Embedded Control Handbook
Update 1995 / 1996

SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS
TO OUR VALUED CUSTOMERS:


With the 1995/1996 update, Microchip is introducing a library system of 'Volumes' and 'Updates' to PIC16/17, Non-Volatile Memory and other product application notes. Volume I will be published in the fall of 1996, replacing the existing 1994/1995 ECHB. Thereafter, updates will be published annually, providing an uninterrupted flow of current application notes for our customers' convenience and use. These updates, with revised and new application notes, will be incorporated into future volumes as appropriate.

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip product. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced. We welcome your feedback.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department at facsimile 602.917.4150.

DS00134A-page iv © 1995 Microchip Technology Inc.
# Table of Contents

## SECTION 1  MICROCHIP TECHNOLOGY INC. COMPANY PROFILE

| Company Profile | 1-1 |

## SECTION 2  PIC16/17 MICROCONTROLLER APPLICATION NOTES

| AN597 | Implementing Ultrasonic Ranging | 2-1 |
| AN600 | Air Flow Control Using Fuzzy Logic | 2-9 |
| AN606 | Low Power Design Using PIC16/17 | 2-31 |
| AN607 | Power-up Trouble Shooting | 2-47 |
| AN611 | Resistance and Capacitance Meter Using a PIC16C622 | 2-69 |
| AN615 | Clock Design Using Low Power/Cost Techniques | 2-103 |
| AN616 | Digital Signal Processing with the PIC16C74 | 2-121 |
| AN617 | Fixed Point Routines | 2-155 |

## SECTION 3  APPLICATION SPECIFIC STANDARD PRODUCTS APPLICATION NOTES

| AN599 | Energy Management Control System | 3-1 |

## SECTION 4  SERIAL EEPROM APPLICATION NOTES

| AN601 | EEPROM Endurance Tutorial | 4-1 |
| AN602 | How to get 10 Million Cycles out of your Microchip Serial EEPROM | 4-5 |
| AN603 | Continuous Improvement | 4-27 |
| AN608 | Converting to 24LCXXB and 93LCXXX Serial EEPROMs | 4-7 |
| AN609 | Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller | 4-9 |
| AN610 | Using the 24LC21 Dual Mode Serial EEPROM | 4-17 |
| AN613 | Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports | 4-23 |
| AN614 | Interfacing the 8051 with 2-wire Serial EEPROMs | 4-37 |

## SECTION 5  QUALITY AND RELIABILITY APPLICATION NOTES

| AN595 | Improving the Susceptibility of an Application to ESD | 5-1 |
| AN598 | Plastic Packaging and the Effects of Surface Mount Soldering Techniques | 5-19 |
# Table of Contents (continued)

## SECTION 6 DEVELOPMENT SYSTEMS AND SOFTWARE TOOLS

### DEVELOPMENT SYSTEMS:
- **System Support**
  - Development System Selection Chart .................................................. 6-1
- **Microchip BBS**
  - Microchip Bulletin Board Service ......................................................... 6-3
- **Design Kit**
  - Microchip Serial EEPROM Designer's Kit ........................................... 6-5
- **PICMASTER™**
  - Universal In-Circuit Emulator System with MPLAB IDE ......................... 6-7
- **PRO MATE™**
  - Universal Device Programmer ............................................................. 6-13
- **PICSEE™ Tools**
  - PICSEE Product Development Tools ................................................... 6-17
- **PICSTART™-16B1**
  - PIC16CXX Low-Cost Microcontroller Development System .................. 6-25
- **PICSTART-16C**
  - PIC16CXX Low-Cost Microcontroller Development System .................. 6-27
- **PICDEM-1**
  - Low-Cost PIC16/17 Demonstration Board ........................................... 6-29
- **PICDEM-2**
  - Low-Cost PIC16CXX Demonstration Board ........................................... 6-31

### SOFTWARE TOOLS:
- **fuzzyTECH®-MP**
  - Fuzzy Logic Development System for PIC16/17 ...................................... 6-33
- **MPASM**
  - Universal PIC16/17 Microcontroller Assembler Software ..................... 6-35
- **MPSIM**
  - PIC16/17 Microcontroller Simulator .................................................... 6-37
- **MP-C**
  - C Compiler .............................................................................................. 6-39
- **Total Endurance™**
  - Microchip Serial EEPROM Endurance Model ........................................ 6-41
- **TrueGauge™ Tools**
  - TrueGauge Intelligent Battery Management Development Tool ............ 6-43

## SECTION 7 SALES AND SERVICE LOCATIONS

- **Factory Representatives** ................................................................. 7-1
- **Distributors** ....................................................................................... 7-7
- **Factory Sales** .................................................................................... 7-19
Cross Reference Guides

CROSS REFERENCE GUIDE TO UPDATE 1995/1996 APPLICATION NOTES - ALPHABETICAL

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Design Using Low Power/Cost Techniques</td>
<td>AN615</td>
</tr>
<tr>
<td>Continuous Improvement</td>
<td>AN603</td>
</tr>
<tr>
<td>Converting to 24LCXXB and 93LCXX Serial EEPROMs</td>
<td>AN608</td>
</tr>
<tr>
<td>Digital Signal Processing with the PIC16C74</td>
<td>AN616</td>
</tr>
<tr>
<td>EEPROM Endurance Tutorial</td>
<td>AN601</td>
</tr>
<tr>
<td>Energy Management Control System</td>
<td>AN599</td>
</tr>
<tr>
<td>Fixed Point Routines</td>
<td>AN617</td>
</tr>
<tr>
<td>Air Flow Control Using Fuzzy Logic</td>
<td>AN600</td>
</tr>
<tr>
<td>How to get 10 Million Cycles out of your Microchip Serial EEPROM</td>
<td>AN602</td>
</tr>
<tr>
<td>Implementing Ultrasonic Ranging</td>
<td>AN597</td>
</tr>
<tr>
<td>Improving the Susceptibility of an Application to ESD</td>
<td>AN595</td>
</tr>
<tr>
<td>Interfacing Micrchip Serial EEPROMs to Motorola® 68HC11 Microcontroller</td>
<td>AN609</td>
</tr>
<tr>
<td>Interfacing the 8051 with 2-wire Serial EEPROMs</td>
<td>AN614</td>
</tr>
<tr>
<td>Low Power Design Using PIC16/17</td>
<td>AN606</td>
</tr>
<tr>
<td>Plastic Packaging and the Effects of Surface Mount Soldering Techniques</td>
<td>AN598</td>
</tr>
<tr>
<td>Power-up Trouble Shooting</td>
<td>AN607</td>
</tr>
<tr>
<td>Resistance and Capacitance Meter Using a PIC16C622</td>
<td>AN601</td>
</tr>
<tr>
<td>Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports</td>
<td>AN613</td>
</tr>
<tr>
<td>Using the 24LC21 Dual Mode Serial EEPROM</td>
<td>AN610</td>
</tr>
</tbody>
</table>

CROSS REFERENCE GUIDE TO UPDATE 1995/1996 APPLICATION NOTES - NUMERICAL

<table>
<thead>
<tr>
<th>Page</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1</td>
<td>AN595 Improving the Susceptibility of an Application to ESD</td>
</tr>
<tr>
<td>2-1</td>
<td>AN597 Implementing Ultrasonic Ranging</td>
</tr>
<tr>
<td>5-19</td>
<td>AN598 Plastic Packaging and the Effects of Surface Mount Soldering Techniques</td>
</tr>
<tr>
<td>3-1</td>
<td>AN599 Energy Management Control System</td>
</tr>
<tr>
<td>2-9</td>
<td>AN600 Air Flow Control Using Fuzzy Logic</td>
</tr>
<tr>
<td>4-1</td>
<td>AN601 EEPROM Endurance Tutorial</td>
</tr>
<tr>
<td>4-5</td>
<td>AN602 How to get 10 Million Cycles out of your Microchip Serial EEPROM</td>
</tr>
<tr>
<td>4-27</td>
<td>AN603 Continuous Improvement</td>
</tr>
<tr>
<td>2-31</td>
<td>AN606 Low Power Design Using PIC16/17</td>
</tr>
<tr>
<td>2-47</td>
<td>AN607 Power-up Trouble Shooting</td>
</tr>
<tr>
<td>4-7</td>
<td>AN608 Converting to 24LCXXB and 93LCXX Serial EEPROMs</td>
</tr>
<tr>
<td>4-9</td>
<td>AN609 Interfacing Micrchip Serial EEPROMs to Motorola® 68HC11 Microcontroller</td>
</tr>
<tr>
<td>4-17</td>
<td>AN610 Using the 24LC21 Dual Mode Serial EEPROM</td>
</tr>
<tr>
<td>2-69</td>
<td>AN611 Resistance and Capacitance Meter Using a PIC16C622</td>
</tr>
<tr>
<td>4-23</td>
<td>AN613 Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports</td>
</tr>
<tr>
<td>4-37</td>
<td>AN614 Interfacing the 8051 with 2-wire Serial EEPROMs</td>
</tr>
<tr>
<td>2-103</td>
<td>AN615 Clock Design Using Low Power/Cost Techniques</td>
</tr>
<tr>
<td>2-121</td>
<td>AN616 Digital Signal Processing with the PIC16C74</td>
</tr>
<tr>
<td>2-155</td>
<td>AN617 Fixed Point Routines</td>
</tr>
<tr>
<td>Subject</td>
<td>Application Notes</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Clocks</td>
<td>AN615</td>
</tr>
<tr>
<td>Digital Signal Processing</td>
<td>AN616</td>
</tr>
<tr>
<td>EEPROM, 24 Series</td>
<td>AN608</td>
</tr>
<tr>
<td></td>
<td>AN610</td>
</tr>
<tr>
<td>EEPROM, 93 Series</td>
<td>AN608</td>
</tr>
<tr>
<td></td>
<td>AN613</td>
</tr>
<tr>
<td>EEPROM, Endurance</td>
<td>AN601</td>
</tr>
<tr>
<td></td>
<td>AN602</td>
</tr>
<tr>
<td></td>
<td>AN603</td>
</tr>
<tr>
<td>EEPROM, Interfacing</td>
<td>AN609</td>
</tr>
<tr>
<td></td>
<td>AN613</td>
</tr>
<tr>
<td></td>
<td>AN614</td>
</tr>
<tr>
<td>Energy Management</td>
<td>AN599</td>
</tr>
<tr>
<td>ESD</td>
<td>AN595</td>
</tr>
<tr>
<td>Fixed Point Routines</td>
<td>AN617</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>AN600</td>
</tr>
<tr>
<td>Improvement</td>
<td>AN603</td>
</tr>
<tr>
<td>Low Power Design</td>
<td>AN606</td>
</tr>
<tr>
<td>Math</td>
<td>AN615</td>
</tr>
<tr>
<td>PIC16/17</td>
<td>AN617</td>
</tr>
<tr>
<td>PIC16C622</td>
<td>AN607</td>
</tr>
<tr>
<td>PIC16C74</td>
<td>AN611</td>
</tr>
<tr>
<td>RC Meter</td>
<td>AN611</td>
</tr>
<tr>
<td>Surface Mount Soldering</td>
<td>AN598</td>
</tr>
<tr>
<td>Ultrasonic Ranging</td>
<td>AN597</td>
</tr>
</tbody>
</table>
Cross Reference Guides

**ADDITIONAL APPLICATION NOTES AVAILABLE FROM MICROCHIP**

The following is a list of application notes that are available in the Microchip Technology Inc. *1994/1995 Embedded Control Handbook*. Please see your local Microchip Sales Representative, Distributor or Sales Office for the latest copy (order number DS00092C).

<table>
<thead>
<tr>
<th>Application Note</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN510</td>
<td>Implementation of an Asynchronous Serial I/O</td>
</tr>
<tr>
<td>AN511</td>
<td>PLD Replacement</td>
</tr>
<tr>
<td>AN512</td>
<td>Implementing Ohmmeter/Temperature Sensor</td>
</tr>
<tr>
<td>AN513</td>
<td>Analog to Digital Conversion</td>
</tr>
<tr>
<td>AN514</td>
<td>Software Interrupt Techniques</td>
</tr>
<tr>
<td>AN515</td>
<td>Communicating with the I²C Bus Using the PIC16C5X</td>
</tr>
<tr>
<td>AN517</td>
<td>24C01A Compatibility Issues and Its Mobility for Memory Upgrade</td>
</tr>
<tr>
<td>AN519</td>
<td>Implementing a Simple Serial Mouse Controller</td>
</tr>
<tr>
<td>AN520</td>
<td>A Comparison of Low End 8-Bit Microcontrollers</td>
</tr>
<tr>
<td>AN521</td>
<td>Interfacing to AC Power Lines</td>
</tr>
<tr>
<td>AN522</td>
<td>Power-Up Considerations</td>
</tr>
<tr>
<td>AN526</td>
<td>PIC16C5X/16CXX Math Utility Routines</td>
</tr>
<tr>
<td>AN527</td>
<td>Software Stack Management</td>
</tr>
<tr>
<td>AN528</td>
<td>Implementing Wake-Up on Key Stroke</td>
</tr>
<tr>
<td>AN529</td>
<td>Multiplexing LED Drive and a 4x4 Keypad Sampling</td>
</tr>
<tr>
<td>AN530</td>
<td>Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers</td>
</tr>
<tr>
<td>AN531</td>
<td>Intelligent Remote Positioner (Motor Control)</td>
</tr>
<tr>
<td>AN532</td>
<td>Servo Control of a DC-Brush Motor</td>
</tr>
<tr>
<td>AN534</td>
<td>Saving and Restoring Status on Interrupt (Implementing a Parameter Stack)</td>
</tr>
<tr>
<td>AN535</td>
<td>Logic Powered Serial EEPROMs</td>
</tr>
<tr>
<td>AN536</td>
<td>Basic Serial EEPROM Operation</td>
</tr>
<tr>
<td>AN537</td>
<td>Everything a System Engineer Needs to Know About Serial EEPROM Endurance</td>
</tr>
<tr>
<td>AN538</td>
<td>Using PWM to Generate Analog Output</td>
</tr>
<tr>
<td>AN539</td>
<td>Frequency and Resolution Options for PWM Outputs</td>
</tr>
<tr>
<td>AN540</td>
<td>Implementing IIR Digital Filters</td>
</tr>
<tr>
<td>AN541</td>
<td>Using a PIC16C5X as a Smart I²C Peripheral</td>
</tr>
<tr>
<td>AN542</td>
<td>Implementation of Fast Fourier Transforms</td>
</tr>
<tr>
<td>AN543</td>
<td>Tone Generation</td>
</tr>
<tr>
<td>AN544</td>
<td>Math Utility Routines</td>
</tr>
<tr>
<td>AN545</td>
<td>Using the Capture Module</td>
</tr>
<tr>
<td>AN546</td>
<td>Using the Analog to Digital Converter</td>
</tr>
<tr>
<td>AN547</td>
<td>Serial Port Utilities</td>
</tr>
<tr>
<td>AN548</td>
<td>Implementing Table Read and Table Write</td>
</tr>
<tr>
<td>AN550</td>
<td>1.8 Volt Technology - Benefits</td>
</tr>
<tr>
<td>AN551</td>
<td>Serial EEPROM Solutions vs. Parallel Solutions</td>
</tr>
<tr>
<td>AN552</td>
<td>Implementing Wake Up on Keystroke</td>
</tr>
<tr>
<td>AN554</td>
<td>Software Implementation of I²C Bus Master</td>
</tr>
<tr>
<td>AN555</td>
<td>Software Implementation of Asynchronous Serial I/O</td>
</tr>
<tr>
<td>AN556</td>
<td>Implementing a Table Read</td>
</tr>
<tr>
<td>AN557</td>
<td>Four Channel Digital Volt Meter with Display and Keyboard</td>
</tr>
</tbody>
</table>
# Cross Reference Guides

| AN558   | Using the 24XX65 and 24XX32 with Stand Alone PIC16C54 Code |
| AN559   | Optimizing Serial Bus Operations with Proper Write Cycle Times |
| AN560   | Using the 93LC56 and 93LC66 |
| AN562   | Using the Microchip Endurance Predictive Software |
| AN563   | Using PIC16C5X Microcontrollers as LCD Drivers |
| AN564   | Using the PWM |
| AN566   | Using the PortB Interrupt on Change as an External Interrupt |
| AN567   | Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54 |
| AN569   | Hardware and Software Resolution For a Pointing Device |
| AN570   | TrueGauge™ Calibrating the MTA11200 System |
| AN571   | Communicating with EEPROM in MTA85XXX |
| AN572   | Questions and Answers Concerning Serial EEPROMs |
| AN575   | IEEE 754 Compliant Floating-Point Routines |
| AN576   | Techniques to Disable Global Interrupts |
| AN577   | PIC16C54A EMI Results |
| AN578   | Use of the SSP Module in the i²C Multi-Master Environment |
| AN579   | Using the 8-Bit Parallel Slave Port |
| AN580   | Using Timer1 in Asynchronous Clock Mode |
| AN581   | Implementing Long Calls |
| AN582   | Low Power Real-Time Clock |
| AN583   | Implementation of the Data Encryption Standard Using PIC17C42 |
| AN584   | PICMASTER™ Support of Microsoft® Windows® DDE |
| AN585   | A Real-Time Operating System for PIC16/17 |
| AN586   | Macros for Page and Bank Switching |
| AN587   | Interfacing to an LCD Module |
| AN588   | PIC16/17 Oscillator Design Guide |
| AN589   | A PC-Based Development Programmer for the PIC16C84 |
| AN590   | A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs |
| AN591   | Apple® Desktop Bus (ADB™) |
| AN592   | Frequency Counter Using PIC16C5X |
| AN593   | Serial Port Routines Without Using the RTCC |
| AN594   | Using the CCP Modules |
INTRODUCTION TO THE EMBEDDED CONTROL SOLUTIONS COMPANY™

Microchip Technology's mission is to offer industry leading 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: 8-bit PIC16/17 microcontrollers and Serial EEPROMS. These product lines provide the solutions to many of the problems facing designers of embedded control systems.

We publish the Microchip Data Books and Embedded Control Handbook to assist our customers, existing and new, in their efforts to design and produce state-of-the-art embedded control systems.

HIGHLIGHTS

Inside Microchip Technology you'll find:

• A focus on providing high-performance, cost-effective, field-programmable embedded control solutions
• An experienced executive team focused on innovation and committed to listening to our customers
• 8-bit RISC field-programmable microcontrollers and supporting logic products
• Serial and Parallel EEPROMs and EPROMs

• A variety of end-user Application-Specific Standard Products
• Fully integrated manufacturing capabilities
• A global network of manufacturing and customer support facilities
• A unique corporate culture dedicated to continuous improvement
• Distributor network support worldwide including certified distribution FAEs

BUSINESS SCOPE

Microchip Technology Inc. manufactures and markets a variety of VLSI CMOS semiconductor components to support the market for cost-effective embedded control solutions. In particular, the company specializes in highly integrated, field-programmable RISC microcontrollers, application-specific standard products and related non-volatile memory products to meet growing market requirements for high performance, yet economical embedded control capability in products. Microchip's products feature the industry's most economical OTP (One-Time-Programmable) EPROM, reprogrammable EEPROM, and ROM 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.

Chandler, Arizona:
Company headquarters near Phoenix, Arizona; executive offices, R&D, and wafer fabrication occupy this 242,000 square-foot multi-building facility.

Tempe, Arizona:
Microchip's 170,000 square-foot wafer fabrication facility provides increased manufacturing capacity today and for the future.
Microchip Technology Incorporated is a leading supplier of field-programmable embedded control solutions by providing RISC microcontrollers and related non-volatile memory products. In order to contribute to the ongoing success of customers, shareholders and employees, our mission is to focus resources on high-value, high-quality products and to continuously improve all aspects of our business, providing a competitive return on investment.

- Guiding Values -

Customers Are Our Focus: We establish successful customer partnerships by exceeding customer expectations for products, services and attitude. We start by listening to our customers, earning our credibility by producing quality products, delivering comprehensive services and meeting commitments. 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 promoting employee teamwork, productivity, creativity, pride in work, trust, integrity, fairness, involvement, development and empowerment. 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 competitive and comprehensive employee benefits.

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

Total Cycle Times Are Optimized: We focus resources to optimize cycle times to our internal and external 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 And Growth Provide For Everything We Do: We strive to generate and maintain competitive rates of company profits and growth as they allow continued investment for the future, enhanced employee opportunity and represent the overall success of Microchip.

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

Suppliers, Representatives, And Distributors Are Our Partners: We strive to maintain professional and 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.
MARKET FOCUS

Microchip targets selected markets where our advanced designs, progressive process technology, and industry-leading product performance enable us to deliver decidedly superior performance. The company 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 which are found throughout the consumer, automotive, telecommunication, office automation and industrial control markets.

FULLY INTEGRATED MANUFACTURING

Microchip delivers fast turnaround and consistent quality through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities wholly-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 resulted in high and consistent yields which have positioned Microchip as a quality leader in its global 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 design and technology advancement facility is located in Chandler, Arizona. Product and technology development is located here, along with front-end wafer fabrication and wafer probe and sort.

In 1994, Microchip purchased a second wafer fabrication facility in Tempe, Arizona – thirteen miles from its Chandler, Arizona, headquarters. The additional 170,000 square foot facility meets the increased production requirements of a growing customer base, and provides production capacity which more than doubles that of Chandler. Assembly and test facilities, predominantly located in the Philippine Islands, Kaohsiung, Taiwan, and Bangkok, Thailand, house the technology and assembly and test equipment necessary for modern plastic and ceramic packaging.

Sales and application offices are located in key cities throughout the Americas, Asia/Pacific, Japan and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical and business support.

EMBEDDED CONTROL OVERVIEW

Unlike "processor" applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are buried inside the application. The user of the product is only concerned with the very top-level user interface (such as keypads, displays and high-level commands). Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in all market segments: consumer, commercial, PC peripherals, telecommunications (including fast-emerging personal telecommunication products), automotive and industrial. Most often embedded control products must meet special requirements: cost-effectiveness, low power, small footprint, and a high level of system integration.

Typically, most embedded control systems are designed around a microcontroller which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems also usually require complementary Serial EEPROM memories, display drivers, keypads or small displays.

Microchip Technology has established itself as a leading supplier of field-programmable embedded control solutions. The combination of high-performance microcontrollers from the PIC17CXX, PIC16CXX and PIC16C5X families, along with industry leading non-volatile memory products, provide the basis for this leadership.

Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.
MICROCONTROLLERS

PIC16/17 microcontrollers from Microchip combine high performance, low cost, and small package size, offering the best price/performance ratio in the industry. More than 200 million of these devices have been used in cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, and security and telecommunication applications.

PIC16/17 MICROCONTROLLER OVERVIEW AND ROADMAP

Microchip offers three families of 8-bit microcontrollers to best fit your needs:

- PIC16C5X: Base-Line 8-bit Family
- PIC16CXX: Mid-Range 8-bit Family
- PIC17CXX: High-End 8-bit Family

All families offer One-Time-Programmable, low-voltage and low-power options, as well as various packaging options. Selected members are available in ROM and reprogrammable versions.

The widely-accepted PIC16C5X, 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.

PIC16C5X: BASE-LINE FAMILY

PIC16C5X is the well-established base-line family offering the most cost-effective solution. These PIC16C5X products have a 12-bit wide instruction set and are currently offered in 18-, 20- or 28-pin packages. In the SOIC and SSOP packaging options, these are the smallest footprint controllers. Low-voltage operation down to 2.0V makes this family ideal for battery operated applications.

PIC16CXX: MID-RANGE FAMILY

PIC16CXX mid-range family offers a wide-range of options, from 18-pin to 44-pin packages as well as low to high levels of peripheral integration. This family has a 14-bit wide instruction set, interrupt handling capability and a deeper 8-level hardware stack. The PIC16CXX family provides the performance and versatility to meet the requirements of more demanding, yet cost-sensitive, mid-range 8-bit applications.

The PIC16CXX mid-range family is rapidly gaining acceptance with several of its members introduced: PIC16C520, PIC16C521, PIC16C622, PIC16C61, PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C71, PIC16C73, PIC16C74 and PIC16C84.

PIC17CXX: HIGH-END FAMILY

The PIC17CXX high-end family offers the world's fastest execution performance of any 8-bit microcontroller family in the industry. 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. A powerful array of precise on-chip peripheral features provide the performance for the most demanding 8-bit applications.

All three members of the PIC17CXX family have been announced and are available in production.

Current PIC16/17 microcontroller product families include advanced features such as sophisticated timers, embedded Analog-to-Digital converters, extended instruction/data memory, inter-processor communication (I²C™ bus, SPI and USARTs) and ROM, RAM, EPROM and EEPROM memories.

All three families; PIC16C5X, PIC16CXX and PIC17CXX, are supported by user-friendly development systems including; assembler, software simulator, C Compiler, fuzzy logic development software, programmers and in-circuit emulators.

---

<table>
<thead>
<tr>
<th>Performance</th>
<th>ROM</th>
<th>EPROM</th>
<th>EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC17CXX</td>
<td>PIC17C4X</td>
<td>PIC17C8X</td>
<td>High-End 16-Bit Instructions</td>
</tr>
<tr>
<td>PIC16CXX</td>
<td>PIC16C8X</td>
<td>PIC16C7X</td>
<td>Mid-Range 14-Bit Instructions</td>
</tr>
<tr>
<td>PIC16C5X</td>
<td>PIC16C5X</td>
<td>PIC16C5X</td>
<td>Base-Line 12-Bit Instructions</td>
</tr>
</tbody>
</table>

Onboard Memory Technology

CMOS PIC16/17
Microcontroller Families

*In development
FIGURE 1: PIC16/17 MICROCONTROLLER MIGRATION PATH

FIGURE 2: PIC16/17 SYNERGISTIC DEVELOPMENT TOOLS

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Name</th>
<th>PIC16C5X</th>
<th>PIC16CXX</th>
<th>PIC17CXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>MPASM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Software Simulator</td>
<td>MPSIM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C Compiler*</td>
<td>MP-C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Entry Level Development Kit</td>
<td>PICSTART™</td>
<td>✓</td>
<td>✓</td>
<td>Planned</td>
</tr>
<tr>
<td>Universal Programmer</td>
<td>PRO MATE™</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Universal In-Circuit Emulator</td>
<td>PICMASTER™</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fuzzy Logic Development Tool</td>
<td>fuzzyTECH®-MP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

* Available from Byte Craft Limited in Canada and supported by Microchip.

© 1995 Microchip Technology Inc.
The PIC16/17 architecture offers users a wider range of cost/performance options of any 8-bit microcontroller family. In order to identify the families, the following naming conventions have been applied to the PIC16/17 microcontrollers.

**TABLE 1: PIC16/17 NAMING CONVENTION**

<table>
<thead>
<tr>
<th>Family</th>
<th>Architectural Features</th>
<th>Name</th>
<th>Technology</th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C5X</td>
<td>Base-Line 8-bit Microcontroller Family • 12-bit wide instruction set • DC - 20 MHz clock speed • 200 ns instruction cycle</td>
<td>PIC16C5X</td>
<td>OTP program memory, digital only</td>
<td>PIC16C54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16C5XA</td>
<td></td>
<td>PIC16C54A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16CR5X</td>
<td>ROM program memory, digital only</td>
<td>PIC16CR54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16CR5XA</td>
<td></td>
<td>PIC16CR54A</td>
</tr>
<tr>
<td>PIC16C6X</td>
<td>Mid-Range 8-bit Microcontroller Family • 14-bit wide instruction set • Internal/external interrupts • DC - 20 MHz clock speed (Note 3) • 200 ns instruction cycle (@ 20 MHz)</td>
<td>PIC16C6X</td>
<td>OTP program memory, digital</td>
<td>PIC16C61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16CR6X</td>
<td>ROM program memory, digital only</td>
<td>Planned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16C62X</td>
<td>OTP program memory with comparators</td>
<td>PIC16C62A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16C7X</td>
<td>OTP program memory, with analog functions (i.e., A/D)</td>
<td>PIC16C71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16C8X</td>
<td>EEPROM program and data memory</td>
<td>PIC16C84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC16CR8X</td>
<td>ROM program and EEPROM data memory</td>
<td>Planned</td>
</tr>
<tr>
<td>PIC17C4X</td>
<td>High-End 8-bit Microcontroller Family • 16-bit wide instruction set • Internal/external interrupts • DC - 25 MHz clock speed • 160 ns instruction cycle</td>
<td>PIC17C4X</td>
<td>OTP program memory, digital only</td>
<td>PIC17C42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIC17CR4X</td>
<td>ROM program memory, digital only</td>
<td>Planned</td>
</tr>
</tbody>
</table>

*Note 1: "A" designates a more advanced process technology, generally offering customers the benefits of lower power, higher speed, etc. (example: PIC16C54, PIC16C54A). Sometimes it designates additional functions such as the addition of Brown-out Detect.

*Note 2: The numbering system within each family is not necessarily significant.

*Note 3: The maximum clock speed for some devices is less than 20 MHz.

Please check with your local Microchip distributor, sales representative or sales office for the latest product information.
FIGURE 3: PIC16/17 8-BIT MICROCONTROLLER FAMILY

- **PIC17C4X:**
  - OTP Program Memory

- **PIC16C8X:**
  - EEPROM Program and Data Memory

- **PIC16C7X:**
  - OTP Program Memory with Analog

- **PIC16C62X:**
  - OTP Program Memory with Comparators

- **PIC16C6X OR PIC16CR6X:**
  - OTP or ROM Program Memory

- **PIC16C5X/5XA:**
  - OTP Program Memory

- **PIC16CR5X/5XA:**
  - ROM Program Memory
THE ADVANTAGE OF FIELD PROGRAMMABILITY

The PIC16/17 microcontroller family provides a unique combination of a high-performance RISC processor with cost-effective One-Time-Programmable (OTP) technology. Cost-effective OTP provides many benefits to the user at prices which can be comparable to competing ROM solutions. The benefits include: 1) quick time-to-market, 2) ease of code changes, 3) ability to provide adaptable solutions to end-customer requirements, 4) ability to meet upside potential via inventory positions at Microchip or worldwide distribution, 5) reduced scrap in manufacturing, 6) reduced inventory in manufacturing, and 7) reduced work-in-process liability.

For most manufacturers, getting the product to market quickly has become the number one goal as global markets have become more competitive. Time-to-market puts pressure on all functions within the manufacturing process: development, purchasing, production, marketing and sales. Field-programmable OTP technology streamlines the process for all stages in the product life cycle.

In the early product development stages, a programmable microcontroller allows much of the functionality to be implemented in software which can be modified more easily than hardware-only solutions. In the manufacturing stage, the compression of the product life cycle curve puts pressure on the management of inventory and manufacturing cycle times. Minimizing inventory reduces the ability to meet upside demand. Using a traditional ROM-based microcontroller limits the ability to respond to the market with product enhancements or semi-customized products for specific customers. Using the standard OTP-based PIC16/17 microcontroller solves all these issues. Inventory can be managed effectively by using the same device in several systems. Costs can be reduced due to volume purchasing. Upsides can be met from either safety stock, directly from Microchip, or local distributors who regularly inventory all PIC16/17 microcontroller devices. A sudden decline in demand means no work-in-process ROM-based inventory and any excess safety stock can be consumed by the other products using the same standard device.

OTP is the 'Flexible Manufacturing' technology of the microcontroller world. As competition intensifies, the demand for customer-specific products increases. Having the ability to change (for example, the appearance of LCD displays or add extra features in a timely manner) can be a key competitive advantage. Programming the OTP device on the manufacturing floor allows simple customization and internal tracking of the devices for each specific customer. Customization can significantly increase the overall product life cycle to provide better return on investment and help minimize the threat of competition.

DEVELOPMENT SYSTEMS

Microchip is committed to providing useful and innovative solutions to your embedded system designs. Among support products offered are the PICMASTER™ Real-Time Universal In-circuit Emulator running under the Windows® environment. PICMASTER is designed to provide product development engineers with an optimized design tool for developing target applications. This universal in-circuit emulator provides a complete microcontroller design tool set for all microcontrollers in the PIC16C5X, PIC16CXX and PIC17CXX families. PRO MATE™, the full-featured device programmer, enables you to quickly and easily program user software into PIC16C5X, PIC16CXX and PIC17CXX CMOS microcontrollers. The PRO MATE operates as a stand-alone unit or in conjunction with a PC compatible host system. The PICSTART™ development kit, a low-cost development system for the PIC16C5X/16CXX families of microcontrollers, includes an assembler for code development, a simulator for debug, and a development programmer board. PICSEEKIT and PICSEESTART provide product development engineers with a cost-effective and timely design tool solution for the MTA8XXXX family of ASSP products.

The Serial EEPROM Designer's Kit includes everything necessary to read, write, erase, or program special features of any Microchip Serial EEPROM product including Smart Serial™ and secure serials. The Total Endurance™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

The TrueGauge™ development tool supports system development with the MTA11200 TrueGauge Intelligent Battery Management IC.
SOFTWARE SUPPORT

Microchip’s PIC16/17 microcontroller families are supported by an assembler, compiler, software simulator and fuzzy logic development software. MPASM is a universal macro assembler supporting Microchip’s entire product line of microcontrollers. MPSIM, a discrete event software simulator, is designed to imitate operation of PIC16C5X, PIC16CXX and PIC17CXX microcontrollers. It allows the user to debug software that will use any of these microcontrollers.

A full-featured C-Compiler and Fuzzy Logic tools are also available for all three microcontroller families.

Microchip endeavors at all times to provide the best service and responsiveness possible to its customers. The Microchip Systems Bulletin Board Service (BBS) is one service to facilitate this service. It’s a multi-faceted tool that can provide you with information on a number of different topics.

The Microchip Internet Home Page can provide you with technical information, application notes and promotional news on Microchip products and technology. The Microchip Web address is http://www.mchip.com/biz/mchip.

Special Interest Groups available through the BBS can provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. The BBS is regularly used to distribute technical information, application notes, source code, errata sheets, bug reports, interim patches for Microchip systems products, and user contributed files for distribution. Please see Microchip BBS connection information (Section 6, Page 6-3).

APPLICATION-SPECIFIC STANDARD PRODUCTS (ASSPs)

Microchip’s Application-Specific Standard Products (ASSP) provide value-added embedded control solutions by combining PIC16/17 microcontroller architecture, non-volatile memory, and innovative software technology for vertical applications. These products incorporate technology that offers a complete solution that is both unique to the customer and standard in manufacture to Microchip. In addition, Microchip ASSPs reduce or remove the barriers for customers to use Microchip solutions, in their products, through the use of software, embedded in secure OTP- or ROM-based microcontrollers. These microcontrollers are packaged to provide the highest integration, to the customer, at the best overall system cost.

The MTA11200 family is the most accurate and most integrated battery management and charging solution available today. The TrueGauge family incorporates Microchip/SPAN patented technology which digitally integrates battery charge and discharge current to provide an accurate (~97% typical) state of charge indication. The family operates with NiCd and NiMH and lead acid battery packs from 3 VDC to 25 VDC. These products are ideal for portable PC, cellular phone, and portable consumer product applications.

The MTA14000 programmable Intelligent Battery Management IC allows engineers to design intelligent controllers for smart batteries, battery chargers, battery status monitoring, uninterruptible power supplies, HVAC, and other data acquisition and processing required for managing energy. The MTA14000’s programmable 4K words of program memory and 192 bytes of RAM allows it to support any battery technology including Li Ion, NiMH, NiCd, Pb acid, Zinc Air. In addition, the product’s I²C™ port enables any system OLM, battery pack VAR, and battery manufacturer to design, build, and market SBD-compliant products supporting the System Management Bus standard.

The MTE1122 Energy Management Controller combines Microchip’s proprietary PIC16/17 8-bit RISC microcontroller technology with a unique, patent pending power management firmware algorithm in a single package. This device, by monitoring and controlling the supply requirements into an AC induction motor, effectively reduces the power consumed by the motor. The MTE1122 is available in both plastic DIP and space-saving SOIC packages, and operates over commercial and industrial ranges.

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 MTA8XXXX PICSEE™ family of cost-effective system solutions integrates PIC16/17 microcontrollers with EEPROM technology. These PICSEE devices are ideally suited for automotive security, keyless entry, remote control, telecommunication applications and data acquisition. 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.
SERIAL EEPROM OVERVIEW

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available 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 breakthrough "AA" family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCd battery powered applications.

Temperature Ranges:
Like all Microchip devices, Serial EEPROMs are offered in Commercial (0°C to 70°C), Industrial (-40°C to 85°C) and Automotive (-40°C to 125°C) operating temperature ranges.

Packages:
The focus is on small packages. Small footprint packages include: 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths, and 14-lead SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Technology Leadership:
Microchip's Serial EEPROMs are backed by a 10 million Erase/Write cycle guarantee — an endurance breakthrough unmatched by its competitors. Microchip's erase/write cycle endurance is among the best in the world, and only Microchip offers such unique and powerful development tools as the Total Endurance disk. This mathematical software model is an innovative tool used by system designers to optimize Serial EEPROM performance and reliability within the application.

The Company has also developed the world's first 64K Smart Serial EEPROM which provides four times the speed, four times the memory, and four times the features of any competitive 2-wire Serial EEPROM. Device densities range from 256 bits up to 64K bits. Another first is the 24LC21, the only single chip DDC1/DDC2™-compatible solution for plug-and-play video monitors.

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 300 million units have been produced. Microchip continues to develop new Serial EEPROM solutions for embedded control applications.
PARALLEL EEPROM OVERVIEW

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 typical. Data retention is more than 10 years. Fast write times are less than 200 µs. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from -40°C to +85°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, telecommunications and pattern recognition.

OTP EPROM OVERVIEW

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. High-speed EPROMs have access times as low as 55 ns. Typical applications include computer peripherals, 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 a leading supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.
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 products prior to delivery to the system manufacturing operation. PIC16/17, EPROM and Serial EEPROM products can be automatically programmed, with the user's program, during the final stages of the test operation at Microchip's assembly and test operations in the Philippine Islands, Taiwan and Thailand. 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 one- to four-week lead times on QTP products, 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 products, ensures the off-the-shelf availability of program memory space that can be programmed with a unique id, 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 is 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, pioneered by Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.

FUTURE PRODUCTS AND TECHNOLOGY

New process technology is constantly being developed for microcontroller, ASSP, EEPROM, and high-speed EPROM products. Advanced process technology modules and products are being developed that will be integrated into present product lines to continue to achieve a range of compatible processes. Current production technology utilizes lithography dimensions down to 0.9 microns. Products using 0.7 microns technology are in development.

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 online. Cycle times for new technology development are continuously reduced by using in-house mask generation, 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 co-operation with our many customer-partners worldwide.
SECTION 2
PIC16/17 MICROCONTROLLER
APPLICATION NOTES

AN597  Implementing Ultrasonic Ranging ................................................................. 2-1
AN600  Air Flow Control Using Fuzzy Logic ............................................................... 2-9
AN606  Low Power Design Using PIC16/17 ............................................................... 2-31
AN607  Power-up Trouble Shooting ............................................................................ 2-47
AN611  Resistance and Capacitance Meter Using a PIC16C622 ............................... 2-69
AN615  Clock Design Using Low Power/Cost Techniques ......................................... 2-103
AN616  Digital Signal Processing with the PIC16C74 .............................................. 2-121
AN617  Fixed Point Routines ...................................................................................... 2-155
Appendix A: Algorithms ......................................................................................... 2-161
Appendix B: Flowcharts ......................................................................................... 2-162
Appendix C: Multiply Routines for the PIC16C5X/PIC16CXX ............................... 2-164
Appendix D: PIC16C5X/PIC16CXX Divide Routines .............................................. 2-254
Appendix E: PIC17CXX Multiply Routines .............................................................. 2-348
Appendix F: PIC17CXX Divide Routines ................................................................. 2-386
INTRODUCTION

Object ranging is essential in many types of systems. One of the most popular ranging techniques is ultrasonic ranging. Ultrasonic ranging is used in a wide variety of applications including:

- Auto focus cameras
- Motion detection
- Robotics guidance
- Proximity sensing
- Object ranging

This application note describes a method of interfacing PIC16CXX microcontrollers to the Polaroid 6500 Ranging Module. This implementation uses a minimum of microcontroller resources, a CCP module and two I/O pins. The two major components of the system are:

- Microcontroller
- Polaroid 6500 Ranging Module

The microcontroller performs the intelligence and arithmetic functions for ultrasonic ranging, while the Polaroid 6500 Ranging Module performs the ultrasonic signal transmissions and echo detection.

FIGURE 1: RANGING MODULE INTERFACE

THEORY OF OPERATION

Ultrasonic ranging entails transmitting a sound wave and measuring the time that it takes for the sound wave to reflect off of an object and back to the origin. The reflection time is proportional to the distance that the object is from the source. In this implementation, the sound wave is transmitted and received from the same transducer. Therefore, a blanking interval is required between signal transmission and reception to eliminate false echoes (i.e., a transmitted signal being detected as its own echo).

CIRCUIT CONFIGURATION

In this implementation, a PIC16C74 is connected to the ranging module as shown in Figure 1. The RE0 and RE1 I/O pins are configured as digital outputs and are tied to INIT and BINH, respectively. The CCP1 pin is configured as a digital input and is tied to ECHO through a pull-up resistor. The pull-up resistor is needed since the ECHO signal is an open-collector output. The CCP1 pin is configured for capture mode (CCP1CON). Figure 2 shows the timing relationship for Vdd and the three signal lines (INIT, BINH, and ECHO).

Note: The ranging module requires 5.0 milliseconds to stabilize during power-up.
FIGURE 2: TIMING DIAGRAM OF RANGING MODULE CONTROL LINES

The PIC16C74 is configured to use one of its internal timers, Timer1, in capture mode to measure the time between signal transmission and echo detection. The resolution of the timer is determined by the microcontroller clock frequency. For this application, a 4 MHz external oscillator was used, giving a resolution of 1 ms per bit. The PIC16C74 initiates a ranging cycle by first clearing Timer1. Timer1 is then enabled and INIT is immediately asserted on the ranging module. When INIT is asserted, the ranging module transmits a series of 16 pulses on the transducer at 49.4 kHz. The transmitted pulses reflect off the object and are received back at the transducer.

The transducer is used for both transmitting and receiving sound waves. A blanking interval is needed to ensure that the transmitted signal has decayed on the transducer, in order not to receive false echoes. In normal operation, the ranging module has a blanking interval of 2.38 milliseconds, which corresponds to a minimum detection distance of approximately 17 inches. However, the BINH (blank inhibit) signal can be manipulated to reduce the blanking time on the transducer to allow for object ranging as close as 6 inches.

In this implementation, the PIC16C74 asserts the BINH signal approximately 0.9 milliseconds after signal transmission. This enables the transducer to receive reflections off objects at a distance of 6 inches. The ranging module asserts the ECHO signal when a valid reflection has been detected. The PIC16C74 uses the ECHO signal to trigger a capture of the Timer1 value. The capture register contains the 16-bit value representing the elapsed time between signal transmission and echo detection. The PIC16C74 then calculates object distance based on the Timer1 value, microcontroller clock speed, and the velocity of sound in the atmosphere. The basic equation for calculating distance is given below:

\[
\text{Distance (inches)} = \frac{\text{TECHO time}}{147.9 \text{ microseconds}}
\]

DESIGN CONSIDERATIONS

There are several design considerations which must be taken into account and are listed below.

The absolute measuring distance supported by the ranging module is 6 inches to 35 feet with an accuracy of +/- 1%.

The distance output from the ranging module can be averaged over time to filter distance calculations.

In some applications, the gain of the receiver amplifier may be too low or too high and may need to be adjusted. For example, if the transducer is mounted in a cylinder, the gain may need to be lowered to reduce false echoes within the cylinder. In this case, R1 (refer to the Polaroid Ultrasonic Ranging System manual) may be replaced with a 20 kΩ potentiometer to tweak the gain of the receiver amplifier to reduce false echoes.

In order for the Polaroid 6500 ranging module to operate properly, the power supply must be capable of handling high current transients (2.5 A) during the

Parameter | Symbol | Characteristic | Min | Typ | Max | Units
---|---|---|---|---|---|---
1 | Tpu | Ranging Module Stabilization Time | 5.0 | - | - | ms
2 | TBINH | Blank Inhibit Time | 0.9 | 2.38 | - | ms
3 | TECO | Echo Time | - | - | - | -
4 | TINIT_H | High Time for INIT | 100 | - | - | ms
5 | TINIT_L | Low Time for INIT | 100 | - | - | ms

Note: The minimum high and low time for INIT is 100 milliseconds, as seen in Figure 2.
transmit pulse. The instantaneous drain on the power supply can be mitigated by installing a storage capacitor across the power lines at the ranging module. A value of 500 microfarads is recommended.

A 200 millisecond interval is recommended between ranging cycles (Figure 2) to allow the transducer to clear.

The ECHO line requires a pull-up resistor (4.7 kΩ was used in this application).

There must be a common ground between the PIC16C74 circuitry and the ranging module.

Some applications may not need the resources of the higher end PIC16CXX devices. It is still possible to do this application using a device that does not contain a CCP module (for ECHO timing). The capture function can be implemented in firmware. The effect of a firmware implementation is that the resolution of the ECHO time would be 3 Tcy cycles versus 1 Tcy cycle for the CCP module. Also, the firmware implementation would not allow other tasks to be performed while the capture function was occurring. Refer to Appendix A for general ranging module specifications.
APPENDIX A: POLAROID MODULE SPECIFICATIONS

Note: This appendix contains general specifications from the Polaroid Ultrasonic Ranging System Manual. Please refer to the current Polaroid Ultrasonic Ranging System Manual for current information regarding ranging module design considerations.

DESIGN CONSIDERATIONS IN ULTRASONICS

Range: (with user custom designed processing electronics)

Farther
a) Use an acoustic horn to “focus” the sound (narrowing the beamwidth).
b) Use two transducers – 1 receiver and 1 transmitter – facing each other.
c) Lower the transmitting frequency (which will decrease the attenuation in air).

Closer
a) Use a shorter transmit signal (such as four cycles).
   a) Use two transducers – one to transmit, one to receive (eliminates waiting for damping time).

Resolution
a) Above all, know the target and range well, and design a system with them in mind.
b) Use a higher transmit frequency.
c) Look at phase differences of a given cycle of the transmitted signal and received echo (as opposed to using and integration technique).
d) Increase the clock frequency of the timer.

Accuracy: (again, you must have a well defined target)

Temperature Compensate
a) Use a second small target, as a reference, at a known distance in the ranging path (such as a 1/4” rod several feet away), process both echoes, then normalize the second distance with respect to the first, since t1/d1 = t2/d2.
b) Incorporate a temperature sensing integrated circuit to drive a VCO to do the distance interval clocking.
c) To increase sensitivity of detection circuit change the value of C4 from 3300 pF to 1000 pF on the 6500 Series Ranging Module.

Beam Width:

Increase
a) Use an acoustic lens (to disperse the signal).
b) Decrease the transmitting frequency.
c) Use several transducers to span an area.

Decrease
a) Use an acoustic horn (to focus an area).
b) Increase the transmitting frequency.

TABLE 1: RECOMMENDED OPERATING CONDITIONS

<table>
<thead>
<tr>
<th></th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage, Vcc</td>
<td>4.5</td>
<td>6.8</td>
<td>V</td>
</tr>
<tr>
<td>High-level input voltage, Vih</td>
<td>2.1</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Low-level input voltage, Vil</td>
<td>0.6</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>ECHO and OSC output voltage</td>
<td>6.8</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Delay time, power up to INIT high</td>
<td>5</td>
<td></td>
<td>ms</td>
</tr>
<tr>
<td>Recycle period</td>
<td>80</td>
<td></td>
<td>ms</td>
</tr>
<tr>
<td>Operating free-air temperature, Ta</td>
<td>0</td>
<td>40</td>
<td>°C</td>
</tr>
</tbody>
</table>
**TABLE 2: ELECTRICAL CHARACTERISTICS OVER RECOMMENDED RANGES OF SUPPLY VOLTAGE AND OPERATING FREE-AIR TEMPERATURE (UNLESS OTHERWISE NOTED)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test Conditions</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input current</td>
<td>BINH, INIT</td>
<td>V1 = 2.1V</td>
<td></td>
<td>1</td>
<td>mA</td>
</tr>
<tr>
<td>High-level output current, IOH</td>
<td>ECHO, OSC</td>
<td>VOH = 5.5V</td>
<td>100</td>
<td></td>
<td>μA</td>
</tr>
<tr>
<td>Low-level output voltage, VOL</td>
<td>ECHO, OSC</td>
<td>IOL = 1.6 mA</td>
<td>0.4</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Transducer bias voltage</td>
<td>TA = 25°C</td>
<td>200</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Transducer output voltage (peak-to-peak)</td>
<td>TA = 25°C</td>
<td>400</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Number of cycles for XDCR output to reach 400V</td>
<td>C = 500 pF</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal blanking interval</td>
<td></td>
<td>2.38*</td>
<td></td>
<td></td>
<td>ms</td>
</tr>
<tr>
<td>Frequency during 16-pulse transmit period</td>
<td>OSC output</td>
<td>49.4*</td>
<td></td>
<td></td>
<td>kHz</td>
</tr>
<tr>
<td>Frequency after 16 pulse transmit period</td>
<td>OSC output</td>
<td>93.3*</td>
<td></td>
<td></td>
<td>kHz</td>
</tr>
<tr>
<td>Supply current, ICC</td>
<td>During transmit period</td>
<td>2000</td>
<td></td>
<td></td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>After transmit period</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* These typical values apply for a 420 kHz ceramic resonator.
APPENDIX B: FIRMWARE LISTING

This routine continually executes ranging cycles in the following order:

1) Timers and Flags are cleared
2) Ranging Cycle Executes
3) Distance is Calculated (to 0.5 inch)
4) HW is re-initialized for next cycle

The processor uses a 4MHz oscillator, so all timing calculations are referenced to that. The calculated distance is a 16-bit result in the ACChI:ACCbLO registers.

LIST P=16C74, F=INHX8M

TMR1 is off, Prescaler is 1 for a capture timeout of 65 msec

clr TICON
Set to capture on every rising edge

movlw 0x05
movwf CCP1CON
Clear the Ports

clr PORT_A
clr PORT_B
clr PORT_C
clr PORT_D
clr PORT_E

Clear RPO

Configure CCP1 (RC2) as an input, and all other ports as Outputs, (RR0 = INIT, RE1 = BINH)

bsf STATUS,RPO ; Set RPO
Port A is Digital, Port E is Digital
movlw 0x07
movwf ADCON1

Configure CCP1 (RC2) as an input, and all other ports as Outputs, (RR0 = INIT, RE1 = BINH)

clr TRIS_A
clr TRIS_B
movlw 0x04
movwf TRIS_C

clr TRIS_D
clr TRIS_E

Clear RPO

Initialize Timers and Flags

Disable TMR1

Clear Timer1 Overflow Flag & Timer1 Capture Flag
0014 018E 0069 clear TMR1L; Clear TMR1L
0015 018F 0070 clear TMR1H; Clear TMR1H
0016 0195 0071 clear CCPR1L; Clear CCPR1L
0017 0196 0072 clear CCPR1H; Clear CCPR1H
0018 1409 0073 bsf PORT_E,0; Set INIT High on Ranging Module
0019 1410 0074 bsf TlCON,0; Enable TMR1
001A 21F3 0075 call DEL_9; Delay 0.9 msec for transducer to stabilize
001B 1489 0076 bsf PORT_E,1; Enable Transducer to Receive (BINH)
001C 0077 chk_t1
001C 190C 0078 btfscc PIR1,2; Check for Capture
001D 2822 0079 goto chk_done; Jump if Capture
001E 190C 0080 btfscc PIR1,0; Check for TMR1 Overflow
001F 282C 0081 goto chk_t1; Loop if nothing happened
0020 1010 0082 bcf TlCON,0; Turn off TMR1
0021 2833 0083 goto ovr_flo; Capture event did not occur
0022 008A chk_done
0085 ;
0086 ; Calculate distance to 0.5 inch resolution
0087 ;
0022 1010 0088 bcf TlCON,0; Turn off TMR1
0023 0815 0089 movf CCPR1L,W; Move LSB into W
0024 00A2 0090 movwf ACCbLO; Move LSB into ACCbLO
0025 0816 0091 movf CCPR1H,W; Move MSB into W
0026 00A3 0092 movwf ACCbHI; Move MSB into ACCbHI
0027 0A0A 0093 movlw 0x4A; Move 75usec/0.50in into W
0028 00A0 0094 movwf ACCaLO; Move LSB into ACCaLO
0029 01A1 0095 clrf ACCaHI; Clear MSB (ACCaHI)
002A 208F 0096 call D_divF; Call 16-bit/8-bit routine
0097 ; which is described in
0098 ; Application Note 544
002B 3025 0099 movlw 0x25; Check remainder to see if
002C 0224 0100 subwf ACCcLO,W; we should round up...
002D 1803 0101 btfscc STATUS,CARRY; If Remainder < (0.5 * Divisor), skip
002E 00A2 0102 incf ACCbLO,F; Round up
002F 1903 0103 btfscc STATUS,Z; Check low byte for wrap around
0030 00A3 0104 incf ACCbHI,F; If LSB wrapped, increment high byte
0031 1D03 0105 btfscc STATUS,Z; Check high byte for wrap around
0032 2835 0106 goto done; High byte didn't wrap
0033 0107 ovr_flo
0033 01A2 0108 clrf ACCbLO
0034 01A3 0109 clrf ACCbHI
0035 0110 done
0035 21FD 0111 call DEL_100; Wait 100 msec before clearing HW.
0036 1009 0112 bcf PORT_E,0; Disable INIT
0037 1089 0113 bcf PORT_E,1; Disable BINH
0038 21FD 0114 call DEL_100; Wait 100 msec before enabling HW.
0039 2812 0115 goto Xdcr
0116
0120
0149
0155 end
0151

MEMORY USAGE MAP ('X' = Used, '-' = Unused)
0000 : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX------
0040 : ---------------------------------------------
All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
Air Flow Control Using Fuzzy Logic

Author: Robert Schreiber
Logic Products Division

INTRODUCTION

Fuzzy logic control can be used to implement a wide variety of intelligent functions including everything from consumer electronic goods and household appliances to auto electronics, process control, and automation.

Typically, fuzzy logic control applications fall into two categories. First, it can be used to enhance existing products with intelligent functions. Second, it can utilize sensors that continuously respond to changing input conditions. In addition, fuzzy logic simplifies dealing with non-linearities in systems, and allows for quicker product development cycles.

This application note will step the user through a fuzzy logic control design utilizing sensors. The development tool used is Inform® Software's fuzzyTEC®-MP. The development tool allows for an all-graphical editor, analyzers, and debug capability.

PROJECT DESCRIPTION

The block diagram of the project is shown in Figure 1 and operates as follows.

FIGURE 1: BLOCK DIAGRAM
The control panel prompts the user to enter the desired beach ball height on the 16-key keypad. The keypad input is echoed on the LCD module and the user is prompted for confirmation. Upon confirmation of user input, the control panel initiates a ranging cycle to calculate the current height of the beach ball. The desired height and current height are continually displayed on the LCD module. From the current height, the control panel calculates both the velocity and the delta height (difference in desired height from current height). This information, along with the desired height, is transmitted to the PC via an RS-232 link. The fuzzy logic algorithm, running on the PC, calculates the appropriate duty cycle of the DC fan and transmits this information to the control panel. This emulates a "real world" environment in which system level debug can be done on the PC in real-time. The control panel controls the duty cycle of the DC fan with this input. The above listed ranging process continues indefinitely until interrupted by the user.

The control panel houses an ultrasonic ranging module and the microcontroller. The microcontroller handles all of the peripheral interfaces including the 16-key keypad, the LCD display, the ultrasonic ranging module, and the RS-232 serial link. The project required a microcontroller that could handle the data throughput and all of these peripherals with little or no external components. The microcontroller used was the PIC16C74, which contains 4K of on-chip program memory and 192 bytes of on-chip data memory. Furthermore, the interrupt capabilities, I/O pins, PWM module, capture and compare modules, timer modules, Serial Communications Interface (SCI), and A/D converter make it an excellent fit for the application. In addition, the on-chip pulse-width-modulation (PWM) module allows for a single component (FET) interface for the DC fan control and the ranging module can interface directly to the microcontroller (refer to Application Note AN597, "Implementing Ultrasonic Ranging").

**FUZZY DESIGN**

Fuzzy logic first translates the crisp inputs from the sensor into a linguistic description. Then it evaluates the control strategy contained in fuzzy logic rules and translates the result back into a crisp value.

The first step in fuzzy logic control design is system definition. The only possible sources of inputs to the fuzzy logic control algorithm are the ultrasonic transducer, the user, and the DC fan. The key is to decide which of these inputs are significant and which are not. Basically, the behavior of the beach ball was characterized by asking the following questions from the beach ball's perspective:

- Where am I?
- How far am I from where I want to be?
- How fast am I getting there?
- What external force will get me there?

The nice thing about fuzzy logic control is that the linguistic system definition becomes the control algorithm.

The variables were defined as follows:

- **Current Height** [Where am I?]
- **Delta Height** [How far am I from where I want to be?]
- **Velocity** [How fast am I getting there?]
- **Duty Cycle** [What external force will get me there?]

Defining the variables was the starting point, but for the algorithm to work smoothly, it isn't good enough to say "the beach ball has velocity," you need to know to what degree the beach ball has velocity. This is accomplished by defining terms that more fully describe the variable. The combination of variables and terms gives a linguistic description of what is happening to the system. From this, the Velocity variable can be described as having a "positive small velocity" or a "positive big velocity," not just a "velocity."
There is no fixed rule on how many terms to define per variable. Typically, three to five terms are defined, but more or less may be needed based on the control algorithm. In retrospect, we probably could have reduced Current Height to three terms and Velocity to five terms. Table 1 lists the four variables that are used for the trade show demo and their associated terms.

Once the linguistic variables are defined, data types and values need to be defined. For this application, data types were defined as 8-bit integers (16-bit definition is also possible). After defining the data types, the shell and code values for each variable were specified. A shell value is used within the fuzzy logic development tool and a code value is used when the code is generated.

The best way to describe shell and code values is using the analogy of a D/A converter. If we have a 5.0V, 8-bit D/A converter, the digital input would correspond to the code value and the analog output would correspond to the shell value. This is, if we write (or pass) a value of 128 to the D/A we would get a 2.51V out. Applying this analogy to our project, we would pass a crisp value (digital) to the fuzzy world and the fuzzy world would use the fuzzy value (analog).

Therefore, when we define shell and code values, we are basically defining the "D/A converter." For example, you can define the shell value for Duty Cycle to be a minimum of 0 and a maximum of 100 (percent). Therefore, within the fuzzy logic development tool, Duty Cycle will take on a value between 0 and 100, inclusive.

The code value is limited by the data type, but can take on any or all of the digital range. That is, if the shell value is 0 to 100, the code values could be defined as 0 to 100. But to get full resolution, the code value should be defined over the entire range (i.e., 0 to 255 for 8-bit data types). The code values and shell values were defined as shown in Table 2. Note that for the height and velocity variables, the shell values are scaled by 2 (i.e., a Current Height with a crisp value of 60 would correspond to 30 inches).

---

**TABLE 1: INPUT AND OUTPUT VARIABLES AND TERMS**

<table>
<thead>
<tr>
<th>Input Variables</th>
<th>Output Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Height</td>
<td>Delta Height</td>
</tr>
<tr>
<td>very lo</td>
<td>neg big</td>
</tr>
<tr>
<td>lo</td>
<td>neg small</td>
</tr>
<tr>
<td>medium</td>
<td>zero</td>
</tr>
<tr>
<td>hi</td>
<td>pos small</td>
</tr>
<tr>
<td>very hi</td>
<td>pos big</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 2: SHELL AND CODE VALUES**

<table>
<thead>
<tr>
<th>Shell Value</th>
<th>Code Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Min.</td>
</tr>
<tr>
<td>Current Height</td>
<td>0</td>
</tr>
<tr>
<td>Delta Height</td>
<td>-50</td>
</tr>
<tr>
<td>Velocity</td>
<td>-5</td>
</tr>
<tr>
<td>Duty Cycle</td>
<td>0</td>
</tr>
</tbody>
</table>
Next, the membership functions were defined to further describe the variables. The fuzzy logic development tool creates the membership functions automatically. This gives a good starting point, but the membership functions still need to be fine-tuned during the debug phase. In this application, only the linear shaped functions (Pi, Z, S and Lambda types) were used as seen in Figure 2.

**FIGURE 2: STANDARD MEMBERSHIP FUNCTION TYPES**

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z-Type</td>
<td></td>
</tr>
<tr>
<td>Pi-Type</td>
<td></td>
</tr>
<tr>
<td>Lambda-Type</td>
<td></td>
</tr>
<tr>
<td>S-Type</td>
<td></td>
</tr>
</tbody>
</table>

**FUZZIFICATION**

Fuzzification entails translating a crisp value into a fuzzy value. Once all of the variables have been defined, the interfaces between the variables need to be defined. The interfaces for the input variables contain the fuzzification procedures. In defining the interfaces, the input variable's fuzzification method needs to be defined. The computation of fuzzification is carried out at runtime for code efficiency. The type of fuzzification used in this project is membership function computation. This is largely due to the code space efficiency and accuracy associated with this method. Once fuzzification has taken place, the algorithm is performed in the fuzzy world according to the rule base.

**FIGURE 3: DEGREE OF MEMBERSHIP**

The entire fuzzy inference is contained within the rule blocks of a system. For example, if the beach ball is near the top of the tube and it was commanded to be near the bottom of the tube, the rule that described the situation would be:

**IF CURRENT HEIGHT = VERY HI AND DELTA HEIGHT = NEGATIVE BIG THEN DUTY CYCLE = SLOW**

The above rule describes one situation, but the rule definition would continue until the system was adequately described. The rule block is the collection of all rules that describe the system.

The rules of the rule block can also be defined in terms of how much a specific rule is supported when calculating inference. The support of a rule, or plausibility, is known as the degree of support for that rule. A plausible rule is defined by a 1.0, a totally implausible rule is defined by 0.0. In this project all rules are fully supported.

The degree to which a crisp value belongs to a term is known as the degree of membership. For example, the terms Medium and Hi for the variable Current Height were defined as a Lambda-type membership function centered around the crisp values 52 (26 inches) and 82 (41 inches), respectively, as shown in Figure 3.
Therefore, if the beach ball was at 26 inches, the degree of membership would be 1.0 for Medium and 0.0 for Hi. However, as the beach ball rises in height, the degree of membership for the term Medium would decrease and the degree of membership for the term Hi would increase. The interplay of these linguistic variable terms is controlled by the rule base. The rule base defines not only the relationship between the terms, but also how much each rule is supported, as described previously.

From the list of rules, a Fuzzy Associative Map (FAM) is constructed (see below). The FAM shows the plausibility (degree of support) of each rule as seen in Figure 4 and Figure 5.

**FIGURE 4: MATRIX RULE EDITOR WITH FAM RULES**

![Matrix Rule Editor with FAM Rules](image)
FIGURE 5: 3-D RULE DISPLAY

delta_height
neg_big
neg_small
zero
pos_small
pos_big
current_height
very_lo
lo
medium
hi
very_hi

Ok
**DEFUZZIFICATION**

Defuzzification entails translating a fuzzy value to a crisp value. The interface for the output variables contains the defuzzification procedures. For most control applications (and this project), the center-of-maximum (CoM) method is used for defuzzification. CoM evaluates more than one output term as valid and compromises between them by computing a weighted mean of the term membership maxima. Example 1 and Figure 6 show the defuzzification of the linguistic variable Duty Cycle using CoM.

**EXAMPLE 1: DEFUZZIFICATION OF DUTY CYCLE**

The crisp values of the three input variables are as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Height</td>
<td>30</td>
</tr>
<tr>
<td>Delta Height</td>
<td>0</td>
</tr>
<tr>
<td>Velocity</td>
<td>0</td>
</tr>
</tbody>
</table>

The crisp value can be calculated using the CoM method with the following equation.

\[
C = \sum_{i} \left[ I \cdot \max_x (M) \cdot \arg (\max_x (M)) \right] / \sum_{i} I
\]

- \(C\) = crisp output value
- \(i\) = linguistic term
- \(I\) = inference result
- \(M\) = membership function of linguistic term

For this example, when the crisp values are fuzzified, the Duty Cycle variable is defined to be mostly "medium" (degree of membership of 0.7) and somewhat "medium fast" (degree of membership 0.1). The arguments for the "medium" and "medium fast" term membership maxima are 165 and 178, respectively.

\[
\frac{(0.7 \cdot 1.0 \cdot 165) + (0.1 \cdot 1.0 \cdot 178)}{(0.7 + 0.1)} = 166
\]

**FIGURE 6: DEFUZZIFICATION OF DUTY CYCLE**

![Defuzzification Diagram](image_url)

The crisp value is 166.0000.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_height: 30.0000</td>
<td>duty_cycle: 166.0000</td>
</tr>
<tr>
<td>delta_height: 0.0000</td>
<td></td>
</tr>
<tr>
<td>velocity: 0.0000</td>
<td></td>
</tr>
</tbody>
</table>

Value: 0.0000  Steps [%]: 10.00
DEBUGGING

In serial debug mode, one can graphically adjust the variable terms and see the results in "real time." On this project, the first variable adjusted was the Duty Cycle variable. Duty Cycle was adjusted so that the beach ball reached 30 inches (Figure 7). The Delta Height terms were fine-tuned -- negative small, zero, and positive small were bunched together -- and the beach ball stabilized at 30 inches (Figure 8). There was virtually no fluctuation in the height. In order for the system to self-correct for environmental (external) changes, the Velocity variable was used. The velocity variable is calculated by the difference in height between consecutive height calculations. A few rules were added that used the Velocity variable to nudge the ball into place when the environmental conditions changed (Figure 9).

Another advantage of fuzzy logic is that it simplifies dealing with non-linearities of the system. The system was highly non-linear, so it was tested at the extremes and moving the beach ball at different rates from one extreme to the other. The Current Height variable needed almost no adjustment (Figure 10). The variable that required the most work was the Duty Cycle variable, but in less than a day, the algorithm was working well within specifications. The beach ball could go from a resting position, with the DC fan off, to the maximum allowable height of 42 inches in less than 8 seconds with no overshoot. Operation between the minimum and maximum height was much quicker, also with no overshoot.

The final graphical representation of the linguistic variables are shown in Figure 7 through Figure 10.

FIGURE 7: DUTY CYCLE VARIABLE

<table>
<thead>
<tr>
<th>Term</th>
<th>0</th>
<th>255</th>
</tr>
</thead>
<tbody>
<tr>
<td>very_slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>med_slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>medium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>med_fast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>very_fast</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Another advantage of fuzzy logic is that it simplifies dealing with non-linearities of the system. The system was highly non-linear, so it was tested at the extremes and moving the beach ball at different rates from one extreme to the other. The Current Height variable needed almost no adjustment (Figure 10). The variable that required the most work was the Duty Cycle variable, but in less than a day, the algorithm was working well within specifications. The beach ball could go from a resting position, with the DC fan off, to the maximum allowable height of 42 inches in less than 8 seconds with no overshoot. Operation between the minimum and maximum height was much quicker, also with no overshoot.

The final graphical representation of the linguistic variables are shown in Figure 7 through Figure 10.
FIGURE 8: DELTA HEIGHT VARIABLE

<table>
<thead>
<tr>
<th>Term</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>neg_big</td>
<td>1</td>
</tr>
<tr>
<td>neg_small</td>
<td>0</td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>pos_small</td>
<td>0</td>
</tr>
<tr>
<td>pos_big</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ x = -16.6667 \]

\[ y = 0 \]

FIGURE 9: VELOCITY VARIABLE

<table>
<thead>
<tr>
<th>Term</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>neg_big</td>
<td>1</td>
</tr>
<tr>
<td>neg_med</td>
<td>0</td>
</tr>
<tr>
<td>neg_small</td>
<td>0</td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>pos_small</td>
<td>0</td>
</tr>
<tr>
<td>pos_med</td>
<td>0</td>
</tr>
<tr>
<td>pos_big</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ x = -3.78431 \]

\[ y = 1 \]
INTEGRATION

The system parameters and graphical variable representations are captured in a Fuzzy Technology Language (FTL) file. The FTL file is a vendor and hardware independent language which defines the fuzzy logic based system. The FTL file for this project can be seen in Appendix A. The FTL file is used to generate the public variable definitions and code which can be embedded in the microcontroller. The appropriate device family from the pre-assembler code are generated by simply selecting the compile pull-down menu. Once the pre-assembler file is generated, the "hooks" to the main program must be added.

The best way to embed the code is to use the template MYMAIN.ASM. The template for each of the families of devices (PIC16C5X, PIC16CXX and PIC17CXX) is included in the fuzzyTECH®-MP development kit. The template shown in Appendix B is for the PIC16CXX family.

The file MYMAIN.ASM should contain your program in the "main_loop" section. The only other modifications required to the template are listed below and are specified in the left hand column of Appendix B.

1. Processor Type definition
2. Code Start Address
3. Fuzzy RAM Start Address
4. Include Public Variable Definition file (myproj.var), which was created by fuzzyTECH®-MP
5. Include Pre-Assembler Code (myproj.asm) which was created by fuzzyTECH®-MP
6. Call Initialization (initmyproj) which was created by fuzzyTECH®-MP
7. Set Crisp Input Value(s)
8. Call Fuzzy Logic System (myproj)
9. Read Crisp Output Value(s)

For this project, the fuzzy logic algorithm assembled to 704 words of program memory and 41 bytes of data memory.

SUMMARY

This project demonstrates many aspects of fuzzy logic control - quick development cycle, real-time debug, sensor integration, and non-linear system control. The total development time for the application took less than a week and performed well within system specifications.
APPENDIX A: FUZZY TECHNOLOGY LANGUAGE FILE

PROJECT {
  NAME = B_BALL.FTL;
  AUTHOR = ROBERT SCHREIBER;
  DATEFORMAT = M. D. YY;
  LASTCHANGE = 9.16.94;
  CREATED = 9.14.94;
  SHELL = MP;
  COMMENT {
/* COMMENT */
  SHELLOPTIONS {
    ONLINE_REFRESH_TIME = 55;
    ONLINE_TIMEOUT_COUNT = 0;
    ONLINE_CODE = OFF;
    TRACE_BUFFER = (OFF, PAR(10000));
    BSUM_AGGREGATION = OFF;
    PUBLIC_IO = ON;
    FAST_CMBF = ON;
    FAST_COA = OFF;
    SCALE_MBF = OFF;
    FILE_CODE = OFF;
    BTYPE = 8_BIT;
  } /* SHELLOPTIONS */
  MODEL {
    VARIABLE SECTION {
      LVAR {
        NAME = current_height;
        BASEVAR = Current_Height;
        LV RANGE = MIN(0.000000), MAX(120.000000),
                    MINDEF(0), MAXDEF(255),
                    DEFAULT OUTPUT(120.000000);
        RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                     SHOWGRID (ON), SNAPTOGRID(ON);
        TERM {
          TERMNAME = very_lo;
          POINTS = (0.000000, 1.000000),
                    (14.117647, 0.000000),
                    (120.000000, 0.000000);
          SHAPE = LINEAR;
          COLOR = RED (255), GREEN (0), BLUE (0);
        }
        TERM {
          TERMNAME = lo;
          POINTS = (0.000000, 0.000000),
                    (5.176471, 0.000000),
                    (24.941176, 1.000000),
                    (40.941176, 0.000000),
                    (120.000000, 0.000000);
          SHAPE = LINEAR;
          COLOR = RED (0), GREEN (255), BLUE (0);
        }
        TERM {
          TERMNAME = medium;
          POINTS = (0.000000, 0.000000),
                    (27.294118, 0.000000),
                    (51.294118, 1.000000),
                    (120.000000, 0.000000);
          SHAPE = LINEAR;
          COLOR = RED (0), GREEN (0), BLUE (255);
        }
      }
    }
  }
} /* PROJECT */
(66.352941, 0.000000),
(120.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (0), BLUE (255);
}
TERM {
TERMNAME = hi;
POINTS = (0.000000, 0.000000),
(55.529412, 0.000000),
(82.352941, 1.000000),
(106.352941, 0.000000),
(120.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
TERMNAME = very_hi;
POINTS = (0.000000, 0.000000),
(73.411765, 0.000000),
(113.411765, 1.000000),
(120.000000, 1.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (128), BLUE (0);
}
TERM {
TERMNAME = neg_big;
POINTS = (-50.000000, 1.000000),
(-16.666667, 0.000000),
(50.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (255), GREEN (0), BLUE (0);
}
TERM {
TERMNAME = neg_small;
POINTS = (-50.000000, 0.000000),
(-21.764706, 0.000000),
(-6.470588, 1.000000),
(-0.588235, 0.000000),
(50.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (255), BLUE (0);
}
TERM {
TERMNAME = zero;
POINTS = (-50.000000, 0.000000),
(-12.352941, 0.000000),
(0.196078, 1.000000),
(13.529412, 0.000000),
(50.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (0), BLUE (255);
TERM {
    TERMNAME = pos_small;
    POINTS = (-50.000000, 0.000000),
             (0.196078, 0.000000),
             (10.000000, 1.000000),
             (10.392157, 1.000000),
             (32.745098, 0.000000),
             (50.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
    TERMNAME = pos_big;
    POINTS = (-50.000000, 0.000000),
             (26.470588, 0.000000),
             (39.803922, 1.000000),
             (50.000000, 1.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (0);
}
/* LVAR */
LVAR {
    NAME = duty_cycle;
    BASEVAR = Duty_Cycle;
    LV RANGE = MIN(0.000000), MAX(255.000000),
               MINDEF(0), MAXDEF(255),
               DEFAULT_OUTPUT(0.000000);
    RESOLUTION = XRGRID(0.000000), YGRID(1.000000),
                 SHOWGRID (ON), SNAPTOGRID(ON);
}
TERM {
    TERMNAME = very_slow;
    POINTS = (0.000000, 0.000000),
             (1.000000, 0.000000),
             (103.000000, 1.000000),
             (113.000000, 1.000000),
             (147.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (255), GREEN (0), BLUE (0);
}
TERM {
    TERMNAME = slow;
    POINTS = (0.000000, 0.000000),
             (108.000000, 0.000000),
             (127.000000, 1.000000),
             (131.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (255), BLUE (0);
}
TERM {
    TERMNAME = med_slow;
    POINTS = (0.000000, 0.000000),
             (133.000000, 0.000000),
             (142.000000, 1.000000),
             (162.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (128);
TERM {
    TERMNAME = medium;
    POINTS = (0.000000, 0.000000),
             (151.000000, 0.000000),
             (164.000000, 1.000000),
             (166.000000, 1.000000),
             (174.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (0), BLUE (255);
}

TERM {
    TERMNAME = med_fast;
    POINTS = (0.000000, 0.000000),
             (166.000000, 0.000000),
             (178.000000, 1.000000),
             (193.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (255), GREEN (0), BLUE (128);
}

TERM {
    TERMNAME = fast;
    POINTS = (0.000000, 0.000000),
             (189.000000, 0.000000),
             (202.000000, 1.000000),
             (232.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (128), GREEN (0), BLUE (0);
}

TERM {
    TERMNAME = very_fast;
    POINTS = (0.000000, 0.000000),
             (206.000000, 0.000000),
             (255.000000, 1.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (0);
}

} /* LVAR */

LVAR {
    NAME = velocity;
    BASEVAR = Velocity;
    LVRANGE = MIN(-5.000000), MAX(5.000000),
              MINDEF(0), MAXDEF(255),
              DEFAULT_OUTPUT(0.000000);
    RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                 SHOWGRID (OFF), SNAPTOGRID(ON);
}

TERM {
    TERMNAME = neg_big;
    POINTS = (-5.000000, 1.000000),
             (-3.784314, 1.000000),
             (-2.529412, 0.000000),
             (5.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (255), GREEN (0), BLUE (0);
}

TERM {
    TERMNAME = neg_med;
POINTS = (-5.000000, 0.000000),
(-3.784314, 0.000000),
(-2.529412, 1.000000),
(-1.274510, 0.000000),
(5.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (255), BLUE (0);
}
TERM {
TERMINAME = neg_small;
POINTS = (-5.000000, 0.000000),
(-2.568627, 0.000000),
(-1.313725, 1.000000),
(-0.058824, 0.000000),
(5.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (0), BLUE (255);
}
TERM {
TERMINAME = zero;
POINTS = (-5.000000, 0.000000),
(-1.000000, 0.000000),
(0.960784, 0.000000),
(5.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
TERMINAME = pos_small;
POINTS = (-5.000000, 0.000000),
(1.117647, 1.000000),
(2.372549, 0.000000),
(5.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (128), BLUE (0);
}
TERM {
TERMINAME = pos_med;
POINTS = (-5.000000, 0.000000),
(1.078431, 0.000000),
(2.333333, 1.000000),
(3.588235, 0.000000),
(5.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (0), BLUE (128);
}
TERM {
TERMINAME = pos_big;
POINTS = (-5.000000, 0.000000),
(2.294118, 0.000000),
(3.549020, 1.000000),
(5.000000, 1.000000);
SHAPE = LINEAR;
COLOR = RED (255), GREEN (0), BLUE (128);
}
} /* IVAR */
} /* VARIABLE SECTION */
OBJECT SECTION {
  INTERFACE {
    INPUT = (current_height, FCMBF);
    POS = -213, -137;
    RANGECHECK = ON;
  }
  INTERFACE {
    INPUT = (delta_height, FCMBF);
    POS = -216, -83;
    RANGECHECK = ON;
  }
  INTERFACE {
    OUTPUT = (duty_cycle, COM);
    POS = 158, -79;
    RANGECHECK = ON;
  }
  RULEBLOCK {
    INPUT = current_height, delta_height, velocity;
    OUTPUT = duty_cycle;
    AGGREGATION = (MIN_MAX, PAR (0.000000));
    COMPOSITION = (GAMMA, PAR (0.000000));
    POS = -39, -113;
    RULES {
      IF current_height = very_lo
      AND delta_height = neg_big
      THEN duty_cycle = slow WITH 1.000;
      IF current_height = very_lo
      AND delta_height = neg_small
      THEN duty_cycle = med_slow WITH 1.000;
      IF current_height = very_lo
      AND delta_height = zero
      THEN duty_cycle = medium WITH 1.000;
      IF current_height = very_lo
      AND delta_height = pos_small
      THEN duty_cycle = fast WITH 1.000;
      IF current_height = very_lo
      AND delta_height = pos_big
      THEN duty_cycle = very_fast WITH 1.000;
      IF current_height = lo
      AND delta_height = neg_big
      THEN duty_cycle = slow WITH 1.000;
      IF current_height = lo
      AND delta_height = neg_small
      THEN duty_cycle = med_slow WITH 1.000;
      IF current_height = lo
      AND delta_height = zero
      THEN duty_cycle = medium WITH 1.000;
      IF current_height = lo
      AND delta_height = pos_small
      THEN duty_cycle = fast WITH 1.000;
      IF current_height = lo
      AND delta_height = pos_big
      THEN duty_cycle = very_fast WITH 1.000;
      IF current_height = medium
      AND delta_height = neg_big
      THEN duty_cycle = very_slow WITH 1.000;
      IF current_height = medium
      AND delta_height = neg_small
      THEN duty_cycle = med_slow WITH 1.000;
      IF current_height = medium
      AND delta_height = medium
      THEN duty_cycle = medium
    }
  }
}
AND delta_height = zero
THEN duty_cycle = med_fast WITH 1.000;
IF current_height = medium
AND delta_height = pos_small
THEN duty_cycle = fast WITH 1.000;
IF current_height = medium
AND delta_height = pos_big
THEN duty_cycle = very_fast WITH 1.000;
IF current_height = hi
AND delta_height = neg_big
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = hi
AND delta_height = zero
THEN duty_cycle = med_slow WITH 1.000;
IF current_height = hi
AND delta_height = pos_small
THEN duty_cycle = fast WITH 1.000;
IF current_height = hi
AND delta_height = pos_big
THEN duty_cycle = very_fast WITH 1.000;
IF current_height = very_hi
AND delta_height = neg_big
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = very_hi
AND delta_height = zero
THEN duty_cycle = med_slow WITH 1.000;
IF current_height = very_hi
AND delta_height = pos_small
THEN duty_cycle = medium WITH 1.000;
IF current_height = very_hi
AND delta_height = pos_big
THEN duty_cycle = very_fast WITH 1.000;
IF current_height = very_lo
AND delta_height = neg_small
AND velocity = zero
THEN duty_cycle = fast WITH 1.000;
IF current_height = very_lo
AND delta_height = neg_small
AND velocity = pos_small
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = very_lo
AND delta_height = neg_small
AND velocity = pos_med
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = very_lo
AND delta_height = neg_small
AND velocity = pos_big
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = very_lo
AND delta_height = pos_small
AND velocity = zero
THEN duty_cycle = fast WITH 1.000;
IF current_height = very_lo
AND delta_height = pos_small
AND velocity = pos_small
AN600

AND velocity = neg_small
THEN duty_cycle = fast WITH 1.000;
IF current_height = very_lo
AND delta_height = pos_small
AND velocity = neg_med
THEN duty_cycle = fast WITH 1.000;
IF current_height = very_lo
AND delta_height = pos_small
AND velocity = neg_big
THEN duty_cycle = fast WITH 1.000;
IF current_height = lo
AND delta_height = neg_small
AND velocity = zero
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = lo
AND delta_height = neg_small
AND velocity = pos_small
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = lo
AND delta_height = neg_small
AND velocity = pos_med
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = lo
AND delta_height = neg_small
AND velocity = pos_big
THEN duty_cycle = very_slow WITH 1.000;
IF current_height = lo
AND delta_height = pos_small
AND velocity = zero
THEN duty_cycle = fast WITH 1.000;
IF current_height = lo
AND delta_height = pos_small
AND velocity = neg_small
THEN duty_cycle = fast WITH 1.000;
IF current_height = lo
AND delta_height = pos_small
AND velocity = neg_med
THEN duty_cycle = fast WITH 1.000;
IF current_height = lo
AND delta_height = pos_small
AND velocity = neg_big
THEN duty_cycle = fast WITH 1.000;
IF current_height = medium
AND delta_height = neg_small
AND velocity = zero
THEN duty_cycle = slow WITH 1.000;
IF current_height = medium
AND delta_height = neg_small
AND velocity = pos_small
THEN duty_cycle = slow WITH 1.000;
IF current_height = medium
AND delta_height = neg_small
AND velocity = pos_med
THEN duty_cycle = slow WITH 1.000;
IF current_height = medium
AND delta_height = neg_small
AND velocity = pos_big
THEN duty_cycle = slow WITH 1.000;
IF current_height = medium
AND delta_height = pos_small
AND velocity = zero
THEN duty_cycle = fast WITH 1.000;

IF current_height = medium
   AND delta_height = pos_small
   AND velocity = neg_small
THEN duty_cycle = fast WITH 1.000;

IF current_height = medium
   AND delta_height = pos_small
   AND velocity = neg_med
THEN duty_cycle = fast WITH 1.000;

IF current_height = medium
   AND delta_height = pos_small
   AND velocity = neg_big
THEN duty_cycle = fast WITH 1.000;

IF current_height = hi
   AND delta_height = neg_small
   AND velocity = zero
THEN duty_cycle = med_slow WITH 1.000;

IF current_height = hi
   AND delta_height = neg_small
   AND velocity = pos_small
THEN duty_cycle = med_slow WITH 1.000;

IF current_height = hi
   AND delta_height = neg_small
   AND velocity = pos_med
THEN duty_cycle = med_slow WITH 1.000;

IF current_height = hi
   AND delta_height = neg_small
   AND velocity = pos_big
THEN duty_cycle = med_slow WITH 1.000;

IF current_height = very_hi
   AND delta_height = neg_small
   AND velocity = zero
THEN duty_cycle = medium WITH 1.000;

IF current_height = very_hi
   AND delta_height = neg_small
   AND velocity = pos_small
THEN duty_cycle = medium WITH 1.000;

IF current_height = very_hi
   AND delta_height = neg_small
   AND velocity = pos_med
THEN duty_cycle = medium WITH 1.000;

IF current_height = very_hi
   AND delta_height = neg_small
   AND velocity = pos_big
THEN duty_cycle = medium WITH 1.000;

IF current_height = very_hi
   AND delta_height = neg_small
   AND velocity = very_hi
   AND delta_height = neg_small
   AND velocity = very_hi
THEN duty_cycle = medium WITH 1.000;
AND velocity = pos_big
THEN duty_cycle = medium WITH 1.000;
IF current_height = very_hi
AND delta_height = pos_small
AND velocity = zero
THEN duty_cycle = very_fast WITH 1.000;
IF current_height = very_hi
AND delta_height = pos_small
AND velocity = neg_small
THEN duty_cycle = very_fast WITH 1.000;
IF current_height = very_hi
AND delta_height = pos_small
AND velocity = neg_med
THEN duty_cycle = very_fast WITH 1.000;
IF current_height = very_hi
AND delta_height = pos_small
AND velocity = neg_big
THEN duty_cycle = very_fast WITH 1.000;

} /* RULES */

} /* OBJECT_SECTION */

} /* MODEL */

} /* PROJECT */

TERMINAL {
BAUDRATE = 9600;
STOPBITS = 1;
PROTOCOL = NO;
CONNECTION = PORT1;
INPUTBUFFER = 4096;
OUTPUTBUFFER = 1024;
} /* TERMINAL */
APPENDIX B: MYMAIN.ASM TEMPLATE FOR THE PIC16CXX FAMILY

1. PROCESSOR 16C71
   -----------------------------------------------
   USER MAIN FILE
   -----------------------------------------------

2. CODE_START EQU 0x100 ; code startadr for 16C71
   RESET_ADDR EQU 0x000 ; reset vector

3. FUZZY_RAM_START EQU 0x00C ; first free RAM location for 16C71
   include "myproj.var" ; include preassembler variables
   CBLOCK ; starts after fuzzy ram locations
   user1 ; reserve 1 byte (example)
   ENDC

4. ORG CODE_START ; example start adress for code

5. mymain
   call initmyproj ; call init once
   movlw 000 ; example
   movwf lv0_input_1 ; set 1st crisp input
   movlw 0A0 ; example
   movwf lv1_input_2 ; set 2nd crisp input
   call myproj ; call preassembler code
   movf invalidflags, W ; test if the project is completely defined
   goto case_no_fire

6. case_fire
   ; proj OK
   movf lv2_output, W ; fetch crisp output
   goto main_loop

7. case_no_fire
   ; no rule defined for this input combination
   ; call default_handling_routine
   ; user code
   goto main_loop

8. INCLUDE "myproj.asm" ; include preassembler code

9. ORG RESET_ADDR ; jump to program code
   goto mymain ; end for assembler (only here)

Note: Refer to the "integration" section for the number descriptions.
INTRODUCTION

Power consumption is an important element in designing a system, particularly in today's battery powered world. The PIC16/17 family of devices has been designed to give the user a low-cost, low-power, and high performance solution to this problem. For the application to operate at the lowest possible power, the designer must ensure that the PIC16/17 devices are properly configured. This application note describes some design techniques to lower current consumption, some battery design considerations, and suggestions to assist the designer in resolving power consumption problems.

DESIGN TECHNIQUES

Many techniques are used to reduce power consumption in the PIC16/17 devices. The most commonly used methods are SLEEP Mode or external events. These modes are the best way to reduce lpd in a system. The PIC16/17 device can periodically wake-up from Sleep using the Watchdog Timer or external interrupt, execute code and then go back into SLEEP Mode. In SLEEP Mode the oscillator is shut off, which causes the PIC16/17 device to consume very little current. Typical lpd current in most PIC16/17 devices is on the order of a few microamps.

In cases where the PIC16/17 uses an RC oscillator but cannot use SLEEP Mode, another technique is used to lower power consumption. An I/O pin can remove a parallel resistance from the oscillator resistor while waiting for an event to occur. This would slow down the internal clock frequency, by increasing the resistance, and thus reduce lpd. Once an event occurs the resistor can be switched in and the PIC16/17 device can process the event at full speed. Figure 1 shows how to implement this technique. The resistor R1 would be used to increase the clock frequency by making the I/O pin an output and setting it to Vdd.

External events can be used to control the power to PIC16/17 devices. For these cases, the Watchdog Timer can be disabled to further reduce current consumption. Figure 2 shows an example circuit that uses an external event to latch power on for the PIC16/17 device. Once the device has finished executing code, it disables power by resetting the latch. The latching circuit uses a low-power 4000 series CMOS quad chip which consumes a typical of 10 µA of current. The measured value of current consumption for the complete circuit with the PIC16/17 powered-down was 1 nA. Current consumption for a PIC16/17 in SLEEP Mode is typically 1 µA.
Power consumption is dependent on the oscillator frequency of the system. The device must operate fast enough to interface with external circuitry, yet slow enough to conserve power. The designer must account for oscillator start-up time, external circuitry initialization, and code execution time when calculating device power consumption. Table 1 shows various frequency oscillators, oscillator modes and the average current consumption of each mode. A PIC16C54 was used to collect data for Table 1 and the code is shown in Example 1. A current profile for a PIC16C54 in RC oscillator mode running at 261 kHz is shown in Figure 3. Figure 4 shows a current profile for a PIC16C54 in XT mode running at 1 MHz. The current profile includes three regions: power-up, active, and sleep. The power-up region is defined as the time the PIC16/17 device is in Power-On Reset and/or Oscillator Start-up Time. The active region is the time that the PIC16/17 device is executing code and the sleep region is the time the device is in SLEEP Mode. When using a 32.768 kHz crystal in LP oscillator mode, the designer must check that the oscillator has stabilized during the Power-On Reset. Otherwise, the device may not come out of reset properly.

### Table 1: Oscillator Modes

<table>
<thead>
<tr>
<th>Osc. Type</th>
<th>Frequency</th>
<th>Osc. Mode</th>
<th>Power-up Region Current, Time</th>
<th>Active Region Current, Time</th>
<th>Sleep Region Current, Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistor / Capacitor</td>
<td>261 kHz</td>
<td>RC</td>
<td>51.2 µA, 17.5 ms</td>
<td>396 µA, 12.8 ms</td>
<td>0.32 µA, 140 ms</td>
</tr>
<tr>
<td>Resistor / Capacitor</td>
<td>1.13 MHz</td>
<td>RC</td>
<td>61.4 µA, 17.5 ms</td>
<td>810 µA, 2.5 ms</td>
<td>0.3 µA, 140 ms</td>
</tr>
<tr>
<td>Crystal</td>
<td>32.768 kHz</td>
<td>LP</td>
<td>51.2 µA, 19 ms</td>
<td>23.5 µA, 93 ms</td>
<td>0.3 µA, 140 ms</td>
</tr>
<tr>
<td>Crystal</td>
<td>50 kHz</td>
<td>LP</td>
<td>61.4 µA, 16 ms</td>
<td>39.4 µA, 48.5 ms</td>
<td>0.28 µA, 140 ms</td>
</tr>
<tr>
<td>Crystal</td>
<td>1 MHz</td>
<td>XT</td>
<td>92 µA, 17.5 ms</td>
<td>443 µA, 3 ms</td>
<td>0.35 µA, 140 ms</td>
</tr>
<tr>
<td>Crystal</td>
<td>8 MHz</td>
<td>HS</td>
<td>123 µA, 18 ms</td>
<td>2.11 mA, 250 µs</td>
<td>0.3 µA, 140 ms</td>
</tr>
<tr>
<td>Resonator</td>
<td>455 kHz</td>
<td>XT</td>
<td>38.4 µA, 17.3 ms</td>
<td>421 µA, 7 ms</td>
<td>0.34 µA, 140 ms</td>
</tr>
<tr>
<td>Resonator</td>
<td>8 MHz</td>
<td>HS</td>
<td>143 µA, 18 ms</td>
<td>2.5 mA, 250 µs</td>
<td>0.29 µA, 140 ms</td>
</tr>
</tbody>
</table>

### Example 1: Current Profile Code

```assembly
TITLE "Current Profiling Program"
LIST P=16C54, F=INHX8M 
INCLUDE "C:\PICMASTR\Pl6C5X.INC"
;***************************************************************************
;***************************************************************************
; ; This program initializes the PIC16C54, delays for 256 counts, then goes
; ; to sleep. The WDT wakes up the PIC16C54.
;***************************************************************************
;***************************************************************************
; Define General Purpose register locations
LSB EQU 0x10 ;delay control register
Reset Vector
ORG 0
START
MOVLW 0x0B ;WDT Prescaler of 1:8
OPTION
CLRF PORTA ;clear PORTA
CLRF PORTB ;clear PORTB
CLRW ;make PORTA and PORTB pins outputs
TRIS PORTA
TRIS PORTB
CLRF LSB
LOOP DECFSZ LSB,1
GOTO LOOP
SLEEP
END
```

© 1995 Microchip Technology Inc.
Designing a system for lower supply voltages, typically 3V, is another method to reduce IPD. This type of design is best utilized in a battery powered system where current consumption is very low. A wide range of devices from op-amps and Analog-to-Digital (A/D) converters to CMOS logic products are being manufactured for low voltage operation. This gives the designer the flexibility to design a low voltage system with the same type of components that are available for a 5V design. Refer to the PIC16/17 device data sheets for IPD vs. VDD data.

Since any I/O pin can source or sink up to 20 mA, the PIC16/17 devices can provide power to other components. Simply connect the VDD pin of an external component to an I/O pin. Currently, most of the op-amps, A/D converters, and other devices manufactured today are low-power and can be powered by this technique. This provides the ability to turn off power to sections of the system during periods of inactivity.

Temperature will affect the current consumption of the PIC16/17 devices in different ways. Typically devices will consume more current at extreme temperatures and batteries will have less available current at those same temperatures. PIC16/17 devices will exhibit higher IpD currents at high temperatures. Refer to the PIC16/17 device data sheets for IpD vs. Temperature data.
TROUBLESHOOTING IpD

The first step in troubleshooting IPD problems is to measure the IPD that the circuit is consuming. Circuits to measure IPD for all oscillator modes are shown in Figure 5 for PIC16/17 devices. The resistor Rp is used to measure the amount of current entering the Vdd pin when resistor Rg is shorted. The resistor Rg is used to measure the amount of current leaving the Vss pin when resistor Rp is shorted. The value of Rp and Rg should be approximately 100Ω for all oscillator modes. The two values of current should be approximately the same when the PIC16/17 is operating at the lowest possible power. If you find that the values of IPD measured from both configurations are not equivalent or are higher than the specifications, the following suggestions should help to find the source of extra current.

FIGURE 5: CIRCUITS TO MEASURE IPD FOR PIC16/17 DEVICES

Basically, if Ip is not equal to Ig, then an I/O pin is either sourcing (Ip>Ig) current or sinking (Ip<Ig) current.

- Is the MCLR pin tied to Vdd? Is the rate of rise of Vdd slower than 0.05 V/ms? Does Vdd start at Vss then rise? These conditions will not guarantee that the chip will come out of reset and function properly. Some of the circuits on PIC16/17 devices will start operating at lower voltage levels than other circuits. See Application Note AN522 "Power-Up Considerations" in the Microchip Embedded Control Handbook.

- Are all inputs being driven to Vss or Vdd? If any input is not driven to either Vss or Vdd, it will cause switching currents in the digital (i.e., flashing) input buffers. The exceptions are the oscillator pins and any pin configured as an analog input. During Power-on Reset or Oscillator Start-up time, pins that are floating may cause increased current consumption.

- All unused I/O pins should be configured as outputs and set high or low. This ensures that switching currents will not occur due to a floating input.

- Is the TMR0 (RTCC) pin pulled to Vss or Vdd? The TMR0 pin of PIC16C5X devices should be tied to Vss or Vdd for the lowest possible current consumption.

- If an analog voltage is present at a pin, is that pin configured as an analog input? If an analog voltage is present at a pin configured as a digital input, the digital input buffers devices will consume more current due to switching currents.

- Are all on-chip peripherals turned off? Any on-chip peripheral that can operate with an external clock source, such as the A/D converter or asynchronous timers, will consume extra current.

- Are you using the PORTB internal pull-up resistors? If so and if any PORTB I/O pin is driving or receiving a zero, the additional current from these resistors must be considered in the overall current consumption.

- Is the Power-Up Timer being used? This will add additional current drain during power-up.

- If the currents measured at the Rp and Rg resistors are not the same, then current is being sourced or sunk by an I/O pin. Make sure that all I/O pins that are driving external circuitry are switched to a low power state. For instance, an I/O pin that is driving an LED should be switched to a state where the LED is off.

- Is the window of a JW package device covered? Light will affect the current consumption of a JW package device with the window left uncovered.
IPD Analysis Using A Random Sample

The Microchip 1994 Microchip Data Book specifies the typical IPD current for a PIC16C5X part at 4µA and the maximum IPD current at 12 µA. These values are valid at a VDD voltage of 3V and a temperature range of 0°C to 70°C with the Watchdog Timer enabled. A control group of fifty PIC16C54’s were randomly selected with pre-production and production samples. IPD tests were run on the group for a voltage range of 2.5V to 6.5V and for a temperature range of 0°C to 70°C. Table 2 compares the median and maximum values obtained by the IPD tests to the typical and maximum values in the data book. The IPD test data and the data book values are based on VDD = 3.0V, Watchdog Timer Enabled, and a temperature range of 0°C to 70°C.

The values in the data book are obtained from devices in which the manufacturing process has been skewed to various extremes. This should produce devices which function close to the minimum and maximum operating ranges for each parameter shown in the data book. The typical values obtained in the data book are actually the mean value of characterization data at a temperature of 25°C. The minimum and maximum values shown in the data book are the mean value of the characterization data at the worst case temperature, plus or minus three times the standard deviation. Statistically this means that 99.5% of all devices will operate at or below the typical value and much less than the maximum value.

<table>
<thead>
<tr>
<th>Source</th>
<th>Typical or Median IPD</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Group</td>
<td>2.349 µA</td>
<td>3.048 µA</td>
</tr>
<tr>
<td>1994 Microchip Data Book</td>
<td>4 µA</td>
<td>12 µA</td>
</tr>
</tbody>
</table>

TABLE 2: IPD COMPARISON OF CONTROL GROUP vs. DATA BOOK VALUES

BATTERY DESIGN

When designing a system to use batteries, the designer must consider the maximum current consumption, operating voltage range, size and weight constraints, operating temperature range, and the frequency of operation. Once the system design is finished, the designer must again ask some questions that will define what type of battery to use. What is the operating voltage range? What is the current drain rate? What are the size constraints? How long will the system be used? What type of battery costs can be tolerated? What range of temperatures will the system be operated?

It is difficult to state a rule of thumb for selecting batteries because there are many variables to consider. For example, operating voltages vary from one battery type to another. Lithium cells typically provide 3.0V while Nickel-Cadmium cells provide 1.2V. On the other hand, Lithium cells can withstand minimal discharge rates while Nickel-Cadmium can provide up to 30A of current. A designer must consider all characteristics of each battery type when making a selection. Appendix B contains a simple explanation of batteries, a characteristic table for some common battery types, and discharge curves for the common batteries.

It is very important when doing a low power design to correctly estimate the required capacity of the power source. At this point, the designer should be able to estimate the operating voltage, current drain rates and how long the system is supposed to operate. To explain how to estimate the required capacity of a system, we will use the first entry from Table 1 using an RC oscillator set at 261 kHz. Figure 3 shows the current profile for this entry. It can be seen that the profile has a period of 170.3 ms with a 17.5 ms power-up region, a 12.8 ms active region, and a 140 ms sleep region. Assuming that the system will be required to operate for six months, we can now calculate the capacity required to power this system. Example 2 will illustrate the procedure. If a system does not have a periodic current profile, then the percentages obtained in step 1 of Example 2 will have to be estimated.
EXAMPLE 2: CAPACITY CALCULATION

1. Calculate the percentage of time spent in power-up, active, and sleep regions.
   - **power-up**
     \[
     \frac{17.5 \text{ ms}}{170.3 \text{ ms}} \times 100 = 10.3\%
     \]
   - **active**
     \[
     \frac{12.8 \text{ ms}}{170.3 \text{ ms}} \times 100 = 7.5\%
     \]
   - **sleep**
     \[
     \frac{140 \text{ ms}}{170.3 \text{ ms}} \times 100 = 82.2\%
     \]

2. Calculate the number of hours in 6 months.
   - 6 months
     \[
     x \left( \frac{30 \text{ days}}{\text{month}} \right) \times \left( \frac{24 \text{ hours}}{\text{day}} \right) = 4320 \text{ hours}
     \]

3. Using the number of hours, percentages, and currents calculate the capacity for each period of time.
   - **power-up**
     \[
     4320 \text{ hours} \times 10.3\% \times 51.2 \mu\text{A} = 22.8 \text{ mAh}
     \]
   - **active**
     \[
     4320 \text{ hours} \times 7.5\% \times 396 \mu\text{A} = 128.3 \text{ mAh}
     \]
   - **sleep**
     \[
     4320 \text{ hours} \times 82.2\% \times 0.32 \mu\text{A} = 1.14 \text{ mAh}
     \]

4. Sum the capacities of each period
   - \[22.8 \text{ mAh} + 128.3 \text{ mAh} + 1.14 \text{ mAh} = 152.2 \text{ mAh}\]

The capacity required to operate the circuit for six months is 152.2 mAh. Example 2 does not take into consideration temperature effects or leakage currents that are associated with batteries. The load resistance of a battery is affected by temperature which in turn changes the available voltage and current; however, the self discharge rate is higher.

EXAMPLE DESIGN

A PIC16C54 with an LP oscillator of 32.768 kHz is used in this design. A Linear Technology low-power 12-bit A/D converter samples a temperature sensor. The data is transmitted via an LED at 300 baud to a receiver. The A/D converter, op-amp, and temperature sensor are powered from an I/O pin on the PIC16C54. The Watchdog Timer is enabled to periodically wake the system up from Sleep and take a sample. Figure 6 shows the schematic for the example design and Appendix A contains the source code.

This circuit has two operating modes, active and sleep. There was not a distinct power-up region in this design. In the circuit with the peripheral chips powered directly from the battery, the example design consumed 8mA of current in the active mode and 6.5 mA in SLEEP Mode. With the peripheral chips powered from an I/O pin of the PIC16C54, the example design consumed 4 mA of current in the active mode and 0.5 µA in SLEEP Mode. The advantage of using an I/O pin to provide power to peripherals can be seen in a calculation of the capacity required to operate the circuit for one month. Example 3 details the two capacity calculations.

EXAMPLE 3: CAPACITY CALCULATION FOR THE EXAMPLE DESIGN

1. Calculate the percentage of time spent in the active and SLEEP Modes.
   - **active - battery power**
     \[
     \frac{210 \text{ ms}}{2.61 \text{ s}} \times 100 = 8\%
     \]
   - **sleep - battery power**
     \[
     \frac{2.4 \text{ s}}{2.61 \text{ s}} \times 100 = 92\%
     \]
   - **active - I/O power**
     \[
     \frac{188 \text{ ms}}{2.638 \text{ s}} \times 100 = 7.1\%
     \]
   - **sleep - I/O power**
     \[
     \frac{2.45 \text{ s}}{2.638 \text{ s}} \times 100 = 92.9\%
     \]

2. Calculate the number of hours in 1 month.
   - 1 month
     \[
     x \left( \frac{30 \text{ days}}{\text{month}} \right) \times \left( \frac{24 \text{ hours}}{\text{day}} \right) = 720 \text{ hours}
     \]

3. Using the number of hours, percentages and currents calculate the capacity for each period of time.
   - **active - battery power**
     \[
     720 \text{ hours} \times 8\% \times 8 \text{ mA} = 461 \text{ mAh}
     \]
   - **sleep - battery power**
     \[
     720 \text{ hours} \times 92\% \times 6.5 \text{ mA} = 4306 \text{ mAh}
     \]
   - **active - I/O power**
     \[
     720 \text{ hours} \times 7.1\% \times 4 \text{ mA} = 205 \text{ mAh}
     \]
   - **sleep - I/O power**
     \[
     720 \text{ hours} \times 92.9\% \times 0.5 \mu\text{A} = 0.4 \text{ mAh}
     \]

4. Sum the capacities of each period.
   - **battery power**
     \[
     461 \text{ mAh} + 4306 \text{ mAh} = 4767 \text{ mAh}
     \]
   - **I/O power**
     \[
     205 \text{ mAh} + 0.4 \text{ mAh} = 206 \text{ mAh}
     \]

The capacity required to operate this circuit for one month can be reduced by a factor of twenty just by powering the peripheral components from an I/O pin. The example design will use two Panasonic® BR2325 Lithium batteries in series to provide power to the circuit. This results in a Vbatt of 6V and a capacity of 165 mAh. Using the estimation process, the circuit should function for approximately 24 days. The actual time of operation was 24.2 days with the system running in an ambient temperature of 22°C.
SUMMARY

This application note has described some of the methods used to lower IPD and reduce overall system current consumption. Some obvious methods such as SLEEP Mode and low voltage design were given. Techniques such as powering components from I/O pins and oscillator mode and frequency selection can also be important in reducing IPD and overall system current. Some suggestions for troubleshooting IPD problems were presented. Finally, some considerations for designing a battery powered system were offered.
APPENDIX A: EXAMPLE DESIGN CODE

MPASM 01.02.05 Released LOWPWR.ASM 1-9-1995 13:2:42 PAGE 1
Ipd/Battery Apnote Example Design
LOC OBJECT CODE LINE SOURCE TEXT
VALUE

0001   TITLE "Ipd/Battery Apnote Example Design"
0002   LIST P=16C54, F=INHX8M
0003
0004   INCLUDE "C:\PICMASTR\P16C5X.INC"
0005 ; P16C5X.INC Standard Header File, Version 0.1 Microchip Technology, Inc.
0004
0005
0006 ;****************************************************************************
0007 ;***************************************************************************
0008 ; Filename:  lowpwr.asm
0009 ; REVISION: 9 Jan 95
0010 ;***************************************************************************
0011
0012 ;****************************************************************************
0013 ; This program initializes the PIC, takes a sample, and outputs the
0014 ; value to PORTB pin 0 (the LED), and then goes to Sleep. The
0015 ; Watchdog Timer wakes the device up from Sleep. PORTA pin 0 is used
0016 ; to control power to peripherals.
0017 ;***************************************************************************
0018
0019 ;****************************************************************************
0020 ;***************************************************************************
0021
0022 ; Define variable registers
0023 00010 MSB EQU OxlO
0024 00011 LSB EQU Oxll
0025 00012 DELAY_CNT EQU Oxl2
0026 00013 SHIFT EQU Ox13
0027 00014 COUNT EQU Ox14
0028
0029 ; Reset Vector
0030 01FF ORG 0x1FF
0031 00030 GOTO START
0032
0033 ; Start of main code
0034 ORG 0
0035
0036 ;***************************************************************************
0037 ; Main routine which initializes the PIC, and has main loop.
0038 ;***************************************************************************
0000 0039 START
0001 0040 MOVLW 0x2F ; 1:128 WDT PRESCALER
0002 0041 OPTION
0003 0042 MOVLW 0x02 ; RA1 SET HIGH
0004 0043 MOVWF PORTA ; ALL PINS SET TO Vss
0005 0044 CLR PORTA ; RA3-DATA INPUT
0006 0045 MOVLW 0x08 ; RA0-POWER,RA1-CS,RA2-CLOCK OUTPUTS
0007 0046 TRIS PORTA
0008 0047 CLRW ; PORTB ALL OUTPUTS, RB0-LED OUTPUT
0009 0048 TRIS PORTB
000A 0049 CLR LSB ; CLEAR A/D RESULT REGISTERS
000B 0050 CLR MSB
000C 0051
000D 0052 CLRWD
000E 0053 CALL SAMPLE ; GET SAMPLE FROM A/D
000F 0054 CLRWD
0010 0055 CALL OUTPUT ; OUTPUT SAMPLE TO LED AT 300 BAUD
0011 0056 CLRWD
0012 0057 SLEEP
0013 0058
0014 0059 ;***************************************************************************
0015 ; Main routine for retrieving a sample from the A/D.
0016 ;***************************************************************************
0017 0060 SAMPLE
0018 0061 BSF PORTA,0 ; TURN POWER ON TO PERIPHERALS
0019 0062 CALL DELAY ; WAIT FOR PERIPHERALS TO STABILIZE
001A 0063 MOVLW 0x0B ; DATA COUNTER, 12 BIT A/D
001B 0064 MOVLW 0x08 ; SET SHIFT REGISTER
001C 0065 MOVLW 0x08 ; SET SHIFT REGISTER
001D 0066 MOVWF COUNT
001E 0067 MOVWF COUNT
001F 0068 MOVWF COUNT
0020 0069 MOVWF SHIFT
0021 0070 NOP
0022 0071 BCF PORTA,1 ; ENABLE A/D
0023 0072 NOP
0024 0073 BSF PORTA,2 ; 1ST CLOCK RISE
0025 0074 NOP
0026 0075 BCF PORTA,2 ; 1ST CLOCK FALL
0027 0076 NOP
0028 0077 BSF PORTA,2 ; NULL BIT CLOCK RISE
0029 0078 NOP
002A 0079 BCF PORTA,2 ; NULL BIT CLOCK FALL
002B 0080 NOP
0022 0933
0023 0000
0024 0545
0025 0000
0026 0445
0027 0000
0028 02F4
0029 0A22
002A 0933
002B 0000
002C 0545
002D 0000
002E 0525
002F 0000
0030 0445
0031 0405
0032 0800
0033 0004
0034 0774
0035 0A3B
0036 0765
0037 0A3F
0038 0213
0039 01F0
003A 0A3F
003B 0765
003C 0A3F
003D 0213
003E 01F1
003F 0333
0040 0603
0041 0333
0042 0800
0043 0004
0044 0072
0045 02F2

0081
0082 LOOP CALL READ ;READ DATA BIT
0083 NOP
0084 BSF PORTA,2 ;BIT CLOCK RISE
0085 NOP
0086 BCF PORTA,2 ;BIT CLOCK FALL
0087 NOP
0088 DECSZ COUNT,F ;CHECK LOOP COUNTER
0089 GOTO LOOP
0090 CALL READ ;READ LAST BIT
0091 NOP
0092 BSF PORTA,2 ;SET CLOCK
0093 NOP
0094 BSF PORTA,1 ;SET CS
0095 NOP
0096 BCF PORTA,2 ;CLEAR CLOCK
0097 BCF PORTA,0 ;POWER DOWN PERIPHERALS
0098 RETURN
0099
0100 ;***************************************************************************
0101 ; Reads a bit from PORTA, data line from the A/D.
0102 ;***************************************************************************
0103 READ
0104 CLRWDT
0105 BTFSS COUNT,3 ;CHECK IF AT BIT 8 - 11
0106 GOTO RLOW ;GOTO BITS 0 - 7
0107 BTFSS PORTA,3 ;CHECK IF DATA IS CLEAR
0108 GOTO REND ;GOTO EXIT
0109 MOVF SHIFT,W ;ADD A ONE TO MSB IN THE CORRECT
0110 ADDWF MSB,F ;BIT POSITION
0111 GOTO REND
0112 BTFSS PORTA,3
0113 GOTO REND
0114 MOVF SHIFT,W ;ADD A ONE TO LSB IN THE CORRECT
0115 ADDWF LSB,F ;BIT POSITION
0116 REND RRF SHIFT,F ;SHIFT
0117 BTFSC STATUS,C ;IF ONE IS IN THE CARRY
0118 RRF SHIFT,F ;SHIFT AGAIN
0119 RETURN
0120
0121 ;***************************************************************************
0122 ; Simple delay loop for 772 clock cycles.
0123 ;***************************************************************************
0124 DELAY
0125 CLRWDT ;RESET WATCHDOG TIMER
0126 CLRF DELAY_CNT
0127 DLOOPL DECSZ DELAY_CNT,F
0046 OA45 0128 GOTO DLOOP
0047 0800 0129 RETURN
0130
0131
0132 ;**********************************************************************
0133 ; Output sample to LED at 300 baud.
0134 ;**********************************************************************
0048 0135 OUTPUT
0048 0C08 0136 MOV LW 0x08 ;SHIFT 8 MSB BITS OUT
0049 0034 0137 MOVF COUNT
0138
004A 0370 0139 MSROUT RLF MSB, F ;SHIFT LSB INTO CARRY
004B 0703 0140 BTFSS STATUS, C ;IF CARRY IS SET
004C 0A50 0141 GOTO MSBCLR
004D 0506 0142 BSF PORTB, 0 ;SET PORTB, 0
004E 0968 0143 CALL BAUD
004F 0A54 0144 GOTO MSBCK ;CHECK FOR ALL 8 BITS TO BE SENT
0050 0406 0145 MSBCLR BCF PORTB, 0 ;OTHERWISE CLEAR PORTB, 0
0051 0000 0146 NOP ;WAIT TO SET BAUD RATE 600
0052 0000 0147 NOP
0053 0968 0148 CALL BAUD
0054 02F4 0149 MSBCK DECFSZ COUNT ;CHECK FOR 8 BITS TO BE SENT
0055 OA4A 0150 GOTO MSBOUT
0151
0056 0C08 0152 MOV LW 0x08 ;SHIFT 8 LSB BITS OUT
0057 0034 0153 MOVF COUNT
0154
0058 0371 0155 LSROUT RLF LSB, F ;SHIFT LSB INTO CARRY
0059 0703 0156 BTFSS STATUS, C ;IF CARRY IS SET
005A 0A5E 0157 GOTO LSBCLR
005B 0506 0158 BSF PORTB, 0 ;SET PORTB, 0
005C 0968 0159 CALL BAUD
005D 0A62 0160 GOTO LSBCK ;CHECK FOR 8 BITS TO BE SENT
005E 0406 0161 LSBCLR BCF PORTB, 0 ;OTHERWISE CLEAR PORTB, 0
005F 0000 0162 NOP ;WAIT TO SET BAUD RATE 600
0060 0000 0163 NOP
0061 0968 0164 CALL BAUD
0062 02F4 0165 LSBCK DECFSZ COUNT ;CHECK FOR 8 BITS TO BE SENT
0063 OA58 0166 GOTO LSBOUT
0064 0406 0167 BCF PORTB, 0 ;CLEAR PORTB, 0
0065 0071 0168 CLR LF LSB ;CLEAR LSB
0066 0070 0169 CLR LF MSB ;CLEAR MSB
0067 0800 0170 RETURN
0171
0172 ;**********************************************************************
0173 ; Delay loop for sending data to the LED at 300 baud.
0174 ;**********************************************************************
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0068</td>
<td>0175</td>
<td>BAUD</td>
</tr>
<tr>
<td>0068 0000</td>
<td>0176</td>
<td>NOP</td>
</tr>
<tr>
<td>0069 0000</td>
<td>0177</td>
<td>NOP</td>
</tr>
<tr>
<td>006A 0000</td>
<td>0178</td>
<td>NOP</td>
</tr>
<tr>
<td>006B 0000</td>
<td>0179</td>
<td>NOP</td>
</tr>
<tr>
<td>006C 0000</td>
<td>0180</td>
<td>NOP</td>
</tr>
<tr>
<td>006D 0000</td>
<td>0181</td>
<td>NOP</td>
</tr>
<tr>
<td>006E 0000</td>
<td>0182</td>
<td>NOP</td>
</tr>
<tr>
<td>006F 0000</td>
<td>0183</td>
<td>NOP</td>
</tr>
<tr>
<td>0070 0000</td>
<td>0184</td>
<td>NOP</td>
</tr>
<tr>
<td>0071 0000</td>
<td>0185</td>
<td>NOP</td>
</tr>
<tr>
<td>0072 0000</td>
<td>0186</td>
<td>NOP</td>
</tr>
<tr>
<td>0073 0000</td>
<td>0187</td>
<td>NOP</td>
</tr>
<tr>
<td>0074 0000</td>
<td>0188</td>
<td>NOP</td>
</tr>
<tr>
<td>0075 0800</td>
<td>0189</td>
<td>RETURN</td>
</tr>
<tr>
<td></td>
<td>0190</td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>0191</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0192</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0193</td>
<td></td>
</tr>
</tbody>
</table>

MEMORY USAGE MAP ('X' = Used, ' ' = Unused)

```
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX-

0180 : --------------- --------------- --------------- ---------------
01C0 : --------------- --------------- --------------- ---------------X
```

All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
APPENDIX B: BATTERY DESCRIPTIONS

Presently there are two types of batteries that are manufactured, primary and secondary. Primary batteries are those that must be thrown away once their energy has been expended. Low current drain, short duty cycles, and remote operation favor primary batteries such as Carbon Zinc and Alkaline. Secondary batteries can be recharged once they have exhausted their energy. High current drain or extended usage favors secondary batteries especially when the cost of replacement of disposable batteries is not feasible. Secondary batteries include Nickel-Cadmium and Nickel Metal Hydride.

