLLERS

Microchip’s PICSTART-16B microcontroller development tool provides evaluation and development support for the company’s 8 bit PIC microcontroller family.

It supports the range of low-end PIC16C5X microcontrollers as well as the mid-range PIC16C71 which includes adc and the forthcoming PIC16C84 complete with eeprom.

The PICSTART-16B package includes Microchip’s MPALC Assembler, a pc based symbolic cross-assembler; the MPSIM Simulator, a discrete event simulator; and a 3 x 5 in development programmer board. Also included is a copy of Microchip’s embedded controller handbook. PICSTART-16B’s programmer board connects to a pc and accepts 18 and 28 lead otp pdip PIC16CXX devices. The kit includes software to read and program all PIC16CXX microcontroller products.

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.
Microchip Tech rewrites the E²PROM

Chandler, Ariz.—Seeing a fundamental shift in the way serial E²PROMs are used, Microchip Technology Inc. is altering the design of these previously jelly-bean parts, moving them into the high-density, smart-memory arena.

The changes mark a new product direction for Microchip and might also signal a new course for serial E²PROMs in general, along with a possible reconsideration of the role technology will play in systems design.

“Traditionally, small serial E-squareds have been used as a replacement for DIP switches,” observed Microchip senior product marketing engineer Peter Sorrells. “But that is changing.”

In many designs, the electrically erasable but non-volatile parts just store a few installation parameters or pieces of calibration data in a 1- or 2-kbit chip. The parts’ serial interface is valued because it requires little board space and only a few pins on a microcontroller. The data in E²PROM is rarely changed; moreover, such parts’ serial interface is valued because it requires little board space and only a few pins on a microcontroller. The data in E²PROM is rarely changed; moreover, such chips are never required to store large amounts of information. Consequently, customers have seen tiny packages and spare-change prices as the greatest virtue of the devices.

But Microchip claims a new generation of systems—from cellular phones to data-acquisition systems to keyless entry systems—values the non-volatility and compactness of the devices, but is much more demanding.

“In the cellular telephone market, for instance, we have talked to people who are using 64-kbit parallel E²PROMs now and are moving to serial,” Sorrells said. “They are storing hundreds of phone numbers, big configuration tables and code to support several different cellular protocols, so they need a lot of memory. But they don’t need parallel speed—a fast serial I/O scheme is sufficient.

SMART GENERATION

Microchip is responding to these needs with the first of what promises to be a new generation of smart, but not application-specific, memory parts. While retaining the familiar FC serial interface, the chips differ from conventional 1- or 2-kbit serial E-squareds in almost every other regard.

The most obvious change is size. Microchip’s 24C65 is the world’s first 64-bit serial E²PROM—four times the capacity of the largest previously announced part. This gives the user enough space not only for the DIP-switch settings, but also for storage of larger data structures and even code segments that traditionally would have ended up in a parallel EPROM or masked ROM. Systems designers have found—particularly in space-constrained handheld systems—that the ability to save a package by making the tiny serial EPROM do double duty can more than offset the part’s higher cost-per-bit.

But this means the chip will require both density and considerably greater endurance that has been typical of small E²PROMs. Microchip has addressed this issue by splitting the memory array into two sections.

One 4-kbit block of the memory uses Microchip’s fully redundant cell design to give very high endurance—typically 1 million erase/write cycles. The remaining 60 kbits use a higher density cell rated for at least 10,000 cycles. By steering information of the memory array, customers can maintain both highly volatile data and non-volatile code or data tables in the same device.

To act as a multipurpose memory, the chip will need high bus bandwidth as well as high capacity and endurance. Consequently, Microchip has augmented the FC bus to run at a minimum of 400 kHz. Sorrells claimed the interface had run considerably faster in the lab, but that 400 kHz would be the initial spec. The address format has been expanded to permit up to a 512-kbyte address range, allowing multiple chips to inhabit the buds in a single address bank.

In addition, the chip has been furnished with a 64-byte input write cache to keep the bus free during the rather long self-timed write operations. This permits a microcontroller to burst anything from an individual byte to several pages of data over the bus, and then go on with its business, addressing other chips.

In an unusual feature specifically for the instrumentation market, the write cache can be configured as a capture buffer. In this mode, the cache shifts incoming data until a signal tells to freeze and perform a write operation.

In keeping with the industry’s power fetish, the new architecture also has integral power management. Since the memory array is inherently non-volatile, Microchip has developed circuitry to completely power the array down between operations. On completion of a write or read operation, the chip goes to standby, a state in which the current is under 5 microamps. Power-up is fast enough to be transparent to the bus.

Finally, the chip is protected against incidental erasure or write operations during power hits, according to Microchip’s memory director Mitch Little.

With the architectural and circuit changes Microchip has made, Little seems confident that the part can take on and expanded role in systems designs.
SERIAL EEPROM FOR EMBEDDED APPLICATIONS

The optimum non-volatile memory technology for a system that requires small board area, byte level flexibility, low power demand and favourable costing? Consider serial EEPROMs, says Richard Fisher

Serial EEPROM technology has recently emerged as a leading non-memory solution for embedded control applications. The technology offers a number of benefits, which make it well-suited for a wide variety of applications.

For example, the small footprint, the ability to operate at low voltages and the lower power dissipation of serial EEPROM devices compared to parallel types makes them a good solution in portable systems such as cellular telephones, cameras and keyless entry systems; byte-level erase, write and read is of particular use in TV tuners; and the availability of multiple non-volatile functions in the same application offers many advantages in VCRs.

SIMPLIFIED DESIGN

A serial EEPROM requires only 10 percent of the board space required by a parallel device, significantly fewer I/O lines from the microcontroller, and draws much less current (see Fig. 1). The only significant limitation of a serial EEPROM is a lower read speed but, in many applications, this parameter is restricted more by the protocol than the hardware. For example, in two-wire I²C (Inter-Integrated Circuit) systems, large internal delays are needed to slow the part to meet the 100 kHz protocol requirements. Characterization of three-wire bus serial EEPROMs has indicated that clock frequencies of over 6 MHz can be supported.

PROTOCOL OPTIONS

Once a designer has decided to use a serial EEPROM solution, the next key step is to choose either a two- or three-wire protocol. Often, this decision is taken for reasons of familiarity, rather than based on a full appraisal of benefits.

**TABLE 1**

<table>
<thead>
<tr>
<th>3-Wire Bus Serial EEPROMs</th>
<th>2 Wire Bus Serial EEPROMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Vdd supply of &lt;2V to 5.5V</td>
<td>Single Vdd supply of &lt;2V to 5.5V</td>
</tr>
<tr>
<td>Very low current consumption</td>
<td>Very low current consumption</td>
</tr>
<tr>
<td>Reduced overall component cost</td>
<td>Reduced overall component cost</td>
</tr>
<tr>
<td>Four pins (other than Vcc &amp; GND are required for operation</td>
<td>Two pins (other than Vcc &amp; GND are required for operation</td>
</tr>
<tr>
<td>X16 bit and X8 bit data widths</td>
<td>X8 bit data width</td>
</tr>
<tr>
<td>Software WRITE Protection</td>
<td>Hardware WRITE Protection</td>
</tr>
<tr>
<td>Edge triggered clocks and signals filters for high noise immunity</td>
<td>Level triggered clocks and signals and inputs glitch</td>
</tr>
<tr>
<td>2MHz+ operation</td>
<td>I²C standard 100kHz and 400kHz protocols with a 1MHz option</td>
</tr>
<tr>
<td>Ready/Busy data polling</td>
<td>Page WRITE capability to 16 bytes</td>
</tr>
<tr>
<td>Security options available</td>
<td>Software and hardware compatible from 2k to 16k densities</td>
</tr>
<tr>
<td>Less complex protocol</td>
<td></td>
</tr>
</tbody>
</table>
A two-wire product is usually used in applications that require an I2C bus, noise immunity, or a write buffer for multiple bytes to be stored in one instruction, and where there is limited microcontroller I/O pin availability. A three-wire protocol is the preferred solution in applications that have limited protocol requirements, an SPI protocol, higher clock frequency requirements, or a x16 data width. (See Table 1).

Many serial EEPROM data sheets are written in a conventional memory data sheet format that emphasises the features of the part more than the basic operating principles. Serial EEPROMs are not conventional memories, due to the serial communications protocols involved.