A battery may be discharged by different means depending on the type of load. The type of load will have a significant effect on the life of the battery. The typical modes of discharge are constant resistance, constant current, and constant power. Constant resistance is when the load maintains a constant resistance throughout the discharge cycle. Constant current is the mode where the load draws the same current during discharge. Finally, constant power is defined as the current during a discharge increases as the battery voltage decreases.

The constant resistance mode results in the capacity of the battery being drained at a rapid and excessive rate, resulting in a short life. This is caused by the current during discharge following the drop in battery voltage. As a result, the levels of current and power during discharge are in excess of the minimum required.

The constant current mode has lower current and power throughout the discharge cycle than the constant resistance mode. The average current drain on the battery is lower and the discharge time to the end-voltage is longer.

The constant power discharge mode has the lowest average current drain and therefore has the longest life. During discharge, the current is lowest at the beginning of the cycle and increases as the battery voltage drops. Under this mode the battery can be discharged below its end voltage, because the current is increased as the voltage drops. The constant power mode provides the most uniform performance throughout the life of the battery and has the most efficient use of the energy in the battery.

The nominal voltage is the no-load voltage of the battery, the operating voltage is the battery voltage with a load, and the end-of-life voltage is the voltage when the battery has expended its energy. Energy Density is used to describe the amount of energy per unit of volume or mass (Wh/kg or Wh/l). Generally, energy density decreases with decreasing battery size within a particular type of battery. Most batteries are rated by an amp-hour (Ah) or milliamp-hour (mAh) rating. This rating is based on a unit of charge, not energy. A 1-amp current corresponds to the movement of 1 coulomb (C) of charge past a given point in 1 second (s). Table B-1 lists some typical characteristics of the most common types of batteries.
### TABLE B-1: TYPICAL BATTERY CHARACTERISTICS

<table>
<thead>
<tr>
<th></th>
<th>Carbon Zinc</th>
<th>Alkaline</th>
<th>Nickel Cadmium</th>
<th>Lithium</th>
<th>Nickel Metal Hydride</th>
<th>Zinc Air</th>
<th>Silver Oxide</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cell Voltage</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal</td>
<td>1.5</td>
<td>1.5</td>
<td>1.2</td>
<td>3.0</td>
<td>1.2</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>Operating</td>
<td>1.25-1.15</td>
<td>1.25-1.15</td>
<td>1.25-1.00</td>
<td>2.5-3.0</td>
<td>1.25-1.0</td>
<td>1.35-1.1</td>
<td>1.5</td>
</tr>
<tr>
<td>End of life</td>
<td>0.8</td>
<td>0.9</td>
<td>0.9</td>
<td>1.75</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>Operating</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>-5°C to 45°C</td>
<td>-20°C to 55°C</td>
<td>-40°C to 70°C</td>
<td>-30°C to 70°C</td>
<td>-20°C to 50°C</td>
<td>0°C to 45°C</td>
<td>-20°C to 50°C</td>
</tr>
<tr>
<td>Energy Density</td>
<td>70</td>
<td>85</td>
<td>30</td>
<td>300</td>
<td>55</td>
<td>300</td>
<td>100</td>
</tr>
<tr>
<td><strong>Capacity</strong></td>
<td>60mAh to 18Ah</td>
<td>30mAh to 45Ah</td>
<td>150mAh to 4Ah</td>
<td>35mAh to 4Ah</td>
<td>500mAh to 5Ah</td>
<td>50mAh to 520mAh</td>
<td>15mAh to 210mAh</td>
</tr>
<tr>
<td><strong>Advantages</strong></td>
<td>High capacity, good low temp</td>
<td>good low temp, good high rate discharge</td>
<td>good low and high temp, good high rate discharge, long shelf life</td>
<td>better capacity than Nicad for same size</td>
<td>high energy density, good shelf life</td>
<td>good low temp, good shelf life</td>
<td></td>
</tr>
<tr>
<td><strong>Limitations</strong></td>
<td>Low energy density, poor low temp, poor high rate discharge</td>
<td>poor low rate discharge, disposal hazards</td>
<td>Violent reaction to water</td>
<td>Cannot stop reaction once started</td>
<td>poor high rate discharge</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Relative Cost</strong></td>
<td>low</td>
<td>low</td>
<td>medium</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Primary</td>
<td>Primary</td>
<td>Secondary</td>
<td>Primary</td>
<td>Secondary</td>
<td>Primary</td>
<td>Primary</td>
</tr>
</tbody>
</table>

Typical discharge curves for alkaline, carbon zinc, lithium, nickel cadmium, nickel metal hydride, silver oxide, and zinc air are shown in Figure B-1 through Figure B-7. These curves are only typical representations of each battery type and are not specific to any battery manufacturer. Also the load and current drain are different for each type of battery.

**FIGURE B-1: ALKALINE DISCHARGE CURVE (16 mA LOAD)**

![Alkaline Discharge Curve](image)
FIGURE B-2: CARBON ZINC DISCHARGE CURVE (16 mA LOAD)

FIGURE B-3: LITHIUM DISCHARGE CURVE (2.8 mA LOAD)

FIGURE B-4: NICKEL CADMIUM DISCHARGE CURVE (500 mA LOAD)
FIGURE B-5: NICKEL METAL HYDRIDE DISCHARGE CURVE (1500 mA LOAD)

FIGURE B-6: SILVER OXIDE DISCHARGE CURVE (1 mA LOAD)

FIGURE B-7: ZINC AIR DISCHARGE CURVE (1.3 mA LOAD)
INTRODUCTION

For any application to begin proper operation, the application must power-up properly. Many criteria must be taken into account to ensure this. The PIC16/17 devices integrate several features to simplify the design for the power-up sequence. These integrated features also reduce the total system cost.

This application note describes the requirements for the device to properly power-up, common pitfalls that designers encounter, and methods to assist in solving power-up problems.

THE POWER-UP SEQUENCE

There are several factors that determine the actual power-up sequence that a device will go through. These factors are:

- The Processor Family
  - PIC16C5X
  - PIC16CXX
  - PIC17CXX
- Oscillator Configuration
- Device Configuration
- MCLR pin

The Power-on Reset (POR) signal generation is discussed, followed by the power-up sequence for the specific device families.

Power-on Reset (POR) signal

The data sheets show a Power-on Reset (POR) pulse, as in Figure 1. The POR signal is a level triggered signal. This representation will help in the understanding of future devices, which may have a brown-out reset capability.

The power-up sequence begins by increasing the voltage on the VDD pin (from 0V). If the slope of the VDD rise time is faster than 0.05 V/μs, the internal circuitry is capable of generating an internal reset signal. Depending on the device family, different power-up sequences will occur after this POR signal.

If the slope is less than 0.05 V/μs, the MCLR pin should be held low, by external circuitry, until a valid operating VDD level is reached.

The VDD rise time specification needs to be met, until the POR signal is generated. After the POR signal is generated the slope of the VDD rise can change (to a faster or slower rise). This may have other ramifications, see the "Power-up Consideration" section. In general, the POR signal will trip (PORTPR) somewhere between 1.2V to 2.0V (Figure 1).

FIGURE 1: INTERNAL POR SIGNAL

When VDD is falling, the voltage at which the internal POR signal returns to a low level is processor/device dependent. To ensure that a device will have a POR, the device voltage must return to VSS before power is re-applied.

The POR will be generated regardless of the level of the MCLR pin. The PIC16/17 device families are different on what triggers the power-up sequence. Table 1 describes the events that cause the POR sequence to occur.

After reaching the POR trip point (PORTPR), the POR sequence holds the device in reset for a given time. Once this time has elapsed, the device voltage must be valid or the MCLR pin must be low. The time from the POR rising edge to the time that VDD must be valid level is the \( T_{POR2VDD} \) time.
**TABLE 1: EVENTS THAT TRIGGER POR SEQUENCE**

<table>
<thead>
<tr>
<th>Device</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C5X</td>
<td>Both the POR signal rising edge and any MCLR rising edge(1)</td>
</tr>
<tr>
<td>PIC16CXX</td>
<td>The POR signal rising edge</td>
</tr>
<tr>
<td>PIC17CXX</td>
<td>Either the POR signal rising edge or the first MCLR rising edge (if MCLR is low when the POR occurs). After this event, all following MCLR rising edges(1) cause the device to start program execution immediately.</td>
</tr>
</tbody>
</table>

Note 1: The POR low-to-high transition causes Special Function Register (SFR) bits/registers to a specified value. The SFR bits/registers are not identically affected by the MCLR signal. Refer to the device data sheet to see how the bits are affected by these two conditions.

The POR sequence for each of the PIC16/17 families is described in the following three sections:

- **PIC16C5X Family**
- **PIC16CXX Family**
- **PIC17CXX Family**

**PIC16C5X Family**

After the MCLR pin has reached a high level, the device is held in reset for typically 18 ms. This time is determined by an on-chip RC oscillator and 8-bit ripple counter. This Device Reset Timer (DRT), allows most crystals (except low frequency crystals) to start-up and stabilize. Due to the characteristics of resistors and capacitors, this time is extremely variable over temperature and voltage. There is also a device to device variation. See the data sheet for the range of this time-out.

**TABLE 2: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)**

<table>
<thead>
<tr>
<th>Oscillator Configuration</th>
<th>Power-up</th>
<th>Wake-up from SLEEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>XT, HS, LP(1)</td>
<td>18 ms</td>
<td>18 ms</td>
</tr>
<tr>
<td>RC</td>
<td>18 ms</td>
<td>18 ms</td>
</tr>
</tbody>
</table>

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators a typically <1 ms. All these times are voltage dependent.

**FIGURE 2: PIC16C5X POWER-UP SEQUENCE**

![PIC16C5X Power-Up Sequence Diagram]

- VDD (Supply Voltage)
- POR Signal
- MCLR (Master Clear Logic)
- VIL (Low Logic Level)
- VIH (High Logic Level)
- VTPR (POR Rising Time)
- VTPF (POR Falling Time)
- TPOR (POR Time)
- TPOR2VDDV (POR to VDDV Time)
- Reset
- Execution

DS00607A-page 2-48 © 1995 Microchip Technology Inc.
PIC16CXX Family

After the POR rising edge has occurred, the device can have up to 2 time-out sequences that occur in series. The first being the Power-up Timer (PWRT), the second being the Oscillator Start-up Timer (OST).

The Power-up Timer time-out will occur if enable fuse PWRTE is read as a ‘1’. The PWRT uses a 10-bit counter, with the clock from an internal RC. Due to the characteristics of resistors and capacitors, this time is extremely variable over temperature and voltage. There is also a device to device variation. See the data sheet for the range of this time-out.

The OST will occur on power-up/wake-up when the device has oscillator mode selected. This allows the oscillator to stabilize before program execution begins. The OST uses a 10-bit counter, with the clock from the OSC pin. The time is dependent on the frequency of the input clock. This timer is disabled if the oscillator is configured as RC.

Figure 3 shows how the two timers work in the power-up sequence. VDD must be valid when program execution starts. The TPWRT + TOST times can be thought of as the time that the device gives for the VDD to become valid (TPOR2VDDV). Figure 4 shows when device execution begins for the case of the MCLR pin going high before TPOR2VDDV times out. Figure 5 shows when the MCLR pin is held low longer than the TPOR2VDDV time. The device starts execution immediately when MCLR goes high. Table 3 gives the typical reset times.

TABLE 3: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

<table>
<thead>
<tr>
<th>Oscillator Configuration</th>
<th>Power-up</th>
<th>Wake-up from SLEEP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PWRT E = 1</td>
<td>PWRT E = 0</td>
</tr>
<tr>
<td>XT, HS, LP(1)</td>
<td>72 ms + 1024 Tosc</td>
<td>1024 Tosc</td>
</tr>
<tr>
<td>RC</td>
<td>72 ms</td>
<td>—</td>
</tr>
</tbody>
</table>

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20ms. Resonators are typically <1 ms. All these times are voltage dependent.

Note 2: Future devices will change the polarity of this configuration bit. Refer to the specific data sheet for the polarity of the PWRT Configuration Bit.
PIC17CXX Family

When the MCLR pin comes to a high level, after the POR rising edge, the device has 2 time-out sequences that occur in parallel. One is the Power-up Timer (PWRT), the other is the Oscillator Start-up Timer (OST). The timer with the greater time holds the device in reset. Figure 6 shows the sequence with MCLR tied to VDD. Figure 7 show the time-out when MCLR is independent of VDD. The PWRT time is generally longer, except for low frequency crystals/resonators. The OST time does not include the start-up time of the oscillator/resonator.

The PWRT uses a 10-bit counter, with the clock from an internal RC. The characteristics of the RC vary from device to device and over temperature and voltage. The specification for the time-out range can be found in the electrical specification of the data sheet.

The OST uses a 10-bit counter, with the clock from the OSC pin. The time is dependent on the frequency of the input clock.

Until MCLR has reached a high level, the POR sequence will not start. While the POR signal remains high, all following MCLR pulses will not cause the POR sequences to occur (Figure 8).

### TABLE 4: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

<table>
<thead>
<tr>
<th>Oscillator Configuration</th>
<th>Power-up</th>
<th>Wake-up from Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC, EC</td>
<td>Greater of 80 ms and 1024 Tosc</td>
<td>—</td>
</tr>
<tr>
<td>XT, LR(1)</td>
<td>Greater of 80 ms and 1024 Tosc</td>
<td>1024 Tosc</td>
</tr>
</tbody>
</table>

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators are typically <1 ms. All these times are voltage dependent.

### FIGURE 6: PIC17CXX POWER-UP SEQUENCE (MCLR TIED TO VDD)

### FIGURE 7: PIC17CXX POWER-UP SEQUENCE (MCLR NOT TIED TO VDD)

### FIGURE 8: MCLR OPERATION
POWER-UP CONSIDERATIONS

The device must be at a valid operating voltage when the device exits reset. This can be done by ensuring that the power supply rise time is fast enough to guarantee an operating VDD level, or by using an external reset circuit which will hold MCLR low until the operating VDD level is reached.

When the rise time of VDD is very fast, there will be a time delay before the Power-on Reset (POR) signal will rise to a logic high (TTP2PORH). This delay is in the 1-5 µs range, as shown in Figure 9.

Figure 10, Figure 11, and Figure 12 show the maximum time from the POR sequence beginning to the device having a valid operating voltage. Table 5 gives the TPOR2VDD times. When determining the time at which VDD must be valid, the POR trip point must be assumed to be at the minimum POR voltage trip point.

How Crystal Frequencies affect Start-up time

Both the PIC16CXX and PIC17CXX families may have start-up times that include the contributions of the oscillator. Table 5 shows how the oscillator can affect each mode of operation, with Table 6 giving the reset time that an oscillator generates. This time can be used in the equation to calculate the total reset time, at the given frequency. This time may vary slightly due to the initial start-up characteristics of the crystal/oscillator circuit.

<table>
<thead>
<tr>
<th>Osc Mode</th>
<th>Maximum Time</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP, XT, HS, and RC</td>
<td>9 ms</td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>28 ms</td>
<td></td>
</tr>
<tr>
<td>LP, XT, and HS</td>
<td>28 ms + 1024 Tosc</td>
<td>PWRTE = 1</td>
</tr>
<tr>
<td>LP, XT, and HS</td>
<td>1024 Tosc</td>
<td>PWRTE = 0</td>
</tr>
<tr>
<td>LF, XT, EC, and RC</td>
<td>Greater of (40 ms or 1024 Tosc)</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 6: RESET TIME DUE TO OSCILLATOR

<table>
<thead>
<tr>
<th>Clock Frequency</th>
<th>32 kHz</th>
<th>1 MHz</th>
<th>2 MHz</th>
<th>4 MHz</th>
<th>8 MHz</th>
<th>10 MHz</th>
<th>16 MHz</th>
<th>20 MHz</th>
<th>25 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 Tosc</td>
<td>32 ms</td>
<td>1.0 ms</td>
<td>512 µs</td>
<td>256 µs</td>
<td>128 µs</td>
<td>102.4 µs</td>
<td>64 µs</td>
<td>51.2 µs</td>
<td>41 µs</td>
</tr>
</tbody>
</table>
FIGURE 10: MAXIMUM POWER-UP TIME, MCLR TIED TO VDD (PIC16C5X, PIC16CXX, PIC17CXX)

Minimum Operating Vdd

Maximum POR Trip Point

Minimum POR Trip Point
Vdd and MCLR

POR signal

Reset
TPOR2VDD

Execution

FIGURE 11: MAXIMUM POWER-UP TIME, MCLR NOT TIED TO VDD (PIC16CXX)

Minimum Operating Vdd

Maximum POR Trip Point

Minimum POR Trip Point
Vdd

MCLR

POR signal

Reset
TPOR2VDD

Execution

FIGURE 12: MAXIMUM POWER-UP TIME, MCLR NOT TIED TO VDD (PIC16C5X AND PIC17CXX)

Minimum Operating Vdd

Maximum POR Trip Point

Minimum POR Trip Point
Vdd

MCLR

POR signal

Reset
TPOR2VDD

Execution
Oscillator and Resonator Considerations

Oscillators and resonators from different manufacturers may have different characteristics. The recommended capacitor selection can be found in each device's data sheet. When we do the capacitor selection, during the oscillator/resonator characterization, we are currently using devices from one of several manufacturers. Generally we use oscillators from either ECS, CTS, FOX or Epson, and ceramic resonators from either Murata Erie or Panasonic. Other manufacturers may be used in the future, depending on availability and other factors.

Other manufacturers devices may have significantly different characteristics. To ensure proper oscillator operation, the circuit should be verified at the lowest temperature/highest VDD (to ensure that the crystal is not overdriven), and with the highest temperature/lowest VDD (to ensure the device still starts up) that the device will be subjected to while in the application. This ensures a stable start-up and frequency for this device, at the extreme conditions of the application.

For production purposes, the above testing should be done with many different samples of the components selected. This is so the part to part variation of the capacitors, resistors, crystals/resonators, and PIC16/17 devices are taken into account. All PIC16/17 final data sheets supply the characterization information on the transconductance of the oscillator (measurement of gain). This information can be used to check part to part variations of the PIC16/17.

When selecting the crystal, the designer must ensure that it is a parallel cut type. Failure to use a parallel cut crystal may cause:

- Frequency operation out of the specified range of the crystal.
- Unreliable oscillator start-up.
- Device or crystal damage.

RAM and Special Function Register Initialization

After a successful Power-up Reset, the device will begin to execute the firmware program. To have expected operation, ALL RAM should be initialized by the program. This includes the Special Function Registers (SFR) and the general purpose data memory. The use (read) of an uninitialized RAM location will cause the program to do exactly what you told it to do, with the unexpected RAM value. It should not be expected that all devices will power-up with the same uninitialized device values.

There are many factors that contribute to how a RAM cell powers up, but the most common “gotcha” is between the Windowed and OTP device types. Many times a user forgets to cover the window after erasing the Windowed device. When the device is powering up, and the light is able to shine onto the device die, the transistor characteristics will shift. This can cause the device RAM to have a different power-up value than a device where no light can shine onto the die (OTP or covered).

Note: RAM locations should be initialized before they are used. Use of an uninitialized location will cause proper device operation with the improper values. That is, it will do what you told it to do, not what you wanted it to do.

Valid Operating Voltage Levels

When the device is operating, the device voltage must be within the specified Min/Max limits. Operation of the device outside these limits may cause unexpected device operation.

One of the primary functional failure modes of a device is when the applied voltage is lower than the specified minimum requirement. This functional failure is called Brown-out. Brown-out causes the program memory not to be read correctly. For example, the program counter may be pointing to a MOVE instruction, but the device reads it as a GOTO instruction (with a random destination). This can have disastrous affects to the operation of the application. If brown-out conditions are possible, the application needs to be protected by using a brown-out circuit. A brown-out circuit works with the MCLR pin to put the device in RESET before the device's actual voltage violates the minimum limit.

Figure 13 shows a low cost brown-out protection circuit. The voltage at which the circuit causes a reset is dependent upon the tolerances of the components. Figure 14 shows the use of a Dallas Semiconductor EconoReset. This device monitors the status of the power supply, and generates a reset when an out-of-tolerance condition is detected. Motorola also makes some 3-terminal devices to monitor the power supply, such as the MC34164, MC34064, MC33064. Their data sheets should be reviewed to ensure that the device is suitable for that devices application.
During a brown-out, improper program execution can occur due to an EPROM read failure. This program execution can also corrupt data memory locations, which include the Special Function Registers (SFRs). Corrupting the control registers may cause hardware conflicts. For example, an input may become an output. Other conflicts are possible, but the situation is application dependent.

As the device voltage gets lower, internal logic can become corrupted. This can include the Program Counter (PC) value, Stack Pointer and contents, State machines, Data Memory, etc.

When a valid voltage is returned, the device may be at an unexpected program location, possibly using corrupted values. In this situation, the device would not be expected to operate as intended and could get into a state that appears locked-up.

For the PIC17C42 in code protected microcontroller mode, once the Program Counter (PC) exceeds the 32K-word boundary, the device will become locked-up. The PC can exceed the 32K-word boundary from the execution of incorrect instructions (due to failure reading the EPROM) or by the PC becoming corrupted.

If the WDT is to be used to reset the device, care must be used in structuring the program. Optimally, only one `CLRWDT` instruction should be used. This minimizes the possibility of program execution returning to a loop which clears the WDT. This loop could then lock-up the device, since other control registers are corrupted and the device is not configured as expected. An example is: if the loop was waiting for an interrupt, but the bit that enables global interrupts was disabled the device would no longer respond to the interrupts and would appear locked-up.

Example 1 shows a simple implementation of using the WDT reset for system recovery. The program loops, waiting for a WDT time-out (which clears the TO bit). After the WDT reset, the TO bit needs to be set (by executing a `CLRWDT` instruction). The program should then initialize the device. Then application code can start executing. There is a possibility of the TO bit being corrupted by low voltage, and the device not being in a reset state when the software initializes the device.

The WDT example in Appendix B: uses a different method, independent of the TO bit. This uses RAM locations which get loaded with a value. A WDT time-out (or other reset) needs to occur. The RAM locations are verified to contain the same values. Once the RAM is verified, it is cleared, and the device should be initialized. These RAM locations can be used by the application program.
EXAMPLE 1: USING WDT RESET

org Reset_Address
GOTO TO_TEST ;At any reset,
;test the TO bit
org TO_TEST
BTFSC STATUS, TO ;WDT Time-Out?
HERE GOTO Time_Out ;NO, Wait for TO
HERE
GOTO Time_Out ;YES, Good Reset
CLR WDT ;Start here
: ;Initialize Device
: ;Application Code

False Power-down

In applications where power is removed from the device’s supply lines, but voltage is still applied to an I/O pin, unexpected operation may occur. Power is able to be supplied to the device through this I/O pin. Since the device is still partially powered, the internal logic is never completely powered down. Figure 15 shows the general structure of an I/O pin. Figure 16 depicts the internal voltage level that is actually applied to some device logic, versus what is seen at the pin.

To guarantee a Power-on Reset (POR) rising edge, the device voltage (Vdd) must start from Vss. When the device is inadvertently powered from an I/O pin, the voltage at the Vdd pin may appear to be near ground but may actually be higher in the device. With some of the internal logic powered, the characteristics of the device can be similar to a brown-out situation. Similar design practices to brown-out should be implemented.

A method for protecting the device from being powered from an I/O pin is shown in Figure 17.

FIGURE 15: TYPICAL ELECTRICAL STRUCTURE OF I/O PIN

FIGURE 16: FALSE POWER-DOWN

FIGURE 17: POWERED INPUT PROTECTION

In general, a brown-out detect circuit should cause the PIC16/17 to RESET (MCLR forced low). This ensures that the internal logic is in a known state until a valid device voltage level is reached. The actual brown-out circuit depends on the voltage range of the device and the application requirements. A comprehensive brown-out circuit would use a dedicated device to monitor the voltage and force the MCLR pin low when the voltage becomes lower than specified.

Another case of false power-down situations is when the power is removed from the system, but the capacitor loading keeps a non-zero voltage on the Vdd pin. When power is reapplied, the device never powered down so no power-on-reset will occur. A simple Brown-out circuit should fix this.
TROUBLESHOOTING

There are several techniques that can be used to troubleshoot problems related to powering up. First it is important to try to locate the source of the problem. These sources could be:

- No oscillation on OSC1/OSC2 pins
- Improper/no Program Execution

In cases where there is no oscillation on the OSC1/OSC2 pins, some of the following should be tried:

a) Verify that there are good connections/the components are good.
b) Verify that the crystal/resonator manufacturer is one that has been tested, if not try other capacitor values.
c) See if an external clock (from a function generator) causes device operation to begin.
d) Verify that all components are well grounded.
e) If a scope probe is connected to the oscillator output, it must be a low capacitance/high impedance probe. If it is not, the oscillator may stop.

In cases where program execution is not as expected:

a) Use a minimal program with external clock input.
b) Tie MCLR to ground until solid power is applied to the device then release MCLR (bring high).
c) Measure VDD rise time to determine if an external reset circuit is needed, and, if so, what type of reset circuit should be used.
d) Verify that the device program memory and configuration fuses are programmed to their expected states.

The flowchart shown in Figure 18 can be used to troubleshoot power-up problems. This flowchart is only intended to be the first level diagnostic in trying to solve a power-up problem. Many other flowcharts can be used, depending on the characteristics of the problem and the set-up of the application.

CONCLUSION

Understanding the criteria for the powering up of a device will allow you to make better design choices. If device power-up problems are still encountered, many techniques can be used to solve the problem. Appendix B contains example code which can be used to verify that a device is operating (powered-up correctly). This eliminates the possibility of the program as the cause, and allows debug on the hardware.
FIGURE 18: TROUBLESHOOTING FLOWCHART

1. **VCC** ramp meets specifications?
   - **Yes**: VCC in device operating range?
     - **Yes**: Hold MCLR low until VCC valid, then raise MCLR.
     - **No**: Is clock present?
       - **Yes**: Correct fuse setting?
         - **Yes**: Check the following:
           - OSC Output is not over/under driven
           - Capacitors are at their proper values
           - The Oscillator/Resonator gives the desired value
           - All Components are well grounded
         - **No**: Try using an External Clock (Select Proper Fuse Option for External Clock)
       - **No**: Is clock at desired frequency?
         - **Yes**: I/Os powering device prior to VCC rise?
           - **Yes**: Does minimal program work? (Appendix B)
             - **Yes**: Isolate device pins from voltage
             - **No**: Suspect device, programmer or circuit
           - **No**: Suspect software
APPENDIX A: Q & As

Q. When I use a windowed device (JW), my application works as expected. When I program an OTP device, it no longer works as expected. Why is this?

A. The silicon is the same between the OTP and windowed devices. If the windowed device’s window is not covered (with black tape), light shines onto the silicon. The light causes the potential levels of gates to shift. This in turn can cause RAM to be initialized to an unknown state, which could be different than in the OTP device. If RAM is not initialized by the program before it is used, these different power-up states of the RAM could be the cause of the problem. Ensure that all RAM is initialized in the device. This includes the SFRs.

Q. My oscillator is not oscillating, what could be wrong?

A. There are several possibilities, some which include:
   1. The wrong oscillator fuse setting is selected. The erased (default) state is RC oscillator mode.
   2. The wrong capacitor values are installed. Refer to the most current data sheet for recommended values.
   3. The characteristics of your manufacturers crystal are different than those that are characterized by Microchip. Generally our tests have been done with one of the following manufacturers’ crystals/resonators: ECS, FOX, Murata Erie, or Panasonic.
   4. The external connections to the device are wrong. Verify that all connections to the device are correct and that good signals/levels are being applied.
   5. The cut of the crystal is a series type, as opposed to the specified parallel type.
   6. No bypassing capacitors were used on the device. The noise on VDD could be affecting the oscillator circuitry.

Q. The device was powered-down and then powered back up, but the device does not operate. What could be wrong?

A. Possibilities include:
   1. If power was applied to an I/O pin when the device was “powered-down”, the device would be powered through the I/O pin. The internal logic is not actually powered-down, and Power-on Reset (POR) will not occur.
   2. When VDD was powered-down, VDD was not given enough time to settle to 0V.
   3. The VDD ramp rate is too slow.

Q. My oscillator is oscillating, but the device is not working. What could be wrong?

A. There are several possibilities, some which include:
   1. Slow VDD rise time, which was too slow to cause a Power-on Reset (POR). The rise time should not exceed the minimum device specification. For most devices this is 0.05 V/ms. Also the device must be at the minimum operating VDD of the processor when reset is exited.
   2. Ensure that the MCLR pin is not low. This holds the device in RESET.
   3. A brown-out has occurred, and has corrupted the internal state machines (including the WDT). An external brown-out circuit is recommended to hold the device in RESET during the brown-out condition.
   4. The CLRWDT instruction is not being used (often enough) when the WDT is enabled.

Q. When I power-up the device, it does not operate and it gets hot.

A. Your design is probably permitting fast high voltage signals (spike) onto one of the device pins. This sudden high voltage (and associated current) is in excess of the protection diode limit. The device must be powered-down (to Vss) to release this condition. This condition may cause a functional failure or affect device reliability. All Microchip devices meet or exceed the Human Body Model (HBM) and Machine Model (MM) for ESD and latch-up.

© 1995 Microchip Technology Inc.
Q. My oscillator is oscillating, but not at the expected frequency. What could be wrong.

A. For many designers, working with oscillators and their related issues are a "black magic", since the characteristics can vary widely between manufacturers. I suggest that you read all the application notes that we have available on oscillators. Some quick possibilities are:

1. The cut of the crystal is a series type, as opposed to the specified parallel type.
2. No bypassing capacitors were used on the device. The noise on VDD could affect the oscillator circuitry.
3. The capacitor values used are causing the oscillator to operate in one of the harmonic frequencies.

Note: This is not an all inclusive list. You may need to investigate other design aspects.

Q. The device seems to never exit reset, or is continually resetting.

A. The CLRWDT instruction is not being used (often enough) when the WDT is enabled.

Q. The device was powered-down and back up again, but it does not reset. It just starts operating immediately.

A. Possibilities include:

1. If power was applied to an I/O pin when the device was "powered-down", the device would be powered through the I/O pin. The internal logic is not actually powered-down, and a Power-on Reset (POR) will not occur.
2. When VDD was powered down, VDD was not given enough time to settle to 0V.

Q. The oscillator is operating (I check it with a scope), yet when I look at other pins the program is not executing. Why?

A. One possible reason is that when the oscilloscope probe is placed on the OSC2 pin, the additional capacitance is enough to cause oscillation to start. Removing the capacitive load of the probe causes the oscillation to stop.
APPENDIX B: TEST PROGRAMS

PIC16C5X BIT TOGGLE

MPASM 01.02.04 Intermediate C5X_BOT.ASM 12-20-1994 9:25:7 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT

0001 LIST P = 16C54, F = INHX8M, n = 66
0002 ;*******************************************************************************
0003 ;
0004 ; This program is a minimum program to toggle a single I/O port pin for the
0005 ; 16C5x family of devices. The only initialization is that of the data
0006 ; direction register (TRIS) of the I/O pin and the toggling of the pin.
0007 ; The waveform will be 1 unit high and 3 units low.
0008 ;
0009 ;
0010 ; Program: C5X_BOT.ASM
0011 ; Revision Date: 12-20-94
0012 ;
0013 ;*******************************************************************************
0014 ;
0015 ; HARDWARE SETUP
0016 ;
0017 ; None
0018 ;
0019 ;
0020 INCLUDE <pl6C5x.inc>
0021 ;
0022 _FUSES (_CP_OFF & _MWT_OFF & _XT_OSC )
0024 ;
0026 ;*******************************************************************************
0027 ;
0028 START
0029 CLRF STATUS
0030 MOVLW OxOO
0031 MOVLW PORTB
0032 TRIS PORTB
0033 BSF PORTB, BC
0034 GOTO lzz
0035 lzz
0036 BCF PORTB, BC
0037 GOTO lzz
0038
0039
0040
0041 ; Reset address. Determine type of RESET
0042 IFDEF 16C54
0044 01FF RESF V EQU 0x1FF
0046 ENDFIF
0047 ;
0048 IFDEF 16C54A
0049 RESF V EQU 0x1FF
0050 ENDFIF
0051 ;
0052 IFDEF 16C55
0053 RESF V EQU 0x1FF
0054 ENDFIF
0055 ;
0056 IFDEF 16C56
0057 RESF V EQU 0x3FF
0058 ENDFIF

© 1995 Microchip Technology Inc.
LOC  OBJECT CODE  LINE SOURCE TEXT
       VALUE

0059 ;
0060 IFDEF __16C57
0061 RESET_V EQU 0x7FF
0062 ENDIF
0063 ;
0064 IFDEF __16C58A
0065 RESET_V EQU 0x7FF
0066 ENDIF
0067 ;
01FE 0066 PROG_MEM_END EQU RESET_V - 1
0069 ;
0070 ;
0071 org PROG_MEM_END ; End of Program Memory
01FE 0BF0 0072 ERR_LP_1 GOTO ERR_LP_1 ; If you get here your program was lost
0073 ;
0074 org RESET_V ; RESET vector location
01FF 0A00 0075 R_VECTOR GOTO START ;
0076 ;
0077 ;
0078 end
0079
0080
0081

MEMORY USAGE MAP (‘X’ = Used, ‘-’ = Unused)

0000 : XXXXXXX-------- ------------------------ ------------------------
0040 : ------------------------ ------------------------ ------------------------
0180 : ------------------------ ------------------------ ------------------------
01C0 : ------------------------ ------------------------ XX

All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
This program is a minimam program to toggle a single I/O port pin for the 16Cxx family of devices. The only initialization is that of the data direction register (TRIS) of the I/O pin and the Toggling of the pin.

Program: ClCX_BOT.ASM
Revision Date: 12-20-94

Hardware Setup
Includ e <p16cxx.inc>

Power_On Reset (Beginning of program)
Do initialization (Bank 0)
Specify value for PortB output latch
Bank 1
Specify which PortB pins are inputs / outputs
Bank 0

Loop
End of Program Memory

© 1995 Microchip Technology Inc.
LOC OBJECT CODE LINE SOURCE TEXT

VALUE

0xFFF

IFDEF _16C74
PROG_MEM_END EQU 0xFFF
ENDIF

IFDEF _16C61
PROG_MEM_END EQU 0x3FF
ENDIF

IFDEF _16C62
PROG_MEM_END EQU 0xFFF
ENDIF

IFDEF _16C64
PROG_MEM_END EQU 0xFFF
ENDIF

IFDEF _16C65
PROG_MEM_END EQU 0xFFF
ENDIF

IFDEF _16C66
PROG_MEM_END EQU 0xFFF
ENDIF

IFDEF _16C84
PROG_MEM_END EQU 0xFFF
ENDIF

IFDEF _16C84A
PROG_MEM_END EQU 0xFFF
ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDED
This program is a minimum program to toggle a single I/O port pin for the 17Cxx family of devices. The only initialization is that of the data direction register (TRIS) of the I/O pin and the toggling of the pin. The waveform will be 1 unit high and 1 unit low.

Program:

HARDWARE SETUP

START

POWER_ON Reset (Beginning of program)

CLRF ALOSTA

Do initialization

CLRF BSR Bank 0

MOVWF PORTB

Specify value for PortB output latch

MOVWF DDRB

Specify which PortB pins are inputs / outputs

lzz BTG PORTB,

Toggle level on BO

GOTO lzz

Loop

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
This program is a minimum program to recover from a brown-out condition, thru the use of the WDT. The method is to load RAM locations with a known value and compare these locations after any RESET. If the RAM location matches the expected value then program flow can continue. The longer this RAM string is, the greater the probability that the RAM would NOT power up in that state.

**NOTE:** This does not Guarantee device recovery, Due to the random start-up point after brown-out. This point could be a loop with a CLRWD T instruction. The recommended solution is to always use a brown-out circuit.

**Program:**

```
; HARDWARE SETUP
None

TRUE
FALSE

#define __CONFIG __FUSES

INCLUDE <DEV_FAM.inc>

if ( Pl6C5X )

INCLUDE <p16C5x.inc>

__CONFIG ( _CP_OFF & _WDT_ON & _XT_OSC )
endif

if ( Pl6CXX )

INCLUDE <p16Cxx.inc>

LIST

_P16CXX. INC Standard Header File. Version 2.01 Microchip Technology, Inc.
endif

if ( Pl6C5X + Pl6CXX + Pl7CXX != 1 )

MESSG "WARNING - USER DEFINED: One and only one device family can be selected"
endif

INCLUDE <BO_RAMT.inc>

INCLUDE <PMEM_END.inc>

list

list

Start program here.
```

**Start program here.**
00069 ;
00070  if ( P16CXX )
00071  CLR PCLATH ; Force program memory to Page 0
00072  CLR STATUS ; Force Data Memory to Bank 0
00073  endif
00074 ;
00075  if ( P17CXX )
00076  CLR PCLATH, F ; Force program memory to Page 0
00077  CLR BSR, F ; Force Peripheral / GP Data Memory to Bank 0
00078  endif
00079 ;
00080  GOTO RAM_TEST ; At any reset,
00081  ; test the RAM
00082 ;
00083 ; In RAM_TEST, program execution is held-off until a valid "warm" reset
00084 ; occurs. That is, the contents of some RAM locations retain the
00085 ; values that were written to them. The probability that the RAM would power-up
00086 ; in that state is dependent on the number of bytes of RAM used. The
00087 ; more RAM, the less the probability (probability = 1 / ( 2 ** 8(N+1) ).
00088 ;
00089 ;
0100  org MAIN ; In Program Memory Page 0
0100  RAM_TEST
0100  B0A5
0101  MOVLW BYTE_0
0102  SUBWF RAM0, F ; Result = 0?
0103  goto ( P16CXX
0104  goto ( P17CXX
0105  B0F
0106  SUBWF RAM1, F ; location
0107  goto ( P16CXX
0108  goto ( P17CXX
0109  B10
010A  goto ( P16CXX
010B  goto ( P17CXX
010C  2920
010D  2921
010E  2922
010F  C110
0110  goto ( P16CXX
0111  goto ( P17CXX
0112  GOTO Time_Out ; Initialize Device
0113  ;
0114  ;
0115  ;
0116  ;
0117  ;
0118  ;
0119  ;
0120  ;
0121  ;
0122  ;
0123  ;
0124  ;
0125  ;
0126  ;
0127  ;
0128  ;
0129  ;
0130  ;
0131  ;
0132  ;
0133  ;
0134  ;
0135  ;
0136  ;
0137  ;
0138  ;
0139  ;
0140  ;
0141  ;
0142  ;
0143  ;
0144  ;
0145  ;
0146  ;
0147  ;
0148  ;
0149  ;
0150  ;
0151  ;

© 1995 Microchip Technology Inc.
00152 ;
00153 if ( P16CXX )
00154 CLRF PORTB ; PORTB output latch is cleared
00155 BSF STATUS, RP0 ; Bank 1
00156 CLRF TRISB ; Port B is output
00157 BCF STATUS, RP0 ; Bank 0
00158 BCF PORTB, 0 ;
00159 BSF PORTB, 0 ; Toggle pin B0
00160 endif
00161 ;
00162 if ( P17CXX )
0118 2912
00163 CLRF PORTB, F ; PORTB output latch is cleared
0119 2911
00164 CLRF DORB, F ; Port B is output
011A 8012
00165 BCF PORTB, 0 ;
011B 8012
00166 BSF PORTB, 0 ; Toggle pin B0
011C C117
00167 endif
00168 ;
011D C117
00169 GOTO Time_Out ; Return to start of Program
00170 ;
1FFF 00171 ;
1FFF 00172 org PROG_MEM_END ; End of Program Memory
1FFF 00173 ERR_LP_1
1FFF DFFF 00174 GOTO ERR_LP_1 ; If you get here your program was lost
00175 ;
00176 if ( P16C5X )
00177 NOP ; This will cause the Program memory rollover
00178 ; for PIC16C5x devices
00179 endif
00180 ;
00181 ;
00182 end

MEMORY USAGE MAP ( 'X' = Used, '=' = Unused)

0000 : XXX------------------------- ------------------------- -----
0040 : ------------------------------- ------------------------- -----
0100 : X------------------------------- X------------------------- -----
0140 : ------------------------------- ------------------------- ----------------
1F80 : ------------------------------- ------------------------- ----------------
1FC0 : ------------------------------- X------------------------- -----
FE00 : X----------------------------- ------------------------- -----
FE40 : ------------------------------- ------------------------- -----

All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
INTRODUCTION

The PIC16C62X devices create a new branch in Microchip's PIC16CXX 8-bit microcontroller family by incorporating two analog comparators and a variable voltage reference on-chip. The comparators feature programmable input multiplexing from device inputs and an internal voltage reference. The internal voltage reference has two ranges, each capable of 16 distinct voltage levels. Typical applications such as appliance controllers or low-power remote sensors can now be implemented using fewer external components thus reducing cost and power consumption. The 18-pin SOIC or 20-pin SSOP packages are ideal for designs having size constraints.

The PIC16C62X family includes some familiar PIC16CXX features such as:
- 8-bit timer/counter with 8-bit prescaler
- PORTB interrupt on change
- 13 I/O pins
- Program and Data Memory

<table>
<thead>
<tr>
<th>Device</th>
<th>Program Memory</th>
<th>Data Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C620</td>
<td>512 x 14</td>
<td>80 x 8</td>
</tr>
<tr>
<td>PIC16C621</td>
<td>1K x 14</td>
<td>80 x 8</td>
</tr>
<tr>
<td>PIC16C622</td>
<td>2K x 14</td>
<td>128 x 8</td>
</tr>
</tbody>
</table>

This family of devices also introduce on-chip brown-out detect circuitry and a filter on the reset input (MCLR) to the PIC16CXX mid-range microcontrollers. Brown-out Detect holds the device in reset while Vdd is below the Brown-out Detect voltage of 4.0V, ± 0.2V. The reset filter is used to filter out glitches on the MCLR pin.

This application note will describe:
- Comparator module
  - operation
  - initialization
  - outputs
- Voltage Reference module
  - operation
  - initialization
  - outputs
- Linear slope integrating Analog to Digital conversion techniques
  - advantages
  - disadvantages
- Overview of the application circuit
- Detailed description of the measurement techniques used in the application circuit
COMPARATOR MODULE

The comparator module contains two analog comparators with eight modes of operation. The inputs to the comparators are multiplexed with the RA0 through RA3 pins. The on-chip voltage reference can also be selected as an input to the comparators. The Comparator Control Register (CMCON) controls the operation of the comparator and contains the comparator output bits. Figure 1 shows the CMCON register.

FIGURE 1: CMCON REGISTER

<table>
<thead>
<tr>
<th>R</th>
<th>R</th>
<th>U</th>
<th>U</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2OUT</td>
<td>C1OUT</td>
<td>-</td>
<td>-</td>
<td>CIS</td>
<td>CM2</td>
<td>CM1</td>
<td>CM0</td>
</tr>
</tbody>
</table>

| bit7 | bit0 |

Register: CMCON
Address: 1Fh
POR Value: 00h

R: Readable & W: Writable
U: Unimplemented, read as '0'

CM<2:0>: Comparator mode
See Figures 3 through 10.

CIS: Comparator Input Switch
When CM<2:0>= 001:
1 = C1 VIN− connects to RA3
0 = C1 VIN− connects to RA0

When CM<2:0>= 010:
1 = C1 VIN− connects to RA3, C2 VIN− connects to RA2,
0 = C1 VIN− connects to RA0, C2 VIN− connects to RA1

C1OUT: Comparator 1 output
1 = C1 VIN+ > C1 VIN−
0 = C1 VIN+ < C1 VIN−

C2OUT: Comparator 2 output
1 = C2 VIN+ > C2 VIN−
0 = C2 VIN+ < C2 VIN−
A single comparator is shown in Figure 2. The relationship between the inputs and the output is also shown. When the voltage at VIN+ is less than the voltage at VIN-, the output of the comparator is at a digital low level. When the voltage at VIN+ is greater than the voltage at VIN-, the output of the comparator is at a digital high level. The shaded areas of the comparator output waveform represent the uncertainty due to input offsets and response time.

FIGURE 2: SINGLE COMPARATOR

The TRISA register controls the I/O direction of the PORTA pins regardless of the comparator mode. If the comparator mode configures a pin as an analog input and the TRISA register configures that pin as an output, the contents of the PORTA data latch are placed on the pin. The value at the pin, which can be a digital high or low voltage, then becomes the input signal to the comparators. This technique is useful to check the functionality of the application circuit and the comparator module.

Comparator Operating Modes

The analog inputs to the comparator module must be between Vss and Vdd and one input must be in the Common Mode Range (CMR). The CMR is defined as Vdd-1.5 volt to Vss. The output of a comparator will default to a high level if both inputs are outside of the CMR. If the input voltage deviates above Vdd or below Vss by more than 0.6 volt, the microcontroller may draw excessive current. A maximum source impedance to the comparators of 10 kΩ is recommended. Figure 3 through Figure 10 show the eight modes of operation.

FIGURE 3: COMPARATORS RESET

The Comparators Reset Mode (Figure 3) is considered the lowest power mode because the comparators are turned off and RA0 through RA3 are analog inputs. The comparator module defaults to this mode on Power-on Reset.

FIGURE 4: COMPARATORS OFF

The Comparators Off Mode (Figure 4) is the same as the Comparators Reset Mode except that RA0 through RA3 are digital I/O. This mode may consume more current if RA0 through RA3 are configured as inputs and the pins are left floating.

FIGURE 5: TWO INDEPENDENT COMPARATORS

The Two Independent Comparators Mode (Figure 5) enables both comparators to operate independently.
The Four Inputs Multiplexed to Two Comparators Mode (Figure 6) allows two inputs into the VIN- pin of each comparator. The internal voltage reference is connected to the VIN+ pin input of each comparator. The CIS bit, CMCON<3>, controls the input multiplexing to the VIN- pin of each comparator. Table 1 shows this relationship.

<table>
<thead>
<tr>
<th>CIS</th>
<th>C1 VIN-</th>
<th>C2 VIN-</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RA0</td>
<td>RA1</td>
</tr>
<tr>
<td>1</td>
<td>RA3</td>
<td>RA2</td>
</tr>
</tbody>
</table>

The Two Common Reference Comparators Mode (Figure 7) configures the comparators such that the signal present on RA2 is connected to the VIN+ pin of each comparator. RA3 is configured as a digital 1/0 pin.

The Two Common Reference Comparators with Outputs Mode (Figure 8) connects the outputs of the comparators to an I/O pin. These outputs are digital outputs only with RA3 defined as a CMOS output and RA4 defined as an open drain output. RA4 requires a pull-up resistor to function properly. The value of resistance used for the pull-up will affect the response time of comparator C2. The signal present on RA2 is connected to the VIN+ pin of both comparators.

The One Independent Comparator Mode (Figure 9) turns comparator C1 off making both RA0 and RA3 digital I/O. Comparator C2 is operational with analog inputs from RA1 and RA2.
FIGURE 10: THREE INPUTS MULTIPLEXED TO TWO COMPARATORS

The Three Inputs Multiplexed to Two Comparators Mode (Figure 10) connects the VIN+ pin of each comparator to RA2. The VIN- pin of comparator 2 is connected to RA1. The CIS bit, CMCON<3>, controls the input to the VIN- pin of comparator 1. If CIS = 0, then RA0 is connected to the VIN- pin. Otherwise RA3 is connected to the VIN- pin of comparator 1.

Note: Each comparator that is active will consume less power when the output is at a high level.

Clearing the Comparator Interrupt Flag

The comparator interrupt flag, CMIF, is located in the PIR1 register. This flag must be cleared after changing comparator modes. Whenever the comparator mode or the CIS bit is changed, the CMIF may be set due to the internal circuitry switching between modes. Therefore, comparator interrupts should be disabled before changing modes. Then, a delay of 10 µs should be used after changing modes to allow the comparator circuitry to stabilize.

The steps to clear the CMIF flag when changing modes are as follows:

- Change the comparator mode or CIS bit
- 10 µs delay
- Read the CMCON register to end the “mismatch” condition
- Clear the CMIF bit of the PIR1 register

The value of C1OUT and C2OUT are internally latched on every read of the CMCON register. The current values of C1OUT and C2OUT are compared with the latched values, and when these values are different a “mismatch” condition occurs. The CMIF interrupt flag will not be cleared if the CMCON register has not been read.

Using the Comparator Module

The CMCON register contains the comparator output bits C1OUT and C2OUT, CMCON<7:6>. These bits are read only. C1OUT and C2OUT follow the output of the comparators and are not synchronized to any internal clock edges. Therefore, the firmware will need to maintain the status of these output bits to determine the actual change that has occurred. The PIR1 register contains the comparator interrupt flag CMIF, PIR1<6>. The CMIF bit is set whenever there is a change in the output value of either comparator relative to the last time the CMCON register was read.

Note: If a change in C1OUT or C2OUT should occur when a read operation on the CMCON register is being executed (start of the Q2 cycle), the CMIF interrupt flag may not be set.

When reading the PORTA register, all pins configured as analog inputs will read as a '0'. Analog levels on any pin that is defined as a digital input may cause the input buffer to consume more current than is specified.

The code in Example 1 shows the steps required to configure the comparator module. RA3 and RA4 are configured as digital outputs. RA0 and RA1 are configured as the VIN- inputs to the comparators and RA2 is the VIN+ input to both comparators.

EXAMPLE 1: INITIALIZING THE COMPARATOR MODULE

```
CLRF PORTA ;init PORTA
MOVLW 0x03 ;Two Common
MOVF CMCON ;Reference
;Comparators
;mode selected
BSF STATUS,RP0 ;go to Bank 1
MOVLW 0x07 ;Set RA<2:0> as
MOVWF TRISA ;inputs, RA<4:3>
;as outputs
BCF STATUS,RP0 ;go to Bank 0
CALL DELAY10 ;10µs delay
MOVF CMCON,F ;read the CMCON
BCF PIR1,CMIF ;clear the CMIF
BSF STATUS,RP0 ;go to Bank 1
BSF PIE1,CMIE ;enable comparator
BCF STATUS,RP0 ;go to Bank 0
BSF INTCON,PEIE ;enable peripheral
BSF INTCON,GIE ;interrupts
```

The comparators will remain active if the device is placed in sleep mode, except for the Comparators Off Mode (CM<2:0>=111) and Comparators Reset Mode (CM<2:0>=000). In these modes the comparators are turned off and are in a low power state. A comparator interrupt, if enabled, will wake-up the device from sleep in all modes except Off and Reset.
Comparator Timings

The comparator module has a response time and a mode change to output valid timing associated with it. The response time is defined as the time from when an input to the comparator changes until the output of that comparator becomes valid. The response time is faster when the output of the comparator transitions from a high level to a low level. The mode change to output valid time refers to the amount of time it takes for the output of the comparators to become valid after the mode has changed. The internal voltage reference may contribute some delay if used in conjunction with the comparators (see Voltage Reference Settling Time).

FIGURE 11: VRCON REGISTER

VOLTAGE REFERENCE MODULE

The voltage reference is a 16-tap resistor ladder network that is segmented to provide two ranges of VREF values. Each range has 16 distinct voltage levels. The voltage reference has a power-down function to conserve power when the reference is not being used. The voltage reference also has the capability to be connected to RA2 as an output. Figure 11 shows the Voltage Reference Control Register (VRCON) register which controls the voltage reference. Figure 12 shows the block diagram for the voltage reference module.

FIGURE 11: VRCON REGISTER

<table>
<thead>
<tr>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>U</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>VREN</td>
<td>VROE</td>
<td>VRR</td>
<td>-</td>
<td>VR3</td>
<td>VR2</td>
<td>VR1</td>
<td>VR0</td>
<td></td>
</tr>
</tbody>
</table>

| bit7 | bit0 |

Register: VRCON
Address: 9Fh
POR Value: 00h

R: Readable
W: Writable
U: Unimplemented, read as '0'

VR<3:0>: VREF value selection 0 ≤ VR [3:0] ≤ 15
when VRR = 1: VREF = (VR<3:0>/ 24) * VDD
when VRR = 0: VREF = 1/4 * VDD + (VR<3:0>/ 32 * VDD)

VRR: VREF Range selection
1 = Low Range
0 = High Range

VROE: VREF Output Enable
1 = VREF is output on RA2 pin
0 = VREF is disconnected from RA2 pin

VREN: VREF Enable
1 = VREF circuit powered on
0 = VREF circuit powered down, no IDD drain

FIGURE 12: VOLTAGE REFERENCE BLOCK DIAGRAM

Note: The voltage reference is VDD derived and therefore, the VREF output changes with fluctuations in VDD.
Using the Voltage Reference

The voltage reference module operates independently of the comparator module. The output of the voltage reference may be connected to the RA2 pin at any time by setting the TRISA<2> bit and the VRCON<6> bit (VROE). It should be noted that enabling the voltage reference with an input signal present will increase current consumption. Configuring the RA2 pin as a digital output with the VREF output enabled will also increase current consumption. The increases in current are caused by the voltage reference output conflicting with an input signal or the digital output. The amount of increased current consumption is dependent on the setting of VREF and the value of the input signal or the digital output.

The full range of Vss to VDD cannot be realized due to the construction of the module (Figure 12). The transistors on the top and bottom of the resistor ladder network keep VREF from approaching Vss or VDD. Equation 1 and Equation 2 are used to calculate the output of the voltage reference.

EQUATION 1: VOLTAGE REFERENCE EQUATION, VRR=1

\[ V_{REF} = \left( V_{R<3:0>/24} \right) \times V_{DD} \]

EQUATION 2: VOLTAGE REFERENCE EQUATION, VRR=0

\[ V_{REF} = \left( V_{DD}/4 \right) + \left( V_{R<3:0>/32} \right) \times V_{DD} \]

An example of how to configure the voltage reference is given in Equation 2. The reference is set for an output voltage of 1.25V at a VDD of 5.0V.

EXAMPLE 2: VOLTAGE REFERENCE CONFIGURATION

```
MOVLW 0x02 ;4 Inputs Muxed
MOVWF CMCON ; to 2 comps.
BSF STATUS.RPO ; go to Bank 1
MOVLW 0x07 ;RA3-RA0 are
MOVWF TRISA ; outputs
MOVLW 0xA6 ; enable VREF,
MOVWF VRCON ; low range
; set V<s:0>=6
BCF STATUS,RPO ; go to Bank 0
CALL DELAY10 ; 10µs delay
```

If the voltage reference is used with the comparator module, the following steps should be followed when making changes to the voltage reference.

1. Disable the comparator interrupts
2. Make changes to the voltage reference
3. Delay 10 µs to allow VREF to stabilize
4. Delay 10 µs to allow comparators to settle
5. Clear the comparator interrupt flag
   - Read the CMCON register
   - Clear the CMIF bit
6. Enable comparator interrupts

The output of the voltage reference may be used as a simple DAC. However, the VREF output has limited drive capability when connected to the RA2 pin. In fact the amount of drive the voltage reference can provide is dependent on the setting of the tap on the resistor ladder. If VREF is used as an output, an external buffer must be utilized.

Voltage Reference Settling Time

Settling time of the voltage reference is defined as the time it takes the output voltage to settle within 1/4 LSB after making a change to the reference. The changes include adjusting the tap position on the resistor ladder, enabling the output, and enabling the reference itself. If the voltage reference is used with the comparator module, the settling time must be considered.
MAKING SIMPLE A/D CONVERSIONS

Linear slope integrating A/D converters are very simple to implement and can achieve high linearity and resolution for low conversion rates. The three types of converters that will be discussed are the single-slope, dual-slope, and modified single-slope converters. The following material was referenced from application note AN260, "A 20-Bit (1ppm) Linear Slope-Integrating A/D Converter", found in the Linear Applications Handbook from National Semiconductor®.

Single-Slope Integrating Converter

A single-slope integrating converter is shown in Figure 13. In a single-slope converter, a linear ramp is compared against an unknown input AIN. When the switch S1 is opened the ramp begins. The time interval between the opening of the switch and the comparator changing state is proportional to the value of AIN.

The basic assumptions are that the integrating capacitor C1 and the clock used to measure the time interval remain constant over time and temperature. This type of converter is heavily dependent on the stability of the integrating capacitor.

FIGURE 13: SINGLE-SLOPE INTEGRATING CONVERTER

S1
Comparator

Dual-Slope Integrating Converter

Figure 14 shows a dual-slope integrating converter. The dual-slope converter integrates the AIN input for a predetermined length of time. The voltage reference is then switched into the integrator input, using S2, which integrates in a negative direction from the AIN slope. The length of time the reference slope requires to return to zero is proportional to the value of AIN. Both slopes are made with the same integrating capacitor C1 and measured with the same clock, so they need only to be stable over one conversion cycle.

FIGURE 14: DUAL-SLOPE INTEGRATING CONVERTER

The dual-slope converter essentially removes the stability factor of the integrating capacitor from a conversion, however, the dielectric absorption of C1 has a direct effect. Dielectric absorption not only creates residual non-linearity in the dual-slope converter, but causes the converter to output different values for a fixed input as the conversion rate is varied. Dielectric absorption is defined as the capacitor dielectric's unwillingness to accept or give up charge instantaneously. This effect is modeled as a parasitic RC network across the main capacitor. A charged capacitor will require some time to discharge, even through a dead short, due to the parasitic RC network and some amount of charge will be absorbed by the parasitic C after charging of the main capacitor has stopped. Typically, Teflon, polystyrene and polypropylene dielectrics offer better performance than paper, mylar, or glass. Electrolytics have the worst dielectric absorption characteristics and should be avoided for use in slope integrating converters.
Modified Single-Slope Converter

The modified single-slope converter has been designed to compensate for the effects present in the previous converters. Resolutions of up to 16-bits can be achieved using high precision components and voltage reference source. Figure 15 shows the modified single-slope converter. Some features of this converter are:

- Continuously corrects for zero and full-scale drifts in all components of the circuit.
- The integrating capacitor C1 is charged periodically and always in the same direction. The error induced from dielectric absorption will be small and can be compensated by using an offset term in the calibration procedure.
- The ramp voltage always approaches the comparator trip point from the same direction and slew rate.
- There is no noise rejection capability because the input signal is directly coupled to the comparator input. A filter at the comparator input would cause a delay due to the settling time of the filter.

**FIGURE 15: MODIFIED SINGLE-SLOPE INTEGRATING CONVERTER**

The microcontroller sends a periodic signal to the switch S1 regardless of the operating mode of the system. The output of the integrator is a fixed frequency, period and height signal which is fed into the input of the comparator. The time between ramps is long enough to allow the integrating capacitor C1 to discharge completely. The other input is multiplexed with ground, reference, and the AIN through switch S2. When the microcontroller starts a conversion, the ground signal is switched into the comparator and the time for the ramp to cross zero is measured and stored. The same measurements are repeated for the reference and AIN signals. Assuming that the integrator ramps are highly linear, Equation 3 is used to determine the value of AIN.

**EQUATION 3: OUTPUT EQUATION FOR THE MODIFIED-SLOPE CONVERTER**

\[
AIN = \frac{t_{AIN} - t_{GND}}{t_{VREF} - t_{GND}} \times K \ \mu V
\]

where \( t_{AIN} \) is the measured time for the AIN signal, \( t_{VREF} \) is the measured time for the voltage reference signal, \( t_{GND} \) is the measured time for the ground signal, and \( K \) is a constant (typically \( 10^7 \)).

**APPLICATION CIRCUIT**

The application circuit, called PICMETER, uses a PIC16C622 as a resistance and capacitance meter. The PICMETER uses a variation of the single-slope integrating converter. The linear slope and integrator of Figure 13 are replaced with the exponential charge waveform of a RC Network. The charge time of a known component is compared against the charge time of an unknown component to determine the value of the unknown component.

A schematic of the PICMETER is shown in Figure 16. All reference designators cited in this section refer to this schematic. Results are transmitted to a PC which displays the value measured. The PICMETER can measure resistance in the range 1 kΩ to 999 kΩ and capacitance from 1 nF to 999 nF.

The following sections describe, in detail, the hardware, firmware, and PC software used in the application circuit. Appendix A shows the PICMETER firmware and Appendix B has the PC software. Appendix C contains the PCB layout.
Power

The RS-232 serial port provides power to the PICMETER. The RTS and DTR lines from the serial port output 3V to 11V to the PICMETER. The diodes D2 and D3 prevent any damage to the PC’s serial port. Resistor R10 is used to current limit the Zener diode, D4. D4 is used to regulate the RTS and DTR voltage to 5.6V. Capacitors C3 and C4 provide power supply filtering to the Zener diode and the PIC16C622. This method of supplying power to devices using a serial port, such as a trackball or mouse, is very simple considering that the PICMETER requires approximately 7 mA to function.

Switches

Switch S1 is used to select either a resistor or capacitor measurement. RB5 of the PIC16C622 is used to detect what type of component is being measured. This switch also swaps the unknown component into the RC network. If a resistor is the unknown component and a capacitor measurement is requested, the circuit reduces to a resistor divider on the VIN- pin of the comparator. This would result in a measured value of 0 pF if the voltage on the resistor divider network is greater than the voltage reference setting. Otherwise an error is detected. If a capacitor is the unknown component and a resistor measurement is selected, the circuit reduces to a capacitor divider network on the VIN- pin of the comparator. This case will also produce an error message.

Resistor measurements that are started without any component connected to the measuring terminals will cause an error. Capacitor measurements without a component connected to the measuring terminals will give a result of 0 pF.

Switch S2 is used to initiate a measurement. The switch is connected to RB6 of the PIC16C622 and the PORTB wake-up on change interrupt is used to detect a key press. A modified version of the firmware in AN552, “Implementing Wake-up on Key Stroke” was used to control the interrupt.

Measuring the Charge Time

The procedures for measuring a resistor or capacitor are the same except for the I/O pins used to control the RC networks. This also applies when measuring a known or unknown component.

Measurement Overview

The charge time of the unknown RC network is measured using Timer0. This value is multiplied by the known value of resistance or capacitance and stored in an accumulator. Then the charge time of the known RC network is measured. The accumulator is divided by the known RC network charge time to give the value of resistance or capacitance of the unknown component. Equation 4 shows the equation used to calculate resistance and Equation 5 shows the capacitance equation.

EQUATION 4: RESISTANCE EQUATION

\[ R_{UNK} = \frac{t_{UNK} \times R_{KN}}{t_{KN}} \]

EQUATION 5: CAPACITANCE EQUATION

\[ C_{UNK} = \frac{t_{UNK} \times C_{KN}}{t_{KN}} \]

\( R_{UNK} \) and \( C_{UNK} \) are the unknown resistor or capacitor values. \( R_{KN} \) and \( C_{KN} \) are the known resistor and capacitor values. \( t_{UNK} \) and \( t_{KN} \) are the charge times for the unknown and known components.
Detailed Measurement Description

The first step in measuring the charge time of either the known or the unknown RC networks is to reconfigure the I/O pins. The default state of the PORTA and PORTB pins connected to the RC network are all grounded outputs. This discharges all capacitors in the RC networks. The unknown component is measured first, so the known component, R4 or C1, is removed from the RC network. This is accomplished by making RB0 or RB2 on the PIC16C622 an input. Connections to the other RC network are kept grounded.

The analog modules are now initialized. The mode of the comparators is set to Four Inputs Multiplexed to Two Comparators (Figure 6). The CIS bit, CMCON<3>, is cleared to select RA0 as the VIN- input to comparator 1 and RA1 as the VIN- input to comparator 2. The voltage reference is enabled, the output is disabled, and the high range is selected. The tap on the resistor ladder is set to 12. The value of 12 was selected because it is the lowest value of VREF that will trip the comparators, yet gives a time constant long enough to achieve good resolution for the measurement. After a 20 msec delay, which allows the analog modules to stabilize, the comparator flag is cleared. Comparator interrupts are enabled and Timer0 is cleared. Finally, the PEIE bit is set to enable comparator interrupts and the GIE bit is set to enable interrupts.

Now that the analog systems are ready, Timer0 is cleared again and power is applied to the unknown RC network by setting RB1 or RB3 high. Timer0 begins to increment a set of three registers which are cascaded together. These registers contain the charge time of the component. While waiting for the DONE flag, the ERROR flag is checked. See the Error Message section for an explanation of error detection. When the capacitor voltage trips the comparator, Timer0 is prevented from further incrementing the time registers and the DONE flag is set. The value in the time registers is tUNK.

The analog modules are now disabled. The comparator interrupts are disabled and the comparators are turned off (CM<2:0>=111). RA0 through RA3 and RB0 through RB4 are set up as grounded outputs to discharge the capacitors in the RC networks. This prevents a false reading during the next measurement. The voltage reference is disabled to conserve power and all interrupt flags are cleared. Extra delay loops are added at this time to ensure that the capacitors are discharged.

The charge time, tUNK, is then multiplied by the value of known resistance or capacitance. These values, in pF or Ω, were obtained by measuring the known RC networks with a Fluke meter. Each of these values is a 24-bit number. The result of multiplication is a 56-bit number which is stored in accumulators ACCb (most significant 24-bits) and ACCc (least significant 24-bits).

The process now repeats itself, except this time the charge time of the known RC network is measured. Now the unknown component is removed from the RC network by making the connections from the PIC16C622 inputs. The analog modules are initialized and the same procedure explained above is followed to measure the charge time of the known RC network. The 56-bit result previously stored in accumulators ACCb and ACCc is now divided by the charge time of the known component, tKN. This result is a 24-bit number which has the units of pF or Ω. This value is then transmitted to the PC.
RS-232 Transmission

PICMETER uses a transmit only, software implemented serial port adapted from AN593, "Serial Port Routines Without Using the RTCC". Hardware hand-shaking is not used. Since the serial port is realized in software, all interrupts must be disabled during transmission otherwise the baud rate can get corrupted.

On power-up, PICMETER sends a boot message to the PC which is "PICMETER Booted!". Otherwise, a four byte packet structure with a command byte and 3 data bytes is used. The command byte contains one of four possible commands:

- ASCII 'S' signifies that a measurement has been initiated
- ASCII 'E' tells the PC that an error has been detected
- ASCII 'R' tells the PC that resistance data is contained in the three data bytes
- ASCII 'C' tells the PC that capacitance data is contained in the three data bytes

The first data byte for the 'R' and 'C' commands contain the MSB of the measured value. The last data byte contains the LSB of the measured value. The three data bytes for the commands 'S' and 'E' do not contain any useful information at this time.

An 'S' command is issued every time the start switch, S2, is pressed. PICMETER then sends an 'R' or 'C' command for a valid measurement or an 'E' command when an error is detected.

Since the PICMETER operates from a single supply voltage, a discrete transistor is used as a level shifter. This insures that a low output on the RS-232 TXD line is between -3V and -11V. When the TXD line, RB7, from the PIC16C622 is at a logic high level, the transistor Q1 is off. The RXD line of the computer will then be at approximately the same voltage as the TXD line, -11V to -3V. A logic low level from RB7 of the PIC16C622 will turn on transistor Q1. This will bring the RXD line of the computer to about the same voltage of the DTR or RTS line, +3V to +11V.

The pins of interest on the DB9 connector CON1 are:

- pin 2 - RXD
- pin 3 - TXD
- pin 4 - DTR
- pin 5 - GND
- pin 7 - RTS

RTS, DTR, and GND provide power and ground to the PICMETER. RXD is connected to the collector of transistor Q1. TXD is connected to RXD through resistor R14. Since hardware hand-shaking is not implemented on the PICMETER, DSR (pin 6) and CTS (pin 8) are left disconnected.

The demo board developed by Microchip was intended to connect directly to a 9-pin serial port. A 9-pin male-to-female cable may also be used. These boards were manufactured by Southwest Circuits located in Tucson, Arizona (Appendix C). The PCB layout for this demo board is shown in Appendix C.

Error Message

The error message is sent only when the PICMETER is making a measurement and detects an error. The range of resistance that the PICMETER measures is 1 kΩ to 999 kΩ. Using the value of C2, 1 µF, the range of charging times for resistance measurements is 1 msec to 999 ms. The range of capacitor charging times is also 1 ms to 999 ms using the resistance value of R3, 1 MΩ, and a capacitor measuring range of 1 nF to 999 nF. A ceramic resonator of 4 MHz gives Timer0 a resolution of 1 µsec. Therefore, the highest count that the time registers should reach is 999,000. This is a 20-bit number. If the 21st bit should ever be set, it is assumed that the PICMETER is trying to measure the air gap between the measuring terminals, a component that is out of range, or switch S1 is not set correctly for the component in the measuring terminals.

24-Bit Math Routines

The 24-bit math routines were developed using simple algorithms found in any computer math book. These math routines include addition, subtraction, multiplication, division, and 2's complement. Four 24-bit accumulators located in the general purpose RAM area of the PIC16C622 are used by the math routines: ACCa, ACCb, ACCc, and ACCd. Table 2 shows the relationship between the math routines and the accumulators.

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
<th>Result</th>
<th>Temp. Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>ACCa + ACCb</td>
<td>ACCb</td>
<td>N/A</td>
</tr>
<tr>
<td>Subtract</td>
<td>2's Comp ACCa then</td>
<td>ACCa</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>ACCa + ACCb</td>
<td>ACCa</td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td>ACCb x ACCc</td>
<td>ACCb (MSB's) ACCc (LSB's)</td>
<td>ACCd</td>
</tr>
<tr>
<td>Divide</td>
<td>ACCb:ACCc ACCa</td>
<td>quotient in ACCc remainder in ACCb</td>
<td>ACCd</td>
</tr>
<tr>
<td>2's Comp</td>
<td>NOT(ACCa) + 1</td>
<td>ACCa</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Computer Program

The program that receives data from the PICMETER was written in Visual Basic® from Microsoft® for the Windows® environment. Figure 17 show the display of the Windows based PICMETER program.

FIGURE 17: PICMETER PC PROGRAM

The operation of this program is simple. A functional description is given below:

a) Select the appropriate COM port by clicking on the COM1 or COM2 buttons.

b) Turn power on to the PICMETER by clicking on the PICMETER Power button.

c) The frame message should read "PICMETER Booted!", the frame contents will be cleared, and the LED on the PICMETER should be on.

d) The switch S1 selects the type of component that is in the measuring terminals.

e) Pressing the START button, S2, on the PICMETER will initiate a measurement. The frame message should read "Measuring Component" and the contents of the frame will be cleared.

f) When the measurement is complete, the frame message will read "Resistance" or "Capacitance" depending on the position of switch S1. The value of the component will be displayed in the frame as well as the units.

g) If an error is detected, the frame message will read "Error Detected". This is only a measurement error. Check the component on the measuring terminals and the position of switch S1.

h) Turn off the PICMETER by clicking on the PICMETER Power button. The frame message will change to "PICMETER Power OFF", the frame contents will be cleared, and the LED on the PICMETER will turn off.

Appendix B contains a complete listing of the Visual Basic program.
PICMETER ACCURACY

The PICMETER measures capacitance in the range of 1 nF to 999 nF. Table 3 shows a comparison of various capacitors. All capacitors have a tolerance of 10% and have various dielectrics. The average error percentage is 3%.

TABLE 3: CAPACITANCE MEASUREMENTS

<table>
<thead>
<tr>
<th>Capacitance Accuracy</th>
<th>Fluke Value</th>
<th>PICMETER Value</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marked Value</td>
<td>2.2 nF</td>
<td>2.3 nF</td>
<td>4.3</td>
</tr>
<tr>
<td>2.2 nF</td>
<td>2.5 nF</td>
<td>2.63 nF</td>
<td>4.9</td>
</tr>
<tr>
<td>20 nF</td>
<td>16.5 nF</td>
<td>16.3 nF</td>
<td>1.2</td>
</tr>
<tr>
<td>33 nF</td>
<td>35.2 nF</td>
<td>35.8 nF</td>
<td>1.7</td>
</tr>
<tr>
<td>47 nF</td>
<td>45 nF</td>
<td>44.5 nF</td>
<td>1.1</td>
</tr>
<tr>
<td>50 nF</td>
<td>52 nF</td>
<td>52.9 nF</td>
<td>1.7</td>
</tr>
<tr>
<td>100 nF</td>
<td>99.7 nF</td>
<td>93 nF</td>
<td>6.7</td>
</tr>
<tr>
<td>0.1 µF</td>
<td>95 nF</td>
<td>96.1 nF</td>
<td>1.2</td>
</tr>
<tr>
<td>0.22 µF</td>
<td>215 nF</td>
<td>215.2 nF</td>
<td>0.1</td>
</tr>
<tr>
<td>470 nF</td>
<td>508 nF</td>
<td>518.9 nF</td>
<td>2.1</td>
</tr>
<tr>
<td>940 nF</td>
<td>922 nF</td>
<td>983.1 nF</td>
<td>6.6</td>
</tr>
</tbody>
</table>

The 2.5 nF, 100 nF and 940 nF capacitors all have polyester dielectric material. The Equivalent Series Resistance (ESR) of polyester capacitors is typically high which would cause the PICMETER to have a larger error than other dielectrics. If the error percentages for these capacitors is ignored, the average error decreases to 1.9%.

The resistance range of the PICMETER is 1 kΩ to 999 kΩ. Table 4, Resistance Measurements, shows a comparison of various resistors in this range. All resistors have a tolerance of 5%. The average error percentage is 1%.

TABLE 4: RESISTANCE MEASUREMENTS

<table>
<thead>
<tr>
<th>Resistance Accuracy</th>
<th>Marked Value</th>
<th>Fluke Value</th>
<th>PICMETER Value</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2K</td>
<td>1.215K</td>
<td>1.2K</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>5.1K</td>
<td>5.05K</td>
<td>5.0K</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>8.2K</td>
<td>8.47K</td>
<td>8.3K</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>10K</td>
<td>10.2K</td>
<td>10K</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>15K</td>
<td>15.36K</td>
<td>15.1K</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>20K</td>
<td>20.8K</td>
<td>20.5K</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>30K</td>
<td>30.4K</td>
<td>30K</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>51K</td>
<td>50.3K</td>
<td>49.8K</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>75K</td>
<td>75.5K</td>
<td>74.8K</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>91K</td>
<td>96.4K</td>
<td>95.9K</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>150K</td>
<td>146.3K</td>
<td>145.6K</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>200K</td>
<td>195.5K</td>
<td>195K</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>300K</td>
<td>309K</td>
<td>309.5K</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>430K</td>
<td>433K</td>
<td>434.5K</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>560K</td>
<td>596K</td>
<td>599.6K</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>680K</td>
<td>705K</td>
<td>709.8K</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>820K</td>
<td>901K</td>
<td>907.3K</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>910K</td>
<td>970K</td>
<td>977.8K</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>
The accuracy of the PICMETER is dependent on the range of components being measured. If auto-ranging could be implemented, the accuracy of the PICMETER could be improved. The addition of capacitors in parallel with $C_2$ of Figure 16 would allow auto-ranging for resistor measurements. Additional resistors in parallel with $R_3$ would give auto-ranging capability to capacitor measurements. Figure 18 shows a simple implementation of auto-ranging given that the I/O pins are available. The $R_?_C$ and $C_?_C$ are the extra components that are added to the PICMETER circuit. These components should be optimized for a particular range of devices.

**FIGURE 18: AUTO-RANGING TECHNIQUE**

Another addition to the PICMETER that would increase the accuracy of components being measured is a constant current source. The source would feed into the resistor of the RC networks. This provides the same charging current to all RC networks being measured. Figure 19 shows a bilateral current source and Figure 20 shows a precision current source.

**FIGURE 19: BILATERAL CURRENT SOURCE**

Another addition to the PICMETER that would increase the accuracy of components being measured is a constant current source. The source would feed into the resistor of the RC networks. This provides the same charging current to all RC networks being measured. Figure 19 shows a bilateral current source and Figure 20 shows a precision current source.

**FIGURE 20: PRECISION CURRENT SOURCE**

The alternative to the previous current sources is a single chip solution. A 3-terminal adjustable current source, such as a LM134/LM234/LM334 from National Semiconductor, is an ideal choice. This output current is programmable from 1 µA to 10 mA and requires a single external resistor to set the value of current. Figure 21 shows a block diagram of the LM334Z.

**FIGURE 21: LM334Z BLOCK DIAGRAM**

CONCLUSION

The PIC16C62X devices add two significant analog features to the PIC16CXX mid-range family: comparators and a voltage reference. The flexibility of eight operating modes for the comparator module allows the designer to tailor the PIC16C62X device to the application. The addition of an on-chip voltage reference simplifies the design by removing at least one external component and power consumption. These analog modules coupled with the PIC16CXX mid-range family core create a new path to achieve high resolution results.
APPENDIX A: PICMETER Firmware

MPASM 01.02.05 Intermediate PICMETER.ASM 5-1-1995 11:29:17 PAGE 1
PICMETER Firmware for PIC16C622

LOC OBJECT CODE  LINE SOURCE TEXT
VALUE

0001  TITLE "PICMETER Firmware for PIC16C622"
0002  LIST P = 16C622, F = INHX8M
0003  INCLUDE "C:\PICMASTR\P16CXX.INC"
0004 ; P16CXX.INC Standard Header File, Version 0.2 Microchip Technology, Inc.
0005
0006  FUSES _BODEN_OFF&_CP_OFF&_PWDT_ON&_WDT_OFF&_XT_OSC
0007
0008 ;************************************************************************
0009 ;*----------------------------------------------------------------------*
0010 ;*-
0011 ;*-
0012 ;*-
0013 ;*-
0014 ;*-
0015 ;*-
0016 ;*-
0017 ;*-
0018 ;*-
0019 ;*-
0020 ;*-
0021 ;*-
0022 ;*-
0023 ;*-
0024 ;*-
0025 ;*-
0026 ;*-
0027 ;*-
0028 ;*-
0029 ;*-
0030 ;*-
0031 ;*-
0032 ;*-
0033 ;*-
0034 ;*-
0035 ;*-
0036
0037
0038 ;************************************************************************
0039 ;*----------------------------------------------------------------------*
0040 ;************************************************************************
0041 ;************************************************************************
0042 ;************************************************************************
0043 ;************************************************************************
0044 ;************************************************************************
0045 xtal equ .4000000
0046 baud equ .9600
0047 fclk equ xtal/4
0048 ;************************************************************************
0049 ;The value baudconst must be a 8-bit value only
0050 baudconst equ ((fclk/baud)/3-2)
0051 ;************************************************************************
0052
0053

© 1995 Microchip Technology Inc.
0054 ;**********************************************************************
0055 ; Bit Equates
0056 ;**********************************************************************
0060 BEGIN equ 0 ;begin a measurement flag
0067 DONE equ 7 ;done measuring flag
0069 WHICH equ 5 ;R or C measurement flag
006F F_ERROR equ 3 ;error detection flag
006F EMPTY equ 5 ;flag if component is connected
006F V0 equ 0 ;power for R reference ckt
006F V1 equ 1 ;power for C reference ckt
006F V2 equ 2 ;ground for R reference ckt
006F V3 equ 3 ;power for unknown R ckt
006F V4 equ 4 ;ground for unknown C ckt
006F msh_bit equ 7 ;define for bit 7
006F lab_bit equ 0 ;define for bit 0
006F RkHI equ 0x07 ;value of the known resistance, R4, in ohms
006F RkMID equ 0x0D ;measured by a Fluke meter
006F RkLO equ 0x38 ;power for R reference ckt
006F CkHI equ 0x07 ;value of the known capacitance, C1, in pF
006F CkMID equ 0x30 ;measured by a Fluke meter
006F CkLO equ 0x20 ;define for bit 7
006F count equ 0 ;define for bit 0
006F txreg equ 0 ;RS232 register
006F delay equ 0 ;RS232 delay register
006F offset equ 0 ;table position register
006F msb equ 0 ;general delay register
006F lab equ 0 ;general delay register
006F W TEMP equ 0x20 ;Bank 0 temporary storage for W reg
006F STATUS_TEMP equ 0x21 ;temporary storage for STATUS reg
006F Ttemp equ 0x23 ;temporary Time register
006F flags equ 0x24 ;flags register
006F count equ 0x25 ;RS232 register
006F txreg equ 0x26 ;RS232 data register
006F delay equ 0x27 ;RS232 delay register
006F offset equ 0x28 ;table position register
006F msb equ 0x29 ;general delay register
006F lab equ 0x2A ;general delay register
006F TimeLO equ 0x40 ;Time registers
006F TimeMID equ 0x41 ;Time registers
006F TimeHI equ 0x42 ;Time registers
006F Math related registers
006F ACCaHI equ 0x50 ;24-Bit accumulator a
006F ACCaMID equ 0x51 ;24-Bit accumulator b
006F ACCaLO equ 0x52 ;24-Bit accumulator c
006F ACCbHI equ 0x53 ;24-Bit accumulator d
006F ACCbMID equ 0x54 ;24-Bit accumulator d
006F ACCbLO equ 0x55 ;24-Bit accumulator d
006F ACCcHI equ 0x56 ;24-Bit accumulator e
006F ACCcMID equ 0x57 ;24-Bit accumulator e
006F ACCcLO equ 0x58 ;24-Bit accumulator e
006F ACCdHI equ 0x59 ;24-Bit accumulator e
006F ACCdMID equ 0x5A ;24-Bit accumulator e
006F ACCdLO equ 0x5B ;24-Bit accumulator e
006F temp equ 0x5C ;temporary storage
006F User Registers Bank 1
006F W TEMP equ 0x00 ;Bank 1 temporary storage for W reg
006F tx PORTB,7 ;define for RS232 TXD output pin
006F User defines
006F org 0x0 ;define for RS232 TXD output pin
006F goto init

© 1995 Microchip Technology Inc.
0004 28B9
0120 org 0x4
0121 goto ServiceInterrupts
0122
0123 org 0x10
0124 init
0100 1283
0125 bcf STATUS,RPO ;select bank 0
0101 0185
0126 clrf PORTA ;clear PORTA and PORTB
0102 0186
0127 clrf PORTB
0103 1786
0128 bsf tx ;set TXD output pin
0104 01A4
0129 clrf flags ;clear flags register
0105 3010
0130 movlw 0x10 ;load table offset register
0106 00A8
0131 movwf offset
0107 0188
0132 clrf INTCON ;clear interrupt flags and disable interrupts
0108 0807
0133 movlw 0x07 ;turn off comparators, mode 111
0109 009F
0134 movwf CMCON
0110 2140
0135 call delay20 ;wait for comarators to settle
0111 089F
0136 movf CMCON,F
0112 133C
0137 bcf F1,F1,CMIF
0113 1683
0138 bsf STATUS,RPO ;select bank 1
0114 3088
0139 movlw 0x88 ;WDT prescalar, internal TMR0 increment
0115 0081
0140 movwf OPTION_REG
0116 0185
0141 clrf TRISA ;PORTA all outputs, discharges RC ckt
0117 3060
0142 movlw 0x60 ;PORTA<7,4:0> outputs, PORTA<6:5> inputs
0118 0086
0143 movwf TRISB
0119 300C
0144 movlw 0x0C ;setup Voltage Reference
0120 009F
0145 movwf VRCON
0121 1283
0146 bcf STATUS,RPO ;select bank 0
0122 3088
0147 movlw 0x88 ;enable RBIE interrupt
0123 008B
0148 movwf INTCON
0124 213D
0149 call vlong ;delay before transmitting boot message
0125 213D
0150 call vlong ;to allow computer program to setup
0126 213D
0151 call vlong
0127 2131
0152 call BootMSG ;transmit boot message
0128 178B
0153 bcf INTCON,GIE ;enable global interrupt bit
0129 014F
0154 start
012A 1C24
0155 start
012B 282D
0156 btfss flags,BEGIN ;wait for a start measurement key press
012C 102F
0157 goto start
012D 0186
0158 bcf flags,BEGIN ;clear start measurement flag
012E 018B
0159
012F 305B
0160 bcf INTCON,GIE ;transmit a start measurement message
0130 0061
0161 movlw 'S' ;to the PC
0131 20AD
0162 call Send
0132 178B
0163 bcf INTCON,GIE
0133 014F
0164
0134 01C2
0165 clrf TimeHI ;reset Time registers
0135 01C1
0166 clrf TimeMID
0136 01C0
0167 clrf TimeLO
0137 1856
0168 btfss PORTB,WHICH ;detect if resistor or capacitor measure
0138 2862
0169 goto Capacitor
0139 0170
0170
013A 1683
0171 bcf STATUS,RPO ;set V0 to input
013B 1406
0172 bcf TRISV,0
013C 1283
0173 bcf STATUS,RPO
013D 20FB
0174 call AnalogOn ;turn analog on
013E 0181
0175 clrf TMR0
013F 0000
0176 ncp
0140 1586
0177 bcf PORTB,V3 ;turn power on to unknown RC ckt
0141 19A4
0178 RwaitU btfsc flags,F_ERROR ;detect if an error occurs
0142 288B
0179 goto ErrorDetect
0143 1FA4
0180 btfsc flags,DONE ;measurement completed flag
0144 2840
0181 goto RwaitU
0145 13A4
0182 bcf flags,DONE ;clear measurement completed flag
0146 2111
0183 call AnalogOff ;turn analog off
0147 0185

© 1995 Microchip Technology Inc.
AN611
0046 2126 0186 call SwapTtoA ;swap Time to accumulator a
0047 3007 0187 movlw RkHI ;swap known resistance value
0048 00D3 0188 movwf ACCbHI ;to accumulator b
0049 309D 0189 movlw RkMID
004A 00D4 0190 movwf ACCbMID
004B 3038 0191 movlw RKLO
004C 0005 0192 movwf ACCbLO
004D 2230 0193 call Mpy24 ;multiply accumulator a and b
004E 1683 0194 bsf STATUS,RP0 ;set V3 to input
004F 1586 0195 bsf TRISB,V3
0050 1283 0196 bcf STATUS,RP0
0051 20F8 0197 call AnalogOn ;turn analog on
0052 0181 0198 clrf TMRO
0053 0000 0199 nop
0054 1406 0200 bcf PORTB,V0 ;turn power on to known RC ckt
0055 13A4 0201 RwaitK btfsb flags,F_ERROR ;detect if an error occurs
0056 288B 0202 goto ErrorDetect
0057 1FA4 0203 btfsb flags,DONE ;measurement completed flag
0058 2955 0204 goto RwaitK
0059 13A4 0205 bcf flags,DONE ;clear measurement completed flag
005A 2111 0206 call AnalogOff ;turn analog off
005B 2126 0207 call SwapTtoA ;swap Time to accumulator a
005C 224B 0208 call Div24 ;divide multiply by known time
005D 139B 0209 call
005E 3052 0210 bcf INTCON,GIE ;disable all interrupts
005F 19A4 0211 movlw 'R' ;transmit, for R measurement
0060 1783 0212 call Send
0061 282D 0213 bcf INTCON,GIE ;enable global interrupt bit
0062 0181 0214 goto start ;restart
0063 221D 0215 call
0064 0000 0216 bcf
0065 1406 0217 Capacitor
0066 1683 0218 bcf STATUS,RP0 ;set V2 to input
0067 1506 0219 bcf TRISB,V2
0068 1283 0220 bcf STATUS,RP0
0069 20FB 0221 call AnalogOn ;turn analog on
006A 0181 0222 clrf TMRO
006B 0000 0223 nop
006C 1486 0224 bcf PORTB,V1 ;turn power on to unknown RC ckt
006D 13A4 0225 bcf flags,DONE ;clear measurement completed flag
006E 2111 0226 call AnalogOff ;turn analog off
006F 2126 0227 call SwapTtoA ;swap Time to accumulator a
0070 3007 0228 movlw CKHI ;swap known resistance value
0071 00D3 0229 movwf ACCbHI ;to accumulator b
0072 30C8 0230 movlw CKMID
0073 00D4 0231 movwf ACCbMID
0074 3030 0232 movlw CKLO
0075 0005 0233 movwf ACCbLO
0076 2230 0234 call Mpy24 ;multiply accumulator a and b
0077 1683 0235 bcf STATUS,RP0 ;set V3 to input
0078 1606 0236 bcf TRISB,V4
0079 1283 0237 bcf STATUS,RP0
007A 20F8 0238 call AnalogOn ;turn analog on
007B 0181 0239 clrf TMRO
007C 0000 0240 nop
007D 1486 0241 bcf PORTB,V1 ;turn power on to known RC ckt
007E 19A4 0242 BwaitK btfsb flags,F_ERROR ;detect if an error occurs
007F 288B 0243 goto ErrorDetect
0080 1FA4 0251 btfss flags, DONE ;measurement completed flag
0081 127E 0252 goto CwaitK
0082 1FA4 0253 bcf flags, DONE ;clear measurement completed flag
0083 2111 0254 call AnalogOff ;turn analog off
0084 2126 0255
0085 224B 0256 call SwapToA ;swap Time to accumulator a
0086 138B 0257 call Div24 ;divide multiply by known time
0087 200A 0258
0088 138B 0259 bcf INTCON, GIE ;disable all interrupts
0089 138B 0260 movlw 'C' ;transmit, for C measurement
008A 138B 0261 call Send
008B 178B 0262 bcf INTCON, GIE ;enable global interrupt bit
008C 282D 0263 goto start ;restart
008D 265 0264 ErrorDetect
008E 1283 0265 bcf STATUS, Rp0 ;disable TMR0
008F 128B 0266 bcf INTCON, TOIE
0090 110B 0267 bcf INTCON, TOIF
0091 2111 0268 call AnalogOff ;turn analog off
0092 138B 0269
0093 138B 0270 bcf INTCON, GIE ;disable all interrupts
0094 138B 0271
0095 282D 0272 movlw 'E' ;transmit, for C measurement
0096 206D 0273 call Send
0097 178B 0274
call Send
0098 178B 0275 bcf INTCON, GIE ;enable global interrupt bit
0099 282D 0276 goto start ;restart
0277 ;******************************************************************
0278 ;*:-----------------------------------------------*----------------
0279 ;*: RS232 Transmit Routine
0280 ;*:--------------------------------------------------*----------------
0281 ;*: Borrowed from AN593, "Serial Port Routines Without Using the RXC"
0282 ;*: Author: Stan D'Souza
0283 ;*: This is the routine that interfaces directly to the hardware
0284 ;*:--------------------------------------------------*----------------
0285 ;******************************************************************
0309 1283 0286 Transmit
0095 1283 0287 bcf STATUS, Rp0
0096 00A6 0288 movwf txreg
0097 1386 0289 bcf tx ;send start bit
0098 3020 0290 movlw baudconst
0099 00A7 0291 movwf delay
009A 3009 0292 movlw 0x9
009B 00A5 0293 movwf count
009C 0294 txbaudwait
009C 0BA7 0295 decfsz delay
009D 289C 0296 goto txbaudwait
009B 3020 0297 movlw baudconst
009F 00A7 0298 movwf delay
00A0 0BA5 0299 decfsz count
00A1 28A6 0300 goto SendNextBit
00A2 3009 0301 movlw 0x9
00A3 00A5 0302 movwf count
00A4 1786 0303 bcf tx ;send stop bit
00A5 0008 0304 return
00A6 0305 SendNextBit
00A6 0CA6 0306 rrf txreg
00A7 1C03 0307 btfss STATUS, C
00A8 28A8 0308 goto Setlo
00A9 1786 0309 bcf tx
00AA 289C 0310 goto txbaudwait
00AB 1386 0311 Setlo bcf tx
00AC 289C 0312 goto txbaudwait
0313 ;******************************************************************
0314 ;******************************************************************
0315 ;******************************************************************
0316 ;*---------------------------------------------------------------*
Generic Transmit Routine

Sends what is currently in the W register and accumulator ACCc

Send call Transmit ;delay between bytes
movf ACCcHI,W ;transmit high resistance byte
call Transmit ;delay between bytes
movf ACCcMID,W ;transmit mid resistance byte
call Transmit ;delay between bytes
movf ACCcLO,W ;transmit low resistance byte
call Transmit ;delay between bytes

return

Interrupt Service Routines

servicelnterrupts

movw W_TEMP ;Pseudo push instructions
swapf STATUS,W
bcf STATUS,RPO
movwf STATUS_TEMP

TMRO,W ;Pseudo pop instructions
Ttemp

servicetimer

btsc INCON,T0IF ;Service Timer 0 overflow
bcf INTCON,R0IE

servicemparator

btsc PIR1,CMIF ;Stops Timer0, Records Value
bcf STATUS,RPO

servicekeystroke ;Starts a measurement

btsc INCON,R1IF ;Service pushbutton switch
bsf STATUS,Z

servicetimerelease

call ServiceKeystroke ;Starts a measurement
notswitch

de-bounce switch for 16msec
check for key release

return
0383 0384 NotSwitch ;detected other PORTB pin change
0385 0386 bcf INTCON,RBIF ;reset RBI interrupt
0387 0388 return
0389 0390 KeyRelease
0391 0392 call delay16 ;debounce switch
0393 0394 comf PORTB,W ;read PORTB
0395 0396 bcf INTCON,RBIF ;clear flag
0397 0398 bsf INTCON,RIE ;enable interrupt
0399 0400 andlw B'01000000' 
0401 0402 if KEYRB.DONE
0403 0404 ;******************************************************************
0405 ;*----------------------------------------------------------------* 
0406 ServiceTimer
0407 ;******************************************************************
0408 0409 incf TimeMID,F;increment middle Time byte
0410 0411 btfsc STATUS,Z ;if middle overflows,
0412 0413 incf TimeHI,F ;increment high Time byte
0414 0415 bcf INTCON,TOIF ;clear flag
0416 0417 return
0418 ;******************************************************************
0419 0420 ServiceComparator
0421 ;******************************************************************
0422 0423 incf STATUS,RP0 ;select bank 0
0424 0425 bcf PORTB,WHICH ;detect which measurement, R or C?
0426 0427 goto capcomp
0428 0429 goto scstop
0430 0431 capcomp
0432 0433 btfsc CMCON,C2OUT ;detect if C ckt has interrupted
0434 0435 goto scend
0436 0437 bcf INTCON,T0IE ;disable TMR0 interrupts
0438 0439 movf Ttemp,W
0440 0441 movwf TimeLO
0442 0443 scend
0444 0445 bcf PIR1,CMIF ;clear comparator interrupt flag
0446 0447 0448 Turn Comparators and Vref On
0449 ;******************************************************************
0449 ;******************************************************************************
0450 AnalogOn
00FB 1283 0451 bcf STATUS,RPO ;select bank 0
00FC 3002 0452 movlw 0x02 ;turn comparators on, mode 010
00FD 009F 0453 movwf CMCON ;4 inputs multiplexed to 2 comparators
00FE 1683 0454 bcf STATUS,RPO ;select bank 1
00FF 300F 0455 movlw 0x0F ;make PORTA<3:0> all inputs
0100 0085 0456 movwf TRISA
0101 179F 0457 bcf VRCON,VREN
0102 1283 0458 bcf STATUS,RPO ;select bank 0
0103 2140 0459 call delay20 ;20msec delay
0104 009F 0460 movf CMCON,F ;clear comparator mismatch condition
0105 130C 0461 bcf PI1,CMIF ;clear comparator interrupt flag
0106 1683 0462 bcf STATUS,RPO
0107 170C 0463 bcf PI1,CMIE ;enable comparator interrupts
0108 1283 0464 bcf STATUS,RPO
0109 170B 0465 bcf INTCN,PEIE ;enable peripheral interrupts
010A 11A4 0466 bcf flags,F_ERROR
010B 0181 0467 clrf TMRO ;clear TMRO counter
010C 0000 0468 nop
010D 0000 0469 nop
010E 110B 0470 bcf INTCN,TOIF ;clear TMRO interrupt flag
010F 168B 0471 bcf INTCN,TOIE ;enable TMRO interrupts
0110 0008 0472 return
0111 0480 AnalogOff
0111 1283 0481 bcf STATUS,RPO
0112 130B 0482 bcf INTCN,PEIE
0113 3080 0483 movlw 0x80 ;select bank 1
0114 0086 0484 movwf PORTB
0115 1683 0485 bcf STATUS,RPO ;select bank 1
0116 130C 0486 bcf PI1,CMIE ;disable comparator interrupts
0117 0185 0487 clrf TRISA ;set PORTA pins to outputs, discharge RC ckt
0118 3060 0488 movlw 0x60 ;set PORTB 7,4-0 as outputs, 6,5 as inputs
0119 0086 0489 movwf TRISB
011A 139F 0490 bcf VRCON,VREN ;disable Vref
011B 1283 0491 bcf STATUS,RPO ;select bank 0
011C 3007 0492 movlw 0x07
011D 009F 0493 movwf CMCON ;disable comparators
011E 2140 0494 call delay20 ;20msec delay
011F 009F 0495 movf CMCON,F ;clear comparator mismatch condition
0120 130C 0496 bcf PI1,CMIF ;clear comparator interrupt flag
0121 110B 0497 bcf INTCN,TOIF ;clear Timer0 interrupt flag
0122 213D 0498 call vlong ;long delay to allow capacitors to discharge
0123 213D 0499 call vlong
0124 213D 0500 call vlong
0125 0008 0501 return
0126 0509 SwapTtoA
0126 1233 0510 bcf STATUS,RPO
0127 0842 0511 movf TimeHI,W
0128 00D0 0512 movwf ACCaHI
0129 0841 0513 movf TimeMID,W
012A 00D1 0514 movwf ACCaMID

DS00611A-page 2-92 © 1995 Microchip Technology Inc.
012B 0840 0515 movf TimeLO,W
012C 00D2 0516 movwf ACCaLO
012D 01C2 0517 clrf TimeHI
012E 01C1 0518 clrf TimeMID
012F 01C0 0519 clrf TimeLO
0130 0008 0520 return
0521 ;
0522 ;******************************************************************
0523 ;*----------------------------------------------------------------*
0524 ;*- Transmit the Boot Message
0525 ;*----------------------------------------------------------------*
0526 ;******************************************************************
0131 0528 BootMSG
0131 1283 0529 bcf STATUS,RP0 ;select bank 0
0132 3002 0530 msg movlw HIGH Table ;init the PCH for a table call
0133 008A 0531 movwf PCLATH
0134 0828 0532 movf offset,W ;move table offset into W
0135 2200 0533 call Table ;get table value
0136 2085 0534 call Transmit ;transmit table value
0137 2146 0535 call delay1 ;delay between bytes
0138 0BA8 0536 decfsz offset,F ;check for end of table
0139 2932 0537 goto msg
013A 3010 0538 movlw 0x10 ;reset table offset
013B 00A8 0539 movwf offset
013C 0008 0540 return
0541 ;
0542 ;******************************************************************
0543 ;*----------------------------------------------------------------*
0544 ;*- Delay Routines
0545 ;*----------------------------------------------------------------*
0546 ;******************************************************************
0547 ;******************************************************************
013D 30FF 0548 vlong movlw Oxff ;very long delay, approx 200msec
013E 00A9 0549 movwf msb
013F 2948 0550 goto d1
0140 0551 delay20 ;20 msec delay
0140 301A 0552 movlw .26
0141 00A9 0553 movwf msb
0142 2948 0554 goto d1
0143 0555 delay16 ;16 msec delay
0143 3015 0556 movlw .21
0144 00A9 0557 movwf msb
0145 2948 0558 goto d1
0146 0559 delay1 ;approx 750nsec delay
0146 3001 0560 movlw .1
0147 00A9 0561 movwf msb
0148 30FF 0562 d1 movlw 0xff
0149 00AA 0563 movwf lsb
014A 0BA8 0564 d2 decfsz lsb,F
014B 294A 0565 goto d2
014C 0BA9 0566 decfsz msb,F
014D 2948 0567 goto d1
014E 0008 0568 return
0569 ;
0570 0571 0572 org 0x200
0573 0574 0575 ;******************************************************************
0576 ;*----------------------------------------------------------------*
0577 ;*- Table for Boot Message
0578 ;*----------------------------------------------------------------*
0579 ;******************************************************************
0120 01C2
0121 0580 Table ;boot message "PICMETER Booted!"
0200 0782 0581 addwf PCL ;add W to PCL
0201 3400 0582 retlw 0
0202 3421 0583 retlw '!' 
0203 3464 0584 retlw 'd' 
0204 3465 0585 retlw 'e' 
0205 3474 0586 retlw 't' 
0206 346F 0587 retlw 'o' 
0207 346F 0588 retlw 'o' 
0208 3442 0589 retlw 'B' 
0209 3420 0590 retlw ' ' 
020A 3472 0591 retlw 'r' 
020B 3465 0592 retlw 'e' 
020C 3474 0593 retlw 't' 
020D 3465 0594 retlw 'e' 
020E 346D 0595 retlw 'm' 
020F 3443 0596 retlw 'C' 
0210 3449 0597 retlw 'I' 
0211 3450 0598 retlw 'P'

0599 ;
0600
0601 ;************************************************************************************
0602 ;*-----------------------------------------------------------------*
0603 ;*- 24-bit Addition
0604 ;*- Uses ACCa and ACCb
0605 ;*- ACCa + ACCb -> ACCb
0606 ;*------------------------------------------------------------------
0607 ;************************************************************************************
0608 ;************************** 24-bit Addition **************************************
0609 ;********************************************************************************* 24-bit Addition

0212 0610 Add24
0212 0852 0611 movf ACCaLo,W
0213 07D5 0612 addwf ACCbLO ;add low bytes
0214 1803 0613 btfsc STATUS,C ;add in carry if necessary
0215 2A1D 0614 goto A2
0216 0851 0615 A1 movf ACCaMid,W
0217 07D4 0616 addwf ACCbMid ;add mid bytes
0218 1803 0617 btfsc STATUS,C ;add in carry if necessary
0219 0AD3 0618 incf ACCbHI
021A 0850 0619 movf ACCbHI,W
021B 07D3 0620 addwf ACCbLO ;add high bytes
021C 3400 0621 retlw 0
021D 0AD4 0622 A2 incf ACCbMID
021E 1903 0623 btfsc STATUS,Z
021F 0AD3 0624 incf ACCbHI
0220 2A16 0625 goto A1
0626 ;
0627
0628 ;********************************************************************************* 24-bit Addition
0629 ;********************************************************************************* 24-bit Addition

0630 ;*-- Subtraction ( 24 - 24 -> 24 )
0631 ;*--
0632 ;*-- Uses ACCa, ACCb, ACCd
0633 ;*--
0634 ;*-- ACCa -> ACCd,
0635 ;*-- 2's complement ACCa,
0636 ;*-- call Add24 ( ACCa + ACCb -> ACCb ),
0637 ;*-- ACCd -> ACCa
0638 ;*------------------------------------------------------------------
0639 ;********************************************************************************* 24-bit Addition

0221 0640 Sub24
0221 0850 0641 movf ACCaHI,W ;Transfer ACCa to ACCd
0222 0009 0642 movwf ACCbHI
0223 0851 0643 movf ACCaMID,W
0224 00DA 0644 movwf ACCbMID
0225 0852 0645 movf ACCaLO,W
0226 00DB 0646 movwf ACCbLO

DS00611A-page 2-94 © 1995 Microchip Technology Inc.
0227 2275 0647  call  compA  ;2's complement ACCa
0228 2212 0648  call  Add24  ;Add ACCa to ACCb
0229 0859 0649  movf  ACCcHI,W  ;Transfer ACCd to ACCa
022A 00D0 0650  movwf  ACCaHI
022B 085A 0651  movf  ACCcMID,W
022C 00D1 0652  movwf  ACCaMID
022D 085B 0653  movf  ACCcLO,W
022E 00D2 0654  movwf  ACCaLO
022F 3400 0655  retlw  0

0656  ;
0657  0658  ;******************************************************************************
0659  ;*---------------------------------------------------------------------------*
0660  ;* Multiply ( 24 X 24 -> 56 )                                           *
0661  ;*---------------------------------------------------------------------------*
0662  ;* Uses ACCa, ACCb, ACCc, ACCd                                           *
0663  ;*---------------------------------------------------------------------------*
0664  ;* ACCa * ACCb -> ACCc,ACCb 56-bit output                                *
0665  ;* with ACCb (ACCcHI,ACCcMID,ACCcLO) with 24 msb's and                   *
0666  ;* ACCc (ACCcHI,ACCcMID,ACCcLO) with 24 lsb's                            *
0667  ;*---------------------------------------------------------------------------*
0668  ;******************************************************************************
0669  0230 0669  Mpy24
0230 223F 0670  call  Msetup
0231 0CD9 0671  mloop  rrf  ACCcHI  ;rotate d right
0232 0CDA 0672  rrf  ACCcMID
0233 0CD8 0673  rrf  ACCcLO
0234 1803 0674  btfsc  STATUS,C  ;need to add?
0235 2212 0675  call  Add24
0236 0CD3 0676  rrf  ACCbHI
0237 0CD4 0677  rrf  ACCbMID
0238 0CD5 0678  rrf  ACCbLO
0239 0CD6 0679  rrf  ACCcHI
023A 0CD7 0680  rrf  ACCcMID
023B 0CD8 0681  rrf  ACCcLO
023C 0BDC 0682  decfsz  temp  ;loop until all bits checked
023D 2A31 0683  goto  mloop
023E 3400 0684  retlw  0
023F 0685
023F 3018 0686  movlw  0x18  ;for 24 bit shifts
0240 00DC 0687  movwf  temp
0241 0853 0688  movf  ACCcHI,W  ;move ACCb to ACCd
0242 00D9 0689  movwf  ACCcHI
0243 0854 0690  movf  ACCcMID,W
0244 00DA 0691  movwf  ACCcMID
0245 0855 0692  movf  ACCcLO,W
0246 00DB 0693  movwf  ACCcLO
0247 01D3 0694  clrf  ACCbHI
0248 01D4 0695  clrf  ACCbMID
0249 01D5 0696  clrf  ACCbLO
024A 3400 0697  clrf  ACCcLO
024F 0698  retlw  0

0699  ;
0700  0701  ;******************************************************************************
0702  ;*---------------------------------------------------------------------------*
0703  ;* Division ( 56 / 24 -> 24 )                                           *
0704  ;*---------------------------------------------------------------------------*
0705  ;* Uses ACCa, ACCb, ACCc, ACCd                                           *
0706  ;*---------------------------------------------------------------------------*
0707  ;* 56-bit dividend in ACCb,ACCc ( ACCb has msb's and ACCc has lsb's)        *
0708  ;* 24-bit divisor in ACCa                                                  *
0709  ;* quotient is stored in ACCc                                             *
0710  ;* remainder is stored in ACCb                                           *
0711  ;*---------------------------------------------------------------------------*
0712  ;******************************************************************************

© 1995 Microchip Technology Inc. DS00611A-page 2-95
024B 0713 Div24
024B 2272 0714 call Dsetup
0715
024C 1003 0716 dloop bcf STATUS,C
024D 0DD8 0717 rlf ACCcLO ;Rotate dividend left 1 bit position
024E 0DD7 0718 rlf ACCcMID
024F 0DD6 0719 rlf ACCcHI
0250 0DD5 0720 rlf ACCbLO
0251 0DD4 0721 rlf ACCbMID
0252 0DD3 0722 rlf ACCbHI
0253 1803 0723
0254 2A58 0724 btfsc STATUS,C ;invert carry and exclusive or with the
0255 1FD0 0725 goto clear ;msb of the divisor then move this bit
0256 0A08 0726 btfsc ACCcHI,msb_bit ;into the lsb of the dividend
0257 2A05 0727 incf ACCcLO
0258 1BD0 0728 goto cont
0259 0A08 0729 clear btfsc ACCcHI,msb_bit
025A 0A08 0730 incf ACCcLO
025B 1858 0731
025C 2212 0732 cont btfsc ACCcLO,lsb_bit ;check the lsb of the dividend
025D 2A5E 0733 goto minus
025E 2212 0734 call Add24 ;if = 0, then add divisor to upper 24 bits
025F 2212 0735 goto check ;of dividend
0260 2212 0736 minus call Sub24 ;if = 1, then subtract divisor from upper
0261 0BDC 0737 ;24 bits of dividend
0262 2A4C 0738
0263 1003 0739 check decfsz temp,f ;do 24 times
0264 0DD8 0740 goto dloop
0265 0DD7 0741
0266 0DD6 0742 bcf STATUS,C ;shift lower 24 bits of dividend 1 bit
0267 0DD5 0743 rlf ACCcLO ;position left
0268 0DD4 0744 rlf ACCcMID
0269 0DD3 0745 rlf ACCcHI
026A 1BD3 0746 btfsc ACCbHI,msb_bit ;exclusive or the inverse of the msb of the
026B 2A6A 0747 goto w1 ;dividend with the msb of the divisor
026C 1FD0 0748 btfsc ACCcHI,msb_bit ;store in the lsb of the dividend
026D 0A08 0749 incf ACCcLO
026E 2A6C 0750 goto wzd
026F 0A08 0751 w1 btfsc ACCcHI,msb_bit
0270 1BD0 0752 incf ACCcLO
0271 1BD0 0753 wzd btfsc ACCbHI,msb_bit ;if the msb of the remainder is set and
0272 2A71 0754 goto wend
0273 1BD0 0755 btfsc ACCcHI,msb_bit ;the msb of the divisor is not
0274 2A71 0756 goto wend
0275 2212 0757 call Add24 ;add the divisor to the remainder to correct
0276 0BDC 0758 ;for zero partial remainder
0277 3400 0759
0278 3400 0760 wend retlw 0 ;quotient in 24 lsb's of dividend
0761 ;remainder in 24 msb's of dividend
0762
0279 3018 0763 Dsetup
027A 00DC 0764 movlw 0x18 ;loop 24 times
027B 3400 0765 movwf temp
0766
027C 3400 0767 retlw 0
0768 ;
0769
0770 ;******************************************************************
0771 ;*----------------------------------------------------------------
0772 ;*----------------------------------------------------------------
0773 ;*----------------------------------------------------------------
0774 ; 2's Complement
0775 ;Uses ACCa
0776 ;Performs 2's complement conversion on ACCa
0777 ;-------------------------------------------------------------------
0778 ;******************************************************************
0275 0779 compA
0275 09D2 0780 comp ACCaLO ;invert all bits in accumulator a
0276 09D1 0781 comp ACCaMID
0277 09D0 0782 comp ACCaHI
0278 0AD2 0783 incf ACCaLO ;add one to accumulator a
0279 1903 0784 btfsc STATUS,Z
027A 0AD1 0785 incf ACCaMID
027B 1903 0786 btfsc STATUS,Z
027C 0AD0 0787 incf ACCaHI
027D 3400 0788 retlw 0
0789 ;______________________________
0790 0791 END
0792

0000: X---X------------- XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
0040: XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
0080: XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
00C0: XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
0100: XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
0140: XXXXXXXXXXXXXXX-- ----------------- ----------------- ----------------- -----------------
0200: XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
0240: XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX--

All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
APPENDIX B: VISUAL BASIC PROGRAM

PICMTR.FRM

Sub Form_Load()
    'Initialize the program
    Image1.Height = 600
    Image1.Width = 2700
    Framel.Caption = "PICMETER Power Off"
    Label1.Caption = ""
    Label2.Caption = ""

    'Initialize Comm Port
    Comml.RThreshold = 1
    Comml.Handshaking = 0
    Comml.Settings = "9600,n,8,1"
    Comml.CommPort = 2
    Comml.PortOpen = True

    'Initialize the global variable First%
    First% = 0
End Sub

Sub Form_Unload(Cancel As Integer)
    'Unload PICMETER
    Comml.RTSEnable = False
    Comml.DTREnable = False
    Comml.PortOpen = False
    Unload PICMETER
End Sub

Sub Comml_OnComm()
    Dim Value As Double
    Dim High As Double
    Dim Medium As Double
    Dim Low As Double

    'Received a character
    If Comml.CommEvent = 2 Then
        If First% = 0 Then
            If Comml.InBufferCount = 16 Then
                InString$ = Comml.Input
                If InString$ = "PICMETER Booted!" Then
                    Framel.Caption = "PICMETER Booted!"
                End If
            End If
        Else
            If Comml.InBufferCount >= 4 Then
                InString$ = Comml.Input
                If Left$(InString$, 1) = "R" Then
                    Framel.Caption = "Resistance"
                    Label2.FontName = "Symbol"
                    Label2.Caption = 0 "KW"
                    Labell.FontSize = 24
                ElseIf Left$(InString$, 1) = "C" Then
                    Framel.Caption = "Capacitance"
                    Label2.FontName = "MS Sans Serif"
                    Label2.Caption = "nF"
                    Labell.FontSize = 24
                ElseIf Left$(InString$, 1) = "E" Then
                    Framel.Caption = "Error Detected"
                    Label2.Caption = ""
                ElseIf Left$(InString$, 1) = "S" Then
                    Framel.Caption = "Measuring Component"
                End If
            Else
                If Comml.InBufferCount < 4 Then
                    InString$ = Comml.Input
                    If Left$(InString$, 1) = "R" Then
                        Framel.Caption = "Resistance"
                        Label2.FontName = "Symbol"
                        Label2.Caption = 0 "KW"
                        Labell.FontSize = 24
                    ElseIf Left$(InString$, 1) = "C" Then
                        Framel.Caption = "Capacitance"
                        Label2.FontName = "MS Sans Serif"
                        Label2.Caption = "nF"
                        Labell.FontSize = 24
                    ElseIf Left$(InString$, 1) = "E" Then
                        Framel.Caption = "Error Detected"
                        Label2.Caption = ""
                    ElseIf Left$(InString$, 1) = "S" Then
                        Framel.Caption = "Measuring Component"
                    End If
                End If
            End If
        End If
    End If
End Sub
Label2.Caption = ""
Else
    Framel.Caption = "Error Detected"
    Label2.Caption = ""
End If

If Framel.Caption = "Error Detected" Then
    Label1.Caption = ""
ElseIf Framel.Caption = "Measuring Component" Then
    Label1.Caption = ""
Else
    High = 65536# * Asc(Mid$(InString$, 2, 1))
    Medium = 256# * Asc(Mid$(InString$, 3, 1))
    Low = Asc(Mid$(InString$, 4, 1))
    Label1.Caption = Format$(High + Medium + Low) / 1000, "###0.0"
End If
End Sub

Sub Check3DL_Click (Value As Integer)
'Control Power to the PICMETER
If Check3DL.Value = False Then
    Comml.InputLen = 0
    Label1.Caption = ""
    Label2.Caption = ""
    Comml.RTSEnable = False
    Comml.DTREnable = False
    Framel.Caption = "PICMETER Power Off"
    InString$ = Comml.Input
Else
    Framel.Caption = ""
    First% = 0
    Comml.InputLen = 0
    InString$ = Comml.Input
    Comml.RTSEnable = True
    Comml.DTREnable = True
End If
End Sub

Sub menExitTop_Click ()
'Unload PICMETER
Unload PICMETER
End Sub

Sub Option1_Click ()
'Open COM1 for communications
If Option1.Value = True Then
    If Comml.CommPort = 2 Then
        Comml.PortOpen = False
        Comml.CommPort = 1
        Comml.PortOpen = True
    End If
End If
End Sub

Sub Option2_Click ()
'Open COM2 for communications
If Option2.Value = True Then
    If Comml.CommPort = 1 Then
        Comml.PortOpen = False
        Comml.CommPort = 2
        Comml.PortOpen = True
    End If
End If
End Sub
APPENDIX C: PICMETER PCB LAYOUT

Boards Manufactured by: Southwest Circuits
Contact: Perry Groves
3760 E. 43rd Place
Tucson, AZ 85713
1-520-745-8515

The following artwork is not printed to scale:

Component Side

Solder Side

© 1995 Microchip Technology Inc.
INTRODUCTION

Typical embedded control applications place demands such as low power consumption, small size, low cost and reduced component count onto the microcontroller. This application implements a 24-hour digital clock, alarm and 99 minute 59 second count down timer, yet operates on two "AA" batteries. The PIC16C54A is perfect for this application, due to it's small size, high current I/Os with direct LED drive, low cost, fast instruction throughput and low frequency/current operation.

System cost

The objective of this design was to implement the maximum number of features with the least expensive and smallest device. The PIC16C54A is Microchip's lowest cost microcontroller and it has 12 I/O lines, each capable of sinking 25 mA and sourcing 20 mA. High efficiency common cathode LED displays were chosen for their 3.5 mA current requirement, eliminating the need for any external transistors for display drive. A low impedance direct drive piezo buzzer was chosen and it's tone is generated by the software of the PIC16C54A to further reduce system cost.

Operating power

In battery powered applications, the operating current determines the lifetime of the batteries. There are many ways to reduce the operating current of any application, including low frequency operation and the use of sleep modes. Since the clock has to keep track of time, SLEEP mode could not be used and the processor must be kept running all of the time. The PIC16C54A supports the 32.768 kHz "watch" crystal and typically consumes less than 15 µA of current in this configuration. Since the PIC16C54A executes instructions in one cycle and it's instruction set is very efficient, this application was able to be implemented using a low frequency crystal. Another solution to this problem comes with the PIC16C74/73/65/63 in it's Timer1 module. This timer will run when the device is asleep, so it could have been used to keep track of time, simplifying the software.

Clock system

A 32.768 kHz crystal was chosen for the clock due to the low power and cost requirements of this design. The four internal phases of this input clock create an internal instruction cycle. Therefore, the instruction time is calculated as follows:

\[
\text{Instruction rate} = \frac{1}{(\text{CLKIN}/4)} = \frac{1}{32,768/4} = \frac{1}{8,192}
\]

Instruction cycle = 122.07 µs

This means that every instruction executes in 122.07 µs or we execute exactly 8,192 instructions per second.

Display and keypad multiplexing

The display contains four digits with seven segments each; therefore a multiplexing scheme was used to reduce the number of I/O lines needed to drive the displays. There are 4 common cathode display connections (one for each display digit and connected to PORTA for convenience so that rotates and moves can be used) and 7 segments (connected to PORTB for convenience so that moves can be used) for a total of 11 I/O lines needed for the display. Common Cathode displays were chosen, since the PIC16C54A can sink 5 mA more current than it can source. The last I/O line (RB7) was used to drive the buzzer. The three keys for setting the time are multiplexed onto the LED display segments to eliminate the need for additional I/O lines.
SOFTWARE IMPLEMENTATION

The main loop of the software must perform the following tasks to implement the clock’s functionality:

1. Determine when one second has passed (when bit 7 on the TMR0 register changes state 4 times), increment the current time and (if enabled) decrement the countdown timer.
2. Determine if any of the alarms (countdown timer or the alarm time) are currently alarming or should be alarming. If so, the buzzer is buzzed.
3. Check for any keys that are pressed. If MODE is pressed, the current mode is incremented and if UP or DOWN is pressed, the time displayed is modified.
4. Automatically turn the displays on/off for power management.
5. Multiplex the displays every 3.9 ms (32 instruction cycles).

General purpose registers are defined and used for the following purposes:

- **DISPSEGS_A** through **DISPSEGS_D** store the bit pattern that is to be displayed on each of the four 7-segment displays.
- **CLK_SEC** stores the second counter for the current clock time (values from 0-59 decimal are stored).
- **CLK_MIN_LD, CLK_MIN_HD** store the upper and lower minute digit of the current time.
- **CLK_HOUR_LD, CLK_HOUR_HD** store the upper and lower hour digit of the current time.
- **ALM_MIN_LD, ALM_MIN_HD** store the upper and lower minute digit of the alarm time.
- **ALM_HOUR_LD, ALM_HOUR_HD** store the upper and lower hour digit of the alarm time.
- **TMR_SEC_LD, TMR_SEC_HD** store the upper and lower second digit of the countdown timer.
- **TMR_MIN_LD, TMR_MIN_HD** store the upper and lower minute digit of the countdown timer.
- **KEYPAT** stores a pattern showing the currently pressed keys:
  - **UP = bit 6**
  - **DOWN = bit 5**
  - **MODE = bit 4**
- **FLAGS** stores key flag bits such as the current mode, display on, alarm on, etc.
- **PREVTMR0** stores previous TMR0 values so that the differences can be detected the next time the TMR0 is polled.
- **TEMP** is a temporary register used for various routines.
- **DISPONCNT** stores the remaining number of seconds the displays should be on.
- **MODE_COUNT** stores the number of 1/2 seconds the MODE and UP or DOWN buttons are pressed. Used to switch from setting minutes to hours.
- **ALARMCNT** stores the number of beeps remaining to be driven into the buzzer.

FLAGS Register

Most designs require flag or state bits to indicate current modes or the state of a software routines. In this design, the FLAGS register is defined as follows:

- **Bits 0,1** - indicates the current operating mode (changed by pressing and releasing the MODE button):
  - **00** - Display OFF
  - **01** - Display/Set countdown timer
  - **10** - Display/Set alarm time
  - **11** - Display/Set clock (current time)
- **Bit 3** - indicates if (alarm time) = (current clock time)
- **Bit 4** - indicates if (count down timer) = 0
- **Bits 5,6,7** - Used as a divide by four counter to keep track of seconds

The software is broken up into the following routines for modularity:

**buzz_now routine** - Output buzzing tone during alarm for 156 ms.

Buzzers are available in self-oscillating and direct drive models. To save cost, a low impedance, direct drive model was selected. The buzz_now routine is called by the main_loop and it chirps the buzzer for 156 ms at a 1638 Hz frequency. This routine first turns off the LEDs (by clearing PORTB) and then uses TEMP to count for 256 pulses. The pulse is sent to the buzzer by the **BSF BUZZEROUT** and **BCF BUZZEROUT** instructions. This routine returns once 256 pulses are sent to the buzzer. This is necessary, since the controller cannot buzz the buzzer and keep track of time at the same time (running at such a low frequency), so these two functions are multiplexed.
task_scan routine - multiplex LEDs to display the next digit, only one digit is lit at a time.

The PIC16C5X family is designed for polled I/O applications and does not contain a hardware interrupt structure. To achieve the lowest cost design, the PIC16C54A was selected and all modules are written to CALL this task_scan routine within the multiplexing time frame of 3.9 ms or 32 instruction cycles. This routine first synchronizes itself with the TMRO register, bit0 to ensure that the scanning occurs at the same point in time, regardless of when the routine is called. Next, PREVSCAN is rotated, setting up the CARRY bit correctly. The bit pattern for the next digit to be displayed is then moved into the W. register The display is blanked, PORTA is rotated to (select the next digit) and the next display bit pattern is moved to PORTB to display it. For ESD integrity, PORTA is later restored from the PREVSCAN register. This routine takes a total of 21 cycles (including the CALL and RETLW instructions) to execute and the displays are scanned every 3.9 ms (32 instruction cycles); therefore, this routine needs to be called after every 11 instruction cycles from every routine to maintain proper display multiplexing.

disp_value routine - Update the display registers with the bitmap of what digits are to be displayed next.

Indirect addressing is used here to reduce the amount of code needed and to simplify the routine. Since the clock, alarm or countdown timer could each be displayed, the W register contains the base address (in the register file) of the four digits that are to currently be displayed. The W register is first moved to the FSR register so that the indirect address register contains the first digit to be displayed. The first digit is first converted into the segment bit pattern by calling the led_lookup table and then the bit pattern is moved to DISPSSEGAS. The FSR register is incremented (moving to the next digit) and the process is repeated for the remaining 3 digits. To maintain proper multiplexing, task_scan is called throughout this routine.

turnon_scan routine - Turns on the LEDs and restores a legal scan position.

To save battery power, the displays are automatically shut off after 8 seconds when no buttons are pressed. The DISPON bit is used to preset the remaining display on time to 8 seconds. This routine sets this flag (to later turn on the displays) and then checks to see if the PREVSCAN register contains a legal value (an illegal value of "FFh" is used to turn off all of the displays) and it restores a legal value if the displays were off.

scan_keys routine - Turns off LEDs for a moment and scans the push-button inputs.

To reduce the number of I/Os needed by this application, the three user input keys are multiplexed onto the LED display segments through PORTB. First, the PORTB is cleared and PORTA is set to '0fh', turning off the LED displays. Next, PORTB is set up with bits 4,5 and 6 as inputs to read the keys. TEMP is then loaded with the keys that have changed state (to detect the falling edge of a key press) and KEYPAT is loaded with a pattern ('0' = not pressed, '1' = pressed) for the keys that are pressed. Lastly, PORTB is restored to all outputs and the current multiplex scan is restored to PORTA.

check_time routine - Checks for alarm or countdown timer expiration.

Each second, alarm conditions must be detected and the buzzer sounded if an alarm condition is true. ALARMNOW and EGGNOW are flag bits that are used by the main program to sound the buzzer if they are set. This routine starts by setting both ALARMNOW and EGGNOW. Next, the current time hours and minutes are compared (through a subtraction and a test of the STATUS register Z bit) with the alarm time. If there is any miscompare, the ALARMNOW bit is cleared. To finish, the countdown timer time minutes and seconds digits are each compared with zero. If there is any miscompare, the EGGNOW bit is cleared. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

inc_time routine - Adds one second, minute or hour to the clock, alarm or timer.

Every second, inc_time is called by main_loop to increment the seconds count for the clock. This routine is also called when the "UP" key is pressed and "MODE" key is held down to adjust the current time, alarm time or set the countdown timer. This routine uses indirect addressing to reduce the amount of code and simplify it's operation. Before this routine is called, the W register is loaded with the address of the clock second register and the routine is called. The FSR register is loaded with this value and the indirect address register is incremented (effectively incrementing the seconds counter).

Once the second counter is incremented, this register is checked for overflow (greater than 59 seconds) and if no overflow occurred, the routine returns. If an overflow happened, the second counter is cleared and the minute low digit is incremented. This register is then checked for an overflow (greater than 9 minutes) and so on until all the digits are updated.

This routine can also be called from multiple points. If called with the label inc_min_id, only the minutes (and hours if an overflow occurs) will be incremented. Additionally, calling inc_hour_id will increment only the hour digits. These features are used when setting the clock or alarm function. The FLAGS register (bits 0 and 1) is used to determine the current mode (clock, alarm or countdown timer) and ensure proper overflow calculations. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.
flag bits. If the alarm is enabled, the buzzer is buzzed by calling buzz_now; however, the main timer updatescaler for the TMRO register and internal instruction of time.

divide this 250 ms event by 4 to call inc_time every sec­
and bit7 changes every (122.07 µs • 16 • 16 • 128) =
250 ms. Bits 5 and 6 of the FLAGS register are used to
need to occur in between buzzer beeps to keep track
of the seconds counter).

Once the second counter LSD is decremented, this register is checked for underflow (less than 0 seconds) and if no underflow occurred, the routine returns. If an underflow happened, the second counter LSD is set to 9 and the second MSD is decremented. This register is then checked for an underflow (less than 0 seconds) and so on until all digits are updated.

This routine also can be called from multiple points. If called with the label dec_hour ld_vec, only the hour digits (or minutes if it is the countdown timer) will be decremented. This feature is used when setting the clock or alarm function. The FLAGS register (bits 0 and 1) is used to determine the current mode (clock, alarm or countdown timer) and ensure proper underflow calcu­lations. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

main_loop routine - Calls the above routines as needed and keeps track of when to increment the clock or decrement the countdown timer.

The main_loop calls all of the previous routines as neces­sary to maintain time, LED multiplexing, alarming and setting each function. The OPTION register is loaded with a 03h value to set up a Divide by 16 prescaler for the TMRO register and internal instruction cycle increment. The instruction cycle is 122.07 µs; therefore, bit0 changes every (122 µs • 16) = 1.953 ms and bit7 changes every (122.07 µs • 16 • 16 • 128) = 250 ms. Bits 5 and 6 of the FLAGS register are used to divide this 250 ms event by 4 to call inc_time every sec­ond.

The check_time routine is called after calling inc_time (every second), setting the EGGNOW or ALARMNOW flag bits. If the alarm is enabled, the buzzer is buzzed by calling buzz_now; however, the main timer updates need to occur in between buzzer beeps to keep track of time.

Every 500 ms, the keys are scanned and the edges on the MODE key are detected. Pressing the UP or DOWN key will shut off the buzzer (clearing the enable bits) and pressing the MODE key will advance the current mode. The mode is a 4-state state machine, revolving between the following states:

1. Display OFF - saves battery power - defaults to this mode if no keys are pressed for 8 seconds.
2. Display or Set countdown timer (holding MODE key allows setting).
3. Display or Set Alarm time (holding MODE key allows setting).
4. Display or Set Clock time (holding MODE key allows setting).

Next, the UP and DOWN keys are tested and if the MODE and UP are both pressed, the currently displayed mode time is incremented or decre­mented. If MODE is not pressed and UP or DOWN is pressed, the displays are turned on, but the displayed time is not altered.

DISPONCNT is used to keep track of how long the dis­plays have been on once all buttons are released. After 8 seconds, the displays are automatically turned off to save power. MODE_COUNT is used to switch from setting the right hand displays (minutes or seconds) to the left hand displays (hours or minutes). When the UP or DOWN button is held with mode for more than 4 seconds consecutively, MODE_COUNT reaches zero, switching from the right to left hand displays.

Finally, the main_loop finishes by updating the display registers by calling disp_value and if DISPONCNT has decreased to zero, the displays are turned off.

Lookup Tables - Convert a number into a bit pattern or RAM address.

There are three lookup tables used in this design for BCD to 7-Segment decoding, manufacturing diagnostics and looking up the address of the currently displayed mode.

- mode_timer - look-up the address of the clock, Alarm or Timer data storage RAM.
- led_lookup - look-up table contains the bitmap display pattern for displaying digits 0-9.
- mfg_led_lookup - look-up table contains the bit­map display pattern used for manufacturing mode. Only one segment is lit at a time.

Miscellaneous routines used for initialization and manufacturing test:

- init - Initializes all of RAM to zero, sets up the I/O ports and sets default time values.
- mfg_selftest - Used in manufacturing mode only - tests each LED segment, push-button, buzzer and display separately to expose bad keys, connec­tions, buzzer or displays.
CONCLUSION

The implementation of this application highlights the PIC16C54’s highly efficient instruction set, low power and frequency operation, high current direct LED drive capability and high performance instruction execution. Many of the routines used in this application note apply to a variety embedded control applications.

Ram Used: 25 Bytes

Code Space
  Used: 444 Words (without manufacturing diagnostics)

510 Words (including manufacturing diagnostics)
### APPENDIX A: CODE

**MPASM 01.21.03 Intermediate CLK8.ASM 8-21-1995 9:17:56 PAGE 1**

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJECT LINE SOURCE TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>; ***********************</td>
</tr>
<tr>
<td>00002</td>
<td>* PIC Egg Timer Give-Away</td>
</tr>
<tr>
<td>00003</td>
<td>*</td>
</tr>
<tr>
<td>00004</td>
<td>* Author: John Day</td>
</tr>
<tr>
<td>00005</td>
<td>* Sr. Field Applications Engineer</td>
</tr>
<tr>
<td>00006</td>
<td>* Northeast Region</td>
</tr>
<tr>
<td>00007</td>
<td>*</td>
</tr>
<tr>
<td>00008</td>
<td>* Revision: 1.2</td>
</tr>
<tr>
<td>00009</td>
<td>* Date September 22, 1994</td>
</tr>
<tr>
<td>00010</td>
<td>* Part: PIC16C54-LP/P or PIC16LC54A/P</td>
</tr>
<tr>
<td>00011</td>
<td>* Fuses: OSC: LP</td>
</tr>
<tr>
<td>00012</td>
<td>* WDT: OFF</td>
</tr>
<tr>
<td>00013</td>
<td>* PuT: OFF</td>
</tr>
<tr>
<td>00014</td>
<td>* CP: OFF</td>
</tr>
<tr>
<td>00015</td>
<td>; ***********************</td>
</tr>
</tbody>
</table>

This program is intended to run on a 32 Khz watch crystal and connects to four multiplexed seven segment displays. It displays the current time, alarm time and egg count down timers. There are switches that allow the user to set the alarm, timer and clock functions.

```
00022 LIST F=INHX8M,P=16C54
00023 INCLUDE "pl6C5X.inc"
00001 LIST
Pl6C5X.INC Standard Header File, Version 2.02 Microchip Technology, Inc.

~FUSES
_CP_OFF&_WDT_OFF&_LP_OSC

ORG 07h

00002
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015

; Static RAM Register File Definitions

00030 INDDOR EQU 0 ; Indirect address register
00031 DISPSEG_A EQU 07h ; Current Display A segment bit pattern
00032 DISPSEG_B EQU 08h ; Current Display B segment bit pattern
00033 DISPSEG_C EQU 09h ; Current Display C segment bit pattern
00034 DISPSEG_D EQU 0Ah ; Current Display D segment bit pattern
00035 CLK_SEC EQU 08h ; Clock second counter (0-59)
00036 CLK_MIN_LD EQU 0Ch ; Clock minute low digit counter (0-9)
00037 CLK_MIN_HD EQU 0Dh ; Clock minute high digit counter (0-5)
00038 CLK_HOUR_LD EQU 0Eh ; Clock hour low digit counter (0-9)
00039 CLK_HOUR_HD EQU 0Fh ; Clock hour high digit counter (0-2)
00040 ALM_MIN_LD EQU 10h ; Alarm minute low digit counter (0-9)
00041 ALM_MIN_HD EQU 11h ; Alarm minute high digit counter (0-5)
00042 ALM_HOUR_LD EQU 12h ; Alarm hour low digit counter (0-9)
00043 ALM_HOUR_HD EQU 13h ; Alarm hour high digit counter (0-2)
00044 TMR_SEC_LD EQU 14h ; Timer second low digit counter (0-9)
00045 TMR_SEC_HD EQU 15h ; Timer second high digit counter (0-5)
00046 TMR_MIN_LD EQU 16h ; Timer minute low digit counter (0-9)
00047 TMR_MIN_HD EQU 17h ; Timer minute high digit counter (0-2)
00048 KEYPAT EQU 18h ; Currently pressed key bits
00049 FLAGS EQU 19h ; Status of alarms, display on, etc.
00050 PRETMRO EQU 1Ah ; Used to determine which TMRO bits changed
00051 PREVSCAN EQU 1Bh ; Store Common Cathode display scan state
00052 TEMP EQU 1Ch ; Temporary storage
00053 DISPOCNT EQU 1Dh ; Time the displays have been on
```
# AN615

000001E 00054 MODE_COUNT EQU 1En ; Current mode state
000001F 00055 ALARMCNT EQU 1Fh ; Time the alarm has been sounding
000002 00056 ; **************************************************
000002 00057 ; * Flag and state bit definitions *
000002 00058 ; **************************************************
000002 00059 #define SECBIT TEMP,7 ; Bit to spawn 1/4 second count
000002 00060 #define SCANBIT TMRO,0 ; Bit to spawn display MUX
000002 00061 #define MODEKEY KEYPAT,4 ; Bit for MODEKEY pressed
000002 00062 #define UPKEY KEYPAT,6 ; Bit for UPKEY pressed
000002 00063 #define DOWNKEY KEYPAT,5 ; Bit for DOWNKEY pressed
000002 00064 #define MODEKEYCHG TEMP,4 ; Bit for delta MODEKEY
000002 00065 #define TIMENOW FLAGS,7 ; Flag to indicate 1 second passed
000002 00066 #define ALARMNOW FLAGS,3 ; Flag to indicate wakeup alarm
000002 00067 #define EGGNOW FLAGS,4 ; Flag to indicate egg timer alarm
000002 00068 #define ALARMOK STATUS,PAO ; Flag to enable wakeup alarm
000002 00069 #define EGGOK STATUS,PA1 ; Flag to enable timer alarm
000002 00070 #define BUZZEROUT PORTB,7 ; Pin for pulsing the buzzer
000002 00071 #define DISPON DISPONCNT,4 ; Bit to turn on LED displays
000002 00072
000002 00073 ; **************************************************
000002 00074 ; * Various Constants used throughout the program *
000002 00075 ; **************************************************
000002 00076 SEC_MAX EQU .60 ; Maximum value for second counter
000002 00077 MIN_LD_MAX EQU .10 ; Maximum value for low digit of minute
000002 00078 MIN_HD_MAX EQU .6 ; Maximum value for high digit of minute
000002 00079 HOUR_LD_MAX EQU .4 ; Maximum value for low digit of hour
000002 00080 HOUR_HD_MAX EQU .2 ; Maximum value for high digit of hour
000002 00081 OPTION_SETUP EQU b'00000011' ; TMRO - internal, /16 prescale
000002 00082 BUZINITVAL EQU 7 ;
000002 00083 INIT_MODE_COUNT EQU 8 ; Digit counts to move to hour digits
000002 00084 ALARMCYCNT EQU .40 ; Alarm for 10 seconds (ALARMCYCNT/4)
000002 00085
000002 00086 ORG 01FFh ; The PIC55x reset vector is at end of memory
000002 00087 reset_vector
000002 00088 GOTO init ; Jump to the initialization code
000002 00089
000002 00090 ORG 0
000002 00091 ; **************************************************
000002 00092 ; * Current mode look-up table *
000002 00093 ; **************************************************
000002 00094 mode_timer
000002 00095 ANDLW 3 ; Mask off upper bits just in case
000002 00096 ADDWF PCL,F ; Jump to one of 4 look-up entries
000002 00097 RETLW TMRO_SEC_LD ; Return the address of the 99 min timer RAM
000002 00098 RETLW ALM_MIN_LD ; Return the address of the alarm RAM
000002 00099 RETLW CLK_MIN_LD ; Return the address of the clock RAM
000002 000100 RETLW CLK_MIN_LD ; Return the address of the clock RAM
000002 000101
000002 000102 ; **************************************************
000002 000103 ; * Buzz the buzzer for 1/8 second *
000002 000104 ; **************************************************
000002 000105 buzz_now
000002 000106 CLRF PORTB ; Shut off the segments
000002 000107 buzz_now_dispon
000002 000108 CLRF TEMP ; Buzz for 256 pulses
000002 000109 loop_buz
000002 000110 BSF BUZZEROUT ; Send out pulse
000002 000111 BCF BUZZEROUT ; Clear out the pulse
000002 000112 DECSZ TEMP,F ; Decrement counter and skip when done
000002 000113 GOTO loop_buz ; Go back and send another pulse
000002 000114 RETLW 0 ; We are done so come back!
000002 000115
000002 000116 ; **************************************************
000002 000117 ; * Mux drive the next LED display digit *
000002 000118 ; **************************************************
000002 000119 task_scan ; (19 (next_scan) + 2 = 21 cycles - must be called every 11 cy)
00D 0601    00120    BTFSC ScanBit    ; Synch up with 3.9 mS timer bit
00E 0A0D    00121    GOTO task_scan    ; Jump back until bit is clear

00F 035B    00124    RLF PREVSCAN,W    ; Move to the next digit select into C
0010 073B    00125    BTFSS PREVSCAN,1    ; 0 Check if display A was on before
0011 0209    00126    MOVF DISPSSEG5_C,W    ; Place display B value into W
0012 071B    00127    BTFSS PREVSCAN,0    ; 1 Check if display B was on before
0013 0208    00128    MOVF DISPSSEG5_B,W    ; Place display C value into W
0014 077B    00129    BTFSS PREVSCAN,3    ; 2 Check if display C was on before
0015 0207    00130    MOVF DISPSSEG5_A,W    ; Place display D value into W
0016 075B    00131    BTFSS PREVSCAN,2    ; 3 Check if display D was on before
0017 020A    00132    MOVF DISPSSEG5_D,W    ; Place display A value into W
0018 0066    00133    CLR PORTB    ; Turn off all segments
0019 037B    00134    RLF PREVSCAN,F    ; Move to the next digit
001A 0365    00135    RLF PORTA,F    ; Move port to the next digit
001B 0026    00136    MOVWF PORTB    ; Place next segment value on PORTB
001C 021B    00137    MOVF PREVSCAN,W    ; Restore the port in case it is wrong
001D 0025    00138    MOVWF PORTA    ; Restore the port
001E 0800    00139    RETLW 0    ; Display is updated - now return

00140
00141

00142 ;********************************************************************************
00143 ; * Move new digit display info out to display *
00144 ;********************************************************************************

001F 0024    00146    MOVWF FSR    ; Place W into FSR for indirect addressing
0020 090D    00147    CALL task_scan    ; Scan the next LED digit.
0021 0200    00148    MOVF INDADDR,W    ; Place display value into W
0022 0937    00149    CALL led_lookup    ; Look up seven segment value
0023 0027    00150    MOVWF DISPSSEG5_A    ; Move value out to display register A
0024 02A4    00151    INCF FSR,F    ; Go to next display value
0025 090D    00152    CALL task_scan    ; Scan the next LED digit.
0026 0200    00153    MOVF INDADDR,W    ; Place display value into W
0027 0937    00154    CALL led_lookup    ; Look up seven segment value
0028 0028    00155    MOVWF DISPSSEG5_B    ; Move value out to display register B
0029 02A4    00156    INCF FSR,F    ; Go to next display value
002A 090D    00157    CALL task_scan    ; Scan the next LED digit.
002B 0200    00158    MOVF INDADDR,W    ; Place display value into W
002C 0937    00159    CALL led_lookup    ; Look up seven segment value
002D 0029    00160    MOVWF DISPSSEG5_C    ; Move value out to display register C
002E 02A4    00161    INCF FSR,F    ; Go to next display value
002F 090D    00162    CALL task_scan    ; Scan the next LED digit.
0030 0200    00163    MOVF INDADDR,W    ; Place display value into W
0031 0643    00164    BTFSC STATUS, Z    ; ZBLEFT - Check for a zero
0032 02A0    00165    CALL led_lookup    ; Place display D value into W
0033 0937    00166    CALL led_lookup    ; Look up seven segment value
0034 002A    00167    MOVWF DISPSSEG5_D    ; Move value out to display register D
0035 090D    00168    CALL task_scan    ; Scan the next LED digit.
0036 0800    00169    RETLW 0

00170

00171 ;******************************************************************************
00172 ; * Convert display value into segments *
00173 ;******************************************************************************

0077 014F    00175    ANDLW 0Fh    ; Strip off upper digits
0078 0182    00176    ADDWF PCL,F    ; Jump into the correct location
0079 083F    00177    RETLW b'00111111'    ; Bit pattern for a Zero
007A 0806    00178    RETLW b'00000010'    ; Bit pattern for a One
007B 085B    00179    RETLW b'01011011'    ; Bit pattern for a Two
007C 084F    00180    RETLW b'01001111'    ; Bit pattern for a Three
007D 0866    00181    RETLW b'01100110'    ; Bit pattern for a Four
007E 086D    00182    RETLW b'01101101'    ; Bit pattern for a Five
007F 087D    00183    RETLW b'01111101'    ; Bit pattern for a Six
0040 0807    00184    RETLW b'00000111'    ; Bit pattern for a Seven
0041 087F    00185    RETLW b'01111111'    ; Bit pattern for a Eight
0042 085F 00186 RETLW b'01101111' ; Bit pattern for a Nine
0043 0800 00187 RETLW 0 ; Turn display off - ILLEGAL VALUE
0044 0800 00188 RETLW 0 ; Turn display off - ILLEGAL VALUE
0045 0800 00189 RETLW 0 ; Turn display off - ILLEGAL VALUE
0046 0800 00190 RETLW 0 ; Turn display off - ILLEGAL VALUE
0047 0800 00191 RETLW 0 ; Turn display off - ILLEGAL VALUE
0048 0800 00192 RETLW 0 ; Turn display off - ILLEGAL VALUE
00193
00194 ; **************************************************
00195 ; Convert display value into single segment ON for manufacturing diags *
00196 ; **************************************************

0049 00197 mfg_led_lookup

0049 0E07 00198 ANDLW 07h ; Strip off upper digits
004A 01E2 00199 ADCWF PCL,F ; Jump into the correct location
004B 0801 00200 RETLW b'00000001' ; Bit pattern for segment A on only
004C 0802 00201 RETLW b'00000010' ; Bit pattern for segment B on only
004D 0804 00202 RETLW b'00000100' ; Bit pattern for segment C on only
004E 0808 00203 RETLW b'00001000' ; Bit pattern for segment D on only
004F 0810 00204 RETLW b'00010000' ; Bit pattern for segment E on only
0050 0820 00205 RETLW b'00100000' ; Bit pattern for segment F on only
0051 0840 00206 RETLW b'01000000' ; Bit pattern for segment G on only
0052 087F 00207 RETLW b'01111111' ; Bit pattern for all segments on
00208
00209 ; **************************************************
00210 ; * Wake-up and turn on the displays *
00211 ; **************************************************

0053 00212 turnon_scan

0053 059D 00213 BSF DISPON ; Set display ON bit
0054 0CE8 00214 MOVLW b'11101110' ; Place digit 0 scan pattern in W
0055 019B 00215 XORWF PREVSCAN,W ; See if this is the current scan
0056 0643 00216 BTFS SCAN, Z ; Skip if this is not the current scan
0057 0800 00217 RETLW 0 ; Legal scan value - we are done!
0058 OCDD 00218 MOVLW b'11011101' ; Place digit 1 scan pattern in W
0059 019B 00219 XORWF PREVSCAN,W ; See if this is the current scan
005A 0643 00220 BTFS SCAN, Z ; Skip if this is not the current scan
005B 0800 00221 RETLW 0 ; Legal scan value - we are done!
005C 0CB8 00222 MOVLW b'10111011' ; Place digit 2 scan pattern in W
005D 019B 00223 XORWF PREVSCAN,W ; See if this is the current scan
005E 0643 00224 BTFS SCAN, Z ; Skip if this is not the current scan
005F 0800 00225 RETLW 0 ; Legal scan value - we are done!
0060 0C77 00226 MOVLW b'01111011' ; Place digit 3 scan pattern in W
0061 019B 00227 XORWF PREVSCAN,W ; See if this is the current scan
0062 0643 00228 BTFS SCAN, Z ; Skip if this is not the current scan
0063 0800 00229 RETLW 0 ; Legal scan value - we are done!
0064 0CE8 00230 MOVLW 0EH ; Move digit 0 scan value into W
0065 003B 00231 MOVLW PREVSCAN ; Move it into scan pattern register
00232
00233 ; **************************************************
00234 ; * Scan for pressed keys *
00235 ; **************************************************

0066 0066 00237 CLRIF PORTB ; Turn off all of the segments
0067 0CFF 00238 MOVLW 0FFH ; Place FF into W
0068 0025 00239 MOVF PORTA ; Make PORT A all ones
0069 0C70 00240 MOVLW b'01110000' ; Place 70 into W
006A 0006 00241 TRIS PORTB ; Make RB4,5,6 inputs others outputs
006B 0206 00242 MOVF PORTB,W ; Place keyscan value into W
006C 019B 00243 XORWF KEYPAT,W ; Place Delta key press into W
006D 003C 00244 MOVF TEMW ; Place Delta key press into TEMP
006E 01B8 00245 XORWF KEYPAT,F ; Update KEYPAT reg to buttons pressed
006F 0040 00246 CLRW ; Place 0 into W
0070 0006 00247 TRIS PORTB ; Make PORT B outputs
0071 021B 00248 MOVF PREVSCAN,W ; Place previous scan value into W
0072 0025 00249 MOVF PORTA ; Turn on the scan
0073 0800 00250 RETLW 0
00251 ; **************************************************
00252 ; * Check if alarm or timer is expired *
00253 ; ********************************************

00254 ;
0074 090D 00255 CALL task_scan ; Scan the next LED digit.
0075 0579 00256 BSF ALARMNOW ; Set the alarm bit
0076 0599 00257 BSF EGNNOW ; Set the Egg timer alarm bit
0077 0210 00258 MOVF ALM_MIN_LD,W ; Place alarm minute counter into W
0078 008C 00259 SUBWF CLK_MIN_LD,W ; CLK_MIN_LD - W -> W
0079 0743 00260 BTFSS STATUS,Z ; Skip if they are equal
007A 0479 00261 BCF ALARMNOW ; They are not equal so clear alarm bit
007B 0211 00262 MOVF ALM_MIN_HI,W ; Place alarm minute counter into W
007C 008D 00263 SUBWF CLK_MIN_HI,W ; CLK_MIN_HI - W -> W
007D 0743 00264 BTFSS STATUS,Z ; Skip if they are equal
007E 0479 00265 BCF ALARMNOW ; They are not equal so clear alarm bit
007F 090D 00266 CALL task_scan ; Scan the next LED digit.
0080 0212 00267 MOVF ALM_HOUR_LD,W ; Place alarm hour counter into W
0081 008E 00268 SUBWF CLK_HOUR_LD,W ; CLK_HOUR_LD - W -> W
0082 0743 00269 BTFSS STATUS,Z ; Skip if they are equal
0083 0479 00270 BCF ALARMNOW ; They are not equal so clear alarm bit
0084 0213 00271 MOVF ALM_HOUR_HI,W ; Place alarm hour counter into W
0085 008F 00272 SUBWF CLK_HOUR_HI,W ; CLK_HOUR_HI - W -> W
0086 0743 00273 BTFSS STATUS,Z ; Skip if they are equal
0087 0479 00274 BCF ALARMNOW ; They are not equal so clear alarm bit
0088 090D 00275 CALL task_scan ; Scan the next LED digit.
0089 0214 00276 MOVF TMR_SEC_LD,W ; Set the Z bit to check for zero
008A 0743 00277 BTFSS STATUS,Z ; Skip if this digit is zero
008B 0499 00278 BCF EGNNOW ; Timer is not zero so clear egg alarm bit
008C 0215 00279 MOVF TMR_SEC_HI,W ; Set the Z bit to check for zero
008D 0743 00280 BTFSS STATUS,Z ; Skip if this digit is zero
008E 0499 00281 BCF EGNNOW ; Timer is not zero so clear egg alarm bit
008F 0216 00282 MOVF TMR_MIN_LD,W ; Set the Z bit to check for zero
0090 0743 00283 BTFSS STATUS,Z ; Skip if this digit is zero
0091 0499 00284 BCF EGNNOW ; Timer is not zero so clear egg alarm bit
0092 090D 00285 CALL task_scan ; Scan the next LED digit.
0093 0217 00286 MOVF TMR_MIN_HI,W ; Set the Z bit to check for zero
0094 0743 00287 BTFSS STATUS,Z ; Skip if this digit is zero
0095 0499 00288 BCF EGNNOW ; Timer is not zero so clear egg alarm bit
0096 0799 00289 BTFSS STATUS,Z ; Skip if we are still at EGG Time
0097 05C3 00290 BSF EGGOK ; If we are not at EGG time, re-set egg alarm
0098 0779 00291 BTFSS ALARMNOW ; Skip if we are still at Alarm time
0099 05A3 00292 BSF ALARMNOW ; If we are not at Alarm time, re-set alarm
009A 090D 00293 CALL task_scan ; Scan the next LED digit.
009B 0800 00294 RETLW 0
00295
00296 ; ********************************************
009C 00297 ; * Increment the clock, timer or alarm *
009D 00298 ; ********************************************

009E 00299 inc_time
009F 0029A MOVWF FSR ; Add one to clock second counter
009G 0029B CALL task_scan ; Scan the next LED digit.
009H 0029C INCF INADDR,f ; Add one to minute lower digit
009I 0029D MOVWL SEC_MAX ; Place second max value into W
009J 0029E SUBWF INADDR,W ; CLOCK_SEC - SEC_MAX -> W
009K 0029F BTFSS STATUS,C ; Skip if there is an overflow
009L 002A0 RETLW 0 ; We are done so let's get out of here!
009M 002A1 CLRF CLK_SEC ; Clear CLK_second counter
009N 002A2 INCF FSR,F ; Move to the next digit
009O 002A3 INCF INADDR,F ; Add 1 to minute LOW digit
009P 002A4 GOTO skip_min_fsr ; Jump to the next digit
009Q 002A5 inc_min_fsr
009R 002A6 00311 MOVWF FSR ; Add one to clock second counter
009S 002A7 CALL task_scan ; Scan the next LED digit.
009T 002A8 00312 INCF INADDR,F ; Add 1 to minute LOW digit
009U 002A9 00314 skip_min_fsr
009V 002AA CALL task_scan ; Scan the next LED digit.
009W 002AB 00316 MOVWL MIN LD_MAX ; Place minute lower digit max value into W
009X 002AC 00317 SUBWF INADDR,W ; CLK_MIN_LD - MIN LD_MAX -> W
00318  BTFSS STATUS,C ; Skip if there is an overflow
00319  RETLW 0 ; We are done so let's get out of here!
00320  CLRF INADDR ; Clear CLK minute low digit
00321  INCF FSR,F ; Move to the minute high digit
00322  INCF INADDR,F ; Add one to minute high digit
00323  inc_min_hd
00324  CALL task_scan ; Scan the next LED digit.
00325  MOVLW MIN_HD_MAX ; Place minute high digit max value into W
00326  SUBWF INADDR,W ; CLK_MIN_HD - MIN_HD_MAX -> W
00327  BTFSS STATUS,C ; Skip if there is an overflow
00328  RETLW 0 ; We are done so let's get out of here!
00329  CLRF INADDR ; Clear CLK minute high digit
00330  INCF FSR,F ; Move to the hour low digit
00331  INCF INADDR,F ; Add one to hour low digit
00332  GOTO skip_hour_fsr ; Jump to the next digit
00333  inc_hour_ld
00334  MOVLW FSR
00335  INCF FSR,F
00336  INCF FSR,F
00337  INCF INADDR,F ; Add 1 to minute LOW digit
00338  skip_hour_fsr
00339  CALL task_scan ; Scan the next LED digit.
00340  MOVLW MIN_LD_MAX ; Place hour lower digit max value into W
00341  SUBWF INADDR,W ; CLK_HOUR_LD - HOUR_LD_MAX -> W
00342  BTFSS STATUS,C ; Skip if there is an overflow
00343  GOTO check_inc ; We need to check for overflow
00344  CLRF INADDR ; Clear CLK hour low digit
00345  INCF FSR,F ; Move to the hour high digit
00346  INCF INADDR,F ; Add one to hour high digit
00347  GOTO inc_hour_hd
00348  check_inc
00349  INCF FSR,F ; Move to hour high digit
00350  inc_hour_hd
00351  CALL task_scan ; Scan the next LED digit.
00352  MOVLW HOUR_HD_MAX ; Place hour high digit max value into W
00353  BTFSC FLAGS,1
00354  GOTO off_model
00355  BTFSC FLAGS,0
00356  MOVLW MIN_LD_MAX-1
00357  off_model
00358  SUBWF INADDR,W ; CLK_HOUR_LD - HOUR_LD_MAX -> W
00359  BTFSS STATUS,C ; Skip if there is an overflow
00360  RETLW 0 ; We are done so let's get out of here!
00361  DECF FSR,F ; Move to the hour low digit
00362  CALL task_scan ; Scan the next LED digit.
00363  MOVLW HOUR_HD_MAX ; Place hour high digit max value into W
00364  BTFSC FLAGS,1
00365  GOTO off_mode2
00366  BTFSC FLAGS,0
00367  MOVLW 0 ; Clear W
00368  off_mode2
00369  SUBWF INADDR,W ; CLK_HOUR_LD - HOUR_LD_MAX -> W
00370  BTFSS STATUS,C ; Skip if there is an overflow
00371  RETLW 0 ; We are done so let's get out of here!
00372  CALL task_scan ; Scan the next LED digit.
00373  CLRF INADDR ; Clear hour high digit
00374  BTFSC FLAGS,1
00375  GOTO off_mode3
00376  BTFSC FLAGS,0
00377  off_mode3
00378  NOP
00379  INCF FSR,F ; Move to the hour high digit
00380  CLRF INADDR ; Clear one hour low digit
00381  CALL task_scan
00382  RETLW 0 ; We are done so let's get out of here!
0085 00384 dec_hour_ld
0085 0AF9 00385 GOTO dec_hour_ld_vect ; ran out of CALL space....
0086 00386
0087 00387 ; ****************************************
0088 00388 * Decriment the clock, alarm or timer *
0089 00389 ; ****************************************
0086 00390 dec_time
0086 00391 dec_min __ ld
0087 0101 00402 INCF FSR,F ; Move the pointer to Min HIGH DIGIT
0088 0102 00403 DECF INADDR,F ; Subtract one from CLK_MIN_HD
0089 0103 00404 COMF INADDR,W ; Set the Z bit to check for zero
0080 0743 00405 BTFSS STATUS,Z ; Skip if CLK_MIN_LD is zero
007E 0800 00406 RETLW 0 ; We are done... Let's get out of here
007C 0C09 00407 MOVLW MIN_HD_MAX - 1 ; Place minute lower digit max value into W
007E 0220 00408 MOVF INADDR ; MIN_LD_MAX -> CLK_MIN_LD
007E 00409 MOVLW MIN_HD_MIN - 1 ; Place minute lower digit max value into W
007E 00410 CALL task_scan ; Scan the next LED digit.
007E 02A4 00411 INCF FSR,F ; Move the pointer to Hour LOW DIGIT
007E 0AFF 00412 dec_hour_ld_vect
007E 0024 00413 MOVWF FSR ; Set up pointer for indirect address
007E 02A4 00414 INCF FSR,F ; Move the pointer to Min LOW DIGIT
007E 02A4 00415 INCF FSR,F
007E 090D 00416 CALL task_scan ; Scan the next LED digit.
007E 02A4 00417 skip_dhour_fsr
007E 0024 00418 MOVF FSR
007E 02A4 00419 INC FSR,F
007E 02A4 00420 INC FSR,F
007E 090D 00421 GOTO check_hour
007E 0B06 00422 MOVLW MIN_LD_MAX - 1 ; Place minute lower digit max value into W
007E 0220 00423 MOVF INADDR ; MIN_LD_MAX -> CLK_MIN_LD
007E 02A4 00424 INCF FSR,F ; Move the pointer to Hour LOW DIGIT
007E 0020 00425 DECF INADDR,F ; Subtract one from CLK_MIN_HD
007E 0020 00426 GOTO dec_hour_hd
007E 02A4 00427 check_hour
007E 02A4 00428 INCF FSR,F ; Point to hour high digit
007E 02A4 00429 dec_hour_hd
007E 090D 00430 CALL task_scan ; Scan the next LED digit.
007E 0240 00431 COMF INADDR,W
007E 0743 00432 BTFSS STATUS,Z
007E 0800 00433 RETLW 0
007E 090D 00434 CALL task_scan ; Scan the next LED digit.
007E 00E4 00435 DECF FSR,F
007E 00E4 00436 MOVLW .9 ; Reset digit to 9
007E 0080 00437 SUBWF INADDR,W
007E 0743 00438 BTFSS STATUS,Z ; Skip if CLK_MIN_LD is zero
007E 0800 00439 RETLW 0 ; We are done... Let's get out of here
007E 090D 00440 CALL task_scan ; Scan the next LED digit.
007E 02A4 00441 INCF FSR,F
007E 0C02 00442 MOVLW HOUR_HD_MAX ; Place minute lower digit max value into W
007E 0739 00443 BTFSS FLAGS,1 ; Skip if CLOCK or ALARM mode
007E 0C09 00444 MOVLW .9 ; Reset digit to 9
007E 0020 00445 MOVF INADDR ; HOUR_LD_MAX -> CLK_HOUR_HD
007E 0C03 00446 MOVLW HOUR_HD_MIN - 1 ; Place minute lower digit max value into W
007E 0739 00447 BTFSS FLAGS,1 ; Skip if CLOCK or ALARM mode
007E 0C09 00448 MOVLW .9 ; Reset digit to 9
007E 00E4 00449 DECF FSR,F ; Move the pointer to Min LOW DIGIT
007E 0080 00450 MOVWF INADDR ; INADDR -> CLK_MIN_HD
AN615

011B 0020 00450 MOVWF INADDR ; HOUR_LD_MAX -> CLK_HOUR_LD
011C 090D 00451 CALL task_scan ; Scan the next LED digit.
011D 0800 00452 RETLW 0 ; We are done... Let's get out of here
00453
00454 ; ******************************************************
00455 ; * Main loop calls all tasks as needed *
00456 ; ******************************************************
011E 00457 main_loop
011E 090D 00458 CALL task_scan ; Scan the next LED digit.
011F 0201 00459 MOVFP TMRO,W ; Place current TMRO value into W
0120 019A 00460 XORWF PREVTMRO,W ; Lets see which bits have changed...
0121 003C 00461 MOVFP TEMP ; All changed bits are placed in temp for test
0122 018A 00462 XORWF PREVTMRO,F ; Update Previous TMRO value.
0123 07FC 00463 BTFS FF ; Skip if it is not time to increment second
0124 0B1E 00464 GOTO main_loop ; Go back to main loop if 250 ms not passed
0125 0C20 00465 MOVLW b'00100000' ; Bits 6 and 5 of FLAGS used as divide by 4
0126 01F9 00466 ADDWF FLAGS,F ; Add one to bit 5
0127 07FF 00467 BTFS TIME NOW ; Check bit 7 - if four adds occur, skip
0128 0B38 00468 GOTO skip_timer ; One second has not passed - skip timers
0129 090D 00469 CALL task_scan ; Scan the next LED digit.
012A 04F9 00470 BCF TIME_NOW ; Clear out second passed flag
012B 0C0B 00471 MOVLW CLK_SEC ; Place pointer to increment clock
012C 099C 00472 CALL inc_time ; Increment the clock
012D 0974 00473 CALL check_time ; Check for alarm or timer conditions
012E 0699 00474 BTFS EGGNOW ; Do NOT decrease timer if zero
012F 0B38 00475 GOTO skip_timer ; Jump out if egg timer is zero
0130 068D 00476 BTFS UPKEY ; Skip if UP key is NOT pressed
0131 0B39 00477 GOTO skip_timer ; Jump out if UP key is pressed
0132 06B8 00478 BTFS DOWNKEY ; Skip if DOWN key is NOT pressed
0133 0B38 00479 GOTO skip_timer ; Jump out if DOWN key is pressed
0134 0C14 00480 MOVLW TMR_SEC_LD ; Place pointer to decrement timer
0135 0986 00481 CALL dec_time ; Decrement countdown timer
0136 0C28 00482 MOVLW ALARMCYCCNT ; Place the number of alarm beeps into W
0137 003F 00483 MOVWF ALARMCNT ; Move beep count to ALARMCNT
0138 00484 skip_timer
0138 07A3 00485 BTFS ALARMOK ; Skip if this is the first pass into alarm
0139 0B3F 00486 GOTO skip_wakeup ; Second pass - do not re-init ALARMCNT
013A 0779 00487 BTFS ALARMNOW ; Skip if this is alarm pass
013B 0B3F 00488 GOTO skip_wakeup ; Countdown timer - do not re-init ALARMCNT
013C 0C29 00489 MOVLW ALARMCYCCNT ; Place the number of alarm beeps into W
013D 003F 00490 MOVWF ALARMCNT ; Move beep count to ALARMCNT
013E 04A3 00491 BCF ALARMOK ; Clear flag for second pass
013F 00492 skip_wakeup
013F 090D 00493 CALL task_scan ; Scan the next LED digit.
0140 0679 00494 BTFS ALARMNOW ; Skip if alarm clock is not set
0141 0B45 00495 GOTO send_alarm ; Blast out a beep
0142 0699 00496 BTFS EGGNOW ; Skip if countdown timer is not alarming
0143 0B45 00497 GOTO send_alarm ; Blast out a beep
0144 0B4A 00498 GOTO skip_alarm ; Skip beeping and continue
0145 00499 send_alarm
0145 021F 00500 MOVF ALARMCNT,W ; Place ALARMCNT into W
0146 0643 00501 BTFS STATUS,Z ; Skip if not zero
0147 0B4A 00502 GOTO skip_alarm ; We are done beeping - skip and continue
0148 02FF 00503 DECFSP ALARMCNT,F ; Decrement beep count and skip when zero
0149 0906 00504 CALL buzz_now ; Blast out the beep!!!
014A 00505 skip_alarm
014A 07B9 00506 BTFS FLAGS,5 ; Skip if it is time to scan the keys 1/2 sec
014B 089A 00507 goto finish_update ; Jump to finish updates - don't scan
014C 0966 00508 CALL scan_keys ; Scan the keys and load value into KEYPAT
014D 090D 00509 CALL task_scan ; Scan the next LED digit.
014E 0798 00510 BTFS MODEKEY ; Skip if the MODEKEY is pressed
014F 0B55 00511 GOTO same_mode ; Not pressed so it is the same mode...
0150 079C 00512 BTFS MODEKEYCHG ; Skip if the is pressing edge
0151 0B55 00513 GOTO same_mode ; Button is held so it is the same mode...
0152 0289 00514 INCFSZ FLAGS,F ; Advance the mode by incrementing bits 0,1
0153 0459 00515 BCF FLAGS,2 ; Force mode to wrap-around by clearing bit 2
0154 0953 00516 CALL turnon_scan ; Mode button pressed - must turn on LEDs
0155 00517
0155 090D 00519 call task_scan ; Scan the next LED digit.
0156 06D8 00520 BTFSS UPKEY ; Skip if the UP key is not pressed
0157 0B66 00521 GOTO serve_up_key ; UP key is pressed - jump to serve it!
0158 06B8 00522 BTFSS DOWNKEY ; Skip if the DOWN key is not pressed
0159 0B81 00523 GOTO serve_down_key ; DOWN key is pressed - jump to serve it!
015A 0C08 00524 MOVVLW INIT_MODE_COUNT ; UP and DOWN not pressed - re-init mode count
015B 003E 00525 MOVFW MODE_COUNT ; Change back to lower digits for setting
015C 023D 00526 MOVF DISPONCNT,F ; Update Z bit in STATUS reg display on time
015D 0743 00527 BTFSS STATUS,Z ; Skip if displays should be OFF
015E 09FD 00528 DECF DISPONCNT,F ; Decrement display ON counter
015F 0743 00529 BTFSS STATUS,Z ; Skip if displays should be OFF
0160 0B9A 00530 GOTO finish_update ; Displays are ON - jump to finish updates
0161 0419 00531 BCF FLAGS,0 ; Restore the mode to displays OFF
0162 0439 00532 BCF FLAGS,1 ; Restore the mode to displays OFF
0163 0066 00533 CLRPF PORTB ; Clear out segment drives on PORTB
0164 0065 00534 CLRPF PORTA ; Clear out common digit drives on PORTA
0165 0B9A 00535 GOTO finish_update ; Jump to finish updates
0166 00536 serve_up_key
0166 090D 00537 call task_scan ; Scan the next LED digit.
0167 0619 00538 BTFSS Flags,0 ; Skip if not in TIMER or CLOCK mode
0168 0B6D 00539 GOTO no_up_display ; Currently in TIMER or CLOCK - keep mode
0169 0639 00540 BTFSS Flags,1 ; Skip if not in ALARM mode
016A 0B6D 00541 GOTO no_up_display ; Currently in ALARM - keep mode
016B 0519 00542 BSF Flags,0 ; Set to CLOCK mode
016C 0539 00543 BSF Flags,1 ; Set to CLOCK mode
016D 00544 no_up_display
016D 007F 00545 CLRF ALARMCNT ; A key was pressed, so turn off alarm
016E 0953 00546 call turnon_scan ; Turn on the LEDs
016F 079B 00547 BTFSS MODEKEY ; Skip if MODE is pressed as well
0170 0B9A 00548 GOTO finish_update ; MODE is not pressed - jump to finish update
0171 021E 00549 MOVF MODE_COUNT,W ; Update STATUS Z bit for mode count
0172 0743 00550 BTFSS STATUS,Z ; Skip if we have counted down to zero
0173 00FE 00551 DECF MODE_COUNT,F ; Decrement the mode count
0174 090D 00552 call task_scan ; Scan the next LED digit.
0175 021E 00553 MOVF MODE_COUNT,W ; Update the Z bit to check for zero
0176 0743 00554 BTFSS STATUS,Z ; Skip if we have incremented for 7 times
0177 0B7C 00555 GOTO serve_min_up ; Increment the minutes digits
0178 00D9 00556 DECF FLAGS,W ; Place current mode into W
0179 0900 00557 CALL mode_timer ; Lock-up register RAM address for current mode
017A 099A 00558 CALL inc_hour_ld ; Add one hour to the current display
017B 0B9A 00559 GOTO finish_update ; Jump to finish updates
017C 00560 serve_min_up
017C 090D 00561 call task_scan ; Scan the next LED digit.
017D 00D9 00562 DECF FLAGS,W ; Place current mode into W
017E 0900 00563 CALL mode_timer ; Lock-up register RAM address for current mode
017F 09A7 00564 CALL inc_min_ld ; Add one minute to the current display
0180 0B9A 00565 GOTO finish_update ; Jump to finish updates
0181 00566 serve_down_key
0181 090D 00567 call task_scan ; Scan the next LED digit.
0182 0619 00568 BTFSC Flags,0 ; Skip if not in TIMER or CLOCK mode
0183 08B8 00569 GOTO no_dn_display ; Currently in TIMER or CLOCK - keep mode
0184 0639 00570 BTFSC Flags,1 ; Skip if not in ALARM mode
0185 08B8 00571 GOTO no_dn_display ; Currently in ALARM - keep mode
0186 0519 00572 BSF Flags,0 ; Set to CLOCK mode
0187 0539 00573 BSF Flags,1 ; Set to CLOCK mode
0188 007F 00574 CLRF ALARMCNT ; A key was pressed, so turn off alarm
0189 0953 00576 CALL turnon_scan ; Turn on the LEDs
018A 079B 00577 BTFSS MODEKEY ; Skip if MODE is pressed as well
018B 0B9A 00578 GOTO finish_update ; MODE is not pressed - jump to finish update
018C 021E 00579 MOVF MODE_COUNT,W ; Update STATUS Z bit for mode count
018D 0743 00580 BTFSS STATUS,Z ; Skip if we have counted down to zero
018E 00FE 00581 DECF MODE_COUNT,F ; Decrement the mode count
AN615

00582
018F 090D 00583  call task_scan ; Scan the next LED digit.
0190 021E 00584  MOVF MODE_COUNT,W ; Update the Z bit to check for zero
0191 0743 00585  BTFSS STATUS,Z ; Skip if we have incremented for 7 times
0192 0B97 00586  GOTO serve_min_down ; Decrement the minutes digits
0193 00D9 00587  DECF FLAGS,W ; Place current mode into W
0194 0900 00588  CALL mode_timer ; Look-up register RAM address for current mode
0195 09B5 00589  CALL dec_min_ld  ; Subtract one hour from the current display
0196 0B9A 00590  GOTO finish_update ; Jump to finish updates
0197 00591  serve_min_down
0197 00D9 00592  DECF FLAGS,W ; Place current mode into W
0198 0900 00593  CALL mode_timer ; Look-up register RAM address for current mode
0199 0986 00594  CALL dec_min_ld  ; Subtract one hour from the current display
019A 00595  finish_update
019A 090D 00596  call task_scan ; Scan the next LED digit.
019B 0619 00597  BTFSC FLAGS,0 ; Skip if in mode OFF or ALARM
019C 0BA4 00598  GOTO new_display  ; Jump to update LED display registers
019D 0639 00599  BTFSC FLAGS,1 ; Skip if in mode OFF
019E 0BA4 00600  GOTO new_display  ; Jump to update LED display registers
019F 0067 00601  CLRF DISPSEGS_A ; Clear display regs to Shut off LED display
01A0 0068 00602  CLRF DISPSEGS_B ; Clear display regs to Shut off LED display
01A1 0069 00603  CLRF DISPSEGS_C ; Clear display regs to Shut off LED display
01A2 006A 00604  CLRF DISPSEGS_D ; Clear display regs to Shut off LED display
01A3 0B1E 00605  GOTO main_loop ; We are done - go back and do it again!
01A4 00606  new_display
01A4 00D9 00607  DECF FLAGS,W ; Move current mode state into W
01A5 0900 00608  CALL mode_timer ; Look-up register address of value to display
01A6 091F 00609  CALL disp_value  ; Update display registers with new values
01A7 0B1E 00610  GOTO main_loop ; We are done - go back and do it again!
01A8 00611
01A8 0C03 00616  MOVlw OPTION_SETUP ; Place option reg setup into W
01A9 0002 00617  OPTION ; Set up OPTION register
01AA 0C05 00618  MOVlw PORTA ; Place beginning of RAM/Port location into W
01AB 0024 00619  MOVWF FSR ; Now initialize FSR with this location
01AC 00620  clear_mem
01AC 00621  CLRF INDADDR ; Clear the FSR pointed memory location
01AD 03B4 00622  INCFSS FSR,F ; Point to the next location
01AE 0BAC 00623  GOTO clear_mem  ; Jump back to clear memory routine
01AF 0572 00624  BSF ALM_HOUR_LD,3 ; Place 0:00 into alarm register
01B0 02AE 00625  INCF CLK_HOUR_LD,F ; Place 1:00 into clock register
01B1 0CBE 00626  MOVlw 00BH ; Turn on display A scan line, others off.
01B2 003B 00627  MOVWF PRESCAN ;
01B3 0040 00628  CLRw
01B4 0006 00629  TRIS PORTB ; Make all Port B pins outputs.
01B5 0005 00630  TRIS PORTA ; Make all Port A pins outputs.
01B6 0539 00631  BSF FLAGS,1 ; Set up current mode to CLOCK, display ON
01B7 0519 00632  BSF FLAGS,0 ;
01B8 04A3 00633  BCF ALARMOK ; Don't want to trigger alarms
01B9 04C3 00634  BCF EGGOK ;
01BA 059D 00635  BSF DISPon ; Turn on the displays
01BB 0083 00636  mfg_checkkey
01BB 0966 00637  CALL scan_keys ; Lets see what is pressed
01BC 07D8 00638  BTFSS UPKEY ; Goto self-test if UP key is pressed at pwup
01BD 0B1E 00639  GOTO main_loop ; Normal operation - Jump to the main loop
01BE 00640
01BE 0C70 00645  MOVlw b'01110000' ; Place all key on pattern into W
01BF 002D 00646  MOVWF CLK_MIN_HD ; Use CLK_MIN_HD for keystuck ON test
01CO 006F 00647  CLRF CLK_HOUR_HD ; Use CLK_HOUR_HD for keystuck OFF test
Current segment display count -> W
Look-up the next segment pattern to display
Place current TMRO value into W
Lets see which bits have changed...
All changed bits are placed in temp for test
Update Previous TMRO value.
It is not time to move to next digit - go back
Move to the next display pattern
Skip if we have timed out waiting for button
Jump to check for the next button press
Clear out timer
Send out a buzzer beep!
Skip if we have NOT tested the last digit
Select the next digit through a rotate..
Place next digit select into W
Jump to the end after last digit tested
Clear out timer
Send out a buzzer beep!
Jump to check for the next button press
Scan the keys to see what is pressed...
Place pattern into W
Make sure keys are not stuck ON
Make sure each key is pressed at least once
Set flag bit to indicate we are done!
Place 8 into W
SKIP if we are NOT at the last digit
Place 8 into W
Jump to main loop
Restore temp registers to zero
Restore temp registers to zero
Jump to main loop
Beep the buzzer constantly for a few secs
DECFSZ CLK_MIN_LD,F ; Decrement counter and skip when done
GOTO loop_bad_sw ; Not done buzzing - go back and do it again
GOTO mfg_cleanup ; Done buzzing - clean-up and run clock

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0040 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0080 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00C0 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0100 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0140 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0180 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
01C0 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
01E0 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0220 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0260 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0280 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
02A0 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
02B0 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
02C0 : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

All other memory blocks unused.

Errors : 0
Warnings : 0
Messages : 0
INTRODUCTION

The use of general purpose microcontrollers for low-end digital signal processing applications has become more commonplace these days with the availability of higher speed processors. Since most signal processing systems consist of a host processor and dedicated DSP chip, the use of a single microcontroller to perform both these functions provides a simpler and lower cost solution. In addition, the single chip design will consume less power which is ideal for battery powered applications. The PIC16C74 with its on-chip A/D, PWM module, and fast CPU is an ideal candidate for use in these low-bandwidth signal processing applications.

A typical signal processing system includes an A/D converter, D/A converter, and CPU that performs the signal processing algorithm as shown in Figure 1.

The input signal, $x(t)$, is first passed through an input filter (commonly called the anti-aliasing filter) whose function is to bandlimit the signal to below the Nyquist rate (one half the sampling frequency) to prevent aliasing. The signal is then digitized by the A/D converter at a rate determined by the sample clock to produce $x(n)$, the discrete-time input sequence. The system transfer function, $H(z)$, is typically implemented in the time-domain using a difference equation. The output sample, $y(n)$, is then converted back into the continuous-time signal, $y(t)$, by the D/A converter and output low-pass filter.

The calculation of the output signal using a difference equation requires a multiply and accumulate (MAC) operation. This is typically a single-cycle instruction on DSP chips but can take many cycles to perform on a standard microcontroller since it must be implemented in code. Since the digitization of the signal, calculation of the output, and output to the D/A converter all must be completed within the sample clock period, the speed at which this can be done determines the maximum bandwidth that can be achieved with the system. The relatively slow speed of most microcontrollers is the major limitation when they are used in DSP applications but the PIC16C74's fast instruction execution speed (as fast as 200 ns/instruction) can provide the performance required to implement relatively low bandwidth systems. In addition, the device's on-chip A/D and PWM modules provide all the functions needed for a single chip system. Only a few external components are needed to use the PIC16C74 for tone generation, filtering of transducer signals, or low bandwidth control.

This application note describes the basic issues that need to be addressed in order to implement digital signal processing systems using the PIC16C74 and provides application code modules and examples for DTMF tone generation, a 60 Hz notch filter, and a simple PID compensator for control systems. These routines can also be used with other PIC16C6X and PIC16C7X processors with minor modifications and the addition of external analog I/O devices.

FIGURE 1: TYPICAL SIGNAL PROCESSING SYSTEM
CODE DEVELOPMENT TOOLS

The code for these applications was written using Byte Craft's MPC C compiler. The MPC compiler provides an Integrated Development Environment (IDE) and generates highly optimized code for the entire PIC16CXX/17CXX family. For new PIC16CXX/17CXX users that are familiar with C, this is an ideal way to quickly develop code for these processors. In addition, the listing files can be studied in order to learn the details of PIC16CXX/17CXX assembly language. The modules and examples for this application note use C for the main program body and in-line assembly language for the time-critical routines. MPC provides interrupt support so that interrupt service routines (ISRs) can be easily written in either C or assembly. This feature was used to provide a timer ISR for one of the code modules. The compiler proved to be a valuable tool that allowed both high level and assembly language routines to be written and tested quickly.

In order to provide the double precision math functions required for this application note, a couple of existing math functions written for the PIC16C54 (see AN525) were converted for use with MPC. The double precision multiply and addition routines were modified by first changing all RAM declarations done in EQU statements to C "unsigned char" variable declarations. The main body of assembly language code was preceded by "#asm" and ended by "#endasm" preprocessor directives which tell the compiler where the in-line assembly code starts and ends. Finally, any macro sections and register names that are defined differently in MPC were changed.

The assembly language routines for tone generation and filtering were also written as C functions using the compiler. Assembly language routines written in this way can be called directly from other assembly language modules or called directly from C by using the label name as a C function. Source listings for all the modules and example programs can be found in the appendices at the end of this application note. These modules can be directly compiled using the MPC compiler or, alternatively, the assembly language sections can be used with MPASM with minor modifications.

Number Representation and Math Routines

One of the challenges of using any general purpose microcontroller for signal processing algorithms is in implementing the finite word-length arithmetic required to perform the calculations. As mentioned before, the speed at which the MAC operations can be performed limits the maximum achievable bandwidth of the system. Therefore, the routines that perform the multiplication and the main signal processing algorithms need to be optimized for speed in order to obtain the highest possible bandwidth when using the PIC16C74.

The selection of word size and coefficient scaling are also important factors in the successful implementation of signal processing systems. The effects of using a fixed word length to represent the signal and do calculations fall into three categories: signal quantization, round-off error, and coefficient quantization. The signal quantization due to the A/D converter and round-off error due to the finite precision arithmetic affect the overall signal-to-noise performance of the system. Scaling of the input signal should be done before the A/D converter to use the full input range and maximize the input signal-to-noise ratio. The use of double precision math for all calculations and storing intermediate results, even if the input and output signals are represented as 8-bit words, will help to reduce the round-off error noise to acceptable levels. Coefficient quantization occurs when the calculated coefficients are truncated or rounded off to fit within the given word length. This has the effect of moving the system transfer function poles and zeros which can change the system gain, critical frequencies of filters, or stability of the system. The successful implementation of these systems requires careful design and modeling of these effects using one of the many software programs that are available. The code written for this application note was first modeled using PC MATLAB before being implemented on the PIC16C74.

The algorithms in this application note are all implemented using fixed point two's compliment arithmetic. Two math libraries were used for the examples: one 8-bit signed multiply routine that was written specifically for the tone generation algorithm, and the modified double precision routines for the PIC16C54 that were used in the filtering routine. All numbers are stored in fractional two's compliment format where the MSB is the sign bit and there is an implied decimal point right after it. This is commonly referred to as Qx format where the number after the Q represents the number of fractional bits in the word. For instance, 16 bit words with the decimal point after the MSB would be referred to as Q15. This format allows numbers over the range of -1 to 0.99 to be represented and, because the magnitude of all numbers is less than or equal to one, has the advantage that there can be no overflow from a multiplication operation.

Since calculations are done using two's compliment arithmetic, values read by the PIC16C74's A/D converter need to be converted to this format. This can be easily done if the input is set up to read values in offset binary format. In this representation, the most negative input voltage is assigned to the number 0, zero volts is assigned the number 128, and the most positive voltage input is assigned 255. Since the PIC16C74 has a unipolar input A/D converter, a bipolar input signal must be scaled to be between 0 and SV. One way to accomplish this is to use an op-amp scaling and offset circuit. The signal should be centered at 2.5V and have a peak to peak voltage swing of
4 to 4.5V. The offset binary number can be converted to two's complimentary format by simply complementing the MSB of the word. Once the signal processing calculations are completed, the number can be converted back to offset binary by complementing the MSB before it is written to the PWM module. A similar level shifting circuit can be used at the PWM output to restore the DC level of the signal. Using this technique allows a wide range of analog input voltages to be handled by the PIC16C74.

A/D and D/A Conversion

The PIC16C74's internal 8-bit A/D converter and PWM modules can be used to implement analog I/O for the system. The A/D converter along with an external anti-aliasing filter provides the analog input for the system. Depending on the input signal bandwidth and the sampling frequency, the filter can be a simple single pole RC filter or a multiple pole active filter. The PWM output along with an external output "smoothing" filter provides the D/A output for the system. This can be a simple RC filter if the PWM frequency is much higher (five to ten times) than the analog signal that is being output. Alternatively, an active filter can also be used at the PWM output. Since the use of the A/D and PWM modules is covered in detail in the data sheet for the PIC16C71 and the PWM module is the same as the PIC16C74, the use of these is also covered in application notes AN546, AN538, and AN539.

Appendix A contains the listing for the C module "ANALOGIO.C" that has the functions that read the A/D converter input, initialize the PWM module, and write 8-bit values to the PWM module. The number format (offset binary or two's complimentary) for the A/D and PWM values as well as the PWM resolution and mode are set using "#define" pragmas at the beginning of the module. The get_sample() function takes the A/D input multiplexor channel number as an argument and returns the measured input value. The init_PWM() function takes the PWM period register PR2 value as an argument. The write_PWM() function takes the output values for PWM module 1 and 2 and writes them to the appropriate registers using the specified resolution. If the second argument to the function is 0, the registers for PWM module 2 are unaffected. The PWM resolution is always 8-bits but the mode used depends on the PWM frequency.

The A/D conversions need to be performed at the system sample rate which requires that some form of sample clock be generated internally or input from an external source. One way to generate this clock internally, in software with minimal effort, is to use the Timer2 interrupt. Since Timer2 is used to generate the PWM period, enabling the Timer2 interrupt and using the Timer2 postscaler can generate an interrupt at periods that are integer divisors of the PWM period. The ISR can set a software "sample flag" that is checked by the main routine. Once the sample flag is asserted by the ISR, the main routine can then clear it and perform the signal processing operation, output the next sample, and then wait for the sample flag to be asserted true again. Alternatively, a separate timer/counter or external clock input can be used for the system sample clock. The latter two methods have the advantage that the PWM frequency can be set independent of the sampling period. For best results, the PWM frequency should be set for at least five times the maximum frequency of the analog signal that is being reproduced. The example programs illustrate the use of both of the methods for generating an internal sample clock.

Tone Generation

For systems that need to provide audible feedback or to provide DTMF signaling for telecom applications, the PIC16C74's PWM module can be used to generate these signals. One way to do this is to output samples of a sinusoidal waveform to the PWM module at the system sampling rate. This method is relatively simple but is limited to single tones and may require large amounts of memory depending on the number of samples used per cycle of the waveform and the number of tones that need to be generated. A more efficient method of generating both single and dual-tone signals is to use a difference equation method. This method uses a difference equation that is derived from the z-transform of a sinusoid as follows:

The z-transform of a sinusoid is

$$Y(z) = \frac{z^{-1}\sin\omega T}{1 - 2z^{-1}\cos\omega T + z^{-2}}$$

where the period $\omega = 2\pi f$ and $T$ is the sampling period.

If this is interpreted as the transfer function $H(z) = Y(z)/X(z)$ then the difference equation can be found taking the inverse z-transform and applying the associated shift theorem as follows:

rearranging:

$$Y(z)(1 - 2z^{-1}\cos\omega T + z^{-2}) = X(z)(z^{-1}\sin\omega T)$$

$$Y(z) = z^{-1}X(z)\sin\omega T + z^{-1}Y(z)2\cos\omega T - z^{2}Y(z)$$

taking the inverse z-transform:

$$Z^{-1}[Y(z)] = Z^{-1}[z^{-1}X(z)\sin\omega T + z^{-1}Y(z)2\cos\omega T - z^{2}Y(z)]$$

$$y(n) = \sin\omega T x(n - 1) + 2\cos\omega T y(n - 1) - y(n - 2)$$

If we let $a = \sin\omega T$ and $b = \cos\omega T$, the equation can be written as:

$$y(n) = \alpha x(n - 1) + \beta y(n - 1) - y(n - 2)$$
thus we have a difference equation with coefficients $a$ and $b$. Note that only two coefficients are needed to generate a sinusoidal output sequence. These are calculated from the relationship above and stored in memory for use by the tone generation algorithm.

If we input an impulse to this system ($x(n) = 1$ at $n = 0$ and is zero elsewhere) then the output of the system will be a discrete-time sinusoidal sequence. Note that at $n = 0$, the output will always be 0 and $x(n)$ is only 1 at $n = 1$ so the sequence becomes:

$$
y(0) = 0
$$
$$
y(1) = a
$$
$$
y(n) = 2b y(n - 1) - y(n - 2)
$$
for $n$ equal to or greater than 2.

In order to further simplify the implementation of the algorithm, we can omit the first sample period. Since the output is already 0 before starting, this will make no difference in the final output other than the fact that it will be time shifted by one sample. To generate dual tones, the algorithm is executed once for each tone and the two output samples are summed together. Since the output must be calculated and output to the D/A each sample period, a limitation exists on the frequency of the tone that can be produced for a given sample rate and processor speed. The higher the ratio of the sample clock to the tone frequency, the better, but a sample rate of at least three to four times the highest tone output should produce a sine wave with acceptable distortion.

Appendix B contains the listing for the "PICTONE.C" module which uses the difference equation method to produce variable length tones from the PWM module. Timer2 is used to generate the PWM period as well as the sample clock and tone duration timer. To send a tone, the coefficients and duration are written to the appropriate variables and then the tone routine is called. If the $a2$ and $b2$ coefficients are cleared, the routine will only generate a single tone sequence. The difference equation algorithm uses 8-bit signed math routines for the multiply operations. Using 8-bit coefficients reduces the accuracy by which the tones can be generated but greatly reduces the number of processor cycles needed to perform the algorithm since only single precision arithmetic is used. The spectrum of a single tone signal generated using this routine is shown in Figure 2.

Note that the second harmonic is better than 40 dB below the fundamental. Accuracy of this particular tone is better than 0.5%.

An example program "DTMFGEN.C" illustrates the use of the tone module to generate the 16 standard DTMF tones used for dialing on the telephone system. A sampling rate of 6.5 kHz was used which allows dual tones to be generated on a processor running at 10 MHz. Accuracy with respect to the standard DTMF frequencies is better than 1% for all tones and all harmonics above the fundamental frequency are greater than 30 dB down.

**FIGURE 2: SINGLE TONE SIGNAL**
Digital Filters

Digital filters with critical frequencies up to a kilohertz or so can be implemented on the PIC16C74. Digital filters fall into two classes: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. FIR filters require more coefficients and multiplication operations to implement practical filters and are not as well suited for implementation on the PIC16C74. IIR type filters are typically designed starting with an analog filter prototype and then performing an analog to digital transformation to produce the digital filter coefficients. The subject of digital filter design is not within the scope of this application note but there are many excellent texts that cover the theory and design of these filters.

The implementation of a second-order IIR filter is done by using a second-order difference equation. A second-order infinite impulse response (IIR) filter has a transfer function of the form:

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \]

Where \( a_1, a_2, b_0, b_1, \) and \( b_2 \) are the coefficients of the polynomials of the system transfer function that, when factored, yield the system poles and zeros. The difference equation found by taking the inverse z-transform and applying the shift theorem is:

\[ y(n) = b_0 x(n) + b_1 x(n - 1) + b_2 x(n - 2) - a_1 y(n - 1) - a_2 y(n - 2) \]

Since the transfer function coefficients are used directly in the difference equation, this is often called the "Direct Form I" implementation of a digital filter. This form has its limitations due to numerical accuracy issues but is effective for implementing second-order systems.

Appendix C contains the listing for the general-purpose filter routine "IIR_FILT.C" that can be used to implement low-pass, high-pass, bandpass, and bandstop (notch) filters. The filter() function takes an 8-bit input value \( x(n) \) and calculates the output value \( y(n) \). The filter coefficients are stored as 16-bit two’s compliment numbers and computation of the output is done using double precision arithmetic. Since the coefficients generated from the filter design program will be in decimal form, they need to be scaled to be less than 1 and then multiplied by 32,768 to put them in Q15 format. Additional scaling by factors of two may be required to prevent overflow of the sum during calculations. If this is done, the output must be multiplied by this scale factor to account for this. The "IIR_FILT.C" module contains two other subroutines required for the filtering program. One if these is a decimal adjust subroutine to restore the decimal place after two 16-bit Q15 numbers are multiplied. The subroutine shifts the 32-bit result left by one to get rid of the extra sign bit.

The other routine scales the output by factors of two and is used after the output of the filter has been calculated to account for the scaling of the coefficients.

An example program "NOTCH_60.C" is provided that illustrates the implementation of a 60 Hz notch filter using the "IIR_FILT.C" module. The filter was modeled and designed using PC MATLAB before being implemented on the PIC16C74. A sample rate of 1 kHz is used which means that signals up to a few hundred hertz can be processed. The filter provides an attenuation of about 40 dB at 60 Hz and can be used to remove interference from sensor signals in a system.

Digital Control

A low bandwidth digital control system can be implemented on the PIC16C74 using the analog I/O and IIR filter routines. A typical digital control system is shown below:

**FIGURE 3: TYPICAL DIGITAL CONTROL SYSTEM**

The input, \( r \), is the reference input and \( y(t) \) is the continuous-time output of the system. \( G(s) \) is the analog transfer function of the plant (controlled system) and \( K(z) \) is the digital compensator. The error signal is calculated by subtracting the measured output signal, \( y(n) \), from the reference. The controller transfer function is essentially a filter that is implemented in the time-domain using a difference equation. Since digital control system design is a complex subject and the design of a suitable compensator depends on the system being controlled and the performance specifications, only the implementation issues will be discussed.
One popular and well understood compensator is the Proportional-Integral-Derivative (PID) controller whose transfer function is of the form:

\[ K(z) = K_p + \frac{K_i}{1 - z^{-1}} + K_d(1 - z^{-1}) \]

Where \( K_p \) is the proportional gain, \( K_i \) is the integral gain, and \( K_d \) is the derivative gain. The transfer function can be implemented directly or can be put in the form of a standard second-order difference equation from the modified transfer function as shown below:

\[
H(z) = \frac{(K_i T^2 + K_p T + K_d) - (2K_d + K_p T)z^{-1} + K_d z^{-2}}{T(1 - z^{-1})}
\]

\[
y(n) = \left( K_p + K_i T + \frac{K_d}{T} \right) x(n) - \left( K_p + \frac{2K_d}{T} \right) x(n - 1) + \frac{K_d}{T} x(n - 2) - y(n - 1)
\]

Since the numerator coefficients will be greater than one, a gain factor \( K \) needs to be factored out so that the resulting coefficients are less than one. In this way, the IIR filter routine can be used to implement the controller. After the filter routine, the output \( y \) needs to be multiplied by \( K \) before being output to the PWM module. Since the gain can be high, this result needs to be checked for overflow and limited to the maximum 8-bit value, if required. Saturating the final result prevents the system from going unstable if overflow in the math does occur. The gains can also be applied externally at the D/A output. For example, the PWM can drive a power op-amp driver that provides a +/- 20 volt swing for a DC motor.

**RESULTS AND CONCLUSION**

The results obtained using the PIC16C74 in these applications were impressive. The tone generation routines produce very clean sinusoidal signals and DTMF tones generated using this routine have been used to dial numbers over the telephone system with excellent results. In addition, tones used for audible feedback are more pleasing to the ear than those generated from a port pin as is typically done on processors without PWM modules. Using the PIC16C74 to generate these tones eliminates the need for special DTMF generator IC's thus reducing the cost and simplifying the design. The tone routine requires approximately 125 instruction cycles to calculate an output sample for a single tone output and 230 instruction cycles to calculate an output sample for a dual tone output.

The IIR filtering routines produce good results and have been used to filter 60 Hz signals on sensor lines and also to implement a simple PID controller system with excellent results. The IIR routine takes approximately 1670 instruction cycles to calculate the output. Table 1 shows the performance that can be expected with the PIC16C74 for various processor speeds.

In conclusion, the PIC16C74 provides the necessary performance to provide these simple, low bandwidth signal processing operations. This means that products using this device can benefit from cost and power savings by eliminating specialized components that normally perform these tasks.

**References**


**TABLE 1: PIC16C74 IIR FILTER PERFORMANCE**

<table>
<thead>
<tr>
<th></th>
<th>4 MHz</th>
<th>8 MHz</th>
<th>10 MHz</th>
<th>16 MHz</th>
<th>20 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/D Input (35 cycles + 15 ms)</td>
<td>50 µs</td>
<td>32.5</td>
<td>29</td>
<td>23.75</td>
<td>22</td>
</tr>
<tr>
<td>IIR Filter (1850 cycles)</td>
<td>1850</td>
<td>925</td>
<td>740</td>
<td>462.5</td>
<td>370</td>
</tr>
<tr>
<td>PWM Output (62 cycles)</td>
<td>62</td>
<td>31</td>
<td>24.8</td>
<td>15.5</td>
<td>12.4</td>
</tr>
<tr>
<td>Total</td>
<td>1962</td>
<td>988.5</td>
<td>793.8</td>
<td>501.75</td>
<td>368.4</td>
</tr>
<tr>
<td>Max. Sampling Frequency</td>
<td>~500 Hz</td>
<td>~1000 Hz</td>
<td>~1250 Hz</td>
<td>~2000 Hz</td>
<td>~2500 Hz</td>
</tr>
</tbody>
</table>
Signal In
(DC Coupled)

Input Level Adjust

Signal Output
(DC Coupled)

+V (12 VDC in)

<table>
<thead>
<tr>
<th>10 µF</th>
</tr>
</thead>
<tbody>
<tr>
<td>C9</td>
</tr>
</tbody>
</table>

VR1

100k

+5V Input Offset Adjust

10k

R10

20k

Output Offset Adjust

10k

VR3

CA (see table)

Cb (see table)

Cc (see table)

LM324

3 Pole Chebyshev Filter - 1 dB Passband Ripple

250 Hz 3.4 kHz

Capacitor Values for 250 Hz and 3.4 kHz

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Ca</th>
<th>Cb</th>
<th>Cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>250 Hz</td>
<td>1 µF</td>
<td>0.15 µF</td>
<td>0.0039 µF</td>
</tr>
<tr>
<td>3.4 kHz</td>
<td>0.082 µF</td>
<td>0.012 µF</td>
<td>300 pF</td>
</tr>
</tbody>
</table>

LMC7660IN

PICDEM2 Board
(See Figure 5)

+V (12 VDC in)
Figure 5: Printed Schematic, IN

Notes:
- Unless otherwise specified, resistor values are in kΩ and capacitor values are in μF.
- For clarity, some components may be simplified or omitted from the schematic.
- The given values are approximate and may need to be adjusted for specific applications.

AN616
APPENDIX A: ANALOG I/O MODULE

/****************************************************************************
* Analog I/O Module
* Written for *Digital Signal Processing with the PIC16C74* Application Note
* This module contains functions that read the A-D inputs, initialize the PWM
* ports, and write values to the PWM ports.
* D. Mostowfi 4/95
****************************************************************************/

#define active 1 /* define active as 1 */
#define LOW 0 /* define LOW as 0 */
#define HIGH 1 /* define HIGH as 1 */

#define OFFSET 0 /* define offset binary mode as 0 */
#define TWOS 1 /* define two's compliment mode as 1 */
#define AD_FORMAT TWOS /* define A-D format as TWOS */
#define PWM_FORMAT TWOS /* define PWM format as TWOS */
#define PWM_RES HIGH /* define PWM resolution as HIGH */

bits FLAGS; /* general purpose flags */
#define sample_flag FLAGS.1 /* define sample_flag as FLAGS.1 */

/****************************************************************************
* A-D Converter Routine - reads A-D converter inputs
* usage:
*   - call get_sample(channel #)
*   - returns 8 bit value
****************************************************************************/

char get_sample(char channel)
{
    char i;
    ADRES=0; /* clear ADRES */
    STATUS.C=0; /* clear carry */
    RLCF(channel); /* and rotate channel 3 times */
    RLCF(channel); /* to put in proper position */
    RLCF(channel); /* for write to ADCONO */
    ADCONO=channel; /* write channel to ADCONO */
    ADCONO.0=1; /* turn on A-D */
    i=0; /* set delay loop variable to 0 */
    while(i++<=5){ /* delay (to ensure min sampling time) */
        ADCONO.2=1; /* start conversion */
        while(ADCONO.2){ /* wait for eoc */
            ADCONO.0=0; /* turn off a-d converter */
        }
        if(AD_FORMAT==TWOS){ /* if format is two's compliment */
            ADRES.7=!ADRES.7; /* compliment MSB */
        }
    }
    return ADRES; /* return value in a-d result reg */
}

/****************************************************************************
* PWM Initialization Routine - sets up PR2, sets output to mid-point, and
* starts timer 2 with interrupts disabled.
* usage:
*   - call init_PWM(PR2 register value)
****************************************************************************/
void init_PWM(char _pr2)
{
    PR2=_pr2; /* reload value for 40KHz PWM period */
    CCP1CON.5=0; /* set CCP1CON = 0 for 50% output */
    CCP1CON.4=0;
    CCP2CON.5=0;
    CCP2CON.4=0;
    if(PWM_RES==HIGH){ /* if resolution is high, set CCPRxH=0 and */
        CCPR1H=0x00; /* CCPRxL=0x20 for 50% PWM duty cycle */
        CCPR1L=0x20;
        CCPR2H=0x00;
        CCPR2L=0x20;
    } else{
        CCPR1H=0x00; /* if resolution is low, set CCPRxH=0 and */
        CCPR1L=0x80; /* CCPRxL=0x80 for 50% PWM duty cycle */
        CCPR2H=0x00;
        CCPR2L=0x80;
    }
    T2CON.TMR2ON=1; /* start timer 2 */
    PIE1.TMR2IE=0; /* and disable timer 2 interrupt */
}

/******************************************************************************
* PWM Output Routine - writes output values to PWM ports
* Both high resolution and low resolution modes write 8 bit values - use of *
* high or low resolution depends on PWM output period.
* usage:
* - call write_PWM(channel 1 value, channel 2 value)
*   if channel 2 value=0, PWM port 2 not written
*******************************************************************************/
void write_PWM(bits pwm_out1, bits pwm_out2)
{
    if(PWM_FORMAT==TWOS){ /* if format is two’s compliment */
        pwm_out1.7=!pwm_out1.7; /* compliment msb’s */
        pwm_out2.7=!pwm_out2.7;
    }
    if(PWM_RES==HIGH){ /* if resolution is high */
        STATUS.C=0; /* clear carry */
        pwm_out1=RRCF(pwm_out1); /* rotate right and write two lsb’s */
        CCP1CON.4=STATUS.C; /* to CCP1CON4 and CCP1CON5 */
        STATUS.C=0;
        pwm_out1=RRCF(pwm_out1);
        CCP1CON.5=STATUS.C;
        if(pwm_out2!=0){ /* if pwm_out2 not 0, do the same */
            STATUS.C=0; /* for channel 2 */
            pwm_out2=RRCF(pwm_out2);
            CCP2CON.4=STATUS.C;
            STATUS.C=0;
            pwm_out2=RRCF(pwm_out2);
            CCP2CON.5=STATUS.C;
        }
    }
    CCP1L=pwm_out1; /* write value to CCP1L */
    if(pwm_out2!=0){ /* if pwm_out2 not 0, do the same */
        CCP2L=pwm_out2; /* for CCP2L */
    }
}/* done */
APPENDIX B: TONE GENERATION MODULE

/***************************************************************************/
* Tone Generation Module
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
* This module contains a C callable module that generates single or dual tones using a difference equation method:
* \[ y_1(n) = a_1 \cdot x(n-1) + b_1 \cdot y_1(n-1) - y_1(n-2) \]
* \[ y_2(n) = a_2 \cdot x(n-1) + b_2 \cdot y_2(n-1) - y_2(n-2) \]
* The routine is written in assembly language and uses the optimized signed 8x8 multiply routine and scaling routine in the file 8BITMATH.C.
* D. Mostowfj 2/95
***************************************************************************/
#include "mpc\apnotes\8bitmath.c" /* 8 bit signed math routines */
#define sample_flag FLAGS.l /* sample flag */
#define no_tone2 FLAGS,2 /* no tone 2 flag */
extern char ms_cntr; /* millisecond counter for tone loop */
char al; /* first tone (low-group) coefficient 1 */
char a2; /* first tone (low-group) coefficient 2 */
char bl; /* second tone (high group) coefficient 1 */
char b2; /* second tone (high group) coefficient 2 */
char duration; /* tone duration */
char yl; /* output sample y1(n) for tone 1 */
char y2; /* output sample y2(n) for tone 2 */

/***************************************************************************/
* Tone function - generates single or dual tone signals out PWM port 1.
* usage:
* - write coefficients for tone 1 to al and bl
* - write coefficients for tone 2 to a2 and b2 (0 if no tone 2)
* - write duration of tone in milliseconds to duration
* - call tone() function
/***************************************************************************/
void tone(void)
{
char y1_l; /* y1(n-1) */
char y1_2; /* y1(n-2) */
char y2_l; /* y2(n-1) */
char y2_2; /* y2(n-2) */
P1IF = 0; /* clear timer 2 interrupt flag */
P1IE = 1; /* and enable timer 2 interrupt */
ms_cntr = 0; /* clear ms counter */
STATUS.RPO = 0; /* set proper bank!!! */
asm
    clrf y1 ; clear output byte and taps
    clrf y2 ;
    clrf y1_l ;
    clrf y1_2 ;
    clrf y2_l ;

© 1995 Microchip Technology Inc.
 inaugural sample:

```assembly
clrf y2_2 ;
bcf no_tone2 ; clear no tone 2 flag
clrf ms_cntr ; clear millisecond counter

first_sample:
movf a1,W ; first iteration
movwf y1 ; y1(n)=a1
movwf y1_1 ;
movlw 0x00 ;
ierwf a2,W ;
btfsc STATUS,Z ; generate second tone (a2 !=0) ?
bcf no_tone2 ;
movf a2,W ; y2(n)=a2
movwf y2 ;
movwf y2_1 ;
movf y2,W ;
addwf y1,F ; y1(n)=y1(n)+y2(n) (sum two tone outputs)

tone_loop:
movf ms_cntr,W ; test to see if ms=duration (done?)
subwf duration,W ;
btfsc STATUS,Z ;
goto tone_done ;

wait_PWM:
btfss FLAGS,1 ; test sample flag (sample period elapsed?)
goto wait_PWM ; loop if not

bcf FLAGS,1 ; if set, clear sample flag

#endasm

write_PWM((char)yl,0); /* write yl to PWM port */

#asm

next_sample:
movf b1,W ; y1(n)=b1*y1(n-1)-y1(n-2)
movwf multcnd ;
movf y1_1,W ;
movwf multplr ;
call _8x8smul ;
call scale_16 ;
movf y1_2,W ;
subwf result_1,W ;
movf y1 ;
movf y1_1,W ; y1(n-2)=y1(n-1)
movwf y1_2 ;
movf y1,W ; y1(n-1)=y1(n)
movwf y1_1 ;
btfsc no_tone2 ;
goto tone_loop ;
movf b2,W ; y2(n)=b2*y2(n-1)-y2(n-2)
movwf multcnd ;
movf y2_1,W ;
movwf multplr ;
call _8x8smul ;
call scale_16 ;
movf y2_2,W ;
subwf result_1,W ;
movf y2 ;
movf y2_1,W ; y2(n-2)=y2(n-1)
movf y2_2 ;
movf y2,W ; y2(n-1)=y2(n)
movf y2_1 ;
movf y2,W ;
```

© 1995 Microchip Technology Inc.
addwf yl,F ; y1(n)=y1(n)+y2(n) (sum two tone outputs)
goto tone_loop ; go and calculate next sample
tone_done:
#endasm

CCPlCON.5=0;       /* reset PWM outputs to mid value */
CCPlCON.4=0;
CCP2CON.5=0;
CCP2CON.4=0;
CCPR1H=0x00;
CCPR1L=0x20;
CCPR2H=0x00;
CCPR2L=0x20;

PIR1.TMR2IE=0;     /* disable timer 2 interrupts */
PIR1.TMR2IF=0;     /* and clear timer 2 interrupt flag */
APPENDIX C: DTMF TONE GENERATION

*****************************************************************************
* DTMF tone generation using PIC16C74
* Written for the "Digital Signal Processing Using the PIC16C74" Ap Note
* Generates 16 DTMF tones (1-9,0,#,A,B,C,D) out PWM port 1
* Uses PICTONE.C and ANALOGIO.C modules
* D. Mostowfi 4/95
******************************************************************************/
#include "\mpc\include\delay14.h"
#include "\mpc\include\16c74.h" /* c74 header file */
#include "\mpc\math.h"
#include "\mpc\apnotes\analogio.c" /* analog I/O module */
#include "\mpc\apnotes\pictone.c" /* tone generation module */

bits pwml;

/* Function Prototypes */
void main_isr();
void timer2_isr();

/* 16C74 I/O port bit declarations */

/* global program variables */
char tmr2_cntr; /* timer 2 interrupt counter */
char delay_cntr; /* delay time counter (10ms ticks)*/

/* Tone Coefficients for DTMF Tones */
const DTMF_5[4] = {33, 48, 56, 19};
const DTMF_6[4] = {33, 48, 64, 11};
const DTMF_7[4] = {36, 45, 48, 27};
const DTMF_8[4] = {36, 45, 56, 19};
const DTMF_9[4] = {36, 45, 64, 11};
const DTMF_0[4] = {40, 41, 56, 19};
const DTMF_pound[4] = {40, 41, 64, 11};
const DTMF_B[4] = {33, 48, 75, 1};
const DTMF_C[4] = {36, 45, 75, 2};
const DTMF_D[4] = {40, 41, 75, 2};

/*****************************************************************************
* main_isr - 16C74 vectors to 0004h (MPC __INT() function) on any interrupt *
* assembly language routine saves W and Status registers then tests flags in *
* INTCON to determine source of interrupt. Routine then calls appropriate isr. *
* Restores W and status registers when done.
******************************************************************************/
void __INT(void)
{

    if(PIR1.TMR2IF){ /* timer 2 interrupt? */
        PIR1.TMR2IF=0; /* clear interrupt flag */
        timer2_isr(); /* and call timer 2 isr */
    }

    /* Restore W, Wimage, and STATUS registers */
    #asm
    BCF STATUS,RPO ;Bank 0
    MOVF temp_PCLATH, W
    MOVWF PCLATH ;PCLATH restored
    MOVF temp_Wimage, W
    MOVWF _Wimage ;_Wimage restored
    MOVF temp_FSR, W
    MOVWF PSR ;FSR restored
    SWAPF temp_STATUS,W
    SWAPF temp_STATUS,F
    SWAPF temp_WREG,F
    SWAPF temp_WREG,W
    ;Bank 0
    ;PCLATH restored
    ;_Wimage restored
    ;FSR restored
    ;RPO restored
    ;W restored
    #endasm

    /* timer 2 isr - provides PWM sample clock generation and millisecond counter */
    /* for tone routine */
    void timer2_isr(void)
    {
        sample_flag=active; /* set sample flag (150us clock) */
        PORTB.7=!PORTB.7; /* toggle PORTB.7 at sample rate */
        if(tmr2_cntr++==7){ /* check counter */
            tmr2_cntr=0; /* reset if max */
            ms_cntr++; /* and increment millisecond ticks */
        }
    }

    void main()
    {
        /* initialize OPTION register */
        OPTION=Ob11001111;

        /* initialize INTCON register (keep GIE inactive!) */
        INTCON=Ob00000000; /* disable all interrupts */

        /* initialize PIE1 and PIE2 registers (peripheral interrupts) */
        PIE1=Ob00000000; /* disable all interrupts */
        PIE2=Ob00000000;

        /* initialize T1CON and T2CON registers */
        T1CON=Ob00000000; /* T1 not used */
        T2CON=Ob00101000; /* T2 postscaler=5 */

        /* initialize CCPxCON registers */
        CCP1CON=Ob00000110; /* set CCP1CON for PWM mode */
        CCP2CON=Ob00001110; /* set CCP2CON for PWM mode (not used in demo) */

        /* initialize SSPCON register */
        SSPCON=Ob00000000; /* serial port - not used */

        /* initialize ADCONx registers */
ADCON0=0b00000000;    /* A-D converter */
ADCON1=0b00000010;

/* initialize TRISx register (port pins as inputs or outputs) */
TRISA=0b00001111;
TRISB=0b00000000;
TRISC=0b10000000;
TRISD=0b00001111;
TRISE=0b00000000;

/* clear watchdog timer (not used) */
CLRWDT();

/* initialize program variables */
tmr2_cntr=0;

/* initialize program bit variables */
FLAGS=0b00000000;

/* initialize output port pins (display LED's on demo board) */
PORTB=0;

/* enable interrupts... */

INTCON.ADIE=1;    /* Peripheral interrupt enable */
INTCON.GIE=1;     /* global interrupt enable */

init_PWM(0x3e);    /* initialize PWM port */

PORTB=0x01;       /* write a 1 to PORTB */
a1=DTMF_1[0];    /* and send a DTMF "1" */
b1=DTMF_1[1];
a2=DTMF_1[2];
b2=DTMF_1[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */

PORTB=0x02;       /* write a 2 to PORT B */
a1=DTMF_2[0];    /* and send a DTMF "2" */
b1=DTMF_2[1];
a2=DTMF_2[2];
b2=DTMF_2[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */

PORTB=0x03;       /* write a 3 to PORTB */
a1=DTMF_3[0];    /* and send a DTMF "3" */
b1=DTMF_3[1];
a2=DTMF_3[2];
b2=DTMF_3[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */

PORTB=0x04;       /* write a 4 to PORTB */
a1=DTMF_4[0];    /* and send a DTMF "4" */
b1=DTMF_4[1];
a2=DTMF_4[2];
b2=DTMF_4[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x05;
if (a1=DTMF_5[0]; /* write a 5 to PORTB */
b1=DTMF_5[1]; /* and send a DTMF *5*/
a2=DTMF_5[2];
b2=DTMF_5[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x06;
if (a1=DTMF_6[0]; /* write a 6 to PORTB */
b1=DTMF_6[1]; /* and send a DTMF *6*/
a2=DTMF_6[2];
b2=DTMF_6[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x07;
if (a1=DTMF_7[0]; /* write a 7 to PORTB */
b1=DTMF_7[1]; /* and send a DTMF *7*/
a2=DTMF_7[2];
b2=DTMF_7[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x08;
if (a1=DTMF_8[0]; /* write a 8 to PORTB */
b1=DTMF_8[1]; /* and send a DTMF *8*/
a2=DTMF_8[2];
b2=DTMF_8[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x09;
if (a1=DTMF_9[0]; /* write a 9 to PORTB */
b1=DTMF_9[1]; /* and send a DTMF *9*/
a2=DTMF_9[2];
b2=DTMF_9[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x0;
if (a1=DTMF_0[0]; /* write a 0 to PORTB */
b1=DTMF_0[1]; /* and send a DTMF *0*/
a2=DTMF_0[2];
b2=DTMF_0[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x0e;
if (write a 0x0e to PORTB */
a1=DTMF_star[0];    /* and send a DTMF ** */
a2=DTMF_star[2];
b2=DTMF_star[3];
duration=250;
tone();
Delay_Ms_20MHz(200);    /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0f;
/* write a 0x0f to PORTB */

a1=DTMF_pound[0];    /* and send a DTMF * */
a2=DTMF_pound[2];
b2=DTMF_pound[3];
duration=250;
tone();
Delay_Ms_20MHz(200);    /* delay 100ms (200/2 using MPC delays) */
Delay_Ms_20MHz(200);    /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0a;
/* write a 0x0a to PORTB */

a1=DTMF_A[0];    /* and send a DTMF A */
a2=DTMF_A[2];
b2=DTMF_A[3];
duration=250;
tone();
Delay_Ms_20MHz(200);    /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0b;
/* write a 0x0b to PORTB */

a1=DTMF_B[0];    /* and send a DTMF B */
a2=DTMF_B[2];
b2=DTMF_B[3];
duration=250;
tone();
Delay_Ms_20MHz(200);    /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0c;
/* write a 0x0c to PORTB */

a1=DTMF_C[0];    /* and send a DTMF C */
a2=DTMF_C[2];
b2=DTMF_C[3];
duration=250;
tone();
Delay_Ms_20MHz(200);    /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0d;
/* write a 0x0d to PORTB */

a1=DTMF_D[0];    /* and send a DTMF D */
a2=DTMF_D[2];
b2=DTMF_D[3];
duration=250;
tone();
PORTB=0;
/* write a 0 to PORTB */
while(1){}    /* done (loop) */
APPENDIX D: IIR FILTER MODULE

/******************************************************************************
* Second-Order IIR Filter Module
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
*
* This routine implements an IIR filter using a second order difference
* equation of the form:
*
* y(n) = b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*y(n-2)
*
* D. Mostowfi 3/95
******************************************************************************/

#include "\mpc\apnotes\dbl_math.c"

bits
unsigned long
unsigned long
char
#define A1_H
#define A1_L
#define A2_H
#define A2_L
#define B0_H
#define B0_L
#define B1_H
#define B1_L
#define B2_H
#define B2_L

/******************************************************************************
* Filter initialization - clears all taps in memory.
*
* usage:
* - call init_filter()
* use at program initialization
******************************************************************************/

void init_filter() {

    #asm
        clrf y_n ; clear output value
        clrf y_n+1 ;
        clrf y_n+1 ; and all filter "taps"
        clrf y_n+1 ;
        clrf y_n+2 ;
        clrf x_n ;
        clrf x_n+1 ;
        clrf x_n+2 ;
        clrf x_n+2+1 ;
        clrf x_n+2+1 ;
    #endasm

}
/******************************************************************************
* Assembly language subroutines for main filter() function
******************************************************************************/

#include

; Add Remainder subroutine - adds remainder from multiplies to ACCc

add_r_start:
  movf   rmnd1,W ; get low byte of remainder
  addwf  ACCcLO ; and add to ACCcLO
  btfsc  STATUS,C ; check for overflow
  incf   ACCcHI ; if overflow, increment ACCcHI
  movf   rmnd1_h,W ; get high byte of remainder
  addwf  ACCcHI ; and add to ACCcHI
  btfsc  STATUS,C ; check for overflow
  incf   ACCcLO ; if overflow, increment ACCcLO
  btfss  sign,7 ; check if result negative
  goto   add_r_done ; if not, go to add_r_done
  comf   ACCcLO ; if so, negate result
  incf   ACCcLO ;
  btfsc  STATUS,Z ;
  decf   ACCcHI ;
  comf   ACCcHI ;
  btfsc  STATUS,Z ;
  comf   ACCcLO ;
  incf   ACCcLO ;
  btfsc  STATUS,Z ;
  decf   ACCcHI ;
  comf   ACCcHI ;

add_r_done:
  retlw 0 ; done

; Decimal Adjust Subroutine - used after each Q15 multiply to convert Q30 result
; to Q15 number

dec_adjust:
  bcf    sign,7 ; clear sign
  btfss  ACCcHI,7 ; test if number is negative
  goto   adjust ; go to adjust if not
  bcf    sign,7 ; set sign if negative
  comf   ACCcLO ; and negate number
  incf   ACCcLO
  btfsc  STATUS,Z

© 1995 Microchip Technology Inc.
Output Scaling Routine - used to scale output samples by factors of 2, 4, or 8 at end of filter routine

scale_y_n:
  bcf    sign,7 ; clear sign,7
  btfss y_n+1,7 ; test if y(n) negative
  goto   start_scale ; go to start_scale if not
  bcf    sign,7 ; set sign,7 if negative
  comf   y_n ; and compliment y(n)
  incf   y_n
  btfsc STATUS,Z
  decf   y_n+1
  comf   y_n+1

start_scale:
  bcf    STATUS,C ; clear carry
  rlf    y_n+1 ; and rotate y(n) left
  rlf    y_n
  bcf    STATUS,C
  rlf    y_n+1
  rlf    y_n
  bcf    STATUS,C
  rlf    y_n+1
  rlf    y_n
  btfss sign,7 ; test if result is negative
  goto   scale_y_done ; go to scale_y_done if not
  comf   y_n ; negate y(n) if result is negative
  incf   y_n
  btfsc STATUS,Z
  decf   y_n+1
  comf   y_n+1

scale_y_done:
  retlw 0 ; done
///////////////////////////////////////////
* Filter function - filter takes current input sample, x(n), and outputs next *
* output sample, y(n).
*  
* usage:
*  - write sample to be filtered to x_n
*  - call filter()
*  - output is in MSB of y_n (y_n=MSB, y_n+1=LSB)
*  
******************************************************************************/

void filter(){

#if 0

clr y_n ; clear y(n) before starting
clr y_n+1 ;
        
crflf ACCbLO ; move x(n) to ACCbHI
movf x_n,W ; (scale 8 bit - 16 bit input)
movwf ACCbHI ;

movlw B0_H ; get coefficient b0
movwf ACCaHI ; y(n)=b0*x(n)
movlw B0_L ;
movwf ACCaLO ;
call D_mpyF ;
movf ACCchl,W ; save remainder from multiply
movwf rmndr_h ;
movf ACCcl,w ;
movwf rmndr_l ;
call dec_adjust ;
movf ACCbHI,W ;
movf y_n+1 ;
movf ACCbLO,W ;
movwf y_n ;

movlw B1_H ; get coefficient b1
movwf ACCaHI ; y(n)=y(n)+b1*x(n-1)
movlw B1_L ;
movwf ACCaLO ;
movf x_n_l+1,W ;
movwf ACCbHI ;
movf x_n_l,w ;
movwf ACCbLO ;
call D_mpyF ;
call add_rmndr ; add in remainder from previous multiply
movf ACCchl,W ; and save new remainder
movwf rmndr_h ;
movf ACCcl,w ;
movwf rmndr_l ;
call dec_adjust ;
movf y_n+1,w ;
movwf ACCaHI ;
movf y_n,w ;
movwf ACCaLO ;
call D_add ;
movf ACCbHI,w ;
movwf y_n+l ;
movf ACCbLO,w ;
movwf y_n ;

movlw B2_H ; get coefficient b2
movwf ACCaHI ; y(n)=y(n)+b2*x(n-2)
movlw B2_L ;
movwf ACCaLO ;
movf x_n_2+1,w ;
movwf ACCbHI ;

#endif

}
movf x_n_2, W ;
movwf ACCbLO ;
call D_mpyF ;
call add_rmndr ; ; add in remainder from previous multiply
movf ACCcHI, W ; ; and save new remainder
movf rmndr_h, W
movf ACCcLO, W
movf rmndr_l, W
call dec_adjust ;
movf y_n+1, W
movf ACCaHI ;
movf y_n, W
movf ACCaLO ;
call D_add ;
movf ACCbHI, W
movf y_n+1
movf ACCbLO, W
movwf y_n

movlw A1_H ; ; get coefficient a1
movwf ACCaHI ; ; y(n)=y(n)+a1*y(n-1)
movlw A1_L ;
movwf ACCaLO ;
movf y_n+1+W
movwf ACCbHI ;
movf y_n-1+W
movwf ACCbLO ;
call D_mpyF ;
call add_rmndr ; ; add in remainder from previous multiply
movf ACCcHI, W ; ; and save new remainder
movf rmndr_h, W
movf ACCcLO, W
movf rmndr_l, W
call dec_adjust ;
movf y_n+1, W
movf ACCaHI ;
movf y_n, W
movf ACCaLO ;
call D_sub ;
movf ACCbHI, W
movf y_n+1
movf ACCbLO, W
movwf y_n

movlw A2_H ; ; get coefficient a2
movwf ACCaHI ; ; y(n)=y(n)+a2*y(n-2)
movlw A2_L ;
movwf ACCaLO ;
movf y_n_2+1, W
movwf ACCbHI ;
movf y_n_2, W
movwf ACCbLO ;
call D_mpyF ;
call add_rmndr ;
call dec_adjust ;
movf y_n+1, W
movf ACCaHI ;
movf y_n, W
movf ACCaLO ;
call D_sub ;
movf ACCbHI, W
movf y_n+1
movf ACCbLO, W
movwf y_n

movf x_n_1, W ; ; x(n-2)=x(n-1)
movf x_n_2,
movf x_n_1+1,W
movwf x_n_2+1
movf x_n,W ; x(n-1)=x(n)
movwf x_n_1+1
clrf x_n_1

movf y_n_1,W ; y(n-2)=y(n-1)
movwf y_n_2
movf y_n_1+1,W
movwf y_n_2+1
movf y_n,W ; y(n-1)=y(n)
movwf y_n_1
movf y_n+1,W
movwf y_n_1+1

call scale_y_n

movf y_n+1,W ; shift lsb of y_n to msb
movwf y_n

@endasm
APPENDIX E: NOTCH FILTER

 /*****************************************************************************
 * 60 Hertz Notch Filter
 * 
 * Written for "Digital Signal Processing with the PIC16C74" Application Note.
 * 
 * This example program use the filter() function to implement a 60Hz notch
 * filter. T0 is used to generate a 1kHz sample clock. The program samples the
 * input signal x(n) on A-D channel 1, calls the filter routine signal, and
 * outputs y(n) to PWM channel 1.
 * 
 * If FILTER set to 0, performs straight talkthru from A-D to PWM output.
 * T0 period can be changed to carry the sample rate.
 * 
 * D. Mostowfi 4/95
 ******************************************************************************/

 #include "mpc\include\16c74.h" /* c74 header file */
 #include "mpc\apnotes\analogio.c" /* analog I/O module */
 #include "mpc\apnotes\iir_filt.c" /* iir filter module */

 #define FILTER 1

 /* Function Prototypes */
 void main_isr();
 void timer0_isr();

 /*****************************************************************************
 * main isr - 16C74 vectors to 0004h (MPC __INT() function) on any interrupt *
 * assembly language routine saves W and Status registers then tests flags in
 * INTC ON to determine source of interrupt. Routine then calls appropriate isr.
 * Restores W and status registers when done.
 ******************************************************************************/

 void __INT(void)
 {
   if(INTCON.T0IF) { /* timer 0 interrupt ? */
     INTCON.T0IF=0; /* clear interrupt flag */
     timer0_isr(); /* and call timer 0 isr */
   }

   /* Restore W, WImage, and STATUS registers */

   #asm
   BCF STATUS,RPO ;Bank 0
   MOVF temp_PCLATH, W ;PCLATH restored
   MOVWF PCLATH
   MOVF temp_WImage, W ;WImage restored
   MOVWF _Wimage
   MOVF temp_FSR, W ;FSR restored
   MOVWF FSR
   SWAPF temp_STATUS,W ;RP0 restored
   MOVWF STATUS
   SWAPF temp_MREG,F
   SWAPF temp_MREG,W
   ;W restored
   #endasm

   /* timer 0 interrupt service routine */

 void timer0_isr(void)
 {
TMRO=100; /* reload value for 1ms period */
PORTB.0=!PORTB.0; /* toggle PORTB.0 */
sample_flag=active; /* set sample flag */

void main()
{
    /* initialize OPTION register */
    OPTION=0b00000011; /* assign prescaler to T0 */
    /* initialize INTCON register (keep GIE inactive!) */
    INTCON=0b00000000; /* disable all interrupts */
    /* initialize PIE1 and PIE2 registers (peripheral interrupts) */
    PIE1=0b00000000; /* disable all peripheral interrupts */
    PIE2=0b00000000;
    /* initialize T1CON and T2CON registers */
    T1CON=0b00000000; /* T1 not used */
    T2CON=0b00000000; /* T2 not used */
    /* initialize CCPxCON registers */
    CCP1CON=0b00001100; /* set CCP1CON for PWM mode */
    CCP2CON=0b00000000; /* CCP2CON=0 (PWM 2 not used) */
    /* initialize SSPCON register */
    SSPCON=0b00000000; /* serial port - not used */
    /* initialize ADCONx registers */
    ADCON0=0b00000000; /* a-d converter */
    ADCON1=0b00000010;
    /* initialize TRISx register (port pins as inputs or outputs) */
    TRISA=0b00011111;
    TRISB=0b00000000;
    TRISC=0b11011111;
    TRISD=0b11110111;
    TRISE=0b11111111;
    /* clear watchdog timer (not used) */
    CLRWDT();
    /* initialize program bit variables */
    FLAGS=0b00000000;
    /* initialize output port pins */
    PORTB=0;
    /* enable interrupts... */
    INTCON.TOIE=1; /* peripheral interrupt enable */
    INTCON.GIE=1; /* global interrupt enable */
    init_PWM(0x40); /* init PWM port */
    init_filter(); /* init filter */
    while(1){
        while(!sample_flag){ /* wait for sample clock flag to be set */
            sample_flag=0; /* clear sample clock flag */
            x_n=get_sample(); /* read ADC channel 1 into x(n) */
            if(FILTER==1){ /* if filter enabled */
                filter(); /* call filter routine */
            }
            else{ /* or else write x(n) to y(n) (talkthru) */
                y_n=x_n;
            }
            write_PWM((char)y_n,0); /* write y_n to PWM port */
        }
    }
}
APPENDIX F: 8-BIT MULTIPLY AND SCALING Routines

/**************************************************************************
* 8 bit Multiply and Scaling Routines
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
*
* This module provides a 8 bit signed multiply and scaling routine for the
* PICTONE.C tone generation program. The routines are adapted from "Math
* Routines for the 16C5x" in Microchip's Embedded Controller Handbook.
*
* All numbers are assumed to be signed 2's compliment format.
*
* D. Mostowfi 11/94
**************************************************************************/

char multcnd; /* 8 bit multiplicand */
char multplr; /* 8 bit multiplier */
char result_h; /* result - high byte */
char result_l; /* result - low byte */
char sign; /* result sign */

#asm

; 8x8 signed multiply routine
; called from PICTONE.C module (assembly language routine)

.MACRO mult_core bit
  btfss multplr,bit
  goto \no_add
  movf multcnd,W
  addwf result_h,F
.
\no_add:  
  rrf result_h
  rrf result_l
.
.ENDM

_8x8smul:
  movf multcnd,W ; get multiplicand
  xorwf multplr,W ; and xor with multiplier
  movwf sign ; and save sign of result
  btfss multcnd,7 ; check sign bit of multiplicand
  goto chk_multiplr ; go and check multiplier if positive
  comf multcnd ; negate if negative
  incf multcnd ;

chk_multiplr:
  btfss multplr,7 ; check sign bit of multiplier
  goto multiply ; go to multiply if positive
  comf multplr ; negate if negative
  incf multplr ;

multiply:
  movf multcnd,W ; set up multiply registers
  bcf STATUS,C ;
  clrf result_h ;
  clrf result_l ;
  mult_core 0 ; and do multiply core 8 times
  mult_core 1 ;
  mult_core 2 ;
mult_core 3
mult_core 4
mult_core 5
mult_core 6
mult_core 7

set_sign:
  btfss sign,7 ; test sign to see if result negative
  retlw 0 ; done if not! (clear W)
  comf result_l ; negate result if sign set
  incf result_l ;
  btfsc STATUS,Z ;
  decf result_h ;
  comf result_h ;
  retlw 0 ; done (clear W)

; Scaling Routine (used after a multiply to scale 16 bit result)
; Operates on result_h and result_l - final result is in result_l
; routine divides by 32 to restore Q7 result of 2*b*y(n-1) in tone
; generation algorithm
;
scale_16:
  btfss sign,7 ; test if negative (sign set from mult)
  goto start_shift ; go to start shift if pos.
  comf result_l ; negate first if neg.
  incf result_l ;
  btfsc STATUS,Z ;
  decf result_h ;
  comf result_h ;

start_shift:
  bcf STATUS,C ; clear status
  rrf result_h ; and shift result left 5x (/32)
  rrf result_l ;
  rrf result_l ;
  rrf result_l ;
  rrf result_l ;
  rrf result_l ;
  rrf result_l ;
  rrf result_l ;
  btfss sign,7 ; test if result negative
  goto scale_done ; done if not negative
  comf result_l ; negate result if negative
  incf result_l ;
  btfsc STATUS,Z ;
  decf result_h ;
  comf result_h ;

scale_done:
  retlw 0 ; done (clear W)

#endasm
APPENDIX G: DOUBLE PRECISION MATH ROUTINES

**************************************************************************
* Double Precision Math Routines
* 
* This module contains assembly language routines from "Math Routines for the
* 16C5x" from Microchip's Embedded Controller Handbook that have been adapted
* for use with the Bytecraft MPC C Compiler.
* 
* Routines are used IIR_FILTER.C module written for "Digital Signal Processing
* with the PIC16C74" Application Note.
* 
* D. Mostowfi 3/95
**************************************************************************

/*
Start of converted MPASM modules:

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

char ACCaLO; //equ 10 changed equ statements to C char variables
char ACCaHI; //equ 11
char ACCbLO; //equ 12
char ACCbHI; //equ 13

#include "mpreg.h" commented out these
org 0 two lines (MPASM specific)

;*************************************************************
D_sub call neg_A2 ; At first negate ACCa; Then add
;*************************************************************

DS00616A-page 2-150 © 1995 Microchip Technology Inc.
D_add  movf ACCaLO,W
    addwf ACCbLO     ;add lsb
    btfsc STATUS,C   ;add in carry
    incf ACCbHI
    movf ACCaHI,C
    addwf ACCbHI     ;add msb
    retlw 0

neg_A2  comf ACCaLO ; negate ACCa ( -ACCa -> ACCa )
    incf ACCaLO
    btfsc STATUS,Z
    decf ACCaHI
    comf ACCaHI
    retlw 0

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

; 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 1s.
 ;
 ; The performance specs are for Unsigned arithmetic ( i.e.
 ; with 'SIGNED equ FALSE ').
 ;
;*****************************************************************************;

#include "mpreg.h"
org 0
Conmented out - already defined in Dbl_add
changed equ statements to C char variables
commented out these
two lines (MPASM specific)

SIGNED equ 1 ; 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

.Macro mulMac
LOCAL NO_ADD

rrf ACCbHI ; rotate d right
rrf ACCbLO
btfss STATUS,C ; need to add?
goto \NO_ADD ; no addition necessary
movf ACCaLO,W ; Addition ( ACCb + ACCa -> ACCb )
addwf ACCbLO ; add lsb
btfsc STATUS,C ; add in carry
incf ACCbHI
movf ACCaHI,W
addwf ACCbHI ; add msb
\NO_ADD rrf ACCbHI
rrf ACCbLO
rrf ACCcHI
rrf ACCcLO

.ENDM ; end of modified macro

; 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

call setup

; use the mulMac macro 16 times

mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac

.IF SIGNED
btfss sign,7
retlw 0
comf ACCcLO ; negate ACCa ( -ACCa -> ACCa )
incf ACCcLO
btfsc STATUS,Z
deef ACCcHI
comf ACCcHI
btfsc STATUS,Z
neg_B  komf  ACCbLO  ; negate ACCb
  incf  ACCbLO
  btfsc  STATUS,Z
  decf  ACCbHI
  komf  ACCbHI
  retlw  0
  .ELSE
  retlw  0
  .ENDIF

;*******************************************************************************
; setup movlw i6  ; for i6 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  komf  ACCaLO  ; negate ACCa ( -ACCa -> ACCa )
  incf  ACCaLO
  btfsc  STATUS,Z
  decf  ACCaHI
  komf  ACCaHI
  retlw  0

;*******************************************************************************
; Assemble this section only if Signed Arithmetic Needed
; .IF  SIGNED
; S_SIGN  movf  ACCaHI,W
  xorwf  ACCbHI,W
  movwf  sign
  btfss  ACCbHI,7  ; if MSB set go & negate ACCb
  goto  chek_A
;
  komf  ACCbLO  ; negate ACCb
  incf  ACCbLO
  btfsc  STATUS,Z
  decf  ACCbHI
  komf  ACCbHI
;
chek_A  btfss  ACCaHI,7  ; if MSB set go & negate ACCa
  retlw  0
  goto  neg_A
;
.ENDIF

@endasm
INTRODUCTION

This application note presents an implementation of the following fixed point math routines for the PIC16/17 microcontroller families:

- Multiplication
- Division

Routines for the PIC16/17 families are provided in a variety of fixed point formats, including both unsigned and signed two’s complement arithmetic.