Three-wire bus operation Microchip’s devices for three-wire bus operation are contained in the 93XXX family, ranging in density from 256bits to 4kbits. These devices require four pins in addition to Vcc and Gnd: CS (chip select), CLK (clock), DI (data in) and DO (data out). All 93XXX parts are hardware compatible for these four pins, although there may be compatibility issues for other pins. Software compatibility is a key issue, since there may be subtle differences in each manufacturer’s protocol. Software compatibility for density migration should also be reviewed by designers on a case-by-case basis. There is no software industry compatibility from a 256bit to 4kbit part.

Data can be organised as either x8 or x16. This selection is determined either by the ORG pin or by purchasing a standard x16 organisation. Units always power-up in an EWDS (Erase/Write Disable State). All Erase and Write functions are disabled until the EWEN (Erase/Write Enable) instruction is performed. This feature prevents accidental data corruption. An Auto-Erase cycle is performed during each Write cycle.

Each instruction set requires the following: a start bit, which is the first Data-in high signal clocked in after CS is high; two bits of opcode to identify the instruction; a number of address bits depending on device type; and data. All members of the 93XXX family have separate Data-in and Data-out pins. However, these pins may be tied together for true three-wire operation.

**TWO-WIRE BUS OPERATION**

The 24XXX and 85XXX families of devices operate in two-wire bus systems. Only the SCL (serial clock) and SDA (serial data) pins are essential for bus operation.
The other pins on the devices, WP (active high Write protection) and A0/A1/A2 (chip or clock select), are supplementary. Data is organised as x8, and signals are level triggered, not edge triggered. Also, filters on the inputs will filter noise glitches less than 100nsec wide. As with the three-wire devices, an Auto-Erase cycle is performed during each Write cycle.

**PC BUS**

One of the most popular two-wire protocols is PC, which uses master/slave bi-directional communication. A device that sends data onto the bus is defined as a transmitter and a device that is receiving data is the receiver. Both the master and the slave can act as either transmitter or receiver. The bus must be controlled by a master device (usually a microcontroller) which generates the serial clock, controls the bus direction and generates the Start and Stop conditions.

The serial EEPROM is the slave. It will be the bus transmitter during Read operations and at times when it must acknowledge data transmitted by the master. Bus activity is controlled by Start and Stop bits from the master. After a Start bit, each command must begin with an 8bit control byte. This byte must identify the serial EEPROM as the slave addressed on the bus; select the specific serial EEPROM or the internal memory block on the bus (there may be up to eight serial EEPROMs on the bus); and to select the Read or Write function for the next command transmitted by the master. With this selection scheme, devices from 2kbit to 16kbit are software compatible. For example, four 2kbit devices or one 8kbit device could be connected to the bus with the same software.

**CONCLUSION**

As the architecture of serial EEPROMs has evolved, and density has increased, these devices have become a realistic alternative in many applications that were previously restricted to using parallel non-volatile memory. Once the advantages and disadvantages of the two protocols reviewed above are understood by the designer, the serial EEPROM has much to offer.
A TOOL FOR CALCULATING EEPROM ENDURANCE

Despite the widespread use of serial EEPROMs, EEPROM endurance is still not very well defined or understood concept in the industry. We are all used to blanket claims such as ‘1 million erase/write cycles typical’, but such figures are not terribly useful - e2 endurance is dependent not only on the design of the device but also on the application environment in which it sits. Voltage, number of writes per day and the number of bytes written all have an effect.

Up until now there has been no tool available for predicting the endurance of a particular EEPROM device within a set of application parameters. Trade off analysis can be painfully time consuming and only marginally accurate without specific knowledge of the behaviour of the device under different conditions of use.

To address this problem Microchip has announced the Total Endurance Disk - a software emulator of the serial EEPROM hardware environment that allows the designer to place specific serial EEPROM devices into a theoretical application during the design and trade off stages of system development.

The Microchip Total Endurance disk allows you to trade off voltage, temperature, write cycles, number of bytes written, number of writes per day, ppm and FIT rates, as well as years of use in order to optimise the system accurately predict product lifetimes and reliability.

APPLICATION EXAMPLE

Here is an example using the Endurance Disk to help design an electronic phone book/auto dialer.

The auto-dialer may have new numbers added or changed several times per day, but how can manufacturers specify the life of the unit, and at what rate of update of the phone numbers? First the designer must make some assumptions. If we assume that the average user will change or add 50 phone numbers per day, and the manufacture is willing to live with a 0.1 percent failure rate (1,000 ppm) after 10 years of use, then we have almost enough information to verify whether we are in the ball park given the physics of the EEPROM device which will store the numbers.

We need to know the operating voltage and temperature of the applications; we will say that a 3.3V lithium button battery is powering the unit and the temperature range is limited to that for which the led display will function: 0 to 70degC. End of life voltage for the battery is approximately 2.0V; assuming that asic or microcontroller in the applications will operate down to 2.5V, the EPROM also has 2.5V requirement. The designer would like to be able to store 100 phone numbers of 16 bytes each, which results in a 1.6kbyte requirement for the serial EEPROM. Because 1.6kbytes is equal to 12.8kbits, a 16kbit 2-wire serial EEPROM will more than suffice. Specifically, Microchip’s 24LC16B will operate down to 2.5V and even includes a write-protect feature which can be used to block inadvertent writes in a noisy environment.

Here is a summary of the application:

<table>
<thead>
<tr>
<th>Device</th>
<th>Voltage</th>
<th>Temperature</th>
<th>Cycles per day</th>
<th>Bytes per cycle</th>
<th>Application life</th>
</tr>
</thead>
<tbody>
<tr>
<td>24LC16B</td>
<td>2.5V - 3.3V</td>
<td>0 to 70degC (55degC typ)</td>
<td>50</td>
<td>16</td>
<td>10 years</td>
</tr>
</tbody>
</table>

Once these values are entered into the Total Endurance program, it outputs the following:

**Device Data**

<table>
<thead>
<tr>
<th>Device</th>
<th>Input Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>24LC16B</td>
<td>24LC16B</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3V</td>
</tr>
<tr>
<td>Temperature</td>
<td>55</td>
</tr>
<tr>
<td>Bytes/Cycles per day</td>
<td>16</td>
</tr>
<tr>
<td>E/W</td>
<td>50</td>
</tr>
<tr>
<td>Application life (Years)</td>
<td>10 years</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>byte</td>
</tr>
<tr>
<td>Pulse Widths (Ms)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>Random</td>
</tr>
</tbody>
</table>

**Device Data**

<table>
<thead>
<tr>
<th>Device</th>
<th>Input Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT</td>
<td>21.0</td>
</tr>
<tr>
<td>PPM</td>
<td>1,842</td>
</tr>
<tr>
<td>Time</td>
<td>10.00</td>
</tr>
<tr>
<td>Write Cycles</td>
<td>182,500</td>
</tr>
</tbody>
</table>

Unfortunately for our designer, the desired 0.1 percent failure rate has almost doubled to 0.18 percent (1842ppm). But fortunately for the designer, the Endurance disk makes tradeoff analysis very simple and fast. At this point there are at least three options: 1)
live with almost 2000 ppm 2) look at the endurance plot and check whether there is a reasonable number of erase/write cycles which will provide a 1000 ppm failure rate or 3) specify a ppm rate to the Endurance program and let it crank out the number of cycles it will take.

You can see by reducing the number of cycles from the 182500 which resulted from our first trial to about 100,000, we can achieve a ppm rate of about 1000 (0.1 percent). But how does 100,000 cycles translate into applications life or cycles per day?

Given a 1000 ppm failure rate, you can ask for the application life of the product. Here are the results:

<table>
<thead>
<tr>
<th>Device Data</th>
<th>Input Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>24LC16B</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3V</td>
</tr>
<tr>
<td>Temperature</td>
<td>55</td>
</tr>
<tr>
<td>Bytes/Cycles per day</td>
<td>16</td>
</tr>
<tr>
<td>E/W</td>
<td>50</td>
</tr>
<tr>
<td>PPM Level (Years)</td>
<td>1000</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>byte</td>
</tr>
<tr>
<td>Pulse Widths (Ms)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>Random</td>
</tr>
</tbody>
</table>

Now we have some more options: 1) specify the product life at five years or 2) trade off other parameters of the application or 3) device which is more important - a 10 years product lifetimes or the ability to change 50 numbers every single day. Realistically a user may enter or change quite a few numbers the first week or two of the application, and after that the unit will be used mostly for reading and dialing numbers.