FIXED POINT ARITHMETIC

Unsigned fixed point binary numbers \( A \), can be represented in the form

\[
A = \sum_{k=0}^{n-1} a(k) \cdot 2^{k-r} = 2^{-r} \sum_{k=0}^{n-1} a(k) \cdot 2^k
\]

where \( n \) is the number of bits, \( a(k) \) is the \( k \)th bit with \( a(0) = \text{LSb} \), and \( r \) indicates the location of the radix point. For example, in the case where \( A \) is an integer, \( r = 0 \) and when \( A \) is a fraction less than one, \( r = n \). The value of \( r \) only affects the interpretation of the numbers in a fixed point calculation, with the actual binary representation of the numbers independent of the value of \( r \). Factoring out of the above sum, it simply locates the radix point of the representation and is analogous to an exponent in a floating point system. Using the notation \( O_{i,j} \) to denote a fixed point binary number with \( i \) bits to the left of the radix point and \( j \) to the right, the above \( n \)-bit format is in \( O_{n-r,r} \). With care, fixed point calculations can be performed on operands in different \( O \) formats. Although the radix point must be aligned for addition or subtraction, multiplication provides an illustrative example of the simple interpretive nature of \( r \). Consider the unsigned product of a \( Q20.4 \) number with a \( Q8.8 \). After calling the appropriate unsigned \( 24 \times 16 \) bit multiply for these fixed point arguments, the 40-bit fixed point result is in \( Q28.12 \), where the arguments of the \( Q \) notation are summed respectively. Similar arguments can be made for two's complement arithmetic, where the negative representation of a positive number is obtained by reversing the value of each bit and incrementing the result by one. Producing a unique representation of zero, and covering the range \(-2^{n-1} \) to \( 2^{n-1} - 1 \), this is more easily applied in addition and subtraction operations and is therefore the most commonly used method of representing positive and negative numbers in fixed point arithmetic.

The above analysis in \( Q \) notation can be employed to build dedicated fixed point algorithms, leading to improved performance over floating point methods in cases where the size of the arguments required for the range and precision of the calculations is not large enough to destroy gains made by fixed point methods.
The fixed point library routines support 8-, 16-, 24- and 32-bit formats in the following combinations:

<table>
<thead>
<tr>
<th>Division Library Names</th>
<th>Format</th>
<th>Multiplication Library Names</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C5X/PIC16CXX Routines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FXD0808S, FXD0808U, FXD0807U</td>
<td>8/8</td>
<td>FXM0808S, FXM0808U, FXM0807U</td>
<td>8•8</td>
</tr>
<tr>
<td>FXD1608S, FXD1608U, FXD1607U, FXD1507U</td>
<td>16/8</td>
<td>FXM1608S, FXM1608U, FXM1507U</td>
<td>16•8</td>
</tr>
<tr>
<td>FXD1616S, FXD1616U, FXD1515U</td>
<td>16/16</td>
<td>FXM1616S, FXM1616U, FXM1515U</td>
<td>16•16</td>
</tr>
<tr>
<td>FXD2416S, FXD2416U, FXD2315U</td>
<td>24/16</td>
<td>FXM2416S, FXM2416U, FXM2315U</td>
<td>24•16</td>
</tr>
<tr>
<td>FXD2424S, FXD2424U, FXD2323U</td>
<td>24/24</td>
<td>FXM2424S, FXM2424U, FXM2323U</td>
<td>24•24</td>
</tr>
<tr>
<td>FXD3216S, FXD3216U, FXD3115U</td>
<td>32/16</td>
<td>FXM3216S, FXM3216U, FXM3115U</td>
<td>32•16</td>
</tr>
<tr>
<td>FXD3224S, FXD3224U, FXD3123U</td>
<td>32/24</td>
<td>FXM3224S, FXM3224U, FXM3123U</td>
<td>32•24</td>
</tr>
<tr>
<td>FXD3232S, FXD3232U, FXD3131U</td>
<td>32/32</td>
<td>FXM3232S, FXM3232U, FXM3131U</td>
<td>32•32</td>
</tr>
<tr>
<td>PIC17CXX Functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FXD0808S, FXD0808U, FXD0807U</td>
<td>8/8</td>
<td>FXM0808S, FXM0808U, FXM0807U</td>
<td>8•8</td>
</tr>
<tr>
<td>FXD1608S, FXD1608U, FXD1607U, FXD1507U</td>
<td>16/8</td>
<td>FXM1608S, FXM1608U, FXM1507U</td>
<td>16•8</td>
</tr>
<tr>
<td>FXD1616S, FXD1616U, FXD1615U, FXD1515U</td>
<td>16/16</td>
<td>FXM1616S, FXM1616U, FXM1515U</td>
<td>16•16</td>
</tr>
<tr>
<td>FXD2416S, FXD2416U, FXD2315U</td>
<td>24/16</td>
<td>FXM2416S, FXM2416U, FXM2315U</td>
<td>24•16</td>
</tr>
<tr>
<td>FXD2424S, FXD2424U, FXD2323U</td>
<td>24/24</td>
<td>FXM2424S, FXM2424U, FXM2323U</td>
<td>24•24</td>
</tr>
<tr>
<td>FXD3216S, FXD3216U, FXD3115U</td>
<td>32/16</td>
<td>FXM3216S, FXM3216U, FXM3115U</td>
<td>32•16</td>
</tr>
<tr>
<td>FXD3224S, FXD3224U, FXD3123U</td>
<td>32/24</td>
<td>FXM3224S, FXM3224U, FXM3123U</td>
<td>32•24</td>
</tr>
<tr>
<td>FXD3232S, FXD3232U, FXD3131U</td>
<td>32/32</td>
<td>FXM3232S, FXM3232U, FXM3131U</td>
<td>32•32</td>
</tr>
</tbody>
</table>

Note: U - unsigned math operation, S - signed math operation

These general format combinations are implemented in both signed and unsigned versions. Additional unsigned routines are implemented with arguments reduced by one bit to accommodate the case of operations on signed numbers, with arguments known to be nonnegative, thereby, resulting in some performance improvement.
DATA RAM REQUIREMENTS

The following contiguous data ram locations are used by the library:

- ACCB7 = REMB3 = AEXP = EXP
- ACCB6 = REMB2 = BEXP
- ACCB5 = REMB1
- ACCB4 = REMB0
- ACCB3 = AARGB3
- ACCB2 = AARGB2
- ACCB1 = AARGB1
- ACCB0 = AARGB0 = ACC
- SIGN
- FPFLAGS
- BARGB3
- BARGB2
- BARGB1
- BARGB0
- TEMPB3
- TEMPB2
- TEMPB1
- TEMPB0

These definitions are identical with those used by the IEEE 754 compliant floating point library[6], AN575.

USAGE

Multiplication assumes the multiplicand in AARG, multiplier in BARG, and produces the result in ACC. Division assumes a dividend in AARG, divisor in BARG, and quotient in ACC with remainder in REM.

ADDITION/SUBTRACTION

Because of the generally trivial nature of addition and subtraction, the call and return overhead outweighs the need for explicit routines and so they are not included in the library. However, the PIC16C5X/PIC16CXX families do not have an add with carry or subtract with borrow instruction, leading to subtleties regarding production of a correct carry-out in a multiple byte add or subtract. In the case of a two byte add or subtract, the most elegant solution to these difficulties, requiring 6 cycles, appears to be given by the following code in Example 1.

EXAMPLE 1: TWO BYTE ADDITION/SUBTRACTION Routines

```
ADD
MOVF AARGBl, W
ADDWF BARGBl
MOVF AARGB0, W
BTFSC _c
INCFSZ AARGB0, W
ADDWF BARGBO
SUB
MOVF AARGBl, W
SUBWF BARGBl
MOVF AARGB0, W
BTFSS _c
INCFSZ AARGB0, W
SUBWF BARGBO
```

The four instructions after the initial add/subtract, can be easily concatenated for operations involving more than two bytes. Because addition and subtraction are required in standard algorithms for multiplication and division, these issues permeate the implementation of both fixed and floating point algorithms for the PIC16C5X/PIC16CXX families.

MULTIPLICATION

The fixed point multiply routine FXPMxxyy, takes an xx-bit multiplicand in AARG, a yy-bit multiplier in BARG and returns the (xx+yy)-bit product in ACC. The implementation uses a standard sequential add-shift algorithm, negating both factors if BARG < 0, to produce the positive multiplier required by the method. Analogous to simple longhand binary multiplication, the multiplier bits are sequentially tested, with one indicating an add-shift and zero simply a shift. The shift
is required to align the partial product for the next possible add[1]. Several examples are shown in Example 2.

**EXAMPLE 2: MULTIPLICATION EXAMPLES**

FXM2416S(0xC11682,0x608B)

= FXM2416S(-4123006,24715)

= 0xE84647F896

= -101900093290

FXM1616U(0x0458,0x822C)

= FXM1616U(1112,33324)

= 0x02356F20

= 37056288

**TABLE 1: PIC17CXX FIXED POINT MULTIPLY PERFORMANCE DATA: APPENDIX E**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Max Cyc</th>
<th>Min Cyc</th>
<th>PM</th>
<th>DM</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXM0808S</td>
<td>50/53</td>
<td>26</td>
<td>65</td>
<td>5</td>
<td>2-375</td>
</tr>
<tr>
<td>FXM0808U</td>
<td>39</td>
<td>23</td>
<td>53</td>
<td>3</td>
<td>2-375</td>
</tr>
<tr>
<td>FXM0707U</td>
<td>37</td>
<td>21</td>
<td>49</td>
<td>3</td>
<td>2-375</td>
</tr>
<tr>
<td>FXM1608S</td>
<td>74/79</td>
<td>35</td>
<td>100</td>
<td>6</td>
<td>2-376</td>
</tr>
<tr>
<td>FXM1608U</td>
<td>75</td>
<td>24</td>
<td>90</td>
<td>6</td>
<td>2-376</td>
</tr>
<tr>
<td>FXM1507U</td>
<td>69</td>
<td>24</td>
<td>82</td>
<td>6</td>
<td>2-376</td>
</tr>
<tr>
<td>FXM1616S</td>
<td>168/175</td>
<td>24</td>
<td>197</td>
<td>8</td>
<td>2-377</td>
</tr>
<tr>
<td>FXM1616U</td>
<td>156</td>
<td>41</td>
<td>191</td>
<td>8</td>
<td>2-377</td>
</tr>
<tr>
<td>FXM1515U</td>
<td>150</td>
<td>39</td>
<td>185</td>
<td>8</td>
<td>2-377</td>
</tr>
<tr>
<td>FXM2416S</td>
<td>213/223</td>
<td>43</td>
<td>253</td>
<td>10</td>
<td>2-378</td>
</tr>
<tr>
<td>FXM2416U</td>
<td>203</td>
<td>43</td>
<td>239</td>
<td>10</td>
<td>2-378</td>
</tr>
<tr>
<td>FXM1615U</td>
<td>194</td>
<td>41</td>
<td>229</td>
<td>10</td>
<td>2-378</td>
</tr>
<tr>
<td>FXM2424S</td>
<td>334/346</td>
<td>60</td>
<td>392</td>
<td>12</td>
<td>2-379</td>
</tr>
<tr>
<td>FXM2424U</td>
<td>316</td>
<td>60</td>
<td>364</td>
<td>12</td>
<td>2-379</td>
</tr>
<tr>
<td>FXM3232U</td>
<td>308</td>
<td>58</td>
<td>354</td>
<td>12</td>
<td>2-380</td>
</tr>
<tr>
<td>FXM3216S</td>
<td>258/270</td>
<td>46</td>
<td>301</td>
<td>13</td>
<td>2-380</td>
</tr>
<tr>
<td>FXM3216U</td>
<td>265</td>
<td>44</td>
<td>298</td>
<td>13</td>
<td>2-380</td>
</tr>
<tr>
<td>FXM3115U</td>
<td>254</td>
<td>42</td>
<td>285</td>
<td>13</td>
<td>2-381</td>
</tr>
<tr>
<td>FXM3224S</td>
<td>403/417</td>
<td>62</td>
<td>464</td>
<td>15</td>
<td>2-381</td>
</tr>
<tr>
<td>FXM3224U</td>
<td>410</td>
<td>61</td>
<td>459</td>
<td>15</td>
<td>2-382</td>
</tr>
<tr>
<td>FXM3123U</td>
<td>399</td>
<td>59</td>
<td>446</td>
<td>15</td>
<td>2-382</td>
</tr>
<tr>
<td>FXM3232S</td>
<td>556/572</td>
<td>79</td>
<td>635</td>
<td>16</td>
<td>2-383</td>
</tr>
<tr>
<td>FXM3232U</td>
<td>563</td>
<td>78</td>
<td>628</td>
<td>16</td>
<td>2-383</td>
</tr>
<tr>
<td>FXM3131U</td>
<td>543</td>
<td>76</td>
<td>606</td>
<td>16</td>
<td>2-384</td>
</tr>
</tbody>
</table>

**TABLE 2: PIC16C5X/PIC16CXX FIXED POINT MULTIPLY PERFORMANCE DATA: APPENDIX C**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Max Cyc</th>
<th>Min Cyc</th>
<th>PM</th>
<th>DM</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXM0808S</td>
<td>79/82</td>
<td>55</td>
<td>33</td>
<td>5</td>
<td>2-252</td>
</tr>
<tr>
<td>FXM0808U</td>
<td>73</td>
<td>54</td>
<td>21</td>
<td>4</td>
<td>2-252</td>
</tr>
<tr>
<td>FXM0707U</td>
<td>67</td>
<td>48</td>
<td>23</td>
<td>4</td>
<td>2-252</td>
</tr>
<tr>
<td>FXM1608S</td>
<td>122/128</td>
<td>55</td>
<td>44</td>
<td>7</td>
<td>2-246</td>
</tr>
<tr>
<td>FXM1608U</td>
<td>126</td>
<td>59</td>
<td>31</td>
<td>7</td>
<td>2-247</td>
</tr>
<tr>
<td>FXM1507U</td>
<td>114</td>
<td>52</td>
<td>35</td>
<td>7</td>
<td>2-247</td>
</tr>
<tr>
<td>FXM1616S</td>
<td>260/269</td>
<td>105</td>
<td>74</td>
<td>9</td>
<td>2-240</td>
</tr>
<tr>
<td>FXM1616U</td>
<td>256</td>
<td>107</td>
<td>58</td>
<td>9</td>
<td>2-241</td>
</tr>
<tr>
<td>FXM1515U</td>
<td>244</td>
<td>103</td>
<td>63</td>
<td>9</td>
<td>2-241</td>
</tr>
<tr>
<td>FXM2416S</td>
<td>334/346</td>
<td>108</td>
<td>92</td>
<td>12</td>
<td>2-231</td>
</tr>
<tr>
<td>FXM2416U</td>
<td>334</td>
<td>110</td>
<td>70</td>
<td>12</td>
<td>2-232</td>
</tr>
<tr>
<td>FXM2315U</td>
<td>319</td>
<td>104</td>
<td>76</td>
<td>12</td>
<td>2-232</td>
</tr>
<tr>
<td>FXM2424S</td>
<td>520/535</td>
<td>157</td>
<td>126</td>
<td>13</td>
<td>2-221</td>
</tr>
<tr>
<td>FXM2424U</td>
<td>512</td>
<td>159</td>
<td>98</td>
<td>13</td>
<td>2-222</td>
</tr>
<tr>
<td>FXM3224S</td>
<td>634/652</td>
<td>160</td>
<td>152</td>
<td>15</td>
<td>2-196</td>
</tr>
<tr>
<td>FXM3224U</td>
<td>630</td>
<td>162</td>
<td>151</td>
<td>15</td>
<td>2-197</td>
</tr>
<tr>
<td>FXM3123U</td>
<td>610</td>
<td>157</td>
<td>129</td>
<td>15</td>
<td>2-197</td>
</tr>
<tr>
<td>FXM3232S</td>
<td>868/889</td>
<td>207</td>
<td>189</td>
<td>17</td>
<td>2-181</td>
</tr>
<tr>
<td>FXM3232U</td>
<td>856</td>
<td>209</td>
<td>168</td>
<td>17</td>
<td>2-182</td>
</tr>
<tr>
<td>FXM3131U</td>
<td>836</td>
<td>204</td>
<td>168</td>
<td>17</td>
<td>2-182</td>
</tr>
</tbody>
</table>

Legend: PM - Program memory, DM - Data Memory
DIVISION

The fixed point divide routine FXPDxxyy, takes an xx-bit dividend in AARG, a yy-bit divisor in BARG and returns the xx-bit quotient in ACC and yy-bit remainder in REM. Unlike multiplication, division is not deterministic, requiring a trial-and-error sequential shift and subtract process. Binary division is less complicated than decimal division because the possible quotient digits are only zero or one. If the divisor is less than the partial remainder, the corresponding quotient bit is set to one followed by a shift and subtract. Otherwise, the divisor is greater than the partial remainder, the quotient bit is set to zero and only a shift is performed. The intermediate partial remainder may be restored at each stage as in restoring division, or corrected at the end as in nonrestoring division. Implementation dependent trade-offs between worst case versus average performance affect the choice between these two approaches, and therefore, macros for each method are provided.

Note: A test for divide by zero exception is not performed and must be explicitly provided by the user.

The results of the division process for AARG/BARG, satisfy the relation

AARG = BARG \cdot QUOTIENT + REMAINDER,

where the remainder has the same sign as the quotient, and represents the fraction of the result in units of the denominator BARG. Some simple examples are given in Example 3.

EXAMPLE 3: DIVISION EXAMPLES

FXD1608S(0xC116,0x60) = 0xFF59, 0xB6
FXD1616U(0x9543,0x4AA1) = 0x0002, 0x0001
### TABLE 3: PIC17CXX FIXED POINT DIVIDE PERFORMANCE DATA: APPENDIX F

<table>
<thead>
<tr>
<th>Routine</th>
<th>Max Cyc</th>
<th>Min Cyc</th>
<th>PM</th>
<th>DM</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXD0808S</td>
<td>71/77</td>
<td>71/77</td>
<td>77</td>
<td>4</td>
<td>2-449</td>
</tr>
<tr>
<td>FXD0808U</td>
<td>75</td>
<td>67</td>
<td>74</td>
<td>3</td>
<td>2-449</td>
</tr>
<tr>
<td>FXD0807U</td>
<td>66</td>
<td>66</td>
<td>65</td>
<td>3</td>
<td>2-450</td>
</tr>
<tr>
<td>FXD0707U</td>
<td>61</td>
<td>61</td>
<td>60</td>
<td>3</td>
<td>2-450</td>
</tr>
<tr>
<td>FXD1608S</td>
<td>135/146</td>
<td>135/146</td>
<td>146</td>
<td>5</td>
<td>2-448</td>
</tr>
<tr>
<td>FXD1608U</td>
<td>196</td>
<td>156</td>
<td>195</td>
<td>4</td>
<td>2-448</td>
</tr>
<tr>
<td>FXD1607U</td>
<td>130</td>
<td>130</td>
<td>129</td>
<td>4</td>
<td>2-448</td>
</tr>
<tr>
<td>FXD1507U</td>
<td>125</td>
<td>125</td>
<td>124</td>
<td>4</td>
<td>2-449</td>
</tr>
<tr>
<td>FXD1616S</td>
<td>201/214</td>
<td>187/200</td>
<td>241</td>
<td>7</td>
<td>2-446</td>
</tr>
<tr>
<td>FXD1616U</td>
<td>294</td>
<td>230</td>
<td>159</td>
<td>4</td>
<td>2-447</td>
</tr>
<tr>
<td>FXD1615U</td>
<td>197</td>
<td>182</td>
<td>216</td>
<td>6</td>
<td>2-447</td>
</tr>
<tr>
<td>FXD1515U</td>
<td>191</td>
<td>177</td>
<td>218</td>
<td>6</td>
<td>2-447</td>
</tr>
<tr>
<td>FXD2416S</td>
<td>297/314</td>
<td>272/289</td>
<td>353</td>
<td>8</td>
<td>2-444</td>
</tr>
<tr>
<td>FXD2416U</td>
<td>365</td>
<td>339</td>
<td>453</td>
<td>8</td>
<td>2-445</td>
</tr>
<tr>
<td>FXD2415U</td>
<td>294</td>
<td>268</td>
<td>339</td>
<td>8</td>
<td>2-445</td>
</tr>
<tr>
<td>FXD2315U</td>
<td>287</td>
<td>262</td>
<td>330</td>
<td>8</td>
<td>2-446</td>
</tr>
<tr>
<td>FXD2424S</td>
<td>371/390</td>
<td>344/363</td>
<td>482</td>
<td>10</td>
<td>2-475</td>
</tr>
<tr>
<td>FXD2424U</td>
<td>440</td>
<td>412</td>
<td>577</td>
<td>10</td>
<td>2-476</td>
</tr>
<tr>
<td>FXD2423U</td>
<td>369</td>
<td>341</td>
<td>460</td>
<td>9</td>
<td>2-476</td>
</tr>
<tr>
<td>FXD2323U</td>
<td>361</td>
<td>334</td>
<td>448</td>
<td>9</td>
<td>2-476</td>
</tr>
<tr>
<td>FXD3216S</td>
<td>393/414</td>
<td>363/384</td>
<td>476</td>
<td>9</td>
<td>2-473</td>
</tr>
<tr>
<td>FXD3216U</td>
<td>485</td>
<td>459</td>
<td>608</td>
<td>9</td>
<td>2-474</td>
</tr>
<tr>
<td>FXD3215U</td>
<td>390</td>
<td>359</td>
<td>451</td>
<td>8</td>
<td>2-474</td>
</tr>
<tr>
<td>FXD3115U</td>
<td>383</td>
<td>353</td>
<td>442</td>
<td>8</td>
<td>2-475</td>
</tr>
<tr>
<td>FXD3224S</td>
<td>491/514</td>
<td>456/479</td>
<td>639</td>
<td>11</td>
<td>2-419</td>
</tr>
<tr>
<td>FXD3224U</td>
<td>584</td>
<td>548</td>
<td>769</td>
<td>11</td>
<td>2-420</td>
</tr>
<tr>
<td>FXD3223U</td>
<td>489</td>
<td>453</td>
<td>612</td>
<td>10</td>
<td>2-421</td>
</tr>
<tr>
<td>FXD3123U</td>
<td>481</td>
<td>446</td>
<td>600</td>
<td>10</td>
<td>2-421</td>
</tr>
<tr>
<td>FXD3232S</td>
<td>589/614</td>
<td>552/577</td>
<td>800</td>
<td>13</td>
<td>2-418</td>
</tr>
<tr>
<td>FXD3232U</td>
<td>683</td>
<td>645</td>
<td>930</td>
<td>13</td>
<td>2-419</td>
</tr>
<tr>
<td>FXD3231U</td>
<td>588</td>
<td>550</td>
<td>773</td>
<td>12</td>
<td>2-419</td>
</tr>
<tr>
<td>FXD3131U</td>
<td>579</td>
<td>542</td>
<td>758</td>
<td>12</td>
<td>2-419</td>
</tr>
</tbody>
</table>

Legend: PM - Program memory, DM - Data Memory

### TABLE 4: PIC16C5X/PIC16CXX FIXED POINT DIVIDE PERFORMANCE DATA: APPENDIX D

<table>
<thead>
<tr>
<th>Routine</th>
<th>Max Cyc</th>
<th>Min Cyc</th>
<th>PM</th>
<th>DM</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXD0808S</td>
<td>90/96</td>
<td>90/96</td>
<td>41</td>
<td>5</td>
<td>2-344</td>
</tr>
<tr>
<td>FXD0808U</td>
<td>100</td>
<td>92</td>
<td>15</td>
<td>4</td>
<td>2-345</td>
</tr>
<tr>
<td>FXD0807U</td>
<td>88</td>
<td>88</td>
<td>21</td>
<td>4</td>
<td>2-345</td>
</tr>
<tr>
<td>FXD0707U</td>
<td>80</td>
<td>80</td>
<td>44</td>
<td>4</td>
<td>2-345</td>
</tr>
<tr>
<td>FXD1608S</td>
<td>176/188</td>
<td>176/188</td>
<td>67</td>
<td>6</td>
<td>2-338</td>
</tr>
<tr>
<td>FXD1608U</td>
<td>294</td>
<td>230</td>
<td>41</td>
<td>7</td>
<td>2-339</td>
</tr>
<tr>
<td>FXD1607U</td>
<td>174</td>
<td>174</td>
<td>41</td>
<td>5</td>
<td>2-339</td>
</tr>
<tr>
<td>FXD1507U</td>
<td>166</td>
<td>166</td>
<td>44</td>
<td>5</td>
<td>2-340</td>
</tr>
<tr>
<td>FXD1616S</td>
<td>304/319</td>
<td>269/284</td>
<td>74</td>
<td>8</td>
<td>2-330</td>
</tr>
<tr>
<td>FXD1616U</td>
<td>373</td>
<td>277</td>
<td>27</td>
<td>7</td>
<td>2-330</td>
</tr>
<tr>
<td>FXD1515U</td>
<td>294</td>
<td>274</td>
<td>45</td>
<td>7</td>
<td>2-331</td>
</tr>
<tr>
<td>FXD2416S</td>
<td>417/438</td>
<td>389/410</td>
<td>140</td>
<td>8</td>
<td>2-321</td>
</tr>
<tr>
<td>FXD2416U</td>
<td>529</td>
<td>501</td>
<td>172</td>
<td>8</td>
<td>2-322</td>
</tr>
<tr>
<td>FXD2315U</td>
<td>407</td>
<td>379</td>
<td>120</td>
<td>7</td>
<td>2-322</td>
</tr>
<tr>
<td>FXD2424S</td>
<td>541/565</td>
<td>509/533</td>
<td>253</td>
<td>12</td>
<td>2-311</td>
</tr>
<tr>
<td>FXD2424U</td>
<td>676</td>
<td>644</td>
<td>226</td>
<td>13</td>
<td>2-312</td>
</tr>
<tr>
<td>FXD2323U</td>
<td>531</td>
<td>499</td>
<td>211</td>
<td>12</td>
<td>2-313</td>
</tr>
<tr>
<td>FXD3216S</td>
<td>551/578</td>
<td>515/551</td>
<td>201</td>
<td>10</td>
<td>2-299</td>
</tr>
<tr>
<td>FXD3216U</td>
<td>703</td>
<td>667</td>
<td>243</td>
<td>9</td>
<td>2-300</td>
</tr>
<tr>
<td>FXD3115U</td>
<td>541</td>
<td>505</td>
<td>160</td>
<td>9</td>
<td>2-300</td>
</tr>
<tr>
<td>FXD3224S</td>
<td>715/742</td>
<td>675/702</td>
<td>280</td>
<td>11</td>
<td>2-287</td>
</tr>
<tr>
<td>FXD3224U</td>
<td>867</td>
<td>827</td>
<td>299</td>
<td>11</td>
<td>2-288</td>
</tr>
<tr>
<td>FXD3123U</td>
<td>705</td>
<td>665</td>
<td>232</td>
<td>10</td>
<td>2-288</td>
</tr>
<tr>
<td>FXD3232S</td>
<td>879/912</td>
<td>835/868</td>
<td>357</td>
<td>13</td>
<td>2-271</td>
</tr>
<tr>
<td>FXD3232U</td>
<td>1031</td>
<td>987</td>
<td>364</td>
<td>13</td>
<td>2-272</td>
</tr>
<tr>
<td>FXD3131U</td>
<td>869</td>
<td>825</td>
<td>304</td>
<td>13</td>
<td>2-272</td>
</tr>
</tbody>
</table>

Legend: PM - Program memory, DM - Data Memory

### REFERENCES

APPENDIX A: ALGORITHMS

Several algorithms for decimal to binary conversion are given below. The integer and fractional conversion algorithms are useful in both native assembly as well as high level languages.

A.1 Integer conversion algorithm[3]:

Given an integer \( I \), where \( d(k) \) are the bit values of its \( n \) bit binary representation with \( d(0) = \text{LSB} \),

\[
I = \sum_{k=0}^{n-1} d(k) \cdot 2^k
\]

\( k = 0 \)
\( I(k) = I \)
while \( I(k) = \neq 0 \)
\( d(k) = \text{remainder of } l(k)/2 \)
\( l(k+1) = \left\lfloor \frac{l(k)}{2} \right\rfloor \)
\( k = k + 1 \)
endw

where \( \left\lfloor \cdot \right\rfloor \) denotes the greatest integer function (or ceiling function).

A.2 Fractional conversion algorithm[3]:

Given a fraction \( F \), where \( d(k) \) are the bit values of its \( n \) bit binary representation with \( d(1) = \text{MSB} \),

\[
F = \sum_{k=1}^{n} d(k) \cdot 2^{-k}
\]

\( k = 0 \)
\( F(k) = F \)
while \( k <= n \)
\( d(k) = \left\lceil F(k) \cdot 2 \right\rceil \)
\( F(k+1) = \text{fractional part of } F(k) \cdot 2 \)
\( k = k + 1 \)
endw
APPENDIX B: FLOWCHARTS

FIGURE B-1: MULTIPLICATION FLOWCHART

For FXPMxxU:

- **i = 0**
- Clear high order partial product
- **i = i + 1**
- Right shift partial product
- **Yes**
- **i < n**?
- **No**
- Return 0

For FXPMxxS:

- **i = 0**
- Clear high order partial product
- **Yes**
- Multiplier bit i = 1?
- **No**
- Right shift partial product with sign extension
- **Yes**
- **i < n - 1**?
- **No**
- Return 0
FIGURE B-2: DIVISION FLOWCHART

FXPDxxU

i = 0
clear remainder
partial quotient = dividend

i < n
No

left shift
(remainder, quotient)

Return 0

Yes

subtract division
from partial remainder

result > 0
Yes

Yes

i < n - 1
No

i = 0
clear remainder
partial quotient = dividend

No

negate dividend

negate divisor

result > 0
No

partial quotient, LSb = 1

No

partial quotient, LSb = 0

Yes

dividend < 0
Yes

Return 0

A

FXPDxxS

compute sign of quotient

divisor < 0
No

negate divisor

result > 0
Yes

A

left shift
(remainder, quotient)

A

restore partial quotient, LSb = 0

restore partial quotient, LSb = 0

A

A

A
APPENDIX C: MULTIPLY ROUTINES FOR THE PIC16C5X/PIC16CXX

Table of Contents for Appendix C

C.1 32x32 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ....................................................... 2-164
C.2 32x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ....................................................... 2-182
C.3 32x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ....................................................... 2-197
C.4 24x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ....................................................... 2-209
C.5 24x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ....................................................... 2-222
C.6 16x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ....................................................... 2-232
C.7 16x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ......................................................... 2-241
C.8 8x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines ........................................................... 2-247

C.1 32x32 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

; 32x32 PIC16 FIXED POINT MULTIPLY ROUTINES
; Input: fixed point arguments in AARG and BARG
; Output: product AARGxBARG in AARG
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed multiply application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in performance.
; Routine Clocks Function
; FXM3232S 889 32x32 -> 64 bit signed fixed point multiply
; FXM3232U 856 32x32 -> 64 bit unsigned fixed point multiply
; FXM3131U 836 31x31 -> 64 bit unsigned fixed point multiply
; The above timings are based on the looped macros. If space permits,
; approximately 128-168 clocks can be saved by using the unrolled macros.
list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

;************************************************************
; Test suite storage
RANHIEqu 0x1A ; random number generator registers
RANDLoEqu 0x1B
DATAeq 0x20 ; beginning of test data
;*******************************************************************************
; Test suite for 32x32 bit fixed point multiply algorithms
org 0x0005
MAIN
MOVLW RAMSTART
MOVWF FSR
MEMLOOP
CLRF INDF
INCF FSR
MOVLW RAMSTOP
SUBWF FSR,W
BTFSS _Z
GOTO MEMLOOP
MOVLW 0x45 ; seed for random numbers
MOVWF RANDLO
MOVLF 0x30
MOVWF RANDHI
MOVWF RANDLO
MOVWF DATA
MOVWF FSR
BCF _RF0
BCF _RF1
BCF _RF2
CALL TFMX3232
SELF
GOTO SELF
RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMPBO
SWAPF RANDHI
SWAPF RANDLO,W
; Test suite for FXM3232

CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGB0
BCF BARGBO,MSB
MOVF BARGB0,W
MOVWF INDF
INCF FSR
MOVF RANDLO,W
MOVWF BARGB1
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGB2
MOVWF INDF
INCF FSR
MOVF RANDLO,W
MOVWF BARGB3
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF AARGB0
BCF AARGBO,MSB
MOVF AARGB0,W
MOVWF INDF
INCF FSR
MOVF RANDLO,W
MOVWF AARGB1
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF AARGB2
MOVWF INDF
INCF FSR
MOVF RANDLO,W
MOVWF AARGB3
MOVWF INDF
INCF FSR
CALL FXM3131U
MOVF AARGBO,W
MOVWF INDF
INCF FSR
MOVF AARGB1,W
MOVWF AARGB4,W
MOVWF INDF
INCF FSR
MOVF AARGB2,W
MOVWF INDF
INCF FSR
MOVF AARGB4,W
MOVWF INDF
;********************************************************************************************
;**********************************************************************************************
32x32 Bit Multiplication Macros

; SMUL3232L macro
; Max Timing: 2+13*26+25+2+7*27+26+2+7*28+26+2+7*29+28+9 = 851 clks
; Min Timing: 2+7*6+5+1+7*6+5+1+7*6+5+2+6*6+5+6 = 192 clks
; PM: 31+25+2+6*26+2+26+2+27+2+28+9 = 152 DM: 17

MOVLW 0x8
MOVWF LOOPCOUNT

LOOPSM3232A
RRF BARGB3
BTFSC _C
GOTO ALSM3232NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3232A
MOVWF LOOPCOUNT

LOOPSM3232B
RRF BARGB2
BTFSC _C
GOTO BLSM3232NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3232B
MOVWF LOOPCOUNT

LOOPSM3232C
RRF BARGB1
BTFSC _C
GOTO CLSM3232NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3232C
MOVWF LOOPCOUNT

LOOPSM3232D
RRF BARGBO
BTFSC _C
GOTO DLSM3232NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3232D
CLRF AARGBO
CLRF AARGBl
CLRF AARGB2
CLRF AARGB3
RETLW 0x00

ALOOPSM3232
RRF BARGB3
BTFSS _C
GOTO ALSM3232NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
RETLW 0x00

© 1995 Microchip Technology Inc.
ADDWF   ACCB1
MOVF   TEMPB0,W
BTFS   _C
INCFSZ   TEMPB0,W
ADDWF   ACCB0
RLF   TEMPB0,W
RRF   ACCB0
RRF   ACCB1
RRF   ACCB2
RRF   ACCB3
RRF   ACCB4
DECFSZ   LOOPCOUNT
GOTO   ALOOPSM3232
MOVFW   0x8
MOVWF   LOOPCOUNT

ALSM3232NA

BLOOPSM3232

RRF   BARGB2
BTFS   _C
GOTO   BLSM3232NA
MOVF   TEMPB3,W
ADDWF   ACCB3
MOVF   TEMPB2,W
BTFS   _C
INCFSZ   TEMPB2,W
ADDWF   ACCB2
MOVF   TEMPB1,W
BTFS   _C
INCFSZ   TEMPB1,W
ADDWF   ACCB1
MOVF   TEMPB0,W
BTFS   _C
INCFSZ   TEMPB0,W
ADDWF   ACCB0
RLF   TEMPB0,W
RRF   ACCB0
RRF   ACCB1
RRF   ACCB2
RRF   ACCB3
RRF   ACCB4
RRF   ACCB5
DECFSZ   LOOPCOUNT
GOTO   BLOOPSM3232
MOVFW   0x8
MOVWF   LOOPCOUNT

BLSM3232NA

CLOOPSM3232

RRF   BARGB1
BTFS   _C
GOTO   CLSM3232NA
MOVF   TEMPB3,W
ADDWF   ACCB3
MOVF   TEMPB2,W
BTFS   _C
INCFSZ   TEMPB2,W
ADDWF   ACCB2
MOVF   TEMPB1,W
BTFS   _C
INCFSZ   TEMPB1,W
ADDWF   ACCB1
MOVF   TEMPB0,W
BTFS   _C
INCFSZ   TEMPB0,W
ADDWF   ACCB0
RLF   TEMPB0,W
RRF   ACCB0
RRF   ACCB1
RRF   ACCB2
AN617

RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
DECFSZ LOOPCOUNT
GOTO CLOOPSM3232
MOVlw 0x7
MOVWF LOOPCOUNT

DLOOPSM3232
RRF BARGBO
BTFSS _c
GOTO DLSM3232NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
DECFSZ LOOPCOUNT
GOTO DLOOPSM3232
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7

endm

UMUL3232L macro
; Max Timing: 2+15*6+25+24+2+7*26+25+2+7*27+26+2+7*28+27 = 842 clks
; Min Timing: 2+7*6+5+1+7*6+5+1+7*6+5+1+7*6+5+6 = 197 clks
; PM: 38+24+2+25+2+26+2+2+9 = 155 DM: 17
MOVlw 0x80
MOVwf LOOPCOUNT

LOOPUM3232A
RRF BARGBO
BTFSS _c
GOTO ALUM3232NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3232A
MOVWF LOOPCOUNT

LOOPUM3232B
RRF BARGBO
BTFSS _c
GOTO BLUM3232NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3232B

© 1995 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Line</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-02</td>
<td>MOVWF LOOFCOUNT</td>
</tr>
<tr>
<td>03-04</td>
<td>RRF BARGB1</td>
</tr>
<tr>
<td>05-06</td>
<td>BTFSC _c</td>
</tr>
<tr>
<td>07-09</td>
<td>GOTO CLUM3232NAP</td>
</tr>
<tr>
<td>10-11</td>
<td>DECFSZ LOOFCOUNT</td>
</tr>
<tr>
<td>12-13</td>
<td>GOTO LOOPUM3232C</td>
</tr>
<tr>
<td>14-15</td>
<td>MOVWF LOOFCOUNT</td>
</tr>
<tr>
<td>16-18</td>
<td>RRF BARGB0</td>
</tr>
<tr>
<td>19-20</td>
<td>BTFSC _c</td>
</tr>
<tr>
<td>21-23</td>
<td>GOTO DLUM3232NAP</td>
</tr>
<tr>
<td>24-26</td>
<td>DCRYFSZ LOOFCOUNT</td>
</tr>
<tr>
<td>27-28</td>
<td>GOTO LOOPUM3232D</td>
</tr>
<tr>
<td>29-31</td>
<td>CLRFL AARGB0</td>
</tr>
<tr>
<td>32-34</td>
<td>CLRFL AARGB1</td>
</tr>
<tr>
<td>35-37</td>
<td>CLRFL AARGB2</td>
</tr>
<tr>
<td>38-39</td>
<td>CLRFL AARGB3</td>
</tr>
<tr>
<td>40-41</td>
<td>RLSTW 0x00</td>
</tr>
<tr>
<td>42-43</td>
<td>BCF _c</td>
</tr>
<tr>
<td>44-45</td>
<td>GOTO ALUM3232NA</td>
</tr>
<tr>
<td>46-48</td>
<td>BCF _c</td>
</tr>
<tr>
<td>49-50</td>
<td>GOTO BLUM3232NA</td>
</tr>
<tr>
<td>51-53</td>
<td>BCF _c</td>
</tr>
<tr>
<td>54-55</td>
<td>GOTO CLUM3232NA</td>
</tr>
<tr>
<td>56-58</td>
<td>BCF _c</td>
</tr>
<tr>
<td>59-60</td>
<td>GOTO DLUM3232NA</td>
</tr>
<tr>
<td>61-62</td>
<td>RRF BARGB3</td>
</tr>
<tr>
<td>63-64</td>
<td>BTFSS _c</td>
</tr>
<tr>
<td>65-67</td>
<td>GOTO ALUM3232NA</td>
</tr>
<tr>
<td>68-70</td>
<td>MOVF TEMPB3,W</td>
</tr>
<tr>
<td>71-73</td>
<td>ADDWF ACCB3</td>
</tr>
<tr>
<td>74-75</td>
<td>MOVF TEMPB2,W</td>
</tr>
<tr>
<td>76-77</td>
<td>BTFSS _c</td>
</tr>
<tr>
<td>78-80</td>
<td>INCFSZ TEMPB2,W</td>
</tr>
<tr>
<td>81-83</td>
<td>ADDWF ACCB2</td>
</tr>
<tr>
<td>84-85</td>
<td>MOVF TEMPB1,W</td>
</tr>
<tr>
<td>86-87</td>
<td>BTFSS _c</td>
</tr>
<tr>
<td>88-90</td>
<td>INCFSZ TEMPB1,W</td>
</tr>
<tr>
<td>91-93</td>
<td>ADDWF ACCB1</td>
</tr>
<tr>
<td>94-95</td>
<td>MOVF TEMPB0,W</td>
</tr>
<tr>
<td>96-97</td>
<td>BTFSS _c</td>
</tr>
<tr>
<td>98-100</td>
<td>INCFSZ TEMPB0,W</td>
</tr>
<tr>
<td>101-103</td>
<td>ADDWF ACCB0</td>
</tr>
<tr>
<td>104-105</td>
<td>RRF ACCB0</td>
</tr>
<tr>
<td>106-107</td>
<td>RRF ACCB1</td>
</tr>
<tr>
<td>108-109</td>
<td>RRF ACCB2</td>
</tr>
<tr>
<td>110-111</td>
<td>RRF ACCB3</td>
</tr>
<tr>
<td>112-113</td>
<td>RRF ACCB4</td>
</tr>
<tr>
<td>114-116</td>
<td>DECFSZ LOOFCOUNT</td>
</tr>
<tr>
<td>117-119</td>
<td>GOTO ALOOPUM3232</td>
</tr>
<tr>
<td>120-121</td>
<td>MOVLW 0x08</td>
</tr>
<tr>
<td>122-123</td>
<td>MOVWF LOOFCOUNT</td>
</tr>
<tr>
<td>124-125</td>
<td>RRF BARGB2</td>
</tr>
<tr>
<td>126-127</td>
<td>BTFSS _c</td>
</tr>
<tr>
<td>128-129</td>
<td>GOTO BLUM3232NA</td>
</tr>
<tr>
<td>130-132</td>
<td>MOVF TEMPB3,W</td>
</tr>
<tr>
<td>133-135</td>
<td>ADDWF ACCB3</td>
</tr>
<tr>
<td>136-137</td>
<td>MOVF TEMPB2,W</td>
</tr>
<tr>
<td>138-139</td>
<td>BTFSS _c</td>
</tr>
<tr>
<td>140-142</td>
<td>INCFSZ TEMPB2,W</td>
</tr>
<tr>
<td>143-145</td>
<td>ADDWF ACCB2</td>
</tr>
<tr>
<td>146-147</td>
<td>MOVF TEMPB1,W</td>
</tr>
</tbody>
</table>
BTFSC  _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC  _C
INCFSZ TEMPB0,W
ADDWF ACCB0

BLUM3232NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
DECFNZ LOOPCOUNT
GOTO BLOOPUM3232
MOVLW 0x08
MOVF LOOPCOUNT

CLOOPUM3232
RRF BARGB1
BTFSS _C
GOTO CLUM3232NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC  _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC  _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC  _C
INCFSZ TEMPB0,W
ADDWF ACCB0

CLUM3232NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
DECFNZ LOOPCOUNT
GOTO CLOOPUM3232
MOVLW 0x08
MOVF LOOPCOUNT

DLOOPUM3232
RRF BARGB0
BTFSS _C
GOTO DLUM3232NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC  _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC  _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC  _C
INCFSZ TEMPB0,W
ADDWF ACCB0
DLUM3232NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
DECFSZ LOOPCOUNT
GOTO DLOOPUM3232
endm
UMUL3131L
macro
; Max Timing: 2+15+6*25+24+2+7*26+25+2+7*27+26+2+6*28+27+8 = 822 clks
; Min Timing: 2+7*6+5+1+7*6+5+1+7*6+5+2+6*6+5+6 = 192 clks
; PM: 39+24+25+2+26+2+27+8 = 155 DM: 17
MOVLW Ox8
MOVF LOOPCOUNT
LOOPUM3131A
RRF BARGB3
BTFSC _c
GOTO ALUM3131NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3131A
MOVF LOOPCOUNT
LOOPUM3131B
RRF BARGB2
BTFSC _c
GOTO BLUM3131NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3131B
MOVF LOOPCOUNT
LOOPUM3131C
RRF BARGB1
BTFSC _c
GOTO CLUM3131NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3131C
MOVLW Ox7
MOVF LOOPCOUNT
LOOPUM3131D
RRF BARGBO
BTFSC _c
GOTO DLUM3131NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3131D
CLRF AARGBO
CLRF AARGB1
CLRF AARGB2
CLRF AARGB3
RETLW 0x00
ALUM3131NAP BCF _c
GOTO ALUM3131NA
BLUM3131NAP BCF _c
GOTO BLUM3131NA
CLUM3131NAP BCF _c
GOTO CLUM3131NA
DLUM3131NAP BCF _c
GOTO DLUM3131NA
ALOOPUM3131 BRF BARGB3
BTFSS _c
GOTO ALUM3131NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

ALUM3131NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO ALOOPUM3131
MOVLW 0x08
MOVWF LOOPCOUNT

ALUM3131NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO ALOOPUM3131
MOVLW 0x08
MOVWF LOOPCOUNT

BLOOPUM3131
RRF BARGB2
BTFSS _C
GOTO BLUM3131NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

BLOOPUM3131
RRF BARGB2
BTFSS _C
GOTO BLUM3131NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

CLOOPUM3131
RRF BARGB1
BTFSS _C
GOTO CLUM3131NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDF W ACCB1
MOVF TEMPB0,W
BTFSC __C
INCFSZ TEMPB0,W
ADDF W ACCB0

CLUM3131NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
DECFSZ LOOPCOUNT
GOTO CLOOPUM3131
MOV LW 0x07
MOV W LOOPCOUNT

DLOOPUM3131
RRF BARGB0
BTFSS __C
GOTO DLUM3131NA
MOVF TEMPB3,W
ADDF W ACCB3
MOVF TEMPB2,W
BTFSC __C
INCFSZ TEMPB2,W
ADDF W ACCB2
MOVF TEMPB1,W
BTFSC __C
INCFSZ TEMPB1,W
ADDF W ACCB1
MOVF TEMPB0,W
BTFSC __C
INCFSZ TEMPB0,W
ADDF W ACCB0

DLUM3131NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
DECFSZ LOOPCOUNT
GOTO DLOOPUM3131
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7

SMUL3232 macro
; Max Timing: 9*7*22+8*23+8*24+7*25+9 = 723 clks
; Min Timing: 62+6 = 68 clks
; FM: 68+6+7*22+8*23+8*24+7*25+9 = 788

variable i
i = 0

while i < 8
BTFSC  BARGB3,i
GOTO  SM3232NA$v(i)
i = i + 1
endw

i = 8
while i < 16
BTFSC  BARGB2,i-8
GOTO  SM3232NA$v(i)
i = i + 1
endw

i = 16
while i < 24
BTFSC  BARGB1,i-16
GOTO  SM3232NA$v(i)
i = i + 1
endw

i = 24
while i < 31
BTFSC  BARGB0,i-24
GOTO  SM3232NA$v(i)
i = i + 1
endw

CLRFB  ACCBO
; if we get here, BARG = 0
CLRFB  ACCB1
CLRFB  ACCB2
CLRFB  ACCB3
RETFW  0

SM3232NA0
RLF  TEMPBO,W
RRF  ACCBO
RRF  ACCB1
RRF  ACCB2
RRF  ACCB3
RRF  ACCB4
i = 1
while i < 8
BTFSS  BARGB3,i
GOTO  SM3232NA$v(i)
SM3232A$v(i)
MOVF  TEMPB3,W
ADDWF  ACCB3
MOVF  TEMPB2,W
BTFSC  _C
INCFSZ  TEMPB2,W
ADDWF  ACCB2
MOVF  TEMPB1,W
BTFSC  _C
INCFSZ  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPB0,W
BTFSC  _C
INCFSZ  TEMPB0,W
ADDWF  ACCBO

SM3232NA$v(i)
RLF  TEMPB0,W
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
RRF  ACCB3
RRF  ACCB4
i = i + 1
endw

i = 8
while \( i < 16 \)
\[
\begin{align*}
\text{BTFSS} & \quad \text{BARGB2,} i-8 \\
\text{GOTO} & \quad \text{SM3232A#v(i)} \\
\text{MOVF} & \quad \text{TEMPB3,} W \\
\text{ADDFW} & \quad \text{ACCB3} \\
\text{MOVF} & \quad \text{TEMPB2,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB2,} W \\
\text{ADDFW} & \quad \text{ACCB2} \\
\text{MOVF} & \quad \text{TEMPB1,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB1,} W \\
\text{ADDFW} & \quad \text{ACCB1} \\
\text{MOVF} & \quad \text{TEMPB0,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB0,} W \\
\text{ADDFW} & \quad \text{ACCB0} \\
\text{SM3232A#v(i)} \\
\text{RLF} & \quad \text{TEMPB0,} W \\
\text{R寻} & \quad \text{ACCB0} \\
\text{RFF} & \quad \text{ACCB1} \\
\text{RFF} & \quad \text{ACCB2} \\
\text{RFF} & \quad \text{ACCB3} \\
\text{RFF} & \quad \text{ACCB4} \\
\text{RFF} & \quad \text{ACCB5} \\
i = i + 1 \\
\text{endw}
\end{align*}
\]
\( i = 16 \)
while \( i < 24 \)
\[
\begin{align*}
\text{BTFSS} & \quad \text{BARGB1,} i-16 \\
\text{GOTO} & \quad \text{SM3232A#v(i)} \\
\text{MOVF} & \quad \text{TEMPB3,} W \\
\text{ADDFW} & \quad \text{ACCB3} \\
\text{MOVF} & \quad \text{TEMPB2,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB2,} W \\
\text{ADDFW} & \quad \text{ACCB2} \\
\text{MOVF} & \quad \text{TEMPB1,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB1,} W \\
\text{ADDFW} & \quad \text{ACCB1} \\
\text{MOVF} & \quad \text{TEMPB0,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB0,} W \\
\text{ADDFW} & \quad \text{ACCB0} \\
\text{SM3232A#v(i)} \\
\text{RLF} & \quad \text{TEMPB0,} W \\
\text{RFF} & \quad \text{ACCB0} \\
\text{RFF} & \quad \text{ACCB1} \\
\text{RFF} & \quad \text{ACCB2} \\
\text{RFF} & \quad \text{ACCB3} \\
\text{RFF} & \quad \text{ACCB4} \\
\text{RFF} & \quad \text{ACCB5} \\
i = i + 1 \\
\text{endw}
\end{align*}
\]
\( i = 24 \)
while \( i < 31 \)
\[
\begin{align*}
\text{BTFSS} & \quad \text{BARGB0,} i-24 \\
\text{GOTO} & \quad \text{SM3232A#v(i)} \\
\text{MOVF} & \quad \text{TEMPB3,} W \\
\text{ADDFW} & \quad \text{ACCB3} \\
\text{MOVF} & \quad \text{TEMPB2,} W \\
\text{BTFSC} & \quad _C \\
\text{INCFSZ} & \quad \text{TEMPB2,} W \\
\text{ADDFW} & \quad \text{ACCB2} \\
\text{MOVF} & \quad \text{TEMPB1,} W \\
\text{BTFSC} & \quad _C
\end{align*}
\]
INCFSZ TEMPBl,W
ADDWF ACCB1
MOVF TEMPBO,W
BTFSC _C
INCFSZ TEMPBO,W
ADDWF ACCB0
SM3232NA#v(i) RLF TEMPBO,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
i = i + 1
endw
RLF TEMPBO,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
endm

UMUL3232 macro
; Max Timing: 9+8*21+8*22+8*23+8*24 = 729 clks
; Min Timing: 63+6 = 69 clks
; DM: 69+6+8*21+8*22+8*23+8*24 = 795
variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 8
BTFSC BARGB3,i
GOTO UM3232NA#v(i)
i = i + 1
endw
i = 8
while i < 16
BTFSC BARGB2,i-8
GOTO UM3232NA#v(i)
i = i + 1
endw
i = 16
while i < 24
BTFSC BARGB1,i-16
GOTO UM3232NA#v(i)
i = i + 1
endw
i = 24
while i < 32
BTFSC BARGB0,i-24
GOTO UM3232NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
```assembly
CLRPF ACCB1
CLRPF ACCB2
CLRPF ACCB3
RETEW 0
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = 1
while i < 8
BTFSS BARGB3,i
GOTO UM3232NA#v(i)
UM3232A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
BTFSC _C
INCFSZ TEMPB3,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
UM3232NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGB2,i-8
GOTO UM3232NA#v(i)
UM3232A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
UM3232NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
i = i + 1
endw
i = 16
while i < 24
BTFSS BARGB1,i-16
GOTO UM3232NA#v(i)
UM3232A#v(i)
MOVF TEMPB3,W
ADDF W ACCB3
```
UM3232A#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6

while i < 32
BTFSS BARGBO,i-24
GOTO UM3232NA#v(i)

UM3131 macro
; Max Timing: 9+7*21+8*22+8*23+7*24+9 = 693 clks
; Min Timing: 62+6 = 68 clks
; PM: 68+6+7*22+8*23+8*24+7*25+9 = 788 DM: 16

variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 8
BTFSS BARGB3,i
GOTO UM3131NA#v(i)

endw

UM3232A#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7

endw

UMUL3131

; Max Timing: 9+7*21+8*22+8*23+7*24+9 = 693 clks
; Min Timing: 62+6 = 68 clks
; PM: 68+6+7*22+8*23+8*24+7*25+9 = 788 DM: 16

variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 8
BTFSS BARGB3,i
GOTO UM3131NA#v(i)

endw

i = i + 1

endw

i = 24
while i < 32
BTFSS BARGBO,i-24
GOTO UM3232NA#v(i)

UM3232A#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7

i = i + 1
endw

UM3232A#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
BTFSC BARGB2, i-8
GOTO UM3131NA#v(i)
i = i + 1
endw
i = 16
while i < 24
BTFSC BARGBl, i-16
GOTO UM3131NA#v(i)
i = i + 1
endw
i = 24
while i < 31
BTFSC BARGBO, i-24
GOTO UM3131NA#v(i)
i = i + 1
endw
CLRF ACCBO
if we get here, BARG = 0
CLRF ACCBl
CLRF ACCB2
CLRF ACCB3
RETFEW 0
UM3131NA0
RRF ACCBO
RRF ACCBl
RRF ACCB2
RRF ACCB3
i = 1
while i < 8
BTFSS BARGB3, i
GOTO UM3131NA#v(i)
MOVF TEMPB3, W
ADDWF ACCB3
MOVF TEMPB2, W
BTFSS _C
INCFSZ TEMPB2, W
ADDWF ACCB2
MOVF TEMPBl, W
BTFSS _C
INCFSZ TEMPBl, W
ADDWF ACCB1
MOVF TEMPO, W
BTFSS _C
INCFSZ TEMPO, W
ADDWF ACCB0
UM3131A#v(i)
RRF ACCB0
RRF ACCBl
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGB2, i-8
GOTO UM3131NA#v(i)
MOVF TEMPB3, W
ADDWF ACCB3
MOVF TEMPB2, W
BTFSS _C
INCFSZ TEMPB2, W
ADDWF ACCB2
MOVF TEMPO, W
BTFSS _C
INCFSZ TEMPO, W

© 1995 Microchip Technology Inc. DS00617A-page 2-179
ADDWF ACCB1
MOVF TEMPB0,W
BTFS  _c
INCF    TEMPB0,W
ADDWF ACCB0
UM3131NA#v(i)  RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
i = i + 1
endw
i = 16
while  i < 24
BTFS  BARGB1,i-16
GOTO UM3131NA#v(i)
UM3131A#v(i)  MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFS  _c
INCF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFS  _c
INCF TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFS  _c
INCF TEMPB0,W
ADDWF ACCB0
UM3131NA#v(i)  RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
i = i + 1
endw
i = 24
while  i < 31
BTFS  BARGB0,i-24
GOTO UM3131NA#v(i)
UM3131A#v(i)  MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFS  _c
INCF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFS  _c
INCF TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFS  _c
INCF TEMPB0,W
ADDWF ACCB0
UM3131NA#v(i)  RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
\[ i = i + 1 \]
endw

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
RRF ACCB7
endm

;********************************************************************************
; 32x32 Bit Signed Fixed Point Multiply 32x32 -> 64
; Input: 32 bit signed fixed point multiplicand in AARGBO
; 32 bit signed fixed point multiplier in BARGB0
; Use: CALL FXM3232S
; Output: 64 bit signed fixed point product in AARGBO
; Result: AARG <- AARG x BARG
; Max Timing: \( 15 \times 851 + 2 = 868 \) clks \( B > 0 \)
; \( 36 \times 851 + 2 = 889 \) clks \( B < 0 \)
; Min Timing: \( 15 \times 192 = 207 \) clks
; DM: 17

FXM3232S BTFS ACCB0, MSB
GOTO M3232SOK
COMP BARGB3
INC Z
BTFS BARGB2
DECF BARGB2
BTFS _Z
DECF BARGB1
COMP BARGB1
BTFS _Z
DECF BARGB0
COMP BARGB0
INCF AARGB3
BTFS AARGB2
DECF AARGB2
BTFS _Z
DECF AARGB1
COMP AARGB1
BTFS _Z
DECF AARGB0
COMP AARGB0

M3232SOK CLRF ACCB4
CLRF ACCB5
CLRF ACCB6
MOVF AARGB0, W
MOVWF TEMPB0
MOVF AARGB1, W
MOVWF TEMPB1
MOVF AARGB2, W
MOVWF TEMPB2
MOVF AARGB3, W
MOVWF TEMPB3
SMUL3232L
RETLW 0x00

;********************************************************************************
; 32x32 Bit Unsigned Fixed Point Multiply 32x32 -> 64
; Input: 32 bit unsigned fixed point multiplicand in AARGBO

© 1995 Microchip Technology Inc.
32 bit unsigned fixed point multiplier in BARGB0

Use: CALL FXM3232U
Output: 64 bit unsigned fixed point product in AARGB0
Result: AARG <-- AARG x BARG
Max Timing: 12+842+2 = 856 clks
Min Timing: 12+197 = 209 clks
PM: 12+155+1 = 168

FXM3232U
CLRFB ACCB4 ; clear partial product
CLRFB ACCB5
CLRFB ACCB7
CLRFB ACCB6
MOVF AARGBO,W
MOVF TEMPBO
MOVF AARGB1,W
MOVF TEMPB1
MOVF AARGB2,W
MOVF TEMPB2
MOVF AARGB3,W
MOVF TEMPB3
UMUL3232L
RETLW 0x00

31x31 Bit Unsigned Fixed Point Divide 31x31 -> 62
Input: 31 bit unsigned fixed point multiplicand in AARGBO
Use: CALL FXM3131U
Output: 62 bit unsigned fixed point product in AARGB0
Result: AARG <-- AARG x BARG
Max Timing: 12+822+2 = 836 clks
Min Timing: 12+192 = 204 clks
PM: 12+155+1 = 168

FXM3131U
CLRFB ACCB4 ; clear partial product
CLRFB ACCB5
CLRFB ACCB7
CLRFB ACCB6
MOVF AARGBO,W
MOVF TEMPBO
MOVF AARGB1,W
MOVF TEMPB1
MOVF AARGB2,W
MOVF TEMPB2
MOVF AARGB3,W
MOVF TEMPB3
UMUL3131L
RETLW 0x00

32x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

32x24 PIC16 FIXED POINT MULTIPLY ROUTINES

Input: fixed point arguments in AARG and BARG
Output: product AARG x BARG in AARG
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two argument can be called in a signed multiply application where it is known that the respective argument is nonnegative, thereby offering some improvement in performance.

Routine Clocks Function
FXM3224S 652 32x24 -> 56 bit signed fixed point multiply
FXM3224U 630 32x24 -> 56 bit unsigned fixed point multiply
FXM3123U 610 31x23 -> 54 bit unsigned fixed point multiply
The above timings are based on the looped macros. If space permits, approximately 80-97 clocks can be saved by using the unrolled macros.

```assembly
list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

; Test suite storage
RANDHI equ OxlA ; random number generator registers
RANDLO equ OxlB
DATA equ 0x20 ; beginning of test data

; Test suite for 32x24 bit fixed point multiply algorithms
org 0x0005
MAIN
MOVLW RAMSTART
MOVWF FSR
MEMLOOP
CLRF INDF
INCRC FSR
MOVLW RAMSTOP
SUBWF FSR,W
BTFSS _Z
GOTO MEMLOOP
MOVLW 0x45 ; seed for random numbers
MOVWF RANDLO
MOVLW 0x30
MOVWF RANDHI
MOVLW DATA
MOVWF FSR
BCF _RP0
BCF _RP1
BCF _IRP
CALL TFXM3224
SELF
GOTO SELF
RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMPB0
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPB1
RLF TEMPB1,W
RLF TEMPB1
MOVF TEMPB1,W
XORWF RANDHI,W
SWAPF RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETELW 0

; Test suite for FXM3224
TFXM3224
CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGB0
BCF BARGB0,MSB
MOVF BARGB0,W
MOVWF INDF
INCFC FSR
MOVF RANDLO,W
MOVWF BARGB1
MOVWF INDF
INCFC FSR
CALL RANDOM16
```

© 1995 Microchip Technology Inc.
MOVF RANDHI, W
MOVF RANDLO, W
MOVF RANDHI, W
MOVF RANDLO, W
MOVF RANDHI, W
MOVF RANDLO, W
MOVLW 0x8
MOVWF LOOPCOUNT

; 32x24 Bit Multiplication Macros
SMUL3224L macro
; Max Timing:     2+13+6*26+25+2+7*27+26+2+6*28+27+2+6*29+28+6+2+6*30+29+8 = 618 clks
; Min Timing:     2+7*6+5+1+7*6+5+2+6*6+5+6 = 146 clks
; PM: 25+25+2+26+2+27+2+8 = 115
; DM: 15
MOVLW 0x8
MOVWF LOOPCOUNT

LOOPSM3224A
RRF BARGBl
BTFSC _C
GOTO ALSM3224NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3224A
MOVWF LOOPCOUNT

LOOPSM3224B
RRF BARGB2
BTFSC _C
AN617

GOTO BLSM3224NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3224B
MOVLW 0x7
MOVWF LOOPCOUNT

LOOPSM3224C
RRF BARGBO
BTFSC _C
GOTO CLSM3224NA
DECFSZ LOOPCOUNT
GOTO LOOPSM3224C
CLRF AARGB0
CLRF AARGB1
CLRF AARGB2
CLRF AARGB3
RETLW 0x00

ALOOPSM3224
RRF BARGB2
BTFSS _C
GOTO ALSM3224NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

ALSM3224NA
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO ALOOPSM3224
MOVLW 0x8
MOVWF LOOPCOUNT

BLOOPSM3224
RRF BARGB1
BTFSS _C
GOTO BLSM3224NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

BLSM3224NA
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2

© 1995 Microchip Technology Inc.
AN617

```assembly
    RRF   ACCB3
    RRF   ACCB4
    RRF   ACCB5
    DECFSZ LOOPCOUNT
    GOTO   BLOOPSM3224
    MOVLW   0x7
    MOVWF   LOOPCOUNT

CLOOPSM3224
    RRF   BARGB0
    BTFSS    _c
    GOTO   CLSM3224NA
    MOVF   TEMPB3,W
    ADDWF   ACCB3
    MOVF   TEMPB2,W
    BTFSC    _c
    INCFSZ   TEMPB2,W
    ADDWF   ACCB2
    MOVF   TEMPB1+W
    BTFSC    _c
    INCFSZ   TEMPB1,W
    ADDWF   ACCB1
    MOVF   TEMPB0,W
    BTFSC    _c
    INCFSZ   TEMPB0,W
    ADDWF   ACCB0
    RLF   TEMPB0,W
    RRF   ACCB0
    RRF   ACCB1
    RRF   ACCB2
    RRF   ACCB3
    RRF   ACCB4
    RRF   ACCB5
    RRF   ACCB6
    DECFSZ   LOOPCOUNT
    GOTO   CLOOPSM3224

CLSM3224NA
    RLF   TEMPB0,W
    RRF   ACCB0
    RRF   ACCB1
    RRF   ACCB2
    RRF   ACCB3
    RRF   ACCB4
    RRF   ACCB5
    RRF   ACCB6
    DECFSZ   LOOPCOUNT
    GOTO   CLOOPSM3224

UMUL3224L
    macro
; Max Timing: \(2+15+6*25+24+2+7*26+25+2+7*27+26 = 617\) clks
; Min Timing: \(2+7*6+5+1+7*6+5+1+7*6+5+6 = 151\) clks
; PM: \(31+24+2+25+2+26+2+27 = 139\)    DM: 15
    MOVLW   0x08
    MOVWF   LOOPCOUNT

LOOPUM3224A
    RRF   BARGB2
    BTFSC    _c
    GOTO   ALUM3224NAP
    DECFSZ   LOOPCOUNT
    GOTO   LOOPUM3224A
    MOVWF   LOOPCOUNT

LOOPUM3224B
    RRF   BARGB1
    BTFSC    _c
    GOTO   BLUM3224NAP
    DECFSZ   LOOPCOUNT
    GOTO   LOOPUM3224B
    MOVWF   LOOPCOUNT

LOOPUM3224C
    RRF   BARGB0
```

Max Timing: \(2+15+6*25+24+2+7*26+25+2+7*27+26 = 617\) clks
Min Timing: \(2+7*6+5+1+7*6+5+1+7*6+5+6 = 151\) clks
PM: \(31+24+2+25+2+26+2+27 = 139\)    DM: 15
BTFSC _c  GOTO  CLUM3224NAP
DECFSZ LOOPCOUNT  GOTO  LOOPUM3224C
CLRF  AARGBO
CLRF  AARGBl
CLRF  AARGB2
CLRF  AARGB3
RETLW  0x00

ALUM3224NAP  BCF  _c  GOTO  ALUM3224NA

BLUM3224NAP  BCF  _c  GOTO  BLUM3224NA

CLUM3224NAP  BCF  _c  GOTO  CLUM3224NA

ALOOPUM3224  RRF  BARGB2
BTFSS  _c  GOTO  ALUM3224NA
MOVF  TEMPB3,W
ADDWF  ACCB3
MOVF  TEMPB2,W
BTFSC  _c
INCFSZ  TEMPB2,W
ADDWF  ACCB2
MOVF  TEMPB1,W
BTFSC  _c
INCFSZ  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPB0,W
BTFSC  _c
INCFSZ  TEMPB0,W
ADDWF  ACCB0

ALUM3224NA  RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
RRF  ACCB3
RRF  ACCB4
DECFSZ LOOPCOUNT  GOTO  ALOOPUM3224
MOVLW  0x08
MOVF  LOOPCOUNT

BLOOPUM3224  RRF  BARGB1
BTFSS  _c  GOTO  BLUM3224NA
MOVF  TEMPB3,W
ADDWF  ACCB3
MOVF  TEMPB2,W
BTFSC  _c
INCFSZ  TEMPB2,W
ADDWF  ACCB2
MOVF  TEMPB1,W
BTFSC  _c
INCFSZ  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPB0,W
BTFSC  _c
INCFSZ  TEMPB0,W
ADDWF  ACCB0

BLUM3224NA  RRF  ACCB0
AN617

RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
DECFSZ LOOPCOUNT
GOTO BLOOPUM3224
MOVLW 0x08
MOVF LOOPCOUNT

CLOOPUM3224
RRF BARGB0
BTFSS _c
GOTO CLUM3224NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0

CLUM3224NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
DECFSZ LOOPCOUNT
GOTO CLOOPUM3224

UMUL3123L macro
; Max Timing: 2+15+6*25+24+2+7*26+25+2+6*27+26+7 = 597 clks
; Min Timing: 2+7*6+5+1+7*6+5+2+6*6+5+6 = 146 clks
; PM: 31+24+2+25+2+26+7 = 117 DM: 15
MOVLW 0x08
MOVF LOOPCOUNT

LOOPUM3123A
RRF BARGB2
BTFSC _c
GOTO ALUM3123NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3123A

LOOPUM3123B
RRF BARGB1
BTFSC _c
GOTO BLUM3123NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3123B

LOOPUM3123C
RRF BARGB0
BTFSC _c
GOTO CLUM3123NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3123C
CLR PY
AN617

CLRF AARGB1
CLRF AARGB2
CLRF AARGB3
RETLW 0x00

ALUM3123NAP
BCF _c
GOTO ALUM3123NA

BLUM3123NAP
BCF _c
GOTO BLUM3123NA

CLUM3123NAP
BCF _c
GOTO CLUM3123NA

ALUM3123NA
RRF BARGB2
BTFSS _c
GOTO ALUM3123NA
MOVF TEMPB3,W
ADDFW ACCB3
MOVFB TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDFW ACCB2
MOVFB TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDFW ACCB1
MOVFB TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDFW ACCB0

BLUM3123NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO BLOOPUM3123
MOVLW 0x08
MOVWF LOOPCOUNT

BLOOPUM3123
RRF BARGB2
BTFSS _c
GOTO BLUM3123NA
MOVF TEMPB3,W
ADDFW ACCB3
MOVFB TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDFW ACCB2
MOVFB TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDFW ACCB1
MOVFB TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDFW ACCB0

BLUM3123NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
AN617

DECFSZ  LOOPCOUNT
GOTO   BLOOPUM3123
MOVLW  0x07
MOVWF  LOOPCOUNT

CLOOPUM3123
RRF     BARGBO
BTFSS   _C
GOTO   CLUM3123NA
MOVF   TEMPB3,W
ADDWF   ACCB3
MOVF   TEMPB2,W
BTFSC   _C
INCFSZ  TEMPB2,W
ADDWF   ACCB2
MOVF   TEMPB1,W
BTFSC   _C
INCFSZ  TEMPB1,W
ADDWF   ACCB1
MOVF   TEMPB0,W
BTFSC   _C
INCFSZ  TEMPB0,W
ADDWF   ACCB0

CLUM3123NA
RRF     ACCB0
RRF     ACCB1
RRF     ACCB2
RRF     ACCB3
RRF     ACCB4
RRF     ACCB5
RRF     ACCB6
DECFSZ  LOOPCOUNT
GOTO   CLOOPUM3123
RRF     ACCB0
RRF     ACCB1
RRF     ACCB2
RRF     ACCB3
RRF     ACCB4
RRF     ACCB5
RRF     ACCB6
endm

SMUL3224 macro
;   Max Timing:  9+7*22+8*23+7*24+8 = 23 clks
;   Min Timing:  40+6 = 46 clks
;   PM: 46+6=7+22+8*23+7*24+8 = 566                DM: 14

variable i
i = 0

while i < 8
BTFSC   BARGBO,i
GOTO   SM3224NA#$v(i)
i = i + 1
endw

i = 8

while i < 16
BTFSC   BARGBO,i-8
GOTO   SM3224NA#$v(i)
i = i + 1
endw

i = 16

while i < 23

DS00617A-page 2-190  © 1995 Microchip Technology Inc.
BTFSC BARGB0, i-16
GOTO SM3224NA#v(i)
i = i + 1
endw
CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETEW 0
SM3224NA0
RLF TEMPB0,W
RRF ACCBO
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = 1
while i < 8
BTFSS BARGB2, i
GOTO SM3224NA#v(i)
SM3224A#v(i)
MOVF TEMPB3,W
ADDFW ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDFW ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDFW ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDFW ACCB0
SM3224NA#v(i)
RLF TEMPB0,W
RRF ACCBO
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGB1, i-8
GOTO SM3224NA#v(i)
SM3224A#v(i)
MOVF TEMPB3,W
ADDFW ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDFW ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDFW ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDFW ACCB0
SM3224NA#v(i)
RLF TEMPB0,W
RRF ACCBO
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
i = i + 1 
endw 

while i < 23 
BTFS BARGB0,i-16 
GOTO SM3224NA#v(i) 
SM3224NA#v(i) 
MOV P TEMPB3,W 
ADDW ACCB3 
MOV P TEMPB2,W 
BTFS C 
INCFSZ TEMPB2,W 
ADDW ACCB2 
MOV P TEMPB1,W 
BTFS C 
INCFSZ TEMPB1,W 
ADDW ACCB1 
MOV P TEMPB0,W 
BTFS C 
INCFSZ TEMPB0,W 
ADDW ACCB0 
SM3224NA#v(i) 
RLF TEMPB0,W 
RRF ACCB0 
RRF ACCB1 
RRF ACCB2 
RRF ACCB3 
RRF ACCB4 
RRF ACCB5 
RRF ACCB6 
i = i + 1 
endw 
RLF TEMPB0,W 
RRF ACCB0 
RRF ACCB1 
RRF ACCB2 
RRF ACCB3 
RRF ACCB4 
RRF ACCB5 
RRF ACCB6 
endm 

UMUL3224 macro 
; Max Timing: 9+8*21+8*22+8*23 = 537 clks 
; Min Timing: 41+6 = 47 clks 
; PM: 47+6+8*21+8*22+8*23 = 581 
variable i 
i = 0 
BCF C ; clear carry for first right shift 
while i < 8 
BTFS BARGB2,i 
GOTO UM3224NA#v(i) 
i = i + 1 
endw 
i = 8 
while i < 16 
BTFS BARGB1,i-9 
GOTO UM3224NA#v(i) 
i = i + 1 
endw 
i = 16 
while i < 24 
BTFS BARGB0,i-16 

© 1995 Microchip Technology Inc.
GOTO UM3224NA#v(i)

i = i + 1
endw

CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETEW 0

UM3224NA0
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4

i = 1
while i < 8
BTFSS BARGB2,i
GOTO UM3224NA#v(i)

UM3224A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

UM3224NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4

i = i + 1
endw

i = 8
while i < 16
BTFSS BARGB1,i-8
GOTO UM3224NA#v(i)

UM3224A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

UM3224NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5

i = i + 1
endw

i = 16
while i < 24
UM3224A#v(i)

MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0

UM3224NA#v(i)

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6

i = i + 1
endw
endm

UMUL3123 macro

; Max Timing: 9*7+21+8*22+7*23+7 = 500 clks
; Min Timing: 41+6 = 47 clks
; DM: 14

variable i

i = 0
BCF _c ; clear carry for first right shift
while i < 8

BTFS BARGB2,i
GOTO UM3123NA#v(i)
i = i + 1
endw

i = 8
while i < 16

BTFS BARGB1,i-8
GOTO UM3123NA#v(i)
i = i + 1
endw

i = 16
while i < 23

BTFS BARGB0,i-16
GOTO UM3123NA#v(i)
i = i + 1
endw

CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETEW 0

UM3123NA0

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4

i = 1
while i < 8

BTFS BARGB2,i
GOTO UM3123A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0

UM3123NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4

i = i + 1
endw

while i < 16
BTFSS BARGB1,i-8
GOTO UM3123A#v(i)

UM3123A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0

UM3123NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5

i = i + 1
endw

i = 16
while i < 23
BTFSS BARGB0,i-16
GOTO UM3123A#v(i)

UM3123A#v(i)
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSC _c
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _c
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0
UM3123NA\#v(i) RRF
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
i = i + 1
endw
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
RRF ACCB6
endm

;********************************************************************************************
;*********************************************************************************************
32x24 Bit Signed Fixed Point Multiply 32x24 -> 56
Input: 32 bit signed fixed point multiplicand in AARGBO, AARGBl, AARGBO2, AARGBO3
24 bit signed fixed point multiplier in BARGBO, BARGBl, BARGB2
Use: CALL FXM3224S
Output: 56 bit signed fixed point product in AARGBO
Result: AARG <- AARG x BARG
Max Timing: 14+618+2 = 634 clks B > 0
32+618+2 = 652 clks B < 0
Min Timing: 14+146 = 160 clks
PM: 36+115+1 = 152 DM: 15

FXM3224S
BTFSS BARGBO,MSB
GOTO M3224SOK
; make multiplier BARG > 0
COMF BARGBO2
INCf BARGBO2
BTFSc _Z
DECf BARGBO1
COMF BARGBO1
BTFSc _Z
DECf BARGBO0
COMF BARGBO0
COMF AARGBO3
INCf AARGBO3
BTFSc _Z
DECf AARGBO2
COMF AARGBO2
BTFSc _Z
DECf AARGBO1
COMF AARGBO1
BTFSc _Z
DECf AARGBO0
COMF AARGBO0
M3224SOK
CLRF ACCBO4
; clear partial product
CLRF ACCBO5
CLRF ACCBO6
MOVf AARGBO,W
MOVWF TEMPBO
MOVf AARGBO1,W
MOVWF TEMPBI
MOVf AARGBO2,W
MOVWF TEMPB2
MOVf AARGBO3,W
MOVWF TEMPB3
SMUL3224L
C.3 32x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

32x16 PIC16 FIXED POINT MULTIPLY ROUTINES VERSION 1.2

Input: fixed point arguments in AARG and BARG
Output: product AARGxBARG in AARG
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two argument can be called in a signed multiply application where it is known that the respective argument is nonnegative, thereby offering some improvement in performance.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Clocks</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXM3216S</td>
<td>423</td>
<td>32x16 -&gt; 48 bit signed fixed point multiply</td>
</tr>
<tr>
<td>FXM3216U</td>
<td>412</td>
<td>32x16 -&gt; 48 bit unsigned fixed point multiply</td>
</tr>
<tr>
<td>FXM3115U</td>
<td>392</td>
<td>31x15 -&gt; 46 bit unsigned fixed point multiply</td>
</tr>
</tbody>
</table>

The above timings are based on the looped macros. If space permits, approximately 65-88 clocks can be saved by using the unrolled macros.

```assembly
; Test suite storage
RANDHI equ 0x1A ; random number generator registers
RANDLO equ 0x1B
TESTCOUNT equ 0x20 ; counter
DATA equ 0x21 ; beginning of test data

; Test suite for 32x16 bit fixed point multiply algorithms
org 0x0005
MAIN
    MOVFW RAMSTART
    MOVWF FSR
MEMLOOP
    CLRWF INDF
    INCWF FSR
    MOVLF RAMSTOP
    SUBWF FSR, W
    BTFSS _Z
    GOTO MEMLOOP
    MOVWF 0x45       ; seed for random numbers
    MOVWF RANDLO
    MOVWF 0x30
    MOVWF RANDHI
    MOVWF DATA
    MOVWF FSR
    BCF _RP0
    BCF _RP1
    BCF _IRP
    CALL TFXM3216
    GOTO SELF
SELF
    CALL TFXM3216
    GOTO SELF
RANDOM16
    RLF RANDHI, W  ; random number generator
    XORWF RANDHI, W
    MOVWF TEMPB0
    SWAPF RANDHI
    SWAPF RANDLO, W
    MOVWF TEMPB1
    RLF TEMPB1, W
    RLF TEMPB1
    MOVF TEMPB1, W
    XORWF RANDHI, W
    SWAPF RANDHI
    ANDLW 0x01
    RLF TEMPB0
    RLF RANDLO
    XORWF RANDLO
    RLF RANDHI
    RETEW 0

; Test suite for FXM3216
TFXM3216
    MOVFW 1
    MOVWF TESTCOUNT
M3216LOOP
    CALL RANDOM16
```

© 1995 Microchip Technology Inc.
MOVF RANDHI, W
MOVWF BARGBO
BCF BARGBO, MSB
MOVF BARGBO, W
MOVWF INDF
INCF FSR
MOVF RANDLO, W
MOVWF BARG Bl
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI, W
MOVWF AARGBO
BCF AARGBO, MSB
MOVF AARGBO, W
MOVWF INDF
INCF FSR
MOVF RANDLO, W
MOVWF AARGB3
MOVWF INDF
INCF FSR
CALL FXM3115U
MOVF AARGB0, W
MOVWF INDF
INCF FSR
MOVF AARGB1, W
MOVWF INDF
INCF FSR
MOVF AARGB2, W
MOVWF INDF
INCF FSR
MOVF AARGB3, W
MOVWF INDF
INCF FSR
MOVF AARGB4, W
MOVWF INDF
INCF FSR
MOVF AARGB5, W
MOVWF INDF
INCF FSR
DECFSZ TESTCOUNT
GOTO M3216LOOP
RETLW Ox00

;******************************************************************************************
;******************************************************************************************
32x16 Bit Multiplication Macros

SMUL3216L  macro
;  Max Timing:  2+13+6*26+25+2+6*27+26+7 = 393 clks
;  Min Timing:  2+7*6+5+2+6*6+5+6 = 98 clks
;  PM: 19+60 = 79  DM: 11
MOVLW 0xB
MOVWF LOOPCOUNT
LOOPSM3216A
RRF _c
BTFSC _C
GOTO ALSM3216NA
DECFSZ LOOPCOUNT

© 1995 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOTO</td>
<td>LOOPSM3216A</td>
</tr>
<tr>
<td>MOVWF</td>
<td>LOOPCOUNT</td>
</tr>
<tr>
<td>RRF</td>
<td>BARGBO</td>
</tr>
<tr>
<td>BTFSC</td>
<td>_c</td>
</tr>
<tr>
<td>GOTO</td>
<td>BLSM3216NA</td>
</tr>
<tr>
<td>DECFSZ</td>
<td>LOOPCOUNT</td>
</tr>
<tr>
<td>GOTO</td>
<td>LOOPSM3216B</td>
</tr>
<tr>
<td>CLRF</td>
<td>AARGBO</td>
</tr>
<tr>
<td>CLRF</td>
<td>AARGB1</td>
</tr>
<tr>
<td>CLRF</td>
<td>AARGB2</td>
</tr>
<tr>
<td>RETLW</td>
<td>0x00</td>
</tr>
<tr>
<td>RLF</td>
<td>TEMPB0,W</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB4</td>
</tr>
<tr>
<td>DECFSZ</td>
<td>LOOPCOUNT</td>
</tr>
<tr>
<td>GOTO</td>
<td>ALOOPSM3216</td>
</tr>
<tr>
<td>MOVWF</td>
<td>LOOPCOUNT</td>
</tr>
<tr>
<td>RRF</td>
<td>BARGBO</td>
</tr>
<tr>
<td>BTFSS</td>
<td>_c</td>
</tr>
<tr>
<td>GOTO</td>
<td>BLSM3216NA</td>
</tr>
<tr>
<td>MOVF</td>
<td>TEMPB3,W</td>
</tr>
<tr>
<td>ADDWF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>MOVF</td>
<td>TEMPB2,W</td>
</tr>
<tr>
<td>BTFSC</td>
<td>_c</td>
</tr>
<tr>
<td>INCFSZ</td>
<td>TEMPB2,W</td>
</tr>
<tr>
<td>ADDWF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>MOVF</td>
<td>TEMPB1,W</td>
</tr>
<tr>
<td>BTSSC</td>
<td>_c</td>
</tr>
<tr>
<td>INCFSZ</td>
<td>TEMPB1,W</td>
</tr>
<tr>
<td>ADDWF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>MOVF</td>
<td>TEMPB0,W</td>
</tr>
<tr>
<td>BTSSC</td>
<td>_c</td>
</tr>
<tr>
<td>INCFSZ</td>
<td>TEMPB0,W</td>
</tr>
<tr>
<td>ADDWF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RLF</td>
<td>TEMPB0,W</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB4</td>
</tr>
<tr>
<td>RLF</td>
<td>TEMPB0,W</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>RRF</td>
<td>ACCB4</td>
</tr>
</tbody>
</table>
RRF       ACCB5
DECFSZ    LOOPCOUNT
GOTO     BLOOPS3216
RLF       TEMPB0,W
RRF       ACCB0
RRF       ACCB1
RRF       ACCB2
RRF       ACCB3
RRF       ACCB4
RRF       ACCB5
endm

UMUL3216L
macro

Max Timing: 2*15+6*25+24+2*7*26+25 = 400 clks
Min Timing: 2*7*6+5*1+7*6+5+6 = 103 clks
PM: 73 DM: 11

MOVLW 0x08
MOVWF LOOPCOUNT

LOOPUM3216A
RRF       BARGBl
BTFSC _C
GOTO    ALUM3216NAP
DECFSZ    LOOPCOUNT
GOTO     LOOPUM3216A
MOVWF    LOOPCOUNT

LOOPUM3216B
RRF       BARGBO
BTFSC _C
GOTO    BLUM3216NAP
DECFSZ    LOOPCOUNT
GOTO     LOOPUM3216B
CLR       AARGB0
CLR       AARGB1
CLR       AARGB2
CLR       AARGB3
RETLW 0x00

BLUM3216NAP
BCF _C
GOTO    BLUM3216NA

ALUM3216NAP
BCF _C
GOTO    ALUM3216NA

ALOOPUM3216
RRF       BARGB1
BTFSS _C
GOTO    ALUM3216NA
MOVF      TEMPB3,W
ADDWF     ACCB3
MOVF      TEMPB2,W
BTFSS _C
INCFSZ    TEMPB2,W
ADDWF     ACCB2
MOVF      TEMPB1,W
BTFSS _C
INCFSZ    TEMPB1,W
ADDWF     ACCB1
MOVF      TEMPB0,W
BTFSS _C
INCFSZ    TEMPB0,W
ADDWF     ACCB0

ALUM3216NA
RRF       ACCB0
RRF       ACCB1
RRF       ACCB2
RRF       ACCB3
RRF       ACCB4
DECFSZ    LOOPCOUNT
GOTO ALOOPUM3216
MOVLW 0x08
MOVF LOOPCOUNT

BLOOPUM3216
RRF BARGBO
BTFSS _C
GOTO BLUM3216NA
MOVF TEMPB3,W
ADDWF ACCB3
MOVF TEMPB2,W
BTFSS _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSS _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSS _C
INCFSZ TEMPB0,W
ADDWF ACCB0

BLUM3216NA
RRF ACCBO
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
DECFSZ LOOPCOUNT
GOTO BLOOPUM3216
endm

UMUL3115L macro
; Max Timing: 2+15+6*25+24+2+6*26+25+6 = 380 clks
; Min Timing: 2+7*6+5+2+6*6+5+6 = 96 clks
; PM: 80  DM: 11
MOVLW 0x8
MOVF LOOPCOUNT

LOOPUM3115A
RRF BARGB1
BTFSS _C
GOTO ALUM3115NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3115A
MOVLW 0x7
MOVF LOOPCOUNT

LOOPUM3115B
RRF BARGBO
BTFSS _C
GOTO BLUM3115NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM3115A
CLRF AARGB0
CLRF AARGB1
CLRF AARGB2
CLRF AARGB3
RETLW 0x00

BLUM3115NAP
BCF _C
GOTO BLUM3115NA

ALUM3115NAP
BCF _C
GOTO ALUM3115NA

ALOOPUM3115
RRF BARGB1
BTFSS _C
GOTO ALUM3115NA
AN617

MOVF TEMPB3,W
ADDFP ACCB3
MOVF TEMPB2,W
BTFS C _c
INCFPS TEMPB2,W
ADDFP ACCB2
MOVF TEMPB1,W
BTFS C _c
INCFPS TEMPB1,W
ADDFP ACCB1
MOVF TEMPB0,W
BTFS C _c
INCFPS TEMPB0,W
ADDFP ACCB0

ALUM3115NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO ALOOPUM3115
MOVLW 0x07
MOVF ACCB0

BLOOPUM3115
RRF BARGB0
BTFS S _c
GOTO BLUM3115NA
MOVF TEMPB3,W
ADDFP ACCB3
MOVF TEMPB2,W
BTFS C _c
INCFPS TEMPB2,W
ADDFP ACCB2
MOVF TEMPB1,W
BTFS C _c
INCFPS TEMPB1,W
ADDFP ACCB1
MOVF TEMPB0,W
BTFS C _c
INCFPS TEMPB0,W
ADDFP ACCB0

BLUM3115NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
DECFSZ LOOPCOUNT
GOTO BLOOPUM3115
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
endm

SMUL3216 macro
; Max Timing: 5+8+7*20+7*21+5 = 305 clks
; Min Timing: 5+24+21+7 = 57 clks
; PM: 5+24+21+6+5+20+7*21+5 = 353 DM: 10
variable i
i = 0
BTFS C AARGBO,MSB
while i < 8
  BTFSC BARGB1,i
  GOTO SM3216NA#v(i)
  BCF ACCB4,7-i
  i = i + 1
endw
i = 8

while i < 15
  BTFSC BARGB0,i-8
  GOTO SM3216NA#v(i)
  BCF ACCB5,15-i
  i = i + 1
endw

CLRFR ACCB0 ; if we get here, BARG = 0
CLRFR ACCB1
CLRFR ACCB2
CLRFR ACCB3
CLRFR ACCB5
RETEW 0

SM3216NA0

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = 1
while i < 8
  BTFSS BARGB1,i
  GOTO SM3216NA#v(i)
SM3216A#v(i)
  MOVF TEMPB3,W
  ADDWF ACCB3
  MOVF TEMPB2,W
  BTFSC _C
  INCFSZ TEMPB2,W
  ADDWF ACCB2
  MOVF TEMPB1,W
  BTFSC _C
  INCFSZ TEMPB1,W
  ADDWF ACCB1
  MOVF TEMPB0,W
  BTFSS _C
  INCFSZ TEMPB0,W
  ADDWF ACCB0
SM3216NA#v(i)

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 8
while i < 15
  BTFSS BARGB0,i-8
  GOTO SM3216NA#v(i)
SM3216A#v(i)
  MOVF TEMPB3,W
  ADDWF ACCB3
  MOVF TEMPB2,W

AN617

SM3216NA\#v(i)

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5

i = i + 1
endw

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5

dem

UMUL3216

; Max Timing: 1+8+7*21+8*22 = 332 clks
; Min Timing: 1+2*8+2*8+6 = 39 clks
; DM: 10

variable i

i = 0

BCF _C ; clear carry for first right shift

while i < 8

BTFSC BARGB1,i
GOTO UM3216NA\#v(i)
i = i + 1
endw

i = 8

while i < 16

BTFSC BARGB0,i-8
GOTO UM3216NA\#v(i)
i = i + 1
endw

CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETEW 0

UM3216NA0

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4

i = 1

while i < 8

BTFSS BARGB1,i
GOTO UM3216NA\#v(i)

UM3216A\#v(i)

MOV BARGB3,W
ADDWF ACCB3
BTFSC _C
INCFSZ TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
MOVF TEMPB3,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGB0,i-8
GOTO UM3216NA#v(i)
MOVF TEMPB3,W
RRF ACCB3
RRF ACCB2
RRF ACCB1
RRF ACCB0
i = i + 1
endw
i = 8
while i < 16
BTFSC BARGB1,i
GOTO UM3215NA#v(i)
MOVF TEMPB2,W
RRF ACCB2
RRF ACCB1
RRF ACCB0
i = i + 1
endw
i = 8
while i < 15
BTFSC BARGB0,i-8
GOTO UM3115NA#v(i)
i = i + 1
endw
CLRF ACCB0
; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETFW 0
UM3115NA0
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = 1
while i < 8
BTFSS BARGBl,i
GOTO UM3115NA#v(i)
UM3115A#v(i)
MOVF TEMPB3,W
ADDFWF ACCB3
MOVF TEMPB2,W
BTFS C _C
INCFSZ TEMPB2,W
ADDFWF ACCB2
MOVF TEMPB1,W
BTFS C _C
INCFSZ TEMPB1,W
ADDFWF ACCB1
MOVF TEMPB0,W
BTFS C _C
INCFSZ TEMPB0,W
ADDFWF ACCB0
UM3115NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 8
while i < 15
BTFSS BARGBO,i-8
GOTO UM3115NA#v(i)
UM3115A#v(i)
MOVF TEMPB3,W
ADDFWF ACCB3
MOVF TEMPB2,W
BTFS C _C
INCFSZ TEMPB2,W
ADDFWF ACCB2
MOVF TEMPB1,W
BTFS C _C
INCFSZ TEMPB1,W
ADDFWF ACCB1
MOVF TEMPB0,W
BTFS C _C
INCFSZ TEMPB0,W
ADDFWF ACCB0
UM3115NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
i = i + 1
endw
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
endm
32x16 Bit Signed Fixed Point Multiply 32x16 -> 32
Input: 16 bit signed fixed point multiplicand in AARGBO
Use: CALL FXM3216S
Output: 32 bit signed fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 13+393+2 = 408 clks B > 0
Min Timing: 28+393+2 = 423 clks B < 0
Min Timing: 13+98 = 111 clks
PM: 10+79+1 = 98 DM: 9

FXM3216S
BTFSS BARGBO,MSB
GOTO M3216SOK
COMF BARGBO1
BTFSC _Z
DCFC BARGBO0
COMF BARGBO0
COMF AARGB3
INCF AARGB3
BTFSC _Z
DCFC AARGB2
COMF AARGB2
BTFSC _Z
DCFC AARGB1
COMF AARGB1
BTFSC _Z
DCFC AARGB0
COMF AARGB0
M3216SOK
CLRF ACCB4 ; make multiplier BARG > 0
CLRF ACCB5
MOVF AARGBO0,W
MOVWF TEMPB0
MOVF AARGB1,W
MOVWF TEMPB1
MOVF AARGB2,W
MOVWF TEMPB2
MOVF AARGB3,W
MOVWF TEMPB3
SMUL3216L
RETLW 0x00

32x16 Bit Unsigned Fixed Point Multiply 32x16 -> 32
Input: 16 bit unsigned fixed point multiplicand in AARGBO
Use: CALL FXM3216U
Output: 32 bit unsigned fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 10+400+2 = 412 clks
Min Timing: 10+104 = 114 clks
PM: 10+79+1 = 98 DM: 9

FXM3216U
CLRF ACCB4 ; clear partial product
CLRF ACCB5
MOVF AARGBO0,W
MOVWF TEMPB0
MOVF AARGB1,W
MOVWF TEMPB1
MOVF AARGB2,W
MOVWF TEMPB2
MOVF AARGB3,W
MOVWF TEMPB3
```assembly
MOVF AARG3,W
MOVWF TEMPB3
UMUL3216L
RETLW 0x00

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

; 31x15 Bit Unsigned Fixed Point Divide 31x15 -> 30
; Input: 15 bit unsigned fixed point multiplicand in AARG0
; 15 bit unsigned fixed point multiplier in BARG0
; Use: CALL FXM3115U
; Output: 30 bit unsigned fixed point product in AARG0
; Result: AARG <- AARG x BARG
; Max Timing: 10+380+2 = 392 clks
; Min Timing: 10+96 = 106 clks
; FM: 10+80+1 = 91 DM: 9
FXM3115U
CLRF ACCB4 ; clear partial product
CLRF ACCB5
MOVF AARG0,W
MOVWF TEMPB0
MOVF AARG1,W
MOVWF TEMPB1
MOVF AARG2,W
MOVWF TEMPB2
MOVF AARG3,W
MOVWF TEMPB3
UMUL3115L
RETLW 0x00

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

END
```

---

**C.4 24x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines**

```
; 24x24 PIC16 FIXED POINT MULTIPLY ROUTINES VERSION 1.2
; Input: fixed point arguments in AARG and BARG
; Output: product AARGxBARG in AARG
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed multiply application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in
; performance.
; Routine Clocks Function
; FXM2424S 535 24x24 -> 48 bit signed fixed point multiply
; FXM2424U 512 24x24 -> 48 bit unsigned fixed point multiply
; FXM2323U 497 23x23 -> 46 bit unsigned fixed point multiply
; The above timings are based on the looped macros. If space permits,
; approximately 61-95 clocks can be saved by using the unrolled macros.

; list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

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

; Test suite storage
RANDHI equ 0x1A ; random number generator registers
RANDLO equ 0x1B
DATA equ 0x20 ; beginning of test data

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

; Test suite for 24x24 bit fixed point multiply algorithms
org 0x0005
MAIN
MOVLW RAMSTART
MOVWF FSR
MEMLOOP
CLRF INDF
INC W
MOVLW RAMSTOP
```
SUBWF FSR, W
BTFSS _Z
GOTO MEMLOOP
MOVLW 0x45
MOVWF RANDLO
MOVLW 0x30
MOVWF RANDHI
MOVLW DATA
MOVWF FSR
BCF _RP0
BCF _RP1
BCF _IRP
CALL TFXM2424
SELF
GOTO SELF
RANDOM16
RLF RANDHI, W
XORWF RANDHI, W
MOVWF TEMPB0
SWAPF RANDHI
SWAPF RANDLO, W
MOVWF TEMPB1
RLF TEMPB1, W
RLF TEMPB1
MOVF TEMPB1, W
XORWF RANDHI, W
SWAPF RANDHI
XORWF RANDLO
SWAPF RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI
RETESW 0
; Test suite for FXM2424
TFXM2424
CALL RANDOM16
MOVWF RANDHI, W
MOVWF BARGBO
BCF BARGBO, MSB
MOVF BARGBO, W
MOVWF INDF
INCF FSR
MOVF RANDLO, W
MOVWF BARGB1
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI, W
MOVWF BARGB2
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI, W
MOVWF AARGBO
BCF AARGBO, MSB
MOVF AARGBO, W
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI, W
MOVWF AARGB2
BCF AARGB2, MSB
MOVF AARGB2, W
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI, W
MOVWF AARGB2
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI, W
MOVWF AARGB2
MOVWF INDF
INCF FSR
CALL FXM2323U
MOVF AARGBO,W
MOVF INDF
INCF FSR
MOVF AARGB1,W
MOVF INDF
INCF FSR
MOVF AARGB2,W
MOVF INDF
INCF FSR
MOVF AARGB3,W
MOVF INDF
INCF FSR
MOVF AARGB4,W
MOVF INDF
INCF FSR
MOVF AARGB5,W
MOVF INDF
INCF FSR
RETLW Ox00

;**********************************************************************************************
;**********************************************************************************************
24x24 Bit Multiplication Macros

SMUL2424L macro
    Max Timing: 2+12+6*21+2+21+2+6+23+22+7 = 506 clks
    Min Timing: 2+7*6+5+1+7*6+5+2+6*6+5+5 = 145 clks
    FM: 24+20+2+21+2+22+7 = 98          DM: 13
    MOVW 0x8
    MOVWF LOOPS2424A
    RRF
    BTFSC _C
    GOTO ALSM2424NA
    DECFSZ LOOPCOUNT
    GOTO LOOPS2424A
    MOVWF LOOPCOUNT
    loopsm2424b
    RRF
    BARGB1
    BTFSC _C
    GOTO BLSM2424NA
    DECFSZ LOOPCOUNT
    GOTO LOOPSM2424B
    MOVW 0x7
    MOVWF LOOPCOUNT
    loopsm2424c
    RRF
    BARGBO
    BTFSC _C
    GOTO CLSM2424NA
    DECFSZ LOOPCOUNT
    GOTO LOOPSM2424C
    CLRWF AARGB0
    CLRWF AARGB1
    CLRWF AARGB2
    RETLW 0x00
    loopsm2424
    ALOOPSM2424
    RRF
    BARGB2
    BTFSS _C
    GOTO ALSM2424NA
    MOVF TEMPB2,W
    ADDWF ACCB2
    MOVF TEMPB1,W
    BTFSC _C
    INCFSZ TEMPB1,W
    ADDWF ACCB1
    MOVF TEMPB0,W
    BTFSC _C
    INCFSZ TEMPB0,W
AN617

INCFSZ TEMPB0,W
ADDFW ACCB0

ALSM2424NA
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
DECFSZ LOOPCOUNT
GOTO ALOOPSM2424
MOVLW 0x8
MOVWF LOOPCOUNT

BLOOPSM2424
RRF BARGB1
BTFSS _C
GOTO BLSM2424NA
MOVF TEMPB2,W
ADDFW ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDFW ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDFW ACCB0

BLSM2424NA
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO BLOOPSM2424
MOVWF 0x7
MOVWF LOOPCOUNT

CLOOPSM2424
RRF BARGB0
BTFSS _C
GOTO CLSM2424NA
MOVF TEMPB2,W
ADDFW ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDFW ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDFW ACCB0

CLSM2424NA
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
DECFSZ LOOPCOUNT
GOTO CLOOPSM2424
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
endm
UMUL2424L macro
; Max Timing: 2+14+6*20+19+2+7*21+20+2+7*22+21 = 501 clks
; Min Timing: 2+7*6+5+1+7*6+5+1+7*6+5+5 = 150 clks
; PM: 23+20+2+21+2+22 = 88 DM: 13
MOVLW 0x08
MOVWF LOOPCOUNT

LOOPUM2424A
RRF BARGB2
BTFSC _C
GOTO ALUM2424NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2424A
MOVWF LOOPCOUNT

LOOPUM2424B
RRF BARGB1
BTFSC _C
GOTO BLUM2424NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2424B
MOVWF LOOPCOUNT

LOOPUM2424C
RRF BARGB0
BTFSC _C
GOTO CLUM2424NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2424C
CLR F AARBGO
CLR F AARGB1
CLR F AARGB2
RETLW 0x00

CLUM2424NAP
BCF _C
GOTO CLUM2424NA

BLUM2424NAP
BCF _C
GOTO BLUM2424NA

ALUM2424NAP
BCF _C
GOTO ALUM2424NA

ALOOPUM2424
RRF BARGB2
BTFSS _C
GOTO ALUM2424NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

ALUM2424NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
DECFSZ LOOPCOUNT
GOTO ALOOPUM2424
MOVLW 0x08
MOVWF LOOPCOUNT

BLOOPUM2424
RRF BARGB1
BTFSS _C
GOTO BLUM2424NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFS C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFS C
INCFSZ TEMPB0,W
ADDWF ACCB0

BLUM2424NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO BLOOPUM2424
MOVLW 0x08
MOVWF LOOPCOUNT

CLOOPUM2424
RRF BARGB0
BTFS C
GOTO CLUM2424NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFS C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFS C
INCFSZ TEMPB0,W
ADDWF ACCB0

CLUM2424NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
DECFSZ LOOPCOUNT
GOTO CLOOPUM2424
endm

UMUL2323L

; Max Timing: 2+15+6*20+19+2+7*21+20+2+6*22+21+6 = 486 clks
; Min Timing: 2+7*6+5+1+7*6+5+2+6*6+5+5 = 145 clks
; PM: 24+20+2+21+2+22+6 = 97     DM: 13
MOVWL 0x8
MOVWF LOOPCOUNT

LOOPUM2323A
RRF BARGB2
BTFS C
GOTO ALUM2323NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2323A
MOVWF LOOPCOUNT

LOOPUM2323B
RRF BARGB1
BTFS C
GOTO BLUM2323NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2323B
MOVWL 0x7
MOVWF LOOPCOUNT

LOOPUM2323C
RRF  BARGBO
BTFSS  _C
GOTO  CLUM2323NAP
DECFSZ  LOOPCOUNT
GOTO  LOOPUM2323C
CLRFB  AARGBO
CLRFB  AARGBl
CLRFB  AARGB2
RETLW  0x00

CLUM2323NAP
BCF  _C
GOTO  CLUM2323NA

BLUM2323NAP
BCF  _C
GOTO  BLUM2323NA

ALUM2323NAP
BCF  _C
GOTO  ALUM2323NA

ALoopUM2323
RRF  BARGB2
BTFSS  _C
GOTO  ALUM2323NA
MOVF  TEMPB2,W
ADDWF  ACCB2
MOVF  TEMPB1,W
BTFSS  _C
INCFSZ  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPB0,W
BTFSS  _C
INCFSZ  TEMPB0,W
ADDWF  ACCB0

ALUM2323NA
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
RRF  ACCB3
DECFSZ  LOOPCOUNT
GOTO  ALoopUM2323
MOVLW  0x08
MOVWF  LOOPCOUNT

BLoopUM2323
RRF  BARGB1
BTFSS  _C
GOTO  BLUM2323NA
MOVF  TEMPB2,W
ADDWF  ACCB2
MOVF  TEMPB1,W
BTFSS  _C
INCFSZ  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPB0,W
BTFSS  _C
INCFSZ  TEMPB0,W
ADDWF  ACCB0

BLUM2323NA
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
RRF  ACCB3
RRF  ACCB4
DECFSZ  LOOPCOUNT
GOTO  BLoopUM2323
MOVLW  0x07
MOVWF  LOOPCOUNT

CLoopUM2323
SMUL2424 macro

; Max Timing: 8+7*17+8*18+7*19+7 = 411 clks
; Min Timing: 46+5 = 51 clks
; PM: 51+4+7*17+8*18+7*19+7 = 466 DM: 12

variable i

i = 0

while i < 8

BTFSC BARGB2,i
GOTO SM2424NA#v(i)
i = i + 1
endw
i = 8

while i < 16

BTFSC BARGB1,i-8
GOTO SM2424NA#v(i)
i = i + 1
endw
i = 16

while i < 23

BTFSC BARGB0,i-16
GOTO SM2424NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETEW 0
SM2424NA0 RLF TEMPB0, W
RRF ACCB0
RRF ACCB1

AN617
```assembly
RRF ACCB2
RRF ACCB3
i = 1
while i < 8
    BTFSN BARGB2,i
    GOTO SM2424A#v(i)
    MOVF TEMPB2,W
    ADDWF ACCB2
    MOVF TEMPB1,W
    BTFSN _C
    INCFSZ TEMPB1,W
    ADDWF ACCB1
    MOVF TEMPB0,W
    BTFSN _C
    INCFSZ TEMPB0,W
    ADDWF ACCB0
SM2424A#v(i) RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = i + 1
endw
i = 8
while i < 16
    BTFSN BARGB1,i-8
    GOTO SM2424A#v(i)
    MOVF TEMPB2,W
    ADDWF ACCB2
    MOVF TEMPB1,W
    BTFSN _C
    INCFSZ TEMPB1,W
    ADDWF ACCB1
    MOVF TEMPB0,W
    BTFSN _C
    INCFSZ TEMPB0,W
    ADDWF ACCB0
SM2424A#v(i) RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 16
while i < 23
    BTFSN BARGB0,i-16
    GOTO SM2424A#v(i)
    MOVF TEMPB2,W
    ADDWF ACCB2
    MOVF TEMPB1,W
    BTFSN _C
    INCFSZ TEMPB1,W
    ADDWF ACCB1
    MOVF TEMPB0,W
    BTFSN _C
    INCFSZ TEMPB0,W
    ADDWF ACCB0
SM2424A#v(i) RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
```
i = i + 1
endw
RLF TEMPO,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
endm

UMUL2424

; Max Timing:  \(8*8*17+8*18*8*19 = 440\) clks
; Min Timing:  \(49+5 = 54\) clks
; PM:  \(54+4*8*17+8*18*8*19 = 490\)

variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 8
BTFSC BARGB2,i
GOTO UM2424NA#v(i)
i = i + 1
endw
i = 8

while i < 16
BTFSC BARGB1,i-8
GOTO UM2424NA#v(i)
i = i + 1
endw
i = 16

while i < 24
BTFSC BARGB0,i-16
GOTO UM2424NA#v(i)
i = i + 1
endw
CLRFB ACCB0 ; if we get here, BARG = 0
CLRFB ACCB1
CLRFB ACCB2
RETFB 0

UM2424NA0
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = 1
while i < 8
BTFSS BARGB2,i
GOTO UM2424NA#v(i)

UM2424A#v(i)
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

UM2424NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
```assembly
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGBl, i-8
GOTO UM2424NA#v(i)
MOVF TEMPB2, W
ADDWF ACCB2
MOVF TEMPB1, W
BTFSC _C
INCFSZ TEMPB1, W
ADDWF ACCB1
MOVF TEMPBO, W
BTFSC _C
INCFSZ TEMPBO, W
ADDWF ACCB0
UM2424NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
i = 16
while i < 24
BTFSS BARGBo, i-16
GOTO UM2424NA#v(i)
MOVF TEMPB2, W
ADDWF ACCB2
MOVF TEMPB1, W
BTFSC _C
INCFSZ TEMPB1, W
ADDWF ACCB1
MOVF TEMPBO, W
BTFSC _C
INCFSZ TEMPBO, W
ADDWF ACCB0
UM2424NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
i = i + 1
endw
endm
UMUL2323
macro
;    Max Timing: 8*7+8*18+7*19+7 = 411 clks
;    Min Timing: 46+5 = 51 clks
;    PM: 51+4*7+17+8*18+7*19+7 = 466    DM: 12
variable i
i = 0
BCF _C
; clear carry for first right shift
while i < 8
BTFSC BARGB2, i
GOTO UM3232NA#v(i)
i = i + 1
endw
i = 8
while i < 16
BTFSC BARGB1, i-8
GOTO UM3232NA#v(i)
i = i + 1
endw
```

i = 16
while i < 23

BTFSC BARGB0,i-16
GOTO UM2323NA#v(i)
i = i + 1
endw

CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RESTDW 0

UM2323NA0
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3

i = 1
while i < 8

BTFSS BARGB2,i
GOTO UM2323NA#v(i)

UM2323A#v(i)
MOVF TEMPB2,W
ADDF W ACCB2
MOVF TEMPB1,W
BTFSC _C
INCF SZ TEMPB1,W
ADDF W ACCB1
MOVF TEMPB0,W
BTFSC _C
INCF SZ TEMPB0,W
ADDF W ACCB0

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3

i = i + 1
endw

i = 8
while i < 16

BTFSS BARGB1,i-8
GOTO UM2323NA#v(i)

UM2323A#v(i)
MOVF TEMPB2,W
ADDF W ACCB2
MOVF TEMPB1,W
BTFSC _C
INCF SZ TEMPB1,W
ADDF W ACCB1
MOVF TEMPB0,W
BTFSC _C
INCF SZ TEMPB0,W
ADDF W ACCB0

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4

i = i + 1
endw

i = 16
while i < 23

BTFSS BARGB0,i-16
GOTO UM2323NA#v(i)

UM2323A#v(i)
MOVF TEMPB2,W
ADDF W ACCB2
MOVF TEMPB1,W
BTFSC _C
INCF SZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPBO, W
BTFSC _C
INCF W
ADDWF ACCB0
UM2323NA4v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
i = i + 1
endw
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
RRF ACCB5
endm

;**********************************************************************************************
;**********************************************************************************************
; 24x24 Bit Signed Fixed Point Multiply 24x24 -> 48
; Input: 24 bit signed fixed point multiplicand in AARGBO
; 24 bit signed fixed point multiplier in BARGBO
; Use: CALL FXM2424S
; Output: 48 bit signed fixed point product in AARGBO
; Result: AARG <-- AARG x BARG
; Max Timing: 12+506+2 = 520 clks B > 0
; 27+506+2 = 535 clks B < 0
; Min Timing: 12+145 = 157 clks
; PM: 27+98+1 = 126 DM: 13
FXM2424S
BTFSS BARGBO, MSB
GOTO M2424SOK
COMF BARGBO2
INCF BARGBO2
BTFSC _2
DECF BARGBO1
COMF BARGBO1
BTFSC _2
DECF BARGBO
COMF BARGBO
COMF AARGBO2
INCF AARGBO2
BTFSC _2
DECF AARGBO1
COMF AARGBO1
BTFSC _2
DECF AARGBO
COMF AARGBO
M2424SOK
CLRF ACCB3
; clear partial product
CLRF ACCB4
CLRF ACCB5
MOVF AARGBO, W
MOVF TEMPBO
MOVF AARGBO1, W
MOVF TEMPBO1
MOVF AARGBO2, W
MOVF TEMPBO2
SMUL2424L
RETLW 0x00
;**********************************************************************************************
;**********************************************************************************************
24x24 Bit Unsigned Fixed Point Multiply 24x24 -> 48

Input: 24 bit unsigned fixed point multiplicand in AARGBO

Output: 48 bit unsigned fixed point product in AARGBO

Result: AARG <-- AARG x BARG

Max Timing: 9+501+2 = 512 clks

Min Timing: 9+150 = 159 clks

PM: 9+88+1 = 98

DM: 13

FXM2424U

CLRF ACCB3 ; clear partial product
CLRF ACCB4
CLRF ACCB5
MOVF AARGBO,W
MOVF TEMPB0
MOVF AARGB1,W
MOVF TEMPB1
MOVF AARGB2,W
MOVF TEMPB2
UMUL2424L
RETLW 0x00

23x23 Bit Unsigned Fixed Point Divide 23x23 -> 46

Input: 23 bit unsigned fixed point multiplicand in AARGBO

Output: 46 bit unsigned fixed point product in AARGBO

Result: AARG <-- AARG x BARG

Max Timing: 9+486+2 = 497 clks

Min Timing: 9+145 = 154 clks

PM: 9+97+1 = 107

DM: 13

FXM2323U

CLRF ACCB3 ; clear partial product
CLRF ACCB4
CLRF ACCB5
MOVF AARGBO,W
MOVF TEMPB0
MOVF AARGB1,W
MOVF TEMPB1
MOVF AARGB2,W
MOVF TEMPB2
UMUL2323L
RETLW 0x00

24x16 PIC16C5X/PIC16CX Fixed Point Multiply Routines

Input: fixed point arguments in AARG and BARG

Output: product AARGxBARG in AARG

All timings are worst case cycle counts

It is useful to note that the additional unsigned routines requiring a non-power of two argument can be called in a signed multiply application where it is known that the respective argument is nonnegative, thereby offering some improvement in performance.

Routine | Clocks | Function |
---------|--------|----------|
FXM2416S | 346    | 24x16 -> 40 bit signed fixed point multiply |
FXM2416U | 334    | 24x16 -> 40 bit unsigned fixed point multiply |
FXM2315U | 319    | 23x15 -> 38 bit unsigned fixed point multiply |

The above timings are based on the looped macros. If space permits, approximately 36-62 clocks can be saved by using the unrolled macros.

C.5
include <MATH16.INC> ; PIC16 math library definitions

;***********************************************************************************************
;***********************************************************************************************
; Test suite storage
RANDHI equ 0x1A ; random number generator registers
RANDLO equ 0x1B
DATA equ 0x20 ; beginning of test data

;***********************************************************************************************
;***********************************************************************************************
; Test suite for 24x16 bit fixed point multiply algorithms

org 0x0005

MAIN
MOVWF RAMSTART

MEMLOOP
CLRF INDF
INCF FSR
MOVWF RAMSTOP
SUBWF FSR,W
BTFSS __Z
GOTO MEMLOOP
MOVWF 0x45 ; seed for random numbers

MOVWF RANDLO
MOVWF 0x30
MOVWF RANDHI
MOVWF DATA
MOVWF FSR
BCF __RP0
BCF __RP1
BCF __IRP
CALL TFXM2416

SELF
GOTO SELF

RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMPB0
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPB1
RLF TEMPB1,W
RLF TEMPB1
MOVWF TEMPB1,W
XORWF RANDHI,W
SWAPF RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETEW 0

; Test suite for TFXM2416

TFXM2416
CALL RANDOM16
MOVWF RANDHI,W
MOVWF BARGBO
BCF BARGBO,MSB
MOVWF BARGBO,W
MOVWF INDF
INCF FSR
MOVWF RANDLO,W
MOVWF BARGBO1
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVWF RANDHI,W
MOVWF AARGBO
BCF AARGBO,MSB
MOVWF AARGBO,W
AN617

MOVF INDF
INCF FSR
MOVF RANDLO,W
MOVF AARGBl
MOVF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVF AARGB2
MOVF INDF
INCF FSR
MOVF AARGB1,W
MOVF INDF
INCF FSR
MOVF AARGB2,W
MOVF INDF
INCF FSR
MOVF AARGB3,W
MOVF INDF
INCF FSR
MOVF AARGB4,W
MOVF INDF
INCF FSR
MOVWF INDF
INCF FSR
MOVF AARGBO,W
MOVF INDF
INCF FSR
MOVF AARGBl,W
MOVF INDF
INCF FSR
MOVF AARGB2,W
MOVF INDF
INCF FSR
MOVF AARGB3,W
MOVF INDF
INCF FSR
MOVF AARGB4,W
MOVF INDF
INCF FSR
RETLW OxF0

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

24x16 Bit Multiplication Macros
SMUL2416L macro
; Max Timing: 2+12+6*21+20+2+6*22+21+6 = 321 clks
; Min Timing: 2+7*6+5+2+6*6+5+5 = 97 clks
; PM: 19+20+2+21+6 = 68
; DM: 12
MOVLW OxF8
MOVWF LOOPCOUNT
LOOPSM2416A
RRF BARGBl
BTFSC _C
GOTO ALSM2416NA
DECFSZ LOOPCOUNT
GOTO LOOPSM2416A
MOVLW OxF7
MOVWF LOOPCOUNT
LOOPSM2416B
RRF BARGBO
BTFSC _C
GOTO BLSM2416NA
DECFSZ LOOPCOUNT
GOTO LOOPSM2416B
CLRF AARGBO
CLRF AARGBl
CLRF AARGB2
RETLW 0xF0
ALOOPSM2416
RRF BARGBl
BTFSS _C
GOTO ALSM2416NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVWF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVWF TEMPO,W

© 1995 Microchip Technology Inc.
UMUL2416L
macro
; Max Timing: 2+14+6*20+19+2+7*21+20 = 324 clks
; Min Timing: 2+7*6+5+1+7*6+5+5 = 102 clks
; PM: 18+20+2+21 = 61  DM: 12
MOVLW Ox08
MOVWF LOOPCOUNT
LOOPEUM2416A
RRF    BARGBO
BTFSC  _C
GOTO   ALUM2416NA
DECFSZ LOOPCOUNT
GOTO   LOOPEUM2416A
MOVWF LOOPCOUNT
LOOPEUM2416B
RRF    BARGBO
BTFSC  _C
GOTO   BLUM2416NA
DECFSZ LOOPCOUNT
GOTO   LOOPEUM2416B
CLRF   AARGBO
CLRF   AARGBl
CLRF   AARGB2
RETLW  0x00
BLUM2416NAP
BCF    _C
AN617

GOTO ALUM2416NA
BCF _C
GOTO ALUM2416NA

ALUM2416NA
RRF BARGB1
BTFSS _C
GOTO ALUM2416NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCBO

ALUM2416
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
DECFSZ LOOPCOUNT
GOTO ALOOPUM2416
MOVLW 0x08
MOVWF LOOPCOUNT

ALoopUM2416
RRF BARGB0
BTFSS _C
GOTO BLUM2416NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCBO

BLUM2416NA
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO BLOOPUM2416
endm

UMUL2315L macro
MOVLW 0xB
MOVWF LOOPCOUNT
LOOPUM2315A
RRF BARGB1
BTFSC _C
GOTO ALUM2315NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2315A
MOVLW 0x7
MOVWF LOOPCOUNT

LOOPUM2315B
RRF BARGB0

Max Timing: 2+15+6*20+19+2+6*21+20+5 = 309 clks
Min Timing: 2+7*6+5+1+6*6+5 = 96 clks
PM: 19+20+2+21+5 = 67 IM: 12

DS00617A-page 2-226 © 1995 Microchip Technology Inc.
AN617

BTFS C _C
GOTO BLUM2315NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM2315B
CLRF AARGBO
CLRF AARGBl
CLRF AARGB2
RETLW OxFF

BLUM2315NAP
BCF _C
GOTO BLUM2315NA

ALUM2315NAP
BCF _C
GOTO ALUM2315NA

ALOOPUM2315
RRF BARGBl
BTFS S _C
GOTO ALUM2315NA
MOVF TEMPBl,W
ADDWF ACCBl
MOVF TEMPBO,W
BTFS C _C
INCFSZ TEMPBO,W
ADDWF ACCBO

ALUM2315NA
RRF ACCBO
RRF ACCBl
RRF ACCB2
RRF ACCB3
DECFSZ LOOPCOUNT
GOTO ALOOPUM2315
MOVLW OxFF
MOVWF LOOPCOUNT

BLOOPUM2315
RRF BARGBO
BTFS S _C
GOTO BLUM2315NA
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPBl,W
BTFS C _C
INCFSZ TEMPBl,W
ADDWF ACCBl
MOVF TEMPBO,W
BTFS C _C
INCFSZ TEMPBO,W
ADDWF ACCBO

BLUM2315NA
RRF ACCBO
RRF ACCBl
RRF ACCB2
RRF ACCB3
RRF ACCB4
DECFSZ LOOPCOUNT
GOTO BLOOPUM2315
RRF ACCBO
RRF ACCBl
RRF ACCB2
RRF ACCB3
RRF ACCB4
endm
SMUL2416 macro
; Max Timing: 8+7*17+7*18+6 = 259 clks
; Min Timing: 30+5 = 35 clks
; PM: 30+4*7+7*18+6 = 285 DM: 11

variable i
i = 0

while i < 8
    BTFSC BARGBl,i
    GOTO SM2416NA#v(i)
i = i + 1
endw
i = 8

while i < 15
    BTFSC BARGBO,i-8
    GOTO SM2416NA#v(i)
i = i + 1
endw

CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETEW 0

SM2416NA0
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = 1
while i < 8
    BTFSS BARGBl,i
    GOTO SM2416NA#v(i)
SM2416A#v(i) MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
SM2416NA#v(i) RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = i + 1
endw
i = 8
while i < 15
    BTFSS BARGBO,i-8
    GOTO SM2416NA#v(i)
SM2416A#v(i) MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
SM2416NA#v(i) RLF TEMPB0,W
UMUL2416

macro

; Max Timing: 8+8*17+8*18 = 288 clks
; Min Timing: 33+5 = 38 clks
; PM: 37+4+8*17+8*18 = 321 DM: 11

variable i
i = 0
BCF _C ; clear carry for first right shift

while i < 8
BTFSC BARGBl,i
GOTO UM2416NA#v(i)
i = i + 1
endw
i = 8

while i < 16
BTFSC BARGBo,i-8
GOTO UM2416NA#v(i)
i = i + 1
endw

CLRFB ACCB0 ; if we get here, BARG = 0
CLRFB ACCB1
CLRFB ACCB2
RETF 0

UM2416NA0
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = 1
while i < 8
BTFSS BARGBl,i
GOTO UM2416NA#v(i)

UM2416A#v(i)
MOVF TEMPB2,W
ADDWF ACCB2
MOVF TEMPB1,W
BTFSC _C
INCFSZ TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0

UM2416NA#v(i)
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGB0, i-8
GOTO UM2416NA#v(i)
UM2416A#v(i) MOVF TEMPB2, W ADDWF ACCB2 MOVF TEMPB1, W BTFSC _C INCFSZ TEMPB1, W ADDWF ACCB1 MOVF TEMPB0, W BTFSC _C INCFSZ TEMPB0, W ADDWF ACCB0
UM2416NA#v(i) RRF ACCB0 RRF ACCB1 RRF ACCB2 RRF ACCB3 RRF ACCB4

i = i + 1
eendw
eendm

UMUL2315 macro
; Max Timing: 8*7*17*7*18+6 = 259 clks
; Min Timing: 31+5 = 36 clks
; PM: 35+4*7*17*7*18+6 = 290 DM: 11

variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 8 BTFSC BARGB1,i GOTO UM2315NA#v(i)
i = i + 1
endw
i = 8
while i < 15 BTFSC BARGB0, i-8 GOTO UM2315NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETFW 0
UM2315NA0 RRF ACCB0 RRF ACCB1 RRF ACCB2 RRF ACCB3
i = 1
while i < 8 BTFSS BARGB1,i GOTO UM2315NA#v(i)
UM2315A#v(i) MOVF TEMPB2, W ADDWF ACCB2 MOVF TEMPB1, W BTFSC _C INCFSZ TEMPB1, W ADDWF ACCB1 MOVF TEMPB0, W BTFSC _C INCFSZ TEMPB0, W ADDWF ACCB0
UM2315NA#v(i) RRF ACCB0 RRF ACCB1 RRF ACCB2 RRF ACCB3
i = i + 1
endw
i = 0
while i < 15
  BTFSS BARGBO, i-8
  GOTO UM2315NA#v(i)
UM2315A#v(i)
  MOVF TEMPB2, W
  ADDWF ACCB2
  MOVF TEMPB1, W
  BTFSC _C
  INCFSZ TEMPB1, W
  ADDWF ACCB1
  MOVF TEMPB0, W
  BTFSC _C
  INCFSZ TEMPB0, W
  ADDWF ACCB0
UM2315NA#v(i)
  RRF ACCB0
  RRF ACCB1
  RRF ACCB2
  RRF ACCB3
  RRF ACCB4
  i = i + 1
endw
  RRF ACCB0
  RRF ACCB1
  RRF ACCB2
  RRF ACCB3
  RRF ACCB4
endm

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

FXM2416S
  BTFSS BARGBO, MSB
  GOTO FXM2416S
  COMP BARGB0
  INCF BARGB0
  BTFSC _Z
  DECF BARGB0
  COMP BARGB0
  COMP AARGB0
  INCF AARGB0
  BTFSC _Z
  DECF AARGB0
  COMP AARGB0
  BTFSC _Z
  DECF AARGBO
  COMP AARGBO
  INCF AARGBO
  BTFSC _z
  DECF AARGBO
  COMP AARGBO
  CLRF ACCB3
  ; make multiplier BARG > 0

M2416SOK
  CLRFB ACCB3
  CLRFB ACCB4
  MOVF AARGB0, W
  MOVWF TEMPB0
  MOVF AARGB1, W
  MOVWF TEMPB1
  MOVF AARGB2, W
  MOVWF TEMPB2
  MOVF AARGB3
  MOVWF TEMPB3
  MOVF AARGB4
  MOVWF TEMPB4
SMUL2416L
RETLW 0x00

; 24x16 Bit Unsigned Fixed Point Multiply 24x16 -> 40
; Input: 24 bit unsigned fixed point multiplicand in AARGBO
; 16 bit unsigned fixed point multiplier in BARGBO
; Use: CALL FXM2416U
; Output: 40 bit unsigned fixed point product in AARGBO
; Result: AARG <-- AARG x BARG
; Max Timing: 8+324+2 = 334 clks
; Min Timing: 8+102 = 110 clks
; PM: 8+61+1 = 70 DM: 12
FXM2416U
CLRF ACCB3 ; clear partial product
CLRF ACCB4
MOVF AARGBO,W
MOVWF TEMPB0
MOVF AARGBl,W
MOVWF TEMPBl
MOVF AARGB2,W
MOVWF TEMPB2
UMUL2416L
RETLW 0x00

; 23x15 Bit Unsigned Fixed Point Divide 23x15 -> 38
; Input: 23 bit unsigned fixed point multiplicand in AARGBO
; 15 bit unsigned fixed point multiplier in BARGBO
; Use: CALL FXM2315U
; Output: 38 bit unsigned fixed point product in AARGBO
; Result: AARG <-- AARG x BARG
; Max Timing: 8+309+2 = 319 clks
; Min Timing: 8+96 = 104 clks
; PM: 8+67+1 = 76 DM: 12
FXM2315U
CLRF ACCB3 ; clear partial product
CLRF ACCB4
MOVF AARGBO,W
MOVWF TEMPB0
MOVF AARGBl,W
MOVWF TEMPBl
MOVF AARGB2,W
MOVWF TEMPB2
UMUL2315L
RETLW 0x00

C.6 16x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

16x16 PIC16 FIXED POINT MULTIPLY ROUTINES

Input: fixed point arguments in AARG and BARG
Output: product AARGxBARG in AARG
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed multiply application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Function</th>
<th>Clocks</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXM1616S</td>
<td>16x16 -&gt; 32 bit signed fixed point multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FXM1616U</td>
<td>16x16 -&gt; 32 bit unsigned fixed point multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FXM1515U</td>
<td>15x15 -&gt; 30 bit unsigned fixed point multiply</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above timings are based on the looped macros. If space permits,
approximately 64-73 clocks can be saved by using the unrolled macros.

list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

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

Test suite storage
RANDHI equ 0x1A ; random number generator registers
RANDLO equ 0x1B
TESTCOUNT equ 0x20 ; counter
DATA equ 0x21 ; beginning of test data

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

; Test suite for 16x16 bit fixed point multiply algorithms

org 0x0005
MAIN
MOVLW RAMSTART
MOVWF FSR
MEMLOOP CLEAR
INCF FSR
MOVLW RAMSTOP
SUBWF FSR,W
BTFSS _Z
GOTO MEMLOOP
MOVLW 0x45 ; seed for random numbers
MOVWF RANDLO
MOVLW 0x30
MOVWF RANDHI
MOVLW DATA
MOVWF FSR
BCF _RF0
BCF _RF1
BCF _RIFP
CALL TFXM1616
SELF
GOTO SELF
RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMPB0
SWAPP RANDHI
SWAPP RANDLO,W
MOVWF TEMPB1
RLF TEMPB1,W
RLF TEMPB1
MOVF TEMPB1,W
MOVF TEMPB1
XORWF RANDHI,W
SWAPP RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETEW 0

; Test suite for FXM1616
TFXM1616
MOVLW 1
MOVWF TESTCOUNT
M1616LOOP
CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGB0
BCF BARGB0,MSB
MOVF BARGB0,W
MOVWF INDF
INCF FSR
MOVF RANDLO,W
MOVWF BARGB1
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVF RANDHI,AARGBO
BCF AARGBO,MSB
INCF FSR
MOVF RANDLO,W
MOVF RANDLO,AARGBl
MOVWF INDF
INC AARGBO
INC AARGBl
INC AARGB2
INC AARGB3
DECFSZ TESTCOUNT
GOTO M1616LOOP
RETLW Ox00

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

16x16 Bit Multiplication Macros

SMUL1616L macro
; Max Timing: 2+11+6*16+15+2+6*17+16+5 = 249 clks
; Min Timing: 2+7*6+5+2+6*6+5+4 = 96 clks
; PM: 55
; DM: 9
MOV AARGBO
MOV AARGBl
LOOPCOUNT
BARGBl
_Rc
ALSM1616NA
LOOPCOUNT
LOOPSM1616A
RRF
BTFSC
GOTO
DECFSZ
GOTO
MOV AARGBl
MOV AARGBl
LOOPCOUNT
LOOPSM1616B
RRF
BTFSC
GOTO
DECFSZ
GOTO
CLRF
CLRF
RETLW
ALSM1616NA
RLF
RRF

© 1995 Microchip Technology Inc.
AN617

RRF ACCB1
RRF ACCB2
DECFSZ LOOPCOUNT
GOTO ALOOPSM1616
MOV LW 0x7
MOVWF LOOPCOUNT

BL00PSM1616
RRF BARGBO
BTFSS _C
GOTO BLSM1616NA
MOVF TEMPBO,W
ADDWF ACCB1
MOVF TEMPBO,W
BTFSC _C
INCFSZ TEMPBO,W
ADDWF ACCB0
BLSM1616NA
RLF TEMPBO,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
DECFSZ LOOPCOUNT
GOTO BLOOPSM1616
RLF TEMPBO,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
endm

UMUL1616L
macro
; Max Timing: 2+13+6*15+14+2+7*16+15 = 248 clks
; Min Timing: 2+7*6+5+1+7*6+5+4 = 101 clks
; PM: 51 DM: 9
MOV LW 0x08
MOVWF LOOPCOUNT

LOOPUM1616A
RRF BARGBO
BTFSC _C
GOTO ALUM1616NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM1616A
MOVWF LOOPCOUNT

LOOPUM1616B
RRF BARGBO
BTFSC _C
GOTO BLUM1616NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM1616B
CLR F AARGBO
CLR F AARGBl
RETLW 0x00

BLUM1616NAP
BCF _C
GOTO BLUM1616NA

ALUM1616NAP
BCF _C
GOTO ALUM1616NA

ALOOPUM1616
RRF BARGBO
BTFSS _C
GOTO ALOOPSM1616
MOVF TEMPBO,W
ADDWF ACCB1
MOVF TEMPBO,W
BTFSC _C
INCFSZ TEMPBO,W

© 1995 Microchip Technology Inc.
ADDWF ACCB0

ALUM1616NA

RRF ACCB0
RRF ACCB1
RRF ACCB2
DECFSZ LOOPCOUNT
GOTO ALOOPUM1616
MOVLW 0x08
MOVF LOOPCOUNT

BLOOPUM1616

RRF BARGBO
BTFSS _c
GOTO BLUM1616NA
MOVF TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0

BLUM1616NA

RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
DECFSZ LOOPCOUNT
GOTO BLOOPUM1616

dma

UMUL1515L

; Max Timing: 2+13+6*15+14+2+6*16+15+4 = 236 clks
; Min Timing: 2+7*6+5+2+6*6+5+4 = 97 clks
; PM: 56 DM: 9

MOVLW 0x8
MOVF LOOPCOUNT

LOOPUM1515A

RRF BARGB1
BTFSC _c
GOTO ALUM1515NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM1515A
MOVLW 0x7
MOVF LOOPCOUNT

LOOPUM1515B

RRF BARGBO
BTFSC _c
GOTO BLUM1515NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM1515B
CLRF AARGB0
CLRF AARGB1
RETLW 0x00

BLUM1515NAP

BCF _c
GOTO BLUM1515NA

ALUM1515NAP

BCF _c
GOTO ALUM1515NA

ALOOPUM1515

RRF BARGB1
BTFSS _c
GOTO ALUM1515NA
MOVF TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _c
INCFSZ TEMPB0,W
ADDWF ACCB0
SMUL1616 macro
;  Max Timing:  5+6+7*11+7*12+4 = 176 clks
;  Min Timing:  5+24+21+5 = 55 clks
;  PM:  5+3*8+3*7+6+7*11+7*12+4 = 221  OM:  8
variable i
  i = 0
  BTFSC AARGBO,MSB
  COMP ACCB2
  MOVF ACCB2,W
  MOVWF ACCB3
  RLF ACCB0,W
while i < 8
  BTFSC BARGB0,i
  GOTO SMUL1616NA#v(i)
  BCF ACCB2,7-i
  i = i + 1
endw
i = 8
while i < 15
  BTFSC BARGB0,i-8
  GOTO SMUL1616NA#v(i)
  BCF ACCB2,15-i
  i = i + 1
endw
  CLR BARG 0
  CLRF ACCB0
  CLRF ACCB1
  CLRF ACCB3
  RETEW 0
SMUL1616NA0
  RRF ACCB0
RRF    ACCB1
RRF    ACCB2
i = 1

while i < 8
BTFSS  BARGB1,i
GOTO   SM1616NA#v(i)

SM1616A#v(i)
MOVF   TEMPB1,W
ADDWF  ACCB1
MOVF   TEMPB0,W
BTFSC  _C
INCFSZ TEMPB0,W
ADDWF  ACCB0

SM1616NA#v(i)
RRF    ACCB0
RRF    ACCB1
RRF    ACCB2
i = i + 1
endw
i = 8

while i < 15
BTFSS  BARGB0,i-8
GOTO   SM1616NA#v(i)

SM1616A#v(i)
MOVF   TEMPB1,W
ADDWF  ACCB1
MOVF   TEMPB0,W
BTFSC  _C
INCFSZ TEMPB0,W
ADDWF  ACCB0

SM1616NA#v(i)
RRF    ACCB0
RRF    ACCB1
RRF    ACCB2
RRF    ACCB3
i = i + 1
endw
RRF    ACCB0
RRF    ACCB1
RRF    ACCB2
RRF    ACCB3
endm

UMUL1616
macro
; Max Timing: 1 + 6 + 7 * 11 + 8 * 12 = 180 clks
; Min Timing: 1 + 2 * 8 + 2 * 8 + 4 = 37 clks
; PM: 1 + 2 * 8 + 2 * 8 + 4 + 7 * 11 + 8 * 12 = 210  DM: 8
variable i
i = 0
BCF   _C    ; clear carry for first right shift
while i < 8
BTFSC  BARGB1,i
GOTO   UM1616NA#v(i)
i = i + 1
endw
i = 8

while i < 16
BTFSC  BARGB0,i-8
GOTO   UM1616NA#v(i)
i = i + 1
endw
CLRF   ACCB0    ; if we get here, BARG = 0
CLRF   ACCB1
RETEW  0

UM1616NA0
RRF    ACCB0
RRF ACCB1
RRF ACCB2
i = 1
while i < 8
BTFSS BARGBl,i
GOTO UM1616NA#v(i)
UM1616A#v(i) MOVF TEMPBl,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
UM1616NA#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
i = i + 1
endw
i = 8
while i < 16
BTFSS BARGBO,i-8
GOTO UM1616NA#v(i)
UM1616A#v(i) MOVF TEMPBl,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
UM1616NA#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
RRF ACCB3
i = i + 1
endw
endm

UMUL1515 macro
; Max Timing: 7+7*11+7*12+4 = 172 clks
; Min Timing: 1+16+14+4 = 35 clks
; PM: 1+2*8+2+7*6+7*11+7*12+4 = 202
variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 8
BTFSC BARGBl,i
GOTO UM1515NA#v(i)
i = i + 1
endw
i = 8
while i < 15
BTFSC BARGBO,i-8
GOTO UM1515NA#v(i)
i = i + 1
endw
CLRFA ACCB0 ; if we get here, BARG = 0
CLRFA ACCB1
RETFW 0
UM1515NAO RRF ACCB0
RRF ACCB1
RRF ACCB2
i = 1
while i < 8
BTFSS BARGBl,i
GOTO UM1515NA#v(i)
UM1515A#v(i) MOVF TEMPBl,W
ADDWF ACCB1
AN617

; 16x16 Bit Signed Fixed Point Multiply 16x16 -> 32
; Input: 16 bit signed fixed point multiplicand in AARGBO
; 16 bit signed fixed point multiplier in BARGBO
; Use: CALL FXM1616S
; Output: 32 bit signed fixed point product in AARGBO
; Result: AARG <--- AARG x BARG
; Max Timing: 9+249+2 = 260 clks  B > 0
; 18+249+2 = 269 clks  B < 0
; Min Timing: 9+96 = 105 clks
; PM: 18+55+1 = 74  DM: 9

FXM1616S
BTFSS BARGBO, MSB
GOTO M1616SOK ; make multiplier BARG > 0
COMF BARGB1
INCF BARGB1
BTFSC _2
DECFS BARGB0
COMF BARGB0
COMF AARGB1
INCF AARGB1
BTFSC _3
DECFS AARGBO
COMF AARGBO
M1616SOK
CLRF ACCB2 ; clear partial product
CLRF ACCB3
MOVF AARGBO,W
MOVWF TEMPB0
MOVF AARGB1,W
MOVWF TEMPB1
SMUL1616L
RETLW 0x00 ;******************************************************************************
;******************************************************************************
16x16 Bit Unsigned Fixed Point Multiply 16x16 -> 32
Input: 16 bit unsigned fixed point multiplicand in AARGBO
16 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM1616U
Output: 32 bit unsigned fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 6+248+2 = 256 clks
Min Timing: 6+101 = 107 clks
PM: 6+51+1 = 58

FXM1616U
CLRF ACCB2 ; clear partial product
CLRF ACCB3
MOVF AARGBO,W
MOVWF TEMPBO
MOVF AARGBl,W
MOVWF TEMPB1
UMUL1616L
RETLW 0x00

15x15 Bit Unsigned Fixed Point Divide 15x15 -> 30
Input: 15 bit unsigned fixed point multiplicand in AARGBO
15 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM1515U
Output: 30 bit unsigned fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 6+236+2 = 244 clks
Min Timing: 6+97 = 103 clks
PM: 6+56+1 = 63

FXM1515U
CLRF ACCB2 ; clear partial product
CLRF ACCB3
MOVF AARGBO,W
MOVWF TEMPBO
MOVF AARGBl,W
MOVWF TEMPB1
UMUL1515L
RETLW 0x00

16x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines
16x8 PIC16 FIXED POINT MULTIPLY ROUTINES VERSION 1.2
Input: fixed point arguments in AARG and BARG
Output: product AARGxBARG in AARG
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed multiply application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine Clocks Function
FXM1608S 128 16x08 -> 24 bit signed fixed point multiply
FXM1608U 126 16x08 -> 24 bit unsigned fixed point multiply
FXM1507U 114 15x07 -> 22 bit unsigned fixed point multiply
The above timings are based on the looped macros. If space permits,
approximately 24-35 clocks can be saved by using the unrolled macros.
list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions
******************************************************************************
******************************************************************************
END

C.7 16x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines
16x8 PIC16 FIXED POINT MULTIPLY ROUTINES VERSION 1.2
Input: fixed point arguments in AARG and BARG
Output: product AARGxBARG in AARG
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed multiply application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine Clocks Function
FXM1608S 128 16x08 -> 24 bit signed fixed point multiply
FXM1608U 126 16x08 -> 24 bit unsigned fixed point multiply
FXM1507U 114 15x07 -> 22 bit unsigned fixed point multiply
The above timings are based on the looped macros. If space permits,
approximately 24-35 clocks can be saved by using the unrolled macros.
list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions
******************************************************************************
******************************************************************************
END
AN617

RANLD e= 0x1B
TESTCOUNT e= 0x20 ; counter
DATA e= 0x21 ; beginning of test data

; Test suite for 16x8 bit fixed point multiply algorithms

;org 0x0005
MAIN
MOVEW RAMSTART
MOVEF FSR
MEMLOOP CLR FSR
MOVEW RAMSTOP
SUBWF FSR,W
BTFSS _Z
GOTO MEMLOOP
MOVEW 0x45 ; seed for random numbers
MOVEF RANDLO
MOVEW 0x30
MOVEF RANDHI
MOVEW DATA
MOVEF FSR
BCF _RP0
BCF _RP1
BCF _IRP
CALL TFXM1608

SELF
GOTO SELF
RANDOM16 RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVEF TEMPB0
SWAPF RANDHI
SWAPF RANDLO,W
MOVEF TEMPB1
RLF TEMPB1,W
RLF TEMPB1
MOVEF TEMPB1,W
XORWF RANDHI,W
SWAPF RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETFIW 0

; Test suite for TFXM1608
TFXM1608 MOVEW 2
MOVEF TESTCOUNT
M1608LOOP CALL RANDOM16
MOVEF RANDHI,W
MOVEF BARGBO
BCF BARGBO,MSB
MOVEF BARGBO,W
MOVEF INDF
INC FSR
CALL RANDOM16
MOVEF RANDHI,W
MOVEF AARGBO
BCF AARGBO,MSB
MOVEF AARGBO,W
MOVEF INDF
INC FSR
MOVEF RANDLO,W
MOVEF AARGBO
MOVEF INDF
INC FSR

© 1995 Microchip Technology Inc.
CALL FXM1507U
MOVF AARGB0,W
MOVWF INDF
INCF FSR
MOVF AARGB1,W
MOVWF INDF
INCF FSR
MOVF AARGB2,W
MOVWF INDF
INCF FSR
DECFSZ RES~COUNT
GOTO M1608LOOP
RETLW 0x20

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

: 16x08 Bit Multiplication Macros
SMUL1608L macro
; Max Timing: 2+11+5*16+15+4 = 112 clks
; Min Timing: 2+6+5+4 = 47 clks
; DM: 7
MOVLW 0x07
MOVWF LOOPCOUNT
LOOPSM1608A
RRF BARGBO
BTFSC _C
GOTO LSM1608NA
DECFSZ LOOPCOUNT
GOTO LOOPSM1608A
CLRF AARGB0
CLRF AARGB1
RETLW 0x00
LOOPSM1608
RRF BARGBO
BTFSS _C
GOTO LSM1608NA
MOVF TEMPB1,W
ADDWF ACCB1
MOVF TEMPB0,W
BTFSC _C
INCFSZ TEMPB0,W
ADDWF ACCB0
LSM1608NA
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPSM1608
RLF TEMPB0,W
RRF ACCB0
RRF ACCB1
RRF ACCB2
endm
UMUL1608L macro
; Max Timing: 2+13+6*15+14 = 119 clks
; Min Timing: 2+7+5+4 = 54 clks
; DM: 7
MOVLW 0x08
MOVWF LOOPCOUNT
LOOPUM1608A
RRF BARGBO
BTFSC _C
GOTO LUM1608NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM1608A
CLRF AARGB0
CLRF AARGB1
endm

© 1995 Microchip Technology Inc.
```assembly
AN617

    RETLW    0x00

LUM1608NAP
    BCF   _C
    GOTO LUM1608NA

LOOPUM1608
    RRF BARGBO
    BTFSS _C
    GOTO LUM1608NA
    MOVF TEMPB1,W
    ADDWF ACCB1
    MOVF TEMPB0,W
    BTFSC _C
    INCFSZ TEMPB0,W
    ADDWF ACCB0
LUM1608NA
    RRF ACCB0
    RRF ACCB1
    RRF ACCB2
    DECSZ LOOPCOUNT
    GOTO LOOPUM1608
    endm

UMUL1507L macro
; Max Timing:  2+13+5*15+14+3 = 107 clks
; Min Timing: 2*6+6+5*4 = 47 elks
; PM: 29      DM: 7
    MOVLW 0x07
    MOVWF LOOPCOUNT

LOOPUM1507A
    RRF BARGBO
    BTFSC _C
    GOTO LUM1507NAP
    DECSZ LOOPCOUNT
    GOTO LOOPUM1507A
    CLRF AARGBO
    CLRF AARGBl
    RETLW 0x00
LUM1507NAP
    BCF _C
    GOTO LUM1507NA

LOOPUM1507
    RRF BARGBO
    BTFSS _C
    GOTO LUM1507NA
    MOVF TEMPB1,W
    ADDWF ACCB1
    MOVF TEMPB0,W
    BTFSC _C
    INCFSZ TEMPB0,W
    ADDWF ACCB0
LUM1507NA
    RRF ACCB0
    RRF ACCB1
    RRF ACCB2
    DECSZ LOOPCOUNT
    GOTO LOOPUM1507
    CLRF AARGBO
    CLRF AARGBl
    RETLW 0x00

SMUL1608 macro
; Max Timing:  3+6+6*11+3 = 78 clks
; Min Timing: 3+21+5 = 29 clks
; PM: 3+3*7+7+6*11+3 = 100      DM: 6
variable i
i = 0
    BTFSC AARGBO,MSB
    COMF ACCB2
```

© 1995 Microchip Technology Inc.
RLF  ACCB0,W

while i < 7

BTFSC  BARGBO,i
GOTO  SM1608NA#v(i)
BCF  ACCB2,7-i
i = i + 1
endw
CLRF  ACCB0  ; if we get here, BARG = 0
CLRF  ACCB1
CLRF  ACCB2
RETEW  0

SM1608NA0
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
i = 1
while i < 7
BTFSS  BARGBO,i
GOTO  SM1608NA#v(i)
SM1608A#v(i)
MOVF  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPBO,W
BTFSC  _C
INCFSZ  TEMPBO,W
ADDWF  ACCB0
SM1608NA#v(i)
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
i = i + 1
endw
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
endm

UMUL1608
;  Max Timing:  1+6*7*11 = 84 clks
;  Min Timing:  1+2*8+4 = 21 clks
;  PM: 1+2*8+4+6*7 = 63  DM: 4
variable i
i = 0
BCF  _C  ; clear carry for first right shift
while i < 8
BTFSC  BARGBO,i
GOTO  UM1608NA#v(i)
UM1608A0
RRF  ACCB0
RRF  ACCB1
RRF  ACCB2
i = 1
while i < 8
BTFSS  BARGBO,i
GOTO  UM1608NA#v(i)
UM1608A#v(i)
MOVF  TEMPB1,W
ADDWF  ACCB1
MOVF  TEMPBO,W
BTFSC  _C
INCFSZ  TEMPBO,W
ADDWF ACCB0
UM1608#v(i) RRF ACCB0
RRF ACCB1
RRF ACCB2
i = i + 1
endw

UMUL1507
;
Max Timing: 7+6*12+4 = 83 clks
;
Min Timing: 14+3 = 17 clks
;
PM: 2*7+7+6*12+4 = 97 DM: 6
variable i
i = 0
BCF _C ; clear carry for first right shift
while i < 7

BTFSC BARGBO,i
GOTO UM1507NA#v(i)
i = i + 1
endw
CLRF ACCBO if we get here, BARG 0
CLRF ACCBl
RETF1W 0
UM1507NA0
RRF ACCB0
RRF ACCB1
RRF ACCB2
i = 1
while i < 7
BTFSS BARGBO,i
GOTO UM1507NA#v(i)

UM1507A#v(i) MOVF TEMPBl,W
ADDWF ACCBl
MOVF TEMPBO,W
BTFSC _C
INCFSZ TEMPBO,W
ADDWF ACCBO
UM1507NA#v(i) RRF ACCBO
RRF ACCBl
RRF ACCB2
i = i + 1
endw

;***************************************************************************;
;***************************************************************************;
16x8 Bit Signed Fixed Point Multiply 16x8 -> 24
Input: 16 bit signed fixed point multiplicand in AARGBO
8 bit signed fixed point multiplier in BARGBO
Use: CALL FXM1608S
Output: 24 bit signed fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 8+112+2 = 122 clks B > 0
14+112+2 = 128 clks B < 0
Min Timing: 8+47 = 55 clks
PM: 14+29+1 = 44 DM: 7
FXM1608S BTFSS BARGBO,MSB
GOTO M1608SOK
COMF BARGBO ; make multiplier BARG > 0
INCIF BARGBO
INCIF AARGBl
INCIF AARGBl
BTFSS _Z
C.8 8x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

8x8 PIC16 FIXED POINT MULTIPLY ROUTINES VERSION 1.2
Input: fixed point arguments in AARG and BARG
Output: product AARGxBARG in AARG
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed multiply application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine Clocks Function
FXM0808S 82 08x08 -> 16 bit signed fixed point multiply
FXM0808U 73 08x08 -> 16 bit unsigned fixed point multiply
FXM0707U 67 07x07 -> 14 bit unsigned fixed point multiply
The above timings are based on the looped macros. If space permits,
approximately 29-35 clocks can be saved by using the unrolled macros.
list  r=dec,x=on,t=off
include <PIC16.INC>  ; general PIC16 definitions
include <MATH16.INC>  ; PIC16 math library definitions

;******************************************************************************************
;******************************************************************************************
Test suite storage
RANDHI  equ  0x1A  ; random number generator registers
RANDLO  equ  0x1B
TESTCOUNT  equ  0x20  ; counter
DATA  equ  0x21  ; beginning of test data
;******************************************************************************************
;******************************************************************************************
;  Test suite for 8x8 bit fixed point multiply algorithms
org  0x0005
MAIN
  MOVILW  RAMSTART
  MOVWF  FSR
MEMLOOP
  CLR  INDF
  INC  FSR
  MOVILW  RAMSTOP
  SUBWF  FSR,W
  BTFSS  _Z
  GOTO  MEMLOOP
  MOVILW  0x45     ; seed for random numbers
  MOVWF  RANDLO
  MOVILW  0x30
  MOVWF  RANDHI
  MOVILW  DATA
  MOVWF  FSR
  BCF  _RP0
  BCF  _RP1
  BCF  _IRP
  CALL  TFM0808
  GOTO  SELF
SELF
  CALL  TFM0808
RANDOM16
  RLF  RANDHI,W  ; random number generator
  XORWF  RANDHI,W
  MOVWF  TEMPB0
  SWAPF  RANDHI
  SWAPF  RANDH,W
  MOVWF  TEMPB1
  RLF  TEMPB1,W
  RLF  TEMPB1
  MOVF  TEMPB1,W
  XORWF  RANDHI,W
  SWAPF  RANDHI
  ANDLW  0x01
  RLF  TEMPB0
  RLF  RANDLO
  XORWF  RANDLO
  RLF  RANDHI
  RETEW  0
;  Test suite for FFM0808
TFM0808
  MOVILW  3  TESTCOUNT
M0808LOOP
  CALL  RANDOM16
  MOVF  RANDHI,W
  MOVWF  BARGBO
  BCF  BARGBO,MSB
  MOVF  BARGBO,W
  MOVWF  INDF
  INC  FSR
  CALL  RANDOM16
  MOVF  RANDHI,W
  MOVWF  AARGBO
  BCF  AARGBO,MSB
MOVF AARGBO,W
MOVF AARGBO,W
INCF FSR
INCF FSR
MOVF AARGBO,W
MOVF AARGBO,W
INCF FSR
DECFSZ TESTCOUNT
GOTO M0808LOOP

;********************************************************************************************
;*******************************************************************************************
; 08x08 Bit Multiplication Macros
SMUL0808L macro
; Max Timing: 7+10+5+9+8+3 = 73 clks
; Min Timing: 7+6+6+5+3 = 51 clks
; PM: 25 DM: 5
MOVLW 0x07
MOVWF LOOPCOUNT
CLRW
BTFSC AARGBO,MSB
MOVLW 0xFF
MOVWF SIGN
MOVF AARGBO,W
LOOPSM0808A
RRF BARGBO
BTFSC _c
GOTO LSM0808NA
DECFSZ LOOPCOUNT
GOTO LOOPSM0808A
CLRF AARGBO
RETLW 0x00

LOOPSM0808
RRF BARGBO
BTFSC _c
ADDFW ACCB0
LSM0808NA
RRF SIGN
RRF ACCB0
RRF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPSM0808
RLF SIGN
RRF ACCB0
RRF ACCB1
endm

UMUL0808L macro
; Max Timing: 3+12+6*8+7 = 70 clks
; Min Timing: 3+7*6+5+3 = 53 clks
; PM: 19 DM: 4
MOVLW 0x08
MOVWF LOOPCOUNT
MOVF AARGBO,W
LOOPUM0808A
RRF BARGBO
BTFSC _c
GOTO LUM0808NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM0808A
CLRF AARGBO
RETLW 0x00

LUM0808NAP
BCF _c
GOTO LUM0808NA
AN617

LOOPUM0808

RRF BARGBO
BTFSC _C
ADDFW ACCB0

LUM0808NA

RRF ACCB0
RRF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPUM0808

endm

UMUL0707L

macro
; Max Timing: 3+12+5*8+7+2 = 64 clks
; Min Timing: 3+6*6+5+3 = 47 clks
; PM: 21

MOVLW 0x07
MOVF ACCB0,W

LOOPUM0707A

RRF BARGBO
BTFSC _C
GOTO LUM0707NAP
DECFSZ LOOPCOUNT
GOTO LOOPUM0707A
CLRWF AARGBO
RETLW 0x00

LUM0707NAP

BCF _C
GOTO LUM0707NA

LOOPUM0707

RRF BARGBO
BTFSC _C
ADDFW ACCB0

LUM0707NA

RRF ACCB0
RRF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPUM0707
RRF ACCB0
RRF ACCB1

endm

SMUL0808

macro
; Max Timing: 5+6*6*5+3 = 44 clks
; Min Timing: 5+14+3 = 22 clks
; PM: 5+2*7+5*6*5+3 = 57

variable i
i = 0
CLRWF
BTFSC AARGBO,MSB
MOVLW 0xFF
MOVF ACCB0,W

while i < 7

BTFSC BARGBO,i
GOTO SM0808NA#v(i)

endw
CLRWF
RETLW 0

SM0808NA0

RLF SIGN
RRF ACCB0
RRF ACCB1
i = 1
while i < 7
BTFSC BARGBO,i
ADDFWF ACCB0

DS00617A-page 2-250 © 1995 Microchip Technology Inc.
SM0808NA#v(i)  RLF    SIGN
    RRF    ACCB0
    RRF    ACCB1
    i = i + 1
endw
    RLF    SIGN
    RRF    ACCB0
    RRF    ACCB1
endm

UMUL0808 macro
    ; Max Timing:  2+5+7*4 = 35 clks
    ; Min Timing:  2+16+3 = 21 clks
    ;  PM: 2+2*8+4*7*4 = 50
variable i
    i = 0
    BCF    _C ; clear carry for first right shift
    MOVF   AARGBO,W
while i < 8
    BTFSC   BARGBO,i
    GOTO    UM0808NA#v(i)
    i = i + 1
endw
    CLRF   ACCB0 ; if we get here, BARG = 0
RETEW  0

UM0808NA0
    RRF    ACCB0
    RRF    ACCB1
    i = 1
while i < 8
    BTFSC   BARGBO,i
    ADDWF   ACCB0
UM0808NA#v(i)  RRF    ACCB0
    RRF    ACCB1
    i = i + 1
endw
endm

UMUL0707 macro
    ; Max Timing:  2+5+6*4+2 = 33 clks
    ; Min Timing:  2+14+3 = 19 clks
    ;  PM: 2+2*7+4*6*4+2 = 46
variable i
    i = 0
    BCF    _C ; clear carry for first right shift
    MOVF   AARGBO,W
while i < 7
    BTFSC   BARGBO,i
    GOTO    UM0707NA#v(i)
    i = i + 1
endw
    CLRF   ACCB0 ; if we get here, BARG = 0
RETEW  0

UM0707NA0
    RRF    ACCB0
    RRF    ACCB1
    i = 1
while i < 7
    BTFSC   BARGBO,i
    ADDWF   ACCB0
UM0707NA#v(i)  RRF    ACCB0
    RRF    ACCB1
    i = i + 1
endw
RRF  ACCB0
RRF  ACCB1
endm
8x8 Bit Signed Fixed Point Multiply 8x8 -> 16

Input: 8 bit signed fixed point multiplicand in AARGBO
8 bit signed fixed point multiplier in BARGBO
Use: CALL FXMO808S
Output: 16 bit signed fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 4+73+2 = 79 clks
Min Timing: 7+73+2 = 82 clks
PM: 7+25+1 = 33

FXMO808S
BTFSS BARGBO, MSB
GOTO M0808SOK
COMF BARGBO
; make multiplier BARG > 0
INCF BARGBO
INCF AARGBO
M0808SOK
CLRF ACCB1
; clear partial product
SMUL0808L
RETLW 0x00

8x8 Bit Unsigned Fixed Point Multiply 8x8 -> 16

Input: 8 bit unsigned fixed point multiplicand in AARGBO
8 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM0808U
Output: 8 bit unsigned fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 1+70+2 = 73 clks
Min Timing: 1+53 = 54 clks
PM: 1+19+1 = 21

FXM0808U
CLRF ACCB1
; clear partial product
UMUL0808L
RETLW 0x00

7x7 Bit Unsigned Fixed Point Divide 7x7 -> 14

Input: 7 bit unsigned fixed point multiplicand in AARGBO
7 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM0707U
Output: 14 bit unsigned fixed point product in AARGBO
Result: AARG <-- AARG x BARG
Max Timing: 1+64+2 = 67 clks
Min Timing: 1+47 = 48 clks
PM: 1+21+1 = 23

FXM0707U
CLRF ACCB1
; clear partial product
UMUL0707L
RETLW 0x00

END
APPENDIX D: PIC16C5X/PIC16CXX DIVIDE ROUTINES

Table of Contents for Appendix D

D.1 32/32 PIC16C5X/PIC16CXX Fixed Point Divide Routines .......................................................... 2-254
D.2 32/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines .......................................................... 2-273
D.3 32/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines .......................................................... 2-288
D.4 24/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines .......................................................... 2-301
D.5 24/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines .......................................................... 2-313
D.6 16/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines .......................................................... 2-323
D.7 16/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines ............................................................ 2-331
D.8 8/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines .............................................................. 2-340

D.1 32/32 PIC16C5X/PIC16CXX Fixed Point Divide Routines

; 32/32 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
; Input: fixed point arguments in AARG and BARG
; Output: quotient AARG/BARG followed by remainder in REM
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed divide application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in performance.
; Routine    Clocks    Function
; FXD3232S     912       32 bit/32 bit -> 32.32 signed fixed point divide
; FXD3232U    1031       32 bit/32 bit -> 32.32 unsigned fixed point divide
; FXD3131U     869       31 bit/31 bit -> 31.31 unsigned fixed point divide

list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

******************************************************************************************
Test suite storage
RANDHI equ Ox1E ; random number generator registers
RANDLO equ Ox1F
DATA equ Ox20
******************************************************************************************
Test suite for 32/32 bit fixed point divide algorithms
org Ox0005
MAIN
 MOVLW RAMSTART
 MOVF FSR

MEMLOOP
 CLR   IND
 INC   FSR
 MOVLW RAMSTOP
 SUBWF FSR,W
 BTFS  _Z
 GOTO MEMLOOP
 MOVLW 0x45 ; seed for random numbers
 MOVF RANDLO
 MOVLW 0x30
 MOVF RANDHI
 MOVLW DATA
 MOVF FSR
 BCF _RP0
 BCF _RP1
 BCF _RP2
 CALL RANDOM16
 MOVF RANDHI,W
 MOVF BARGB0
 BCF BARGB0,MSB
; MOVF BARGB0,W
 MOVF DATA
 MOVF FSR
 INC   IND
 MOVF RANDLO,W
 MOVF BARGB1
AN617

MOVWF INDF
INC FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGB2
MOVWF INDF
INC FSR
MOVF RANDLO,W
MOVWF BARGB3
MOVWF INDF
INC FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF AARGB0
;
BCF AARGB0,MSB
;
MOVF AARGB0,W
MOVWF INDF
INC FSR
MOVF RANDLO,W
MOVWF AARGB1
MOVWF INDF
INC FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF AARGB2
MOVWF INDF
INC FSR
MOVF RANDLO,W
MOVWF AARGB3
MOVWF INDF
INC FSR
CALL FXD3232S
MOVF AARGB0,W
MOVWF INDF
INC FSR
MOVF AARGB1,W
MOVWF INDF
INC FSR
MOVF AARGB2,W
MOVWF INDF
INC FSR
MOVF AARGB3,W
MOVWF INDF
INC FSR
MOVF REMB0,W
MOVWF IND F
INC FSR
MOVF REMB1,W
MOVWF IND F
INC FSR
MOVF REMB2,W
MOVWF IND F
INC FSR
MOVF REMB3,W
MOVWF IND F
INC FSR
SELF GOTO SELF
RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMPB0
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPB1
RLF TEMPB1,W
RLF TEMPB1

© 1995 Microchip Technology Inc.
AN617

MOVF TEMPB1,W
XORWF RANDHI,W
SWAPF RANDHI
ANDLF 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETLW 0

;**********************************************************************************************
;**********************************************************************************************
; 32/32 Bit Division Macros
SDIV3232L macro
; Max Timing: 17+6*27+26+6*27+26+6*27+26+6*27+26+6*27+26+16 = 863 elks
; PM: 17+7*38+16 = 299 DM: 13

MOVF BARGB3,W
SUBWF REMB3
MOVF BARGB2,W
BTFS _C
INCF _Z
SUBWF REMB2
MOVF BARGB1,W
BTFS _C
INCF _Z
SUBWF REMB1
MOVF BARGB0,W
BTFS _C
INCF _Z
SUBWF REMB0
RLF ACCB0
MOVLW 7
MOVF LOOPCOUNT
RLF ACCB0,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB3,W
BTFS ACCB0,LSB
GOTO SADD22LA
SUBWF REMB3
MOVF BARGB2,W
BTFS _C
INCF _Z
SUBWF REMB2
MOVF BARGB1,W
BTFS _C
INCF _Z
SUBWF REMB1
MOVF BARGB0,W
BTFS _C
INCF _Z
SUBWF REMB0
GOTO SOK22LA
ADDWF REMB3
MOVF BARGB2,W
BTFS _C
INCF _Z
ADDWF REMB2
MOVF BARGB1,W
BTFS _C
INCF _Z
ADDWF REMB1
MOVF BARGB0,W

DS00617A-page 2-256 © 1995 Microchip Technology Inc.
BTFSC _C
INCFSZ BARGBO,W
ADDFP REMBO

SOK22L8
RLF ACCB0
DECFSZ LOOPCOUNT
GOTO LOOP3232A
RLF ACCB1,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB3,W
BTFSS ACCBO,LSB
GOTO SADD22L8
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SOK22L8

SADD22L8
ADDFP REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDFP REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDFP REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDFP REMB0

SOK22L8
RLF ACCB1
MOVLW 7
MOVF LOOPCOUNT

LOOP3232B
RLF ACCB1,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB3,W
BTFSS ACCB1,LSB
GOTO SADD22L8
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W

© 1995 Microchip Technology Inc.
SUBWF  REMBO
GOTO  SOK22LB
ADDWF  REMB3
MOVF  BARGB2,W
BTFSC  _C
INCFSZ  BARGB2,W
ADDWF  REMB2
MOVF  BARGB1,W
BTFSC  _C
INCFSZ  BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
ADDWF  REMBO
SOK22LB
RLF  ACCB1
DECFSZ  LOOPCOUNT
GOTO  LOOPS3232B
RLF  ACCB2,W
RLF  REMB3
RLF  REMB2
RLF  REMB1
RLF  REMB0
MOVF  BARGB3,W
BTFSS  ACCB1,LSB
GOTO  SADD22L16
SUBWF  REMB3
MOVF  BARGB2,W
BTFSS  _C
INCFSZ  BARGB2,W
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCFSZ  BARGB1,W
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
SUBWF  REMB0
GOTO  SOK22L16
SADD22L16
ADDWF  REMB3
MOVF  BARGB2,W
BTFSS  _C
INCFSZ  BARGB2,W
ADDWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCFSZ  BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
ADDWF  REMBO
SOK22L16
RLF  ACCB2
MOVLW  7
MOVWF  LOOPCOUNT
LOOPS3232C
RLF  ACCB2,W
RLF  REMB3
RLF  REMB2
RLF  REMB1
RLF  REMB0
MOVF  BARGB3,W
BTFSS  ACCB2,LSB
GOTO  SADD22LC
AN617

SADD22LC
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SADD22L24
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

SOK22LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPCOUNT232LC
RLF ACCB3,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB3,W
BTFSS ACCB2,LSB
GOTO SADD22L24
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SADD22L24
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
SOK22L24

RLF
MOVlw 7
MOVWF LOOPCOUNT

LOOPS3232D

RLF ACCB3,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB3,W
BTFSS ACCB3,LSB
GOTO SADD22LD
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SOK22LD

SADD22LD

ADDF W REMB3
MOVF BARGB2,W
BTFSC _C
INCF SZ BARGB2,W
ADDF W REMB2
MOVF BARGB1,W
BTFSC _C
INCF SZ BARGB1,W
ADDF W REMB1
MOVF BARGB0,W
BTFSC _C
INCF SZ BARGB0,W
ADDF W REMB0

SOK22LD

RLF ACCB3
DECFSZ LOOPCOUNT
GOTO LOOPS3232D
BTFSC ACCB3,LSB
GOTO SOK22LD
MOVF BARGB3,W
ADDF W REMB3
MOVF BARGB2,W
BTFSC _C
INCF BARGB2,W
ADDF W REMB2
MOVF BARGB1,W
BTFSC _C
INCF BARGB1,W
ADDF W REMB1
MOVF BARGB0,W
BTFSC _C
INCF BARGB0,W
ADDF W REMB0

SOK22LD

endm

UDIV3232L macro

; Max Timing: 24+6*32+31+31+31+6*32+31+6*32+31+31+6*32+31+16 = 1025 clks
; Min Timing: 24+6*31+30+30+6*31+30+6*31+30+30+6*31+30+3 = 981 clks
; PM: 359
; DM: 13

CLRF TEMP
AN617

RLF  ACCB0,W
RLF  REMB3
MOVF  BARGB3,W
SUBWF  REMB3
MOVF  BARGB2,W
BTFSS  _C
INCFSZ  BARGB2,W
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCFSZ  BARGB1,W
SUBWF  REMB1
MOVF  BARGBO,W
BTFSS  _C
INCFSZ  BARGBO,W
SUBWF  REMBO
CLRW
BTFSS  _C
MOVLW  1
SUBWF  TEMP
RLF  ACCB0
MOVLW  7
MOVF  LOOPCOUNT
LOOPU3232A
RLF  ACCB0,W
RLF  REMB3
RLF  REMB2
RLF  REMB1
RLF  REMB0
RLF  TEMP
MOVF  BARGB3,W
BTFSS  ACCB0,LSB
GOTO  UADD22LA
SUBWF  REMB3
MOVF  BARGB2,W
BTFSS  _C
INCFSZ  BARGB2,W
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCFSZ  BARGB1,W
SUBWF  REMB1
MOVF  BARGBO,W
BTFSS  _C
INCFSZ  BARGBO,W
SUBWF  REMBO
CLRW
BTFSS  _C
MOVLW  1
SUBWF  TEMP
GOTO  UOK22LA
UADD22LA
ADDWF  REMB3
MOVF  BARGB2,W
BTFSC  _C
INCFSZ  BARGB2,W
ADDWF  REMB2
MOVF  BARGB1,W
BTFSC  _C
INCFSZ  BARGB1,W
ADDWF  REMB1
MOVF  BARGBO,W
BTFSC  _C
INCFSZ  BARGBO,W
ADDWF  REMBO
CLRW
BTFSC  _C
MOVLW  1
ADDWF TEMP

UOK22LA
ADDWF TEMP
ADDWF REMB3
ADDWF REMB2
ADDWF REMB1
ADDWF REMB0
ADDWF TEMP
MOVF BARGB3,W
BTFSS ACCB0,LSB
GOTO UADD22L8
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK22L8

UADD22L8
ADDF M TEMP
ADDF M REMB3
MOV BARGB2,W
BTFSC _C
INCFZ BARGB2,W
ADDWF REMB2
MOV BARGB1,W
BTFSC _C
INCFZ BARGB1,W
ADDWF REMB1
MOV BARGB0,W
BTFSC _C
INCFZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDF M TEMP

UOK22L8
RLF ACCB1
MOVLW 7
MOVF LOOPCOUNT
LOOPU3232B
RLF ACCB1,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOV BARGB3,W
BTFSS ACCB1,LSB
GOTO UADD22L8
SUBWF REMB3
MOV BARGB2,W
BTFSS _C
INCFZ BARGB2,W
AN617

SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK22LB

UADD22LB
ADDF W REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDF W REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDF W REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDF W REMB0
CLRW
BTFSC _C
MOVLW 1
ADDF W TEMP

UOK22LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPU3232B
RLF ACCB2,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF TEM P
MOVF BARGB3,W
BTFSS ACCB1,LSB
GOTO UADD22L16
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK22L16

UADD22L16
ADDF W REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK22L16
RLF ACCB2
MOVLW 7
MOVF LOOPCOUNT

LOOPU3232C
RLF ACCB2,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB3,W
BTFSS ACCB2,LSB
GOTO UADD22LC
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UADD22LC

UADD22LC
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK22LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPU3232C
RLF ACCB3,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB3,W
BTFSS ACCB2,LSB
GOTO UADD22L24
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK22L24
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP
UOK22L24
RLF ACCB3
MOVLW 7
MOVF LOOPCOUNT
LOOPU3232D
RLF ACCB3,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB3,W
BTFSS ACCB3,LSB
GOTO UADD22LD
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
AN617

BTFSS _C
INCFSZ BARGB0,W
SUBWF REMBO
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK22LD
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UADD22LD
RLF ACCB3
DECF #_C
GOTO LOOPCOUNT
BTFSC ACCB3,LSB
GOTO UOK22LD
MOVF BARGB3,W
ADDWF REMB3
MOVF BARGB2,W
BTFSC _C
INCF BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCF BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCF BARGB0,W
ADDWF REMB0

UOK22LD
endm
UDIV3131L macro
; Max Timing: 17+6*27+26+26+6*27+26+26+6*27+26+26+6*27+26+16 = 863 clks
; PM: 17+7*38+16 = 299 DM: 13
MOVF BARGB3,W
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
RLF ACCB0

© 1995 Microchip Technology Inc.
MOVLW 7
MOVWF LOOPCOUNT

LOOPU3131A
RLF ACCBO,W
RLF REMB3
RLF REMB2
RLF REMB1
MOVF BARGB3,W
BTFSS ACCBO,LSB
GOTO UADD11LA
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFNZ REMB3
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFNZ REMB2
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFNZ REMB1
SUBWF REMB0
GOTO UOK11LA

UADD11LA
ADDF W REMB3
MOVF BARGB2,W
BTFSC _C
INCFNZ BARGB2,W
ADDF W REMB2
MOVF BARGB1,W
BTFSC _C
INCFNZ BARGB1,W
ADDF W REMB1
MOVF BARGB0,W
BTFSC _C
INCFNZ BARGB0,W
ADDF W REMB0

UOK11LA
RLF ACCBO
DECFZ LOOPCOUNT
GOTO LOOPU3131A
RLF ACCBL,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB3,W
BTFSS ACCBO,LSB
GOTO UADD11L8
SUBWF REMB3
MOVF BARGB2,W
BTFSS _C
INCFNZ BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFNZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFNZ BARGB0,W
SUBWF REMB0
GOTO UOK11L8

UADD11L8
ADDF W REMB3
MOVF BARGB2,W
BTFSC _C
AN617

INCFSZ BARGB2,W
ADDF W REMB2
MOV F BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDF W REMB1
MOV F BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDF W REMB0

UOK11LB
RLF ACCB1
MOVL W 7
MOVF LOOPCOUNT

LOOPU3131B
RLF ACCB1,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOV F BARGB3,W
BTFSS ACCB1,LSB
GOTO UADD11LB
SUBWF REMB3
MOV F BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
MOV F BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOV F BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK11LB

UADD11LB
ADDF W REMB3
MOV F BARGB2,W
BTFSC _C
INCFSZ BARGB2,W
ADDF W REMB2
MOV F BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDF W REMB1
MOV F BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDF W REMB0

UOK11LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPU3131B
RLF ACCB2,W
RLF REMB3
RLF REMB2
RLF REMB1
RLF REMB0
MOV F BARGB3,W
BTFSS ACCB1,LSB
GOTO UADD11LB
SUBWF REMB3
MOV F BARGB2,W
BTFSS _C
INCFSZ BARGB2,W
SUBWF REMB2
<table>
<thead>
<tr>
<th>Line</th>
<th>Move</th>
<th>Source</th>
<th>Condition</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVF</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBWF</td>
<td>REMb1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBWF</td>
<td>REMb0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>UOK111L16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb2, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb2, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UOK111L16</td>
<td>RLF</td>
<td>ACCb2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVWF</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVWF</td>
<td>LOOPCOUNT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOPU3131C</td>
<td>RLF</td>
<td>ACCb2, W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLF</td>
<td>REMb3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLF</td>
<td>REMb2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLF</td>
<td>REMb1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLF</td>
<td>REMb0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>ACCb2, LSB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>UADD111LC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBWF</td>
<td>REMb3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb2, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb2, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBWF</td>
<td>REMb2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBWF</td>
<td>REMb1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBWF</td>
<td>REMb0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>UOK111LC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UADD111LC</td>
<td>ADDWF</td>
<td>REMb3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb2, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb2, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb1, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTFSS</td>
<td>_C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCFSZ</td>
<td>BARGb0, W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDWF</td>
<td>REMb0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UOK111LC</td>
<td>RLF</td>
<td>ACCb2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECFSZ</td>
<td>LOOPCOUNT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>LOOPU3131C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLF</td>
<td>ACCb3, W</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 1995 Microchip Technology Inc.
RLF    REMB3
RLF    REMB2
RLF    REMB1
RLF    REMB0
MOVF   BARGB3,W
BTFSS  ACCB2,LSB
GOTO   UADD11L24
SUBWF  REMB3
MOVF   BARGB2,W
BTFSS  _C
INCFSZ BARGB2,W
SUBWF  REMB2
MOVF   BARGB1,W
BTFSS  _C
INCFSZ BARGB1,W
SUBWF  REMB1
MOVF   BARGB0,W
BTFSS  _C
INCFSZ BARGB0,W
SUBWF  REMB0
GOTO   UOK11L24
UADD11L24 ADDWF  REMB3
MOVF   BARGB2,W
BTFSC  _C
INCFSZ BARGB2,W
ADDWF  REMB2
MOVF   BARGB1,W
BTFSC  _C
INCFSZ BARGB1,W
ADDWF  REMB1
MOVF   BARGB0,W
BTFSC  _C
INCFSZ BARGB0,W
ADDWF  REMB0
UOK11L24 RLF    ACCB3
MOVlw  7
MOVWF  LOOPCOUNT
LOOPU3131D RLF    ACCB3,W
RLF    REMB3
RLF    REMB2
RLF    REMB1
RLF    REMB0
MOVF   BARGB3,W
BTFSS  ACCB3,LSB
GOTO   UADD11LD
SUBWF  REMB3
MOVF   BARGB2,W
BTFSS  _C
INCFSZ BARGB2,W
SUBWF  REMB2
MOVF   BARGB1,W
BTFSS  _C
INCFSZ BARGB1,W
SUBWF  REMB1
MOVF   BARGB0,W
BTFSS  _C
INCFSZ BARGB0,W
SUBWF  REMB0
GOTO   UOK11LD
UADD11LD ADDWF  REMB3
MOVF   BARGB2,W
BTFSC  _C
INCFSZ BARGB2,W
ADDWF  REMB2
MOVF   BARGB1,W
BTFSC _c
INCFSZ BARGBl,W
ADDWF REMBl
BTFSC _c
INCFSZ BARGBO,W
ADDWF REMBO
UOK11LD
RLF ACCB3
DECFSZ LOOPCOUNT
GOTO LOOPU3131D
BTFSC ACCB3,LSB
GOTO UOK11L
MOVF BARGB3,W
ADDWF REMB3
MOVF BARGB2,W
BTFSC _c
INCF BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _c
INCF BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _c
INCF BARGB0,W
ADDWF REMBO
endm

;********************************************************************************************
;********************************************************************************************
; 32/32 Bit Signed Fixed Point Divide 32/32 -> 32.32
; Input: 32 bit fixed point dividend in AARGBO, AARGBl,AARGB2,AARGB3
; 32 bit fixed point divisor in BARGBO, BARGBl, BARGB2, BARGB3
; Use: CALL FXD3232S
; Output: 32 bit fixed point quotient in AARGBO, AARGBl,AARGB2,AARGB3
; 32 bit fixed point remainder in REMB0, REMB1, REMB2, REMB3
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 13+863+3 = 879 clks A > 0, B > 0
; 23+863+26 = 912 clks A > 0, B < 0
; 23+863+26 = 912 clks A < 0, B > 0
; 33+863+3 = 899 clks A < 0, B < 0
; Min Timing: 13+819+3 = 835 clks A > 0, B > 0
; 23+819+26 = 868 clks A > 0, B < 0
; 23+819+26 = 868 clks A < 0, B > 0
; 33+819+3 = 855 clks A < 0, B < 0
; PM: 33+299+25 = 357 DM: 13
FXD3232S
MOVF AARGB0,W
XORWF BARGB0,W
MOVF AARGB1,W
MOVF AARGB2,W
MOVF AARGB3,W
BTFSS BARGB0,MSB ; if MSB set, negate BARG
GOTO CA3232S
COMF BARGB3
INCF BARGB3
BTFSS _Z
DECF BARGB2
COMF BARGB2
BTFSS _Z
DECF BARGB1
COMF BARGB1
BTFSS _Z
DECF BARGB0
COMF BARGB0

CA3232S
BTFSS AARGB0,MSB ; if MSB set, negate AARG
GOTO C3232S
COMF AARGB3
INCF AARGB3
BTFSC _2
DECF AARGB2
COMF AARGB2
BTFSC _2
DECF AARGB1
COMF AARGB1
BTFSC _2
DECF AARGB0
COMF AARGB0

C3232S
CLRF REMB0
CLRF REMB1
CLRF REMB2
CLRF REMB3
SDIV3232L
BTFSS SIGN,MSB
RETLW Ox00
COMF AARGB3
INCF AARGB3
BTFSC _2
DECF AARGB2
COMF AARGB2
BTFSC _2
DECF AARGB1
COMF AARGB1
BTFSC _2
DECF AARGB0
COMF AARGB0
COMF REMB3
INCF REMB3
BTFSC _2
DECF REMB2
COMF REMB2
BTFSC _2
DECF REMB1
COMF REMB1
BTFSC _2
DECF REMB0
COMF REMB0
RETLW Ox00

;**********************************************************************************************
;**********************************************************************************************
32/32 Bit Unsigned Fixed Point Divide 32/32 -> 32.32
Input: 32 bit unsigned fixed point dividend in AARG0, AARG1,AARG2,AARG3
32 bit unsigned fixed point divisor in BARG0, BARG1, BARG2, BARG3
Use: CALL FXD3232U
Output: 32 bit unsigned fixed point quotient in AARG0, AARG1,AARG2,AARG3
32 bit unsigned fixed point remainder in REM0, REM1, REM2, REM3
Result: AARG, REM <-- AARG / BARG
Max Timing: 4*1025+2 = 1031 clks
Max Timing: 4*981+2 = 987 clks
PM: 4*359+1 = 364 DM: 13

FXD3232U
CLRFR REMO
CLRFR REM1
CLRFR REM2
CLRFR REM3
UDIV3232L
RETLW Ox00

;**********************************************************************************************
;**********************************************************************************************
31/31 Bit Unsigned Fixed Point Divide 31/31 -> 31.31
Input: 31 bit unsigned fixed point dividend in AARG0, AARG1,AARG2,AARG3
31 bit unsigned fixed point divisor in BARG0, BARG1, BARG2, BARG3
Use: CALL FXD3131U
Output: 31 bit unsigned fixed point quotient in AARGBO, AARGB1, AARGB2, AARGB3
Result: AARG, REM \leftarrow AARG / BARG
Max Timing: \( 4+863+2 = 869 \) clks
Min Timing: \( 4+819+2 = 825 \) clks
PM: \( 4+299+1 = 304 \)
DM: 13

FXD3131U
    CLRF REMB0
    CLRF REMB1
    CLRF REMB2
    CLRF REMB3
    UDIV3131L
    RETLW 0x00

;**********************************************************************************************
END

D.2 32/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines

32/24 PIC16 FIXED POINT DIVIDE Routines VERSION 1.7

Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM

All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed divide application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.

Routine Clocks Function
FXD3224S 742 32 bit/24 bit \rightarrow 32.24 signed fixed point divide
FXD3224U 867 32 bit/24 bit \rightarrow 32.24 unsigned fixed point divide
FXD3123U 705 31 bit/23 bit \rightarrow 31.23 unsigned fixed point divide

list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

;**********************************************************************************************
;**********************************************************************************************
Test suite storage
RANDHI equ 0x1E ; random number generator registers
RANDLO equ 0x1F
DATA equ 0x20

;**********************************************************************************************
; Test suite for 32/24 bit fixed point divide algorithms

org 0x0005
MAIN
    MOVLW RAMSTART
    MOVWF FSR
MEMLOOP
    CLRF INDF
    INCF FSR
    MOVWF RAMSTOP
    SUBWF FSR,W
    BTFSS _Z
    GOTO MEMLOOP
    MOVLW 0x45 ; seed for random numbers
    MOVWF RANDLO
    MOVWF 0x30
    MOVWF RANDHI
    MOVWF DATA
    MOVWF FSR
    BCF _RF0
    BCF _RF1
    BCF _IRP
CALL RANDOM16
    MOVF RANDHI,W
    MOVF BARGBO
    BCF BARGBO, MSB
    MOVF BARGBO,W
    MOVF INDF
    INCF FSR
AN617

MOVF RANDLO,W
MOVF BARGB1
MOVF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVF BARGB2
MOVF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
BCF AARGB0,MSB
MOVF AARGB0,W
; MOVF AARGB0,W
MOVF INDF
INCF FSR
MOVF RANDLO,W
MOVF AARGB1
MOVF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVF AARGB2
MOVF INDF
INCF FSR
MOVF RANDLO,W
MOVF AARGB3
MOVF INDF
INCF FSR
CALL FXD3224S
MOVF AARGBO,W
MOVF INDF
INCF FSR
MOVF AARGBl,W
MOVF INDF
INCF FSR
MOVF AARGB2,W
MOVF INDF
INCF FSR
MOVF AARGB3,W
MOVF INDF
INCF FSR
MOVF REMBO,W
MOVF INDF
INCF FSR
MOVF REMBl,W
MOVF INDF
INCF FSR
MOVF REMB2,W
MOVWF INDF
INCF FSR
SELF GOTO SELF
RANDOM16
RLF RANDHI,W
; random number generator
XORWF RANDHI,W
MOVWF TEMPB0
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPB1
RLF TEMPB1,W
RLF TEMPB1
MOVF TEMPB1,W
XORWF RANDHI,W
SWAPF RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETLW 0

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

; 32/24 Bit Division Macros

SDIV3224L macro
; Max Timing: 13+6*22+21+6*22+21+6*22+21+6*22+21+6*22+21+6 = 700 clks
; Min Timing: 13+6*21+20+6*21+20+6*21+20+6*21+20+6*21+20+3 = 660 clks
; PM: 11+3*58+43 = 228
MOVF BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
RLF ACCB0
MOVFW 7
MOVWF LOOPS3224A
RLF ACCB0,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB0,LSB
GOTO SADD24LA
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
GOTO SOK24LA
ADDF W REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDF W REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDF W REMBO
SOK24LA
RLF ACCB0
DECFSZ LOOPCOUNT
GOTO LOOPS3224A
RLF ACCB1,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB0,LSB
GOTO SADD24L8
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W

© 1995 Microchip Technology Inc.
AN617

SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SOK24LB
SADD24LB
ADDFW REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDFW REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDFW REMB0
SOK24LB
RLF ACCB1
MOVLW 7
MOVWF LOOPCOUNT
LOOPS3224B
RLF ACCB1,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB1,LSB
GOTO SADD24LB
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SOK24LB
SADD24LB
ADDFW REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDFW REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDFW REMB0
SOK24LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPS3224B
RLF ACCB2,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB1,LSB
GOTO SADD24LB
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SOK24LB

DS00617A-page 2-276 © 1995 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Label</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SADD24L16</td>
<td>ADDWF REMB2&lt;br&gt;MOVF BARGB1,W&lt;br&gt;BTFSC _C&lt;br&gt;INCFSZ BARGB1,W&lt;br&gt;ADDWF REMB1&lt;br&gt;MOVF BARGB0,W&lt;br&gt;BTFSC _C&lt;br&gt;INCFSZ BARGB0,W&lt;br&gt;ADDWF REMB0</td>
</tr>
<tr>
<td>SOK24L16</td>
<td>RLF ACCB2&lt;br&gt;MOVLW 7&lt;br&gt;MOVWF LOOPCOUNT&lt;br&gt;RLF ACCB2,W&lt;br&gt;RLF REMB2&lt;br&gt;RLF REMB0&lt;br&gt;MOVF BARGB2,W&lt;br&gt;BTFSS ACCB2,LSB&lt;br&gt;GOTO SADD24LC&lt;br&gt;SUBWF REMB2&lt;br&gt;MOVF BARGB1,W&lt;br&gt;BTFSS _C&lt;br&gt;INCFSZ BARGB1,W&lt;br&gt;SUBWF REMB1&lt;br&gt;MOVF BARGB0,W&lt;br&gt;BTFSS _C&lt;br&gt;INCFSZ BARGB0,W&lt;br&gt;SUBWF REMB0&lt;br&gt;GOTO SOK24LC</td>
</tr>
<tr>
<td>LOOPSS224C</td>
<td>RLF ACCB3,W&lt;br&gt;RLF REMB2&lt;br&gt;RLF REMB1&lt;br&gt;RLF REMB0&lt;br&gt;MOVF BARGB2,W&lt;br&gt;BTFSS ACCB2,LSB&lt;br&gt;GOTO SADD24LC&lt;br&gt;SUBWF REMB2&lt;br&gt;MOVF BARGB1,W&lt;br&gt;BTFSS _C&lt;br&gt;INCFSZ BARGB1,W&lt;br&gt;SUBWF REMB1&lt;br&gt;MOVF BARGB0,W&lt;br&gt;BTFSS _C&lt;br&gt;INCFSZ BARGB0,W&lt;br&gt;SUBWF REMB0&lt;br&gt;GOTO SADD24LC</td>
</tr>
<tr>
<td>SADD24LC</td>
<td>ADDWF REMB2&lt;br&gt;MOVF BARGB1,W&lt;br&gt;BTFSC _C&lt;br&gt;INCFSZ BARGB1,W&lt;br&gt;ADDWF REMB1&lt;br&gt;MOVF BARGB0,W&lt;br&gt;BTFSC _C&lt;br&gt;INCFSZ BARGB0,W&lt;br&gt;SUBWF REMB0&lt;br&gt;GOTO SADD24LC</td>
</tr>
<tr>
<td>SOK24LC</td>
<td>RLF ACCB2&lt;br&gt;DECFSZ LOOPCOUNT&lt;br&gt;GOTO LOOPSS224C&lt;br&gt;RLF ACCB3,W&lt;br&gt;RLF REMB2&lt;br&gt;RLF REMB1&lt;br&gt;RLF REMB0&lt;br&gt;MOVF BARGB2,W&lt;br&gt;BTFSS ACCB2,LSB&lt;br&gt;GOTO SADD24L24&lt;br&gt;SUBWF REMB2&lt;br&gt;MOVF BARGB1,W&lt;br&gt;BTFSS _C&lt;br&gt;INCFSZ BARGB1,W&lt;br&gt;SUBWF REMB1&lt;br&gt;MOVF BARGB0,W&lt;br&gt;BTFSS _C&lt;br&gt;INCFSZ BARGB0,W&lt;br&gt;SUBWF REMB0&lt;br&gt;GOTO SOK24L24</td>
</tr>
<tr>
<td>SADD24L24</td>
<td>ADDWF REMB2&lt;br&gt;MOVF BARGB1,W&lt;br&gt;BTFSC _C&lt;br&gt;INCFSZ BARGB1,W&lt;br&gt;ADDWF REMB1&lt;br&gt;MOVF BARGB0,W</td>
</tr>
</tbody>
</table>

© 1995 Microchip Technology Inc.
BTFSC _C
INCFSZ BARGB0,W
ADDFW REMBO

SOK24L24
RLF ACCB3
MOVLW 7
MOVF LOOPCOUNT

LOOPS3224D
RLF ACCB3,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCR3,LSB
GOTO SADD24LD
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO SOK24LD

SADD24LD
ADDFW REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDFW REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDFW REMB0

SOK24LD
RLF ACCB3
DECFSZ LOOPCOUNT
GOTO LOOPS3224D
GOTO SOK24L
MOVF BARGB2,W
ADDFW REMB2
MOVF BARGB1,W
BTFSC _C
INCF BARGB1,W
ADDFW REMB1
MOVF BARGB0,W
BTFSC _C
INCF BARGB0,W
ADDFW REMB0

SOK24L
endm
UDIV3224L macro
; Max Timing: 20+6*27+26+26+6*27+26+6*27+26+6*27+26+12 = 862 clks
; Min Timing: 20+6*26+25+25+6*26+25+6*26+25+6*26+25+3 = 822 clks
; PM: 18+3*75+40+12 = 295 DM: 11
CLRFL TEMP
RLF ACCB0,W
RLF REMB2
MOVF BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C

DS00617A-page 2-278 © 1995 Microchip Technology Inc.
INCFSZ BARGBO,W
SUBWF REMB0
CLR W
BTFSS _C
MOVLW 1
SUBWF TEMP
RLF ACCBO
MOVLW 7
MOVF LOOPCOUNT
LOOPU3224A
RLF ACCBO,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFSS ACCBO,LSB
GOTO UADD24LA
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLR W
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK24LA
UADD24LA ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLR W
BTFSC _C
MOVLW 1
ADDWF TEMP
UOK24LA
RLF ACCBO
DECFSZ LOOPCOUNT
GOTO LOOPU3224A
RLF ACCB1,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFSS ACCBO,LSB
GOTO UADD24L8
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLR W
BTFSS  _C
MOVLW   1
SUBWF   TEMP
GOTO    UOK24L8
ADDWF   REMB2
MOVF    BARGB1,W
BTFSC   _C
INCFSZ  BARGB1,W
ADDWF   REMB1
MOVF    BARGBO,W
BTFSC   _C
INCFSZ  BARGBO,W
ADDWF   REMB0
CLRW
BTFSC   _C
MOVLW   1
ADDWF   TEMP
UADD24L8
RLF     ACCB1
MOVLW   7
MOVWF   LOOPCOUNT
LOOPU3224B RLF     ACCB1,W
RLF     REMB2
RLF     REMB1
RLF     REMB0
RLF     TEMP
MOVF    BARGB2,W
BTFSS   ACCB1,LSB
GOTO    UADD24LB
SUBWF   REMB2
MOVF    BARGB1,W
BTFSS   _C
INCFSZ  BARGB1,W
SUBWF   REMB1
MOVF    BARGBO,W
BTFSS   _C
INCFSZ  BARGBO,W
SUBWF   REMB0
CLRW
BTFSS   _C
MOVLW   1
SUBWF   TEMP
UOK24L8
ADDWF   REMB2
MOVF    BARGB1,W
BTFSC   _C
INCFSZ  BARGB1,W
ADDWF   REMB1
MOVF    BARGBO,W
BTFSC   _C
INCFSZ  BARGBO,W
ADDWF   REMB0
CLRW
BTFSC   _C
MOVLW   1
ADDWF   TEMP
UADD24LB
RLF     ACCB1
DECFSZ  LOOPCOUNT
GOTO    LOOPU3224B
RLF     ACCB2,W
RLF     REMB2
RLF     REMB1
RLF     REMB0
RLF     TEMP
MOVF    BARGB2,W
UOK24LB
BTFSS ACQB1,LSB
GOTO UADD24L16
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK24L16
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UADD24L16
RLF ACQB2
MOVLW 7
MOVF LOOPCOUNT

LOOPU3224C
RLF ACQB2,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFSS ACQB2,LSB
GOTO UADD24LC
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK24LC
ADDWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSS _C

UADD24LC
AN617

MOVLW 1
ADDF W TEMP

UOK24LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPU3224C
RLF ACCB3,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFS S ACCB2,LSB
GOTO UADD24L24
SUBWF REMB2
MOVF BARGB1,W
BTFS S _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFS S _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFS S _C
MOVLW 1
SUBWF TEMP
GOTO UOK24L24

UADD24L24
ADDF W REMB2
MOVF BARGB1,W
BTFS C _C
INCFSZ BARGB1,W
ADDF W REMB1
MOVF BARGB0,W
BTFS C _C
INCFSZ BARGB0,W
ADDF W REMB0
CLRW
BTFS C _C
MOVLW 1
ADDF W TEMP

UOK24L24
RLF ACCB3
MOVLW 7
MOVNF LOOPCOUNT

LOOPU3224D
RLF ACCB3,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFS S ACCB3,LSB
GOTO UADD24LD
SUBWF REMB2
MOVF BARGB1,W
BTFS S _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFS S _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFS S _C
MOVLW 1
SUBWF TEMP
UADD24LD
GOTO UOK24LD
ADDF BARGB1,W
MOVF BARGB0,W
BTFSC _C
INCFZ BARGB1,W
ADDF BARGB1,W
MOVF BARGB0,W
BTFSC _C
INCFZ BARGB0,W
ADDF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDF TEMP

UOK24LD
RLF ACCB3
DECFZ LOOPCOUNT
GOTO LOOPU3224D
BTFSC ACCB3,LSB
GOTO UOK24L
MOVF BARGB2,W
ADDF REMB2
MOVF BARGB1,W
BTFSC _C
INCF BARGB1,W
ADDF REMB1
MOVF BARGB0,W
BTFSC _C
INCF BARGB0,W
ADDF REMBO

UOK24L
endm
UDIV3123L macro
; Max Timing: 13*6+22+21+21+6*22+21+21+6*22+21+21+6*22+21+12 = 700 clks
; Min Timing: 13*6+21+20+20+6*21+20+20+6*21+20+20+6*21+20+3 = 660 clks
; PW: 11+3*58+43 = 228  DM: 10
MOVF BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFZ BARGB0,W
SUBWF REMBO
RLF ACCB0
MOVLW 7
MOVWF LOOPCOUNT
RLF ACCB0,W
RLF REMB2
RLF REMB1
RLF REMBO
MOVF BARGB2,W
BTFSS ACCB0,LSB
GOTO UADD13LA
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFZ BARGB0,W
SUBWF REMBO
GOTO UOK13LA
<table>
<thead>
<tr>
<th>AN617</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UADD13LA</strong></td>
</tr>
<tr>
<td>ADDWF REMB2</td>
</tr>
<tr>
<td>MOVF BARGBl,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBl,W</td>
</tr>
<tr>
<td>ADDWF REMB1</td>
</tr>
<tr>
<td>MOVF BARGBo,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBo,W</td>
</tr>
<tr>
<td>ADDWF REMB0</td>
</tr>
<tr>
<td><strong>UOK13LA</strong></td>
</tr>
<tr>
<td>RLF ACCB0</td>
</tr>
<tr>
<td>DECFSZ LOOPCOUNT</td>
</tr>
<tr>
<td>GOTO LOOFP3123A</td>
</tr>
<tr>
<td>RLF ACCB1,W</td>
</tr>
<tr>
<td>RLF REMB2</td>
</tr>
<tr>
<td>RLF REMB1</td>
</tr>
<tr>
<td>RLF REMB0</td>
</tr>
<tr>
<td>MOVF BARGB2,W</td>
</tr>
<tr>
<td>BTFSSS ACCB0,LSB</td>
</tr>
<tr>
<td>GOTO UADD13L8</td>
</tr>
<tr>
<td>SUBWF REMB2</td>
</tr>
<tr>
<td>MOVF BARGBl,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBl,W</td>
</tr>
<tr>
<td>SUBWF REMB1</td>
</tr>
<tr>
<td>MOVF BARGBo,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBo,W</td>
</tr>
<tr>
<td>SUBWF REMB0</td>
</tr>
<tr>
<td>GOTO UOK13L8</td>
</tr>
<tr>
<td><strong>UADD13L8</strong></td>
</tr>
<tr>
<td>ADDWF REMB2</td>
</tr>
<tr>
<td>MOVF BARGBl,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBl,W</td>
</tr>
<tr>
<td>ADDWF REMB1</td>
</tr>
<tr>
<td>MOVF BARGBo,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBo,W</td>
</tr>
<tr>
<td>ADDWF REMB0</td>
</tr>
<tr>
<td><strong>UOK13L8</strong></td>
</tr>
<tr>
<td>RLF ACCB1</td>
</tr>
<tr>
<td>MOVLF 7</td>
</tr>
<tr>
<td>MOVWF LOOPCOUNT</td>
</tr>
<tr>
<td><strong>LOOFP3123B</strong></td>
</tr>
<tr>
<td>RLF ACCB1,W</td>
</tr>
<tr>
<td>RLF REMB2</td>
</tr>
<tr>
<td>RLF REMB1</td>
</tr>
<tr>
<td>RLF REMB0</td>
</tr>
<tr>
<td>MOVF BARGB2,W</td>
</tr>
<tr>
<td>BTFSSS ACCB1,LSB</td>
</tr>
<tr>
<td>GOTO UADD13LB</td>
</tr>
<tr>
<td>SUBWF REMB2</td>
</tr>
<tr>
<td>MOVF BARGBl,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBl,W</td>
</tr>
<tr>
<td>SUBWF REMB1</td>
</tr>
<tr>
<td>MOVF BARGBo,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBo,W</td>
</tr>
<tr>
<td>SUBWF REMB0</td>
</tr>
<tr>
<td>GOTO UOK13LB</td>
</tr>
<tr>
<td><strong>UADD13LB</strong></td>
</tr>
<tr>
<td>ADDWF REMB2</td>
</tr>
<tr>
<td>MOVF BARGBl,W</td>
</tr>
<tr>
<td>BTFSCL C</td>
</tr>
<tr>
<td>INCFSZ BARGBl,W</td>
</tr>
<tr>
<td>ADDWF REMB1</td>
</tr>
<tr>
<td>MOVF BARGBo,W</td>
</tr>
</tbody>
</table>
BTFSC   _C
INCFSZ   BARGBO,W
ADDWF   REMBO

UOK13LB   RLF   ACCB1
DECFSZ   LOOPCOUNT
GOTO   LOOPU3123B
RLF   ACCB2,W
RLF   REMB2
RLF   REMB1
RLF   REMB0
MOVF   BARGB2,W
BTFSS   ACCB1,LSB
GOTO   UADD13L16
SUBWF   REMB2
MOVF   BARGB1,W
BTFSS   _C
INCFSZ   BARGB1,W
SUBWF   REMB1
MOVF   BARGB0,W
BTFSS   _C
INCFSZ   BARGB0,W
SUBWF   REMB0
GOTO   UOK13L16

UADD13L16   ADDWF   REMB2
MOVF   BARGB1,W
BTFSC   _C
INCFSZ   BARGB1,W
ADDWF   REMB1
MOVF   BARGB0,W
BTFSC   _C
INCFSZ   BARGB0,W
ADDWF   REMB0

UOK13L16   RLF   ACCB2
MOVLW   7
MOVF   LOOPCOUNT
LOOPU3123C   RLF   ACCB2,W
RLF   REMB2
RLF   REMB1
RLF   REMB0
MOVF   BARGB2,W
BTFSS   ACCB2,LSB
GOTO   UADD13L1C
SUBWF   REMB2
MOVF   BARGB1,W
BTFSS   _C
INCFSZ   BARGB1,W
SUBWF   REMB1
MOVF   BARGB0,W
BTFSS   _C
INCFSZ   BARGB0,W
SUBWF   REMB0
GOTO   UOK13LC

UADD13LC   ADDWF   REMB2
MOVF   BARGB1,W
BTFSC   _C
INCFSZ   BARGB1,W
ADDWF   REMB1
MOVF   BARGB0,W
BTFSC   _C
INCFSZ   BARGB0,W
ADDWF   REMB0

UOK13LC   RLF   ACCB2
DECFSZ   LOOPCOUNT
GOTO LOOPU3123C
RLF ACCB3,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB2,LSB
GOTO UADD13L24
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
ADDFP REMB0

UADD13L24
ADDFP REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDFP REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDFP REMB0

UOK13L24
RLF ACCB3
MOVFW 7
MOVWF LOOPCOUNT

LOOPU3123D
RLF ACCB3,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB3,LSB
GOTO UADD13LD
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK13LD

UADD13LD
ADDFP REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDFP REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDFP REMB0

UOK13LD
RLF ACCB3
DECFSZ LOOPCOUNT
GOTO LOOPU3123D
BTFSC ACCB3,LSB
GOTO UOK13L
MOVF BARGB2,W
ADDFP REMB2
MOVF BARGB1,W
### 32/24 Bit Signed Fixed Point Divide 32/24 -> 32.24

**Input:** 32 bit fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3

**Use:** CALL FXD3224S

**Output:** 32 bit fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3

**24 bit fixed point remainder in REMBO, REMB1, REMB2**

**Result:** AARG, REM <-- AARG / BARG

**Max Timing:**
- 12+700+3 = 715 clks
- 19+700+23 = 742 clks
- 19+700+23 = 742 clks
- 29+700+3 = 732 clks

**Min Timing:**
- 12+660+3 = 675 clks
- 19+660+23 = 702 clks
- 19+660+23 = 702 clks
- 29+660+3 = 692 clks

**PM:** 29+228+23 = 280

---

```
FXD3224S
MOVF AARGBO,W
XORWF BARGBO,W
MOVF SIGN
BTFS BARGBO, MSB;
GOTO CA3224S;
if MSB set, negate BARG
COMF BARGB2
INCF BARGB2
BTFS _Z
DECF BARGB1
COMF BARGB1
BTFS _Z
DECF BARGB0
COMF BARGB0
CA3224S
BTFS AARGBO, MSB;
GOTO C3224S;
if MSB set, negate AARG
COMF AARGB3
INCF AARGB3
BTFS _Z
DECF AARGB2
COMF AARGB2
BTFS _Z
DECF AARGB1
COMF AARGB1
BTFS _Z
DECF AARGB0
COMF AARGB0
C3224S
CLRF REMBO
CLRF REMB1
CLRF REMB2
SDIV3224L
BTFS SIGN, MSB
RETLW 0x00
COMF AARGB3
INCF AARGB3
BTFS _Z
DECF AARGB2
COMF AARGB2
BTFS _Z
```
AN617

DECF AARGBl
COMF AARGBl
BTFSC _Z
DECF AARGBO
COMF AARGBO
COMF REMB2
INCF REMB2
BTFSC _Z
DECF REMBl
COMF REMBl
BTFSC z -
DECF REMBO
COMF REMBO
RETLW OxOO

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

32/24 Bit Unsigned Fixed Point Divide 32/24 -> 32.24
Input: 32 bit unsigned fixed point dividend in AARGBO, AARGBl,AARGB2,AARGB3
24 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2
Use: CALL FXD3224U
Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
24 bit unsigned fixed point remainder in REMBO, REMBl, REMB2
Result: AARG, REM <-- AARG
Max Timing: 3+862+2 = 867 clks
Min Timing: 3+822+2 = 827 clks
PM: 3+295+1 = 299
DM: 11
FXD3224U CLRF REMBO
CLRF REMB1
CLRF REMB2
UDIV3224L
RETLW 0x00

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

31/23 Bit Unsigned Fixed Point Divide 31/23 -> 31.23
Input: 31 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3
23 bit unsigned fixed point divisor in BARGBO, BARGBl, BARBB2
Use: CALL FXD33123U
Output: 31 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
23 bit unsigned fixed point remainder in REMBO, REMBl, REMB2
Result: AARG, REM <-- AARG
Max Timing: 3+700+2 = 705 clks
Min Timing: 3+660+2 = 665 clks
PM: 3+228+1 = 232
DM: 10
FXD3123U CLRF REMBO
CLRF REMB1
CLRF REMB2
UDIV3123L
RETLW 0x00

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

END

D.3 32/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines

; 32/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed divide application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine Clocks Function
FXD3216S 578 32 bit/16 bit -> 32.16 signed fixed point divide
FXD3216U 703 32 bit/16 bit -> 32.16 unsigned fixed point divide

DS00617A-page 2-288 © 1995 Microchip Technology Inc.
; FXD3115U  541 31 bit/15 bit -> 31.15 unsigned fixed point divide
list r=dec.x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions
;******************************************************************************
;******************************************************************************
; Test suite storage
RANDHI equ Ox1E ; random number segerator registers
RANDLO equ Ox1F
TDATA equ Ox20
;******************************************************************************
;******************************************************************************
; Test suite for 32/16 bit fixed point divide algorithms
org Ox0005
MAIN
  MOVLW RAMSTART
  MOVWF FSR
MEMLOOP
  CLRF INDF
  INCF FSR
  MOVLW RAMSTOP
  SUBWF FSR,W
  BTFSS - z
  GOTO MEMLOOP
  MOVLW Ox45
  seed for random numbers
  MOVWF RAND LO
  MOVLW Ox30
  MOVWF RAND HI
  MOVLW TDATA
  MOVWF FSR
  BCF _RPO
  BCF _RP0
  BCF _IRP
  CALL RANDOM16
  SWAPF RAND HI
  SWAPF RAND LO
  MOVF RANDHI,W
  MOVWF BARGBO
  BCF BARGBO,MSB
  MOVF BARGBO,W
  MOVWF INDF
  INCF FSR
  MOVF RANDLO,W
  MOVWF BARGBL
  MOVWF INDF
  INCF FSR
  CALL RANDOM16
; SWAPPF RANDHI
; SWAPPF RANDLO
; MOVF RANDHI,W
; MOVWF AARGBO
; BCF AARGBO,MSB
; MOVF AARGBO,W
; MOVWF INDF
; INCF FSR
; MOVF RANDLO,W
; MOVWF AARGBl
; MOVWF INDF
; INCF FSR
; CALL RANDOM16
; MOVF RANDHI,W
; MOVWF AARGB2
; MOVWF INDF
; INCF FSR
; MOVF RANDLO,W
; MOVWF AARGB3
AN617

```
MOVWF    INDF
INCF    FSR
CALL    FXD3216S
MOVF    AARGBO,W
MOVWF    INDF
INCF    FSR
MOVF    AARGB1,W
MOVWF    INDF
INCF    FSR
MOVF    AARGB2,W
MOVWF    INDF
INCF    FSR
MOVF    AARGB3,W
MOVWF    INDF
INCF    FSR
MOVF    REMBO,W
MOVWF    INDF
INCF    FSR
MOVF    REMB1,W
MOVWF    INDF
INCF    FSR
SEF
GOTO    SELF
SELF
GOTO    SELF
RANOM16
RLF    RANDHI,W
XORWF    RANDHI,W
MOVWF    TEMPBO
SWAPF    RANDHI
SWAPF    RANDLO,W
MOVWF    TEMPBl
RLF    TEMPBl
RLF    TEMPBl
MOVF    TEMPBl,W
XORWF    RANDHI,W
SWAPF    RANDHI
ANDLW    OxOl
RLF    TempBo
RLF    RANDLO
XORWF    RANDLO
RETLW    0
```

`random number generator`

```
;******************************************************************************
;******************************************************************************
; 32/16 Bit Division Macros
SDIV3216L  macro
; Max Timing: 9*64+16+6*17+16+16+6*17+16+6*17+16=537 clks
; Min Timing: 9*64+15+15+6*15+15+6*15+15+6*15+15+3 = 501 clks
; PM: 157    DM: 9
MOV     BARGB1,W
SUBWF   REMB1
MOV     BARGBO,W
BTFSS   _C
INCFSZ  BARGBO,W
SUBWF   REMB0
RLF     ACCB0
MOVLW   7
MOVWF   LOOPCOUNT
RLF     ACCB0,W
RLF     REMB1
RLF     REMB0
MOV     BARGB1,W
BTFSS   ACCB0,LSB
GOTO    SADD26LA
SUBWF   REMB0
MOV     BARGBO,W
BTFSS   _C
INCFSZ  BARGBO,W
```

© 1995 Microchip Technology Inc.
SUBWF REMBO
GOTO SOK26LA
ADDF W REMB1
MOVF BARGBO,W
BTFSC _C
INCF SZ BARGBO,W
ADDF W REMBO

SADD26LA
RLF ACCBO
DECFSZ LOOPCOUNT
GOTO LOOPS3216A
RLF ACCBO1,W
RLF REMBO
MOVF BARGBO1,W
BTFS S ACCBO,LSB
GOTO SADD26LB
SUBWF REMBO1
MOVF BARGBO1,W
BTFSC _C
INCF SZ BARGBO1,W
SUBWF REMBO
GOTO SOK26LA

SADD26LB
ADDF W REMBO1
MOVF BARGBO1,W
BTFSC _C
INCF SZ BARGBO1,W
ADDF W REMBO1

SOK26LB
RLF ACCB1
MOVWF 7
MOVWF LOOPCOUNT

LOOP3216B
RLF ACCBO,W
RLF REMBO
RLF REMBO
MOVF BARGBO,W
BTFSS ACCB1,LSB
GOTO SADD26LB
SUBWF REMBO1
MOVF BARGBO1,W
BTFSS ACCBO1,LSB
GOTO SOK26LB

SADD26LB
ADDF W REMBO1
MOVF BARGBO1,W
BTFSC _C
INCF SZ BARGBO1,W
ADDF W REMBO1

SOK26LB
DECF SZ LOOPCOUNT
GOTO LOOP3216B
RLF ACCB2,W
RLF REMBO
RLF REMBO
MOVF BARGBO,W
BTFSS ACCB1,LSB
GOTO SADD26LB16
SUBWF REMBO1
MOVF BARGBO1,W
BTFSC _C
INCF SZ BARGBO1,W
SUBWF REMBO
GOTO SOK26LB16

SADD26LB16
ADDF W REMBO1
AN617

MOVF BARGBO,W
BTFSC _c
INCFSZ BARGBO,W
ADDWF REMBO

SOK26L16
RLF ACCB2
MOVlw 7
MOVWF LOOPCOUNT

LOOPS3216C
RLF ACCB2,W
RLF REMB1
MOVF BARGB1,W
BTFSS ACCB2,LSB
GOTO SADD26LC
SUBWF REMB1
MOVF BARGBO,W
BTFSS _c
INCFSZ BARGBO,W
ADDWF REMB0

SOK26LC
RLF ACCB2
DSCPSZ LOOPCOUNT
GOTO LOOPS3216C
RLF ACCB3,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
BTFSS ACCB2,LSB
GOTO SADD26L24
SUBWF REMB1
MOVF BARGBO,W
BTFSS _c
INCFSZ BARGBO,W
ADDWF REMB0

SADD26LC
ADDWF REMB1
MOVF BARGB0,W
BTFSC _c
INCFSZ BARGB0,W
ADDWF REMB0

SOK26L24
RLF ACCB3
MOVlw 7
MOVWF LOOPCOUNT

LOOPS3216D
RLF ACCB3,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
BTFSS ACCB3,LSB
GOTO SADD26LD
SUBWF REMB1
MOVF BARGB0,W
BTFSS _c
INCFSZ BARGB0,W
ADDWF REMB0

SADD26LD
ADDWF REMB1
MOVF BARGBO,W
BTFSC _c
INCFSZ BARGBO,W

DS00617A-page 2-292 © 1995 Microchip Technology Inc.
ADDWF REMBO

SOK26LD
  RLF ACCB3
  DECFSZ LOOPCOUNT
  GOTO LOOPS216D
  BTFSC ACCB3,LSB
  GOTO SOK26L
  MOVF BARGB1,W
  ADDWF REMB1
  MOVF BARGB0,W
  BTFSC _C
  INCF BARGB0,W
  ADDWF REMB0

SOK26L
  endm
UDIV3216L
  macro
    ; Max Timing: 16*6*22+21+21+6*22+21+6*22+21+6*22+21+6*22+21+6+22+21+6*22+21+6*22+21+6+22+21+R = 699 clks
    ; Min Timing: 16*6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+20+3 = 663 clks
    ; PM: 240
    CLRF TEMP
    RLF ACCB0,W
    RLF REMB1
    MOVF BARGB1,W
    SUBWF REMB1
    MOVF BARGB0,W
    BTFSS _C
    INCF BARGB0,W
    SUBWF REMB0
    CLRwf
    BTFSS _C
    MOVLW 1
    SUBWF TEMP
    RLF ACCB0
    MOVLW 7
    MOVWF LOOPCOUNT

LOOPEN216A
  RLF ACCB0,W
  RLF REMB1
  RLF REMB0
  RLF TEMP
  MOVF BARGB1,W
  BTFSS ACCB0,LSB
  GOTO UADD26LA
  SUBWF REMB1
  MOVF BARGB0,W
  BTFSS _C
  INCF BARGB0,W
  SUBWF REMB0
  CLRw
  BTFSS _C
  MOVLW 1
  SUBWF TEMP
  GOTO UOK26LA

UADD26LA
  ADDWF REMB1
  MOVF BARGB0,W
  BTFSC _C
  INCF BARGB0,W
  ADDWF REMB0
  CLRw
  BTFSC _C
  MOVLW 1
  ADDWF TEMP

UOK26LA
  RLF ACCB0
  DECFSZ LOOPCOUNT
  GOTO LOOPEN216A
  RLF ACCB1,W
RLF REMB1
RLF REMB0
RLF TEMP
MOVFBARGB1,W
BTFSSACCBO,LSB
GOTOUADD26L8
SUBWFREMB1
MOVFBARGB0,W
BTFSS_C
INCFSZBARGB0,W
SUBWFREMB0
CLRWMOVFBARGB1,W
BTFSS_C
MOVLW1
SUBWFTEMP
GOTOUOK26L8

UADD26L8
ADDWFREMB1
MOVFBARGB0,W
BTFSS_C
INCFSZBARGB0,W
ADDWFREMB0
CLRWMOVFBARGB1,W
BTFSS_C
MOVLW1
ADDWFTEMP

UOK26L8
RLFACCB1
MOVLW7
MOVWFLOPCOUNT

LOOPU3216B
RLFACCB1,W
RLFREMB1
RLFREMB0
RLFTEMP
MOVFBARGB1,W
BTFSSACCBO,LSB
GOTOUADD26LB
SUBWFREMB1
MOVFBARGB0,W
BTFSS_C
INCFSZBARGB0,W
SUBWFREMB0
CLRWMOVFBARGB1,W
BTFSS_C
MOVLW1
SUBWFTEMP
GOTOUOK26LB

UADD26LB
ADDWFREMB1
MOVFBARGB0,W
BTFSS_C
INCFSZBARGB0,W
ADDWFREMB0
CLRWMOVFBARGB1,W
BTFSS_C
MOVLW1
ADDWFTEMP

UOK26LB
RLFACCB1
DECFSZLOOPCOUNT
GOTOLOOPU3216B
RLFACCB2,W
RLFREMB1
RLFREMB0
RLFTEMP
MOVFBARGB1,W
BTFSSACCBO,LSB
GOTOUADD26L16
AN617

SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK26L16

UOK26LC
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP
GOTO UOK26LC

LOOPU3216C
RLF ACCB2
MOVLW 7
MOVWF LOOPCOUNT
RLF ACCB2
RLF REMB1
RLF REMBO
RLF TEMP
MOVF BARGBO1,W
BTFSS ACCB2,LSB
GOTO UADD26LC
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK26LC

UOK26LC
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSC _C
MOVFLW 1
ADDWF TEMP
GOTO UOK26LC

UOK26LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPU3216C
RLF ACCB3,W
RLF REMB1
RLF REMBO
RLF TEMP
MOVF BARGBO1,W
BTFSS ACCB2,LSB
GOTO UADD26L24
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
CLRW
BTFSS _C
MOVWF L
SUBWF TEMP
GOTO UOK26L24

UADD26L24
ADDWF REMBI
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSC _C
MOVWF 1
ADDWF TEMP

UOK26L24
RLF ACCB3
MOVWF 7
MOVWF LOOPCOUNT

LOOPU3216D
RLF ACCB3,W
RLF REMBI
RLF REMBO
RLF TEMP
MOVF BARGBO,W
BTFSS ACCB3,LSB
GOTO UADD26LD
SUBWF REMBI
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
CLRW
BTFSS _C
MOVWF 1
SUBWF TEMP
GOTO UOK26LD

UADD26LD
ADDWF REMBI
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSC _C
MOVWF 1
ADDWF TEMP

UOK26LD
RLF ACCB3
DECFSZ LOOPCOUNT
GOTO LOOPU3216D
BTFSC ACCB3,LSB
GOTO UOK26L
MOVF BARGBO,W
ADDWF REMBI
MOVF BARGBO,W
BTFSC _C
INCF BARGBO,W
ADDWF REMBO

UOK26L
endm

UDIV3115L
macro
Max Timing: \(9 + 6 \times 17 + 16 + 16 + 16 + 16 + 16 + 16 + 16 + 16 = 537\) clks
Min Timing: \(9 + 6 \times 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 15 + 3 = 501\) clks
PM: 157

MOVF BARGBO,W
SUBWF REMBI
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
AN617

SUBWF REMBO
RLF ACCBO
MOVW 7
MOVF BARGBl, W
BTFSS ACCBO, LSB
GOTO UADD15LA
SUBWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
SUBWF REMBO
GOTO UOK15LA

LOOPU3115A
RLF ACCBO, W
RLF REMBl
MOVF BARGBl, W
BTFSS ACCBO, LSB
GOTO UADD15LA
SUBWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
ADDWF REMBO

LOOPU3115B
RLF ACCBl, W
RLF REMBl
MOVF BARGBl, W
BTFSS ACCBl, LSB
GOTO UADD15LB
SUBWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
ADDWF REMBO

UADD15LA
ADDWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
ADDWF REMBO

UADD15LB
ADDWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
ADDWF REMBO

UOK15LA
RLF ACCB0
DECFSZ LOOPCOUNT
GOTO LOOPU3115A
RLF ACCBl, W
RLF REMBl
MOVF BARGBl, W
BTFSS ACCBl, LSB
GOTO UADD15LB
SUBWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
SUBWF REMBO
GOTO UOK15LB

UOK15LB
RLF ACCB1
MOVW 7
MOVF BARGBl, W
BTFSS ACCBO, LSB
GOTO UADD15LB
SUBWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
SUBWF REMBO
GOTO UOK15LB

UADD15LB
ADDWF REMBl
MOVF BARGBO, W
BTFS _C
INCFSZ BARGBO, W
ADDWF REMBO

UOK15LB
RLF ACCB1
DECFSZ LOOPCOUNT
AN617

RLF REMBO
MOVF BARGBl,W
BTFSS ACCB3,LSB
GOTO UADD15LD
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMB0
GOTO UOK15LD
ADDWF REMB1
MOVF BARGBO,W
BTFSC ~
INCFSZ BARGBO,W
ADDWF REMBO
UADD15LD ADDWF REMBl
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
UOK15LD RLF ACCB3
DECFSZ LOOPCOUNT
GOTO UOK15L
MOVF BARGBl,W
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCF BARGBO,W
ADDWF REMBO
UOK15L endm

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

32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16

Input: 32 bit fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3
16 bit fixed point divisor in BARGBO, BARGBl

Use: CALL FXD3216S

Output: 32 bit fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
16 bit fixed point remainder in REMB0, REMB1

Result: AARG, REM <-- AARG / BARG

Max Timing: 11+537+3 = 551 clks  A > 0, B > 0
15+537+20 = 572 clks  A > 0, B < 0
21+537+20 = 578 clks  A < 0, B > 0
25+537+3 = 565 clks  A < 0, B < 0

Min Timing: 11+501+3 = 515 clks  A > 0, B > 0
15+501+20 = 536 clks  A > 0, B < 0
21+501+20 = 542 clks  A < 0, B > 0
25+501+3 = 529 clks  A < 0, B < 0

PM: 25+157+19 = 201 DM: 10

FXD3216S MOVF AARGBO,W
XORWF BARGBO,W
MOVF SIGN
BTFSS BARGBO,MSB ; if MSB set, negate BARG
GOTO CA3216S
COMF BARGBl
INCF BARGBl
BTFSC _Z
DECF BARGBO
COMF BARGBO
CA3216S BTFSS AARGBO,MSB ; if MSB set, negate AARG
GOTO C3216S
COMF AARGB3
INCF AARGB3
BTFSC _Z
DECF AARGB2
COMF AARGB2

© 1995 Microchip Technology Inc.
AN617

BTFSC _z
DECF AARGBl
COMF AARGBl
BTFSC _z
DECF AARGBO
COMF AARGBO
C3216S
CLRF REMB0
CLRF REMB1
SDIV3216L
BTFSS SIGN,MSB
RETLW Ox00
COMF AARGB3
INCF AARGB3
BTFSC _z
DECF AARGB2
COMF AARGB2
BTFSC _z
DECF AARGB1
COMF AARGB1
BTFSC _z
DECF BARGBO
COMF BARGBO
BTFSC _z
DECF BARGBl
COMF BARGBl
RETLW 0x00

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

END

DS00617A-page 2-300 © 1995 Microchip Technology Inc.
24/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines

; 24/24 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
; Input: fixed point arguments in AARG and BARG
; Output: quotient AARG/BARG followed by remainder in REM
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed divide application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in
; performance.