Changing the number of erase/write cycles to 20 a day gives us the following results:

<table>
<thead>
<tr>
<th>Device Data</th>
<th>Input Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>24LC16B</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3V</td>
</tr>
<tr>
<td>Temperature</td>
<td>55</td>
</tr>
<tr>
<td>Bytes/Cycles per day</td>
<td>16</td>
</tr>
<tr>
<td>E/W</td>
<td>20</td>
</tr>
<tr>
<td>PPM Level (Years)</td>
<td>1000</td>
</tr>
<tr>
<td>Cycling Mode</td>
<td>byte</td>
</tr>
<tr>
<td>Pulse Widths (Ms)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Pattern</td>
<td>Random</td>
</tr>
</tbody>
</table>

So reducing the number of cycles per day not only brought us back to a 10 year life, it gave some margin on that too. Keeping all the other parameters the same and forcing a 10 year lifetime gives us the following final results:

<table>
<thead>
<tr>
<th>Device Data</th>
<th>Input Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT</td>
<td>7.1</td>
</tr>
<tr>
<td>PPM</td>
<td>625</td>
</tr>
<tr>
<td>Time</td>
<td>10.00</td>
</tr>
<tr>
<td>Write Cycles</td>
<td>73,000</td>
</tr>
</tbody>
</table>

The new ppm rate of 625 gives our designer more than 30 percent margin on his ppm target of 1000.

This example shows the significant reduction in time for design tradeoff analysis and time to market which can be achieved with the Endurance Disk. In addition, it demonstrates the increase in robustness of the system design by proving known quantities and readily accessible handles to modify these quantities in the tradeoff analysis. This tool can literally reduce weeks of effort into a few minutes of point and clock.

The Total Endurance disk runs under Windows and costs $29. Users can copy plots and past them into other windows programs or print them. Microchip plans to update the disk twice a year, adding data for new serial EEPROMs as they are introduced.
APPENDIX
OFFICE LOCATIONS

Factory Representatives .......................................................... A-1-1
Distributors ............................................................................. A-2-1
Field Sales Offices ................................................................. A-3-1
Factory Representatives
Factory Representatives

ASIA

Hong Kong
Memec (Asia Pacific) Ltd.
Unit No. 2520-2525
Tower 1, Metroplaza
Hing Fong Road, Kwai Fong
N.T., Hong Kong
Tel: 852 410 2760
Fax: 852 418 1600

Korea
Memec (Asia Pacific) Ltd.
JE Woong Bldg. Third Floor
176-11 Nonhyum-Dong
Kangnam-ku
Seoul, Korea
Tel: 82 2 518 8181
Fax: 82 2 518 9419

Singapore
Memec (Asia Pacific) Ltd.
Singapore Representative Office
10 Anson Road #14-02
International Plaza
Singapore 0207
Tel: 65 222 4962
Fax: 65 222 4939

Taiwan
Memec (Asia Pacific) Ltd.
14F-1, No. 171, Sec. 5,
Min Sheng East Road
Hai Hwa Bldg.
Taipei, Taiwan
Tel: 886 2 7602028
Fax: 886 2 7651488

Ontario
Dynasty Components, Inc.
1140 Morrison Drive - Unit 110
Ottawa, Ontario K2H 8S9
Tel: 613 596 9800
Fax: 613 596 9886

Quebec
Dynasty Components, Inc.
1870 Blvd. des Sources, #304
Pointe Claire, P.Q. H9R 5N4
Tel: 514 984 5342
Fax: 514 694 6826

SOUTH AMERICA

Brazil
Aplicacoes Eletronicas Artimar Ltda.
Rua Marques De Itu, 70-10 And.
Caixa Postal 5881 e 9498
CEP 01223 - Sao Paulo, Brazil
Tel: 231 0277
Fax: 255 0511

USA

Alabama
Electramark, Inc.
500 Wynn Drive, Suite 521
Huntsville, AL 35816
Tel: 205 830 4400
Fax: 205 830 4406

Arkansas
Comptech Sales, Inc.
9810 E. 42nd Street, Suite 219
Tulsa, OK 74146
Tel: 918 622 7744
Fax: 918 660 0340

California
Trinity Technologies
1261 Oakmead Parkway
Sunnyvale, CA 94086
Tel: 408 733 9000
Fax: 408 733 9970

Competitive Technology, Inc.
200 Baha Street, Suite 101
Costa Mesa, CA 92626
Tel: 714 540 5501
Fax: 714 540 5171

Colorado
Western Region Marketing, Inc.
9176 Marshall Place
Westminster, CO 80030
Tel: 303 428 8088
Fax: 303 426 8585

Connecticut
S. J. Associates Inc.
10 Copper Ridge Circle
Guilford, CT 06437
Tel: 203 458 7558
Fax: 203 458 1181

S.J. Associates Inc.
15 Coventry Lane
Naugatuck, CT 06770
Tel: 203 723 4707
Fax: 203 723 1629

Delaware
S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609 866 1234
Fax: 609 866 8627

© 1993 Microchip Technology Inc.
### Factory Representatives

<table>
<thead>
<tr>
<th>State</th>
<th>Address</th>
<th>Telephone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Florida</strong></td>
<td>Electramark Florida, Inc. 14021-B North Dale Mabry</td>
<td>813 962 1882</td>
<td>813 961 0664</td>
</tr>
<tr>
<td></td>
<td>14021-B North Dale Mabry Tampa, FL 33618</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Electramark Florida, Inc. 401 Whooping Loop, Suite 1565 Altamonte Springs, FL 32701</td>
<td>407 830 0844</td>
<td>407 830 0847</td>
</tr>
<tr>
<td></td>
<td>Electramark Florida, Inc. 10360 NW 18 Manor Plantation, FL 33322</td>
<td>305 424 2872</td>
<td>305 452 1974</td>
</tr>
<tr>
<td><strong>Georgia</strong></td>
<td>Electramark, Inc. 6030H Unity Drive Norcross, GA 30071</td>
<td>404 446 7915</td>
<td>404 263 6389</td>
</tr>
<tr>
<td></td>
<td>6030H Unity Drive Norcross, GA 30071</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Janus Incorporated 650 E. Devon Ave. Itasca, IL 60143</td>
<td>708 250 9650</td>
<td>708 250 8761</td>
</tr>
<tr>
<td><strong>Illinois</strong></td>
<td>Electro Reps, Inc. 407 Airport North Office Park Fort Wayne, IN 46825</td>
<td>219 489 8205</td>
<td>219 489 8408</td>
</tr>
<tr>
<td></td>
<td>407 Airport North Office Park Fort Wayne, IN 46825</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Electro Reps, Inc. 7240 Shadeland Station #275 Indianapolis, IN 46256</td>
<td>317 842 7202</td>
<td>317 841 0230</td>
</tr>
<tr>
<td><strong>Indiana</strong></td>
<td>Spectrum Sales 5382 W. 95th Street Prairie Village, KS 66207</td>
<td>913 648 6811</td>
<td>913 648 6823</td>
</tr>
<tr>
<td></td>
<td>5382 W. 95th Street Prairie Village, KS 66207</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Kansas</strong></td>
<td>Electro Reps, Inc. 7240 Shadeland Station #275 Indianapolis, IN 46256</td>
<td>317 842 7202</td>
<td>317 841 0230</td>
</tr>
<tr>
<td></td>
<td>TMC Electronics 7838 Laurel Avenue Cincinnati, OH 45243</td>
<td>513 271 3860</td>
<td>513 271 6321</td>
</tr>
<tr>
<td><strong>Kentucky</strong></td>
<td>CompTech Sales, Inc. 15415 Katy Fwy., Suite 209 Houston, TX 77094</td>
<td>713 492 0005</td>
<td>713 492 6116</td>
</tr>
<tr>
<td></td>
<td>CompTech Sales, Inc. 2401 Gateway Dr., Suite 114 Irving, TX 75063</td>
<td>214 751 1181</td>
<td>214 550 8113</td>
</tr>
<tr>
<td><strong>Louisiana</strong></td>
<td>Spectrum Sales 5382 W. 95th Street Prairie Village, KS 66207</td>
<td>913 648 6811</td>
<td>913 648 6823</td>
</tr>
<tr>
<td><strong>Maryland</strong></td>
<td>Microchip Technology Inc. Hauppauge, New York</td>
<td>516 273 5305</td>
<td>516 273 5335</td>
</tr>
<tr>
<td><strong>Massachusetts</strong></td>
<td>S. J. Associates Inc. 44 Mall Road Burlington, MA 01803</td>
<td>617 272 5552</td>
<td>617 272 5515</td>
</tr>
<tr>
<td><strong>Michigan</strong></td>
<td>J.L.Montgomery Associates, Inc. 34405 W. 12 Mile Rd., Suite 149 Farmington Hills, MI 48331-5617</td>
<td>313 489 0099</td>
<td>313 489 0189</td>
</tr>
<tr>
<td></td>
<td>S-J Mid-Atlantic, Inc. 131-D Gaither Drive Mt. Laurel, NJ 08054</td>
<td>609 866 1234</td>
<td>609 866 8627</td>
</tr>
</tbody>
</table>