 Routine    Clocks    Function
 FXD2424S   565     24 bit/24 bit -> 24.24 signed fixed point divide
 FXD2424U   676     24 bit/24 bit -> 24.24 unsigned fixed point divide
 FXD2323U   531     23 bit/23 bit -> 23.23 unsigned fixed point divide

;********************************************************************************************
;********************************************************************************************
; Test suite storage
RANDHI       equ   0x1A ; random number seenerator registers
RANDLO       equ   0x1B
DATA         equ   0x20
;********************************************************************************************
;********************************************************************************************
; Test suite for 24/24 bit fixed point divide algorithms
org 0x0005
MAIN
MOVlw RAMSTART
MOVwf FSR
MEMLOOP
CLRlw INDF
INCl FSR
MOVlw RAMSTOP
SUBlw FSR,W
BTFSS _Z
GOTO MEMLOOP
MOVlw 0x45 ; seed for random numbers
MOVwf RANDLO
MOVlw 0x30
MOVwf RANDHI
MOVlw DATA
MOVwf FSR
BCF _RP0
BCF _RP1
BCF _IRP
CALL RANDOM16
MOVf RANDHI,W
MOVwf BARGB0
; BCF BARGBO,MSB
; MOVF BARGBO,W
MOVwf INDF
INCl FSR
MOVF RANDLO,W
MOVwf BARGB1
MOVwf INDF
INCl FSR
CALL RANDOM16
MOVF RANDHI,W
MOVwf BARGB2
MOVwf INDF
INCl FSR
CALL RANDOM16
MOVF RANDHI,W
MOVwf AARGB0
; BCF AARGBO,MSB
; MOVF AARGBO,W
MOVwf INDF

© 1995 Microchip Technology Inc.
INCF FSR
MOVF RANDLO,W
MOVF AARGB1
MOVF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVF AARGB2
MOVF INDF
INCF FSR
MOVF RANDHI,W
MOVF AARGB1,W
MOVF INDF
INCF FSR
MOVF RANDHI,W
MOVF AARGB2,W
MOVF INDF
INCF FSR
MOVF RANDHI,W
MOVF AARGB2,W
MOVF INDF
INCF FSR
MOVF AARGBO,W
MOVF INDF
INCF FSR
MOVF AARGBl,W
MOVF INDF
INCF FSR
MOVF AARGBl,W
MOVF INDF
INCF FSR
MOVF REMBO,W
MOVF INDF
INCF FSR
MOVF REMBl,W
MOVF INDF
INCF FSR
MOVF REMB2,W
MOVF INDF
INCF FSR
SELF GOTO SELF
RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMP BO
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPBl
RLF TEMPBl,W
MOVWF BARGBl,W
BTFSS _C
INCFSZ SUBWF REMBl,W
MOV WF BARGBO,W
BTFSS _C
INCFSZ SUBWF REMBO
XORWF RAND LO
RLF RAND LO
XORWF RAND HI
RETLW 0
;******************************************************************************************
;******************************************************************************************
24/24 Bit Division Macros
SDIV2424L macro
; Max Timing: 13+6*22+21+6*22+21+6*22+21+6*22+21+6*22+21+12 = 526 clks
; Min Timing: 13+6*21+20+6*21+20+6*21+20+6*21+20+6*21+20+3 = 494 clks
; FM: 11+3*51+31+12 = 207 DM: 12
MOV WF AARGB0,W
SUBWF REMB2
MOV WF AARGB1,W
BTFSS _C
INCF SZ BARGB1,W
SUBWF REMB1
MOV WF AARGBO,W
BTFSS _C
INCF SZ BARGBO,W
SUBWF REMBO

© 1995 Microchip Technology Inc.
AN617

RLF   ACCB0
MOV1W  7
MOVWF   LOOPCOUNT

LOOPS2424A
RLF   ACCB0,W
RLF   REMB2
RLF   REMB1
RLF   REMB0
MOVF   BARGB2,W
BTFSS   ACCB0,LSB
GOTO   SADD44LA
SUBWF   REMB2
MOVF   BARGB1,W
BTFSS   _C
INCFSZ   BARGB1,W
SUBWF   REMB1
MOVF   BARGB0,W
BTFSS   _C
INCFSZ   BARGB0,W
SUBWF   REMB0
GOTO   SOK44LA
ADDWF   REMB2
MOVF   BARGB1,W
BTFSC   _C
INCFSZ   BARGB1,W
ADDWF   REMB1
MOVF   BARGB0,W
BTFSC   _C
INCFSZ   BARGB0,W
ADDWF   REMB0

SADD44LA
RLF   ACCB0
DECLZ   LOOPCOUNT
GOTO   LOOPS2424A
RLF   ACCB1,W
RLF   REMB2
RLF   REMB1
RLF   REMB0
MOVF   BARGB2,W
BTFSS   ACCB0,LSB
GOTO   SADD44L8
SUBWF   REMB2
MOVF   BARGB1,W
BTFSS   _C
INCFSZ   BARGB1,W
SUBWF   REMB1
MOVF   BARGB0,W
BTFSS   _C
INCFSZ   BARGB0,W
SUBWF   REMB0
GOTO   SOK44L8
ADDWF   REMB2
MOVF   BARGB1,W
BTFSC   _C
INCFSZ   BARGB1,W
ADDWF   REMB1
MOVF   BARGB0,W
BTFSC   _C
INCFSZ   BARGB0,W
ADDWF   REMB0

SOK44L8
RLF   ACCB1
MOVILW   7
MOVILWF   LOOPCOUNT

LOOPS2424B
RLF   ACCB1,W
RLF   REMB2
RLF   REMB1
RLF  REMB0
MOVF  BARGB2,W
BTFSS  ACCB1,LSB
GOTO  SADD44LB
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCFSZ  BARGB1,W
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
SUBWF  REMB0
GOTO  SOK44LB
SADD44LB  ADDWF  REMB2
MOVF  BARGB1,W
BTFSC  _C
INCFSZ  BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSC  _C
INCFSZ  BARGB0,W
ADDWF  REMB0
SOK44LB  RLF  ACCB1
DECFSZ  LOOPCOUNT
GOTO  LOOPS2424B
RLF  ACCB2,W
RLF  REMB2
RLF  REMB1
RLF  REMB0
MOVF  BARGB2,W
BTFSS  ACCB1,LSB
GOTO  SADD44L16
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCFSZ  BARGB1,W
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
SUBWF  REMB0
GOTO  SOK44L16
SADD44L16  ADDWF  REMB2
MOVF  BARGB1,W
BTFSC  _C
INCFSZ  BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSC  _C
INCFSZ  BARGB0,W
ADDWF  REMB0
SOK44L16  RLF  ACCB2
MOVLW  7
MOVWF  LOOPCOUNT
LOOPS2424C  RLF  ACCB2,W
RLF  REMB2
RLF  REMB1
RLF  REMB0
MOVF  BARGB2,W
BTFSS  ACCB2,LSB
GOTO  SADD44LC
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS _c
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _c
INCFSZ BARGB0,W
SUBWF REMBO
GOTO SOK44LC
ADDFP REMB2
MOVF BARGB1,W
BTFSC _c
INCFSZ BARGB1,W
ADDFP REMB1
MOVF BARGB0,W
BTFSC _c
INCFSZ BARGB0,W
ADDFP REMBO

SADD44LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPU2424A
BTFSC ACCB2,LSB
GOTO SOK44L
MOVF BARGB2,W
ADDFP REMB2
MOVF BARGB1,W
BTFSC _c
INCF BARGB1,W
ADDFP REMB1
MOVF BARGB0,W
BTFSC _c
INCF BARGB0,W
ADDFP REMBO

SOK44LC

ends
UDIV2424L macro
; Max Timing: 20+6*28+27+6*28+27+6*28+27+12 = 671 elks
; Min Timing: 20+6*27+26+26+6*27+26+26+3 639 elks
; PM: 18+2*76+40+12 = 222 DM: 13
CLR W TEMP
RLF ACCB0,W
RLF REMB2
MOVF BARGB2,W
SUBWF REMB2
MOVF BARGB1,W
BTFSS _c
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _c
INCFSZ BARGB0,W
SUBWF REMBO
CLR W
BTFSS _c
MOVLW 1
SUBWF TEMP
RLF ACCB0
MOVLW 7
MOVF BARGB2,W
BTFSS ACCB0,LSB

© 1995 Microchip Technology Inc.
AN617

GOTO UADD44LA
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK44LA

UADD44LA
ADDF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDF TEMP

UOK44LA
RLF ACCBO
DECFSZ LOOPCOUNT
GOTO LOCP2424A
RLF ACCB1,W
RLF REMB2
RLF REMB1
RLF TEMP
MOVF BARGB2,W
BTFSS ACCB0,LSB
GOTO UADD44L8
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK44L8

UADD44L8
ADDF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK44L8
RLF ACCB1
MOVLW 7
MOVF LOOPCOUNT

LOOPU2424B
RLF ACCB1,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFSS ACCB1,LSB
GOTO UADD44LB
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK44LB

UADD44LB
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP
GOTO UOK44LB

UOK44LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPU2424B
RLF ACCB2,W
RLF REMB2
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB2,W
BTFSS ACCB1,LSB
GOTO UADD44L16
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK44L16
UADD44L16
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK44L16
RLF ACCB2
MOVLW 7
MOVWF LOOPCOUNT
LOOPU2424C
RLF ACCB2,W
RLF REMB2
RLF REMB1
RLF TEMP
MOVF BARGB2,W
BTFSS ACCB2,LSB
GOTO UADD44LC
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK44LC
UADD44LC
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK44LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPU2424C
BTFSC ACCB2,LSB
GOTO UOK44LC
MOVF BARGB2,W
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INC BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
UDIV2323L macro
; Max Timing:  13+6*22+21+6*22+21+6*22+21+6*22+21+12 = 526 clks
; Min Timing:  13+6*21+20+6*21+20+6*21+20+6*21+20+3 = 494 clks
; PM: 11+3*51+31+12 = 207 DM: 12

MOVF BARGB2, W
SUBWF REMB2
MOVF BARGB1, W
BTFSS _C
INCFSZ BARGB1, W
SUBWF REMB1
MOVF BARGB0, W
BTFSS _C
INCFSZ BARGB0, W
SUBWF REMB0
RLF ACCB0
MOVLW 7
MOVF LOOPCOUNT

LOOPU2323A
RLF ACCB0, W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2, W
BTFSS ACCB0, LSB
GOTO UADD33LA
SUBWF REMB2
MOVF BARGB1, W
BTFSS _C
INCFSZ BARGB1, W
SUBWF REMB1
MOVF BARGB0, W
BTFSS _C
INCFSZ BARGB0, W
SUBWF REMB0
GOTO UOK33LA

UADD33LA
ADDF W REMB2
MOVF BARGB1, W
BTFSC _C
INCFSZ BARGB1, W
ADDF W REMB1
MOVF BARGB0, W
BTFSC _C
INCFSZ BARGB0, W
ADDF W REMB0

UOK33LA
RLF ACCB0
DECFSZ LOOPCOUNT
GOTO LOOPU2323A
RLF ACCB1, W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2, W
BTFSS ACCB0, LSB
GOTO UADD33LB
SUBWF REMB2
MOVF BARGB1, W
BTFSS _C
INCFSZ BARGB1, W
SUBWF REMB1
MOVF BARGB0, W
BTFSS _C
INCFSZ BARGB0, W
AN617

SUBWF REMB0
GOTO UOK33L8
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

UOK33L8
RLF ACCB1
MOVLW 7
MOVWF LOOPCOUNT
LOOPU2323B
RLF ACCB1,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB1,LSB
GOTO UADD33LB
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK33LB
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

UOK33LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPU2323B
RLF ACCB2,W
RLF REMB2
RLF REMB1
RLF REMB0
MOVF BARGB2,W
BTFSS ACCB1,LSB
GOTO UADD33L16
SUBWF REMB2
MOVF BARGB1,W
BTFSS _C
INCFSZ BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK33L16
ADDWF REMB2
MOVF BARGB1,W
BTFSC _C
INCFSZ BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSC  _C
INCF  BARGB0,W
ADDWF  REMB0

UOK33L16
RLF  ACCB2
MOVLW  7
MOVF  LOOPCOUNT

LOOPU2323C
RLF  ACCB2,W
RLF  REMB2
RLF  REMB1
RLF  REMNEG
MOVF  BARGB0,W
BTFSS  ACCB2,LSB
GOTO  UADD33LC
SUBWF  REMB2
MOVF  BARGB1,W
BTFSS  _C
INCF  BARGB1,W
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCF  BARGB0,W
SUBWF  REMB0
GOTO  UOK33LC

UADD33LC
ADDWF  REMB2
MOVF  BARGB1,W
BTFSC  _C
INCF  BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSC  _C
INCF  BARGB0,W
ADCWF  REMB0

UOK33LC
RLF  ACCB2
DECFSZ  LOOPCOUNT
GOTO  LOOPU2323C
GOTO  UOK33L
MOVF  BARGB0,W
ADDWF  REMB2
MOVF  BARGB1,W
BTFSC  _C
INCF  BARGB1,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSC  _C
INCF  BARGB0,W
ADDWF  REMB0

UOK33L
endm

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

; 24/24 Bit Signed Fixed Point Divide 24/24 -> 24.24
; Input: 24 bit fixed point dividend in AARGBO, AARGB1, AARGB2
; 24 bit fixed point divisor in BARGBO, BARGBl, BARGB2
; Use: CALL FXD2424S
; Output: 24 bit fixed point quotient in AARGB0, AARGB1, AARGB2
; 24 bit fixed point remainder in REMB0, REMB1, REMB2
; Result: AARG, REM <- AARG / BARG
; Max Timing: 12+526+3 = 541 clks  A > 0, B > 0
; 19+526+20 = 565 clks  A > 0, B < 0
; 19+526+20 = 565 clks  A < 0, B > 0

© 1995 Microchip Technology Inc.
Min Timing:

PM: 26+207+20 = 555 elks
12+494+3 = 509 elks
19+494+20 = 533 elks
26+494+3 = 523 elks

DM:
0. B < 0
0. B > 0
0. B < 0
12

FXD2424S MOVF AARGBO,W
XORWF BARGBO,W
MOVWF SIGN
BTFSS BARGBO,MSB ; if MSB set, negate BARG
GOTO CA2424S
COMF BARGBO
BTFSC _2
DECF BARGBO
COMF BARGBO
CA2424S BTFSS AARGBO,MSB ; if MSB set, negate AARG
GOTO C2424S
COMF AARGBO
INCF AARGBO
BTFSC _2
DECF AARGBO
COMF AARGBO
CA2424S
CLRF REMBO
CLRF REMB1
CLRF REMB2
SDIV2424L
BTFSS SIGN,MSB
RETLW 0x00
COMF AARGBO
INCF AARGBO
BTFSC _2
DECF AARGBO
COMF AARGBO
C2424S
CLRF REMBO
CLRF REMB1
CLRF REMB2
SDIV2424L
BTFSS SIGN,MSB
RETLW 0x00

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

; 24/24 Bit Unsigned Fixed Point Divide 24/24 -> 24.24
; Input: 24 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2
; 24 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2
; Use: CALL FXD2424U
; Output: 24 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2
; 24 bit unsigned fixed point remainder in REMBO, REMB1, REMB2
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 3+671+2 = 676 clks
; Max Timing: 3+639+2 = 644 clks
; PM: 3+222+1 = 226 DM: 13
FXD2424U CLRF REMBO

;******************************************************************************
;******************************************************************************
23/23 Bit Unsigned Fixed Point Divide 23/23 -> 23.23
Input: 23 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
23 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
Use: CALL FXD2323U
Output: 23 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
Result: AARG, REM <-- AARG

Max Timing: 3+526+2 499 elks
Min Timing: 3+494+2 499 elks
PM: 3+207+1 211 DM: 12

FXD2323U
CLRF REMBO
CLRF
CLRF
UDIV2323L
RETLW
REMBl
REMB2
Ox00

D.5 24/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines

24/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed divide application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine Clocks Function
FXD2416S 438 24 bit/16 bit -> 24.16 signed fixed point divide
FXD2416U 529 24 bit/16 bit -> 24.16 unsigned fixed point divide
FXD2315U 407 23 bit/15 bit -> 23.15 unsigned fixed point divide

include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

Test suite storage
RANDHI equ 0x1E ; random number generator registers
RANDLO equ 0x1F
DATA equ 0xAF

Test suite for 24/16 bit fixed point divide algorithms
org 0x0005
MAIN MOVLW RAMSTART
MOVF FSR
MEMLOOP CLR FSR
INC FSR
MOVLW RAMSTOP
SUBWF FSR,W
BTFSS _2
GOTO MEMLOOP
MOVLW 0x45 ; seed for random numbers
MOVF RANDLO
MOVLW 0x30
MOVF RANDHI
MOVLW DATA
MOVF FSR
BCF  _RF0
BCF  _RF1
BCF  _IRP
CALL  RANDOM16
MOVF  RANDHI,W
MOVF  BARG0
BCF  BARG0,MSB
MOVF  RANDLO,W
MOVF  BARGB1
MOVF  INDF
INCF  FSR
MOVF  RANDHI,W
MOVF  BARG0
BCF  BARG0,MSB
MOVF  RANDLO,W
MOVF  BARGB1
MOVF  INDF
INCF  FSR
CALL  RANDOM16
MOVF  RANDHI,W
MOVF  AARGB0
MOVF  AARGB0,W
MOVF  INDF
INCF  FSR
CALL  FXD2315U
MOVF  AARGB0,W
MOVF  AARGB1,W
MOVF  INDF
INCF  FSR
MOVF  AARGB2,W
MOVF  INDF
INCF  FSR
MOVF  REMB0,W
MOVF  INDF
INCF  FSR
MOVF  REMB1,W
MOVF  INDF
INCF  FSR
GOTO  SELF
SELF GOTO SELF
; random number generator
RANDOM16
RLF  RANDHI,W
XORWF  RANDHI,W
MOVF  TEMPB0
SWAPF  RANDHI
SWAPF  RANDLO,W
MOVF  TEMPB1
RLF  TEMPB1,W
RLF  TEMPB1
MOVF  TEMPB1,W
XORWF  RANDHI,W
SWAPF  RANDHI
ANDLW  0x01
RLF  TEMPB0
RLF  RANDLO
XORWF  RANDLO
RLF  RANDHI
RETLW  0
24/16 Bit Division Macros

SDIV2416L macro

Max Timing: \(9 \times 6 + 17 + 16 + 16 + 6 \times 17 + 16 + 16 + 6 \times 17 + 16 + 8\) = 403 clks

Min Timing: \(9 \times 6 + 15 + 15 + 6 \times 16 + 15 + 15 + 6 \times 16 + 15 + 3\) = 375 clks

PW: \(7 \times 2 \times 40 + 22 + 8\) = 117

DM: 7

MOVFBARGB1,W
SUBF REMB1
MOVFBARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBF REMBU
RLF ACCB0
MOVILW 7
MOVF WSLOOPCOUNT

LOOPS2416A

RLF ACCB0,W
RLF REMB1
RLF REMB0
MOVFBARGB1,W
BTFSS ACCB0,LSB
GOTO SADD46LA
SUBF REMB1
MOVFBARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBF REMB0
GOTO SOK46LA

SADD46LA

ADDWF REMB1
MOVFBARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

SOK46LA

RLF ACCB0
DECFZ LOOPCOUNT
GOTO LOOPS2416A
RLF ACCB1,W
RLF REMB1
RLF REMB0
MOVFBARGB1,W
BTFSS ACCB0,LSB
GOTO SADD46L8
SUBF REMB1
MOVFBARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBF REMB0
GOTO SOK46L8

SADD46L8

ADDWF REMB1
MOVFBARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

SOK46L8

RLF ACCB1
MOVILW 7
MOVF WSLOOPCOUNT

LOOPS2416B

RLF ACCB1,W
RLF REMB1
RLF REMB0
MOVFBARGB1,W
BTFSS ACCB1,LSB
GOTO SADD46L8
SUBF REMB1
MOVFBARGB0,W
AN617

```
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
GOTO SOK46LB
ADDWF REMB1
MOVF BARGBO,W
BTFSZ _C
INCFSZ BARGBO,W
ADDWF REMBO

SADD46LB

SOK46LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPS2416B
RLF ACCB2,W
RLF REMB1
RLF REMBO
MOVF BARGBO,W
BTFSZ ACCB1,LSB
GOTO SADD46L16
SUBWF REMB1
MOVF BARGBO,W
BTFSZ _C
INCFSZ BARGBO,W
SUBWF REMBO
GOTO SOK46L16
ADDWF REMB1
MOVF BARGBO,W
BTFSZ _C
INCFSZ BARGBO,W
ADDWF REMBO

SADD46L16

SOK46L16
RLF ACCB2
MOVLW 7
MOVWF LOOPCOUNT
LOOPS2416C
RLF ACCB2,W
RLF REMB1
RLF REMBO
MOVF BARGBO,W
BTFSZ ACCB2,LSB
GOTO SADD46LC
SUBWF REMB1
MOVF BARGBO,W
BTFSZ _C
INCFSZ BARGBO,W
SUBWF REMBO
GOTO SOK46LC
ADDWF REMB1
MOVF BARGBO,W
BTFSZ _C
INCFSZ BARGBO,W
ADDWF REMBO

SADD46LC

SOK46LC
RLF ACCB2
DECFSZ LOOPCOUNT
GOTO LOOPS2416C
BTFSZ ACCB2,LSB
GOTO SOK46L
MOVF BARGBO,W
ADDWF REMB1
MOVF BARGBO,W
BTFSZ _C
INCFSZ BARGBO,W
ADDWF REMBO

SOK46L
endm
UDIV2416L macro
```

© 1995 Microchip Technology Inc.
Max Timing: $16 \times 6 \times 22 + 21 + 21 + 6 \times 22 + 21 + 21 + 6 \times 22 + 21 + 8 = 525$ clks

Min Timing: $16 \times 6 \times 21 + 20 + 20 + 6 \times 21 + 20 + 20 + 6 \times 21 + 20 + 3 = 497$ clks

PM: $14 + 31 + 27 + 31 + 27 + 31 + 8 = 169$

DM: 0

CLRF TEMP
RLF ACCB0, W
RLF REMB1
MOVF BARGB1, W
SUBWF REMB1
MOVF BARGB0, W
BTFS S _C
INCFSZ BARGB0, W
SUBWF REMB0
CLR W
BTFS S _C
MOVL W 1
SUBWF TEMP
RLF ACCB0
MOVL W 7
MOVL W LOOPCOUNT

LOOPU2416A RLF ACCB0, W
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB1, W
BTFS S ACCB0, LSB
GOTO UADD46LA
SUBWF REMB1
MOVF BARGB0, W
BTFS S _C
INCFSZ BARGB0, W
SUBWF REMB0
CLR W
BTFS S _C
MOVL W 1
SUBWF TEMP
GOTO UOK46LA

UADD46LA ADDWF REMB1
MOVF BARGB0, W
BTFS C _C
INCFSZ BARGB0, W
ADDWF REMB0
CLR W
BTFS C _C
MOVL W 1
ADDWF TEMP

UOK46LA RLF ACCB0
DECFSZ LOOPCOUNT
GOTO LOOPU2416A
RLF ACCB1, W
RLF REMB1
RLF REMB0
RLF TEMP
MOVF BARGB1, W
BTFS S ACCB0, LSB
GOTO UADD46L8
SUBWF REMB1
MOVF BARGB0, W
BTFS S _C
INCFSZ BARGB0, W
SUBWF REMB0
CLR W
BTFS S _C
MOVL W 1
SUBWF TEMP
GOTO UOK46L8
AN617

UADD46L8
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK46L8
RLF ACCB1
MOVLW 7
MOVF LOOPCOUNT

LOOPU2416B
RLF ACCB1,W
RLF REMB1
RLF REMBO
MOVF BARGBO,W
BTFSS ACCB1,LSB
GOTO UADD46LB
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK46LB

UADD46LB
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
BTFSC _C
MOVLW 1
ADDWF TEMP

UOK46LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPU2416B
RLF ACCB1,LSB
RLF REMB1
RLF REMBO
RLF TEMP
MOVF BARGBO,W
BTFSS ACCB1,LSB
GOTO UADD46L16
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMBO
CLRW
BTFSS _C
MOVLW 1
SUBWF TEMP
GOTO UOK46L16

UADD46L16
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMBO
CLRW
```
BTFSC _c
MOVLW 1
ADDFP TEMP

UOK46L16
RLF
MOVLW 7
ADDFP LOOPCOUNT

LOOPU2416C
RLF
ADDFP ACCB2
RLF
ADDFP REMB1
RLF
ADDFP TEMP
MOVF BARGB1,W
BTFSS ACCB2.LSB
GOTO UADD46LC
SUBWF REMB1
MOVF BARGB0,W
BTFSS _c
INCFSZ BARGB0,W
SUBWF REMB0
CLRWF
BTFSS _c
MOVLW 1
SUBWF TEMP
GOTO UOK46LC

UADD46LC
ADDFP REMB1
MOVF BARGB0,W
BTFSC _c
INCFSZ BARGB0,W
ADDFP REMB0
CLRWF
BTFSC _c
MOVLW 1
ADDFP TEMP

UOK46LC
RLF
ADDFP ACCB2
DECFZ...
GOTO...
BTFSC ACCB2.LSB
GOTO UOK46L
MOVF BARGB1,W
ADDFP REMB1
MOVF BARGB0,W
BTFSC _c
INCIF BARGB0,W
ADDFP REMB0

UOK46L
endm

UDIV2315L
macro

; Max Timing: 9*6*17+16+16+6*17+16+16+6*17+16+8 = 403 clks
; Min Timing: 9*6*16+15+6*16+15+15+6*16+15+3 = 375 clks
; PM: 7*2*40+22+8 = 117
; DM: 7
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _c
INCFSZ BARGB0,W
SUBWF REMB0
RLF
ADDFP ACCB0
MOVLW 7
ADDFP LOOPCOUNT

LOOPU2315A
RLF
ADDFP ACCB0,W
RLF
ADDFP REMB1
RLF
ADDFP REMB0
MOVF BARGB1,W
BTFSS ACCB0.LSB
GOTO UADD35LA
```
AN617

SUBWF REMBl
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK35LA

UADD35LA
ADDWF REMBl
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

UOK35LA
RLF ACCEB0
DECFSZ LOOPCOUNT
GOTO LOOPU2315A
RLF ACCB1,W
RLF REMBl
RLF REMB0
MOVF BARGB1,W
BTFSS ACCB0,LSB
GOTO UADD35L8
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK35L8

UADD35L8
ADDWF REMBl
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

UOK35L8
RLF ACCB1
MOVWF LOOPCOUNT
LOOPU2315B
RLF ACCB1,W
RLF REMBl
RLF REMB0
MOVF BARGB1,W
BTFSS ACCB1,LSB
GOTO UADD35LB
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMB0
GOTO UOK35LB

UADD35LB
ADDWF REMBl
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

UOK35LB
RLF ACCB1
DECFSZ LOOPCOUNT
GOTO LOOPU2315B
RLF ACCB2,W
RLF REMBl
RLF REMB0
MOVF BARGB1,W
BTFSS ACCB1,LSB
GOTO UADD35L16
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C

DS00617A-page 2-320 © 1995 Microchip Technology Inc.
AN617

24/16 Bit Signed Fixed Point Divide 24/16 -> 24.16

Input: 24 bit fixed point dividend in AARGBO, AARB1, AARB2
16 bit fixed point divisor in BARGBO, BARGB1

Use: CALL FXD2416S

Output: 24 bit fixed point quotient in AARGBO, AARB1, AARB2
16 bit fixed point remainder in REMBO, REMB1

Result: AARG, REM <-- AARG / BARG

Max Timing: 11+403+3 = 417 clks A > 0, B > 0
15+403+17 = 435 clks A > 0, B < 0
18+403+17 = 443 clks A < 0, B > 0
22+403+3 = 430 clks A < 0, B < 0

Min Timing: 11+375+3 = 389 clks A > 0, B > 0
15+375+17 = 407 clks A > 0, B < 0
18+375+17 = 415 clks A < 0, B > 0
22+375+3 = 402 clks A < 0, B < 0

PM: 22+117+16 = 140 EM: 8

FXD2416S MOVF AARGBO,W
XORWF AARB1,W
MOVWF SIGN
BTFSS BARGBO,MSB ; if MSB set, negate BARG
24/16 Bit Unsigned Fixed Point Divide 24/16 -> 24.16
Input: 24 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2
16 bit unsigned fixed point divisor in BARGBO, BARGBl
Use: CALL FXD2416U
Output: 24 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2
16 bit unsigned fixed point remainder in REMBO, REMBl
Result: AARG, REM <-- AARG / BARG
Max Timing: 2+525+2 = 529 clks
Min Timing: 2+497+2 = 501 clks
PM: 2+169+1 = 172 DM: 8
FXD2416U CLRF REMBO
FXD2416U CLRF REMB1
FXD2416U UDIV2416L
FXD2416U RETLW 0x00

23/15 Bit Unsigned Fixed Point Divide 23/15 -> 23.15
Input: 23 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2
15 bit unsigned fixed point divisor in BARGBO, BARGBl
Use: CALL FXD2315U
Output: 23 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2
15 bit unsigned fixed point remainder in REMBO, REMBl
Result: AARG, REM <-- AARG / BARG
Max Timing: 2+403+2 = 407 clks
Min Timing: 2+375+2 = 379 clks
PM: 2+117+1 = 120 DM: 7
FXD2315U CLRF REMBO
D.6 16/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines

16/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two argument can be called in a signed divide application where it is known that the respective argument is nonnegative, thereby offering some improvement in performance.

Routine          Clocks Function
FXD1616S         319 16 bit/16 bit -> 16.16 signed fixed point divide
FXD1616U         373 16 bit/16 bit -> 16.16 unsigned fixed point divide
FXD1515U         294 15 bit/15 bit -> 15.15 unsigned fixed point divide
The above timings are based on the looped macros. If space permits, approximately 65-69 clocks can be saved by using the unrolled macros.

Test suite storage
RANDHI           equ 0x1A ; random number senerator registers
RANDLO           equ 0x1B
TESTCOUNT        equ 0x20 ; counter
DATA             equ 0x21 ; beginning of test data

Test suite for 16/16 bit fixed point divide algorithms
org Ox0005
MAIN
MOVLW RAMSTART
MOVWF FSR
MEMLOOP
CLRF INDF
INCF FSR
MOVLW RAMSTOP
SUBWF FSR,W
BTFSS _z
GOTO MEMLOOP
MOVLW Ox45 ; seed for random numbers
MOVWF RANDLO
MOVLW 0x30
MOVWF RANDHI
MOVLW DATA
MOVWF FSR
BCF _RP0
BCF _RP1
BCF _IRP
CALL TFXD1616
MOVLW 0x99
MOVWF AARGB0
MOVLW 0xAB
MOVWF AARGB1
MOVLW 0x00
MOVWF BARGB0
MOVLF 0xFF
MOVWF BARGBL
CALL FXD1616S
SELF
GOTO SELF
RANDOM16
RLF RANDHI,W ; random number generator
XORWF RANDHI,W
AN617

MOVWF TEMPBO
SWAFF RANDHI
SWAFF RANDLO,W
MOVWF TEMPBI
RLF TEMPBI
RLF TEMPBI
MOVF TEMPBI,W
XORWF RANDHI,W
SWAFF RANDHI
ANDlw 0x01
RLF TEMPBO
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETLW 0

; Test suite for FXD1616
TFXD1616 MOVLW 2
MOVWF TESTCOUNT
D1616LOOP CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGBO
; BCF BARGBO,MSB
; MOVF BARGBO,W
MOVWF INDF
INCf FSR
MOVF RANDLO,W
MOVWF BARGBl
MOVWF INDF
INCF FSR
CALL FXD1616S
MOVF AARGBO
BCF AARGBO,MSB
MOVF AARGBO,W
MOVWF INDF
INCf FSR
MOVF AARGBl,W
MOVWF INDF
INCf FSR
MOVF REMBO,W
MOVWF INDF
INCf FSR
MOVF REMBl,W
MOVWF INDF
INCf FSR
DECFSZ TESTCOUNT
GOTO D1616LOOP
RETLW 0x00

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

16/16 Bit Division Macros

SDIV1616L macro
; Max Timing: 13+14*18+17+8 = 290 clks
; Min Timing: 13+14*16+15+3 = 255 clks
; PM: 42
RLF ACCBO,W
RLF   REMB1
RLF   REMB0
MOVF  BARGB1,W
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF  REMB0
RLF   ACCB1
RLF   ACCB0
MOVLW 15
MOVWF LOOPCOUNT

LOOPS1616
RLF   ACCB0,W
RLF   REMB1
RLF   REMB0
MOVF  BARGB1,W
BTFSS ACCB1,LSB
GOTO  SADD66L
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF  REMB0
GOTO  SOK66LL
ADDWF REMB1
MOVF  BARGB0,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

SADD66L
RLF   ACCB1
RLF   ACCB0
DECFSZ LOOPCOUNT
GOTO  LOOPS1616
ADDWF REMB1
MOVF  BARGB1,W
BTFSC _C
INCFSZ BARGB0,W
ADDWF REMB0

SOK66LL
RLF   ACCB1
RLF   ACCB0
MOVLW 16
MOVWF LOOPCOUNT
LOOPU1616
RLF   ACCB0,W
RLF   REMB1
RLF   REMB0
MOVF  BARGB1,W
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF  REMB0
BTFSC _C
GOTO  UOK66LL
MOVF  BARGB0,W
BTFSC _C
INCFSZ BARGB0,W

UDIV1616L macro
; restore = 23 clks, nonrestore = 17 clks
; Max Timing:  2+15*23+22 = 369 clks
; Min Timing:  2+15*17+16 = 273 clks
;  PM: 24       DM: 7
MOVLW 16
MOVWF LOOPCOUNT
UOK66LL
UDIV1515L macro
; Max Timing: 13+14*18+17+8 = 290 clks
; Min Timing: 13+14*17+16+3 = 270 clks
; PM: 42
RLF ACCBO,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMB0
RLF ACCB1
RLF ACCBO
MOVLW 15
MOVWF LOOPCOUNT
UDIV1515 macro
; Max Timing: 7+10+6*14+14+7*14+8 = 221 clks
; Min Timing: 7+10+6*13+13+7*13+3 = 202 clks
; PM: 7+10+6*18+18+7*18+8 = 277 DM: 6
variable i
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INC FSZ BARGBO,W
SUBWF REMB0
RLF ACCB1
RLF ACCBO
DECF SZ LOOPCOUNT
GOTO LOOPU1515
BTFSC ACCB1,LSB
GOTO UOK555L
MOVF BARGB1,W
ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INC F BARGBO,W
ADDWF REMB0
UOK55LL endm
SDIV1616 macro
; Max Timing: 7*10+6*14+14+7*14+8 = 221 clks
; Min Timing: 7*10+6*13+13+7*13+3 = 202 clks
; PM: 7*10+6*18+18+7*18+8 = 277 DM: 6
variable i
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INC FSZ BARGBO,W
SUBWF REMB0
UOK55L endm
```
RLF   ACCB0
RLF   ACCB0,W
RLF   REMB1
RLF   REMB0
MOVF  BARGBl,W
ADDWF  REMB1
MOVF  BARGB0,W
BTFSC  _C
INCFSZ  BARGB0,W
ADDWF  REMB0
RLF   ACCB0
i = 2
while i < 8
RLF   ACCB0,W
RLF   REMB1
RLF   REMB0
MOVF  BARGBl,W
BTFSS  ACCB0,LSB
GOTO  SADD66#v(i)
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
SUBWF  REMB0
GOTO  SOK66#v(i)
SADD66#v(i)
ADDWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
ADDWF  REMB0
SOK66#v(i)
RLF   ACCB0
i = i + 1
endw
RLF   ACCB1,W
RLF   REMB1
RLF   REMB0
MOVF  BARGBl,W
BTFSS  ACCB0,LSB
GOTO  SADD668
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
SUBWF  REMB0
GOTO  SOK668
SADD668
ADDWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
ADDWF  REMB0
SOK668
RLF   ACCB1
i = 9
while i < 16
RLF   ACCB1,W
RLF   REMB1
RLF   REMB0
MOVF  BARGBl,W
BTFSS  ACCB1,LSB
GOTO  SADD66#v(i)
SUBWF  REMB1
MOVF  BARGB0,W
BTFSS  _C
INCFSZ  BARGB0,W
SUBWF  REMB0
GOTO  SOK66#v(i)
SADD66#v(i)
ADDWF  REMB1
```
UDIV1616 macro
; restore = 20 clks, nonrestore = 14 clks
; Max Timing: 16*20 = 320 clks
; Min Timing: 16*14 = 224 clks
; PM: 16*20 = 320  DM: 6
variable i
i = 0
while i < 16
RLF ACCBO,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
ADDWF REMBO
BTFSC _C
GOTO UOK66#v(i)
MOVF BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
ADDWF REMBO
BCF _C
UOK66#v(i)
RLF ACCBO
RLF ACCBO,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMBO
RLF ACCBO
RLF ACCBO,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
ADDWF REMBO
MOVF BARGB0,W

UDIV1515 macro
; Max Timing: 7+10+6*14+14+7*14+8 = 221 clks
; Min Timing: 7+10+6*13+13+7*13+3 = 202 clks
; PM: 7+10+6*18+18+7*18+8 = 277  DM: 6
variable i
MOVF BARGB1,W
SUBWF REMB1
MOVF BARGB0,W
BTFSS _C
INCFSZ BARGB0,W
SUBWF REMBO
RLF ACCBO
RLF ACCBO,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
ADDWF REMB1
MOVF BARGB0,W
ADDWF REMBO
MOVF BARGB0,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMB0
RLF ACCBO
i = 2
while i < 8
RLF ACCBO,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
BTFSS ACCBO,LSB
GOTO UADD55#v(i)
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMB0
GOTO UOK55#v(i)
UADD55#v(i) ADDWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
ADDWF REMB0
UOK55#v(i) RLF ACCBO
i = i + 1
endw
RLF ACCB1,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
BTFSS ACCBO,LSB
GOTO UADD558
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMB0
GOTO UOK558
UADD558 ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMB0
UOK558 RLF ACCB1
i = 9
while i < 16
RLF ACCB1,W
RLF REMB1
RLF REMB0
MOVF BARGB1,W
BTFSS ACCB1,LSB
GOTO UADD55#v(i)
SUBWF REMB1
MOVF BARGBO,W
BTFSS _C
INCFSZ BARGBO,W
SUBWF REMB0
GOTO UOK55#v(i)
UADD55#v(i) ADDWF REMB1
MOVF BARGBO,W
BTFSC _C
INCFSZ BARGBO,W
ADDWF REMB0
UOK55#v(i) RLF ACCB1
i = i + 1
endw
BTFS
ACCB1,LSB
GOTO
UOK55
MOVF
BARGBl,W
ADDWF
REMB1
MOVF
BARGBO,W
BTFS
_c
INCF
BARGBO,W
ADDWF
REMB0
UOK55
endm
**********************************************************************************************

16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16

Input: 16 bit fixed point dividend in AARGBO, AARGBl
16 bit fixed point divisor in BARGBO, BARGBl

Use: CALL FXD1616S

Output: 16 bit fixed point quotient in AARGBO, AARGBl
16 bit fixed point remainder in REMBO, REMBl

Result: AARG, REM <-- AARG

Max Timing: 11+290+3 = 304 clks
15+290+14 = 319 clks
15+290+14 = 319 clks
19+290+3 = 312 clks

Min Timing: 11+255+3 = 269 clks
15+255+14 = 284 clks
15+255+14 = 284 clks
19+255+3 = 277 clks

PM: 19+42+13 = 74
DM: 8

FXD1616S
MOVF
AARGBO,W
XORWF
BARGBO,W
MOVWF
SIGN
BTFS
BARGBO,MSB
; if MSB set, negate BARG
GOTO
CA1616S
COMF
BARGBl
INCF
BARGBl
BTFS
_z
DECF
BARGBO
COMF
BARGBO

CA1616S
BTFS
AARGBO,MSB
; if MSB set, negate AARG
GOTO
C1616S
COMF
AARGBl
INCF
AARGBl
BTFS
_z
DECF
AARGBO
COMF
AARGBO

C1616S
CLR
REMB0
CLR
REMB1
SDIV1616L
BTFS
SIGN,MSB
RETLW
0x00
COMF
AARGBO
INCF
AARGBl
BTFS
_z
DECF
AARGBO
COMF
AARGBO
COMF
REMB1
INCF
REMB1
BTFS
_z
DECF
REMB0
COMF
REMB0
RETLW
0x00
**********************************************************************************************

16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16
Input: 16 bit unsigned fixed point dividend in AARGBO, AARGB1
Use: CALL FXD1616U
Output: 16 bit unsigned fixed point quotient in AARGBO, AARGB1
16 bit unsigned fixed point remainder in REMBO, REMB1
Result: AARG, REM <-- AARG
Max Timing: 2+369+2 = 373 elks
Min Timing: 2+273+2 = 277 elks
PM: 2+24+1 = 27
FXD1616U
CLRFB REMBO
CLRFB REMB1
UDIV1616L
RETLW 0x00

15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15
Input: 15 bit unsigned fixed point dividend in AARGBO, AARGB1
15 bit unsigned fixed point divisor in BARGBO, BARGB1
Use: CALL FXD1515U
Output: 15 bit unsigned fixed point quotient in AARGBO, AARGB1
15 bit unsigned fixed point remainder in REMBO, REMB1
Result: AARG, REM <-- AARG
Max Timing: 2+290+2 = 294 elks
Min Timing: 2+270+2 = 274 elks
PM: 2+42+1 = 45
FXD1515U
CLRFB REMBO
CLRFB REMB1
UDIV1515L
RETLW 0x00

D.7 16/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines

16/8 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed divide applicaiton where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine	Clocks	Function
FXD1608S	188 16 bit/8 bit --> 16.08 signed fixed point divide
FXD1609U	294 16 bit/8 bit --> 16.08 unsigned fixed point divide
FXD1607U	174 16 bit/7 bit --> 16.07 unsigned fixed point divide
FXD1507U	166 15 bit/7 bit --> 15.07 unsigned fixed point divide
The above timings are based on the looped macros. If space permits,
approximately 41-50 clocks can be saved by using the unrolled macros.
list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

Test suite storage
RANDMI		0x1A ; random number generator registers
RANDLO	equ 0x1B
TESTCOUNT	equ 0x20 ; counter
DATA	equ 0x31 ; beginning of test data

Test suite for 16/8 bit fixed point divide algorithms
org 0x0005
MAIN
MOVLW RAMSTART
MOVF  FSR
MEMLOOP

CLRF INDF
INC F FSR
MOVLW RAMSTOP
SUBWF FSR,W
BTFSS _Z
GOTO MEMLOOP
MOVLW 0x45
MOVF RANDLO
MOVLW 0x30
MOVF RANDHI
MOVLW DATA
MOVF FSR
BCF _BP0
BCF _BP1
BCF _IRP
CALL TFXD1608
MOVLW 0x99
MOVF AARGBO
MOVLW 0xAB
MOVF AARGB1
MOVLW 0xFF
MOVF BARGBO
CALL FXD1608U
SELF GOTO SELF
RANDOM16
RLF RANDHI,W
XORWF RANDHI,W
MOVWF TEMPB0
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPB1
RLF TEMPB1,W
RLF TEMPB1
MOVF TEMPB1,W
XORWF RANDHI,W
SWAPF RANDHI
ANDLW 0x01
RLF TEMPB0
RLF RANDLO
XORWF RANDLO
RLF RANDHI
RETLW 0
;
; Test suite for FXD1608
TFXD1608
MOVLW 3
MOVWF TESTCOUNT
D1608LOOP
CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGBO
;
BCF BARGBO,MSB
;
MOVF BARGBO,W
MOVWF INDF
INC F FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF AARGBO
;
BCF AARGBO,MSB
;
MOVF AARGBO,W
MOVWF INDF
INC F FSR
MOVF RANDLO,W
MOVWF AARGB1
MOVWF INDF
INC F FSR
CALL FXD1608U
MOVF AARGBO,W
AN617

MOVWF  INDF
INCF  FSR
MOVF  AARGBl,W
MOVWF  INDF
INCF  FSR
MOVF  REMBO,W
MOVWF  INDF
INCF  FSR
DECFSZ  TESTCOUNT
GOTO  D1608LOOP
RETLW  Ox00

;**********************************************************************************************
; 16/08 BIT Division Macros
SDIV1608L  macro

; Max Timing: 3+5+2+5*11+10+10+6*11+10+2 = 163 clks
; Min Timing: 3+5+2+5*11+10+10+6*11+10+2 = 163 clks
; PM: 42
; DM: 5

;MOVF  BARGBO,W
;SUBWF  REMBO
;RLF  ACCBO
;RLF  ACCBO,W
;RLF  REMBO
;MOVF  BARGBO,W
;ADDWF  REMBO
;RLF  ACCBO
;MOVLW  6
;MOVWF  LOOPCOUNT
;LOOPS1608A  RLF  ACCBO,W
;RLF  REMBO
;MOVF  BARGBO,W
;BTFSC  ACCBO,LSB
;SUBWF  REMBO
;BTFSS  ACCBO,LSB
;ADDWF  REMBO
;RLF  ACCBO
;DECFSZ  LOOPCOUNT
;GOTO  LOOPS1608A
;RLF  ACCBO,W
;RLF  REMBO
;MOVF  BARGBO,W
;BTFSC  ACCBO,LSB
;SUBWF  REMBO
;BTFSS  ACCBO,LSB
;ADDWF  REMBO
;RLF  ACCBO
;MOVLW  7
;MOVWF  LOOPCOUNT
;LOOPS1608B  RLF  ACCBO,W
;RLF  REMBO
;MOVF  BARGBO,W
;BTFSC  ACCBO,LSB
;SUBWF  REMBO
;BTFSS  ACCBO,LSB
;ADDWF  REMBO
;RLF  ACCBO
;DECFSZ  LOOPCOUNT
;GOTO  LOOPS1608B
;RLF  ACCBO1
;RLF  REMBO
;MOVF  BARGBO,W
;BTFSC  ACCBO1,LSB
;SUBWF  REMBO
;BTFSS  ACCBO1,LSB
;ADDWF  REMBO
;RLF  ACCBO1
;DECFSZ  LOOPCOUNT
;GOTO  LOOPS1608B
;BFSS  ACCBO1,LSB
;ADDWF  REMBO
;endm

UDIV1608L  macro

; Max Timing: 2+7*12+11+3+7*24+23 = 291 clks
; Min Timing: 2+7*11+10+3+7*17+16 = 227 clks
; PM: 39
; DM: 7

;MOVLW  8

© 1995 Microchip Technology Inc.
AN617

LOOPU1608A

MOVWF LOOPCOUNT
RLF ACCB0,W
RLF REMB0
MOVF BARGB0,W
SUBWF REMB0
BTFSC _C
GOTO UOK68A
ADDWF REMB0
BCF _C

UOK68A

RLF ACCB0
DECSZ LOOPCOUNT
GOTO LOOPU1608A
CLRF TEMP
MOVLW B
MOVWF LOOPCOUNT

LOOPU1608B

RLF ACCB1,W
RLF REMB0
RLF TEMP
MOVF BARGB0,W
SUBWF REMB0
CLRF ACCB5
CLR W
BTFSC _C
INCFSZ ACCB5,W
SUBWF TEMP
BTFSC _C
GOTO UOK68B
MOVF BARGB0,W
ADDWF REMB0
CLR W
BTFSC _C
INCFSZ ACCB5,W
ADDWF TEMP
BCF _C

UOK68B

RLF ACCB1
DECSZ LOOPCOUNT
GOTO LOOPU1608B

endm

UDIV1607L macro

; Max Timing: 7+6*11+10+6*11+10+2 = 171 clks
; Min Timing: 7+6*11+10+6*11+10+2 = 171 clks
; PM: 39
; DM: 5

RLF ACCB0,W
RLF REMB0
MOVF BARGB0,W
SUBWF REMB0
RLF ACCB0
MOVLW 7
MOVWF LOOPCOUNT

LOOPU1607A

RLF ACCB0,W
RLF REMB0
MOVF BARGB0,W
BTFSC ACCB0,LSB
SUBWF REMB0
BTFSS ACCB0,LSB
ADDWF REMB0
RLF ACCB0
DECSZ LOOPCOUNT
GOTO LOOPU1607A
RLF ACCB1,W
RLF REMB0
MOVF BARGB0,W
BTFSC ACCB0,LSB
SUBWF REMB0
BTFSS ACCB0,LSB

© 1995 Microchip Technology Inc.
ADDWF REMBO
RLF ACCBl
MOVLW 7
MOVF LOOPCOUNT

LOOPU1607B
RLF ACCBl,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBl,LSB
SUBWF REMBO
BTFSS ACCBl,LSB
ADDWF REMBO
RLF ACCBl
DECFSZ LOOPCOUNT
GOTO LOOPU1607B
BTFSS ACCBl,LSB
ADDWF REMBO

UDIV1507L macro
Max Timing: 3+5+2+5*11+10+6*11+10+2 = 163 clks
Min Timing: 3+5+2+5*11+10+6*11+10+2 = 163 clks
PM: 42 DM: 5
MOV BARGBO,W
SUBWF REMBO
RLF ACCB0
RLF ACCB0,W
RLF REMBO
MOVF BARGBO,W
ADDWF REMBO
RLF ACCB0
MOV LW 6
MOVF LOOPCOUNT

LOOPU1507A
RLF ACCB0,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCB0,LSB
SUBWF REMBO
BTFSS ACCB0,LSB
ADDWF REMBO
RLF ACCB0
DECFSZ LOOPCOUNT
GOTO LOOPU1507A
RLF ACCBl,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBl,LSB
SUBWF REMBO
BTFSS ACCBl,LSB
ADDWF REMBO
RLF ACCBl
MOV LW 7
MOVF LOOPCOUNT

LOOPU1507B
RLF ACCBl,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBl,LSB
SUBWF REMBO
BTFSS ACCBl,LSB
ADDWF REMBO
RLF ACCBl
DECFSZ LOOPCOUNT
GOTO LOOPU1507B
BTFSS ACCBl,LSB
ADDWF REMBO
endm

SDIV1608 macro
Max Timing: 3+5+14*8+2 = 122 clks
Min Timing: 3+5+14*8+2 = 122 clks

PM: 122

variable i

MOVF BARGBO,W
SUBWF REMBO
RLF ACCBO
RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
ADDWF REMBO
RLF ACCBO

while i < 8

RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBO,LSB
SUBWF REMBO
BTFSS ACCBO,LSB
ADDWF REMBO
RLF ACCBO
i=i+1
endw

RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBO,LSB
SUBWF REMBO
BTFSS ACCBO,LSB
ADDWF REMBO
RLF ACCBO

i = 9
while i < 16

RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBO,LSB
SUBWF REMBO
BTFSS ACCBO,LSB
ADDWF REMBO
RLF ACCBO
i=i+1
endw

BTFSS ACCBO,LSB
ADDWF REMBO
endm

UDIV1608 macro

restore = 9/21 clks, nonrestore = 8/14 clks

Max Timing: 8*9+1+8*21 = 241 clks

Min Timing: 8*8+1+8*14 = 177 clks

PM: 241

variable i

i = 0
while i < 8

RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
SUBWF REMBO
BTFSC _C
GOTO UOK68#v(i)
ADDWF REMBO
BCF _C

UOK68#v(i) RLFI ACCBO
i=i+1
endw

CLRF TEMP
i = 8
while i < 16
  RLF ACCB1,W
  RLF REMB0
  RLF TEMP
  MOVF BARGBO,W
  SUBWF REMB0
  CLRIF ACCB5
  CLRW
  BTFSS _C
  INCFSZ ACCB5,W
  SUBWF TEMP
  BTFSC _C
  GOTO UOK68#v(i)
  MOVF BARGBO,W
  ADDWF REMB0
  CLRIF ACCB5
  CLRW
  BTFSACC _C
  INCFSZ ACCB5,W
  ADDWF TEMP
  BCF _C
  UOK68#v(i)
  RLF ACCB1
  i = i + 1
endw
endm

UDIV1607

; Max Timing: 5+15*8+2 = 127 clks
; Min Timing: 5+15*8+2 = 127 clks
; PM: 127 DM: 4

variable i
  RLF ACCB0,W
  RLF REMB0
  MOVF BARGBO,W
  SUBWF REMB0
  RLF ACCB0
  i = 1
while i < 8
  RLF ACCB0,W
  RLF REMB0
  MOVF BARGBO,W
  BTFSACC ACCB0,LSB
  SUBWF REMB0
  BTFSACC ACCB0,LSB
  ADDWF REMB0
  RLF ACCB0
  i = i + 1
endw

  RLF ACCB1,W
  RLF REMB0
  MOVF BARGBO,W
  BTFSACC ACCB0,LSB
  SUBWF REMB0
  BTFSACC ACCB0,LSB
  ADDWF REMB0
  RLF ACCB0
  i = 9
while i < 16
  RLF ACCB1,W
  RLF REMB0
  MOVF BARGBO,W
  BTFSACC ACCB1,LSB
  SUBWF REMB0
  BTFSACC ACCB1,LSB
  ADDWF REMB0
  RLF ACCB1
16/8 Bit Signed Fixed Point Divide 16/8 → 16.08
Input: 16 bit signed fixed point dividend in AARGBO, AARGBl
8 bit signed fixed point divisor in BARGBO
Use: CALL FXD1608S
Output: 16 bit signed fixed point quotient in AARGBO, AARGBl
8 bit signed fixed point remainder in REMBO
Result: AARG, REM ← AARG / BARG
Max Timing: 10+163+3 = 176 clks A > 0, B > 0
11+163+11 = 185 clks A > 0, B < 0
; 14+163+11 = 188 clks  A < 0, B > 0
; 15+163+3 = 181 clks  A < 0, B < 0
; Min Timing: 10+163+3 = 176 clks  A > 0, B > 0
; 11+163+11 = 185 clks  A > 0, B < 0
; 14+163+11 = 188 clks  A < 0, B > 0
; 15+163+3 = 181 clks  A < 0, B < 0
; FM: 15*42+10 = 67  DM: 6
FXD1608S
  MOVF AARGBO,W
  XORWF BARGBO,W
  MOVWF SIGN
  BTFS BARGBO,MSB ; if MSB set, negate BARG
  GOTO CA1608S
  COMF BARGBO
  INCF BARGBO
CA1608S
  BTFS AARGBO,MSB ; if MSB set, negate AARG
  GOTO C1608S
  COMF AARGBl
  INCF AARGBl
  BTFS _Z
  DECFS AARGBO
  COMF AARGBO
C1608S
  CLRF REMBO
  SDIV1608L
  BTFS SIGN,MSB
  RETLW 0x00
  COMF AARGBl
  INCF AARGBl
  BTFS _Z
  DECFS AARGBO
  COMF AARGBO
  COMF REMBO
  INCFS REMBO
  RETLW 0x00

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

; 16/8 Bit Unsigned Fixed Point Divide 16/8 -> 16.08
; Input: 16 bit unsigned fixed point dividend in AARGBO, AARGBl
; 8 bit unsigned fixed point divisor in BARGBO
; Use: CALL FXD1608U
; Output: 16 bit unsigned fixed point quotient in AARGBO, AARGBl
; 8 bit unsigned fixed point remainder in REMBO
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 1+291+2 = 294 clks
; Min Timing: 1+227+2 = 230 clks
; PM: 1+39+1 = 41  DM: 7
FXD1608U
  CLRF REMBO
  UDIV1608L
  RETLW 0x00

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

; 16/7 Bit Unsigned Fixed Point Divide 16/7 -> 16.07
; Input: 16 bit unsigned fixed point dividend in AARGBO, AARGBl
; 7 bit unsigned fixed point divisor in BARGBO
; Use: CALL FXD1607U
; Output: 16 bit unsigned fixed point quotient in AARGBO, AARGBl
; 7 bit unsigned fixed point remainder in REMBO
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 1+171+2 = 174 clks
; Min Timing: 1+171+2 = 174 clks
; PM: 1+39+1 = 41  DM: 5
FXD1607U
  CLRF REMBO
  UDIV1607L
  RETLW 0x00

;**********************************************************************************************
D.8 8/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines

8/8 PIC16 FIXED POINT DIVIDE Routines VERSION 1.7

Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG in AARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two argument can be called in a signed divide application where it is known that the respective argument is nonnegative, thereby offering some improvement in performance.

Routine | Clocks | Function
---------|--------|---------
FXD0808S | 96 | 8 bit/8 bit -> 08.08 signed fixed point divide
FXD0808U | 100 | 8 bit/8 bit -> 08.08 unsigned fixed point divide
FXD0807U | 88 | 8 bit/7 bit -> 08.07 unsigned fixed point divide
FXD0707U | 80 | 7 bit/7 bit -> 07.07 unsigned fixed point divide

The above timings are based on the looped macros. If space permits, approximately 19-25 clocks can be saved by using the unrolled macros.

list r=dec,x=on,t=off
include <PIC16.INC> ; general PIC16 definitions
include <MATH16.INC> ; PIC16 math library definitions

Test suite storage
RANDHI equ 0x1A ; random number generator registers
RANDLO equ 0x1B
TESTCOUNT equ 0x20 ; counter
DATA equ 0x21 ; beginning of test data

Test suite for 8/8 bit fixed point divide algorithms
org Ox0005
MAIN
MOVE 0x0005
MEMLOOP
CLR FSR
INC FSR
MOVWF RAMSTOP
SUBWF FSR,W
BTFSS _z
GOTO MEMLOOP
MOVWF RANDLO
MOVWF 0x45 ; seed for random numbers
MOVWF RANDHI
BCF _RF0
BCF _RF1
BCF _IRP
CALL TFXD0808
SELF GOTO SELF
RANDOM16 RLF RANDHI,W ; random number generator
XORWF RANDHI,W
MOVWF TEMPH1
SWAPF RANDHI
SWAPF RANDLO,W
MOVWF TEMPH1
RLF TEMPH1,W
RLF TEMPH1
MOVF TEMPH1,W
XORWF RANDHI,W
SWAPF RANDHI
RANDLW 0x01
RLF TEMPH0
RLF RANDLO
XORWF RANDLO
RLF RANDHI

RETLW 0

; Test suite for FXD0808
TFXD0808 MOVLW 3
MOVWF TESTCOUNT
D0808LOOP
CALL RANDOM16
MOVF RANDHI,W
MOVWF BARGBO
; BCF BARGBO,MSB
; MOVF BARGBO,W
MOVWF INDF
INCF FSR
CALL RANDOM16
MOVF RANDHI,W
MOVWF AARGBO
; BCF AARGBO,MSB
; MOVF AARGBO,W
MOVWF INDF
INCF FSR
CALL FXD0808S
MOVF AARGBO,W
MOVWF INDF
INCF FSR
MOVF REMBO,W
MOVWF INDF
INCF FSR
DCFSZ TESTCOUNT
GOTO D0808LOOP
RETLW 0x00

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

08/08 BIT Division Macros

SDIV0808L macro
; Max Timing: 3+5+2+5*11+10+2 = 77 clks
; Min Timing: 3+5+2+5*11+10+2 = 77 clks
; PM: 22
MOVF BARGBO,W
SUBWF REMBO
RLF ACCBO
RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
ADDWF REMBO
RLF ACCBO
MOVLW 6
MOVWF LOOPCOUNT

LOOPS0808A RLF ACCBO,W
AN617

```
RLF REMB0
MOVF BARGBO,W
BTFSC ACCBO,LSB
SUBWF REMB0
BTFSS ACCBO,LSB
ADDWF REMB0
RLF ACCBO
DECFSZ LOOPCOUNT
GOTO LOOPU0808A
BTFSS ACCBO,LSB
ADDWF REMB0
endm

UDIV0808L macro
; Max Timing: 2+7*12+11 = 97 clks
; Min Timing: 2+7*11+10 = 89 clks
; PM: 13
DM: 4
MOVLW 8
MOVWF LOOPCOUNT
LOPU0808A
RLF ACCBO,W
RLF REMB0
MOVF BARGBO,W
SUBWF REMB0
BTFSC _C
GOTO UOK88A
ADDWF REMB0
BCF _C
UOK88A
RLF ACCBO
DECFSZ LOOPCOUNT
GOTO LOOPU0808A
endm

UDIV0807L macro
; Max Timing: 7+6*11+10+2 = 85 clks
; Min Timing: 7+6*11+10+2 = 85 clks
; PM: 19
DM: 4
RLF ACCBO,W
RLF REMB0
MOVF BARGBO,W
SUBWF REMB0
RLF ACCBO
MOVLW 7
MOVWF LOOPCOUNT
LOPU0807
RLF ACCBO,W
RLF REMB0
MOVF BARGBO,W
BTFSC ACCBO,LSB
SUBWF REMB0
BTFSS ACCBO,LSB
ADDWF REMB0
RLF ACCBO
DECFSZ LOOPCOUNT
GOTO LOOPU0807
BTFSS ACCBO,LSB
ADDWF REMB0
endm

UDIV0707L macro
; Max Timing: 3+5+2+5*11+10+2 = 77 clks
; Min Timing: 3+5+2+5*11+10+2 = 77 clks
; PM: 22
DM: 4
MOVF BARGBO,W
SUBWF REMB0
RLF ACCBO
RLF ACCBO,W
RLF REMB0
MOVF BARGBO,W
ADDWF REMB0
RLF ACCBO
```
MOVLW 6
MOVWF LOOPCOUNT

LOOPU0707
RLF ACCB0, W
RLF REMB0
MOVF BARGBO, W
BTFSC ACCB0, LSB
SUBWF REMB0
BTFSS ACCB0, LSB
ADDF REMB0
RLF ACCB0
DECSZ LOOPCOUNT
GOTO LOOPU0707
BTFSS ACCB0, LSB
ADDF REMB0
endm

SDIV0808
macro
Max Timing: 3+5+6*8+2 = 58 clks
Min Timing: 3+5+6*8+2 = 58 clks
variable i
MOVF BARGBO, W
SUBWF REMB0
RLF ACCB0
RLF ACCB0, W
RLF REMB0
MOVF BARGBO, W
ADDF REMB0
RLF ACCB0
i = 2
while i < 8
RLF ACCB0, W
RLF REMB0
MOVF BARGBO, W
BTFSC ACCB0, LSB
SUBWF REMB0
BTFSS ACCB0, LSB
ADDF REMB0
RLF ACCB0
i=i+1
endw
BTFSS ACCB0, LSB
ADDF REMB0
endm

UDIV0808
macro
restore = 9 clks, nonrestore = 8 clks
Max Timing: 8*9 = 72 clks
Min Timing: 8*8 = 64 clks
variable i
i = 0
while i < 8
RLF ACCB0, W
RLF REMB0
MOVF BARGBO, W
SUBWF REMB0
BTFSC _C
GOTO UOK88#v(i)
ADDF REMB0
BCF _C
UOK88#v(i)
RLF ACCB0
i=i+1
endw
endm

UDIV0807
macro
Max Timing: 5+7*8+2 = 63 clks
Min Timing: 5+7*8+2 = 63 clks
variable i
RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
SUBWF REMBO
RLF ACCBO
i = 1
while i < 8
RLF ACCBO,W
RLF REMBO
MOVF BARGBO,W
BTFSC ACCBO,LSB
SUBWF REMBO
BTFSS ACCBO,LSB
ADDWF REMBO
RLF ACCBO
i = i + 1
endw
BTFSS ACCBO,LSB
ADDWF REMBO
endm

UDIV0707
macro
Max Timing: 3+5+6*8+2 = 58 clks
Min Timing: 3+5+6*8+2 = 58 clks

8/8 Bit Signed Fixed Point Divide 8/8 -> 08.08
Input: 8 bit signed fixed point dividend in AARGBO
8 bit signed fixed point divisor in BARGBO
Use: CALL FXD0808S
Output: 8 bit signed fixed point quotient in AARGBO
8 bit signed fixed point remainder in REMBO
Result: AARG, REM <-- AARG / BARG
Max Timing: 10+77+3 = 90 clks A > 0, B > 0
11+77+8 = 96 clks A > 0, B < 0
11+77+8 = 96 clks A < 0, B > 0
12+77+3 = 92 clks A < 0, B < 0
Min Timing: 10+77+3 = 90 clks A > 0, B > 0
; 11+77+8 = 96 clks  A > 0, B < 0
; 11+77+8 = 96 clks  A < 0, B > 0
; 12+77+3 = 92 clks  A < 0, B < 0
; PM: 12+22+7 = 41  DM: 5

FXD0808S  MOVF  AARGBO,W
           XORWF  BARGBO,W
           MOVWF  SIGN
           BTFSS  BARGBO,MSB  ; if MSB set, negate BARG
           GOTO  CA0808S
           COMP  BARGBO
           INCF  BARGBO

CA0808S  BTFSS  AARGBO,MSB  ; if MSB set, negate AARG
           GOTO  C0808S
           COMP  AARGBO
           INCF  AARGBO

C0808S  CLRF  REMBO
           SDIV0808L
           BTFSS  SIGN,MSB
           RETLW  0x00
           COMP  AARGBO
           INCF  AARGBO
           COMP  REMBO
           INCF  REMBO
           RETLW  0x00

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

; 8/8 Bit Unsigned Fixed Point Divide 8/8 -> 08.08
; Input: 8 bit unsigned fixed point dividend in AARGBO
; 8 bit unsigned fixed point divisor in BARGBO
; Use: CALL FXD0808U
; Output: 8 bit unsigned fixed point quotient in AARGBO
; 8 bit unsigned fixed point remainder in REMBO
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 1+97+2 = 100 clks
; Min Timing: 1+89+2 = 92 clks
; PM: 1+13+1 = 15  DM: 4
FXD0808U  CLRF  REMBO
           UDIV0808L
           RETLW  0x00

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

; 8/7 Bit Unsigned Fixed Point Divide 8/7 -> 08.07
; Input: 8 bit unsigned fixed point dividend in AARGBO
; 7 bit unsigned fixed point divisor in BARGBO
; Use: CALL FXD0807U
; Output: 8 bit unsigned fixed point quotient in AARGBO
; 7 bit unsigned fixed point remainder in REMBO
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 1+85+2 = 88 clks
; Min Timing: 1+85+2 = 88 clks
; PM: 1+19+1 = 21  DM: 4
FXD0807U  CLRF  REMBO
           UDIV0807L
           RETLW  0x00

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

; 7/7 Bit Unsigned Fixed Point Divide 7/7 -> 07.07
; Input: 7 bit unsigned fixed point dividend in AARGBO
; 7 bit unsigned fixed point divisor in BARGBO
; Use: CALL FXD0707U
; Output: 7 bit unsigned fixed point quotient in AARGBO
; 7 bit unsigned fixed point remainder in REMBO
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 1 + 77 + 2 = 80 clks
; Min Timing: 1 + 77 + 2 = 80 clks
; PM: 1 + 22 + 1 = 44  DM: 4
FXD0707U  CLRF  REMBO
UDIV0707L
RETLW 0x00

;**********************************************************************************************
;**************************************************************************************
END
APPENDIX E: PIC17CXX MULTIPLY ROUTINES

; PIC17 FIXED POINT MULTIPLY ROUTINES VERSION 1.3
; Input: fixed point arguments in AARG and BARG
; Output: product AARG*BARG in AARG
; All timings are worst case cycle counts
; It is useful to note that the additional routines FXM3115U, FXM1515U, and
; FXM1507U can be called in a signed multiply application in the special case
; where AARG > 0 and BARG > 0, thereby offering some improvement in
; performance.

Routine   Clocks    Function
FXM0808S  53        08x08 -> 16 bit signed fixed point multiply
FXM0808U  39        08x08 -> 16 bit unsigned fixed point multiply
FXM0707U  37        07x07 -> 14 bit unsigned fixed point multiply
FXM1608S  79        16x08 -> 24 bit signed fixed point multiply
FXM1608U  75        16x08 -> 24 bit unsigned fixed point multiply
FXM1507U  69        15x07 -> 22 bit unsigned fixed point multiply
FXM1616S  175       16x16 -> 32 bit signed fixed point multiply
FXM1616U  156       16x16 -> 32 bit unsigned fixed point multiply
FXM1515U  150       15x15 -> 30 bit unsigned fixed point multiply
FXM2416S  223       24x16 -> 40 bit signed fixed point multiply
FXM2416U  203       24x16 -> 40 bit unsigned fixed point multiply
FXM2315U  194       23x15 -> 38 bit unsigned fixed point multiply
FXM2424S  346       24x24 -> 48 bit signed fixed point multiply
FXM2424U  326       24x24 -> 48 bit unsigned fixed point multiply
FXM2323U  308       23x23 -> 46 bit unsigned fixed point multiply
FXM3216S  270       32x16 -> 48 bit signed fixed point multiply
FXM3216U  265       32x16 -> 48 bit unsigned fixed point multiply
FXM3115U  254       31x15 -> 46 bit unsigned fixed point multiply
FXM3224S  417       32x24 -> 56 bit signed fixed point multiply
FXM3224U  400       32x24 -> 56 bit unsigned fixed point multiply
FXM3123U  399       31x23 -> 54 bit unsigned fixed point multiply
FXM3232S  543       32x32 -> 64 bit signed fixed point multiply
FXM3232U  536       32x32 -> 64 bit unsigned fixed point multiply
FXM3131U  543       31x31 -> 62 bit unsigned fixed point multiply
FXM3224S  417       32x24 -> 56 bit signed fixed point multiply
FXM3232U  410       32x24 -> 56 bit unsigned fixed point multiply
FXM3123U  399       31x23 -> 54 bit unsigned fixed point multiply
FXM3232S  572       32x32 -> 64 bit signed fixed point multiply
FXM3232U  572       32x32 -> 64 bit unsigned fixed point multiply
FXM3131U  543       31x31 -> 62 bit unsigned fixed point multiply

list r=dec,x=on,t=off,p=17C42
include <PIC17.INC> ; general PIC17 definitions
include <MATH17.INC> ; PIC17 math library definitions

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

; Test suite storage
RANDHI    equ    0x2B    ; random number generator registers
RANDLO    equ    0x2C
TESTCODE  equ    0x2D    ; integer code labeling test contained in following data
NUMTESTS  equ    0x2E    ; number of tests contained in following data
TESTCOUNT equ    0x30    ; beginning of test data

; Test suite for fixed point multiplication algorithms
org    0x0021
MAIN   MOVLR   RAMSTART
        MOVPF   WREG,FSR0
MEMLOOP CLRFF INDFO
          INCFSR FSR0
          GOTO   MEMLOOP
          BSF    RTCSTA,5
          MOVLF   RTCCH,WREG
          MOVLF   0x45    ; seed for random numbers
          MOVPP   WREG,RANDLO
          MOVPP   RTCCL,WREG
          MOVLF   0x30
          MOVPP   WREG,RANDHI
AN617

MOVLW 0x30
MOVPF WREG,FSRO
BCF _FS1
BSF _FS0
; CALL TFXM0808
; CALL TFXM1608
CALL TFXM1616
; CALL TFXM2416
; CALL TFXM2424
; CALL TFXM3216
; CALL TFXM3224
; CALL TFXM3232
SELF
GOTO SELF
RANDOM16
RLCF RANDHI,W
XORWF RANDHI,W
RLCF WREG
SWAPF RANDHI
SWAPF RANDLO,W
RLNCF WREG
XORWF RANDHI,W
SWAPF RANDHI
ANDLW 0x01
RLCF RANDLO
XORWF RANDLO
RLCF RANDHI
RETLW 0

; Test suite for FXM0808
TFXM0808
MOVLW 52
MOVPF WREG,TESTCOUNT
MOVPF WREG,NUMTESTS
MOVLW 1
MOVPF WREG,TESTCODE
M0808LOOP
CALL RANDOM16
MOVPF RANDHI,WREG
MOVPF WREG,BARGBO
; BCF BARGBO,MSB
; MOVPF RANDLO,WREG
; MOVPF WREG,AARGBO
; BCF AARGBO,MSB
; MOVPF AARGBO,INDFO
; MOVPF BARGBO,INDFO
CALL TFXM0808S
; CALL TFXM0808U
; CALL TFXM0707U
MOVPF AARGBO,INDFO
MOVPF AARGBO1,INDFO
DECFSZ TESTCOUNT
GOTO M0808LOOP
RETLW 0x00

; Test suite for FXM1608
TFXM1608
MOVLW 34
MOVPF WREG,TESTCOUNT
MOVPF WREG,NUMTESTS
MOVLW 2
MOVPF WREG,TESTCODE
M1608LOOP
CALL RANDOM16
MOVPF RANDHI,WREG
MOVPF WREG,BARGBO
; BCF BARGBO,MSB
CALL RANDOM16
MOVPF RANDHI,WREG
MOVPF WREG,AARGBO

© 1995 Microchip Technology Inc.

DS00617A-page 2-349
AN617

MOVF  RANDLO,WREG
MOVF  WREG,AARGB1
;
BCF  AARGB0,MSB
MOVF  AARGB0,INDF0
MOVF  AARGB1,INDF0
MOVF  BARGB0,INDF0
CALL  FXM1608S
;
CALL  FXM1608U
;
CALL  FXM1507U
MOVF  AARGB0,INDF0
MOVF  AARGB1,INDF0
MOVF  AARGB2,INDF0
DECFSZ TESTCOUNT
GOTO  M1608LOOP
RETLW 0x00

; Test suite for FXM1616
TFXM1616  MOVLW 26
MOVF  WREG,TESTCOUNT
MOVF  WREG,NUMTESTS
MOVLW 3
MOVF  WREG,TESTCODE
M1616LOOP
CALL  RANDOM16
MOVF  RANDHI,WREG
MOVF  WREG,BARGB0
MOVF  RANDLO,WREG
MOVF  WREG,BARGB1
;
BCF  BARGB0,MSB
CALL  RANDOM16
MOVF  RANDHI,WREG
MOVF  WREG,AARGB0
MOVF  RANDLO,WREG
MOVF  WREG,AARGB1
;
BCF  AARGB0,MSB
MOVF  AARGB0,INDF0
MOVF  AARGB1,INDF0
MOVF  BARGB0,INDF0
MOVF  BARGB1,INDF0
;
CALL  FXM1616S
CALL  FXM1616U
;
CALL  FXM1515U
MOVF  AARGB0,INDF0
MOVF  AARGB1,INDF0
MOVF  AARGB2,INDF0
MOVF  AARGB3,INDF0
DECFSZ TESTCOUNT
GOTO  M1616LOOP
RETLW 0x00

; Test suite for FXM2416
TFXM2416  MOVLW 20
MOVF  WREG,TESTCOUNT
MOVF  WREG,NUMTESTS
MOVLW 4
MOVF  WREG,TESTCODE
M2416LOOP
CALL  RANDOM16
MOVF  RANDHI,WREG
MOVF  WREG,BARGB0
MOVF  RANDLO,WREG
MOVF  WREG,BARGB1
;
BCF  BARGB0,MSB
CALL  RANDOM16
MOVF  RANDHI,WREG
MOVF  WREG,AARGB0
MOVF  RANDLO,WREG
MOVF  WREG,AARGB1
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, AARGB2
;
BCF AARGBO, MSB
MOVFP AARGBO, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP BARGBO, INDF0
MOVFP BARGB1, INDF0
CALL FXM2416S
;
CALL FXM2416U
;
CALL FXM2315U
MOVFP AARGBO, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP AARGB3, INDF0
MOVFP AARGB4, INDF0
DECFSZ TESTCOUNT
GOTO M2416LOOP
RETLW 0x00

Test suite for FXM2424
TFXM2424
MOVLW 17
MOVFP WREG, TESTCOUNT
MOVFP WREG, NUMTESTS
MOVLW 5
MOVFP WREG, TESTCODE

M2424LOOP
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, BARGBO
MOVFP RANDLO, WREG
MOVFP WREG, BARGB1
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, BARGB2
BCF BARGBO, MSB
MOVFP RANDLO, WREG
MOVFP WREG, AARGBO
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, AARGB1
MOVFP RANDLO, WREG
MOVFP WREG, AARGB2
BCF AARGBO, MSB
MOVFP AARGBO, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP BARGBO, INDF0
MOVFP BARGB1, INDF0
MOVFP BARGB2, INDF0
;
CALL FXM2424S
;
CALL FXM2424U
;
CALL FXM2323U
MOVFP AARGBO, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP AARGB3, INDF0
MOVFP AARGB4, INDF0
MOVFP AARGB5, INDF0
DECFSZ TESTCOUNT
GOTO M2424LOOP
RETLW 0x00

Test suite for FXM3216
TFXM3216
MOVLW 17
MOVFP WREG, TESTCOUNT

© 1995 Microchip Technology Inc.
MOVF $WREG, NUMTESTS
MOVLW 6
MOVF $WREG, TESTCODE

M3216LOOP
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
MOVF WREG, BARGBO
BCF BARGBO, MSB
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
MOVF WREG, AARGB1
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
MOVF WREG, AARGB2
BCF AARGBO, MSB
MOVF AARGBO, $WREG
MOVF AARGBl, $WREG
MOVF AARGB2, $WREG
MOVF AARGB3, $WREG
CALL FXM3216S
CALL FXM3216U
CALL FXM3115U
MOVF AARGBO, INDF0
MOVF AARGB1, INDF0
MOVF AARGB2, INDF0
MOVF AARGB3, INDF0
MOVF AARGB4, INDF0
MOVF AARGB5, INDF0
DECFSZ TESTCOUNT
GOTO M3216LOOP
RETLW $0x00

; Test suite for FXM3224
TFXM3224
MOVLW 14
MOVF $WREG, TESTCOUNT
MOVF $WREG, NUMTESTS
MOVLW 6
MOVF $WREG, TESTCODE

M3224LOOP
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
MOVF WREG, BARGBO
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
BCF BARGBO, MSB
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
MOVF WREG, AARGB1
CALL RANDOM16
MOVF RANDHI, $WREG
MOVF RANDLO, $WREG
MOVF WREG, AARGB2
MOVF WREG, AARGB3
MOVF WREG, AARGB4
MOVF WREG, AARGB5
DECFSZ TESTCOUNT
GOTO M3224LOOP
RETLW $0x00

© 1995 Microchip Technology Inc.
MOVPF WREG, AARGB3
; BCF AARGB0, MSB
MOVPF AARGB0, INDF0
MOVPF AARGB1, INDF0
MOVPF AARGB2, INDF0
MOVPF AARGB3, INDF0
MOVPF BARGB0, INDF0
MOVPF BARGB1, INDF0
MOVPF BARGB2, INDF0
CALL FXM3224S
; CALL FXM3224U
; CALL FXM3123U
MOVPF AARGB0, INDF0
MOVPF AARGB1, INDF0
MOVPF AARGB2, INDF0
MOVPF AARGB3, INDF0
MOVPF AARGB4, INDF0
MOVPF AARGB5, INDF0
MOVPF AARGB6, INDF0
DFCFSZ TESTCOUNT
GOTO M3224LOOP
RETLW Ox00
Test suite for FXM3232
TFXM3232
MOVLW 13
MOVPF WREG, TESTCOUNT
MOVPF WREG, NUMTESTS
MOVLW 7
MOVPF WREG, TESTCODE
M3232LOOP
CALL RANDOM16
MOVPF RANDHI, WREG
MOVPF WREG, BARGB0
MOVPF RANDLO, WREG
MOVPF WREG, BARGB1
CALL RANDOM16
MOVPF RANDHI, WREG
MOVPF WREG, BARGB2
MOVPF RANDLO, WREG
MOVPF WREG, BARGB3
BCF BARGB0, MSB
CALL RANDOM16
MOVPF RANDHI, WREG
MOVPF WREG, AARGB0
MOVPF RANDLO, WREG
MOVPF WREG, AARGB1
CALL RANDOM16
MOVPF RANDHI, WREG
MOVPF WREG, AARGB2
MOVPF RANDLO, WREG
MOVPF WREG, AARGB3
BCF AARGB0, MSB
MOVPF AARGB0, INDF0
MOVPF AARGB1, INDF0
MOVPF AARGB2, INDF0
MOVPF AARGB3, INDF0
MOVPF BARGB0, INDF0
MOVPF BARGB1, INDF0
MOVPF BARGB2, INDF0
MOVPF BARGB3, INDF0
; CALL FXM3232S
; CALL FXM3232U
CALL FXM3131U
MOVPF AARGB0, INDF0
MOVPF AARGB1, INDF0
MOVPF AARGB2, INDF0
AN617

MOVFP AARGB3,INDP0
MOVFP AARGB4,INDP0
MOVFP AARGB5,INDP0
MOVFP AARGB6,INDP0
MOVFP AARGB7,INDP0
DECFSZ TESTCOUNT
GOTO M3232LOOP
RETLW 0x00

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

Multiplication Macros

SMUL0808 macro
Max Timing: 4+6+6*5+3 = 43 elks
Min Timing: 4+14+3 = 21 elks
PM: 4+2*7+5+6*5+3 = 56 DM: 5
variable i
i = 0
CLRF SIGN
BTFSC AARGB0,MSB
COMF SIGN
MOVFP AARGB0,WREG
while i < 7
BTFSC BARGB0,i
GOTO SM0808NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
RETLW 0
SM0808NA0
RLCF SIGN
RRCF ACCB0
RRCF ACCB1
i = 1
while i < 7
BTFSC BARGB0,i
ADDFP ACCB0
SM0808NA#v(i)
RLCF SIGN
RRCF ACCB0
RRCF ACCB1
i = i + 1
endw
RLCF SIGN
RRCF ACCB0
RRCF ACCB1
endm

UMUL0808 macro
Max Timing: 2+5+7*4 = 35 elks
Min Timing: 2+16+3 = 21 elks
PM: 2+2*8+4+7*4 = 50 DM: 3
variable i
i = 0
BCF _C ; clear carry for first right shift
MOVFP AARGB0,WREG
while i < 8
BTFSC BARGB0,i
GOTO UM0808NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
RETLW 0
UM0808NA0
RRCF ACCB0
RRCF ACCB1
i = 1

© 1995 Microchip Technology Inc.
while \( i < 8 \)

```
BTFSC BARGBO,i
ADDWF ACCB0
UM0808NA\#v(i)
RRCF ACCB0
RRCF ACCB1
i = i + 1
endw
endm
```

**UMUL0707 macro**

; Max Timing: \(2 + 5 + 6 \times 4 + 2 = 33 \text{ clks}\)
; Min Timing: \(2 + 14 + 3 = 19 \text{ clks}\)
; PM: \(2 + 2 \times 7 + 4 + 6 \times 4 + 2 = 46 \text{ DM: } 3\)

variable \(i\)

```
i = 0
BCF _C ; clear carry for first right shift
MOVF AARGBO,WREG
while \( i < 7 \)

BTFSC BARGBO,i
GOTO UM0707NA\#v(i)
i = i + 1
endw
```

```
CLRF ACCB0 ; if we get here, BARG = 0
RETLW 0
```

**UM0707NA0**

```
RRCF ACCB0
RRCF ACCB1
i = 1
while \( i < 7 \)
BTFSC BARGBO,i
ADDWF ACCB0
UM0707NA\#v(i)
RRCF ACCB0
RRCF ACCB1
i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
```

;----------------------------------------------------------------------------

**SSMUL1608 macro**

; Max Timing: \(3 + 6 + 6 \times 9 + 3 = 66 \text{ clks}\)
; Min Timing: \(3 + 21 + 5 = 29 \text{ clks}\)
; PM: \(3 + 3 \times 7 + 7 + 6 \times 9 + 3 = 88 \text{ DM: } 6\)

variable \(i\)

```
i = 0
BTFSC AARGBO,MSB
COMF ACCB2
RLCF ACCB0,W
while \( i < 7 \)

BTFSC BARGBO,i
GOTO SSM1608NA\#v(i)
BCF ACCB2,7-i
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETLW 0
```

**SSM1608NA0**

```
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = 1
while \( i < 7 \)
BTFSS BARGBO,i
```
GOTO SSM1608NA#v(i)

SSM1608A#v(i)
MOVFP TEMPBl,WREG
ADDWF ACCBl
MOVFP TEMPBo,WREG
ADDWFCC ACCB0

SSM1608NA#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
  i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
endm

SMUL1608 macro
;  Max Timing:  7+6*10+4 = 71 clks
;  Min Timing:  7*2+4 = 18 clks
;  PM:  7*2+7+6*10+4 = 85  DM:  6
variable i
i = 0
while i < 7
  BTFSC BARGBo,i
  GOTO SSM1608NA#v(i)
  i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
RETLW 0

SM1608NA0
RLCF TEMPBo,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
  i = 1
while i < 7
  BTFSS BARGBo,i
  GOTO SSM1608NA#v(i)
SM1608A#v(i)
MOVFP TEMPBl,WREG
ADDWF ACCBl
MOVFP TEMPBo,WREG
ADDWFCC ACCB0

SM1608NA#v(i)
RLCF TEMPBo,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
  i = i + 1
endw
RLCF TEMPBo,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
endm

UMUL1608 macro
;  Max Timing:  1+6+7*9 = 70 clks
;  Min Timing:  1+8*2+4 = 21 clks
;  PM:  1+8*2+6*7*9 = 86  DM:  6
variable i
i = 0
BCF _C
while i < 8
  BTFSC BARGBo,i
  GOTO UM1608NA#v(i)
\begin{verbatim}
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
RETLW 0
UM1608NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = 1
while i < 8
BTFSS BARGBO,i
GOTO UM1608NAv(i)
UM1608A\#v(i)
MOVFP TEMPB1,WREG
ADDFP ACCB1
MOVFP TEMPB0,WREG
ADDFPC ACCB0
UM1608NA\#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = i + 1
endw
endm
UMUL1507

Max Timing: 1+6*9+3 = 64 clks
Min Timing: 1+7*2+4 = 19 clks
PM: 1+7*2+6*9+3 = 78 DM: 6

variable i
i = 0
BCF _c
while i < 7
BTFSC BARGBO,i
GOTO UM1507NAv(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
RETLW 0
UM1507NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = 1
while i < 7
BTFSS BARGBO,i
GOTO UM1507NAv(i)
UM1507A\#v(i)
MOVFP TEMPB1,WREG
ADDFP ACCB1
MOVFP TEMPB0,WREG
ADDFPC ACCB0
UM1507NA\#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
endm

SMUL1616

Max Timing: 7+7*10+7*11+5 = 159 clks
Min Timing: 15*2+4 = 34 clks
PM: 15*2+7*7+10*7*11+5 = 193 DM: 8
variable i
i = 0
while i < 15
\end{verbatim}
if i < 8
    BTFS BARGB1,i
else
    BTFS BARGB0,i-8
endif
GOTO SM1616NA#v(i)
i = i + 1
endw
CLR BARGBO
CLR BARGBl
RETLW 0
SM1616NA0
RLCF TEMPB0,W
RCF ACCB0
RCF ACCB1
RCF ACCB2
i = 1
while i < 15
    if i < 8
        BTFS BARGB1,i
    else
        BTFS BARGB0,i-8
    endif
    GOTO SM1616NA#v(i)
    MOVFB TEMPB1,WREG
    ADDWF ACCB1
    MOVFB TEMPB0,WREG
    ADDWF ACCB0
SM1616NA#v(i)
    RLC BARGB0,W
    RCF ACCB0
    RCF ACCB1
    RCF ACCB2
    if i > 7
        RCF ACCB3
    endif
    i = i + 1
endw
RLCF TEMPB0,W
RCF ACCB0
RCF ACCB1
RCF ACCB2
RCF ACCB3
endm
macro
; Max Timing: 1+6+7*9+8*10 = 150 clks
; Min Timing: 1+16*2+4 = 37 clks
; PM: 1+16*2+4+6*7*9+8*10 = 186 DM: 8
variable i
i = 0
BCF _C
while i < 16
    if i < 8
        BTFS BARGB1,i
    else
        BTFS BARGB0,i-8
    endif
    GOTO UM1616NAllv(i)
i = i + 1
endw
CLR ACCB0
CLR ACCB1
RETLW 0
UM1616NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = 1
while i < 16
  if i < 8
    BTFSS BARGBl,i
  else
    BTFSS BARGBo,i-8
  endif
  GOTO UM1616NA#v(i)
UM1616A#v(i)
  MOVFP TEMPBl,WREG
  ADDWF ACCB1
  MOVFP TEMPB0,WREG
  ADDWF ACCB0
UM1616NA#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
  if i > 7
    RRCF ACCB3
  endif
  i = i + 1
endw
UMUL1515 macro
; Max Timing: 1+6+7*9+7*10+4 = 144 clks
; Min Timing: 1+15*2+4 = 35 clks
; PM: 1+16*2+4+6+7*9+7*10+4 = 180 DM: 8
variable i
i = 0
BCF __C
while i < 15
  if i < 8
    BTFSC BARGBl,i
  else
    BTFSC BARGBo,i-8
  endif
  GOTO UM1515NA#v(i)
i = i + 1
endw
CLR F ACCB0 ; if we get here, BARG = 0
CLR F ACCB1
RSET LW 0
UM1515NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
i = 1
while i < 15
  if i < 8
    BTFSS BARGBl,i
  else
    BTFSS BARGBo,i-8
  endif
  GOTO UM1515NA#v(i)
UM1515A#v(i)
  MOVFP TEMPBl,WREG
  ADDWF ACCB1
  MOVFP TEMPB0,WREG
  ADDWF ACCB0
UM1515NA#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
  if i > 7

RRCF ACCB3
endif
i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
endm

;----------------------------------------------------------------------------
SMUL2416 macro
; Max Timing: 8+7*13+7*14+6 = 203 clks
; Min Timing: 15*2+5  = 35 clks
; PM: 15*2+9+7*13+7*14+6 = 234 DM: 10
variable i
i = 0
while i < 15

if i < 8
BTFSC BARGB1,i
else
BTFSC BARGB0,i-8
endif
GOTO SM2416#v(i)
i = i + 1
endw
CLRFB ACCB0 ; if we get here, BARG = 0
CLRFB ACCB1
CLRFB ACCB2
RETLW 0
SM2416NA0
RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
i = 1
while i < 15
if i < 8
BTFSS BARGB1,i
else
BTFSS BARGB0,i-8
endif
GOTO SM2416#v(i)
MOVFP TEMPB2,WREG
ADDFW ACCB2
MOVFP TEMPB1,WREG
ADFWFC ACCB1
MOVFP TEMPB0,WREG
ADFWFC ACCB0
SM2416#v(i)
RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
if i > 7
RRCF ACCB4
endif
i = i + 1
endw
RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
UMUL2416 macro

; Max Timing: 1+7+7*12+8*13 = 196 clks
; Min Timing: 1+16*2+5 = 38 clks
; PM: 1+16*2+5+7+7*12+8*13 = 233

variable i

i = 0
BCF _C
while i < 16

if i < 8
BTFSC BARGBl,i
else
BTFSC BARGBo,i-8
endif
GOTO UM2416NA#v(i)
i = i + 1
endw

CLRF ACCBo ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETLW 0

UM2416NA0

RRCF ACCBo
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
i = 1
while i < 16
if i < 8
BTFSS BARGBl,i
else
BTFSS BARGBo,i-8
endif
GOTO UM2416NA#v(i)

UM2416A#v(i)

MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPBl,WREG
ADDWFC ACCB1
MOVFP TEMPBo,WREG
ADDWFC ACCB0

UM2416NA#v(i)

RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
if i > 7
RRCF ACCB4
endif
i = i + 1
endw
endm

UMUL2315 macro

; Max Timing: 1+7+7*12+7*13+5 = 188 clks
; Min Timing: 1+15*2+5 = 36 clks
; PM: 1+15*2+5+7+7*2+7*13+5 = 223

variable i

i = 0
BCF _C
while i < 15

if i < 8
BTFSC BARGBl,i
else
BTFSC BARGBO,i-8
endif

GOTO UM2315NA#v(i)

i = i + 1
endw

CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3

UM2315NA0
RCCF ACCBO
RCCF ACCB1
RCCF ACCB2
RCCF ACCB3

i = 1
while i < 15
if i < 8
BTFSS BARGBO,i
else
BTFSS BARGBO,i-8
endif
GOTO UM2315NA#v(i)

UM2315A#v(i)
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCB1
MOVFP TEMPB0,WREG
ADDWF ACCB0

UM2315NA#v(i)
RCCF ACCBO
RCCF ACCB1
RCCF ACCB2
RCCF ACCB3

if i > 7
RCCF ACCB4
endif

i = i + 1
endw

RCCF ACCBO
RCCF ACCB1
RCCF ACCB2
RCCF ACCB3
RCCF ACCB4

;----------------------------------------------------------------------------
SMUL2424 macro
Max Timing: 8+7*13+8*14+7*15+7 = 323 elks
Min Timing: 23*2+5 = 51 elks
PM: 23*2+9+7*13+8*14+7*15+7 = 370 DM: 12

variable i
i = 0
while i < 23

if i < 8
BTFSC BARGBO,i
endif

if (i >= 8) && (i < 16)
BTFSC BARGBO,i-8
endif

if (i >= 16)
BTFSC BARGBO,i-16
endif
GOTO SM2424NA#v(i)
i = i + 1
endw

CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETLW 0

SM2424NA0
RLCF TEMPB0,W
RCF ACCB0
RCF ACCB1
RCF ACCB2
RCF ACCB3

i = 1
while i < 23
  if i < 8
    BTFSS BARGB2,i
  endif
  if (i >= 8) && (i < 16)
    BTFSS BARGB1,i-8
  endif
  if (i >= 16)
    BTFSS BARGB0,i-16
  endif
GOTO SM2424A#v(i)

SM2424A#v(i)
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCB1
MOVFP TEMPB0,WREG
ADDWF ACCB0

SM2424A#v(i)
RLCF TEMPB0,W
RCF ACCB0
RCF ACCB1
RCF ACCB2
RCF ACCB3
  if i > 7
    RCF ACCB4
  endif
  if i > 15
    RCF ACCB5
  endif
i = i + 1
eow
RLCF TEMPB0,W
RCF ACCB0
RCF ACCB1
RCF ACCB2
RCF ACCB3
RCF ACCB4
RCF ACCB5
eosm

UMUL2424 macro

; Max Timing: 1+7+7*12+8*13+8*14 = 308 clks
; Min Timing: 1+24*2+5 = 54 clks
; PM: 1+24*2+8+7*12+8*13+8*14 = 357 DM: 12

variable i
i = 0

BCF _C
while i < 24
  if i < 8
    BTFS ACCB1,i
  endif
  if (i >= 8) && (i < 16)
    BTFS ACCB2,i-8
  endif
  if (i >= 16)
    BTFS ACCB3,i-16
  endif

© 1995 Microchip Technology Inc.
GOTO UM2424NA#v(i)
i = i + 1
dendw
CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
RETLW 0
UN2424NA0
RRCF ACCBO
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
i = 1
while i < 24
if i < 8
BTFSS BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSS BARGB1,i-8
endif
if (i >= 16)
BTFSS BARGB0,i-16
endif
GOTO UM2424NA#v(i)
UM2424A#v(i)
MOVFP TEMPB2,WREG
ADDFW ACCB2
MOVFP TEMPB1,WREG
ADDFPC ACCB1
MOVFP TEMPB0,WREG
ADDFPC ACCB0
UM2424NA#v(i)
RRCF ACCBO
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
if i > 7
RRCF ACCB4
endif
if i > 15
RRCF ACCB5
endif
i = i + 1
dendw
dedm
UMUL2323 macro
; Max Timing: 1+7+7*12+8*13+7*14+6 = 300 clks
; Min Timing: 1+23*2+5 = 52 clks
; PM: 1+23*2+8*7*12+8*13+7*14+6 = 347 DM: 12
variable i
i = 0
BCF _C
while i < 23
if i < 8
BTFSSC BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSSC BARGB1,i-8
endif
if (i >= 16)
BTFSSC BARGB0,i-16
endif
GOTO UM2323NA#v(i)
i = i + 1
dendw
CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF  ACCB2
RETLW  0
RRCF  ACCB0
RRCF  ACCB1
RRCF  ACCB2
RRCF  ACCB3
  i = 1
  while  i < 23
    if  i < 8
      BTFSS  BARGB2,i
    endif
    if  (i >= 8) && (i < 16)
      BTFSS  BARGB1,i-8
    endif
    if  (i >= 16)
      BTFSS  BARGB0,i-16
    endif
    GOTO  UM2323NA#v(i)
  endif
  endif
  GOTO  UM2323Atv(i)
  MOVFP  TEMPB2,WREG
  ADDWF  ACCB2
  MOVFP  TEMB1,WREG
  ADDWF  ACCB1
  MOVFP  TEMPB0,WREG
  ADDWF  ACCB0
UM2323A#v(i)
  RRCF  ACCB0
  RRCF  ACCB1
  RRCF  ACCB2
  RRCF  ACCB3
  if  i > 7
    RRCF  ACCB4
  endif
  if  i > 15
    RRCF  ACCB5
  endif
  i = i + 1
endw
RRCF  ACCB0
RRCF  ACCB1
RRCF  ACCB2
RRCF  ACCB3
RRCF  ACCB4
RRCF  ACCB5
endm

;----------------------------------------------------------------------------
SMUL3216  macro
    ; Max Timing:  9+7*16+7*17+7 = 247 clks
    ; Min Timing:  15*2+6 = 36 clks
    ; PM:  15*2+11+7*16+7*17+7 = 279      DM: 13
  variable  i
  i = 0
  while  i < 15
    if  i < 8
      BTFSC  BARGB1,i
    endif
    if  (i >= 8)
      BTFSC  BARGB0,i-8
    endif
    GOTO  SM3216NA#v(i)
  endif
  endif
  i = i + 1
endw
CLRF  ACCB0
 ; if we get here, BARG = 0
CLRF  ACCB1
CLRF  ACCB2
CLRF  ACCB3
RETLW  0

© 1995 Microchip Technology Inc.

AN617
SM3216NA0

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLCF</td>
<td>TEMP0,W</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB4</td>
</tr>
<tr>
<td>i = 1</td>
<td>while i &lt; 15</td>
</tr>
<tr>
<td>if i &lt; 8</td>
<td>BTFSS BARGBl,i</td>
</tr>
<tr>
<td>endif</td>
<td></td>
</tr>
<tr>
<td>if (i &gt;= 8)</td>
<td>BTFSS BARGBO,i-8</td>
</tr>
<tr>
<td>endif</td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>SM3216NA#v(i)</td>
</tr>
</tbody>
</table>

SM3216A#v(i)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVFP</td>
<td>TEMP03,WREG</td>
</tr>
<tr>
<td>ADDWF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>MOVFP</td>
<td>TEMP02,WREG</td>
</tr>
<tr>
<td>ADDWFC</td>
<td>ACCB2</td>
</tr>
<tr>
<td>MOVFP</td>
<td>TEMP01,WREG</td>
</tr>
<tr>
<td>ADDWFC</td>
<td>ACCB1</td>
</tr>
<tr>
<td>MOVFP</td>
<td>TEMP00,WREG</td>
</tr>
<tr>
<td>ADDWFC</td>
<td>ACCB0</td>
</tr>
<tr>
<td>SM3216NA#v(i)</td>
<td>RLCF TEMP00,W</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB4</td>
</tr>
<tr>
<td>if i &gt; 7</td>
<td>RRCF ACCB5</td>
</tr>
<tr>
<td>endif</td>
<td>i = i + 1</td>
</tr>
<tr>
<td>endw</td>
<td></td>
</tr>
<tr>
<td>LRCF</td>
<td>TEMP00,W</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB1</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB2</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB3</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB4</td>
</tr>
<tr>
<td>RRCF</td>
<td>ACCB5</td>
</tr>
<tr>
<td>endm</td>
<td></td>
</tr>
</tbody>
</table>

UMUL3216

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Timing:</td>
<td>1+8+7<em>16+8</em>17 = 257 clks</td>
</tr>
<tr>
<td>Min Timing:</td>
<td>16*2+6 = 38 clks</td>
</tr>
<tr>
<td>PM:</td>
<td>1+16<em>2+10+7</em>16+8*17 = 291 DM: 13</td>
</tr>
<tr>
<td>variable i</td>
<td></td>
</tr>
<tr>
<td>i = 0</td>
<td></td>
</tr>
<tr>
<td>BCF</td>
<td>_C</td>
</tr>
<tr>
<td>while i &lt; 16</td>
<td></td>
</tr>
<tr>
<td>if i &lt; 8</td>
<td>BTFSC BARGBl,i</td>
</tr>
<tr>
<td>endif</td>
<td></td>
</tr>
<tr>
<td>if (i &gt;= 8)</td>
<td>BTFSC BARGBO,i-8</td>
</tr>
<tr>
<td>endif</td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>UM3216NA#v(i)</td>
</tr>
<tr>
<td>i = i + 1</td>
<td></td>
</tr>
<tr>
<td>endw</td>
<td></td>
</tr>
<tr>
<td>CLRF</td>
<td>ACCB0</td>
</tr>
<tr>
<td>RRLT</td>
<td>0</td>
</tr>
</tbody>
</table>

© 1995 Microchip Technology Inc.
RRCF    ACCB1
RRCF    ACCB2
RRCF    ACCB3
RRCF    ACCB4

i = 1
while i < 16
if i < 8
BTFSS    BARGB1,i
endif
if (i >= 8)
BTFSS    BARGB0,i-8
endif
GOTO    UM3216A#v(i)

UM3216A#v(i)
MOVFP    TEMPF3,WREG
ADDFC    ACCB3
MOVFP    TEMPF2,WREG
ADDFC    ACCB2
MOVFP    TEMPF1,WREG
ADDFC    ACCB1
MOVFP    TEMPO,WREG
ADDFC    ACCB0

UM3216NA#v(i)
RRCF    ACCB0
RRCF    ACCB1
RRCF    ACCB2
RRCF    ACCB3
RRCF    ACCB4
if i > 7
RRCF    ACCB5
endif
i = i + 1
endw
endm

UMUL3115

; Max Timing: 1+8+7*16+7*17+6 = 246 clks
; Min Timing: 15*2+6 = 36 clks
; PM: 1+15*2+10+7*16+7*17+6 = 278 DM: 13
variable i
i = 0
BCF    _C
while i < 15
if i < 8
BTFSC     BARGB1,i
endif
if (i >= 8)
BTFSC    BARGB0,i-8
endif
GOTO    UM3115NA#v(i)
i = i + 1
endw

CLRF    ACCB0
; if we get here, BARG = 0
CLRF    ACCB1
CLRF    ACCB2
CLRF    ACCB3
RETLW    0

UM3115NA0
RRCF    ACCB0
RRCF    ACCB1
RRCF    ACCB2
RRCF    ACCB3
RRCF    ACCB4
i = 1
while i < 15
if i < 8
BTFSS    BARGB1,i
endif
if (i >= 8)
BTFSS BARGB0,i-8
endif
GOTO UM3115NA#v(i)
UM3115A#v(i)
MOVFP TEMPB3,WREG
ADDWF ACCB3
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCB1
MOVFP TEMPB0,WREG
ADDWF ACCB0
UM3115NA#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
if i > 7
RRCF ACCB5
endif
i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
RRCF ACCB5
endm
;----------------------------------------------------------------------------
SMUL3224
macro
Max Timing: 9*7+16+8*17+7*18+8 = 391 elks
Min Timing: 23*2+6 = 52 elks
PM: 23*2+11+7*16+8*17+7*18+8 = 439 DM: 15
variable i
i = 0
while i < 23
if i < 8
BTFSS BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSS BARGB1,i-8
endif
if (i >= 16)
BTFSS BARGB0,i-16
endif
GOTO SM3224NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETLW 0
SM3224NA0
RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
i = 1
while i < 23
if i < 8
BTFSS BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSS BARGBl,i-8
endif
if (i >= 16)
BTFSS BARGBO,i-16
endif
GOTO SM3224A#v(i)

SM3224A#v(i)
MOVFP TEMPB3,WREG
ADDWF ACCB3
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCB1
MOVFP TEMPBO,WREG
ADDWF ACCB0

SM3224NA#v(i)
RLCF TEMPBO,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
if i > 7
RRCF ACCB5
endif
if i > 15
RRCF ACCB6
endif
i = i + 1
endw
RLCF TEMPBO,W
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
RRCF ACCB5
RRCF ACCB6
endm

UMUL3224
macro
Max Timing: 1+8+7*16+8*17+8*18 = 401 clks
Min Timing: 24*2+6 = 54 clks
PM: 1+24*2+10+7*16+8*17+8*18 = 451 DM: 15
variable i
i = 0
BCF _C
while i < 24
if i < 8
BTFSC BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSC BARGB1,i-8
endif
if (i >= 16)
BTFSC BARGB0,i-16
endif
GOTO UM3224NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETLW 0

UM3224NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
i = 1
while i < 24
if i < 8
BTFSS BARGB2, i
endif
if (i >= 8) && (i < 16)
BTFSS BARGB1, i-8
endif
if (i >= 16)
BTFSS BARGB0, i-16
endif
GOTO UM3224NUv(i)
UM3224NA#v(i)
MOVFP TEMPB3, WREG
ADDWF ACCB3
MOVFP TEMPB2, WREG
ADDWF ACCB2
MOVFP TEMPB1, WREG
ADDWF ACCB1
MOVFP TEMPB0, WREG
ADDWF ACCB0
UM3224NA#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
if i > 7
RRCF ACCB5
endif
if i > 15
RRCF ACCB6
endif
i = i + 1
endw
endm
UMUL3123
; Max Timing: 1+8+7*16+8*17+7*18+7 = 390 clks
; Min Timing: 23*2+6 = 52 clks
; PM: 1+23+23+10+7*16+8*17+7*18+7 = 438 DM: 15
variable i
i = 0
BCF _C
while i < 23
if i < 8
BTFSC BARGB2, i
endif
if (i >= 8) && (i < 16)
BTFSC BARGB1, i-8
endif
if (i >= 16)
BTFSC BARGB0, i-16
endif
GOTO UM3123NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETLW 0
UM3123NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
i = 1
while i < 23
if i < 8
BTFSS BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSS BARGB1,i-8
endif
if (i >= 16)
BTFSS BARGB0,i-16
endif
GOTO UM3123A#v(i)

MOVFP TEMPB3,WREG
ADDWF ACCB3
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCB1
MOVFP TEMPB0,WREG

UM3123A#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
if i > 7
RRCF ACCB5
endif
if i > 15
RRCF ACCB6
endif
i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
RRCF ACCB5
RRCF ACCB6
endm

;----------------------------------------------------------------------------
SMUL3232 macro
; Max Timing: 9+7*16+8*17+8*18+7*19+9 = 543 clks
; Min Timing: 31*2+6 = 68 clks
; PM: 31*2+11+7*16+8*17+8*18+7*19+9 = 607 DM: 16
variable i
i = 0
while i < 31
if i < 8
BTFSC BARGB3,i
endif
if (i >= 8) && (i < 16)
BTFSC BARGB2,i-8
endif
if (i >= 16) && (i < 24)
BTFSC BARGB1,i-16
endif
if (i >= 24)
BTFSC BARGB0,i-24
endif
GOTO SM3232NA#v(i)
i = i + 1
endw
CLRF ACCB0 ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETLW 0

SM3232NA0
RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
i = 1
while i < 31
if i < 8
BTFSS BARGB3,i
endif
if (i >= 8) && (i < 16)
BTFSS BARGB2,i-8
endif
if (i >= 16) && (i < 24)
BTFSS BARGBl,i-16
endif
if (i >= 24)
BTFSS BARGBO,i-24
endif
GOTO SM3232NA#v(i)

SM3232A#v(i)
MOVFP TEMPB3,WREG
ADDWF ACCB3
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCBl
MOVFP TEMPB0,WREG
ADDWF ACCB0

SM3232NA#v(i)
RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
if i > 7
RRCF ACCB5
endif
if i > 15
RRCF ACCB6
endif
if i > 23
RRCF ACCB7
endif
i = i + 1
endw

RLCF TEMPB0,W
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
RRCF ACCB5
RRCF ACCB6
RRCF ACCB7
endm

UMUL3232
; Max Timing: 1+8+7*16+8*17+8*18+8*19 = 553 elks
; Min Timing: 32*2+6 = 70 clks
; PM: 1+32*2+10+7*16+8*17+8*18+8*19 = 619
; DM: 16
variable i
i = 0
BCF _C
while i < 32

if i < 8
  BTFSC BARGB3,i
endif
if (i >= 8) && (i < 16)
  BTFSC BARGB2,i-8
endif
if (i >= 16) && (i < 24)
  BTFSC BARGB1,i-16
endif
if (i >= 24)
  BTFSC BARBO,i-24
endif
GOTO UM3232NA#v(i)
i = i + 1
endw
CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RTLW 0

UM3232NA0
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4

i = 1
while i < 32
if i < 8
  BTFSS BARGB3,i
endif
if (i >= 8) && (i < 16)
  BTFSS BARGB2,i-8
endif
if (i >= 16) && (i < 24)
  BTFSS BARGB1,i-16
endif
if (i >= 24)
  BTFSS BARBO,i-24
endif
GOTO UM3232NA#v(i)

UM3232A#v(i)
MOVFP TEMPB3,WREG
ADDF.C ACCB3
MOVFP TEMPB2,WREG
ADDF.C ACCB2
MOVFP TEMPB1,WREG
ADDF.C ACCB1
MOVFP TEMPB0,WREG
ADDF.C ACCB0

UM3232NA#v(i)
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
if i > 7
  RRCF ACCB5
endif
if i > 15
  RRCF ACCB6
endif
if i > 23
  RRCF ACCB7
endif
i = i + 1
endw
endm

UMUL3131 macro
; Max Timing: 1+8*7*16+8*17+8*18+7*19+9 = 533 clks
; Min Timing: 31*2+6 = 68 clks
; PM: 1+31*2+10*7*16+8*17+8*18+7*19+9 = 597  DM: 16

variable i
i = 0
BCF _C
while i < 31
    if i < 8
        BTFSC BARGB3,i
        endif
    if (i >= 8) && (i < 16)
        BTFSC BARGB2,i-8
        endif
    if (i >= 16) && (i < 24)
        BTFSC BARGB1,i-16
        endif
    if (i >= 24)
        BTFSC BARGB0,i-24
        endif
    GOTO UM3131NA#v(i)
    i = i + 1
endw

CLRF ACCBO ; if we get here, BARG = 0
CLRF ACCB1
CLRF ACCB2
CLRF ACCB3
RETLW 0

UM3131NA0

RCF ACCB0
RCF ACCB1
RCF ACCB2
RCF ACCB3
RCF ACCB4

i = 1
while i < 31
    if i < 8
        BTFSS BARGB3,i
        endif
    if (i >= 8) && (i < 16)
        BTFSS BARGB2,i-8
        endif
    if (i >= 16) && (i < 24)
        BTFSS BARGB1,i-16
        endif
    if (i >= 24)
        BTFSS BARGB0,i-24
        endif
    GOTO UM3131NA#v(i)

UM3131A#v(i)

MOVFP TEMPB3,WREG
ADDWF ACCB3
MOVFP TEMPB2,WREG
ADDWF ACCB2
MOVFP TEMPB1,WREG
ADDWF ACCB1
MOVFP TEMPB0,WREG
ADDWF ACCB0

UM3131NA#v(i)

RCF ACCB0
RCF ACCB1
RCF ACCB2
RCF ACCB3
RCF ACCB4

if i > 7

...
RRCF ACCB5
endif
if i > 15
RRCF ACCB6
endif
if i > 23
RRCF ACCB7
endif
i = i + 1
endw
RRCF ACCB0
RRCF ACCB1
RRCF ACCB2
RRCF ACCB3
RRCF ACCB4
RRCF ACCB5
RRCF ACCB6
RRCF ACCB7
endm

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

BxB
Bit Signed Fixed Point Multiply
OB x OB -> 16
Input:
B bit signed fixed point multiplicand in AARGBO
B bit signed fixed point multiplier in BARGBO
Use: CALL FXMOBOBS
Output: 16 bit signed fixed point product in AARGBO, AARGBl
Result: AARG <-- AARG • BARG
Max Timing: 5+43+2 = 50 elks
 Min Timing: 5+21 = 26 elks

FXMOBOBS
BTFS RARGBO,MSB
GOTO M0808SOK
COMF BARGBO
INCF BARGBO
COMF AARGBO
INCF AARGBO
M0808SOK
CLRF ACCB1
MOVFF AARGBO,TEMPBO
SMUL0808
RETLW 0x00

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

BxB
Bit Unsigned Fixed Point Multiply
OB x OB -> 16
Input:
B bit unsigned fixed point multiplicand in AARGBO
B bit unsigned fixed point multiplier in BARGBO
Use: CALL FXMOBOBU
Output: 16 bit unsigned fixed point product in AARGBO, AARGBl
Result: AARG <-- AARG * BARG
Max Timing: 1+35+2 = 39 elks
 Min Timing: 2+21 = 23 elks

FXMOBOBU
CLRF ACCBl
MOVFF AARGBO,TEMPBO
UMUL0808
RETLW 0x00

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

7x7 Bit Unsigned Fixed Point Multiply 07 x 07 -> 14
Input: 7 bit unsigned fixed point multiplicand in AARGBO
7 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM0707U
Output: 14 bit unsigned fixed point product in AARGBO, AARGBl

;**********************************************************************************************
;**********************************************************************************************
Result: AARG <-- AARG * BARG
Max Timing: 2+33+2 = 37 clks
Min Timing: 2+19 = 21 clks
PM: 2+46+1 = 49

FXM0707U
CLRF ACCB1
MOVFP AARGBO,TEMPB0
UNUL0707
RETLW 0x00

16x8 Bit Signed Fixed Point Multiply 16 x 08 -> 24
Input: 16 bit signed fixed point multiplicand in AARGBO
8 bit signed fixed point multiplier in BARGBO
Use: CALL FXM1608S
Output: 24 bit signed fixed point product in AARGBO, AARGBl
Result: AARG <-- AARG * BARG
Max Timing: 6+66+2 = 74 clks B > 0
11+66+2 = 79 clks B < 0
Min Timing: 6+29 = 35 clks
PM: 11+88+1 = 100

FXM1608S
BTFSS BARGBO,MSB
GOTO M1608SOK
ICMP BARGBO
ICMP AARGBO
ICMP AARGB1
INFSNZ AARGB1
ICMP AARGB0
M1608SOK
CLRF ACCB2
MOVFP AARGBO,TEMPB0
MOVFP AARGB1,TEMPB1
SSMUL1608
RETLW 0x00

16x8 Bit Unsigned Fixed Point Multiply 16 x 08 -> 24
Input: 16 bit unsigned fixed point multiplicand in AARGBO
8 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM1608U
Output: 24 bit unsigned fixed point product in AARGBO, AARGBl, AARGB2
Result: AARG <-- AARG * BARG
Max Timing: 3+70+2 = 75 clks
Min Timing: 3+21 = 24 clks
PM: 3+86+1 = 90

FXM1608U
CLRF ACCB2
MOVFP AARGBO,TEMPB0
MOVFP AARGB1,TEMPB1
UNUL1608
RETLW 0x00

15x7 Bit Unsigned Fixed Point Multiply 15 x 07 -> 22
Input: 15 bit unsigned fixed point multiplicand in AARGBO, AARGBl
7 bit unsigned fixed point multiplier in BARGBO
Use: CALL FXM1507U
Output: 22 bit unsigned fixed point product in AARGBO, AARGB1, AARGB2
Result: AARG <-- AARG * BARG
Max Timing: 3+64+2 = 69 clks
Min Timing: 3+21 = 24 clks
PM: 3+78+1 = 82

FXM1507U
CLRF ACCB2
MOVFP AARGBO,TEMPB0
MOVFP AARGB1,TEMPB1

© 1995 Microchip Technology Inc.
UMUL1507
RETLW 0x00

;*******************************************************************************
;*******************************************************************************
16x16 Bit Signed Fixed Point Multiply 16 x 16 -> 32
Input: 16 bit signed fixed point multiplicand in AARGBO, AARGBl
16 bit signed fixed point multiplier in BARGBO, BARGBl
Use: CALL FXM1616S
Output: 32 bit signed fixed point product in AARGBO, AARGBl, AARGB2, AARGB3
Result: AARG <- AARG * BARG
Max Timing: 7+159+2 = 168 clks B > 0
14+159+2 = 175 clks B < 0
Min Timing: 6+34 = 24 clks
PM: 11+193+1 = 97 DM: 8

FXM1616S
BTFSS BARGBO,MSB
GOTO M1616SOK
COMF BARGBO
COMF BARGBl
INFSNZ BARGBO
INCF BARGBl
COMF AARGBO
COMF AARGBl
INFSNZ AARGBO
INCF AARGBl
M1616SOK
CLRF ACCB2 ; clear partial product
CLRF ACCB3
MOVPF AARGBO,TEMPBO
MOVPF AARGBl,TEMPBl
UMUL1616
RETLW 0x00

;*******************************************************************************
;*******************************************************************************
16x16 Bit Unsigned Fixed Point Multiply 16 x 16 -> 32
Input: 16 bit unsigned fixed point multiplicand in AARGBO, AARGBl
16 bit unsigned fixed point multiplier in BARGBO, BARGBl
Use: CALL FXM1616U
Output: 32 bit unsigned fixed point product in AARGBO, AARGBl, AARGB2, AARGB3
Result: AARG <- AARG * BARG
Max Timing: 4+150+2 = 156 clks
Min Timing: 4+37 = 41 clks
PM: 4+186+1 = 191 DM: 8

FXM1616U
CLRF ACCB2
CLRF ACCB3
MOVPF AARGBO,TEMPBO
MOVPF AARGBl,TEMPBl
UMUL1616
RETLW 0x00

;*******************************************************************************
;*******************************************************************************
15x15 Bit Unsigned Fixed Point Multiply 15 x 15 -> 30
Input: 15 bit unsigned fixed point multiplicand in AARGBO, AARGBl
15 bit unsigned fixed point multiplier in BARGBO, BARGBl
Use: CALL FXM1515U
Output: 30 bit unsigned fixed point product in AARGBO, AARGBl, AARGB2, AARGB3
Result: AARG <- AARG * BARG
Max Timing: 4+144+2 = 150 clks
Min Timing: 4+35 = 39 clks
PM: 4+180+1 = 185 DM: 8

FXM1515U
CLRF ACCB2
CLRF ACCB3
AN617

FXM2416S

BTFSS BARGBO,MSB
GOTO M2416SOK

; make multiplier BARG > 0

M2416SOK

CLRF ACCB3 ; clear partial product

CLRF ACCB4

MOVPF AARGBO,TEMPB0
MOVPF AARGBl,TEMPBl

SMUL2416

RETLW 0x00

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

FXM2416U

CLRF ACCB3

CLRF ACCB4

MOVPF AARGBO,TEMPB0
MOVPF AARGBl,TEMPBl

UMUL2416

RETLW 0x00

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

FXM2416U

CLRF ACCB3

CLRF ACCB4

MOVPF AARGBO,TEMPB0
MOVPF AARGBl,TEMPBl

UMUL2416

RETLW 0x00

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

DS00617A-page 2-378 © 1995 Microchip Technology Inc.
15 bit unsigned fixed point multiplier in BARGBO, BARGBl

Use: CALL FXM2315U

Output: 38 bit unsigned fixed point product in AARGBO, AARGBl,
AARGB2, AARGB3, AARGB4

Result: AARG <-- AARG * BARG

Max Timing: 4+188+2 = 194 clks

Min Timing: 5+36 = 41 clks

PM: 5+223+1 = 229

DM: 10

FXM2315U
CLRF ACCB3
CLRF ACCB4
MOVVF AARGB0,TEMPB0
MOVVF AARGB1,TEMPB1
MOVVF AARGB2,TEMPB2
UMUL2315
RETLW Ox00

FXM2424S

24x24 Bit Signed Fixed Point Multiply 24 x 24 -> 48

Input: 24 bit signed fixed point multiplicand in AARGBO, AARGBl,
AARGB2

24 bit signed fixed point multiplier in BARGBO, BARGBl,
BARGB2

Use: CALL FXM2424S

Output: 48 bit signed fixed point product in AARGBO, AARGBl,
AARGB2, AARGB3, AARGB4, AARGB5

Result: AARG <-- AARG * BARG

Max Timing: 9+323+2 = 334 clks

B > 0

21+323+2 = 346 clks

B < 0

Min Timing: 9+51 = 60 clks

PM: 21+370+1 = 392

DM: 12

FXM2424S
BTFSS BARGBO,MSB
GOTO M2424SOK
CLRF WREG
COMF BARGBO
COMF BARGBl
INCF BARGB2
ADDWF BARGB2
ADDWF BARGBO
COMF AARGB0
COMF AARGB1
INCF AARGB2
ADDWF AARGB2
ADDWF AARGB0

M2424SOK
CLRF ACCB3
CLRF ACCB4
CLRF ACCB5
MOVVF AARGB0,TEMPB0
MOVVF AARGB1,TEMPB1
MOVVF AARGB2,TEMPB2
SMUL2424
RETLW Ox00

FXM2424U

24x24 Bit Unsigned Fixed Point Multiply 24 x 24 -> 48

Input: 24 bit unsigned fixed point multiplicand in AARGBO, AARGBl,
AARGB2

24 bit unsigned fixed point multiplier in BARGBO, BARGBl,
BARGB2

Use: CALL FXM2424U

Output: 48 bit unsigned fixed point product in AARGBO, AARGBl,
AARGB2, AARGB3, AARGB4, AARGB5

Result: AARG <-- AARG * BARG
AN617

; Max Timing: 6+308+2 = 316 clks
; Min Timing: 6+54 = 60 clks
; PM: 6+357+1 = 364 DM: 12

FXM2424U
CLRFB  ACCB3
CLRFB  ACCB4
CLRFB  ACCB5
MOVPF  AARGB0,TEMPB0
MOVPF  AARGB1,TEMPB1
MOVPF  AARGB2,TEMPB2
UMUL2424
RETLW  0x00

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

FXM3216S
BINPS    BARGBO,MSB
GOTO    M3216SOK
CLRFB    WREG
COMFB    BARGB0
COMFB    BARGB1
INCFB    BARGB2
ADDF3FC  BARGB0
COMBF    AARGB0
COMBF    AARGB1
COMBF    AARGB2
COMBF    AARGB3
INCBF    AARGB3
ADDF3FC  AARGB2
ADDF3FC  AARGB1
ADDF3FC  AARGB0
M3216SOK CLRFB  ACCB4
CLRFB ACCB5

; 23x23 Bit Unsigned Fixed Point Multiply 23 x 23 -> 46
; Input: 23 bit unsigned fixed point multiplicand in AARGBO, AARGB1, AARGB2
; 23 bit unsigned fixed point multiplier in BARGBO, BARGB1, BARGB2
; Use: CALL FXM2323U
; Output: 46 bit unsigned fixed point product in AARGBO, AARGB1, AARGB2, AARGB3, AARGB4, AARGB5
; Result: AARG <-- AARG * BARG
; Max Timing: 6+308+2 = 316 clks
; Min Timing: 6+54 = 60 clks
; PM: 6+357+1 = 364 DM: 12

FXM2323U
CLRFB  ACCB3
CLRFB  ACCB4
CLRFB  ACCB5
MOVPF  AARGB0,TEMPB0
MOVPF  AARGB1,TEMPB1
MOVPF  AARGB2,TEMPB2
UMUL2323
RETLW  0x00

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

FXM3216S
BINPS    BARGBO,MSB
GOTO    M3216SOK
CLRFB    WREG
COMFB    BARGB0
COMFB    BARGB1
INCFB    BARGB2
ADDF3FC  BARGB0
COMBF    AARGB0
COMBF    AARGB1
COMBF    AARGB2
COMBF    AARGB3
INCBF    AARGB3
ADDF3FC  AARGB2
ADDF3FC  AARGB1
ADDF3FC  AARGB0
M3216SOK CLRFB  ACCB4
CLRFB ACCB5

; 32x16 Bit Signed Fixed Point Multiply 32 x 16 -> 48
; Input: 32 bit signed fixed point multiplicand in AARGBO, AARGB1, AARGB2, AARGB3
; 16 bit signed fixed point multiplier in BARGBO, BARGB1
; Use: CALL FXM3216S
; Output: 48 bit signed fixed point product in AARGBO, AARGB1, AARGB2, AARGB3, AARGB4, AARGB5
; Result: AARG <-- AARG * BARG
; Max Timing: 9+247+2 = 258 clks B > 0
; 21+247+2 = 270 clks B < 0
; Min Timing: 10+36 = 46 clks
; PM: 21+279+1 = 301 DM: 13

FXM3216S
BINPS    AARGBO, temp
GOTO    M3216SOK
CLRFB    WRUB0
COMFB    BARGB0
COMFB    BARGB1
INCFB    BARGB2
ADDF3FC  BARGB0
COMBF    AARGB0
COMBF    AARGB1
COMBF    AARGB2
COMBF    AARGB3
INCBF    AARGB3
ADDF3FC  AARGB2
ADDF3FC  AARGB1
ADDF3FC  AARGB0
M3216SOK CLRFB  ACCB4
CLRFB ACCB5

; clear partial product
; 32x16 Bit Unsigned Fixed Point Multiply 32 x 16 -> 48
; Input: 32 bit unsigned fixed point multiplicand in AARGBO, AARGBl,
; AARGB2, AARGB3
; 16 bit unsigned fixed point multiplier in BARGB0, BARGB1
; Use: CALL FXM3216U
; Output: 48 bit unsigned fixed point product in AARGBO, AARGBl,
; AARGB2, AARGB3, AARGB4, AARGB5
; Result: AARG <-- AARG * BARG
; Max Timing: 6+257+2 = 265 clks
; Min Timing: 6+38 = 44 clks
; PM: 6+291+1 = 298
; DM: 13
FXM3216U
CLRF ACCB4
CLRF ACCB5
MOVPF AARGB0,TEMPB0
MOVPF AARGB1,TEMPB1
MOVPF AARGB2,TEMPB2
MOVPF AARGB3,TEMPB3
UMUL3216
RETLW 0x00

; 31x15 Bit Unsigned Fixed Point Multiply 31 x 15 -> 46
; Input: 31 bit unsigned fixed point multiplicand in AARGBO, AARGBl,
; AARGB2, AARGB3
; 15 bit unsigned fixed point multiplier in BARGB0, BARGB1
; Use: CALL FXM3115U
; Output: 46 bit unsigned fixed point product in AARGBO, AARGBl,
; AARGB2, AARGB3, AARGB4, AARGB5, AARGB6
; Result: AARG <-- AARG * BARG
; Max Timing: 6+246+2 = 254 clks
; Min Timing: 6+36 = 42 clks
; PM: 6+278+1 = 285
; DM: 13
FXM3115U
CLRF ACCB4
CLRF ACCB5
MOVPF AARGB0,TEMPB0
MOVPF AARGB1,TEMPB1
MOVPF AARGB2,TEMPB2
MOVPF AARGB3,TEMPB3
UMUL3115
RETLW 0x00

; 32x24 Bit Signed Fixed Point Multiply 32 x 24 -> 56
; Input: 32 bit signed fixed point multiplicand in AARGBO, AARGBl,
; AARGB2, AARGB3
; 24 bit signed fixed point multiplier in BARGB0, BARGB1,
; BARGB2
; Use: CALL FXM3224S
; Output: 56 bit signed fixed point product in AARGBO, AARGBl,
; AARGB2, AARGB3, AARGB4, AARGB5, AARGB6
; Result: AARG <-- AARG * BARG
; Max Timing: 10+391+2 = 403 clks
; Min Timing: 10+52 = 62 clks
; PM: 24+439+1 = 464
; DM: 15
FXM3224S
CLRF ACCB4
CLRF ACCB5
MOVPF AARGB0,TEMPB0
MOVPF AARGB1,TEMPB1
MOVPF AARGB2,TEMPB2
MOVPF AARGB3,TEMPB3
UMUL3224
RETLW 0x00

;******************************************************************************************
;******************************************************************************************
;******************************************************************************************
;******************************************************************************************
AN617

FXM3224S  BTFSS  BARGBO,MSB
  GOTO  M3224SOK
  CLRFB  WREG
  COMFB  BARGBO ; make multiplier BARG > 0
  COMFB  BARGB1
  COMFB  BARGB2
  INCFB  BARGB2
  ADWFPC  BARGB1
  ADWFPC  BARGB0
  COMFB  AARB0
  COMFB  AARB1
  COMFB  AARB2
  COMFB  AARB3
  INCFB  AARB3
  ADWFPC  AARB2
  ADWFPC  AARB1
  ADWFPC  AARB0
M3224SOK  CLRFB  ACCB4 ; clear partial product
  CLRFB  ACCB5
  CLRFB  ACCB6
  MOVFB  AARBG0,TEMPB0
  MOVFB  AARBG1,TEMPB1
  MOVFB  AARBG2,TEMPB2
  MOVFB  AARBG3,TEMPB3
  SMUL3224
  RETLW 0x00

;**********************************************************************************************
;**********************************************************************************************
; 32x24 Bit Unsigned Fixed Point Multiply 32 x 24 -> 56
; Input: 32 bit unsigned fixed point multiplicand in AARBG0, AARBG1, AARBG2, AARBG3
; 24 bit unsigned fixed point multiplier in BARGBO, BARGB1, BARGB2
; Use: CALL FXM3224U
; Output: 56 bit unsigned fixed point product in AARBG0, AARBG1, AARBG2, AARBG3, AARBG4, AARBG5, AARBG6
; Result: AARG <-- AARB * BARG
; Max Timing: 7+401+2 = 410 clks
; Min Timing: 7+54 = 61 clks
; PM: 7+451+1 = 459 DM: 15
FXM3224U  CLRFB  ACCB4
  CLRFB  ACCB5
  CLRFB  ACCB6
  MOVFB  AARBG0,TEMPB0
  MOVFB  AARBG1,TEMPB1
  MOVFB  AARBG2,TEMPB2
  MOVFB  AARBG3,TEMPB3
  UMUL3224
  RETLW 0x00

;**********************************************************************************************
;**********************************************************************************************
; 31x23 Bit Unsigned Fixed Point Multiply 31 x 23 -> 54
; Input: 31 bit unsigned fixed point multiplicand in AARBG0, AARBG1, AARBG2, AARBG3
; 23 bit unsigned fixed point multiplier in BARGBO, BARGB1, BARGB2
; Use: CALL FXM3123U
; Output: 54 bit unsigned fixed point product in AARBG0, AARBG1, AARBG2, AARBG3, AARBG4, AARBG5, AARBG6
; Result: AARG <-- AARB * BARG
; Max Timing: 7+390+2 = 399 clks
; Min Timing: 7+52 = 59 clks
; PM: 7+438+1 = 446 DM: 15
FXM3123U  CLRFB  ACCB4

DS00617A-page 2-382  © 1995 Microchip Technology Inc.
AN617

CLRF ACCB5
CLRF ACCB6
MOVPP AARGB0,TEMPB0
MOVPP AARGB1,TEMPB1
MOVPP AARGB2,TEMPB2
MOVPP AARGB3,TEMPB3
UMUL3123
RETLW 0x00

32x32 Bit Signed Fixed Point Multiply 32 x 32 -> 64
Input: 32 bit signed fixed point multiplicand in AARGB0, AARGB1,
AARGB2, AARGB3
32 bit signed fixed point multiplier in BARGB0, BARGB1,
BARGB2, BARGB3
Use: CALL FXM3232S
Output: 64 bit signed fixed point product in AARGB0, AARGB1,
AARGB2, AARGB3, AARGB4, AARGB5, AARGB6, AARGB7
Result: AARG
Max Timing: 11+543+2 = 556 clks
Min Timing: 11+68 = 79 clks
PM: 27+607+1 = 635
DM: 16
FXM3232S BTFSS BARGBO,MSB
GOTO M3232SOK
CLRF WREG
COMP BARG0
COMP BARG1
COMP BARG2
COMP BARG3
INCF BARG3
ADDWF BARG0
ADDWF BARG1
ADDWF BARG2
ADDWF BARG3
ADDWF AARGB0
ADDWF AARGB1
ADDWF AARGB2
ADDWF AARGB3
M3232SOK CLRF ACCB4
CLRF ACCB5
CLRF ACCB6
CLRF ACCB7
MOVPP AARGB0,TEMPB0
MOVPP AARGB1,TEMPB1
MOVPP AARGB2,TEMPB2
MOVPP AARGB3,TEMPB3
SMUL3232
RETLW 0x00

32x32 Bit Unsigned Fixed Point Multiply 32 x 32 -> 64
Input: 32 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
AARGB2, AARGB3
32 bit unsigned fixed point multiplier in BARGB0, BARGB1,
BARGB2, BARGB3
Use: CALL FXM3232U
Output: 64 bit unsigned fixed point product in AARGB0, AARGB1,
AARGB2, AARGB3, AARGB4, AARGB5, AARGB6, AARGB7
Result: AARG
Max Timing: 8+553+2 = 563 clks
; Min Timing:  8+70 = 78 clks
; PM:  8+619+1 = 628  DM: 16
FXM3232U  CLRF    ACCB4
        CLRF    ACCB5
        CLRF    ACCB6
        CLRF    ACCB7
        MOVPF  AARGBO,TEMPB0
        MOVPF  AARGB1,TEMPB1
        MOVPF  AARGB2,TEMPB2
        MOVPF  AARGB3,TEMPB3
        UMUL3232
        RETLW   0x00

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

3lx3l Bit Unsigned Fixed Point Multiply 31 x 31 -> 62
; Input:  31 bit unsigned fixed point multiplicand in AARGBO, AARGBl, AARGB2, AARGB3
; 31 bit unsigned fixed point multiplier in BARGBO, BARGBl, BARGB2, BARGB3
; Use:    CALL    FXM3131U
; Output: 62 bit unsigned fixed point product in AARGBO, AARGB1, AARGB2, AARGB3, AARGB4, AARGB5, AARGB6, AARGB7
; Result: AARG  <-- AARG * BARG
; Max Timing:  8+533+2 = 543 clks
; Min Timing:  8+68 = 76 clks
; PM:  8+597+1 = 606  DM: 16
FXM3131U  CLRF    ACCB4
        CLRF    ACCB5
        CLRF    ACCB6
        CLRF    ACCB7
        MOVPF  AARGBO,TEMPB0
        MOVPF  AARGB1,TEMPB1
        MOVPF  AARGB2,TEMPB2
        MOVPF  AARGB3,TEMPB3
        UMUL3131
        RETLW   0x00

;******************************************************************************************
;******************************************************************************************
END
APPENDIX F: PIC17CXX DIVIDE ROUTINES

Table of Contents for Appendix F

F.1 PIC17CXX Fixed Point Divide Routines A ................................................................. 2-386
F.2 PIC17CXX Fixed Point Divide Routines B ...................................................................... 2-421
F.3 PIC17CXX Fixed Point Divide Routines C ...................................................................... 2-450

F.1 PIC17CXX Fixed Point Divide Routines A

Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two argument can be called in a signed divide application where it is known that the respective argument is nonnegative, thereby offering some improvement in performance.

Routine | Clocks | Function
--- | --- | ---
FXD3232S | 614 | 32 bit/32 bit → 32.32 signed fixed point divide
FXD3232U | 683 | 32 bit/32 bit → 32.32 unsigned fixed point divide
FXD3231U | 588 | 32 bit/31 bit → 32.31 unsigned fixed point divide
FXD3131U | 579 | 31 bit/31 bit → 31.31 unsigned fixed point divide
FXD3224S | 514 | 32 bit/24 bit → 32.24 signed fixed point divide
FXD3224U | 584 | 32 bit/24 bit → 32.24 unsigned fixed point divide
FXD3223U | 489 | 32 bit/23 bit → 32.23 unsigned fixed point divide
FXD3123U | 481 | 31 bit/23 bit → 31.23 unsigned fixed point divide

list r=dec,x=on,t=off,p=17C42
include <PIC17.INC> general PIC17 definitions
include <MATH17.INC> PIC17 math library definitions

Test suite storage
RANDHI | equ 0x28 | random number generator registers
RANDLO | equ 0x2C |
TESTCODE | equ 0x2D | integer code labeling test contained in following data
NUMTESTS | equ 0x2E | number of tests contained in following data
TESTCOUNT | equ 0x2F | counter
DATA | equ 0x30 | beginning of test data

Test suite for fixed point divide algorithms
org 0x0021
MAIN | MOVLN RAMSTART |
MOVPF WREG,FSR0
MEMLOOP | CLR PF |
INCFSZ FSR0 |
GOTO MEMLOOP |
BSF RTCSTA, 5 |
MOVPF RTCCH, WREG |
MOVLW 0x45 ; seed for random numbers
MOVPF WREG, RANDLO |
MOVPF WREG, RANDHI |
MOVWL 0x30 |
MOVF WREG, FSR0 |
BCF _FS1 |
BSF _FS0 |
CALL TFXD3232 |
CALL TFXD3224 |
SELF |
GOTO SELF |
RANDOM16 | RLCF RANDHI, W | random number generator

© 1995 Microchip Technology Inc.
XORWF  RANDHI,W
RLCF  WREG
SWAPF  RANDHI
SWAPF  RANDLO,W
RLNCF  WREG
XORWF  RANDHI,W
SWAPF  RANDHI
ANDLW  0x01
RLCF  RANDLO
XORWF  RANDLO
RLCF  RANDHI

RETLW  0

; Test suite for FXD3232
TFXD3232
MOVLW  13
MOVPF  WREG,TESTCOUNT
MOVPF  WREG,NUMTESTS
MOVLW  5
MOVPF  WREG,TESTCODE

D3232LOOP
CALL  RANDOM16
MOVPF  RANDHI,WREG
MOVPF  WREG,BARGB0
MOVPF  RANDLO,WREG
MOVPF  WREG,BARGB1
CALL  RANDOM16
MOVPF  RANDHI,WREG
MOVPF  WREG,BARGB2
MOVPF  RANDLO,WREG
MOVPF  WREG,BARGB3
BCF  BARGB0,MSB
MOVPF  BARGB0,INDFO
MOVPF  BARGB1,INDFO
MOVPF  BARGB2,INDFO
MOVPF  BARGB3,INDFO
CALL  RANDOM16
MOVPF  RANDHI,WREG
MOVPF  WREG,AARGB0
MOVPF  RANDLO,WREG
MOVPF  WREG,AARGB1
CALL  RANDOM16
MOVPF  RANDHI,WREG
MOVPF  WREG,AARGB2
MOVPF  RANDLO,WREG
MOVPF  WREG,AARGB3

; BCF  AARGB0,MSB
MOVPF  AARGB0,INDFO
MOVPF  AARGB1,INDFO
MOVPF  AARGB2,INDFO
MOVPF  AARGB3,INDFO
; CALL  FXD3232S
CALL  FXD3232U
; CALL  FXD3231U
; CALL  FXD3131U
MOVPF  AARGB0,INDFO
MOVPF  AARGB1,INDFO
MOVPF  AARGB2,INDFO
MOVPF  AARGB3,INDFO
MOVPF  REMB0,INDFO
MOVPF  REMB1,INDFO
MOVPF  REMB2,INDFO
MOVPF  REMB3,INDFO
DECFSZ  TESTCOUNT
GOTO  D3233LOOP
RETLW  0x00
; Test suite for FXD3224
TFXD3224
MOVLW 14
MOVF WREG,TESTCOUNT
MOVF WREG,NUMTESTS
MOVLW 5
MOVF WREG,TESTCODE
D3224LOOP
CALL RANDOM16
MOVF RANDHI,WREG
MOVF RANDLO,WREG
MOVF WREG,BARGB1
CALL RANDOM16
MOVF RANDHI,WREG
MOVF WREG,BARGB2
BCF BARGB0,MSB
MOVF BARGB0,INDFO
MOVF BARGB1,INDFO
MOVF BARGB2,INDFO
CALL FXD3224S
CALL FXD3224U
CALL FXD3123U
MOVF AARGB0,INDFO
MOVF AARGB1,INDFO
MOVF AARGB2,INDFO
MOVF AARGB3,INDFO
CALL FXD3224S
CALL FXD3224U
CALL FXD3123U
MOVF AARGB0,INDFO
MOVF AARGB1,INDFO
MOVF AARGB2,INDFO
MOVF AARGB3,INDFO
MOVFP AARGB0,INDFO
MOVFP AARGB1,INDFO
MOVFP AARGB2,INDFO
MOVFP AARGB3,INDFO
MOVFP REMBO,INDFO
MOVFP REMBl,INDFO
MOVFP REMB2,INDFO
DECFSZ TESTCOUNT
GOTO D3224LOOP
RETLW 0x00

;**********************************************************************************************/
;**********************************************************************************************/

32/32 Bit Division Macros
SDIV3232 macro
; Max Timing: 9+14+30*18+10 = 573 clks
; Min Timing: 9+14+30*17+3 536 clks
; PM: 9+14+30*24+10 = 753 DM: 12
variable i
MOVF BARGB0,WREG
SUBWF REMB3
MOVF BARGB2,WREG
SUBWFB REMB2
MOVF BARGB1,WREG
SUBWFB REMB1
MOVF BARGB0,WREG
SUBWFB REMB0
RLCF ACCB0
RLCF   ACCB0,W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
MOVFP  BARGB3,WREG
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC REMB2
MOVFP  BARGB1,WREG
ADDWFC REMB1
MOVFP  BARGB0,WREG
ADDWFC REMB0
RLCF   ACCB0
i = 2
while i < 8
RLCF   ACCB0,W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
MOVFP  BARGB3,WREG
BTFSS  ACCB0,LSB
GOTO   SADD22#v(i)
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC REMB2
MOVFP  BARGB1,WREG
ADDWFC REMB1
MOVFP  BARGB0,WREG
ADDWFC REMB0
GOTO   SOK22#v(i)
SADD22#v(i)
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC REMB2
MOVFP  BARGB1,WREG
ADDWFC REMB1
MOVFP  BARGB0,WREG
ADDWFC REMB0
GOTO   SOK22#v(i)
SOK22#v(i)
i = i + 1
diw
RLCF   ACCB1,W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
MOVFP  BARGB3,WREG
BTFSS  ACCB0,LSB
GOTO   SADD228
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC REMB2
MOVFP  BARGB1,WREG
ADDWFC REMB1
MOVFP  BARGB0,WREG
ADDWFC REMB0
GOTO   SOK228
SADD228
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC REMB2
MOVFP  BARGB1,WREG
ADDWFC REMB1
MOVFP  BARGB0,WREG
ADDWFC REMB0
SOK228
RLCF   ACCB1
\[ i = 9 \]
\[ \text{while } i < 16 \]
RLCF \[ \text{ACCB1, W} \]
RLCF \[ \text{REMB3} \]
RLCF \[ \text{REMB2} \]
RLCF \[ \text{REMB1} \]
RLCF \[ \text{REMB0} \]
MOVFP \[ \text{BARGB3, WREG} \]
BTFSS \[ \text{ACCB1, LSB} \]
GOTO \[ \text{SADD22#v(i)} \]
SUBWF \[ \text{REMB3} \]
MOVFP \[ \text{BARGB2, WREG} \]
SUBWF \[ \text{REMB2} \]
MOVFP \[ \text{BARGB1, WREG} \]
SUBWF \[ \text{REMB1} \]
MOVFP \[ \text{BARGB0, WREG} \]
SUBWF \[ \text{REMB0} \]
GOTO \[ \text{SOK2211v(i)} \]
SADD22#v(i) \[ \text{ADDFP REMB3} \]
MOVFP \[ \text{BARGB2, WREG} \]
ADDFC \[ \text{REMB2} \]
MOVFP \[ \text{BARGB1, WREG} \]
ADDFC \[ \text{REMB1} \]
MOVFP \[ \text{BARGB0, WREG} \]
ADDFC \[ \text{REMB0} \]
SOK22#v(i) \[ \text{RLCF ACCB1} \]
i = i + 1
endw
RLCF \[ \text{ACCB2, W} \]
RLCF \[ \text{REMB3} \]
RLCF \[ \text{REMB2} \]
RLCF \[ \text{REMB1} \]
RLCF \[ \text{REMB0} \]
MOVFP \[ \text{BARGB3, WREG} \]
BTFSS \[ \text{ACCB1, LSB} \]
GOTO \[ \text{SADD2216} \]
SUBWF \[ \text{REMB3} \]
MOVFP \[ \text{BARGB2, WREG} \]
SUBWF \[ \text{REMB2} \]
MOVFP \[ \text{BARGB1, WREG} \]
SUBWF \[ \text{REMB1} \]
MOVFP \[ \text{BARGB0, WREG} \]
SUBWF \[ \text{REMB0} \]
GOTO \[ \text{SOK2216} \]
SADD2216 \[ \text{ADDFP REMB3} \]
MOVFP \[ \text{BARGB2, WREG} \]
ADDFC \[ \text{REMB2} \]
MOVFP \[ \text{BARGB1, WREG} \]
ADDFC \[ \text{REMB1} \]
MOVFP \[ \text{BARGB0, WREG} \]
ADDFC \[ \text{REMB0} \]
SOK2216 \[ \text{RLCF ACCB2} \]
i = 17
\[ \text{while } i < 24 \]
RLCF \[ \text{ACCB2, W} \]
RLCF \[ \text{REMB3} \]
RLCF \[ \text{REMB2} \]
RLCF \[ \text{REMB1} \]
RLCF \[ \text{REMB0} \]
MOVFP \[ \text{BARGB3, WREG} \]
BTFSS \[ \text{ACCB2, LSB} \]
GOTO \[ \text{SADD222#v(i)} \]
SUBWF \[ \text{REMB3} \]
MOVFP \[ \text{BARGB2, WREG} \]
SUBWF \[ \text{REMB2} \]
MOVFP \[ \text{BARGB1, WREG} \]
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
GOTO SOK22#v(i)

ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO

SOK22#v(i)
SADD22#v(i)
i = i + 1
endw

SADD2224
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO

SOK2224
SADD2224
while i < 32

ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO

SADD22#v(i)
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO

SOK22#v(i)
SADD22#v(i)
i = i + 1
endw

© 1995 Microchip Technology Inc. DS00617A-page 2-391
BTFSC ACCB3, LSB
GOTO SOK22
MOVFP BARGB3, WREG
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0

SOK22
endm

UDIV3232 macro
; restore = 25/30 clks, nonrestore = 17/20 clks
; Max Timing: 16*25+1+16*30 = 881 clks
; Min Timing: 16*17+1+16*20 = 593 clks
; PM: 16*25+1+16*30 = 881 OM: 13

variable i
i = 0
while i < 8
RLCF ACCB0, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
BTFSC _C
GOTO UOK22#v(i)
MOVFP BARGB3, WREG
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
BCF _C

UOK22#v(i)
RLCF ACCB0
i = i + 1
endw
i = 8
while i < 16
RLCF ACCB1, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
BTFSC _C
GOTO UOK22#v(i)
MOVFP BARGB3, WREG
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
BCF _C

UOK22#v(i)
RLCF ACCB1
i = i + 1
endw

CLRFB TEMP
i = 16
while i < 24
RLCF ACCB2, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMPP
MOVFP BARGB3, WREG
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
CLRF WREG
SUBWF WREG
BTFSC _C
GOTO UOK22#v(i)
MOVFP BARGB3, WREG
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
CLRF WREG
ADDWF TEMP
BCF _C

UOK22#v(i)
RLCF ACCB2
i = i + 1
endw
i = 24
while i < 32
RLCF ACCB3, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMPP
MOVFP BARGB3, WREG
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
CLRF WREG
SUBWF TEMP
BTFSC _C
GOTO UOK22#v(i)
MOVFP BARGB3, WREG
UOK22_iv(i)

\[ i = i + 1 \]

endw

NDIV3232

\[
\begin{align*}
\max\text{ Timing: } & 16 + 31 \times 21 + 10 = 677 \text{ clks} \\
\min\text{ Timing: } & 16 + 31 \times 20 + 3 = 639 \text{ clks} \\
\text{ PM: } & 16 + 31 \times 29 + 10 = 925 \\
\text{ DM: } & 13
\end{align*}
\]

variable \( i \)

\[
\begin{align*}
\text{RLCF ACCB0, W} \\
\text{RLCF REMB3} \\
\text{RLCF REMB2} \\
\text{RLCF REMB1} \\
\text{RLCF REMB0} \\
\text{MOVFP BARGB3,WREG} \\
\text{SUBWF REMB3} \\
\text{MOVFP BARGB2,WREG} \\
\text{SUBWF REMB2} \\
\text{MOVFP BARGB1,WREG} \\
\text{SUBWF REMB1} \\
\text{MOVFP BARGB0,WREG} \\
\text{SUBWF REMB0} \\
\text{CLR B TEMP,W} \\
\text{SUBWF TEMP} \\
\text{RLCF ACCB0} \\
\text{i = 1} \\
\text{while i < 8} \\
\text{RLCF ACCB0, W} \\
\text{RLCF REMB3} \\
\text{RLCF REMB2} \\
\text{RLCF REMB1} \\
\text{RLCF REMB0} \\
\text{RLCF TEMP} \\
\text{MOVFP BARGB3,WREG} \\
\text{BTFSS ACCB0, LSB} \\
\text{GOTO NADD22_iv(i)} \\
\text{SUBWF REMB3} \\
\text{MOVFP BARGB2,WREG} \\
\text{SUBWF REMB2} \\
\text{MOVFP BARGB1,WREG} \\
\text{SUBWF REMB1} \\
\text{MOVFP BARGB0,WREG} \\
\text{SUBWF REMB0} \\
\text{CLR B WREG} \\
\text{SUBWF TEMP} \\
\text{GOTO NOK22_iv(i)} \\
\text{NADD22_iv(i)} \\
\text{ADDWF REMB3} \\
\text{MOVFP BARGB2,WREG} \\
\text{ADDWF REMB2} \\
\text{MOVFP BARGB1,WREG} \\
\text{ADDWF REMB1} \\
\text{MOVFP BARGB0,WREG} \\
\text{ADDWF REMB0} \\
\text{CLR B WREG} \\
\text{ADDWF TEMP}
\end{align*}
\]
NOK22#v(i) RLCF ACCB0
i = i + 1
endw
RLCF ACCB1, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB3, WREG
BTFSS ACCB0, LSB
GOTO NADD228
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWFB REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
CLRWF WREG
SUBWFB TEMP
GOTO NOK228
NADD228 ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLRWF WREG
ADDWFC TEMP
NOK228 RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB3, WREG
BTFSS ACCB1, LSB
GOTO NADD22#v(i)
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWFB REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
CLRWF WREG
SUBWFB TEMP
GOTO NOK22#v(i)
NADD22#v(i) ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLRWF WREG
ADDWFC TEMP
NOK22#v(i) RLCF ACCB1
i = i + 1
endw
RLCF   ACCB2, W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB3, WREG
BTFSS  ACCB2, LSB
GOTO   NADD2216
SUBWFB REMB3
MOVFP  BARGB2, WREG
SUBWFB REMB2
MOVFP  BARGB1, WREG
SUBWFB REMB1
MOVFP  BARGB0, WREG
CLRFB  WREG
SUBWFB TEMP
GOTO   NOK2216
NADD2216
ADDWF  REMB3
MOVFP  BARGB2, WREG
ADDWF   REMB2
MOVFP  BARGB1, WREG
ADDWF   REMB1
MOVFP  BARGB0, WREG
CLRFB  WREG
ADDWF   TEMP
NOK2216
RLCF   ACCB2
i = 17
while i < 24
RLCF   ACCB2, W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB3, WREG
BTFSS  ACCB2, LSB
GOTO   NADD22#v(i)
SUBWFB REMB3
MOVFP  BARGB2, WREG
SUBWFB REMB2
MOVFP  BARGB1, WREG
SUBWFB REMB1
MOVFP  BARGB0, WREG
CLRFB  WREG
SUBWFB TEMP
GOTO   NOK22#v(i)
NADD22#v(i)
ADDWF  REMB3
MOVFP  BARGB2, WREG
ADDWF   REMB2
MOVFP  BARGB1, WREG
ADDWF   REMB1
MOVFP  BARGB0, WREG
CLRFB  WREG
ADDWF   TEMP
NOK22#v(i)
RLCF   ACCB2
i = i + 1
endw
RLCF   ACCB3, W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB3, WREG
BTFSS  ACCB2, LSB
GOTO   NADD2224
SUBWF  REMB3
MOVFP  BARGB2, WREG
SUBWFB REMB2
MOVFP  BARGB1, WREG
SUBWFB REMB1
MOVFP  BARGB0, WREG
SUBWFB REMB0
CLRFP  WREG
SUBWFB TEMP
GOTO   NOK2224
NADD2224 ADDWF  REMB3
MOVFP  BARGB2, WREG
ADDWFC REMB2
MOVFP  BARGB1, WREG
ADDWFC REMB1
MOVFP  BARGB0, WREG
ADDWFC REMB0
CLRFP  WREG
ADDWFC TEMP
GOTO   NOK2224
NOK2224 RLCF   ACCB3
i = 25
while i < 32
RLCF   ACCB3, W
RLCF   REMB3
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB3, WREG
BTFSS  ACCB3, LSB
GOTO   NADD22#v(i)
SUBWF  REMB3
MOVFP  BARGB2, WREG
SUBWFB REMB2
MOVFP  BARGB1, WREG
SUBWFB REMB1
MOVFP  BARGB0, WREG
SUBWFB REMB0
CLRFP  WREG
SUBWFB TEMP
GOTO   NOK22#v(i)
NADD22#v(i) ADDWF  REMB3
MOVFP  BARGB2, WREG
ADDWFC REMB2
MOVFP  BARGB1, WREG
ADDWFC REMB1
MOVFP  BARGB0, WREG
ADDWFC REMB0
CLRFP  WREG
ADDWFC TEMP
GOTO   NOK22#v(i)
NOK22#v(i) RLCF   ACCB3
i=i+1
endw
BTFSC  ACCB3, LSB
GOTO   NOK22
MOVFP  BARGB3, WREG
ADDWF REMB3
MOVF BARGB2,WREG
ADDWFC REMB2
MOVF BARGB1,WREG
ADDWFC REMB1
MOVF BARGBO,WREG
ADDWFC REMB0

NOK22 endm

UDIV3231 macro
; Max Timing: 14+31*18+10 = 582 clks
; Min Timing: 14+31*17+3 = 544 clks
; PM: 14+31*24+10 = 768 DM: 12

variable i

RLCF ACCBO,W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF BARGB3,WREG
SUBWF REMB3
MOVF BARGB2,WREG
SUBWFB REMB2
MOVF BARGB1,WREG
SUBWFB REMB1
MOVF BARGBO,WREG
SUBWFB REMB0
RLCF ACCBO
i = 1
while i < 8
RLCF ACCBO,W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF BARGB3,WREG
BTFSS ACCBO,LSB
GOTO UADD21#v(i)
SUBWF REMB3
MOVF BARGB2,WREG
SUBWFB REMB2
MOVF BARGB1,WREG
SUBWFB REMB1
MOVF BARGBO,WREG
SUBWFB REMB0
GOTO UOK21#v(i)

UADD21#v(i) ADDWF REMB3
MOVF BARGB2,WREG
ADDWFC REMB2
MOVF BARGB1,WREG
ADDWFC REMB1
MOVF BARGBO,WREG
ADDWFC REMB0

UOK21#v(i) RLCF ACCBO
i=i+1
endw

RLCF ACCBO,W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF BARGB3,WREG
BTFSS ACCBO,LSB
GOTO UADD218
SUBWF REMB3
MOVF BARGB2,WREG
SUBWFB  REMB2
MOVFP  BARGB1,WREG
SUBWFB  REMB1
MOVFP  BARGB0,WREG
SUBWFB  REMB0
GOTO  UOK218

UADD218
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC  REMB2
MOVFP  BARGB1,WREG
ADDWFC  REMB1
MOVFP  BARGB0,WREG
ADDWFC  REMB0

UOK218
RLCF  ACCB1

i = 9
while i < 16
RLCF  ACCB1,W
RLCF  REMB3
RLCF  REMB2
RLCF  REMB1
RLCF  REMB0
MOVFP  BARGB3,WREG
BTFSS  ACCB1,LSB
GOTO  UADD218v(i)
SUBWF  REMB3
MOVFP  BARGB2,WREG
SUBWFB  REMB2
MOVFP  BARGB1,WREG
SUBWFB  REMB1
MOVFP  BARGB0,WREG
SUBWFB  REMB0
GOTO  UOK218v(i)

UADD218v(i)
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC  REMB2
MOVFP  BARGB1,WREG
ADDWFC  REMB1
MOVFP  BARGB0,WREG
ADDWFC  REMB0

UOK218v(i)
RLCF  ACCB1

i = i + 1
endw
RLCF  ACCB2,W
RLCF  REMB3
RLCF  REMB2
RLCF  REMB1
RLCF  REMB0
MOVFP  BARGB3,WREG
BTFSS  ACCB1,LSB
GOTO  UADD2116
SUBWF  REMB3
MOVFP  BARGB2,WREG
SUBWFB  REMB2
MOVFP  BARGB1,WREG
SUBWFB  REMB1
MOVFP  BARGB0,WREG
SUBWFB  REMB0
GOTO  UOK2116

UADD2116
ADDWF  REMB3
MOVFP  BARGB2,WREG
ADDWFC  REMB2
MOVFP  BARGB1,WREG
ADDWFC  REMB1
MOVFP  BARGB0,WREG
ADDWFC  REMB0

UOK2116
RLCF  ACCB2

© 1995 Microchip Technology Inc.  DS00617A-page 2-399
\begin{verbatim}
 i = 17
while i < 24
  RLCF ACCB2,W
  RLCF REMB3
  RLCF REMB2
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB3,WRG
  BTFS ACCB2,LSB
  GOTO UADD21\#v(i)
  SUBWF REMB3
  MOVFP BARGB2,WRG
  SUBWFB REMB2
  MOVFP BARGB1,WRG
  SUBWFB REMB1
  MOVFP BARGB0,WRG
  SUBWFB REMB0
  GOTO UOK21\#v(i)
UADD21\#v(i) ADDWF REMB3
  MOVFP BARGB2,WRG
  ADDWFC REMB2
  MOVFP BARGB1,WRG
  ADDWFC REMB1
  MOVFP BARGB0,WRG
  ADDWFC REMB0
  UOK21\#v(i) RLCF ACCB2
  i = i + 1
endw
i = 1 + 1
while i < 32
  RLCF ACCB3,W
  RLCF REMB3
  RLCF REMB2
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB3,WRG
  BTFS ACCB3,LSB
  GOTO UADD2124
  SUBWF REMB3
  MOVFP BARGB2,WRG
  SUBWFB REMB2
  MOVFP BARGB1,WRG
  SUBWFB REMB1
  MOVFP BARGB0,WRG
  SUBWFB REMB0
  GOTO UOK2124
UADD2124 ADDWF REMB3
  MOVFP BARGB2,WRG
  ADDWFC REMB2
  MOVFP BARGB1,WRG
  ADDWFC REMB1
  MOVFP BARGB0,WRG
  ADDWFC REMB0
  UOK2124 RLCF ACCB3
  i = 25
while i < 32
  RLCF ACCB3,W
  RLCF REMB3
  RLCF REMB2
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB3,WRG
  BTFS ACCB3,LSB
  GOTO UADD21\#v(i)
  SUBWF REMB3
  MOVFP BARGB2,WRG
  SUBWFB REMB2
  MOVFP BARGB1,WRG
\end{verbatim}
SUBWBFB REMB1
MOVFP BARGB0, WREG
SUBWBFB REMB0
GOTO UOK21\#v(i)
UADD21\#v(i) ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
UOK21\#v(i) RLCF ACCB3
i = i + i
endw
BTFSC ACCB3, LSB
GOTO UOK21
MOVFP BARGB3, WREG
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWFC REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
UOK21 endm
UDIV3131 macro
; Max Timing:  9 + 14 + 30 * 18 + 10 = 573 clks
; Min Timing:  9 + 14 + 30 * 17 + 3 = 536 clks
; PM:  9 + 14 + 30 * 24 + 10 = 753 DM: 12
variable i
MOVFP BARGB3, WREG
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
RLCF ACCB0
RLCF ACCB0, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
RLCF ACCB0
i = 2
while i < 8
RLCF ACCB0, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
BTFSS ACCB0, LSB
GOTO UADD21\#v(i)
SUBWBFB REMB3
MOVFP BARGB2, WREG
SUBWFB BARGBl, WREG
MOVFP REMBl
SUBWFB BARGBo, WREG
MOVFP REMBo
GOTO UOK111v(i)

UADD11v(i)
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0

UOK11v(i)
RLCF ACCB0

while i < 16
RLCF ACCB1, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
BTFSS ACCB0, LSB
GOTO UADD118

SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0

UADD118
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0

UOK118
RLCF ACCB1

i = 9

while i < 16
RLCF ACCB1, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
BTFSS ACCB1, LSB
GOTO UADD11v(i)

SUBWF REMB3
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0

UADD11v(i)
ADDWF REMB3
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0

UOK11v(i)
RLCF ACCB1
i=i+1
endw

RLCF ACCB2,W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3,WREG
BTFSS ACCB1,LSB
GOTO UADD1116
SUBWF REMB3
MOVFP BARGB2,WREG
SUBWFB REMB2
MOVFP BARGB1,WREG
SUBWFB REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK1116

UADD1116
ADDWF REMB3
MOVFP BARGB2,WREG
ADDWFC REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
GOTO UOK1116

UOK1116

i = 17
while i < 24

RLCF ACCB2,W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3,WREG
BTFSS ACCB2,LSB
GOTO UADD11#v(i)
SUBWF REMB3
MOVFP BARGB2,WREG
SUBWFB REMB2
MOVFP BARGB1,WREG
SUBWFB REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK11#v(i)

UADD11#v(i)
ADDWF REMB3
MOVFP BARGB2,WREG
ADDWFC REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
GOTO UOK11#v(i)

UOK11#v(i)

i=i+1
endw

RLCF ACCB3,W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3,WREG
BTFSS ACCB2,LSB
GOTO UADD1124
SUBWF REMB3
MOVFP BARGB2,WREG
SUBWFB REMB2
MOVFP BARGB1,WREG
AN617

SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK1124
UADD1124
ADDFC REMB3
MOVFP BARGB2, WREG
ADDFC REMB2
MOVFP BARGB1, WREG
ADDFC REMB1
MOVFP BARGB0, WREG
ADDFC REMB0
UOK1124
RLCF ACCB3
i = 25
while i < 32
RLCF ACCB3, W
RLCF REMB3
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
BTFSS ACCB3, LSB
GOTO UADD11#v(i)
SUBWF REMB3
MOVFP BARGB2, WREG
SUBWFB REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK11#v(i)
UADD11#v(i)
ADDFC REMB3
MOVFP BARGB2, WREG
ADDFC REMB2
MOVFP BARGB1, WREG
ADDFC REMB1
MOVFP BARGB0, WREG
ADDFC REMB0
UOK11#v(i)
RLCF ACCB3
i = i + 1
endw
BTFSC ACCB3, LSB
GOTO UOK11
MOVFP BARGB3, WREG
ADDFC REMB3
MOVFP BARGB2, WREG
ADDFC REMB2
MOVFP BARGB1, WREG
ADDFC REMB1
MOVFP BARGB0, WREG
ADDFC REMB0
UOK11
endm

;**********************************************************************************************
;**********************************************************************************************
32/24 Bit Division Macros

SDIV3224
macro
; Max Timing:  7+11+30*15+8 = 476 clks
; Min Timing:  7+11+30*14+3 = 441 clks
; PM: 7+11+30*19+8 = 596  DM: 10

variable i
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0

"
AN617

RLCF ACCBO
RLCF ACCBO,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
ADDWF REMB2
MOVFP BARGB1,WREG
'ADDWFC REMB1
MOVFP BARGBO,WREG
ADDWFC REMB0
RLCF ACCBO

i = 2
while i < 8
RLCF ACCBO,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCBO,LSB
GOTO SADD24#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGBO,WREG
SUBWFB REMB0
GOTO SOK24#v(i)
SADD24#v(i)

RLCF ACCB1,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCBO,LSB
GOTO SADD248
SUBWF REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGBO,WREG
SUBWFB REMB0
GOTO SOK248
SADD248

RLCF ACCB1

i = i+1
endw

SOK24#v(i)

RLCF ACCB1,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCB1,LSB
GOTO SADD24#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGBO,WREG
ADDWFC REMB0
GOTO SOK248
SADD248

RLCF ACCB1

i = 9
while i < 16
RLCF ACCB1,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCB1,LSB
GOTO SADD24#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWFB REMB1

© 1995 Microchip Technology Inc.

DS00617A-page 2-405
AN617

MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK24#v(i)
ADDWF REMBO
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK24#v(i) RLCF ACCB1
i=i+1
endw
RLCF ACCB2,W
RLCF REMB2
RLCF REMB1
RLCF REMBO
MOVFP BARGB2, WREG
BTFSS ACCB1, LSB
GOTO SADD2416
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK2416
SADD2416 ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK2416 RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2,W
RLCF REMB2
RLCF REMB1
RLCF REMBO
MOVFP BARGB2, WREG
BTFSS ACCB2, LSB
GOTO SADD24#v(i)
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK24#v(i)
SADD24#v(i) ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK24#v(i) RLCF ACCB2
i=i+1
endw
RLCF ACCB3,W
RLCF REMB2
RLCF REMB1
RLCF REMBO
MOVFP BARGB2, WREG
BTFSS ACCB2, LSB
GOTO SADD2424
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK2424
SADD2424 ADDWF REMB2
MOVF P BARGBl, WREG
ADDWFC REMB1
MOVF P BARGB0, WREG
ADDWFC REMB0

SOK2424 RLCF ACCB3
i = 25
while i < 32
RLCF ACCB3, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF P BARGB2, WREG
BTFSS ACCB3, LSB
GOTO SADD24#v(i)
SUBWF REMB2
MOVF P BARGBl, WREG
SUBWFB REMB1
MOVF P BARGB0, WREG
SUBWFB REMB0
GOTO SOK24#v(i)

SADD24#v(i) ADDWF REMB2
MOVF P BARGBl, WREG
ADDWFC REMB1
MOVF P BARGB0, WREG
ADDWFC REMB0

SOK24#v(i) RLCF ACCB3
i = i + 1
endw
BTFSC ACCB3, LSB
GOTO SOK24
MOVF P BARGB2, WREG
ADDWF REMB2
MOVF P BARGBl, WREG
ADDWFC REMB1
MOVF P BARGB0, WREG
ADDWFC REMB0

SOK24 endm
UDIV3224 macro
; restore = 20/25 clks, nonrestore = 14/17 clks
; Max Timing: 16*20+1+16*25 = 721 clks
; Min Timing: 16*14+1+16*17 = 497 clks
; PM: 16*20+1+16*25 = 721 DM: 11
variable i
i = 0
while i < 8
RLCF ACCB0, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF P BARGB2, WREG
SUBWF REMB2
MOVF P BARGBl, WREG
SUBWFB REMB1
MOVF P BARGB0, WREG
SUBWFB REMB0
BTFSC _C
GOTO UOK24#v(i)
MOVF P BARGB2, WREG
ADDWF REMB2
MOVF P BARGBl, WREG
ADDWFC REMB1
MOVF P BARGB0, WREG
ADDWFC REMB0
BCF _C

© 1995 Microchip Technology Inc. DS00617A-page 2-407
UOK24#v(i) RLCF ACCB0
  i = i + 1
endw
  i = 8
while i < 16
  RLCF ACCB1.W
  RLCF REMB2
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB2,WREG
  SUBWF REMB2
  MOVFP BARGB1,WREG
  SUBWF REMB1
  MOVFP BARGB0,WREG
  SUBWF REMB0
  BTFSC _C
  GOTO UOK24#v(i)
  MOVFP BARGB2,WREG
  ADDWF REMB2
  MOVFP BARGB1,WREG
  ADDWF REMB1
  MOVFP BARGB0,WREG
  ADDWF REMB0
  BCF _C
UOK24#v(i) RLCF ACCB1
  i = i + 1
endw
  CLRF TEMP
  i = 16
while i < 24
  RLCF ACCB2.W
  RLCF REMB2
  RLCF REMB1
  RLCF REMB0
  RLCF TEMP
  MOVFP BARGB2,WREG
  SUBWF REMB2
  MOVFP BARGB1,WREG
  SUBWF REMB1
  MOVFP BARGB0,WREG
  SUBWF REMB0
  CLRF WREG
  SUBWF TEMP
  BTFSC _C
  GOTO UOK24#v(i)
  MOVFP BARGB2,WREG
  ADDWF REMB2
  MOVFP BARGB1,WREG
  ADDWF REMB1
  MOVFP BARGB0,WREG
  ADDWF REMB0
  CLRF WREG
  ADDWF TEMP
  BCF _C
UOK24#v(i) RLCF ACCB2
  i = i + 1
endw
  i = 24
while i < 32
  RLCF ACCB3.W
  RLCF REMB2
  RLCF REMB1
  RLCF REMB0
  RLCF TEMP
  MOVFP BARGB2,WREG
  SUBWF REMB2
MOVFP  BARGBl,WREG
SUBWFB REMBl
MOVFP  BARGBo,WREG
SUBWFB REMBo
CLRf  WREG
SUBWFB TEMP
BTFSc  _c
GOTO  UKO24#v(i)
MOVFP  BARGB2,WREG
ADDWF  REMB2
MOVFP  BARGBl,WREG
ADDWFC  REMBl
MOVFP  BARGBo,WREG
ADDWFC  REMBo
CLRf  WREG
ADDWFC  TEMP
BCF  _c
UKO24#v(i)  RLCF  ACCB3
i=i+1
endw
endm
NDIV3224  macro
; Max Timing: 13+31*18+8 = 579 clks
; Min Timing: 13+31*17+3 = 543 clks
; PM: 13+31*24+8 = 765  DM: 11
variable i
RLCF  ACCB0,W
RLCF  REMB2
RLCF  REMBl
RLCF  REMBo
MOVFP  BARGB2,WREG
SUBWF  REMB2
MOVFP  BARGBl,WREG
SUBWF  REMBl
MOVFP  BARGBo,WREG
SUBWF  REMBo
CLRf  TEMP,W
SUBWFB TEMP
RLCF  ACCB0
i = 1
while i < 8
RLCF  ACCB0,W
RLCF  REMB2
RLCF  REMBl
RLCF  REMBo
RLCF  TEMP
MOVFP  BARGB2,WREG
BTFSS  ACCB0,LSB
GOTO  NADD24#v(i)
SUBWF  REMB2
MOVFP  BARGBl,WREG
SUBWF  REMBl
MOVFP  BARGBo,WREG
SUBWF  REMBo
CLRf  WREG
SUBWFB TEMP
GOTO  NOK24#v(i)
NADD24#v(i)  ADDWF  REMB2
MOVFP  BARGBl,WREG
ADDWFC  REMBl
MOVFP  BARGBo,WREG
ADDWFC  REMBo
CLRf  WREG
ADDWFC  TEMP
NOK24#v(i)  RLCF  ACCB0
i = i + 1
endw

RLCF   ACCB1, W
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB2, WREG
BTFSS  ACCB1, LSB
GOTO   NADD248
SUBWF  REMB2
MOVFP  BARGB1, WREG
SUBWF  REMB1
MOVFP  BARGB0, WREG
SUBWF  REMB0
CLRIF  WREG
SUBWF  TEMP
GOTO   NOK248

NADD248
ADDWF  REMB2
MOVFP  BARGB1, WREG
ADDWFC  REMB1
MOVFP  BARGB0, WREG
ADDWFC  REMB0
CLRIF  WREG
ADDWFC  TEMP
GOTO   NOK248

NOK248
RLCF   ACCB1
i = 9
while i < 16
RLCF   ACCB1, W
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB2, WREG
BTFSS  ACCB1, LSB
GOTO   NADD24v(i)
SUBWF  REMB2
MOVFP  BARGB1, WREG
ADDWFC  REMB1
MOVFP  BARGB0, WREG
ADDWFC  REMB0
CLRIF  WREG
SUBWF  TEMP
GOTO   NOK24v(i)

NADD24v(i)
ADDWF  REMB2
MOVFP  BARGB1, WREG
ADDWFC  REMB1
MOVFP  BARGB0, WREG
ADDWFC  REMB0
CLRIF  WREG
ADDWFC  TEMP
GOTO   NOK24v(i)

NOK24v(i)
RLCF   ACCB1
i = i + 1
endw

RLCF   ACCB2, W
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
RLCF   TEMP
MOVFP  BARGB2, WREG
BTFSS  ACCB1, LSB
GOTO   NADD2416
SUBWF  REMB2
MOVFP  BARGB1, WREG
```assembly
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
CLR WREG
SUBWFB TEMP
GOTO NOK2416

NADD2416
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLR WREG
ADDWFC TEMP
NOK2416
RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB2, WREG
BTFSS ACCB2, LSB
GOTO NADD24#v(i)
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
CLR WREG
SUBWFB TEMP
GOTO NOK24#v(i)

NADD24#v(i)
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLR WREG
ADDWFC TEMP
NOK24#v(i)
RLCF ACCB2
i = i + 1
endw
RLCF ACCB3, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB2, WREG
BTFSS ACCB2, LSB
GOTO NADD2424
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
CLR WREG
SUBWFB TEMP
GOTO NOK2424

NADD2424
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLR WREG
```
ADDWFC    TEMP

NOK2424   RLCF    ACCB3
i = 25
while i < 32
RLCF    ACCB3,W
RLCF    REMB2
RLCF    REMB1
RLCF    REMB0
RLCF    TEMP
MOVFP    BARGB2, WREG
BTFSS    ACCB3, LSB
GOTO    NADD24#v(i)
SUBWF    REMB2
MOVFP    BARGB1, WREG
SUBWF    REMB1
MOVFP    BARGB0, WREG
SUBWF    REMB0
CLRWF    WREG
SUBWF    TEMP
GOTO    NOK24#v(i)
NADD24#v(i)  ADDWFC    REMB2
MOVFP    BARGB1, WREG
ADDWFC    REMB1
MOVFP    BARGB0, WREG
ADDWFC    REMB0
CLRWF    WREG
ADDWFC    TEMP

NOK24#v(i)  RLCF    ACCB3
i = i + 1
endw
BTFSC    ACCB3, LSB
GOTO    NOK24
MOVFP    BARGB2, WREG
ADDWFC    REMB2
MOVFP    BARGB1, WREG
ADDWFC    REMB1
MOVFP    BARGB0, WREG
ADDWFC    REMB0
NOK24
endm
UDIV3223

macro
; Max Timing: 11+31*15+8 = 484 clks
; Min Timing: 11+31*14+3 = 448 clks
; PM: 11+31*19+8 = 608  DM: 10
variable i
RLCF    ACCB0, W
RLCF    REMB2
RLCF    REMB1
RLCF    REMB0
MOVFP    BARGB2, WREG
SUBWF    REMB2
MOVFP    BARGB1, WREG
SUBWF    REMB1
MOVFP    BARGB0, WREG
SUBWF    REMB0
RLCF    ACCB0
i = 1
while i < 8
RLCF    ACCB0, W
RLCF    REMB2
RLCF    REMB1
RLCF    REMB0
MOVFP    BARGB2, WREG
BTFSS    ACCB0, LSB
GOTO UADD23#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWFB REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK23#v(i)
UADD23#v(i) ADDWF REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
UOK23#v(i) i = i + 1
endw
RLCF ACCB1, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCB0, LSB
GOTO UADD238
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWFB REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK238
UADD238 ADDWF REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
UOK238 RLCF ACCB1
while i < 16
RLCF ACCB1, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCB1, LSB
GOTO UADD23#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWFB REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK23#v(i)
UADD23#v(i) ADDWF REMB2
MOVFP BARGB1,WREG
ADDWFC REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
UOK23#v(i) RLCF ACCB1
i = i + 1
endw
RLCF ACCB2, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFSS ACCB1, LSB
GOTO UADD2316
SUBWF REMB2
MOVFP BARGB1,WREG
AN617

SUBWF   REMB1
MOVFP   BARGB0, WREG
SUBWF   REMB0
GOTO    UOK2316

UADD2316
ADDWF   REMB2
MOVFP   BARGB1, WREG
ADDFWC   REMB1
MOVFP   BARGB0, WREG
ADDFWC   REMB0

UOK2316
RLCF   ACCB2
i = 17
while i < 24
RLCF   ACCB2, W
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
MOVFP   BARGB2, WREG
BTFSS   ACCB2, LSB
GOTO    UADD23#v(i)
SUBWF   REMB2
MOVFP   BARGB1, WREG
ADDFWC   REMB1
MOVFP   BARGB0, WREG
SUBWF   REMB0
GOTO    UOK23#v(i)

UADD23#v(i)
ADDWF   REMB2
MOVFP   BARGB1, WREG
ADDFWC   REMB1
MOVFP   BARGB0, WREG
ADDFWC   REMB0

UOK23#v(i)
RLCF   ACCB2
i = i + 1
endw
RLCF   ACCB3, W
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
MOVFP   BARGB2, WREG
BTFSS   ACCB2, LSB
GOTO    UADD2324
SUBWF   REMB2
MOVFP   BARGB1, WREG
SUBWF   REMB1
MOVFP   BARGB0, WREG
SUBWF   REMB0
GOTO    UOK2324

UADD2324
ADDWF   REMB2
MOVFP   BARGB1, WREG
ADDFWC   REMB1
MOVFP   BARGB0, WREG
ADDFWC   REMB0

UOK2324
RLCF   ACCB3
i = 25
while i < 32
RLCF   ACCB3, W
RLCF   REMB2
RLCF   REMB1
RLCF   REMB0
MOVFP   BARGB2, WREG
BTFSS   ACCB3, LSB
GOTO    UADD23#v(i)
SUBWF   REMB2
MOVFP   BARGB1, WREG
SUBWF   REMB1
MOVFP   BARGB0, WREG
SUBWF   REMB0
GOTO UOK23

UADD23#v(i)  ADDWF REMB2
MOVFP BARGB1,WREG
ADDWF REMB1
MOVFP BARGB0,WREG
ADDWF REMB0

UOK23#v(i)  RLCF ACCB3

i=i+1
endw

BTFS ACCB3,LSB
GOTO UOK23
MOVFP BARGB2,WREG
ADDFC REMB2
MOVFP BARGB1,WREG
ADDWF REMB1
MOVFP BARGB0,WREG
ADDFC REMB0

UOK23
endm

UDIV3123

; Max Timing: 7+11+30*15+8 = 476 clks
; Min Timing: 7+11+30*14+3 = 441 clks
; PM: 7+11+30*19+8 = 596  DM: 10

variable i

MOVFP BARGB2,WREG
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWF REMB0
RLCF ACCB0
RLCF ACCB0,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
ADDFW REMB2
MOVFP BARGB1,WREG
ADDFC REMB1
MOVFP BARGB0,WREG
ADDFC REMB0
RLCF ACCB0

i = 2
while i < 8

RLCF ACCB0,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2,WREG
BTFS ACCB0,LSB
GOTO UADD13#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWF REMB0
GOTO UOK13#v(i)

UADD13#v(i)  ADDWF REMB2
MOVFP BARGB1,WREG
ADDFC REMB1
MOVFP BARGB0,WREG
ADDFC REMB0

UOK13#v(i)  RLCF ACCB0

i=i+1
endw

RLCF ACCB1,W
AN617

```
RLCF  REMB2
RLCF  REMB1
RLCF  REMB0
MOVFPP BARGB2,WREG
BTFSS  ACCB0,LSB
GOTO  UADD138
SUBWF  REMB2
MOVFPP BARGB1,WREG
SUBWF  REMB1
MOVFPP BARGB0,WREG
SUBWF  REMB0
GOTO  UOK138
UADD138
ADDFP  REMB2
MOVFPP BARGB1,WREG
ADDWFC  REMB1
MOVFPP BARGB0,WREG
ADDWFC  REMB0
UOK138
RLCF  ACCB1
i = 9
while i < 16
  RLCF  ACCB1,W
  RLCF  REMB2
  RLCF  REMB1
  RLCF  REMB0
  MOVFPP BARGB2,WREG
  BTFSS  ACCB1,LSB
  GOTO  UADD13#v(i)
  SUBWF  REMB2
  MOVFPP BARGB1,WREG
  SUBWF  REMB1
  MOVFPP BARGB0,WREG
  SUBWF  REMB0
  GOTO  UOK13#v(i)
UADD13#v(i)
  ADDWF  REMB2
  MOVFPP BARGB1,WREG
  ADDWFC  REMB1
  MOVFPP BARGB0,WREG
  ADDWFC  REMB0
UOK13#v(i)
  RLCF  ACCB1
i=i+1
endw
  RLCF  ACCB2,W
  RLCF  REMB2
  RLCF  REMB1
  RLCF  REMB0
  MOVFPP BARGB2,WREG
  BTFSS  ACCB1,LSB
  GOTO  UADD1316
  SUBWF  REMB2
  MOVFPP BARGB1,WREG
  SUBWF  REMB1
  MOVFPP BARGB0,WREG
  SUBWF  REMB0
  GOTO  UOK1316
UADD1316
  ADDWF  REMB2
  MOVFPP BARGB1,WREG
  ADDWFC  REMB1
  MOVFPP BARGB0,WREG
  ADDWFC  REMB0
UOK1316
  RLCF  ACCB2
  i = 17
  while i < 24
  RLCF  ACCB2,W
  RLCF  REMB2
  RLCF  REMB1
  RLCF  REMB0
```
AN617

```
MOVF P BARGB2, WREG
BTFSS ACCB2, LSB
GOTO UADD13#v(i)
SUBWF REMB2
MOVF P BARGB1, WREG
SUBWFB REMB1
MOVF P BARGB0, WREG
SUBWFB REMB0
GOTO UOK13#v(i)

UADD13#v(i) ADDWF REMB2
MOVF P BARGB1, WREG
ADDF C REMB1
MOVF P BARGB0, WREG
ADDF C REMB0

UOK13#v(i) RLCF ACCB2
i = i + 1
endw

RLCF ACCB3, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF P BARGB2, WREG
BTFSS ACCB2, LSB
GOTO UADD1324
SUBWF REMB2
MOVF P BARGB1, WREG
SUBWFB REMB1
MOVF P BARGB0, WREG
SUBWFB REMB0
GOTO UOK1324

UADD1324 ADDWF REMB2
MOVF P BARGB1, WREG
ADDF C REMB1
MOVF P BARGB0, WREG
ADDF C REMB0

UOK1324 RLCF ACCB3
i = 25
while i < 32

RLCF ACCB3, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVF P BARGB2, WREG
BTFSS ACCB3, LSB
GOTO UADD13#v(i)
SUBWF REMB2
MOVF P BARGB1, WREG
SUBWFB REMB1
MOVF P BARGB0, WREG
SUBWFB REMB0
GOTO UOK13#v(i)

UADD13#v(i) ADDWF REMB2
MOVF P BARGB1, WREG
ADDF C REMB1
MOVF P BARGB0, WREG
ADDF C REMB0

UOK13#v(i) RLCF ACCB3
i = i + 1
endw

BTFSC ACCB3, LSB
GOTO UOK13
MOVF P BARGB2, WREG
ADDF C REMB2
MOVF P BARGB1, WREG
ADDF C REMB1
MOVF P BARGB0, WREG
```
; 32/32 Bit Signed Fixed Point Divide 32/32 -> 32.32
; Input: 32 bit signed fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
; 32 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2, BARGB3
; Use: CALL FXD3232S
; Output: 32 bit signed fixed point quotient in AARGBO, AARGB1, AARGB2, AARGB3
; 32 bit fixed point remainder in REMBO, REMBl, REMB2, REMB3
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 13+573+3 = 589 clks A > 0, B > 0
; 20+573+21 = 614 clks A > 0, B < 0
; 20+573+21 = 614 clks A < 0, B > 0
; 27+573+3 = 603 clks A < 0, B < 0
; Min Timing: 13+536+3 = 552 clks A > 0, B > 0
; 20+536+21 = 577 clks A > 0, B < 0
; 20+536+21 = 577 clks A < 0, B > 0
; 27+536+3 = 566 clks A < 0, B < 0
; PM: 27+753+20 = 800

FXD3232S
MOVFP AARGBO, WREG
XORWF BARGBO, W
MOVWF SIGN
CLRF REMBO
CLRF REMBl
CLRF REMB2
CLRF REMB3, W
BTFSS BARGBO, MSB if MSB set, negate BARG
GOTO CA3232S
COMP BARG3
COMP BARGB2
COMP BARGBl
COMP BARGB0
INCF BARGB3
ADDWFC BARGB2
ADDWFC BARGBl
ADDWFC BARGB0
CA3232S
BTFSS AARGBO, MSB if MSB set, negate AARG
GOTO C3232S
COMP AARGB3
COMP AARGB2
COMP AARGB1
COMP AARGB0
INCF AARGB3
ADDWFC AARGB2
ADDWFC AARGB1
ADDWFC AARGB0
ADDWFC AARGB0

C3232S
SDIV3232
BTFSS SIGN, MSB
RETLW 0x00
COMP AARGB3
COMP AARGB2
COMP AARGB1
COMP AARGB0
CLRF WREG
INCF AARGB3
ADDWFC AARGB2
ADDWFC AARGB1
ADDWFC AARGB0
COMP REMB3
COMP REMB2
COMP REMB1
COMP REMB0
INCF REMB3

© 1995 Microchip Technology Inc.
ADDWF C ADDWFC REMB2
ADDWF C ADDWFC REMB1
ADDWF C ADDWFC REMB0
RETLW 0x00

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

; 32/32 Bit Unsigned Fixed Point Divide 32/32 -> 32.32
; Input: 32 bit unsigned fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
; 32 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2, BARGB3
; Use: CALL FXD3232U
; Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1, AARGB2, AARGB3
; 32 bit unsigned fixed point remainder in REMBO, REMBl, REMB2, REMB3
; Result: AARG, REM <- AARG / BARG
; Max Timing: 4+677+2 = 683 clks
; Min Timing: 4+639+2 = 645 clks
; PM: 4+925+1 = 930 DM: 13
FXD3232U CLRFB REMB0
CLRFB REMB1
CLRFB REMB2
CLRFB REMB3
UDIV3232
RETLW 0x00

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

; 32/31 Bit Unsigned Fixed Point Divide 32/31 -> 32.31
; Input: 32 bit unsigned fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
; 31 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2, BARGB3
; Use: CALL FXD3231U
; Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1, AARGB2, AARGB3
; 31 bit unsigned fixed point remainder in REMBO, REMBl, REMB2, REMB3
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 4+582+2 = 588 clks
; Min Timing: 4+544+2 = 550 clks
; PM: 4+768+1 = 773 DM: 12
FXD3231U CLRFB REMB0
CLRFB REMB1
CLRFB REMB2
CLRFB REMB3
UDIV3231
RETLW 0x00

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

; 31/31 Bit Unsigned Fixed Point Divide 31/31 -> 31.31
; Input: 31 bit unsigned fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
; 31 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2, BARGB3
; Use: CALL FXD3131U
; Output: 31 bit unsigned fixed point quotient in AARGBO, AARGB1, AARGB2, AARGB3
; 31 bit unsigned fixed point remainder in REMBO, REMBl, REMB2, REMB3
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 4+573+2 = 579 clks
; Min Timing: 4+536+2 = 542 clks
; PM: 4+753+1 = 758 DM: 12
FXD3131U CLRFB REMB0
CLRFB REMB1
CLRFB REMB2
CLRFB REMB3
UDIV3131
RETLW 0x00

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

; 32/24 Bit Signed Fixed Point Divide 32/24 -> 32.24
; Input: 32 bit signed fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
; 24 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2
AN617

Use: CALL FXD3224S
Output: 32 bit signed fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
24 bit fixed point remainder in REMBO, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing:
12+476+3 = 491 clks A > 0, B > 0
17+476+19 = 512 clks A > 0, B < 0
19+476+19 = 514 clks A < 0, B > 0
24+476+3 = 503 clks A < 0, B < 0
Min Timing:
12+441+3 = 456 clks A > 0, B > 0
17+441+19 = 477 clks A > 0, B < 0
19+441+19 = 479 clks A < 0, B > 0
24+441+3 = 468 clks A < 0, B < 0
PM: 24+596+19 = 639 elks

FXD3224S
MOVFP AARGBO,WREG
XORWF BARGB0,W
MOVWF SIGN
CLRF REMB0
CLRF REMB1
CLRF REMB2,W
BTFSS BARGB0,MSB ; if MSB set, negate BARG
GOTO CA3224S
COMF BARGB2
COMF BARGB1
COMF BARGB0
INCF BARGB2
ADDWF BARGB1
ADDWF BARGB0
CA3224S
BTFSS AARGBO,MSB ; if MSB set, negate AARG
GOTO C3224S
COMF AARGB3
COMF AARGB2
COMF AARGB1
COMF AARGB0
CLRF WREG
INCF AARGB3
ADDWF AARGB2
ADDWF AARGB1
ADDWF AARGB0
COMF REMB2
COMF REMB1
COMF REMB0
INCF REMB2
ADDWF REMB1
ADDWF REMB0
RETLW 0x00

SDIV3224
BTFSS SIGN,MSB
RETLW 0x00
COMF AARGB3
COMF AARGB2
COMF AARGB1
COMF AARGB0
CLRF WREG
INCF AARGB3
ADDWF AARGB2
ADDWF AARGB1
ADDWF AARGB0
COMF REMB2
COMF REMB1
COMF REMB0
INCF REMB2
ADDWF REMB1
ADDWF REMB0
RETLW 0x00

*******************************************************************************
*******************************************************************************
32/24 Bit Unsigned Fixed Point Divide 32/24 -> 32.24
Input: 32 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3
24 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2
Use: CALL FXD3224U
Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1AARGB2,AARGB3
24 bit unsigned fixed point remainder in REMBO, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing: 3+579+2 = 584 clks
32/23 Bit Unsigned Fixed Point Divide 32/23 -> 32.23
Input: 32 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3
23 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2
Use: CALL FXD3223U
Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
23 bit unsigned fixed point remainder in REMBO, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing: 3+484+2 489 clks
Min Timing: 3+448+2 = 453 clks
PM: 3+608+1 612 DM: 10
FXD3223U CLRF REMB0
CLRF REMB1
CLRF REMB2
UDIV3223
RETLW 0x00

31/23 Bit Unsigned Fixed Point Divide 31/23 -> 31.23
Input: 31 bit unsigned fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3
23 bit unsigned fixed point divisor in BARGBO, BARGBl, BARGB2
Use: CALL FXD3123U
Output: 31 bit unsigned fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
23 bit unsigned fixed point remainder in REMBO, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing: 3+476+2 481 clks
Min Timing: 3+441+2 = 446 clks
PM: 3+596+1 600 DM: 10
FXD3123U CLRF REMB0
CLRF REMB1
CLRF REMB2
UDIV3123
RETLW 0x00

F.2 PIC17CXX Fixed Point Divide Routines B

PIC17 FIXED POINT DIVIDE ROUTINES B VERSION 1.8
Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed divide application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine          Clocks  Function
FXD2416S        314  24 bit/16 bit -> 24.16 signed fixed point divide
FXD2416U        365  24 bit/16 bit -> 24.16 unsigned fixed point divide
FXD2415U        294  24 bit/15 bit -> 24.15 unsigned fixed point divide
FXD2315U        287  23 bit/15 bit -> 23.15 unsigned fixed point divide
FXD1616S        214  16 bit/16 bit -> 16.16 signed fixed point divide
FXD1616U        244  16 bit/16 bit -> 16.16 unsigned fixed point divide
FXD1615U        197  16 bit/15 bit -> 16.15 unsigned fixed point divide
FXD1515U        191  15 bit/15 bit -> 15.15 unsigned fixed point divide
FXD1608S        146  16 bit/08 bit -> 16.08 signed fixed point divide
; FXD1608U 196 16 bit/08 bit -> 16.08 unsigned fixed point divide
; FXD1607U 130 16 bit/07 bit -> 16.07 unsigned fixed point divide
; FXD1507U 125 15 bit/07 bit -> 15.07 unsigned fixed point divide
; FXD0808S 77 08 bit/08 bit -> 08.08 signed fixed point divide
; FXD0808U 75 08 bit/08 bit -> 08.08 unsigned fixed point divide
; FXD0807U 66 08 bit/07 bit -> 08.07 unsigned fixed point divide
; FXD0707U 61 07 bit/07 bit -> 07.07 unsigned fixed point divide

list r=dec,x=on,t=off,p=17C42

include <PIC17.INC> ; general PIC17 definitions

include <MATH17.INC> ; PIC17 math library definitions

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

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

Test suite for fixed point divide algorithms

org 0x0021
MAIN
MOVLY RAMSTART
MOVFP WREG,FSRX
MEMLOOP
CLRF INDFX
INCFSZ MEMLOOP
BSF RTCSTA.5
; MOVPF RTCCH,WREG
MOVLY 0x45
MOVPF WREG,RANDLO
; MOVPF WREG,RANDLO
MOVLY 0x30
MOVPF WREG,RANDHI
MOVLY 0x30
MOVPF WREG,FSRX
BCF _FS1
BSF _FS0
; CALL TFXD0808
; CALL TFXD1608
; CALL TFXD1616
; CALL TFXD2416
MOVLY 0xFF
MOVPF WREG,AARGBO
MOVLY 0xFF
MOVPF WREG,AARGB1
MOVLY 0xFF
MOVPF WREG,AARGB2
MOVLY 0xFF
MOVPF WREG,AARGB3
MOVLY 0xFF
MOVPF WREG,BARGBO
MOVLY 0xFF
MOVPF WREG,BARGB1
CALL FXD1616U
SELF
GOTO SELF
RANDOM16
RLCF RANDHI,W
XORWF RANDHI,W
RLCF WREG
SWAPF RANDHI
SWAPF RANDLO,W
RLNCF WREG
XORWF RANDHI,W
SWAPF RANDHI

; random number generator

© 1995 Microchip Technology Inc.
AN617

ANDLW 0x01
RLCF RANDLO
XORWF RANDLO
RLCF RANDHI

RETLW 0

; Test suite for FXD2416
TFXD2416
MOVLW 20
MOVPF WREG,TESTCOUNT
MOVPF WREG,NUMTESTS
MOVLW 1
MOVPF WREG,TESTCODE

D2416LOOP
CALL RANDOM16
MOVPF RANDHI,WREG
MOVPF WREG,BARGBO
MOVPF RANDLO,WREG
MOVPF WREG,BARGB1

; BCF BARGBO,MSB
MOVPF BARGBO,INDFO
MOVPF BARGB1,INDFO
CALL FXD2416S

; CALL FXD2416U
; CALL FXD2415U
; CALL FXD2315U

MOVPF AARGBO,INDFO
MOVPF AARGB1,INDFO
MOVPF AARGB2,INDFO
MOVPF AARGB0,INDFO
MOVF D2416LOOP
CALL FXD216S

; Test suite for FXD1616
TFXD1616
MOVLW 26
MOVPF WREG,TESTCOUNT
MOVPF WREG,NUMTESTS
MOVLW 2
MOVPF WREG,TESTCODE

D1616LOOP
CALL RANDOM16
; SWAPF RANDHI
; SWAPF RANDLO
MOVPF RANDHI,WREG
MOVPF WREG,BARGBO
MOVPF RANDLO,WREG
MOVPF WREG,BARGB1

; BCF BARGBO,MSB
MOVPF BARGBO,INDFO
MOVPF BARGB1,INDFO
CALL RANDOM16

; SWAPF RANDHI
; SWAPF RANDLO
MOVFP RANDHI, WREG
MOVFP WREG, AARGBO
MOVFP RANDLO, WREG
MOVFP WREG, AARGBl
;  
BCF AARGB0, MSB
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
CALL FXD1616U
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
MOVFP REMBO, INDF0
MOVFP REMBl, INDF0
DECFSZ TESTCOUNT
GOTO D1616LOOP
RETLW 0x00
; Test suite for FXD1608
TFXD1608
MOVWL 34
MOVFP WREG, TESTCOUNT
MOVFP WREG, NUMTESTS
MOVWL 3
MOVFP WREG, TESTCODE
D1608LOOP
CALL RANDOM16
;  
SWAPF RANDHI
;  
SWAPF RANDLO
MOVFP RANDHI, WREG
MOVFP WREG, BARGB0
BCF BARGB0, MSB
MOVFP BARGB0, INDF0
CALL RANDOM16
;  
SWAPF RANDHI
;  
SWAPF RANDLO
MOVFP RANDHI, WREG
MOVFP WREG, AARGB0
MOVFP RANDLO, WREG
MOVFP WREG, AARGB1
;  
BCF AARGB0, MSB
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
CALL FXD1608S
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
MOVFP REMBO, INDF0
DECFSZ TESTCOUNT
GOTO D1608LOOP
RETLW 0x00
; Test suite for FXD0808
TFXD0808
MOVWL 52
MOVFP WREG, TESTCOUNT
MOVFP WREG, NUMTESTS
MOVWL 4
MOVFP WREG, TESTCODE
D0808LOOP
CALL RANDOM16
;  
MOVFP RANDHI, WREG
MOVFP WREG, BARGB0
;  
BCF BARGB0, MSB
MOVFP BARGB0, INDF0
MOVFP RANDLO, WREG
MOVFP WREG, AARGB0
;  
BCF AARGB0, MSB
MOVFP AARGB0, INDF0
CALL FXD0808S
MOVFP AARGB0, INDF0
MOVFP REMBO, INDF0
DECFSZ TESTCOUNT
GOTO D808LOOP
RETLW 0x00

******************************************************************************************
*******24/16 Bit Division Macros

SDIV2416 macro
; Max Timing: 5+8+22*12+6 = 283 clks
; Min Timing: 5+8+22*11+3 = 258 clks
; PM: 5+8+22*14+6 = 327 DM: 8

variable i

MOVFP BARGBl,WREG
SUBWF REMBl
MOVFP BARGBO,WREG
SUBWFB REMBO
RLCF ACCBO
RLCF ACCBO,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
ADDWF REMBl
MOVFP BARGBO,WREG
ADDWFC REMBO
RLCF ACCB0

i = 2
while i < 8

RLCF ACCBO,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
BTFSS ACCBO,LSB
GOTO SADD46#v(i)
SUBWF REMBl
MOVFP BARGBO,WREG
GOTO SOK46#v(i)

SADD46#v(i) ADDWF REMBl
MOVFP BARGBO,WREG
SOK46#v(i) RLCF ACCBO
i:;;;;i+l
endw

RLCF ACCB0,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
BTFSS ACCBO,LSB
GOTO SADD468
SUBWF REMBl
MOVFP BARGBO,WREG
GOTO SOK468

SADD468 ADDWF REMBl
MOVFP BARGBO,WREG
SOK468 RLCF ACCB1

i = 9
while i < 16

RLCF ACCB1,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
BTFSS ACCB1,LSB
GOTO SADD46#v(i)
SUBWF REMBl
MOVFP BARGBO,WREG
SUBWFB REMB0
GOTO SOK46#v(i)
ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
SOK46#v(i)
RLCF ACCB1
i=i+1
endw
RLCF ACCB2,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
BTFSS ACCB1,LSB
GOTO SADD4616
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMBO
GOTO SOK4616
SADD4616
ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
SOK4616
i = 17
while i < 24
RLCF ACCB2,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
BTFSS ACCB2,LSB
GOTO SADD46#v(i)
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMBO
GOTO SOK46#v(i)
SADD46#v(i)
ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
SOK46#v(i)
RLCF ACCB2
i=i+1
endw
BTFSC ACCB2,LSB
GOTO SOK46
MOVFP BARGB1,WREG
ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
SOK46
endm
UDIV2416
macro
; restore = 15/20 clks, nonrestore = 11/14 clks
; Max Timing: 16*15+1*8*20 = 401 clks
; Min Timing: 16*11+1*8*14 = 289 clks
; PM: 16*15+1*8*20 = 401
variable i
i = 0
while i < 8
RLCF ACCB0,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMBO
BTFSC _C
GOTO SOK46#v(i)
MOVFP BARGB1,WREG
ADDWF REMB1
MOVFP BARGBO,WREG
ADDWF REMB0
BCF _C
UOK46#v(i)
RLCF ACCB0
i = i+1
endw
while i < 16
    RLCF ACCB1,W
    RLCF REMB1
    RLCF REMB0
    MOVFP BARGB1,WREG
    SUBWF REMB1
    MOVFP BARGB0,WREG
    SUBWFB REMB0
    BTFSC _C
    GOTO UOK46#v(i)
    MOVFP BARGB1,WREG
    ADDWF REMB1
    MOVFP BARGB0,WREG
    ADDWFC REMB0
    BCF _C
UOK46#v(i)
    RLCF ACCB1
    i = i+1
endw
CLRFB TEMP
while i < 24
    RLCF ACCB2,W
    RLCF REMB1
    RLCF REMB0
    RLCF TEMP
    MOVFP BARGB1,WREG
    SUBWF REMB1
    MOVFP BARGB0,WREG
    SUBWFB REMB0
    CLRF WREG
    SUBWFB TEMP
    BTFSC _C
    GOTO UOK46#v(i)
    MOVFP BARGB1,WREG
    ADDWF REMB1
    MOVFP BARGB0,WREG
    ADDWFC REMB0
    CLRFB WREG
    ADDWFC TEMP
    BCF _C
UOK46#v(i)
    RLCF ACCB2
    i = i+1
endw
endm

NDIV2416

; Max Timing: 10+23*15+6 = 361 clks
; Min Timing: 10+23*14+3 = 335 clks
; PW: 10+23*19+6 = 450 DM: 8
variable i
RLCF ACCB0,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
CLRFB TEMP,W
SUBWFB TEMP
RLCF ACCB0
i = 1
while i < 8
RLCF ACCB0,W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1,WREG
BTFSS ACCB0,LSB
GOTO NADD46#v(i)
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
GOTO NOK46#v(i)
NADD46#v(i)
ADDFP REMB1
MOVFP BARGB0,WREG
ADDFC REMB0
CLRF WREG
ADDFC TEMP
NOK46#v(i)
RLCF ACCB0
i = i + 1
endw
RLCF ACCB1,W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1,WREG
BTFSS ACCB1,LSB
GOTO NADD468
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
GOTO NOK468
NADD468
ADDFP REMB1
MOVFP BARGB0,WREG
ADDFC REMB0
CLRF WREG
ADDFC TEMP
NOK468
RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1,W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1,WREG
BTFSS ACCB1,LSB
GOTO NADD46#v(i)
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
GOTO NOK46#v(i)
NADD46#v(i)
ADDFP REMB1
MOVFP BARGB0,WREG
ADDFC REMB0
CLRF WREG
ADDFC TEMP
NOK46\#v(i) RLCF ACCB1
i=i+1
endw
RLCF ACCB2,W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1,WREG
BTFSS ACCB1,LSB
GOTO NADD4616
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
GOTO NOK4616
NADD4616 ADDWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
CLRF WREG
ADDWFC TEMP
NOK4616 RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2,W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1,WREG
BTFSS ACCB2,LSB
GOTO NADD46\#v(i)
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
GOTO NOK46\#v(i)
NADD46\#v(i) ADDWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
CLRF WREG
ADDWFC TEMP
NOK46\#v(i) RLCF ACCB2
i=i+1
endw
BTFSC ACCB2,LSB
GOTO NOK46
MOVFP BARGB1,WREG
ADDWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
NOK46 endm
UDIV2415 macro
; Max Timing: 8+23*12+6 = 290 clks
; Min Timing: 8+23*11+3 = 264 clks
; PM: 8+23*14+6 = 336 DM: 8
variable i
RLCF ACCB0,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0
NOK46 endm
AN617

SUBWFB REMBO
RLCF ACCB0

i = 1

while i < 8

RLCF ACCB0, W
RLCF REMB1
RLCF REMB0
MOVFPP BARGB1, WREG
BTFSS ACCB0, LSB
GOTO UADD45#v(i)
SUBWF REMB1
MOVFPP BARGB0, WREG
SUBWFB REMB0
GOTO UOK45#v(i)

UADD45#v(i)

ADDWF REMB1
MOVFPP BARGB0, WREG
ADDWF REMB0

UOK45#v(i)

i = i + 1
endw

RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFPP BARGB1, WREG
BTFSS ACCB1, LSB
GOTO UADD458
SUBWF REMB1
MOVFPP BARGB0, WREG
SUBWFB REMB0
GOTO UOK458

UADD458

ADDWF REMB1
MOVFPP BARGB0, WREG
ADDWF REMB0

UOK458

RLCF ACCB1

i = 9

while i < 16

RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFPP BARGB1, WREG
BTFSS ACCB1, LSB
GOTO UADD45#v(i)
SUBWF REMB1
MOVFPP BARGB0, WREG
SUBWFB REMB0
GOTO UOK45#v(i)

UADD45#v(i)

ADDWF REMB1
MOVFPP BARGB0, WREG
ADDWF REMB0

UOK45#v(i)

i = i + 1
endw

RLCF ACCB2, W
RLCF REMB1
RLCF REMB0
MOVFPP BARGB2, WREG
BTFSS ACCB2, LSB
GOTO UADD4516
SUBWF REMB1
MOVFPP BARGB0, WREG
SUBWFB REMB0
GOTO UOK4516

UADD4516

ADDWF REMB1
MOVFPP BARGB0, WREG
ADDWF REMB0

UOK4516

RLCF ACCB2
i = 17
while i < 24
  RLCF ACCB2, W
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB1, WREG
  BTFS ACCB2, LSB
  GOTO UADD45#v(i)
  SUBWF REMB1
  MOVFP BARGB0, WREG
  SUBWFB REMB0
  GOTO UOK45#v(i)
UADD45#v(i) ADDWF REMB1
  MOVFP BARGB0, WREG
  ADDWFC REMB0
UOK45#v(i) RLCF ACCB2
i = i + 1
endw
BTFS ACCB2, LSB
GOTO UOK45
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
UOK45 endm
UDIV2315 macro
; Max Timing: 5+8+22*12+6 = 283 clks
; Min Timing: 5+8+22*11+3 = 258 clks
; PM: 5+8+22*14+6 = 327 DM: 8
variable i
  MOVFP BARGB1, WREG
  SUBWF REMB1
  MOVFP BARGB0, WREG
  SUBWFB REMB0
  RLCF ACCB0
  RLCF ACCB0, W
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB1, WREG
  BTFS ACCB0, LSB
  GOTO UADD35#v(i)
  SUBWF REMB1
  MOVFP BARGB0, WREG
  SUBWFB REMB0
  GOTO UOK35#v(i)
UADD35#v(i) ADDWF REMB1
  MOVFP BARGB0, WREG
  ADDWFC REMB0
UOK35#v(i) RLCF ACCB0
i = 2
while i < 8
  RLCF ACCB0, W
  RLCF REMB1
  RLCF REMB0
  MOVFP BARGB1, WREG
  BTFS ACCB0, LSB
  GOTO UADD35#v(i)
  SUBWF REMB1
  MOVFP BARGB0, WREG
  SUBWFB REMB0
  GOTO UOK35#v(i)
UADD35#v(i) ADDWF REMB1
  MOVFP BARGB0, WREG
  ADDWFC REMB0
UOK35#v(i) RLCF ACCB0
i = i + 1
endw
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB0, LSB
GOTO UADD358
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK358
UADD358 ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
UOK358 RLCF ACCB1

while i < 16
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB1, LSB
GOTO UADD35#v(i)
SUBWF REMB1
MOVFP BARGBO, WREG
SUBWFB REMB0
GOTO UOK35#v(i)
UADD35#v(i) ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
UOK35#v(i) RLCF ACCB1
i=i+1
endw

while i < 24
RLCF ACCB2, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB2, LSB
GOTO UADD35#v(i)
SUBWF REMB1
MOVFP BARGBO, WREG
SUBWFB REMB0
GOTO UOK3516
UADD3516 ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
UOK3516 RLCF ACCB2

while i < 32
RLCF ACCB3, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB3, WREG
BTFSS ACCB3, LSB
GOTO UADD35#v(i)
SUBWF REMB1
MOVFP BARGBO, WREG
SUBWFB REMB0
GOTO UOK3516
UADD3516 ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
UOK3516 RLCF ACCB2
; 16/16 Bit Division Macros

SDIV1616 macro

; Max Timing: 5+8+14*12+6 = 187 clks
; Min Timing: 5+8+14*11+6 = 173 clks
; PM: 5+8+14*14+6 = 215  DM:  6

variable i

MOVFP BARGBl,WREG
SUBWF REMBl
MOVFP BARGBO, WREG
SUBWF REMBO
RLCF ACCB0
RLCF REMBO
MOVFP BARGBl,WREG
ADDFW REMBl
MOVFP BARGBO, WREG
ADDFWC REMBO
RLCF ACCB0

i = 2
while i < 8

RLCF ACCB0,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
BTFSS ACCB0,LSB
GOTO SADD66#v(i)
SUBWF REMBl
MOVFP BARGBO, WREG
SUBWF REMBO
GOTO SOK66#v(i)

SADD66#v(i) ADDWF REMBl
MOVFP BARGBO, WREG
ADDFWC REMBO

SOK66#v(i) RLCF ACCB0
i = i+1
endw

RLCF ACCB1,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
BTFSS ACCB0,LSB
GOTO SADD668
SUBWF REMBl
MOVFP BARGBO, WREG
SUBWF REMBO
GOTO SOK668

SADD668 ADDWF REMBl
MOVFP BARGBO, WREG
ADDFWC REMBO

SOK668 RLCF ACCB1
i = 9
while i < 16

RLCF ACCB1,W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl,WREG
BTFSS ACCB1,LSB
GOTO SADD66#v(i)
SUBWF REMBl
AN617

MOVFP BARGB0, WREG
SUBWFB REMBO
GOTO SOK66#v(i)
SADD66#v(i) ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO

SOK66#v(i) RLCF ACCB1
i = i + 1
endw
BTFSC ACCB1, LSB
GOTO SOK66
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
SOK66
endm

UDIV1616 macro
; restore = 15 clks, nonrestore = 11 clks
; Max Timing: 8*15+8*15 = 240 clks
; Min Timing: 8*11+8*11 = 176 clks
; PM: 8*15+8*15 = 240 DM: 6
variable i
i = 0
while i < 8
RLCF ACCB0, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
BTFSC _C
GOTO UOK66#v(i)
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
BCF _C
UOK66#v(i) RLCF ACCB0
i = i + 1
endw
i = 8
while i < 16
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
BTFSC _C
GOTO UOK66#v(i)
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
BCF _C
UOK66#v(i) RLCF ACCB1
i = i + 1
endw
endm

NDIV1616 macro
; Max Timing: 9+15*15+6 = 240 clks
; Min Timing: 9+15*14+6 = 225 clks
; PM: 9+15*19+6 = 300
DM: 7

variable i

```
; RLCF ACCB0, W
RLCF REMB1
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
CLRF TEMP, W
SUBWFB TEMP
RLCF ACCB0
i = 1
while i < 8
RLCF ACCB0, W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1, WREG
BTFSS ACCB0, LSB
GOTO NADD66#v(i)
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK66#v(i)
NADD66#v(i) ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
CLRF WREG
ADDWFC TEMP
NOK66#v(i) RLCF ACCB0
i = i + 1
endw
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1, WREG
BTFSS ACCB0, LSB
GOTO NADD668
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK668
NADD668 ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
CLRF WREG
ADDWFC TEMP
NOK668 RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB1, WREG
BTFSS ACCB1, LSB
GOTO NADD66#v(i)
SUBWF REMB1
MOVFP BARGB0, WREG
```
AN617

SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
GOTO NOK66#v(i)

NADD66#v(i)
ADDWF REMB1
MOVFPP BARGBO,WREG
ADDWFC REMB0
CLRF WREG
ADDWFC TEMP

NOK66#v(i)
RLCF ACCB1
i=i+1
endw
BTFSC ACCB1,LSB
GOTO NOK66
MOVFPP BARGB1,WREG
ADDWF REMB1
MOVFPP BARGBO,WREG
ADDWFC REMB0

NOK66
endm

UDIV1615
macro
; Max Timing: 7+15*12+6  = 193 clks
; Min Timing: 7+15*11+6 = 178 clks
; PM: 7+15*14+6 = 213
variable i
RLCF ACCB0,W
RLCF REMB1
MOVFPP BARGB1,WREG
SUBWFB REMB0
SUBWFB REMB0
RLCF ACCB0
i = 1
while i < 8
RLCF ACCB0,W
RLCF REMB1
RLCF REMB0
MOVFPP BARGB1,WREG
BTFSS ACCB0,LSB
GOTO UADD65#v(i)
SUBWFB REMB1
MOVFPP BARGBO,WREG
SUBWFB REMB0
GOTO UOK65#v(i)

UADD65#v(i)
ADDWF REMB1
MOVFPP BARGBO,WREG
ADDWFC REMB0

UOK65#v(i)
RLCF ACCB0
i=i+1
endw
RLCF ACCB1,W
RLCF REMB1
RLCF REMB0
MOVFPP BARGB1,WREG
BTFSS ACCB0,LSB
GOTO UADD658
SUBWFB REMB1
MOVFPP BARGBO,WREG
SUBWFB REMB0
GOTO UOK658

UADD658
ADDWF REMB1
MOVFPP BARGBO,WREG
ADDWFC REMB0

© 1995 Microchip Technology Inc.
UOK65
RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB1, LSB
GOTO UADD65#v(i)
SUBWF REMB1
MOVFP BARGBO, WREG
SUBWFB REMB0
GOTO UOK65#v(i)
UADD65#v(i) ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
UOK65#v(i) RLCF ACCB1
i = i + 1
endw
BTFSC ACCB1, LSB
GOTO UOK65
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
RLCF ACCB0
i = 2
while i < 8
RLCF ACCB0, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB0, LSB
GOTO UADD55#v(i)
SUBWF REMB1
MOVFP BARGBO, WREG
SUBWFB REMB0
GOTO UOK55#v(i)
UADD55#v(i) ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMB0
UOK55#v(i) RLCF ACCB0
i = i + 1
endw
RLCF ACCB1, W
Extra 16 Bit Divide Macros

DIV1616 macro
        ; Timing: restore = 16 clks, nonrestore = 13 clks
        ; variable i
        ; i = 0
        ; while i < 16
        "RLCF AARGBl"
        "RLCF AARGBO"
        "RLCF REMBl"
        "RLCF REMBO"
        "MOVFP BARGBl, WREG"
        "SUBWF REMBl"
        "MOVFP BARGBO, WREG"
        "SUBWFB REMBO"
        "BTFSS _c"
        "GOTO RS1616_#v(i)"
        "BSF AARGBl, LSB"
        "GOTO OK1616_#v(i)"
        "RS1616_#v(i)"
        "MOVFP BARGBl, WREG"
        "ADDWF REMBl"
        "MOVFP BARGBO, WREG"
        "ADDWF REMBO"
        "BCF AARGBl, LSB"
DIVMAC

: Timing: restore = 19 clks, nonrestore = 14 clks

16*19 = 304 clks

variable i

i = 0

while i < 16

RLCF AARGB1
RLCF AARGB0
RLCF REMB1
RLCF REMB0
MOVFP BARGB0,WREG
SUBWF REMB0,W
BTFSS _Z
GOTO notz#v(i)
MOVFP BARGB1,WREG
SUBWF REMB1,W

notz#v(i)

BTFSS _C
GOTO nosub#v(i)
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
BSF AARGB1,LSB
GOTO ok#v(i)

nosub#v(i)

BCF AARGB1,LSB

ok#v(i)

i=i+1
endw
endm

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

; 16/08 BIT Division Macros

SDIV1608

macro

Max Timing: 3+5+14*8+2 = 122 clks
Min Timing: 3+5+14*8+2 = 122 clks
PM: 3+5+14*8+2 = 122
DM: 4

variable i

MOVFP BARGB0,WREG
SUBWF REMB0
RLCF ACCB0
RLCF ACCB0,W
RLCF REMB0
MOVFP BARGB0,WREG
ADDWF REMB0
RLCF ACCB0

i = 2

while i < 8

RLCF ACCB0,W
RLCF REMB0
MOVFP BARGB0,WREG
BTFSC ACCB0,LSB
SUBWF REMB0
BTFSS ACCB0,LSB
ADDWF REMB0
RLCF ACCB0

i=i+1
endw
endw

RLCF ACCB1,W
RLCF REMB0
MOVFP BARGB0,WREG
BTFSC ACCB0,LSB
SUBWF REMB0
AN617

BTFSS ACCB0, LSB
ADDWF REMB0
RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1, W
RLCF REMB0
MOVFP BARGBO, WREG
BTFSC ACCB1, LSB
SUBWF REMB0
BTFSS ACCB1, LSB
ADDWF REMB0
RCF ACCB1
i = i + 1
endw
BTFSS ACCB1, LSB
ADDWF REMB0
endm

UDIV1608

; restore = 9/15 clks, nonrestore = 8/11 clks
; Max Timing: 8*9+1+8*15 = 193 clks max
; Min Timing: 8*8+1+8*11 = 153 clks min
; PM: 8*9+1+8*15 = 193 DM: 4
variable i
i = 0
while i < 8
RLCF ACCB0, W
RLCF REMB0
MOVFP BARGBO, WREG
SUBWF REMB0
BTFSC _C
GOTO UOK68#v(i)
ADDWF REMB0
BCF _C
UOK68#v(i)
RLCF ACCB0
i = i + 1
endw
CLRF TEMP
i = 8
while i < 16
RLCF ACCB1, W
RLCF REMB0
RLCF TEMP
MOVFP BARGBO, WREG
SUBWF REMB0
CLRWF WREG
SUBWFB TEMP
BTFSC _C
GOTO UOK68#v(i)
MOVFP BARGBO, WREG
ADDWF REMB0
CLRWF WREG
ADDWFC TEMP
BCF _C
UOK68#v(i)
RLCF ACCB1
i = i + 1
endw
endm

NDIV1608

; Max Timing: 7+15*12+3 = 190 clks
; Min Timing: 7+15*11+3 = 175 clks
; PM: 7+15*14+3 = 220 DM: 5
variable i
RLCF ACCB0, W
RLCF REMB0
MOVFP BARGBO, WREG

© 1995 Microchip Technology Inc.
AN617

SUBWF REMBO
CLRF TEMP,W
SUBWFB TEMP
RLCF ACCBO

i = 1

while i < 8

RLCF ACCBO,W
RLCF REMBO
RLCF TEMP
MOVFP BARGBO,WREG
BTFSS ACCBO,LSB
GOTO NADD68#v(i)
SUBWF REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK68#v(i)

NADD68#v(i)
ADDWF REMBO
CLRF WREG
ADDWFC TEMP

NOK68#v(i)
RLCF ACCBO

i = i + 1
endw

RLCF ACCBO,W
RLCF REMBO
RLCF TEMP
MOVFP BARGBO,WREG
BTFSS ACCBO,LSB
GOTO NADD688
SUBWF REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK688

NADD688
ADDWF REMBO
CLRF WREG
ADDWFC TEMP

NOK688
RLCF ACCBO1

i = 9

while i < 16

RLCF ACCBO1,W
RLCF REMBO
RLCF TEMP
MOVFP BARGBO,WREG
BTFSS ACCBO1,LSB
GOTO NADD68#v(i)
SUBWF REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK68#v(i)

NADD68#v(i)
ADDWF REMBO
CLRF WREG
ADDWFC TEMP

NOK68#v(i)
RLCF ACCBO1

i = i + 1
endw

BTFSS ACCBO1,LSB
MOVFP BARGBO,WREG
ADDWF REMBO
endm

UDIV1607

; Max Timing: 5+15*8+2 = 127 clks
; Min Timing: 5+15*8+2 = 127 clks
; PM: 5+15*8+2 = 127 DM: 4

variable i

RLCF ACCBO,W
RLCF REMBO
MOVFP BARGBO,WREG
AN617

```
SUBWF  REMBO
RLCF   ACCBO
i = 1
while i < 8
  RLCF  ACCBO,W
  RLCF  REMBO
  MOVFP BARGBO,WREG
  BTFSC ACCBO,LSB
  SUBWF  REMBO
  BTFSS ACCBO,LSB
  ADDWF  REMBO
  RLCF  ACCBO
  i=i+1
endw
  RLCF  ACCBO,W
  RLCF  REMBO
  MOVFP BARGBO,WREG
  BTFSS ACCBO,LSB
  ADDWF  REMBO
  RLCF  ACCBO
  i = 9
while i < 16
  RLCF  ACCBO,W
  RLCF  REMBO
  MOVFP BARGBO,WREG
  BTFSC ACCBO,LSB
  SUBWF  REMBO
  BTFSS ACCBO,LSB
  ADDWF  REMBO
  RLCF  ACCBO
  i=i+1
endw
BTFSS ACCBO,LSB
ADDWF  REMBO
endm
UDIV1507
; Max Timing:   3+5+14*8+2 = 122 clks
; Min Timing:   3+5+14*8+2 = 122 clks
; PM: 3+5+14*8+2 = 122    DM: 4
variable i
  MOVFP BARGBO,WREG
  SUBWF  REMBO
  RLCF  ACCBO
  RLCF  ACCBO,W
  RLCF  REMBO
  MOVFP BARGBO,WREG
  ADDWF  REMBO
  RLCF  ACCBO
  i = 2
while i < 8
  RLCF  ACCBO,W
  RLCF  REMBO
  MOVFP BARGBO,WREG
  BTFSC ACCBO,LSB
  SUBWF  REMBO
  BTFSS ACCBO,LSB
  ADDWF  REMBO
  RLCF  ACCBO
  i=i+1
endw
  RLCF  ACCBO,W
  RLCF  REMBO
  MOVFP BARGBO,WREG
  BTFSC ACCBO,LSB
```

© 1995 Microchip Technology Inc.
SUBWF REMBO
BTFSS ACCBO,LSB
ADDWF REMBO
RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1,W
RLCF REMBO
MOVF BARGBO,WREG
BTFSC ACCBO,LSB
SUBWF REMBO
BTFSS ACCBO,LSB
ADDWF REMBO
RLCF ACCB1
i = i+1
endw
BTFSS ACCBO,LSB
ADDWF REMBO
endm

;**************************************************************************************
;**************************************************************************************
; 08/08 BIT Division Macros
SDIV0808 macro
; Max Timing: 3+5+6*8+2  = 58 clks
; Min Timing: 3+5+6*8+2  = 58 clks
; PM: 3+5+6*8+2  = 58        DM: 3
variable i
MOVFP BARGBO, WREG
SUBWF REMBO
RLCF ACCB0
RLCF ACCB0,W
RLCF REMBO
MOVF BARGBO,WREG
ADDWF REMBO
RLCF ACCB0
i = 2
while i < 8
RLCF ACCB0,W
RLCF REMBO
MOVF BARGBO,WREG
BTFSC ACCB0,LSB
SUBWF REMBO
BTFSS ACCB0,LSB
ADDWF REMBO
RLCF ACCB0
i = i+1
endw
BTFSS ACCB0,LSB
ADDWF REMBO
endm
UDIV0808 macro
; restore = 9 clks, nonrestore = 8 clks
; Max Timing: 8*9 = 72 clks        max
; Min Timing: 8*8 = 64 clks        min
; PM: 8*9 = 72         DM: 3
variable i
i = 0
while i < 8
RLCF ACCB0,W
RLCF REMBO
MOVF BARGBO,WREG
SUBWF REMBO
BTFSC \_C
GOTO UOK88\#v(i)
ADDWF REMBO
UDIV0807
macro
; Max Timing: 5+7*8+2 = 63 clks
; Min Timing: 5+7*8+2 = 63 clks
; PM: 5+7*8+2 = 63 DM: 3
variable i
RLCF ACCB0
RLCF RMB0
MOVFP BARGBO,WREG
SUBWF RMB0
RLCF ACCB0
i = 1
while i < 8
RLCF ACCB0
RLCF RMB0
MOVFP BARGBO,WREG
BTFSC ACCB0,LSB
SUBWF RMB0
BTFSS ACCB0,LSB
ADDWF RMB0
RLCF ACCB0
i = i+1
endw
BTFSS ACCB0,LSB
ADDWF RMB0
endm

UDIV0707
macro
; Max Timing: 3+5+6*8+2 = 58 clks
; Min Timing: 3+5+6*8+2 = 58 clks
; PM: 3+5+6*8+2 = 58 DM: 3
variable i
MOVFP BARGBO,WREG
SUBWF RMB0
RLCF ACCB0
RLCF ACCB0,W
RLCF RMB0
MOVFP BARGBO,WREG
ADDWF RMB0
RLCF ACCB0
i = 2
while i < 8
RLCF ACCB0,W
RLCF RMB0
MOVFP BARGBO,WREG
BTFSC ACCB0,LSB
SUBWF RMB0
BTFSS ACCB0,LSB
ADDWF RMB0
RLCF ACCB0
i = i+1
endw
BTFSS ACCB0,LSB
ADDWF RMB0
endm

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

; 24/16 Bit Signed Fixed Point Divide 24/16 -> 24.16
; Input: 24 bit fixed point dividend in AARGBO, AARGBl, AARGB2
; 16 bit fixed point divisor in BARGBO, BARGBl
; Use: CALL FXD2416S
; Output: 24 bit fixed point quotient in AARGBO, AARGBl, AARGB2
16 bit fixed point remainder in REMBO, REMBl
Result: AARG, REM <-- AARG / BARG
Max Timing: 11+283+3 = 297 clks  A > 0, B > 0
14+283+15 = 312 clks  A > 0, B < 0
16+283+15 = 314 clks  A < 0, B > 0
19+283+3 = 305 clks  A < 0, B < 0
Min Timing: 11+258+3 = 272 clks  A > 0, B > 0
14+258+15 = 287 clks  A > 0, B < 0
16+258+15 = 289 clks  A < 0, B > 0
19+258+3 = 280 clks  A < 0, B < 0
PM: 14+327+12 = 353  DM: 8

FXD2416S MOVFP AARGBO,WREG
XORWF BARGBO,W
MOVWF SIGN
CLRF REMBO
CLRF REMBl,W
BTFSS BARGBO,MSB if MSB set go & negate BARG
GOTO CA2416S
COMF BARGB1
COMF BARGB0
INCF BARGBl
ADDFPC BARGB0

CA2416S BTFSS AARGBO,MSB ; if MSB set go & negate ACCa
GOTO C2416S
COMF AARGB2
COMF AARGB1
COMF AARGB0
INCF AARGB2
ADDFPC AARGB1
ADDFPC AARGB0

C2416S SDIV2416
BTFSS SIGN,MSB ; negate (ACCc, ACCd)
RETLW 0x00
COMF AARGB2
COMF AARGB1
COMF AARGB0
CLRWF WREG
INCF AARGB2
ADDFPC AARGB1
ADDFPC AARGB0
COMF REMB1
COMF REMB0
INCF REMB1
ADDFPC REMB0
RETLW 0x00

24/16 Bit Unsigned Fixed Point Divide 24/16 -> 24.16
Input: 24 bit unsigned fixed point dividend in AARGBO, AARGBl, AARGB2
16 bit unsigned fixed point divisor in BARGBO, BARGBl
Use: CALL FXD2416U
Output: 24 bit unsigned fixed point quotient in AARGBO, AARGBl, AARGB2
16 bit unsigned fixed point remainder in REMB0, REMB1
Result: AARG, REM <-- AARG / BARG
Max Timing: 2+361+2 = 365 clks
Min Timing: 2+335+2 = 339 clks
PM: 2+450+1 = 453  DM: 8
FXD2416U CLRWF REMB0
CLRWF REMB1
NDIV2416
RETLW 0x00

24/15 Bit Unsigned Fixed Point Divide 24/15 -> 24.15
AN617

Input: 24 bit unsigned fixed point dividend in AARGBO, AARGBl, AARGB2
15 bit unsigned fixed point divisor in BARGBO, BARGBl
Use: CALL FXD2415U
Output: 24 bit unsigned fixed point quotient in AARGBO, AARGBl, AARGB2
15 bit unsigned fixed point remainder in REMBO, REMBl
Result: AARG, REM <- AARG / BARG
Max Timing: 2+290+2 = 294 clks
Min Timing: 2+264+2 = 268 clks
PM: 2+336+1 = 339 DM: 8

FXD2415U
CLRFR
REMBO
CLRFR
REMBl
UDIV2415
RETLW 0x00

23/15 Bit Unsigned Fixed Point Divide 23/15 -> 23.15
Input: 23 bit unsigned fixed point dividend in AARGBO, AARGBl, AARGB2
15 bit unsigned fixed point divisor in BARGBO, BARGBl
Use: CALL FXD2315U
Output: 23 bit unsigned fixed point quotient in AARGBO, AARGBl, AARGB2
15 bit unsigned fixed point remainder in REMBO, REMBl
Result: AARG, REM <- AARG / BARG
Max Timing: 2+283+2 = 287 clks
Min Timing: 2+258+2 = 262 clks
PM: 2+327+1 = 330 DM: 8

FXD2315U
CLRFR
REMBO
CLRFR
REMBl
UDIV2315
RETLW 0x00

16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16
Input: 16 bit fixed point dividend in AARGBO, AARGBl
16 bit fixed point divisor in BARGBO, BARGBl
Use: CALL FXD1616S
Output: 16 bit fixed point quotient in AARGBO, AARGBl
16 bit fixed point remainder in REMBO, REMBl
Result: AARG, REM <- AARG / BARG
Max Timing: 14+187+12 = 215 clks
Min Timing: 17+173+3 = 193 clks
PM: 14+215+12 = 241 DM: 7

FXD1616S
MOVFP
AARGBO,WREG
XORWF
BARGBO,W
MOVWF
SIGN
CLRFR
REMBO
CLRFR
REMBl,W
BTFSS
BARGBO,MSB ; if MSB set go & negate BARG
GOTO
CA1616S
COMF
BARGBO
COMF
BARGBO
INCF
BARGBl
ADDWF
BARGBO
CA1616S
BTFSS
AARGBO,MSB ; if MSB set go & negate ACCa
GOTO
C1616S
COMF
AARGBO
COMF
AARGBO
INCF
AARGBl
ADDWF
AARGBO

© 1995 Microchip Technology Inc.
C1616S
SDIV1616
BTFSS SIGN,MSB ; negate (ACCc,ACCd)
RETLW 0x00
COMF AARGBl
COMF AARGB0
CLRF WREG
INCF AARGBl
ADDWF AARGB0
COMF REMBl
COMF REMB0
INCF REMBl
ADDWF REMB0
RETLW 0x00
;