© 1993 Microchip Technology Inc.
# Factory Representatives

**New Mexico**  
Western High Tech Marketing, Inc.  
9414 E San Salvador, Suite 206  
Scottsdale, AZ 85258  
Tel: 505 884 2256  
Fax: 505 884 2258

**New York**  
Apex Associates, Inc.  
1210 Jefferson Rd.  
Rochester, NY 14623  
Tel: 716 272 7040  
Fax: 716 272 7756

**North Dakota**  
Comp. Technical Sales, Inc.  
6525 City West Parkway  
Eden Prairie, MN 55344  
Tel: 612 941 7181  
Fax: 612 941 4322

**Ohio**  
TMC Electronics  
7838 Laurel Avenue  
Cincinnati, OH 45243  
Tel: 513 271 3860  
Fax: 513 271 6321

**Oklahoma**  
Comptech Sales, Inc.  
18700 Woodbriar Lane  
Catoosa, OK 74015  
Tel: 918 622 7744  
Fax: 918 660 0340

**Oregon**  
Micro Sales  
1865 NW 169th Place, Suite 210  
Beaverton, OR 97006  
Tel: 503 645 2841  
Fax: 503 645 3754

**Pennsylvania**  
S-J Mid-Atlantic, Inc.  
131-D Gaither Drive  
Mt. Laurel, NJ 08054  
Tel: 609 866 1234  
Fax: 609 866 8627

**Rhode Island**  
Microchip Technology Inc.  
Five The Mountain Road  
Suite 120  
Framingham, MA 01701  
Tel: 508 820 3334  
Fax: 508 820 4326

**South Carolina**  
Zucker Associates  
4070 Barrett Drive  
Raleigh, NC 27609  
Tel: 919 782 8433  
Fax: 919 782 8476

**South Dakota**  
Comp. Technical Sales, Inc.  
6525 City West Parkway  
Eden Prairie, MN 55344  
Tel: 612 941 7181  
Fax: 612 941 4322

**Tennessee**  
Electromark, Inc.  
(913 Tennessee)  
4910 Corporate Dr., Suite J  
Huntsville, AL 35805  
Tel: 205 830 4400  
Fax: 205 830 4406

**Texas**  
TMC Sales, Inc.  
11130 Jollyville Rd., Suite 200  
Austin, TX 78759  
Tel: 512 343 0300  
Fax: 512 345 2530

<table>
<thead>
<tr>
<th>State</th>
<th>Factory Representative</th>
<th>Address 1</th>
<th>Address 2</th>
<th>Phone 1</th>
<th>Phone 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Mexico</td>
<td>Western High Tech Marketing, Inc.</td>
<td>9414 E San Salvador, Suite 206</td>
<td>Scottsdale, AZ 85258</td>
<td>Tel: 505 884 2256</td>
<td>Fax: 505 884 2258</td>
</tr>
<tr>
<td>New York</td>
<td>Apex Associates, Inc.</td>
<td>1210 Jefferson Rd.</td>
<td>Rochester, NY 14623</td>
<td>Tel: 716 272 7040</td>
<td>Fax: 716 272 7756</td>
</tr>
<tr>
<td>North Dakota</td>
<td>Comp. Technical Sales, Inc.</td>
<td>6525 City West Parkway</td>
<td>Eden Prairie, MN 55344</td>
<td>Tel: 612 941 7181</td>
<td>Fax: 612 941 4322</td>
</tr>
<tr>
<td>Ohio</td>
<td>TMC Electronics</td>
<td>7838 Laurel Avenue</td>
<td>Cincinnati, OH 45243</td>
<td>Tel: 513 271 3860</td>
<td>Fax: 513 271 6321</td>
</tr>
<tr>
<td>Oklahoma</td>
<td>Comptech Sales, Inc.</td>
<td>18700 Woodbriar Lane</td>
<td>Catoosa, OK 74015</td>
<td>Tel: 918 622 7744</td>
<td>Fax: 918 660 0340</td>
</tr>
<tr>
<td>Oregon</td>
<td>Micro Sales</td>
<td>1865 NW 169th Place, Suite 210</td>
<td>Beaverton, OR 97006</td>
<td>Tel: 503 645 2841</td>
<td>Fax: 503 645 3754</td>
</tr>
<tr>
<td>Pennsylvania</td>
<td>S-J Mid-Atlantic, Inc.</td>
<td>131-D Gaither Drive</td>
<td>Mt. Laurel, NJ 08054</td>
<td>Tel: 609 866 1234</td>
<td>Fax: 609 866 8627</td>
</tr>
<tr>
<td>Rhode Island</td>
<td>Microchip Technology Inc.</td>
<td>Five The Mountain Road</td>
<td>Suite 120</td>
<td>Framingham, MA 01701</td>
<td>Tel: 508 820 3334</td>
</tr>
<tr>
<td>South Carolina</td>
<td>Zucker Associates</td>
<td>4070 Barrett Drive</td>
<td>Raleigh, NC 27609</td>
<td>Tel: 919 782 8433</td>
<td>Fax: 919 782 8476</td>
</tr>
<tr>
<td>South Dakota</td>
<td>Comp. Technical Sales, Inc.</td>
<td>6525 City West Parkway</td>
<td>Eden Prairie, MN 55344</td>
<td>Tel: 612 941 7181</td>
<td>Fax: 612 941 4322</td>
</tr>
</tbody>
</table>
Factory Representatives

**Utah**
Western Region Marketing, Inc.
3539 S Main - Suite 210
Salt Lake City, UT 84115
Tel: 801 268 9768
Fax: 801 268 9796

**Virginia**
Delta III Associates Inc.
12605 Wilde Lake Court
Richmond, VA 23233
Tel: 804 360 7507
Fax: 804 360 5258

**Washington**
Micro Sales
2122-112th Avenue, NE
Bellevue, WA 98004-1493
Tel: 206 451 0568
Fax: 206 453 0092

**West Virginia**
TMC Electronics
7838 Laurel Avenue
Cincinnati, OH 45243
Tel: 513 271 3860
Fax: 513 271 6321

**Wisconsin**
Janus, Inc.
375 Williamstowne
Delafielld, WI 53018
Tel: 414 646 5420
Fax: 414 646 5421

**Wyoming**
Western Region Marketing, Inc.
9176 Marshall Place
Westminster, CO 80030
Tel: 303 428 8088
Fax: 303 426 8585
Distributors
### AFRICA

**South Africa**

Pace Electronic Components Pty
Cnr. Van Acht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
Tel: 11 974 1211
Fax: 11 974 1271

### ASIA

**Hong Kong**

Maxism Limited
Unit 2520-2525 Tower 1
Metroplaza, Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852 4180909
Fax: 852 4181600

**Japan**

Dainichi Contronics Inc.
Dainichi Bldg. 1-7 Karaku
1-Chome, Bunkyo-Ku
Tokyo 112, Japan
Tel: 3 3818 8050
Fax: 3 3818 8088

Marubeni Hytech Co., Ltd.
Marubeni Hytech Building
4-20-22, Koishikawa
Bunkyo-Ku
Tokyo 112, Japan
Tel: 3 817 4921
Fax: 3 817 4880

Nippon Precision Device Corp.
Nichibei Time 24 Bldg.
35 Tansu-Cho, Shinjuku-Ku
Tokyo 162, Japan
Tel: 3 3260 1411
Fax: 3 3260 7100

**Korea**

ProChips Inc.
779-12, Daelim 3-Dong
Youngdeungpo-Ku
Seoul, Korea
Tel: 02 849 8567
Fax: 02 849 8659

**Taiwan, R.O.C.**

Pinnacle Technologies Co. Ltd.
3F, 5 Lane 768, Sec. 4
Pa-Teh Road
Taipei, Taiwan
Tel: 02 788 4800
Fax: 02 788 5969

### CANADA

**Alberta**

Future Electronics
3833 - 29th Street N.E.
Calgary, Alberta, T1Y 6B5
Tel: 403 250 5550
Fax: 403 291 7054

ITT Multicomponents
3015 - 5th Ave.
Suite 210
Calgary, Alberta, T2A 6T8
Tel: 403 273 2780
Fax: 403 273 7458

Future Electronics
4606 - 97th Street
Edmonton, Alberta, T6E 5N9
Tel: 403 438 2858
Fax: 403 434 0812

**British Columbia**

Future Electronics
1695 Boundary Road
Vancouver, B.C., V5K 4X7
Tel: 604 294 1166
Fax: 604 294 1206

Hamilton Hallmark
Burnaby, B.C., V5A 4N6
Tel: 800 332 8638

ITT Multicomponents
8525 Baxter Place, Unit 101
Production Court
Burnaby, B.C., V5A 4V7
Tel: 604 421 6222
Fax: 604 421 0582

Semad Electronics
3700 Gilmore Way
Burnaby, B.C., V5G 4M1
Tel: 604 451 3444
Fax: 604 451 3445

**Manitoba**

Future Electronics
106 King Edward
Winnipeg, Manitoba, R3H ON8
Tel: 204 786 7711
Fax: 204 783 8133

ITT Multicomponents
1313 Border Street
Unit 35
Winnipeg, Manitoba, R3H 0X4
Tel: 204 697 2300
Fax: 204 697 2293

**Ontario**

Future Electronics
1050 Baxter Road
Ottawa, Ontario, K2C 3P2
Tel: 613 820 8313
Fax: 613 820 3271

Hamilton Hallmark
Nepean, Ontario, K2E 7J5
Tel: 800 332 8638
Fax: 613 596 6987

ITT Multicomponents
39 Robertson Road
Suite 506, Bell Mews
Nepean, Ontario, K2H 8R2
Tel: 613 596 6980
Fax: 613 596 6987

Semad Electronics
2781 Lancaster Road, Suite 302
Ottawa, Ontario, K1B 1A7
Tel: 613 526 4866
Fax: 613 523 4372

Future Electronics
5935 Airport Road, Suite 200
Mississauga, Ontario, L4V 1W5
Tel: 416 612 9200
Fax: 416 612 9185

Hamilton Hallmark
Mississauga, Ontario, L5T 2L1
Tel: 800 332 8638

ITT Multicomponents
300 N. Rivermede Road
Concord, Ontario, L4K 2Z4
Tel: 416 798 4884
Fax: 416 798 4889

Semad Electronics
85 Spy Court
Markham, Ontario, L3R 4Z4
Tel: 416 475 3922
Fax: 416 475 4158
### Distributors

#### Quebec
- **Hamilton Hallmark**
  - Ville St. Laurent, Quebec H4T 1V6
  - Tel: 800 332 8638
- **ITT**
  - 5713 Shemin Street Francois Ville St. Laurent, Quebec H4S 1W9
  - Tel: 514 335 7977
  - Fax: 514 335 9330
- **Semad Electronics**
  - 243 Place Frontenac, Pointe Claire, Quebec, H9R 4Z7
  - Tel: 514 694 0860
  - Fax: 514 694 0965
- **Future Electronics**
  - 1000 Ave. St. Jean Baptiste Suite 100, Quebec City, Quebec, G2E 5G5
  - Tel: 418 877 6666
  - Fax: 418 877 6671

#### ITT
- **5713 Shemin Street Francois**
  - Ville St. Laurent, Quebec H4S 1W9
  - Tel: 514 335 7977
  - Fax: 514 335 9330

### Europe

#### Austria
- **Elbatex Electronics GmbH**
  - Rotermuehlgasse 26 A-1120 Wien
  - Tel: 43 222 8135646
  - Fax: 43 222 834276
- **Elbatex Gruppe**
  - Eitnergasse 6 A1233 Wien
  - Tel: 43 1 81602 0
  - Fax: 43 1 863211 201

#### Belgium
- **Sonetech N.V.**
  - De Limburg St rumlaan 243 Bus 3 B-1780 Wemmel
  - Tel: 32 2 460 0707
  - Fax: 32 2 460 1200

### Denmark
- **Exatec A/S**
  - Mileparken 20E 2740 Skovlund
  - Tel: 45 44 92 7000
  - Fax: 45 44 92 6020

#### England
- **Farnell Electronics**
  - Canal Road
  - Leeds, LS12 2TU
  - Tel: 44 532 636311
  - Fax: 44 532 633411
- **Future Electronics Ltd.**
  - Future House Poyle Road
  - Colnbrook, Berks, SL3 0EZ
  - Tel: 44 753 687000
  - Fax: 44 753 689100
- **H.B. Electronics Ltd.**
  - Lever Street
  - Bolton, Lancashire, BL3 6BJ
  - Tel: 44 204 2 5544
  - Fax: 44 204 384911
- **Hawke Components Ltd.**
  - 26 Campbell Court
  - Bramley NR Basingstoke, Hanis, RG26 5EG
  - Tel: 44 256 880800
  - Fax: 44 256 880325

#### Germany
- **Avnet E2000 GmbH**
  - Postfach 820127 D-81801 Muenchen 82
  - Tel: 089 45110 01
  - Fax: 089 45110 210
- **Future Electronics Deutschland GmbH**
  - Munchner Strasse 18 85774 Unterfohring Munich, Germany
  - Tel: 49 89 957 1950
- **Metronik GmbH**
  - Postfach 1328 D82003 Unterhaching
  - Tel: 89 611080
  - Fax: 89 6112246
- **Semitron W. Roeck & Co.**
  - Im Gut 1
  - D79790 Kuessaberg
  - Tel: 49 7742 30010
  - Fax: 49 7742 6901

#### Greece
- **P. Caritato & Associates S.A.**
  - Ilia lliou 31
  - Athens 11743
  - Tel: 30 1 9020115
  - Fax: 30 1 9017024

#### Israel
- **Elina Electronics Ltd.**
  - 14 Raoul Wallenberg St.
  - P.O. Box 13190 Tel Aviv 61131
  - Tel: 972 3 498543
  - Fax: 972 3 498745

#### Italy
- **Eureletronica Srl**
  - Viale E. Fermi 8
  - 20090 Assago Milano
  - Tel: 39 2 457 981
  - Fax: 39 2 488 0275
- **Intesi**
  - Viale Milanofiori E/5
  - 20090 Assago Milano
  - Tel: 39 2 8247 0215
  - Fax: 39 2 8247 0278
- **Kevin**
  - Via del Gradenigo, 3
  - 20148 Milano
  - Tel: 39 2 4870 6300
  - Fax: 39 2 4870 6500

---

DS00056F-page 2 © 1993 Microchip Technology Inc.
Netherlands
Semicon B.V.
Gulberg 33
P.O. Box 258
NL-5670 AG Nuenen
Tel: 31 40 837075
Fax: 31 40 832300

Norway
Odin Elektronik AB
P.O. Box 9376 Gronlund
N0135 Oslo
Tel: 47 22 677 290
Fax: 47 677 380

Spain
Sagitron
Corazon de Maria 80/82
28002 Madrid
Tel: 34 1 416 9261
Fax: 34 1 415 8652

Sweden
MEMEC Scandinavia AB
Kvarnholmsvägen 52
131 31 Nacka
Tel: 46 8 6434190
Fax: 46 8 6431195

Switzerland
Elbatex Gruppe AG
Hardstr. 72
CH-5430 Wettingen
Tel: 41 1 56 275111
Fax: 41 1 56 275454

Turkey
Inter Muehendislik Danismanlik
Ve Ticaret A.S.1
Hasircibasi Caddesi No. 55
81310 Kadikoy
Istanbul
Tel: 90 1 349 94 00
Fax: 90 1 349 94 30

SOUTH AMERICA
Brazil
Aplicacoes Electronicas Artimar
Rue Marques de Itu,70-1- AND.
Caixa Postal 5881*9498
Cep 01223
Sao Paulo, Brazil
Tel: 55112310277
Fax: 55112550511

USA
Alabama
Future Electronics
4835 University Square, Suite 16
Huntsville, AL 35816
Tel: 205 830 2322
Fax: 205 830 6664

Arizona
Bell Industries
140 S. Linden Lane #102
Tempe, AZ 85281
Tel: 602 966 7800
Fax: 602 967 6584

California
Bell Industries
1161 N. Fairoaks Ave.
Sunnyvale, CA 94089
Tel: 408 734 8570
Fax: 408 734 8875

Isando
Pace Electronic Components Ltd.
Cnr. Vanacht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
Tel: 27 11 974 1211/6
Fax: 27 11 974 1271

Pioneer Technical Products
134 Rio Robles
San Jose, CA 95134
Tel: 408 954 9100
Fax: 408 954 9113

Future Electronics
4835 University Square #5
Huntsville, AL 35816
Tel: 205 837 9300
Fax: 205 837 9358

Future Electronics
9301 Oakdale Ave.
Suite 210
Chatsworth, CA 91311
Tel: 818 772 6240
Fax: 818 772 6247

Future Electronics
9301 Oakdale Ave.
Suite 210
Chatsworth, CA 91311
Tel: 818 772 6240
Fax: 818 772 6247
## Distributors

### California (cont.)

- **Hamilton Hallmark**  
  Wooland Hills, CA 91327  
  Tel: 800 332 8638  
- **Future Electronics**  
  1692 Browning Ave.  
  Irvine, CA 92714  
  Tel: 714 250 4141  
  Fax: 714 250 4185  
- **Pioneer Standard**  
  217 Technology Drive #110  
  Irvine, CA 92718  
  Tel: 800 535 1430  
  Fax: 714 753 5074  
- **Aegis Electronic Group, Inc.**  
  1015 Chestnut Ave. Suite G2  
  Carlsbad, CA 92008  
  Tel: 619 729 2026  
  Fax: 619 729 9295  
- **Bell Industries**  
  7827 Convoy Court, Suite 403  
  San Diego, CA 92111  
  Tel: 619 268 1277  
  Fax: 619 268 3733  
- **Future Electronics**  
  5151 Shoreham Place Suite 220  
  San Diego, CA 92122  
  Tel: 619 625 2800  
  Fax: 619 625 2810  
- **Hamilton Hallmark**  
  San Diego, CA 92123  
  Tel: 800 332 8638  

### Colorado (cont.)

- **Hamilton Hallmark**  
  Englewood, CO 80111  
  Tel: 800 332 8638  
- **Future Electronics**  
  24 Stony Hill Road  
  Bethel, CT 06801  
  Tel: 203 743 9594  
  Fax: 203 798 9745  
- **Hamilton Hallmark**  
  Cheshire, CT 06410  
  Tel: 800 332 8638  
- **Phase 1 Technology Corporation**  
  36A Padanarm Road  
  Danbury, CT 06811  
  Tel: 203 791 9042  
  Fax: 201 790 6128  
- **Pioneer Standard**  
  Two Trap Falls #101  
  Shelton, CT 06484  
  Tel: 203 929 5600  
  Fax: 203 929 9791  
- **Future Electronics**  
  4250 C Rivergreen Parkway Suite 206  
  Duluth, GA 30071  
  Tel: 404 623 1003  
  Fax: 404 623 0665  
- **Reptron Electronics**  
  3040 Business Park Drive Suite H  
  Norcross, GA 30071  
  Tel: 404 446 1300  
  Fax: 404 446 2991  

### Florida (cont.)

- **Future Electronics**  
  2200 Tall Pines Drive Suite 108  
  Largo, FL 34641  
  Tel: 813 330 1222  
  Fax: 813 358 9598  
- **Reptron Electronics**  
  14401 McCormick Drive  
  Tampa, FL 33626  
  Tel: 813 854 2351  
  Fax: 813 855 0942  
- **Reptron Electronics**  
  3320 N.W. 53rd Street  
  Ft. Lauderdale, FL 33309  
  Tel: 305 428 8877  
  Fax: 305 481 2950  
- **Hamilton Hallmark**  
  Ft. Lauderdale, FL 33309  
  Tel: 800 332 8638  
- **Future Electronics**  
  6900 Peachtree Industrial Blvd. Suite 300  
  Norcross, GA 30071  
  Tel: 404 446 1300  
  Fax: 404 446 2991  
- **Hamilton Hallmark**  
  Winter Park, FL 32792  
  Tel: 800 332 8638  

### Colorado (cont.)

- **Hamilton Hallmark**  
  Largo, FL 34641  
  Tel: 800 332 8638  
- **Reptron Electronics**  
  14401 McCormick Drive  
  Tampa, FL 33626  
  Tel: 813 854 2351  
  Fax: 813 855 0942  
- **Pioneer Standard**  
  674 S. Military Trail  
  Deerfield Beach, FL 33442  
  Tel: 305 428 8877  
  Fax: 305 481 2950  
- **Hamilton Hallmark**  
  Ft. Lauderdale, FL 33309  
  Tel: 800 332 8638  
- **Reptron Electronics**  
  3320 N.W. 53rd Street Suite 206  
  Ft. Lauderdale, FL 33309  
  Tel: 305 735 1112  
  Fax: 305 735 1121  

### Connecticut

- **Future Electronics**  
  24 Stony Hill Road  
  Bethel, CT 06801  
  Tel: 203 743 9594  
  Fax: 203 798 9745  
- **Hamilton Hallmark**  
  Cheshire, CT 06410  
  Tel: 800 332 8638  
- **Hamilton Hallmark**  
  Meriden, CT 06450  
  Tel: 800 332 8638  
- **Phase 1 Technology Corporation**  
  36A Padanarm Road  
  Danbury, CT 06811  
  Tel: 203 791 9042  
  Fax: 201 790 6128  
- **Pioneer Standard**  
  Two Trap Falls #101  
  Shelton, CT 06484  
  Tel: 203 929 5600  
  Fax: 203 929 9791  
- **Pioneer Technologies**  
  307 South-North Lake #100  
  Altamonte Springs, FL 32701  
  Tel: 407 834 9090  
  Fax: 407 834 0865  
- **Hamilton Hallmark**  
  Winter Park, FL 32792  
  Tel: 800 332 8638  

### Florida

- **Bell Industries**  
  650 S. Northlake Blvd, Suite 400  
  Altamonte Springs, FL 32701  
  Tel: 407 339 0078  
  Fax: 407 339 0139  
- **Future Electronics**  
  650 S. Northlake Blvd. Suite 520  
  Altamonte Springs, FL 32701  
  Tel: 407 767 8414  
  Fax: 407 834 9318  
- **Pioneer Technologies**  
  337 South-North Lake #100  
  Altamonte Springs, FL 32701  
  Tel: 407 834 9090  
  Fax: 407 834 0865  
- **Hamilton Hallmark**  
  Winter Park, FL 32792  
  Tel: 800 332 8638  

### Georgia

- **Hamilton Hallmark**  
  Duluth, GA 30136  
  Tel: 800 332 8638  
- **Pioneer Technologies**  
  4250 C Rivergreen Parkway  
  Duluth, GA 30016  
  Tel: 404 623 1003  
  Fax: 404 623 0665  
- **Future Electronics**  
  4960 Peachtree Industrial Blvd. Suite 300  
  Norcross, GA 30071  
  Tel: 404 446 1300  
  Fax: 404 446 2991  

© 1993 Microchip Technology Inc.
Illinois
Bell Industries
870 Cambridge Drive
Elk Grove Village, IL 60007
Tel: 708 640 1910
Fax: 708 640 0474

Future Electronics
3150 W. Higgins Rd.
Hoffman Estates, IL 60195
Tel: 708 882 1255
Fax: 708 490 9290

Hamilton Hallmark
Bensonville, IL 60106
Tel: 800 332 8638

Hamilton Hallmark
Wood Dale, IL 60191
Tel: 800 332 8638

Pioneer Standard
2171 Executive Drive #200
Addison, IL 60101
Tel: 708 495 9680
Fax: 708 495 9831

Kentucky
Hamilton Hallmark
Lexington, KY 40511
Tel: 800 332 8638

Maryland
Bell Industries
8945 Guilford Rd., Suite 130
Columbia, MD 21046
Tel: 410 290 5100
Fax: 410 290 8006

Future Electronics
6716 Alexander Bell Drive #101
Columbia, MD 21046
Tel: 410 290-0600
Fax: 410 290 0328

Hamilton Hallmark
Columbia, MD 21046
Tel: 800 332 8638

Pioneer Technologies
9100 Gaither Road
Gaithersburg, MD 20877
Tel: 301 921 0660
Fax: 301 921 4255

Vantage Components, Inc.
6925 R. Oakland Mills Road
Columbia, MD 21045
Tel: 401 720 5100
Fax: 401 381 2172

Massachusetts
Bell Industries
100 Burtt Road Suite 106
Andover, MA 01810
Tel: 508 474 8880
Fax: 508 474 8902

Future Electronics
41 Main Street
Bolton, MA 01740
Tel: 508 779 3000
Fax: 508 779 5143

Hamilton Hallmark
Peabody, MA 01960
Tel: 800 332 8638

Pioneer Standard
44 Hartwell Ave.
Lexington, MA 02173
Tel: 617 861 9200
Fax: 617 863 1547

Michigan
Future Electronics
35200 Schoolcraft Road
Suite 106
Livonia, MI 48150
Tel: 313 261 5270
Fax: 313 261 8125

Hamilton Hallmark
Novi, MI 49418
Tel: 800 332 8638

Pioneer Standard
13485 Stamford
Livonia, MI 48150
Tel: 313 525 1800
Fax: 313 427 3720

Minnesota
Digi-Key Corporation
701 Brooks Ave. So.
P.O. Box 677
Thief River Falls, MN 55744
Tel: 218 681 6674
Fax: 218 681 3380

Future Electronics
10025 Valley View Road, Suite 196
Eden Prairie, MN 55344
Tel: 612 994 2200
Fax: 612 994 2520

Hamilton Hallmark
Bloomington, MN 55431
Tel: 800 332 8638

Pioneer Standard
7625 Golden Triangle
Eden Prairie, MN 55344
Tel: 612 944 3355
Fax: 612 944 3794

Missouri
Future Electronics
12125 Woodcrest Executive Dr.
Suite 220
St. Louis, MO 63141
Tel: 314 469 6805
Fax: 314 469 7226

Hamilton Hallmark
Earth City, MO 63045
Tel: 800 332 8638

Indiana
Bell Industries
3433 E. Washington Blvd.
Fort Wayne, IN 46803
Tel: 219 422 4300
Fax: 219 423 3420

Bell Industries
5230 W. 79th St.
Indianapolis, IN 46268
Tel: 317 875 9200
Fax: 317 875 9219

Hamilton Hallmark
Indianapolis, IN 46268
Tel: 800 332 8638

Pioneer Standard
9350 N. Priority Way W. Dr.
Indianapolis, IN 46240
Tel: 317 573 0880
Fax: 317 573 0979

Kansas
Hamilton Hallmark
Lenexa, KS 66219
Tel: 800 332 8638

© 1993 Microchip Technology Inc.
## Distributors

### New Jersey

- **Bell Industries**
  - 271 Route 46 West
  - Suites F202-203
  - Fairfield, NJ 07004
  - Tel: 201 227 6060
  - Fax: 201 227 2626

- **Future Electronics**
  - 12 East Stow Road
  - Suite 200
  - Marlton, NJ 08053
  - Tel: 609 596 4080
  - Fax: 609 596 4266

- **Future Electronics**
  - 1259 Route 46 East
  - Parsippany, NJ 07054
  - Tel: 201 299 0400
  - Fax: 201 299 1377

- **Hamilton Hallmark**
  - Parsippany, NJ 07054
  - Tel: 800 332 8638

- **Hamilton Hallmark**
  - Cherry Hill, NJ 08003
  - Tel: 800 332 8638

- **Phase 1 Technology Corporation**
  - 295 Molnar Drive
  - Elmsford, NY 07407
  - Tel: 201 791 2990
  - Fax: 201 791 2552

- **Pioneer Standard**
  - 14A Madison Road
  - Fairfield, NJ 07006
  - Tel: 201 575 3510
  - Fax: 201 575 3454

- **Vantage Components, Inc.**
  - 23 Sebago Street
  - Clifton, NJ 07013
  - Tel: 201 777 4100
  - Fax: 201 777 6194

### New Mexico

- **Bell Industries**
  - 11728 Linn N.E.
  - Albuquerque, NM 87123
  - Tel: 505 292 2700
  - Fax: 505 275 2819

### New York

#### New York (cont.)

- **Phase 1 Technology Corporation**
  - 46 Jefryne Blvd.
  - Deer Park, NY 11729
  - Tel: 516 254 2600
  - Fax: 516 254 2693

- **Future Electronics**
  - 801 Motor Parkway
  - Hauppauge, NY 11788
  - Tel: 516 234 4000
  - Fax: 516 234 6183

- **Pioneer Standard**
  - 60 Crossways Park West
  - Woodbury, NY 11797
  - Tel: 516 677 1726
  - Fax: 516 921 2143

- **Vantage Components, Inc.**
  - 1056 West Jericho Turnpike
  - Smithtown, NY 11787
  - Tel: 516 543 2000
  - Fax: 516 543 2030

### North Carolina

- **Future Electronics**
  - 5225 Capital Blvd.
  - 1 North Commerce Center
  - Raleigh, NC 27604
  - Tel: 919 790 7111
  - Fax: 919 790 9022

- **Hamilton Hallmark**
  - Raleigh, NC 27604
  - Tel: 800 332 8638

- **Pioneer Technologies**
  - 2200 Gateway Centre Blvd.
  - Suite 215
  - Morrisville, NC 27560
  - Tel: 919 460 1530
  - Fax: 919 460 1540

---

© 1993 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>State</th>
<th>Distributors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ohio</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Future Electronics</td>
</tr>
<tr>
<td></td>
<td>6001-F Landerhaven Dr.</td>
</tr>
<tr>
<td></td>
<td>Mayfield Heights, OH 44124</td>
</tr>
<tr>
<td></td>
<td>Tel: 216 449 6996</td>
</tr>
<tr>
<td></td>
<td>Fax: 216 449 8987</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Solon, OH 44139</td>
</tr>
<tr>
<td></td>
<td>Tel: 800 332 8638</td>
</tr>
<tr>
<td></td>
<td>Fax: 216 587 3906</td>
</tr>
<tr>
<td></td>
<td>Bell Industries</td>
</tr>
<tr>
<td></td>
<td>444 Windsor Park Drive</td>
</tr>
<tr>
<td></td>
<td>Dayton, OH 45459</td>
</tr>
<tr>
<td></td>
<td>Tel: 513 435 8660</td>
</tr>
<tr>
<td></td>
<td>Fax: 513 435 6765</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>4433 Interpoint Blvd.</td>
</tr>
<tr>
<td></td>
<td>Dayton, OH 45424</td>
</tr>
<tr>
<td></td>
<td>Tel: 513 236 9900</td>
</tr>
<tr>
<td></td>
<td>Fax: 513 236 8133</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Worthington, OH 43085</td>
</tr>
<tr>
<td></td>
<td>Tel: 800 332 8638</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>6421 East Main #201</td>
</tr>
<tr>
<td></td>
<td>Reynoldsburg, OH 43068</td>
</tr>
<tr>
<td></td>
<td>Tel: 614 221 0043</td>
</tr>
<tr>
<td></td>
<td>Fax: 614 759 1955</td>
</tr>
<tr>
<td></td>
<td>Oklahoma</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Tulsa, OK 74146</td>
</tr>
<tr>
<td></td>
<td>Tel: 800 332 8638</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>9717 E. 42nd St.</td>
</tr>
<tr>
<td></td>
<td>Tulsa, OK 74146</td>
</tr>
<tr>
<td></td>
<td>Tel: 918 664 7840</td>
</tr>
<tr>
<td></td>
<td>Fax: 918 665 1891</td>
</tr>
<tr>
<td></td>
<td><strong>Oregon</strong></td>
</tr>
<tr>
<td></td>
<td>Future Electronics</td>
</tr>
<tr>
<td></td>
<td>15236 N.W. Greenbrier Pkwy.</td>
</tr>
<tr>
<td></td>
<td>Beaverton, OR 97006</td>
</tr>
<tr>
<td></td>
<td>Tel: 503 645 9454</td>
</tr>
<tr>
<td></td>
<td>Fax: 503 645 1559</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Dallas, TX 75244</td>
</tr>
<tr>
<td></td>
<td>Tel: 214 386 7300</td>
</tr>
<tr>
<td></td>
<td>Fax: 214 490 6419</td>
</tr>
<tr>
<td></td>
<td><strong>Pennsylvania</strong></td>
</tr>
<tr>
<td></td>
<td>Bell Industries</td>
</tr>
<tr>
<td></td>
<td>2550 Metropolitan Drive</td>
</tr>
<tr>
<td></td>
<td>Trevose, PA 19053</td>
</tr>
<tr>
<td></td>
<td>Tel: 215 953 2800</td>
</tr>
<tr>
<td></td>
<td>Fax: 215 364 4928</td>
</tr>
<tr>
<td></td>
<td>Pioneer Technologies</td>
</tr>
<tr>
<td></td>
<td>500 Enterprise Road</td>
</tr>
<tr>
<td></td>
<td>Keith Valley Business Center</td>
</tr>
<tr>
<td></td>
<td>Horsham, PA 21567</td>
</tr>
<tr>
<td></td>
<td>Tel: 215 674 4000</td>
</tr>
<tr>
<td></td>
<td>Fax: 215 674 3107</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>259 Kappa Drive</td>
</tr>
<tr>
<td></td>
<td>Pittsburgh, PA 15238</td>
</tr>
<tr>
<td></td>
<td>Tel: 412 782 2300</td>
</tr>
<tr>
<td></td>
<td>Fax: 412 963 8255</td>
</tr>
<tr>
<td></td>
<td><strong>Texas</strong></td>
</tr>
<tr>
<td></td>
<td>Bell Industries</td>
</tr>
<tr>
<td></td>
<td>1701 Greenville Ave., Suite 306</td>
</tr>
<tr>
<td></td>
<td>Richardson, TX 75081</td>
</tr>
<tr>
<td></td>
<td>Tel: 214 690 0468</td>
</tr>
<tr>
<td></td>
<td>Fax: 214 690 0467</td>
</tr>
<tr>
<td></td>
<td>Future Electronics</td>
</tr>
<tr>
<td></td>
<td>1850 N. Greenville Ave.</td>
</tr>
<tr>
<td></td>
<td>Suite 146</td>
</tr>
<tr>
<td></td>
<td>Richardson, TX 75081</td>
</tr>
<tr>
<td></td>
<td>Tel: 214 437 2437</td>
</tr>
<tr>
<td></td>
<td>Fax: 214 669 2347</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Dallas, TX 75243</td>
</tr>
<tr>
<td></td>
<td>Tel: 800 332 8638</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>13765 Beta Road</td>
</tr>
<tr>
<td></td>
<td>Dallas, TX 75244</td>
</tr>
<tr>
<td></td>
<td>Tel: 214 386 7300</td>
</tr>
<tr>
<td></td>
<td>Fax: 214 490 6419</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Austin, TX 78727</td>
</tr>
<tr>
<td></td>
<td>Tel: 800 332 8638</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>1826-D Kramer Lane</td>
</tr>
<tr>
<td></td>
<td>Austin, TX 78758</td>
</tr>
<tr>
<td></td>
<td>Tel: 512 835 4000</td>
</tr>
<tr>
<td></td>
<td>Fax: 512 835 9829</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>8200 Interstate 10 W. #705</td>
</tr>
<tr>
<td></td>
<td>San Antonio, TX 78230</td>
</tr>
<tr>
<td></td>
<td>Tel: 512 377 3440</td>
</tr>
<tr>
<td></td>
<td>Fax: 512 377 3626</td>
</tr>
<tr>
<td></td>
<td>Future Electronics</td>
</tr>
<tr>
<td></td>
<td>11271 Richmond Ave., Suite 106</td>
</tr>
<tr>
<td></td>
<td>Houston, TX 77082</td>
</tr>
<tr>
<td></td>
<td>Tel: 713 556 8696</td>
</tr>
<tr>
<td></td>
<td>Fax: 713 589 7069</td>
</tr>
<tr>
<td></td>
<td>Hamilton Hallmark</td>
</tr>
<tr>
<td></td>
<td>Houston, TX 77063</td>
</tr>
<tr>
<td></td>
<td>Tel: 800 332 8638</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>10530 Rockley Road</td>
</tr>
<tr>
<td></td>
<td>Houston, TX 77099</td>
</tr>
<tr>
<td></td>
<td>Tel: 713 495 4700</td>
</tr>
<tr>
<td></td>
<td>Fax: 713 495 5642</td>
</tr>
</tbody>
</table>
## Distributors

### Utah
- **Bell Industries**
  - Address: 6912 S. 185 West, Suite B, Midvale, UT 84047
  - Tel: 801 561 9691
  - Fax: 801 255 2477

- **Future Electronics**
  - Address: 3450 S. Highland Drive, Suite 301, Salt Lake City, UT 84106
  - Tel: 801 467 4448
  - Fax: 801 467 3604

- **Hamilton Hallmark**
  - Address: Salt Lake City, UT 84121
  - Tel: 800 332 8638

### Washington
- **Bell Industries**
  - Address: 8553,- 154th Ave. N.E., Redmond, WA 98052
  - Tel: 206 867 5410
  - Fax: 206 867 5159

- **Future Electronics**
  - Address: 19102 N. Creek Parkway South, Suite 118, Bothell, WA 98011
  - Tel: 206 489 3400
  - Fax: 206 489 3411

- **Hamilton Hallmark**
  - Address: Redmond, WA 98052
  - Tel: 800 332 8638

### Wisconsin
- **Bell Industries**
  - Address: W. 226 N. 900 Eastmound Dr., Waukesha, WI 53186
  - Tel: 414 547 8879
  - Fax: 414 547 6547

- **Future Electronics**
  - Address: 20875 Crossroads Circle, Suite 200, Waukesha, WI 53186
  - Tel: 414 786 1884
  - Fax: 414 786 0744

- **Hamilton Hallmark**
  - Address: New Berlin, WI 53146
  - Tel: 800 332 8638

- **Pioneer Standard**
  - Address: 120 Bishops Way #163, Brookfield, WI 53005
  - Tel: 414 784 3480
  - Fax: 414 784 8207
Factory Sales
JAPAN
Microchip Technology International Inc.
Shinjyokohama Gotoh Bldg. 8F, 3-22-4
Shinjyokohama, Kohoku-Ku,
Yokohama-Shi
Kanagawa 222 Japan
Tel: 81 45/471 6166
Fax: 81 45/471 6122

ASIA/PACIFIC
Microchip Technology Inc.
Unit No. 2520-2525
Tower 1, Metroplaza
Hing Fong Road, Kwai Fong
N.T., Hong Kong
Tel: 852 4102716
Fax: 852 410 2518

EUROPE
United Kingdom
Arizona Microchip Technology LTD.
Unit 3, Meadow Bank, Furlong Road
Bourne End, Bucks SL8 5A J
Tel: 44 062 885 1077
Fax: 44 062 885 0178

Germany
Arizona Microchip Technology GMBH
Alte Landstrasse 12-14
D-8012 Ottobrunn, Germany
Tel: 49 089 609 6072
Fax: 49 089 609 1997

France
Arizona Microchip Technology SARL
2, Rue Du Buisson aux Fraises
F-91300 Massy, France
Tel: 33 01 6930 9090
Fax: 33 01 6930 9079

UNITED STATES
Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786 7200
Fax: 602 899 9210

Northeast Region
Microchip Technology Inc.
Five The Mountain Road
Suite 120
Framingham, MA 01701
Tel: 508 820 3334
Fax: 508 820 4326

Mid-Atlantic Region
Microchip Technology Inc.
150 Motor Parkway
Suite 416
Hauppauge, NY 11788
Tel: 516 273 5305
Fax: 516 273 5335

Southeast Region
Microchip Technology Inc.
1521 Johnson Ferry Road NE
Suite 170
Marietta, GA 30062
Tel: 404 509 8800
Fax: 404 509 8600

North Central Region
Microchip Technology Inc.
665 Tollgate Road, Unit C
Elgin, IL 60123-9312
Tel: 708 741 0171
Fax: 708 741 0638

South Central Region
Microchip Technology Inc.
17480 N Dallas Parkway
Suite 114
Dallas, TX 75287
Tel: 214 733 0391
Fax: 214 250 4631

Northwest Region
Microchip Technology Inc.
2107 N First Street
Suite 410
San Jose, CA 95131
Tel: 408 436 7950
Fax: 408 436 7955

Southwest Region
Microchip Technology Inc.
18201 Von Karman
Suite 455
Irvine, CA 92715
Tel: 714 263 1888
Fax: 714 263 1338

© 1993 Microchip Technology Inc.