; 16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16
; Input: 16 bit unsigned fixed point dividend in AARGBO, AARGBl
; 16 bit unsigned fixed point divisor in BARGBO, BARGBl
; Use: CALL FXD1616U
; Output: 16 bit unsigned fixed point quotient in AARGBO, AARGBl
; 16 bit unsigned fixed point remainder in REMBO, REMBl
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 2+240+2 = 244 clks
; Min Timing: 2+176+2 = 180 clks
; PM: 2+240+1 = 243 DM: 6
FXD1616U CLRF REMB0
CLRF REMB1
UDIV1616
RETLW 0x00
;

; 16/15 Bit Unsigned Fixed Point Divide 16/15 -> 16.15
; Input: 16 bit unsigned fixed point dividend in AARGBO, AARGBl
; 15 bit unsigned fixed point divisor in BARGBO, BARGBl
; Use: CALL FXD1615U
; Output: 16 bit unsigned fixed point quotient in AARGBO, AARGBl
; 15 bit unsigned fixed point remainder in REMBO, REMBl
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 2+193+2 = 197 clks
; Min Timing: 2+178+2 = 182 clks
; PM: 2+213+1 = 216 DM: 6
FXD1615U CLRF REMB0
CLRF REMB1
UDIV1515
RETLW 0x00
;

; 15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15
; Input: 15 bit unsigned fixed point dividend in AARGBO, AARGBl
; 15 bit unsigned fixed point divisor in BARGBO, BARGBl
; Use: CALL FXD1515U
; Output: 15 bit unsigned fixed point quotient in AARGBO, AARGBl
; 15 bit unsigned fixed point remainder in REMBO, REMBl
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 2+187+2 = 191 clks
; Min Timing: 2+173+2 = 177 clks
; PM: 2+215+1 = 218 DM: 6
FXD1515U CLRF REMB0
CLRF REMB1
UDIV1515
RETLW 0x00
;
AN617

16/8 Bit Signed Fixed Point Divide 16/08 -> 16.08

Input: 16 bit fixed point dividend in AARGBO, AARGBl
8 bit fixed point divisor in BARGBO

Use: CALL FXD1608S

Output: 16 bit fixed point quotient in AARGBO, AARGBl
8 bit fixed point remainder in REMBO

Result: AARG, REM <-- AARG / BARG

Max Timing: 10+122+3 = 135 clks A > 0, B > 0
11+122+11 = 144 clks A > 0, B < 0
13+122+11 = 146 clks A < 0, B > 0
14+122+3 = 139 clks A < 0, B < 0

Min Timing: 10+122+3 = 135 clks A > 0, B > 0
11+122+11 = 144 clks A > 0, B < 0
13+122+11 = 146 clks A < 0, B > 0
14+122+3 = 139 clks A < 0, B < 0

PM: 14+122+10 = 146 DM: 5

FXD1608S
MOVFP AARGBO, WREG
XORWF BARGBO, W
MOVWF SIGN
CLRF REMBO, W
BTFSS BARGBO, MSB ; if MSB set go & negate BARG
GOTO CA1608S

COMP BARGBO
INCF BARGBO
CA1608S
BTFSS AARGBO, MSB ; if MSB set go & negate ACCa
GOTO C1608S

COMP AARGBl
COMP AARGBl
INCF AARGBl
ADDWFC AARGBl

C1608S
SDIV1608
BTFSS SIGN, MSB ; negate (ACCc, ACCd)
RETLW 0x00
COMP AARGBl
COMP AARGBl
CLRF WREG
INCF AARGBl
ADDWFC AARGBl
INCF REMBO
RETLW 0x00

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

16/8 Bit Unsigned Fixed Point Divide 16/08 -> 16.08

Input: 16 bit unsigned fixed point dividend in AARGBO, AARGBl
8 bit unsigned fixed point divisor in BARGBO

Use: CALL FXD1608U

Output: 16 bit unsigned fixed point quotient in AARGBO, AARGBl
8 bit unsigned fixed point remainder in REMBO

Result: AARG, REM <-- AARG / BARG

Max Timing: 1+193+2 = 196 clks
Min Timing: 1+153+2 = 156 clks
PM: 1+193+1 = 195 DM: 4

FXD1608U
CLRF REMBO
UDIV1608
RETLW 0x00

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

16/7 Bit Unsigned Fixed Point Divide 16/07 -> 16.07

Input: 16 bit unsigned fixed point dividend in AARGBO, AARGBl
7 bit unsigned fixed point divisor in BARGBO

Use: CALL FXD1607U

Output: 16 bit unsigned fixed point quotient in AARGBO, AARGBl

;**********************************************************************************************
;**********************************************************************************************
7 bit unsigned fixed point remainder in REMBO

Result: AARG, REM <-- AARG / BARG

Max Timing: 1+127+2 = 130 clks
Min Timing: 1+127+2 = 130 clks
PM: 1+127+1 = 129 DM: 4

FXD1607U
CLRF REMBO
UDIV1607
RETLW 0x00

15/7 Bit Unsigned Fixed Point Divide 15/07 -> 15.07

Input: 15 bit unsigned fixed point dividend in AARGBO, AARGBl
7 bit unsigned fixed point divisor in BARGBO

Use: CALL FXD1507U

Output: 15 bit unsigned fixed point quotient in AARGBO, AARGBl
7 bit unsigned fixed point remainder in REMBO

Result: AARG, REM <-- AARG / BARG

Max Timing: 1+122+2 = 125 clks
Min Timing: 1+122+2 = 125 clks
PM: 1+122+1 = 124 DM: 4

FXD1507U
CLRF REMBO
UDIV1507
RETLW 0x00

8/8 Bit Signed Fixed Point Divide 08/08 -> 08.08

Input: 8 bit fixed point dividend in AARGBO
8 bit fixed point divisor in BARGBO

Use: CALL FXD0808S

Output: 8 bit fixed point quotient in AARGBO
8 bit fixed point remainder in REMBO

Result: AARG, REM <-- AARG / BARG

Max Timing: 10+58+3 = 71 clks A > 0, B > 0
11+58+8 = 77 clks A > 0, B < 0
11+58+8 = 77 clks A < 0, B > 0
12+58+3 = 73 clks A < 0, B < 0

Min Timing: 10+58+3 = 71 clks A > 0, B > 0
11+58+8 = 77 clks A > 0, B < 0
11+58+8 = 77 clks A < 0, B > 0
12+58+3 = 71 clks A < 0, B < 0

PM: 12+58+7 = 77 DM: 4

FXD0808S
MOVFP AARGBO,WREG
XORWF BARGBO,W
MOVWF SIGN
CLRF REMBO,W
BTFSS BARGBO,MSB
GOTO CA0808S
COMF BARGBO
INCF BARGBO
BTFSS AARGBO,MSB
GOTO C0808S
COMF AARGBO
INCF AARGBO
SDIV0808
BTFSS SIGN,MSB
RETLW 0x00
COMF AARGBO
INCF AARGBO
COMF REMBO
INCF REMBO
RETLW 0x00

8/8 Bit Unsigned Fixed Point Divide 08/08 -> 08.08
Input: 8 bit unsigned fixed point dividend in AARGBO
8 bit unsigned fixed point divisor in BARGBO
Use: CALL FXD0808U
Output: 8 bit unsigned fixed point quotient in AARGBO
8 bit unsigned fixed point remainder in REMBO
Result: AARG, REM <-- AARG
I
BARG
Max Timing: 1+72+2 = 75 clks
Min Timing: 1+64+2 = 67 clks
PM: 1+72+1 = 74
DM: 3
FXD0808U CLRF REMBO
UDIV0808 RETLW 0x00

8/7 Bit Unsigned Fixed Point Divide 08/07 -> 08.07
Input: 8 bit unsigned fixed point dividend in AARGBO
7 bit unsigned fixed point divisor in BARGBO
Use: CALL FXD0807U
Output: 8 bit unsigned fixed point quotient in AARGBO
7 bit unsigned fixed point remainder in REMBO
Result: AARG, REM <-- AARG
I
BARG
Max Timing: 1+63+2 = 66 clks
Min Timing: 1+63+2 = 66 clks
PM: 1+63+1 = 65
DM: 3
FXD0807U CLRF REMBO
UDIV0807 RETLW 0x00

7/7 Bit Unsigned Fixed Point Divide 07/07 -> 07.07
Input: 7 bit unsigned fixed point dividend in AARGBO
7 bit unsigned fixed point divisor in BARGBO
Use: CALL FXD0707U
Output: 7 bit unsigned fixed point quotient in AARGBO
7 bit unsigned fixed point remainder in REMBO
Result: AARG, REM <-- AARG
I
BARG
Max Timing: 1+58+2 = 61 clks
Min Timing: 1+58+2 = 61 clks
PM: 1+58+1 = 60
DM: 3
FXD0707U CLRF REMBO
UDIV0707 RETLW 0x00

F.3 PIC17CXX Fixed Point Divide Routines C

PIC17 FIXED POINT DIVIDE ROUTINES C VERSION 1.8
Input: fixed point arguments in AARG and BARG
Output: quotient AARG/BARG followed by remainder in REM
All timings are worst case cycle counts
It is useful to note that the additional unsigned routines requiring a non-power of two
argument can be called in a signed divide application where it is known that the
respective argument is nonnegative, thereby offering some improvement in
performance.
Routine Clocks Function
FXD3216S 414 32 bit/16 bit -> 32.16 signed fixed point divide
FXD3216U 485 32 bit/16 bit -> 32.16 unsigned fixed point divide
FXD3215U 390 32 bit/15 bit -> 32.15 unsigned fixed point divide
FXD3115U 383 31 bit/15 bit -> 31.15 unsigned fixed point divide
FXD2424S 390 24 bit/24 bit -> 24.24 signed fixed point divide
FXD2424U 440 24 bit/24 bit -> 24.24 unsigned fixed point divide
FXD2423U 369 24 bit/23 bit -> 24.23 unsigned fixed point divide
FXD2323U 361 23 bit/23 bit -> 23.23 unsigned fixed point divide
list r=dec,x=on,t=off,p=17C42
include <PIC17.INC> ; general PIC17 definitions
include <MATH17.INC> ; PIC17 math library definitions

; Test suite storage
RANDHI equ 0x2B ; random number generator registers
RANDLO equ 0x2C
TESTCODE equ 0x2D ; integer code labeling test contained in following data
NUMTESTS equ 0x2E ; number of tests contained in following data
TESTCOUNT equ 0x2F ; counter
DATA equ 0x30 ; beginning of test data

; Test suite for fixed point divide algorithms

org 0x0021

MAIN
MOVLW RAMSTART
MOVWF WRREG,FSR0

MEMLOOP
CLRF INDFO
INCFSZ FSRO
GOTO MEMLOOP

BSF RTCSTA,5

MOVF H RANDHI
MOVF H RANDLO

MOVF H RTCCH,WREG
MOVF H 0x45 ; seed for random numbers

MOVF H WREG, RANDHI
MOVF H RANDHI

MOVF H WREG, RANDLO
MOVF H RANDLO

BCF _FS1
BSF _FSO

CALL TFXD3216
CALL TFXD424

SELF
GOTO SELF

RANDOM16
RLCF RANDHI,W
RLCF RANDHI,W

XORWF RANDHI,W

RLCF WREG
SWAPF RANDHI

SWAPF RANDLO,W

RLNCF WREG
XORWF RANDHI,W

SWAPF RANDHI

ANDLW OxOl

RLCF RANDLO

RLCF RANDHI

RET LW 0

; Test suite for FXD3216

TFXD3216
MOVLW 17

MOVF H WREG,TESTCOUNT

MOVF H WREG,NUMTESTS

MOVLW 1

MOVF H WREG,TESTCODE

D3216LOOP

CALL RANDOM16

; SWAPP RANDHI

; SWAPP RANDLO

MOVF H RANDHI, WREG

MOVF H WREG, BARG0

MOVF H RANDLO, WREG

MOVF H WREG, BARG1

; BCF BARG0, MSB

MOVF H BARG0, INDFO

MOVF H BARG1, INDFO
CALL RANDOM16
;
SWAPF RANDHI
;
SWAPF RANDLO
MOVFP RANDHI, WREG
MOVFP WREG, AARGB0
MOVFP RANDLO, WREG
MOVFP WREG, AARGB1
;
BCF AARGB0, MSB
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, AARGB2
MOVFP RANDLO, WREG
MOVFP WREG, AARGB3
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP AARGB3, INDF0
CALL FXD3216U
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP AARGB3, INDF0
MOVFP REMBO, INDF0
MOVFP REMBl, INDF0
DECFSZ TESTCOUNT
GOTO D3216LOOP
RETLW 0x00

; Test suite for FXD2424
TFXD2424 MOVLW 17
MOVFP WREG, TESTCOUNT
MOVFP WREG, NUMTESTS
MOVFW 6
MOVFP WREG, TESTCODE
D2424LOOP
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, BARGB0
MOVFP RANDLO, WREG
MOVFP WREG, BARGB1
;
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, BARGB2
;
BCF BARGB0, MSB
MOVFP BARGB0, INDF0
MOVFP BARGB1, INDF0
MOVFP BARGB2, INDF0
;
CALL RANDOM16
MOVFP RANDHI, WREG
MOVFP WREG, AARGB0
MOVFP RANDLO, WREG
MOVFP WREG, AARGB1
;
CALL RANDOM16
MOVFP RANDHI, WREG
;
BCF AARGB0, MSB
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
;
CALL FXD2424S
MOVFP AARGB0, INDF0
MOVFP AARGB1, INDF0
MOVFP AARGB2, INDF0
MOVFP REMB0, INDF0
32/16 Bit Division Macros

SDIV3216 macro

Max Timing: \( 5 + 8 + 30 \times 12 + 6 = 379 \) elks

Min Timing: \( 5 + 8 + 30 \times 11 + 6 = 349 \) elks

PM: \( 5 + 8 + 30 \times 14 + 6 = 439 \) DM: 8

variable \( i \)

```
MOVFP BARGBl, WREG
SUBWF REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
RLCF ACCBO
RLCF ACCBO, W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl, WREG
ADDWF REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
RLCF ACCBO
i = 2
```

while \( i < 8 \)

```
RLCF ACCBO, W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl, WREG
BTFSS ACCBO, LSB
GOTO SADD26#v(i)
SUBWF REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
GOTO SOK26#v(i)
```

```
SADD26#v(i) ADDWF REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK26#v(i) RLCF ACCBO
```

```
i = i + 1
```

endw

```
RLCF ACCBl, W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl, WREG
BTFSS ACCBl, LSB
GOTO SADD268
SUBWF REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK268
```

```
SADD268 ADDWF REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK268 RLCF ACCBl
```

```
i = 9
```

while \( i < 16 \)

```
RLCF ACCBl, W
RLCF REMBl
RLCF REMBO
MOVFP BARGBl, WREG
BTFSS ACCBl, LSB
GOTO SADD26#v(i)
```

```
SADD268 ADDWF REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK268 RLCF ACCBl
```

© 1995 Microchip Technology Inc.
AN617

SUBWF REMB1
MOVFP BARGB0,WRREG
SUBWFB REMB0
GOTO SOK26#v(i)
ADDWF REMB1
MOVFP BARGB0,WRREG
ADDWF REMB0
SOK26#v(i)
RLCF ACCB1
i=i+1
endw
RLCF ACCB2,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WRREG
BTFSS ACCB2,LSB
GOTO SADD2616
SUBWF REMB1
MOVFP BARGB0,WRREG
SUBWFB REMB0
GOTO SOK2616
SADD2616
ADDWF REMB1
MOVFP BARGB0,WRREG
ADDWF REMB0
SOK2616
RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WRREG
BTFSS ACCB2,LSB
GOTO SADD26#v(i)
SUBWF REMB1
MOVFP BARGB0,WRREG
SUBWFB REMB0
GOTO SOK26#v(i)
SADD26#v(i)
ADDWF REMB1
MOVFP BARGB0,WRREG
ADDWF REMB0
SOK26#v(i)
RLCF ACCB2
i=i+1
endw
RLCF ACCB3,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WRREG
BTFSS ACCB2,LSB
GOTO SADD2624
SUBWF REMB1
MOVFP BARGB0,WRREG
SUBWFB REMB0
GOTO SOK2624
SADD2624
ADDWF REMB1
MOVFP BARGB0,WRREG
ADDWF REMB0
SOK2624
RLCF ACCB3
i = 25
while i < 32
RLCF ACCB3,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WRREG
BTFSS ACCB3,LSB
GOTO SADD26#v(i)
SUBWF REMB1
MOVFP BARGB0,WRREG
SUBWFB REMB0
GOTO SOK26\#v(i)
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
SOK26\#v(i)
RLCF ACCB3
i = i + 1
endw
BTFSC ACCB3, LSB
GOTO SOK26
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
SOK26
•
UDIV3216
macro
; restore = 15/20 clks, nonrestore = 11/14 clks
; Max Timing: 16*15+1+16*20 = 561 clks
; Min Timing: 16*11+1+16*14 = 401 clks
; PM: 16*15+1+16*20 = 561
; DM: 9

variable i
i = 0
while i < 8
RLCF ACCB0, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
BTFSC _C
GOTO UOK26\#v(i)
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
BCF _C
UOK26\#v(i)
RLCF ACCB0
i = i + 1
endw
i = 8
while i < 16
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
BTFSC _C
GOTO UOK26\#v(i)
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
BCF _C
UOK26\#v(i)
RLCF ACCB1
i = i + 1
endw
CLRF TEMP
i = 16
while i < 24
RLCF ACCB2, W
RLCF REMB1
RLCF REMB0
AN617

```
RLCF        TEMP
MOVFP      BARGB1,WREG
SUBWF      REMB1
MOVFP      BARGB0,WREG
SUBWFB     REMB0
CLRFP      WREG
SUBWFB     TEMP
BTFSCP     _C
GOTO       UOK26#v(i)
MOVFP      BARGB1,WREG
ADDWF      REMB1
MOVFP      BARGB0,WREG
ADDWFC     REMB0
CLRFP      WREG
ADDWFC     TEMP
BCF         _C

UOK26#v(i)

RLCF        ACCB2
i=i+1
endw
i = 24
while i < 32

RLCF        ACCB3,W
RLCF        REMB1
RLCF        REMB0
RLCF        TEMP
MOVFP      BARGB1,WREG
SUBWF      REMB1
MOVFP      BARGB0,WREG
SUBWFB     REMB0
CLRFP      WREG
SUBWFB     TEMP
BTFSCP     _C
GOTO       UOK26#v(i)
MOVFP      BARGB1,WREG
ADDWF      REMB1
MOVFP      BARGB0,WREG
ADDWFC     REMB0
CLRFP      WREG
ADDWFC     TEMP
BCF         _C

UOK26#v(i)

RLCF        ACCB3
i=i+1
endw
endm
```

```
NDIV3216

; Max Timing: 10*31*15+6 = 481 clks
; Min Timing: 10*31*14+6 = 450 clks
; PM: 10*31*19+6 = 605

variable i

RLCF        ACCB0,W
RLCF        REMB1
RLCF        REMB0
MOVFP      BARGB1,WREG
SUBWF      REMB1
MOVFP      BARGB0,WREG
SUBWFB     REMB0
CLRFP      TEMP,W
SUBWFB     TEMP
RLCF        ACCB0
i = 1
while i < 8

RLCF        ACCB0,W
RLCF        REMB1
RLCF        REMB0
RLCF        TEMP
MOVFP      BARGB1,WREG
```

DS00617A-page 2-456 © 1995 Microchip Technology Inc.
BTFSS       ACCBO,LSB
GOTO        NADD26#v(i)
SUBWF       REMB1
MOVFPP      BARGBO,WREG
SUBWFPP     REMBO
CLRF        WREG
SUBWFPP     TEMP
GOTO        NOK26#v(i)
NADD26#v(i)  ADDWF       REMB1
MOVFPP      BARGBO,WREG
ADDWFC      REMBO
CLRF        WREG
ADDWFC      TEMP
NOK26#v(i)  RLCF         ACCBO
i=i+1
endw
RLCF        ACCB1,W
RLCF        REMB1
RLCF        REMBO
RLCF        TEMP
MOVFPP      BARGB1,WREG
BTFSS       ACCB0,LSB
GOTO        NADD268
SUBWF       REMB1
MOVFPP      BARGBO,WREG
SUBWFPP     REMBO
CLRF        WREG
SUBWFPP     TEMP
GOTO        NOK268
NADD268     ADDWF       REMB1
MOVFPP      BARGBO,WREG
ADDWFC      REMBO
CLRF        WREG
ADDWFC      TEMP
NOK268      RLCF         ACCB1
i=9
while i < 16
RLCF        ACCB1,W
RLCF        REMB1
RLCF        REMBO
RLCF        TEMP
MOVFPP      BARGB1,WREG
BTFSS       ACCB1,LSB
GOTO        NADD26#v(i)
SUBWF       REMB1
MOVFPP      BARGBO,WREG
SUBWFPP     REMBO
CLRF        WREG
SUBWFPP     TEMP
GOTO        NOK26#v(i)
NADD26#v(i)  ADDWF       REMB1
MOVFPP      BARGBO,WREG
ADDWFC      REMBO
CLRF        WREG
ADDWFC      TEMP
NOK26#v(i)  RLCF         ACCB1
i=i+1
endw
RLCF        ACCB2,W
RLCF        REMB1
RLCF        REMBO
RLCF        TEMP
MOVFPP      BARGB1,WREG
BTFSS ACCB1, LSB
GOTO NADD2616
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK2616
NADD2616 ADDWF REMB1
MOVFP BARGBO, WREG
ADDWFC REMBO
CLRF WREG
ADDWFC TEMP
NOK2616 RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2, W
RLCF REMB1
RLCF REMBO
RLCF TEMP
MOVFP BARGB1, WREG
BTFSS ACCB2, LSB
GOTO NADD26#v(i)
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK26#v(i)
NADD26#v(i) ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
CLRF WREG
ADDWFC TEMP
NOK26#v(i) RLCF ACCB2
i = i + 1
endw
RLCF ACCB3, W
RLCF REMB1
RLCF REMBO
RLCF TEMP
MOVFP BARGB1, WREG
BTFSS ACCB2, LSB
GOTO NADD2624
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMBO
CLRF WREG
SUBWFB TEMP
GOTO NOK2624
NADD2624 ADDWF REMB1
MOVFP BARGB0, WREG
ADDWFC REMBO
CLRF WREG
ADDWFC TEMP
NOK2624 RLCF ACCB3
i = 25
while i < 32
RLCF ACCB3, W
RLCF REMB1
RLCF REMBO
RLCF TEMP
MOVFP BARGB1, WREG
BTFSS        ACCB3,LSB
GOTO         NADD26#v(i)
SUBWF        REMB1
MOVFPP       BARGBO,WREG
SUBWFBO      REMB0
CLRF         WREG
SUBWFBO      TEMP
GOTO         NOK26#v(i)
NADD26#v(i)  ADDWF        REMB1
MOVFPP       BARGBO,WREG
ADDWF        REMB0
CLRF         WREG
ADDWF        TEMP
NOK26#v(i)   RLCF         ACCB3
i=i+1
endw
BTFSC        ACCB3,LSB
GOTO         NOK26
MOVFPP       BARGB1,WREG
ADDWF        REMB1
MOVFPP       BARGBO,WREG
ADDWF        REMB0
NOK26        endm
UDIV3215     macro
; Max Timing: 8+31*12+6 = 386 clks
; Min Timing: 8+31*11+6 = 355 clks
; PM: 8+31*14+6 = 448 DM: 8
variable i
RLCF         ACCB0,W
RLCF         REMB1
RLCF         REMB0
MOVFPP       BARGB1,WREG
SUBWF        REMB1
MOVFPP       BARGBO,WREG
SUBWFBO      REMB0
RLCF         ACCB0
i = 1
while i < 8
RLCF         ACCB0,W
RLCF         REMB1
RLCF         REMB0
MOVFPP       BARGB1,WREG
BTFSS        ACCB0,LSB
GOTO         UADD25#v(i)
SUBWF        REMB1
MOVFPP       BARGBO,WREG
SUBWFBO      REMB0
GOTO         UOK25#v(i)
UADD25#v(i)  ADDWF        REMB1
MOVFPP       BARGBO,WREG
ADDWF        REMB0
UOK25#v(i)   RLCF         ACCB0
i=i+1
endw
RLCF         ACCB1,W
RLCF         REMB1
RLCF         REMB0
MOVFPP       BARGB1,WREG
BTFSS        ACCB0,LSB
GOTO         UADD258
SUBWF        REMB1
MOVFPP       BARGBO,WREG
SUBWFBO      REMB0
GOTO         UOK258
UADD258
  ADDWF REMB1
  MOVFP BARGB0, WREG
  ADDWFC REMBO

UOK258
  RLCF ACCB1

  i = 9
  while i < 16
    RLCF ACCB1, W
    RLCF REMB1
    RLCF REMBO
    MOVFP BARGB1, WREG
    BTFS ACCB1, LSB
    GOTO UADD25#v(i)
    SUBWF REMB1
    MOVFP BARGB0, WREG
    SUBWFB REMBO
    GOTO UOK25#v(i)
  UADD25#v(i)
    ADDWF REMB1
    MOVFP BARGB0, WREG
    ADDWFC REMBO
    UOK25#v(i)
      RLCF ACCB1
      i = i + 1
  endw
  RLCF ACCB2, W
  RLCF REMB1
  RLCF REMBO
  MOVFP BARGB1, WREG
  BTFS ACCB1, LSB
  GOTO UADD2516
  SUBWF REMB1
  MOVFP BARGB0, WREG
  SUBWFB REMBO
  GOTO UOK2516
  UADD2516
    ADDWF REMB1
    MOVFP BARGB0, WREG
    ADDWFC REMBO
    UOK2516
      RLCF ACCB2

    i = 17
    while i < 24
      RLCF ACCB2, W
      RLCF REMB1
      RLCF REMBO
      MOVFP BARGB1, WREG
      BTFS ACCB2, LSB
      GOTO UADD25#v(i)
      SUBWF REMB1
      MOVFP BARGB0, WREG
      SUBWFB REMBO
      GOTO UOK25#v(i)
    UADD25#v(i)
      ADDWF REMB1
      MOVFP BARGB0, WREG
      ADDWFC REMBO
    UOK25#v(i)
      RLCF ACCB2
      i = i + 1
  endw
  RLCF ACCB3, W
  RLCF REMB1
  RLCF REMBO
  MOVFP BARGB1, WREG
  BTFS ACCB2, LSB
  GOTO UADD2524
  SUBWF REMB1
  MOVFP BARGB0, WREG
  SUBWFB REMBO
  GOTO UOK2524
  UADD2524
    ADDWF REMB1
    MOVFP BARGB0, WREG
ADDWFC REMB0
RLCF ACCB3

UOK2524

i = 25
while i < 32
RLCF ACCB3,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
BTFSS ACCB3,LSB
GOTO UADD25#v(i)
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK25#v(i)

UADD25#v(i)
ADDWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0

UOK25#v(i)
RLCF ACCB3
i=i+1
endw
BTFSC ACCB3,LSB
GOTO UOK25
MOVFP BARGB1,WREG
ADDWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0

UOK25
endm

UDIV3115

macro

Max Timing: 5+8+30*12+6 = 379 clks
Min Timing: 5+8+30*11+6 = 349 clks
PM: 5+8+30*14+6 = 439
DM: 8

variable i
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
RLCF ACCB0
RLCF ACCB0,W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1,WREG
BTFSS ACCB0,LSB
GOTO UADD15#v(i)
SUBWF REMB1
MOVFP BARGB0,WREG
SUBWFB REMB0
GOTO UOK15#v(i)

UADD15#v(i)
ADDWF REMB1
MOVFP BARGB0,WREG
ADDWFC REMB0

UOK15#v(i)
RLCF ACCB0
}

i=i+1
endw
RLCF ACCB1,W
RLCF REMB1
AN617

RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB0, LSB
GOTO UADD158
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK158
UADD158
ADDFW REMB1
MOVFP BARGB0, WREG
ADDFWFC REMB0
UOK158
RLCF ACCB1

while \( i < 16 \)
RLCF ACCB1, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB1, LSB
GOTO UADD15#v(i)
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK15#v(i)
UADD15#v(i)
ADDFW REMB1
MOVFP BARGB0, WREG
ADDFWFC REMB0
UOK15#v(i)
RLCF ACCB1
i = i + 1
endw
RLCF ACCB2, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB2, LSB
GOTO UADD1516
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK1516
UADD1516
ADDFW REMB1
MOVFP BARGB0, WREG
ADDFWFC REMB0
UOK1516
RLCF ACCB2

i = 17
while \( i < 24 \)
RLCF ACCB2, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFSS ACCB2, LSB
GOTO UADD15#v(i)
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK15#v(i)
UADD15#v(i)
ADDFW REMB1
MOVFP BARGB0, WREG
ADDFWFC REMB0
UOK15#v(i)
RLCF ACCB2
i = i + 1
endw
RLCF ACCB3, W
RLCF REMB1
RLCF REMB0
MOVFP BARGB1, WREG
BTFS ACCB2,LSB
GOTO UADD1524
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMBO
GOTO UOK1524
UADD1524 ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
UOK1524 RLCF ACCB3
i = 25
while i < 32
RLCF ACCBO,W
RLCF REMB1
RLCF REMB0
MOVFP BARGBI,WREG
BTFS ACCB3,LSB
GOTO UADD15#v(i)
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWFB REMBO
GOTO UOK15#v(i)
UADD15#v(i) ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
UOK15#v(i) RLCF ACCB3
i = i + 1
endw
BTFS ACCB3,LSB
GOTO UOK15
MOVFP BARGBI,WREG
ADDWF REMB1
MOVFP BARGBO,WREG
ADDWFC REMBO
UOK15 endm

;**********************************************************************************************
;**********************************************************************************************
24/24 Bit Division Macros

SDIV2424 macro

; Max Timing: 7+11+22*15+8 = 356 clks
; Min Timing: 7+11+22*14+3 = 329 clks
; PM: 7+11+22*19+8 = 444 DM: 9

variable i

MOVFP BARGBO,WREG
SUBWF REMB2
MOVFP BARGBI,WREG
SUBWFB REMB1
MOVFP BARGBO,WREG
SUBWFB REMB0
RLCF ACCB0
RLCF ACCB0,W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGBO,WREG
ADDWF REMB2
MOVFP BARGBI,WREG
ADDWFC REMB1
MOVFP BARGBO,WREG
ADDWFC REMB0
RLCF ACCB0
i = 2
while i < 8
RLCF ACCB0,W
RLCF REMB2
RI.CF REMBl
MOVFP BARGB2, WREG
BTFSS ACCB0, LSB
GOTO SADD44#v(i)
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK44#v(i)
SADD44#v(i)
ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK44#v(i)
RlCF ACCB0
i = i + 1
dendw
RI.CF ACCB1, W
RI.CF REMB2
RI.CF REMB1
RlCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB0, LSB
GOTO SADD448
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK448
SADD448
ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK448
RlCF ACCB1
i = 9
while i < 16
RI.CF ACCB1, W
RI.CF REMB2
RI.CF REMB1
RlCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB1, LSB
GOTO SADD44#v(i)
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK44#v(i)
SADD44#v(i)
ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK44#v(i)
RlCF ACCB1
i = i + 1
dendw
RI.CF ACCB2, W
RI.CF REMB2
RI.CF REMB1
RlCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB1, LSB
GOTO SADD4416
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK4416
SADD4416
ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK4416
RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2, W
RLCF REMB2
RLCF REMBl
RLCF REMBO
MOVFP BARGBO, WREG
BTFSS ACCB2, LSB
GOTO SADD44#v(i)
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
GOTO SOK44#v(i)
SADD44#v(i)
ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK44#v(i)
RLCF ACCB2
i = i + 1
endw
BTFSC ACCB2, LSB
GOTO SOK44
MOVFP BARGBO, WREG
ADDWF REMB2
MOVFP BARGBl, WREG
ADDWFC REMBl
MOVFP BARGBO, WREG
ADDWFC REMBO
SOK44
endm
UDIV2424
macro
; restore = 20/25 clks, nonrestore = 14/17 clks
; Max Timing: 16*20+1+8*25 = 521 clks
; Min Timing: 16*14+1+8*17 = 361 clks
; PM: 16*20+1+8*25 = 521 DM: 10
variable i
i = 0
while i < 8
RLCF ACCBO, W
RLCF REMB2
RLCF REMBl
RLCF REMBO
MOVFP BARGBO, WREG
SUBWF REMB2
MOVFP BARGBl, WREG
SUBWFB REMBl
MOVFP BARGBO, WREG
SUBWFB REMBO
BTFSC _C
GOTO UOK44#v(i)

MOVFP BARGB2, WREG

ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
BCF _c

UOK44#v(i)

i = i + 1
endw

i = 8
while i < 16

RLCF ACCB0

UOK44#v(i)

RLCF ACCB1, W

RCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
BTFSC _c
GOTO UOK44#v(i)

MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
BCF _c

UOK44#v(i)

RLCF ACCB1

i = i + 1
endw

CLRF TEMP

i = 16
while i < 24

RLCF ACCB2, W

RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
CLRF WREG
SUBWFB TEMP
BTFSC _c
GOTO UOK44#v(i)

MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
CLRF WREG
ADDWFC TEMP
BCF _c

UOK44#v(i)

RLCF ACCB2

i = i + 1
endw
endm
NDIV2424 macro

; Max Timing: 13+23*18+8 = 435 clks
; Min Timing: 13+23*17+3 = 407 clks
; PM: 13+23*24+8 = 573 DM: 10

variable i

  RLCF ACCBO,W
  RLCF REMB2
  RLCF REMB1
  RLCF REMBO
  MOVFP BARGB2,WREG
  SUBWF REMB2
  MOVFP BARGB1,WREG
  SUBWF WREG
  CLRF TEMP,W
  SUBWF TEMP
  RLCF ACCBO

  i = 1
  while i < 8
    RLCF ACCBO,W
    RLCF REMB2
    RLCF REMB1
    RLCF REMBO
    RLCF TEMP
    MOVFP BARGB2,WREG
    BTFS ACCBO,LSB
    GOTO NADD44#(i)
    SUBWF REMB2
    MOVFP BARGB1,WREG
    ADDWF REMB1
    MOVFP BARGB0,WREG
    ADDWFRemBO
    CLRF WREG
    SUBWF TEMP
    GOTO NOK44#(i)

NADD44#(i)

  ADDWF REMB2
  MOVFP BARGB1,WREG
  ADDWF REMB1
  MOVFP BARGB0,WREG
  ADDWF RemBO
  CLRF WREG
  ADDWF TEMP

NOK44#(i)

  RLCF ACCBO
  i = i+1
  endw

  RLCF ACCBO1,W
  RLCF REMB2
  RLCF REMB1
  RLCF REMBO
  RLCF TEMP
  MOVFP BARGB2,WREG
  BTFS ACCBO,LSB
  GOTO NADD448
  SUBWF REMB2
  MOVFP BARGB1,WREG
  ADDWF REMB1
  MOVFP BARGB0,WREG
  ADDWF RemBO
  CLRF WREG
  SUBWF TEMP
  GOTO NOK448

NADD448

  ADDWF REMB2
  MOVFP BARGB1,WREG
  ADDWF REMB1
AN617

MOVFP BARGB0, WREG
ADDWFC REMB0
CLRF WREG
ADDWFC TEMP

NOK448

RLCF ACCB1
i = 9
while i < 16
RLCF ACCB1, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB2, WREG
BTFSS ACCB1, LSB
GOTO NADD44v(i)
SUBWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
CLR WREG
SUBWF TEMP
GOTO NOK44v(i)

NADD44v(i)

ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
CLR WREG
ADDWFC TEMP

NOK44v(i)

RLCF ACCB1
i = i + 1
endw
RLCF ACCB2, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
RLCF TEMP
MOVFP BARGB2, WREG
BTFSS ACCB1, LSB
GOTO NADD4416
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWF REMB1
MOVFP BARGB0, WREG
SUBWF REMB0
CLR WREG
SUBWF TEMP
GOTO NOK4416

NADD4416

ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLR WREG
ADDWFC TEMP

NOK4416

RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2, W
RLCF REMB2
RLCF REMB1
RLCF REMB0

© 1995 Microchip Technology Inc.
```assembly
AN617

RLCF TEMP
MOVFP BARGB2, WREG
BTFSS ACCB2, LSB
GOTO NADD44#v(i)
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
CLRFP WREG
SUBWFB TEMP
GOTO NOK44#v(i)

NADD44#v(i)
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
CLRFP WREG
ADDWFC TEMP
GOTO NOK44#v(i)

NOK44#v(i) RLCFB ACCB2
i = i + 1
endw
BTFSCP ACCB2, LSB
GOTO NOK44
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
endm

UDIV2423
macro
; Max Timing: 11+23*15+8 = 364 clks
; Min Timing: 11+23*14+3 = 336 clks
; FM: 11+23*19+8 = 456 DM: 9

variable i
RLCF ACCB0, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2, WREG
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
RLCF ACCB0
i = 1
while i < 8
RLCF ACCB0, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB0, LSB
GOTO UADD43#v(i)
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK43#v(i)

UADD43#v(i) ADDWF REMB2
MOVFP BARGB1, WREG
```

© 1995 Microchip Technology Inc. DS00617A-page 2-469
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDWFC REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>ADDWFC REMB0</td>
<td></td>
</tr>
<tr>
<td>UOK43#v(i) RLCF ACCB0</td>
<td></td>
</tr>
<tr>
<td>i=i+1</td>
<td></td>
</tr>
<tr>
<td>endw</td>
<td></td>
</tr>
<tr>
<td>RLCF ACCB1,W</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB2</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB1</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB0</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB2,WREG</td>
<td></td>
</tr>
<tr>
<td>BTFS S ACCB0,LSB</td>
<td></td>
</tr>
<tr>
<td>GOTO UADD438</td>
<td></td>
</tr>
<tr>
<td>SUBWF REMB2</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB1,WREG</td>
<td></td>
</tr>
<tr>
<td>SUBWFB REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>SUBWFB REMB0</td>
<td></td>
</tr>
<tr>
<td>GOTO UOK438</td>
<td></td>
</tr>
<tr>
<td>UADD438 ADDWF REMB2</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB1,WREG</td>
<td></td>
</tr>
<tr>
<td>ADDWFC REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>UOK438 RLCF ACCB1</td>
<td></td>
</tr>
<tr>
<td>i = 9</td>
<td></td>
</tr>
<tr>
<td>while i &lt; 16</td>
<td></td>
</tr>
<tr>
<td>RLCF ACCB1,W</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB2</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB1</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB0</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB2,WREG</td>
<td></td>
</tr>
<tr>
<td>BTFS S ACCB1,LSB</td>
<td></td>
</tr>
<tr>
<td>GOTO UADD43#v(i)</td>
<td></td>
</tr>
<tr>
<td>SUBWF REMB2</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB1,WREG</td>
<td></td>
</tr>
<tr>
<td>SUBWFB REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>SUBWFB REMB0</td>
<td></td>
</tr>
<tr>
<td>GOTO UOK43#v(i)</td>
<td></td>
</tr>
<tr>
<td>UADD43#v(i) ADDWF REMB2</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB1,WREG</td>
<td></td>
</tr>
<tr>
<td>ADDWFC REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>UOK43#v(i) RLCF ACCB1</td>
<td></td>
</tr>
<tr>
<td>i=i+1</td>
<td></td>
</tr>
<tr>
<td>endw</td>
<td></td>
</tr>
<tr>
<td>RLCF ACCB2,W</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB2</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB1</td>
<td></td>
</tr>
<tr>
<td>RLCF REMB0</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB2,WREG</td>
<td></td>
</tr>
<tr>
<td>BTFS S ACCB1,LSB</td>
<td></td>
</tr>
<tr>
<td>GOTO UADD4316</td>
<td></td>
</tr>
<tr>
<td>SUBWF REMB2</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB1,WREG</td>
<td></td>
</tr>
<tr>
<td>SUBWFB REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>SUBWFB REMB0</td>
<td></td>
</tr>
<tr>
<td>GOTO UOK4316</td>
<td></td>
</tr>
<tr>
<td>UADD4316 ADDWF REMB2</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB1,WREG</td>
<td></td>
</tr>
<tr>
<td>ADDWFC REMB1</td>
<td></td>
</tr>
<tr>
<td>MOVFP BARGB0,WREG</td>
<td></td>
</tr>
<tr>
<td>ADDWFC REMB0</td>
<td></td>
</tr>
</tbody>
</table>

AN617

© 1995 Microchip Technology Inc.
UOK4316

```assembly
i = 17
while i < 24
    RLCF ACCB2
    RLCF REMB2
    RLCF REMB1
    RLCF REMB0
    MOVFP BARGB2, WREG
    BTFSS ACCB2, LSB
    GOTO UADD43#v(i)
    SUBWF REMB2
    MOVFP BARGB1, WREG
    SUBWFB REMB1
    MOVFP BARGB0, WREG
    SUBWFB REMB0
    GOTO UOK43#v(i)
UADD43#v(i)
    ADDWF REMB2
    MOVFP BARGB2, WREG
    ADDWFC REMB2
    MOVFP BARGB1, WREG
    ADDWFC REMB1
    MOVFP BARGB0, WREG
    ADDWFC REMB0
    UOK43#v(i)
    RLCF ACCB2
    i = i + 1
endw
BTFSC ACCB2, LSB
GOTO UOK43
MOVFP BARGB2, WREG
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWFC REMB2
MOVFP BARGB0, WREG
ADDWFC REMB1
MOVFP BARGB0, WREG
ADDWFC REMB0
UOK43
endm
UDIV2323
macro
    ; Max Timing: 7+11+22*15+8 = 356 clks
    ; Min Timing: 7+11+22*14+3 = 329 clks
    ; PM: 7+11+22*19+8 = 444
    ; DM: 9
    variable i
    MOVFP BARGB2, WREG
    SUBWF REMB2
    MOVFP BARGB1, WREG
    SUBWFB REMB1
    MOVFP BARGB0, WREG
    SUBWFB REMB0
    RLCF ACCB0
    RLCF ACCB0, W
    RLCF REMB2
    RLCF REMB1
    RLCF REMB0
    MOVFP BARGB2, WREG
    ADDWF REMB2
    MOVFP BARGB1, WREG
    ADDWFC REMB2
    MOVFP BARGB0, WREG
    ADDWFC REMB1
    MOVFP BARGB0, WREG
    ADDWFC REMB0
    RLCF ACCB0
    i = 2
while i < 8
    RLCF ACCB0, W
    RLCF REMB2
    RLCF REMB1
    RLCF REMB0
    MOVFP BARGB2, WREG
    BTFSS ACCB0, LSB
    GOTO UADD33#v(i)
```

© 1995 Microchip Technology Inc.
```
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK33#v(i)

UADD33#v(i)
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0
GOTO UOK33#v(i)
i = i + 1
endw

UOK33#v(i)
RLCF ACCB0

RLCF ACCB1, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB0, LSB
GOTO UADD338
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK338

UADD338
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0

UOK338
RLCF ACCB1
i = 9
while i < 16

RLCF ACCB1, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB0, LSB
GOTO UADD33#v(i)
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
MOVFP BARGB0, WREG
SUBWFB REMB0
GOTO UOK33#v(i)

UADD33#v(i)
ADDWF REMB2
MOVFP BARGB1, WREG
ADDWF REMB1
MOVFP BARGB0, WREG
ADDWF REMB0

UOK33#v(i)
RLCF ACCB1
i = i + 1
endw

RLCF ACCB2, W
RLCF REMB2
RLCF REMB1
RLCF REMB0
MOVFP BARGB2, WREG
BTFSS ACCB1, LSB
GOTO UADD3316
SUBWF REMB2
MOVFP BARGB1, WREG
SUBWFB REMB1
```
MOVFP BARGBO,WREG
SUBWF REMB0
GOTO UOK3316
UADD3316
ADDFP BARGB1,WREG
ADDFFC REMB1
MOVFP BARGBO,WREG
ADDFFC REMB0
UOK3316
RLCF ACCB2
i = 17
while i < 24
RLCF ACCB2,W
RLCF REMB2
RLCF REMB1
MOVFP BARGB2,WREG
BTFSS ACCB2,LSB
GOTO UADD33#v(i)
SUBWF REMB2
MOVFP BARGB1,WREG
SUBWF REMB1
MOVFP BARGBO,WREG
SUBWF REMB0
GOTO UOK33#v(i)
UADD33#v(i)
ADDFP REMB2
MOVFP BARGB1,WREG
ADDFFC REMB1
MOVFP BARGBO,WREG
ADDFFC REMB0
UOK33#v(i)
RLCF ACCB2
i=i+1
endw
BTFSC ACCB2,LSB
GOTO UOK33
MOVFP BARGB2,WREG
ADDFP REMB2
MOVFP BARGB1,WREG
ADDFP REMB1
MOVFP BARGBO,WREG
ADDFFC REMB0
UOK33
endm

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

; FXD3216S
; 32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16
; Input: 32 bit signed fixed point dividend in AARGBO, AARGB1,AARGB2,AARGB3
; 16 bit unsigned fixed point divisor in BARGBO, BARGBl
; Use: CALL FXD3216S
; Output: 32 bit signed fixed point quotient in AARGBO, AARGB1,AARGB2,AARGB3
; 16 bit fixed point remainder in REMBO, REMBl
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 11+379+3 = 393 clks A > 0, B > 0
; 14+379+17 = 410 clks A > 0, B < 0
; 18+379+17 = 414 clks A < 0, B > 0
; 21+379+3 = 403 clks A < 0, B < 0
; Min Timing: 11+349+3 = 363 clks A > 0, B > 0
; 14+349+17 = 380 clks A > 0, B < 0
; 18+349+17 = 384 clks A < 0, B > 0
; 21+349+3 = 373 clks A < 0, B < 0
; PM: 21+439+16 = 476 DM: 9
FXD3216S
MOVFP AARGBO,WREG
XORWF BARGBO,W
MOVWF SIGN
CLRF REMB0
CLRF REMBl,W

© 1995 Microchip Technology Inc.
BTFSS BARGBO, MSB ; if MSB set go & negate BARG
GOTO CA3216S
COMF BARGB1
COMF BARGBO
INCF BARGB1
ADDFPC BARGBO
CA3216S BTFSS AARGBO, MSB ; if MSB set go & negate ACCa
GOTO C3216S
COMF AARGB3
COMF AARGB2
COMF AARGB1
INCF AARGB0
ADDFPC AARGB2
ADDFPC AARGB1
ADDFPC AARGB0
C3216S SDIV3216
BTFSS SIGN, MSB ; negate (ACCc, ACCd)
RETLW 0x00
COMF AARGB3
COMF AARGB2
COMF AARGB1
CLRF WREG
INCF AARGB3
ADDFPC AARGB2
ADDFPC AARGB1
ADDFPC AARGB0
COMF REMB1
COMF REMB0
INCF REMB1
ADDFPC REMB0
RETLW 0x00

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

32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16
Input: 32 bit unsigned fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
16 bit unsigned fixed point divisor in BARGBO, BARGB1
Use: CALL FXD3216U
Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1, AARGB2, AARGB3
16 bit unsigned fixed point remainder in REMB0, REMB1
Result: AARG, REM <-- AARG / BARG
Max Timing: 2+481+2 = 485 clks
Min Timing: 2+450+2 = 459 clks
PM: 2+605+1 = 608 DM: 9
FXD3216U CLRF REMB0
CLRF REMB1
NDIV3216 RETLW 0x00

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

32/15 Bit Unsigned Fixed Point Divide 32/15 -> 32.15
Input: 32 bit unsigned fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
15 bit unsigned fixed point divisor in BARGBO, BARGB1
Use: CALL FXD3215U
Output: 32 bit unsigned fixed point quotient in AARGBO, AARGB1
15 bit unsigned fixed point remainder in REMB0, REMB1
Result: AARG, REM <-- AARG / BARG
Max Timing: 2+386+2 = 390 clks
Min Timing: 2+355+2 = 359 clks
PM: 2+448+1 = 451 DM: 8
FXD3215U CLRF REMB0
CLRF REMB1
UDIV3215 RETLW 0x00

© 1995 Microchip Technology Inc.
31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15

Input: 31 bit unsigned fixed point dividend in AARGBO, AARGB1, AARGB2, AARGB3
15 bit unsigned fixed point divisor in BARGB0, BARGB1

Use: CALL FXD3115U

Output: 31 bit unsigned fixed point quotient in AARGBO, AARGBl
15 bit unsigned fixed point remainder in REMBO, REMBl

Result: AARG, REM <-- AARG / BARG

Max Timing: $2+379+2 = 383$ clks
Min Timing: $2+349+2 = 353$ clks

PM: $2+439+1 = 442$  DM: 8

```assembly
FXD3115U
CLRF REMBO
CLRF REMBl
UDIV3115
RETLW 0x00
```

24/24 Bit Signed Fixed Point Divide 24/24 -> 24.24

Input: 24 bit signed fixed point dividend in AARGBO, AARGB1, AARGB2
24 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2

Use: CALL FXD2424S

Output: 24 bit signed fixed point quotient in AARGBO, AARGBl, AARGB2
24 bit fixed point remainder in REMBO, REMBl, REMB2

Result: AARG, REM <-- AARG / BARG

Max Timing: $12+356+3 = 371$ clks  $A > 0, B > 0$
$17+356+17 = 390$ clks  $A > 0, B < 0$
$17+356+17 = 390$ clks  $A < 0, B > 0$
$22+356+3 = 381$ clks  $A < 0, B < 0$

Min Timing: $12+329+3 = 344$ clks  $A > 0, B > 0$
$17+329+17 = 363$ clks  $A > 0, B < 0$
$17+329+17 = 363$ clks  $A < 0, B > 0$
$22+329+3 = 354$ clks  $A < 0, B < 0$

PM: $22+444+16 = 482$  DM: 10

```assembly
FXD2424S
MOVFP AARGBO,WREG2
XORWF BARGB0,W
MOVWF SIGN
CLRF REMBO
CLRF REMBl
CLRF REMB2,W
BTFSS BARGB0,MSB ; if MSB set, negate BARG
GOTO CA2424S
COMP BARGB2
COMP BARGB1
COMP BARGB0
INCF BARGB2
ADDFWC BARGB1
ADDFWC BARGB0
CA2424S
BTFSS AARGBO,MSB ; if MSB set, negate AARG
GOTO C2424S
COMP AARGB2
COMP AARGB1
COMP AARGBO
INCF AARGB2
ADDFWC AARGB1
ADDFWC AARGBO
C2424S
SDIV2424
BTFSS SIGN,MSB
RETLW 0x00
COMP AARGB2
COMP AARGB1
COMP AARGBO
CLRF WREG
INCF AARGB2
```
ADDWFC AARGBl
ADDWFC AARGB0
ADDWFC AARGB2
ADDWFC AARGBl
ADDWFC AARGB0
ADDWFC AARGB2
RETLW 0x00

24/24 Bit Unsigned Fixed Point Divide 24/24 -> 24.24

24 bit unsigned fixed point dividend in AARGB0, AARGBl, AARGB2
24 bit unsigned fixed point divisor in BARGB0, BARGBl, BARGB2
Use: CALL FXD2424U
Output: 24 bit unsigned fixed point quotient in AARGB0, AARGBl, AARGB2
24 bit unsigned fixed point remainder in REMB0, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing: 3+435+2 = 440 clks
Min Timing: 3+407+2 = 412 clks
PM: 3+573+1 = 577 DM: 10
FXD2424U CLRF REMB0
CLRF REMBl
CLRF REMB2
NDIV2424
RETLW 0x00

24/23 Bit Unsigned Fixed Point Divide 24/23 -> 24.23

24 bit unsigned fixed point dividend in AARGB0, AARGBl, AARGB2
23 bit unsigned fixed point divisor in BARGB0, BARGBl, BARGB2
Use: CALL FXD2423U
Output: 24 bit unsigned fixed point quotient in AARGB0, AARGBl, AARGB2
23 bit unsigned fixed point remainder in REMB0, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing: 3+364+2 = 369 clks
Min Timing: 3+336+2 = 341 clks
PM: 3+456+1 = 460 DM: 9
FXD2423U CLRF REMB0
CLRF REMBl
CLRF REMB2
UDIV2423
RETLW 0x00

23/23 Bit Unsigned Fixed Point Divide 23/23 -> 23.23

23 bit unsigned fixed point dividend in AARGB0, AARGBl, AARGB2
23 bit unsigned fixed point divisor in BARGB0, BARGBl, BARGB2
Use: CALL FXD2323U
Output: 23 bit unsigned fixed point quotient in AARGB0, AARGBl, AARGB2
23 bit unsigned fixed point remainder in REMB0, REMBl, REMB2
Result: AARG, REM <-- AARG / BARG
Max Timing: 3+356+2 = 361 clks
Min Timing: 3+329+2 = 334 clks
PM: 3+444+1 = 448 DM: 9
FXD2323U CLRF REMB0
CLRF REMBl
CLRF REMB2
UDIV2323
RETLW 0x00

END
INTRODUCTION

This application note describes an electronic system to improve the efficiency of certain types of single-phase induction motors.

The system is based around the MTE1122 - an energy management controller IC for induction motors. This CMOS device is based on Microchip’s RISC processor core and proprietary firmware algorithms. When combined with some external analog components, the MTE1122 will provide an electronic system that economically reduces the operating costs of small induction motors by as much as 58%. It will also allow motors to run cooler and with less vibration. The system operates on single phase 110 or 240 VAC.

FIGURE 1: SYSTEM BLOCK DIAGRAM

RECOMMENDED APPLICATION

The schematic diagram shown in Figure 11 is a recommended circuit. See Table 3 for the parts list. It uses low cost, readily available components. Note that Vcc is supplied directly from the AC line without the need for a transformer. Component values have been calculated to work on 110 or 240 VAC lines, with motor current draws of up to 15A RMS continuous. This translates to 1 to 1.5 HP at 110V, and 3 HP at 220V. Motor size can be increased by selecting a triac with a larger current rating.

This system will only work with rotating inductive loads (i.e., motors) that are not otherwise power-factor corrected -- capacitor-run motors will not function with this system, nor will fluorescent lighting. Universal motors (brush-type) will not benefit from the system, either. While use of this system will usually save energy, the greatest savings will be for lightly-loaded motors.

For best results, appliances or systems with other electrical devices in addition to motors should have those devices powered directly from the line, not through the energy management control system (EMC).

Also, note that each motor must have its own control circuit (unless the motors are never activated at the same time).

The circuit can be laid out on a single- or double-sided board, observing the standard layout techniques used with monolithic microcontrollers. LED D3 is lighted during normal operation of the MTE1122, and can be left out of the circuit, if desired.

Heatsinking of the triac will be required, the size depending on the triac rating, the motor current draws and ambient temperatures.

The distance between the motor and the MTE1122 circuit is not critical.

Standard electrical practices should be followed. Agency approvals may be required, depending on the implementation. While this circuit should not be connected to earth ground, any enclosure should be so connected. Also note that the low voltage power generated on this board should not be used to supply any other circuitry, particularly if that circuitry is off the board.
THEORY OF OPERATION

Power Consumption
In an induction motor, the current draw at no load is quite high because the stator windings must supply all the magnetic field energy. This means that, even when idling, the motor draws a major portion of its full-load current. The energy not converted into work is converted into heat and vibration. In addition to being wasted, the heat and vibration shorten the life of lubricants, bearings, and other components in the vicinity.

The torque produced by an AC motor is proportional to the square of the applied voltage. Thus, a motor producing part of its rated load only needs part of its rated voltage.

Power Factor
In an induction motor, the current in the windings lags the voltage, due to the inductive reactance in the windings (Figure 10). The cosine of the amount of lag in degrees is the power factor. Power factors are 1.0 for resistive loads (heaters, etc.) and vary from close to 1 for a fully loaded motor, to as low as 0.1 for an idling motor. The actual power being consumed by the motor is:

\[(\text{Voltage}) \times (\text{Current}) \times (\text{Power Factor})\]

A lightly loaded induction motor has low power factor. As the motor reaches its rated load, its power factor gets closer to 1. How close it gets to 1 will depend on the motor's internal design. Values around 0.65 are typical of single-phase motors.

The MTE1122 calculates motor loading by measuring the time between current and voltage zero-crossings, in effect, power factor. When the load on the motor is low, the power consumed by the motor can be lowered by lowering the voltage applied to the motor, which is done by turning a triac on at the proper time during the voltage cycle. (Figure 10 for waveforms.) The resulting voltage across the motor, and the zero-crossing times, are monitored, and adjusted on a cycle-by-cycle basis, as determined by the proprietary algorithms in the MTE1122. At no load, the voltage to the motor can be as low as 85 VAC, instead of the usual 120 VAC. Power consumption can be cut by as much as 58%, depending on load, and operating temperature lowered by as much as 45°F. Refer to the system block diagram in Figure 1, and the graph in Figure 2 and Table 1.

A motor powered by the MTE1122 and this energy management control circuit will draw less current. Its power factor will also be improved; however, the power factor seen by the line will NOT be improved.

ENERGY MEASUREMENT
To measure the true power of an induction motor requires the use of a true-RMS power meter, one that will measure non-sinusoidal waveforms. Models of this type of instrument are available from Fluke and Tektronix, among others.

Measurements for this Application Note were made using the following equipment:

Hampden Engineering Corp:
CSM-100 1/3 HP motor
DYN-100A Dynamometer
RL-100A Load bank
HPT-100 Digital Photo Tachometer
Tektronix Corp:
THM560 Scopemeter
A622 Current probe

The voltage, current, and true-RMS power supplied to the Energy Management Control System driving a 1/3 HP motor were measured with the meter and current probe. The motor RPM was measured with the photo-tach. The torque supplied by the motor was measured with the dynamometer, which also supplied the adjustable load on the motor.

Motor Power Out in Watts is calculated by:
\[P = (\tau n)/5.18\]

Power Out in Horsepower is calculated by:
\[P = (\tau n)/7142.72\]

\(\tau\) is in Newton-meters
n is RPM

Efficiency, in percent (%), is calculated by:
\[(\text{Power Out})/(\text{Power In}) \times 100\]
CIRCUIT DETAILS

The MTE1122 consists of a high-performance 8-bit microcontroller (U3) with embedded proprietary algorithms, which monitors the voltage across the motor (U1), the voltage zero-crossing (Q2) and the current zero-crossing (by monitoring the signal on Q3). By measuring the time between voltage and current zero-crossings, it calculates the amount of load on the motor.


C2 through C6 and components in between rectify and filter line voltage to provide Vcc.

Q1 and associated components provide power-up reset for the MTE1122.

L1 and C8 form an LC filter for the 5V power supply.

U2 is an opto-triac to trigger the power triac. Q3 is the triac, which in this circuit is rated at 15A.

D3 and R7 are used to indicate "normal operation" of the circuit, and may be left out if desired.

As stated above, it turns out that the energy consumption of a motor running only partly loaded can be lowered by decreasing the current flowing into the motor windings. This can be accomplished by lowering the voltage across the motor windings. If the voltage is not increased when the motor load increases, the internal reactance of the motor decreases, and the windings will draw too much current, and could overheat and be damaged. Because the MTE1122 is an intelligent controller, it is able to monitor motor voltage and motor load, and make corrections within 8 ms, well before there is any potential for motor damage.

See Figure 10 for circuit waveforms.

ENERGY SAVINGS

Reducing the voltage to the motor cuts its power draw. By reviewing Table 1, it can be seen that at no load, the 1/3 HP test motor is dissipating 120W, much of it as heat. With the MTE1122 managing the power load, this drops to 50W, a savings of 58%. At full load, the figures are, respectively, 428W and 406W, for a savings of 5%. The degree of savings are presented in Table 2.

This data is presented graphically in the following figures.

Actual performance figures may vary based on motor size, motor load and motor construction.

ALTERNATIVE APPROACH

There is another way to increase motor efficiency. This approach is to add another winding to the motor, and phase-shift it with capacitance. This produces what is known as a capacitor-run motor, and results in a motor with a power factor close to 1.0 regardless of its load, and considerably lowered idle power consumption. It is a more efficient motor, and produces less vibration.

This approach, however, is neither cost-effective in motors less than 1 hp, nor in motors for residential use.

Thus, for lowest-cost approaches, use of the MTE1122 and associated circuitry is probably the best method of improving motor efficiency.
FIGURE 2: ENERGY SAVINGS

FIGURE 3: MOTOR SPEED CHANGE
FIGURE 4: MOTOR EFFICIENCY

FIGURE 5: EFFICIENCY IMPROVEMENT
FIGURE 6: MOTOR CURRENT DRAW

FIGURE 7: MOTOR POWER DRAW
FIGURE 8: ENERGY SAVINGS FOR 1/4 HP MOTOR

ENERGY SAVINGS for 1/4 HP MOTOR
110V and 220V

- 110V EMC vs no EMC
- 220V EMC vs no EMC
- 220V no EMC vs 110V no EMC
### TABLE 1: OPERATING PARAMETER COMPARISONS

#### 1/3 HP Motor without E.M.C.

<table>
<thead>
<tr>
<th>Load (%)</th>
<th>Load (Nm)</th>
<th>Vrms (A)</th>
<th>Irms (A)</th>
<th>Power Factor</th>
<th>Power In (W)</th>
<th>RPM</th>
<th>Power Out (W)</th>
<th>Power Out (HP)</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>115</td>
<td>5.7</td>
<td>0.18</td>
<td>120</td>
<td>1791</td>
<td>0</td>
<td>0.00</td>
<td>0.2</td>
</tr>
<tr>
<td>10</td>
<td>0.14</td>
<td>115</td>
<td>5.7</td>
<td>0.20</td>
<td>130</td>
<td>1788</td>
<td>26</td>
<td>0.04</td>
<td>20.1</td>
</tr>
<tr>
<td>20</td>
<td>0.29</td>
<td>115</td>
<td>5.7</td>
<td>0.24</td>
<td>160</td>
<td>1781</td>
<td>54</td>
<td>0.07</td>
<td>33.7</td>
</tr>
<tr>
<td>30</td>
<td>0.43</td>
<td>115</td>
<td>5.7</td>
<td>0.29</td>
<td>193</td>
<td>1777</td>
<td>80</td>
<td>0.11</td>
<td>41.4</td>
</tr>
<tr>
<td>40</td>
<td>0.57</td>
<td>115</td>
<td>5.7</td>
<td>0.35</td>
<td>229</td>
<td>1768</td>
<td>105</td>
<td>0.14</td>
<td>46.0</td>
</tr>
<tr>
<td>50</td>
<td>0.72</td>
<td>115</td>
<td>5.8</td>
<td>0.37</td>
<td>249</td>
<td>1764</td>
<td>133</td>
<td>0.18</td>
<td>53.3</td>
</tr>
<tr>
<td>60</td>
<td>0.86</td>
<td>115</td>
<td>5.8</td>
<td>0.42</td>
<td>280</td>
<td>1758</td>
<td>158</td>
<td>0.21</td>
<td>56.4</td>
</tr>
<tr>
<td>70</td>
<td>1.00</td>
<td>115</td>
<td>6.0</td>
<td>0.46</td>
<td>315</td>
<td>1750</td>
<td>183</td>
<td>0.25</td>
<td>58.0</td>
</tr>
<tr>
<td>80</td>
<td>1.14</td>
<td>116</td>
<td>6.1</td>
<td>0.49</td>
<td>348</td>
<td>1744</td>
<td>208</td>
<td>0.28</td>
<td>59.7</td>
</tr>
<tr>
<td>90</td>
<td>1.29</td>
<td>115</td>
<td>6.3</td>
<td>0.53</td>
<td>386</td>
<td>1736</td>
<td>234</td>
<td>0.31</td>
<td>60.6</td>
</tr>
<tr>
<td>100</td>
<td>1.43</td>
<td>116</td>
<td>6.5</td>
<td>0.57</td>
<td>428</td>
<td>1727</td>
<td>258</td>
<td>0.35</td>
<td>60.3</td>
</tr>
</tbody>
</table>

#### 1/3 HP Motor with E.M.C.

<table>
<thead>
<tr>
<th>Load (%)</th>
<th>Load (Nm)</th>
<th>Vrms (A)</th>
<th>Irms (A)</th>
<th>Power Factor</th>
<th>Power In (W)</th>
<th>RPM</th>
<th>Power Out (W)</th>
<th>Power Out (HP)</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>113</td>
<td>3.1</td>
<td>0.14</td>
<td>50</td>
<td>1794</td>
<td>0</td>
<td>0.00</td>
<td>0.4</td>
</tr>
<tr>
<td>10</td>
<td>0.14</td>
<td>113</td>
<td>3.2</td>
<td>0.19</td>
<td>68</td>
<td>1786</td>
<td>26</td>
<td>0.04</td>
<td>38.4</td>
</tr>
<tr>
<td>20</td>
<td>0.29</td>
<td>113</td>
<td>3.5</td>
<td>0.26</td>
<td>104</td>
<td>1775</td>
<td>54</td>
<td>0.07</td>
<td>51.7</td>
</tr>
<tr>
<td>30</td>
<td>0.43</td>
<td>113</td>
<td>3.8</td>
<td>0.32</td>
<td>138</td>
<td>1764</td>
<td>79</td>
<td>0.11</td>
<td>57.4</td>
</tr>
<tr>
<td>40</td>
<td>0.57</td>
<td>113</td>
<td>4.1</td>
<td>0.38</td>
<td>178</td>
<td>1755</td>
<td>104</td>
<td>0.14</td>
<td>58.7</td>
</tr>
<tr>
<td>50</td>
<td>0.72</td>
<td>113</td>
<td>4.3</td>
<td>0.42</td>
<td>206</td>
<td>1749</td>
<td>132</td>
<td>0.18</td>
<td>63.8</td>
</tr>
<tr>
<td>60</td>
<td>0.86</td>
<td>112</td>
<td>4.6</td>
<td>0.47</td>
<td>243</td>
<td>1740</td>
<td>156</td>
<td>0.21</td>
<td>64.3</td>
</tr>
<tr>
<td>70</td>
<td>1.00</td>
<td>112</td>
<td>4.9</td>
<td>0.51</td>
<td>281</td>
<td>1730</td>
<td>181</td>
<td>0.24</td>
<td>64.3</td>
</tr>
<tr>
<td>80</td>
<td>1.14</td>
<td>112</td>
<td>5.3</td>
<td>0.55</td>
<td>329</td>
<td>1722</td>
<td>205</td>
<td>0.27</td>
<td>62.3</td>
</tr>
<tr>
<td>90</td>
<td>1.29</td>
<td>112</td>
<td>5.6</td>
<td>0.59</td>
<td>371</td>
<td>1713</td>
<td>231</td>
<td>0.31</td>
<td>62.2</td>
</tr>
<tr>
<td>100</td>
<td>1.43</td>
<td>111</td>
<td>6.0</td>
<td>0.61</td>
<td>406</td>
<td>1705</td>
<td>255</td>
<td>0.34</td>
<td>62.7</td>
</tr>
</tbody>
</table>

### TABLE 2: ENERGY SAVINGS

<table>
<thead>
<tr>
<th>LOAD (%)</th>
<th>Improvement in Efficiency (%)</th>
<th>Energy Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>140.4</td>
<td>58.3</td>
</tr>
<tr>
<td>10</td>
<td>91.0</td>
<td>47.7</td>
</tr>
<tr>
<td>20</td>
<td>53.3</td>
<td>35.0</td>
</tr>
<tr>
<td>30</td>
<td>38.8</td>
<td>28.5</td>
</tr>
<tr>
<td>40</td>
<td>27.7</td>
<td>22.3</td>
</tr>
<tr>
<td>50</td>
<td>19.8</td>
<td>17.3</td>
</tr>
<tr>
<td>60</td>
<td>14.0</td>
<td>13.2</td>
</tr>
<tr>
<td>70</td>
<td>10.8</td>
<td>10.8</td>
</tr>
<tr>
<td>80</td>
<td>4.4</td>
<td>5.5</td>
</tr>
<tr>
<td>90</td>
<td>2.7</td>
<td>3.9</td>
</tr>
<tr>
<td>100</td>
<td>4.1</td>
<td>5.1</td>
</tr>
</tbody>
</table>
FIGURE 9: WAVEFORMS

MOTOR WAVEFORMS
NORMAL LINE CONNECTION

MOTOR WAVEFORMS
WITH EMC1122
Note: Forms 2-8 ref'd to sig gnd. All are 5 ms per division. 110V line.
### TABLE 3: BILL OF MATERIALS FOR MTE1122 ENERGY MANAGEMENT CONTROL DEMO BOARD

<table>
<thead>
<tr>
<th>Item</th>
<th>Qty.</th>
<th>Reference</th>
<th>Value</th>
<th>Description</th>
<th>Mfg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C1</td>
<td>100 µF</td>
<td>16V</td>
<td>electrolytic</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>C2</td>
<td>1 µF</td>
<td>250VAC</td>
<td>film</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>C3</td>
<td>220 µF</td>
<td>16V</td>
<td>electrolytic</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>C4, C5</td>
<td>100 pF</td>
<td></td>
<td>ceramic</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>C6</td>
<td>0.1 µF</td>
<td></td>
<td>ceramic</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>C7</td>
<td>1 µF</td>
<td>50V</td>
<td>electrolytic</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>C8</td>
<td>2.2 µF</td>
<td>250VAC</td>
<td>film</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>C9, C10</td>
<td>0.1 µF</td>
<td></td>
<td>film</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>D1, D2, D8</td>
<td></td>
<td>1N4007</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>D3</td>
<td></td>
<td>LED</td>
<td>green</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>D4</td>
<td></td>
<td></td>
<td>1N5226</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>D5</td>
<td></td>
<td></td>
<td>1N4733</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>D6, D7</td>
<td></td>
<td>1N5230</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>L1</td>
<td>100 µH</td>
<td>4632</td>
<td>RF Choke</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>Q1</td>
<td></td>
<td>TP2907</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>Q2</td>
<td></td>
<td>TP2222A</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>Q3</td>
<td></td>
<td>Q4015L5</td>
<td>Teccor</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>R11</td>
<td>470 1/2W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>R2</td>
<td>330 1/2W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>R3</td>
<td>560</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>R4</td>
<td>270 1/2W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>R5, R15</td>
<td>30K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>R14</td>
<td>30K 1/2W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>R6</td>
<td>15K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>R7</td>
<td>560</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>R8</td>
<td>2.4K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>R9</td>
<td>1M 1/2W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>R10, R13</td>
<td>562K 1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>R11, R12</td>
<td>12.1K 1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>R16</td>
<td>10K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>U1</td>
<td>TLC271CP</td>
<td>opamp</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>U2</td>
<td>TLP3023</td>
<td>opto-triac</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>U3</td>
<td>MTE1122</td>
<td></td>
<td>Microchip</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>Y1</td>
<td>4 MHz</td>
<td>ceramic resonator</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td></td>
<td></td>
<td>heatsink</td>
<td>as needed</td>
</tr>
</tbody>
</table>

**Note 1:** C2, 8, 9, 10 MUST be AC-rated capacitors. THIS IS CRUCIAL!

**Note 2:** Q3 can be sized to fit the load. A 400V 15A part is called out in the parts list and on the schematic; a 600V 25A part is also listed on the schematic for reference. Nearly any triac can be used here, as long as its trigger current does not exceed that supplied by U2 (typically 50 mA). For higher current applications, two SCR's back-to-back perform well. Performance is improved slightly if the device is NOT operated at its current limit.

**Note 3:** Any opto-triac meeting the current-transfer and current handling specs of the TLP3023 can be used.

**Note 4:** A heat sink is called out. Its size will depend on the particular Triac used, the operating temperature, and the load. Contact a heat sink manufacturer for specific information.
NOTES:
1. ALL RESISTORS ARE 5% EXCEPT WHERE NOTED
2. ALL RESISTORS ARE 1/8 WATT
3. GND SYMBOL IS CIRCUIT COMMON; IT IS NOT TIED TO AC GROUND

Figure 11: Schematic Diagram
SECTION 4

SERIAL EEPROM APPLICATION NOTES

AN601  EEPROM Endurance Tutorial................................................................. 4-1
AN602  How to get 10 Million Cycles out of your Microchip Serial EEPROM........4-5
AN608  Converting to 24LCXXB and 93LCXX Serial EEPROMs .......................... 4-7
AN609  Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller 4-9
AN610  Using the 24LC21 Dual Mode Serial EEPROM ........................................ 4-17
AN613  Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports ....... 4-23
AN614  Interfacing the 8051 with 2-wire Serial EEPROMs..................................... 4-37
INTRODUCTION

The endurance of an EEPROM-based device will be quoted by a manufacturer in terms of the minimum number of erase/write cycles (write cycles) that the device is capable of sustaining before failure. A write cycle is generally considered to be the operation that changes data in a device from one value to the next.

There are several EEPROM-based devices available on the market. Microchip Technology Incorporated makes three general types of EEPROM-based product: Serial EEPROMs, Parallel EEPROMs, and EEPROM-based Microcontrollers. As a manufacturer of many EEPROM products, Microchip is concerned with endurance and continues to try to educate its customers on the importance of this topic.

There are many differences in the interpretation of “endurance” that can result in misleading or inaccurate information being used in design decisions. This paper hopes to clear up any questions that the customer may have in the subject of endurance, without becoming so technical that the information given is not helpful.

There is no widely used standard for any type of endurance test. Each manufacturer will use their own endurance testing methodology. This report will describe all the testing options, and which tests Microchip performs on its EEPROM-based products.

The MIL-STD cycling test (Method 1033) has not been updated since 1977 and is well out of date as applied to EEPROM non-volatile memories. The standard does not distinguish the difference between block cycling and byte cycling, and gives a very poor failure criteria. Microchip does not use this standard.

BASIC TERMS

The definition of “endurance” (as applied to EEPROMs) in the first part of this introduction contains various words and phrases that require clear definition and understanding. As shown in the following paragraphs, different manufacturers will use different standards.

“Endurance cycling” is a test performed by all manufacturers (and some customers) to determine how many “write cycles” the product will achieve before failing.

The “minimum number of write cycles” is the least number of times that you can expect to subject the product to a “write cycle” before it fails.

“Failure” is a somewhat arbitrary definition, since a device only truly fails when it no longer meets the customer expectation, and does not operate in his system. A failure can be defined in this, the loosest of definitions, or the most stringent of definitions (whereby a device would fail if it did not meet any of the data sheet parameters), as well as a wide range in between.

For example, if the device did not correctly store data into a particular address that the customer was not using, then the device would work correctly for the customer but would fail a functional test set by the manufacturer. Likewise, if the device drew more current than the data sheet specified after some time, but the customer application could supply the current needed, the device would work in the customer application but would fail a parametric test set by the manufacturer.

Microchip uses the most stringent definition: A failure occurs when the device fails to meet any data sheet condition under any guaranteed operating condition of temperature and voltage.

The number of devices that can fail before a particular endurance criteria is not met is also somewhat flexible. Even the most quality conscious manufacturer will occasionally have a failure, so a failure level is defined. The industry standard conditions for many types of reliability tests are set by JEDEC (the Joint Electronic Device Engineering Council). JEDEC defines that if 5% or less of a given sample fails at a given endurance goal, then that goal has been met. For example, if a sample of 100 units are endurance cycled to 1 million cycles and 3 have failed at 100,000 cycles and a further 7 have failed at 1 million cycles, then the sample would have an endurance of 100,000 cycles.

Microchip uses a more stringent criteria for endurance: no more than 2.5% of devices can have failed for the given endurance goal to have been met.

A “write cycle” is also a somewhat flexible definition since almost every customer will write the device in a different way. For example, if the customer application uses only the first three bytes of the array to store variable data, and the remainder of the array is used as a lookup table, then a write cycle will be complete when the three data bytes have been re-written to their new data state.
A write cycle is often described as an erase/write cycle, since almost all technologies employ an "auto-erase" before the data is actually written to the array. This is also used by Microchip, but we will use the term "write cycle" since the auto-erase is invisible to, and cannot be suppressed by, the customer.

The term "data changes" is occasionally used in place of "write cycle" or "erase/write cycle." A data change will occur when an auto-erase cycle is initiated, and a second data change will occur upon the write cycle, therefore, one "erase/write cycle" is equivalent to two "data changes." The term "data change" also implies that a different type of cycling is being used than "erase/write cycle." This will be described later.

The term "write cycle" does not define under what conditions the cycling was done (unless explicitly stated) nor does it define the type of cycling that was done. The endurance cycling can be done at any number of conditions of voltage and temperature (e.g., 85°C and 5.5V, or 25°C and 5.0V) that may or may not meet with a customer's application. The cycling mode used in endurance cycling can affect the endurance of the product. All these effects will be described later.

Microchip uses the most stringent conditions that are reasonable for endurance cycling. We use byte or page mode cycling at a temperature of 85°C at 5.5V. All data not explicitly defined at other conditions is taken at these conditions.

A "read cycle" is completed when any number of bytes of data have been read from the device. For the FLOTOKX-design EEPROM-based devices made by Microchip a read cycle does not affect endurance, since the data in the EEPROM is not changed. Other technologies, such as Ferroelectric technology, may have a limited number of read cycles since data is corrupted during a read.

System Design Considerations

There are a number of design considerations that the system designer can use to maximize the endurance of an EEPROM-based device, if endurance is the application's limiting factor.

As will be described in more detail later, if the designer has any control over certain environmental or operation conditions he should observe the following basic guidelines:

- Keep the application temperature as low as possible
- Keep the application voltage (or the Vcc voltage on the EEPROM-based device) as low as possible
- Write as few bytes as possible
- Use page write features wherever possible
- Write data as infrequently as possible

With these basic guidelines applied to the fullest extent, the endurance of EEPROM-based devices can be extended well beyond the guaranteed minimum endurance. Under certain very specific conditions, Microchip Serial EEPROMs have been shown to last well over 100 million cycles.

WRITE MODES IN EEPROMS

There are three ways that EEPROM-based devices can have the entire array data contents changed. These are: byte mode, page mode, and block (or bulk) mode. Some types of devices support all three modes, others may only support one or two modes. The mode that you use to write an EEPROM-based device will affect the long term endurance of the product.

Byte mode writing is used when the contents of the array are changed one byte at a time. For many devices this is the only user-accessible write mode available. To change the entire contents of a Serial EEPROM in this way would take up to 10 seconds (using 10 ms per page on a 64K Serial EEPROM).

Page mode writing is a popular feature on many new designs of EEPROM memory products. This feature allows up to 8 bytes of data to be written to the memory in the same time that one byte would normally take. In this mode, the write time for a 64K Serial EEPROM can be cut from eight seconds to one second.

Block cycle is generally a test mode used by EEPROM manufacturers to make it easier to test the products. Some types of EEPROM-based products have these modes as user options (such as the ERAL and WRAL mode in 93CXX products, or the Chip Clear mode in 28CXXX products) but generally this mode is not user accessible. A block write can be done in as little as 1 ms, allowing millions of write cycles to be completed in a few hours.

A general rule to follow in choosing write modes is that the larger the number of bytes being written in a single instruction, the longer the device will last. For example, in byte mode a device might start to fail after 300,000 cycles under a particular set of conditions, but the device may last 600,000 cycles in page mode under the same conditions. In block mode the device might last 1 million cycles, under the same conditions.

The reason for this is related to the internal design of any FLOTOKX EEPROM-based product. In these devices, an internal "charge pump" takes the applied Vcc voltage (typically 2.5 to 5.5V) and increases it to 15 to 25V. This voltage is required to induce the "Fowler-Nordheim Tunneling" or "Enhanced Emission" effects that are used to program and erase EEPROM-based devices. The specific effect that is used is manufacturer-dependent. Microchip EEPROMs program by Fowler-Nordheim Tunneling.
The charge pump voltage is used to program however many EEPROM-cells are being programmed. For example, in byte mode, all the cells in a byte (8 to 16) are biased with the charge pump voltage. In block mode, all the cells in the array (up to 100,000, depending on the device) are biased with the charge pump voltage. The charge pump is like a current source during conditions of high load, so the voltage put out by the charge pump will be reduced slightly if more bytes are being written. If the whole array is being programmed then the charge pump voltage will be significantly reduced, but the programming current lpp will be very high.

Generally, the lower the charge pump voltage the better the endurance (there is a limit since the charge pump voltage needs to be high enough to program the cell) and so the best endurance is generally achieved by using block mode cycling. Page mode is worse than block mode, but better than byte mode. Block mode is generally not a very useful cycling mode to the end user, since the data contents in the whole array will be changed to the same value (generally 00 or FF).

When Microchip tests EEPROM-based products we use byte mode cycling on devices which do not have a page mode, and page mode cycling for those that do. We encourage our customers to use page mode writing on all products which have page mode, to ensure high endurance.

**ENDURANCE TESTING METHODOLOGIES**

Different manufacturers use different ways to both endurance cycle and test EEPROM-based products. There is no standard for endurance cycling, or testing devices after cycling.

There are two groups of testing that Microchip performs on all products: qualification and production. Qualification testing is done for all new products, and major changes to a product or manufacturing process. Production testing is done on all devices shipped to customers.

Qualification testing at Microchip is used to test the reliability of new products, and to guarantee that the device is reliable. A great deal of testing is done, including endurance testing on all EEPROM-based products. Endurance cycling is done at the maximum rated data book value, generally 85°C or 125°C. After the rated number of cycles (10,000, 100,000, or 1,000,000) the sample (around 300 from multiple wafer lots) is tested to a full production test program. After endurance, the units are subject to "data retention" to guarantee that the required 40 years of data retention will be achieved, after the maximum number of cycles has been completed.

Endurance cycling is done under the conditions previously described, and the data retention test is performed after this. After the data retention stress is completed (which takes six weeks) the devices are tested again, to confirm the functionality of the device to all data sheet parameters.

No more than 2.5% failures are allowed after endurance, and a failure rate higher than 100 FITs after 1000 hours of data retention stress (equivalent to more than 10 years at 55°C) is unacceptable.

Production testing is done by Microchip on all devices shipped to a customer. Production testing begins immediately after a wafer lot is finished being processed, continuing in various stages until the devices are shipped to a customer.

The first tests that are done on EEPROM-based products at Microchip are called wafersort. They are done before the wafer is cut up into dice for assembly. There is a series of tests which include large numbers of write cycles (up to 5000) to ensure reliability by weeding out weak devices so they never get shipped.

After assembly full testing is done which includes further write cycles across the guaranteed temperature range, to ensure device functionality at the temperature extremes. After the normal production testing a sample of 128 units is taken from every wafer lot, and cycled to 10,000 cycles (Parallel EEPROMs) or 1,000,000 cycles (Serial EEPROMs and EEPROM microcontrollers) at 85°C using the conditions already described. After endurance testing the devices are tested for functionality at 85°C.

Any sample of Parallel EEPROMs which shows significant failures at the end of cycling causes the entire wafer lot to be pre-cycled prior to production testing. Serial EEPROMs and EEPROM-based microcontrollers do not receive the same disposition. Lots with significant failures come under scrutiny and full failure analysis with corrective actions is done. This is, however, a rare occurrence.

The testing that Microchip does is unique. Manufacturers will generally do different testing from each other. Microchip firmly believes that our testing ensures excellent quality and reliability.

**THE EFFECT OF TEMPERATURE ON EEPROM ENDURANCE**

The temperature at which cycling is done will affect the number of write cycles that can be executed before the device fails. The higher the temperature, the worse the endurance will be. Generally, and approximately, a device which fails at 10 million cycles at 25°C will fail at 2 million cycles at 85°C and 1 million cycles at 125°C. The reasons for this are not conclusive (although there is much technical literature supporting one theory or another) but it is apparent that the failure mode of EEPROM cells (generally considered to be electron trapping in the tunnel dielectric causing shielding and dielectric breakdown) is strongly dependent on temperature.
Data taken by Microchip suggests that if the typical failure of an EEPROM-based device is 10 million cycles at 25°C, the mean failure will occur at other temperatures according to the following table:

**TABLE 1: TEMPERATURE MEAN FAILURE**

<table>
<thead>
<tr>
<th>Write Cycle Temperature</th>
<th>Mean Failure (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40°C</td>
<td>37.1 million</td>
</tr>
<tr>
<td>0°C</td>
<td>16.7 million</td>
</tr>
<tr>
<td>25°C</td>
<td>10.0 million</td>
</tr>
<tr>
<td>40°C</td>
<td>7.4 million</td>
</tr>
<tr>
<td>55°C</td>
<td>5.4 million</td>
</tr>
<tr>
<td>70°C</td>
<td>4.0 million</td>
</tr>
<tr>
<td>85°C</td>
<td>2.9 million</td>
</tr>
<tr>
<td>100°C</td>
<td>2.2 million</td>
</tr>
<tr>
<td>125°C</td>
<td>1.3 million</td>
</tr>
</tbody>
</table>

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance™ disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics.

As is clearly seen, any cycling done at 25°C can be misleading in the extreme if the application requires a device that can be cycled 10 million times at, say, 55°C.

**THE EFFECT OF VOLTAGE ON EEPROM ENDURANCE**

The voltage at which a device is written can also affect the endurance. This is simply because the charge pump (used to program and erase EEPROM cells) is more powerful at higher voltages. As has already been described, a higher programming voltage will reduce the endurance of an EEPROM cell, and a stronger charge pump will produce a higher voltage.

Data taken by Microchip suggests that if typical failure of an EEPROM-based device is 1 million cycles when endurance cycling is done at 5.5V, mean failure occurs at other temperatures according to the following table:

**TABLE 2: VOLTAGE MEAN FAILURE**

<table>
<thead>
<tr>
<th>Endurance Cycling Voltage</th>
<th>Mean Failure (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5V</td>
<td>1.0 Million</td>
</tr>
<tr>
<td>5.0V</td>
<td>1.2 Million</td>
</tr>
<tr>
<td>4.5V</td>
<td>1.4 Million</td>
</tr>
<tr>
<td>4.0V</td>
<td>1.7 Million</td>
</tr>
<tr>
<td>3.5V</td>
<td>2.0 Million</td>
</tr>
<tr>
<td>3.0V</td>
<td>2.4 Million</td>
</tr>
<tr>
<td>2.5V</td>
<td>2.8 Million</td>
</tr>
<tr>
<td>2.0V</td>
<td>3.3 Million</td>
</tr>
</tbody>
</table>

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics.

**THE EFFECT OF WRITE MODE ON EEPROM ENDURANCE**

As has been discussed there are three basic ways of writing data to an EEPROM-based device:

- Byte mode
- Page mode
- Block mode

This is related to the strength of the charge pump in applying the required programming voltages to the EEPROM cells.

Data taken by Microchip suggests that if the typical failure of an EEPROM-based device is 1 million cycles when the endurance cycling is done in byte mode, the mean failure will occur in other modes according to the following table:

**TABLE 3: ENDURANCE MEAN FAILURE**

<table>
<thead>
<tr>
<th>Endurance Cycling Mode</th>
<th>Mean Failure (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>1.0 Million</td>
</tr>
<tr>
<td>Page</td>
<td>4.6 Million</td>
</tr>
<tr>
<td>Block</td>
<td>13.2 Million</td>
</tr>
</tbody>
</table>

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics. This data was taken from a Microchip 24LC16.

As you can see, the use of the block cycle data to guarantee endurance can be misleading.

**THE TOTAL ENDURANCE PREDICTIVE SOFTWARE**

Microchip has a Windows®-based software model called Total Endurance. This program, based on all of the customers endurance parameters, will predict the failure level at the expected end of application life. This tool is invaluable for system designers who would like to fine-tune their application in favor of endurance. It is available now from your local Microchip distributor.
INTRODUCTION

Microchip Technology Incorporated recently became the first manufacturer of Serial EEPROMs to rate the endurance of 1K to 16K 2-Wire protocols and 1K to 4K 3-Wire protocols at 10 million cycles. This gives the highest endurance available from any manufacturer on such a wide range of Serial EEPROMs. Microchip is also the world leader in endurance application support, including Total Endurance™ software, which allows customers to determine exactly what their endurance will be in their application. This application note will explain our 10 million cycle guarantee in the context of Total Endurance and the customer application.

ENDURANCE

Endurance is the term used to define the number of times an EEPROM-based device can be erased and re-written before an unacceptable failure rate has occurred. Microchip defines an unacceptable failure rate as 2.5%, which is half the “industry standard” (JEDEC) failure rate of 5%.

Microchip uses the industry standard cycling conditions to guarantee 10 million cycles. These are the same conditions used by the major EEPROM manufacturers, unless specifically stated otherwise. The conditions are 25°C block cycle at 5.0V. The data pattern that is written to the device is supplier-dependent. Microchip uses a checkerboard pattern to better simulate actual use.

10 MILLION CYCLE GUARENTEE

We take a great deal of data on the endurance of our devices. We run experiments over temperature, voltage, array size, data pattern, cycling type and other erase/write cycling conditions, to continually learn about our products. The data we take is continually fed back into our manufacturing process tracking, and is also used to periodically update our application predictive software, Total Endurance. The lessons that we learn are passed on to our customers.

Microchip has shown many times before that temperature has the strongest effect on endurance. The higher the temperature of your application is, the lower your endurance will be. For example, no EEPROM manufacturer can deliver a product that will achieve 1 million cycles at 125°C, but many can deliver 1 million cycles at room temperature. Microchip can provide 10 million cycles on all our Serial EEPROMs at room temperature, using the standard cycling conditions.

You can ensure the lowest possible failure rate at 10 million cycles by using the EEPROM in ways that are easiest on endurance. Use the lowest temperature possible. Use the lowest voltage possible. Write as few bytes as possible (the number of reads is unlimited), and use page modes if the device you are using has one (Typically all 2-wire protocols have a page mode, but 3-wire protocols do not).

The best way to determine the endurance of your application is to use the Total Endurance software. This will give accurate results over a very wide range of application-specific conditions, including array size, temperature, and voltage (the main effects on cycling).

The most important thing to remember is that endurance is application-specific. Each application will result in a different endurance level, based on the temperature, voltage, and other conditions under which the application operates. Currently Microchip is the only manufacturer to offer such a wide range of tools to help guide the user in his or her application of Serial EEPROMs.
Microchip Technology Inc. is constantly striving to give you, our customer, greater value. In the Serial EEPROM market this means creating devices that exhibit greater reliability, that consume less power, have more features and are priced aggressively with respect to our competitors. We have created the 24LCXXB and 93LCXX families of Serial EEPROMs to meet all of these goals and more. We are proud to announce that these families of EEPROMs are the world’s first to be guaranteed to 10 Million Erase/Write cycles. Please see Table 1 below for a complete list of the benefits for converting from “C” designated parts from Microchip and other vendors.

For most applications, there are no hardware or software compatibility issues in converting to the “LC” parts, you can drop them into the socket and you will be off and running. A small percentage of designs could require some modification to your hardware or software. Please review Table 2 and Table 3 to ensure that the “LC” parts will work correctly in your systems.

**Table 1: Features and Benefits of Conversion**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher Endurance</td>
<td>Opens new applications for Serial EEPROMs</td>
</tr>
<tr>
<td></td>
<td>Lower FIT rate at given Write Cycle point</td>
</tr>
<tr>
<td>Smaller Die</td>
<td>Lower Cost</td>
</tr>
<tr>
<td>Lower Voltage / Lower Power</td>
<td>Enables new battery powered uses</td>
</tr>
<tr>
<td></td>
<td>Allows for lower overall system Vcc</td>
</tr>
<tr>
<td>Smaller Process Geometry</td>
<td>Avoids obsolescence of older technology</td>
</tr>
<tr>
<td>Schmitt Trigger Inputs (24LCXXB)</td>
<td>Filtering for better noise immunity</td>
</tr>
<tr>
<td>Rotated Pinout Versions (93LCXX/SN)</td>
<td>Conforms to alternate industry pinout</td>
</tr>
<tr>
<td>Hardware Write Protect (24LCXXB)</td>
<td>Enhanced data protection</td>
</tr>
</tbody>
</table>

**Table 2: Conversion Issues for 24LCXXB**

<table>
<thead>
<tr>
<th>Conversion Issue</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addressing</td>
<td>The “LC” versions do not have active address pins and, therefore, you can only have one of these devices on the I²C™ bus.</td>
</tr>
<tr>
<td>Write Speed</td>
<td>The “LC” parts have a maximum write time of 10ms, while the “C” parts have a write time of 1ms.</td>
</tr>
<tr>
<td>Write Disable Voltage</td>
<td>The “LC” versions will write down to 1.5V, while the “C” versions will disable writing below 4.0V.</td>
</tr>
<tr>
<td>Write Protection</td>
<td>The “C” parts have either no write protection or protection on one half of the array, the “LC” parts all have full array protection, implemented by connecting pin 7 to Vcc.</td>
</tr>
</tbody>
</table>
### TABLE 3: CONVERSION ISSUES FOR 93LCXX

<table>
<thead>
<tr>
<th>Conversion Issue</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Speed</td>
<td>The &quot;LC&quot; versions have a maximum write time of 10ms, compared to 1ms for &quot;C.&quot;</td>
</tr>
<tr>
<td>Write Disable Voltage</td>
<td>The &quot;LC&quot; parts will write down to 1.5V while the &quot;C&quot; parts will only write down to 4.0V.</td>
</tr>
<tr>
<td>Start of Write Cycle</td>
<td>The &quot;LC&quot; versions start the write cycle on the low going edge of chip select, the &quot;C&quot; versions start on the last data bit clock.</td>
</tr>
</tbody>
</table>
INTRODUCTION

There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption and low cost, Serial EEPROMs are a popular choice for non-volatile storage.

Microchip Technology Incorporated addresses this need by offering a full line of Serial EEPROMs covering industry standard serial communication protocols for 2-wire and 3-wire communication. The theory, operation and differences of these two protocols are discussed in detail in Microchip's application note AN536. The reader should refer to AN536 if unfamiliar with 2-wire and/or 3-wire communication protocols. Serial EEPROM devices are available in a variety of densities, operational voltage ranges and packaging options.

Microchip realizes that its customer base is very broad, and because of this, different microcontrollers are used to interface to Serial EEPROMs. One of the microcontrollers used in these applications is the Motorola® 68HC11. In order to simplify the design process, Microchip has written a 68HC11 assembly code to communicate with our 2-wire and 3-wire parts that is verified and tested to function properly.

There are 13 programs written for inclusion in this application note. A listing of one of the programs is included with this application note as an example. The source code for all of the programs is available for downloading via Microchip's Bulletin Board (BBS). Users may consult the index of the Microchip Embedded Control Handbook for log-on instructions for the BBS. Once logged on to the Microchip BBS, select the FILE LIBRARY (download files) option from the main menu. Next, select a library. This application note is located in the MEM_APPS file library. Once in the MEM_APPS file library, the proper application note can be selected and downloaded by following the directions supplied by the BSS system. Only one file needs to be downloaded from the Microchip BBS to get all of the source code. All source code files are combined and compressed into one downloadable file called AN609.zip.

For Microchip 2-wire devices (24CXXA, 24LCXX, 24LCXXB, 24AAXX devices, excluding 24XX32 and 24XX65 devices), there are three programs to perform some of the basic communication functions.

- SRDMT2W.ASM - 2-Wire Sequential Read
- BW4MT2W.ASM - 2-Wire Sequential Write
- BW4PMT2W.ASM - 2-Wire Byte Write with Data Polling

Refer to the data sheets in the Microchip Data Book for the 2-wire devices listed above for explanations of sequential read, sequential write and data polling.

Microchip also offers a powerful and flexible family of products referred to as Smart Serial. These devices use the same 2-wire interface as described above, but have added intelligence not found on Microchip's other 2-wire devices. These devices include the 24C32, 24LC32, 24AA32, 24C65, 24LC65, and 24AA65. Among Smart Serial features are: split erase/write cycle endurance (user selectable regions of the device with two different endurance ratings), a 64-byte write cache and the ability to permanently write protect part or all of the array.

- RRDMT65.ASM - Random Read from 64K Smart Serial
- SRDMT65.ASM - Sequential Read from 64K Smart Serial
- CACWMT65.ASM - Full Cache Write to 64K Smart Serial
- WRBMMT65.ASM - Sequential Byte Write to 64K Smart Serial
- WRBPMT65.ASM - Sequential Byte Write with Data Polling to 64K Smart Serial
- SECMT65.ASM - Setting Security Features for 64K Smart Serial

Refer to individual data sheets of the parts listed for the definition and explanation of the functions.

For Microchip 3-wire devices (93CXX, 93LCXX 93LCXXB, and 93AAXX devices), there are three programs included to perform some of the basic communication functions.

- MT4BR3W.ASM - 3-Wire Multiple Word Read
- MT4BW3W.ASM - 3-Wire Multiple Word Write
- MT4BW3PW.ASM - 3-Wire Multiple Word Write with Data Polling
Refer to the individual data sheets of the 3-wire devices listed for explanations of multiple word read, multiple word write and data polling.

Figure 1 describes the hardware schematic for the interface between Microchip's 2-wire devices and the Motorola 68HC11E9. This schematic applies to the connection of the Smart Serial devices as well. Figure 2 describes the hardware schematic used to connect Microchip's 3-wire devices to the Motorola microcontroller. The schematics show the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections.

**FIGURE 1: CIRCUIT DIAGRAM FOR 68HC11 TO 2-WIRE SERIAL EEPROM INTERFACE**
FIGURE 2: CIRCUIT DIAGRAM FOR 68HC11 TO 3-WIRE SERIAL EEPROM INTERFACE
APPENDIX A: SOURCE CODE

; 2-Wire Sequential Write Program (225 bytes)
;
; This program (bwr4mt2w.*) writes 4 bytes of $A5 to a Microchip 24LCxx
; beginning at location $10 using a Motorola 68HC11 microcontroller.
; This code has been verified that it writes properly to a Microchip
; 24LCxx serial EEPROM.
;
; This program was written using a 68HC11EVBU evaluation board.
; This board has a monitor program in firmware of the 68HC11 which
; allows single stepping, register viewing, modifying, etc. Since this
; is the case, the program code will be loaded into the on-chip EEPROM.
; This EEPROM begins at $B600. The control registers are left to their
; default location of $1000 and the RAM is left to its default location.
; RAM locations of $48-$FF are used by the monitor program and are not
; available for program use. Therefore, the stack pointer is set a $47
; and will be able to use all of the RAM to $00, and the RAM variables
; begin at location $100 and go up from there. I cannot program the
; reset vector since it is ROM space (at $FFFE), so the way I run this
; program is to use the monitor program that comes with the evaluation
; board to set the program counter to the starting address of my user
; program ($B600) and begin from there. For users who do have access
; to the reset vector, the label of the beginning program (in my case, it
; is called START) should be loaded at location $FFFE. This program was
; not assembled using a Motorola assembler, but was assembled using
; Universal Cross-Assemblers Cross-32 Meta-Assembler. It has the
; ability to assemble just about any microcontroller code. There are
; certain commands that are unique to the cross-assembler. These
; commands will be commented differently than other commands to
; be recognizable. They will look like this:
;
; ;++++++++++++++++++++++++++++++++
; Special cross-assembler command(s)
; ;++++++++++++++++++++++++++++++++

; I do not know the exact assembler commands required to accomplish
; assembly using a Motorola assembler.
;
; The crystal that comes with the evaluation board is 8 Mhz.
; With this used as the clock input, this code will output a serial
; data stream at approximately 32khz. Although this does not
; meet I2C spec of 100 khz, it communicate with the Microchip
; 24LCxx parts properly. If a different frequency crystal is used,
; the code may need to be modified to meet timing specifications.

; ;++++++++++++++++++++++++++++++++

0000  CPU  "C:\WINC32\68HC11.TBL" ; LOAD TABLE
0000  HOF  "MOTS" ; Hex output is Motorola S-records
0000  PAGE 60 ; Sets # of lines in list file
                ; to 60 before pagebreak

; ;++++++++++++++++++++++++++++++++

; 68HC11 control register locations

; ;BEGINNING OF REGISTERS

1000 =
    REGBS EQU 1000H

; ;++++++++++++++++++++++++++++++++

; ;BEGINNING OF REGISTERS

D00609A-page 4-12 © 1995 Microchip Technology Inc.
BEGINNING OF RAM VARIABLES

DATA DIRECTION REG FOR PORT C
PORT C DATA REGISTER
OFFSET FROM CONTROL REG REG.

; User defined constants

CHIDHI EQU 00000011B ; SET BOTH CLK AND DATA HI
CHIDLO EQU 00000010B ; SET CLK HI AND DATA LO
CLUDHI EQU 00000001B ; SET CLK LO AND DATA HI
CLUDLO EQU 00000000B ; SET CLK AND DATA BOTH LO
DIMASK EQU 00000001B ; BIT MASK FOR DATA IN BIT
DOMASK EQU 10000000B ; BIT MASK FOR DATA OUT BIT
SDAMASK EQU 00000001B ; BIT MASK FOR SERIAL DATA
SCKMASK EQU 00000000B ; BIT MASK FOR SERIAL CLOCK
LEDMASK EQU 00000100B ; BIT MASK FOR ACK FAILED LED

RAM variables begin at 100h

DFS 1 100H ; byte storage
DFS 1 101H ; word storage
DFS 1 102H
DFS 1 103H
DFS 1 104H

Program code cannot be placed in ROM for eval board because eval board firmware is loaded in ROM. Program code will be loaded in EEPROM (which is 512 bytes and begins at B600h).

Serial data (SDA) is located on port c, pin 0
Serial clock (SCK) is located on port c, pin 1
An acknowledge fail LED is connected to port c, pin 2. When the ack is not low like it should be, the ack failed LED is illuminated.

This is the main portion of the code. It is where the reset vector should set program counter to.

START
LDAA #00010000B ; ADDRESS OF MEMORY TO BEGIN WRITING DATA
STAA ADDR ; NUMBER OF BYTES TO WRITE OUT
LDAA #4
STAA BYTECNT ; LOAD $1000 INTO X INDEX REG
LDX #REGS
JSR STRTBIT ; GOTO STRTBIT SUBROUTINE
LDAA #10100000B
STAA TXBUFF ; TXBUFF FOR OUTPUT TOEEPROM
AN609

B615 BDB6A2              JSR TXBYTE          ; OUTPUT 1 BYTE TO SEEPROM
B618 B60102              LDAAA ADDR         ; GET ADDRESS AND LOAD IN
B61B B70100              STAA TXBUFF        ; TXBUFF FOR OUTPUT
B61E BDB6A2              JSR TXBYTE          ; OUTPUT 1 BYTE TO SEEPROM
B621 P60104              LDAB BYTECN         ; DATA BYTE TO OUTPUT IS A5H
B624 86A5 NEXTWR LDAA #10100101B       ;
B626 B70100              STAA TXBUFF        ;
B629 37                 PSHB               ; PUSH DATA BYTE COUNTER TO
B62A BDB6A2              JSR TXBUFF        ; TOP OF STACK
B62D 33                 PUSHB              ; FULL DATA BYTE COUNTER FROM
B62E 5A                 DECB               ; TOP OF STACK
B62F C100              CMPB #00H            ;
B631 26F1              BNE NEXTWR       ; NO, THEN SEND NEXT BIT
B633 BDB653              JSR STOPBIT       ; SEND STOP BIT TO BEGIN
B636 7EB600             JMP START          ; INTERNAL WRITE CYCLE
B639 STRTBIT
B639 B607               LDAA #00000111B    ;
B63B B71007             STAA DDRC          ; PORT C ALL INPUTS EXCEPT BITS 0,1,2
B63E B603               LDAA #CHIDHI       ; SET SCLK AND SDATA HI
B640 B71003             STAA PORTC         ;
B643 01                 NOP                ; OBEY PROPER START BIT SETUP
B644 01                 NOP                ; TIME
B645 01                 NOP                ;
B646 01                 NOP                ;
B647 01                 NOP                ;
B648 1D0301            BCLR PCOFF,X,SDAMASK ; SET DATA LOW FOR STOP BIT
B64B 01                 NOP                ; OBEY PROPER START BIT HOLD
B64C 01                 NOP                ; TIME
B64D 01                 NOP                ;
B64E 01                 NOP                ;
B64F 1D0302            BCLR PCOFF,X,SCKMAS  ; SET CLK LO
B652 39                 RTS                ; END START BIT SUBROUTINE

STOPBIT
B653 B607               LDAA #00000111B    ;
B655 B71007             STAA DDRC          ; PORT C ALL INPUTS EXCEPT FOR
B658 1D0301            BCLR PCOFF,X,SDAMASK ; BITS 0,1
B65B 1C0302            BSET PCOFF,X,SCKMAS  ; CLK BIT HI

;**************************************************************
Start bit output subroutine
;**************************************************************

;**************************************************************
Stop bit output subroutine
;**************************************************************

DS00609A-page 4-14 © 1995 Microchip Technology Inc.
NOP
; OBEY PROPER STOP BIT SETUP
; TIME
NOP
NOP
NOP
BSET PCOFF,X,SDAMASK
RTS
; DATA BIT HI CAUSES STOP BIT
; END STOP BIT SUBROUTINE

; Remember to wait internal write cycle time or acknowledge
; pole here until writing is complete before beginning next
; write to part.

;******************************************************************************
; This routine reads in one bit from the data line
;******************************************************************************

INBIT
LDAA #0000011OB
STAA DDRC
CLR EE....IN GUESS INPUT IS A 0
BSET PCOFF,X,SDAMASK
NOP WAIT TO READ INPUT
LDAA PORTC GET INPUT FROM SDATA
BCLR PCOFF,X,SCKMASK BRING CLK LO AFTER PORT READ
BITA #DIMASK SEE IF INPUT IS 1 OR 0
BEQ DONEIN INPUT IS A ZERO
LDAA #OFFH INPUT BIT IS ACTUALLY A 1
STAA EE_IN STORE BACK IN EE_IN
RTS

;******************************************************************************
;******************************************************************************
; This routine writes out one bit to the sdata line
;******************************************************************************

OUTBIT
LDAA #00000111B
STAA DDRC
LDAA TXBUFF WHAT ARE WE TRYING TO
BITA #OOMASK A 1 OR 0?
BEQ LOWOUT EE_OUT BIT IS 0
BSET PCOFF,X,SDAMASK HIGH NEEDS TO BE SENT OUT
BRA CONTOUT
LOWOUT BCLR PCOFF,X,SDAMASK SEND OUT A LOW
CONTOUT BSET PCOFF,X,SDAMASK SET CLOCK BIT
NOP WAIT PROPER SCLK HI TIME
;*************************************************************
;*************************************************************
; This routine outputs 1 byte of data out the sdata pin
;*************************************************************

B6A2    TXBYTE
B6A2 C608    LDAB #8 ; SET BIT COUNTER

B6A4 BDB682    TXBIT JSR OUTBIT ; SEND 1 BIT
B6A7 790100    ROL TXBUFF ; GET NEXT BIT READY TO XMIT
B6A5 5A    DECB
B6AB C100    CMPB #00H ; HAVE WE OUTPUT ALL 8 BITS
B6AD 26F5    BNE TXBIT ; NO, THEN SEND NEXT BIT

; GET ACK BIT AND TEST IF IT IS LOW. IF NOT, TURN ON ACK FAIL LED
B6AF BDB666    JSR INBIT ; RECEIVE ACK BIT
B6B2 B60101    LDAB EE_IN ;
B6B5 8501    BITA #DIMASK ; TEST IF INPUT IS 0 OR 1
B6B7 2703    BEQ DONETX ; IF ACK IS LOW, GO TO END OF TXBYTE
B6B9 1C0304    BSET PCOFF,X,LEDMASK ; TURN ON ACK FAILED LED
B6BC 39    DONETX RTS ;*************************************************************

;*************************************************************
;*************************************************************
; This subroutine receives 1 byte from SEEPROM
;*************************************************************

B6BD    RXBYTE
B6BD C608    LDAB #8 ; LOAD BIT COUNTER IN ACCB

B6BF 790103    RXBIT ROL RXBUFF ; GET READY TO RECEIVE NEXT BIT
B6C2 B60103    LDAA RXBUFF ; GUESS THAT INPUT BIT IS 0
B6C5 84FE    ANDA #11111110B
B6C7 B70103    STAA RXBUFF ; WRITE ACCA BACK TO RXBUFF

B6CA BDB666    JSR INBIT ; RECEIVES 1 BIT
B6CD B60101    LDAA EE_IN ; IS IN BIT A 1 OR A 0?
B6D0 8501    BITA #DIMASK ;
B6D2 2708    BEQ CONTRX ;

B6D4 B60103    LDAA RXBUFF ; GET RXBUFF INPUT
B6D7 8A01    ORAA #00000001B ; SET INPUT BIT TO 1
B6D9 B70103    STAA RXBUFF ;

B6DC 5A    CONTRX DECB
B6DD C100    CMPB #00H ; HAVE WE OUTPUT ALL 8 BITS
B6DF 26DE    BNE RXBIT ; NO, THEN SEND NEXT BIT
B6E1 39    RTS

;*************************************************************
;*************************************************************
; This command tells the cross-assembler that code starts
; at the location of START

B600    END START
INTRODUCTION

The Microchip Technology Inc. 24LC21 is a 1K-bit (128 x 8) dual mode serial EEPROM that was developed primarily for use in computer monitors. This part was developed with inputs from several computer monitor manufacturers, in accordance with the VESA® (Video Electronics Standards Association) monitor committee. This committee has developed a serial communication protocol called Data Display Channel (DDC™) which was created to eliminate the need to change dip switches when configuring a new system or adding a new monitor or video card. The 24LC21 device is used in the monitor to store and transmit the EDID (extended display ID) table which contains all set-up parameters needed by the video card to operate with a particular monitor. With this system, the user can now plug any compatible monitor into any compatible graphics board and the graphics board will automatically know what type of monitor is being used and configure itself accordingly. This automatic configuration is the cornerstone for Microsott®’s ‘Plug and Play’ capability being built into the new ‘Windows 95™’ release.

DEVICE OPERATION

The 24LC21 can operate in two modes of operation. These two modes of operation are the transmit only mode and bi-directional mode. Upon power-up, the device will always be in the transmit-only mode. Transmit only mode is also referred to as DDC1 mode. The transmit only mode only allows the video card to read the contents of the 24LC21 in a sequential manner, one bit at a time. Writing to the device is not possible in transmit only mode.

The device will automatically transition to the bi-directional mode whenever a falling edge is seen on the SCL pin. Bi-directional mode is also referred to as DDC2 mode, and is implemented as the standard I²C™ protocol. This allows a controller to read and write specific addresses in the device like a standard I²C Serial EEPROM device. Once the device has transitioned to the bi-directional mode, there is no way to return to transmit only mode other than to reset (power-down) the device.

TRANSMIT ONLY MODE (DDC1)

The 24LC21 will always power-up in the transmit only mode. In this mode, the 24LC21 will output one bit of data at the SDA pin for every rising edge on the VCLK pin. The data will be transmitted in 8-bit words, with each word followed by a 9th null bit. This null bit will always be high. A timing diagram for transmit only mode is shown in Figure 1. As long as VCLK is present and no falling edges on SCL are received, the 24LC21 will repeatedly cycle through the entire memory array.

PACKAGE TYPE

Upon power-up, the device will not output valid data until it has been initialized. This initialization procedure (Figure 1) data will not be available until after the first 9 clocks are sent to the device. The exact memory location that the 24LC21 begins to transmit data is unknown at power-up, and the initialization procedure only initializes the device, not the starting address or bit location. In order to for a controller to determine what address is being read, a ‘framing’ or ‘syncing’ procedure must be executed by the video card.
A framing procedure involves looking for the header portion of the EDID table which is a byte of 00H followed by 6 bytes of FFH and another byte of 00H. A framing routine would continue to clock data from the 24LC21 until this unique header has been found. At this point, the current location in the EDID table has been determined and the controller has now synchronized itself with the device. Care must be taken while using the device in the transmit only mode to prevent noise on the SCL pin, as a falling edge seen on this pin will immediately send the part into the bi-directional mode. In a DDC1-only monitor, SCL is not connected to the VGA connector, but must still be terminated to Vcc through a pullup resistor.

**FIGURE 1: TRANSMIT ONLY MODE**

- **SCL**: SCL must remain high for transmit-only mode
- **SDA**: Bit8 (MSB) Bit7 Bit6 Bit5 Bit4 Bit3 Bit2 Bit1 (LSB) Null Bit Bit8 Bit7
- **VCLK**: Clock cycles

**FIGURE 2: DEVICE INITIALIZATION FOR TRANSMIT-ONLY MODE**

- **Vcc**: Power supply
- **SCL**: SDA at high impedance for 9 clock cycles
- **SDA**: Valid data begins on 10th clock
- **VCLK**: Clock cycles

Bit8 Bit7 Bit6 Bit5 Bit4
BI-DIRECTIONAL MODE (DDC2)

Bi-directional mode is essentially the standard I²C protocol and allows the controller to read and write to the device. The 24LC21 supports byte and page writes and byte and sequential reads in the bi-directional mode. This mode will be used primarily before the monitor leaves the factory to load the EDID table into the device, but it also provides a means of updating the table if necessary. It is also used for faster (up to 100 kHz) data transmission, or transmission of only specific requested data in a DDC2 system. (The I²C protocol allows the host to request data from a specific portion of the EDID table rather than waiting for the entire table.) When writing to the device, the VCLK pin must be held high while the write command is being loaded or the write will be aborted and no data will be written. Note that this is the opposite of the 24LC01B, where the WP pin must be held low for the device to be written.

EDID TABLE

The EDID table is the Extended Display ID table, specified by VESA, that will be stored in the 24LC21 and contains information about what type of display it is and the capabilities of the display. The basic EDID table consists of 128 bytes of data. A breakdown of the table is shown below in Table 1. A complete description of the table can be found in the VESA DDC Specification.

TABLE 1: EDID TABLE DESCRIPTION

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Header</td>
</tr>
<tr>
<td>10</td>
<td>Vendor/Product Description</td>
</tr>
<tr>
<td>2</td>
<td>EDID Version/Revision</td>
</tr>
<tr>
<td>15</td>
<td>Basic Display Parameters/Features</td>
</tr>
<tr>
<td>19</td>
<td>Established/Standard Timings</td>
</tr>
<tr>
<td>72</td>
<td>Detailed Timing Descriptions (18 bytes each)</td>
</tr>
<tr>
<td>1</td>
<td>Extension Flag</td>
</tr>
<tr>
<td>1</td>
<td>Checksum</td>
</tr>
</tbody>
</table>
USING THE 24LC21 IN A SYSTEM

In order to use the 24LC21 in a monitor system, it must be programmed with a proper EDID table and then properly connected to the signals coming from the video controller card. The VESA committee has specified that the connections for DDC transmission can be part of the standard 15-pin VGA connector. A table of pinouts for this connector are shown in Table 2. Signals that pertain to the use of the 24LC21 are highlighted.

TABLE 2: VGA CONNECTOR DESCRIPTION

<table>
<thead>
<tr>
<th>Pin</th>
<th>Standard VGA</th>
<th>DDC1 Host</th>
<th>DDC2 Host</th>
<th>DDC1.2 Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red Video</td>
<td>Red Video</td>
<td>Red Video</td>
<td>Red Video</td>
</tr>
<tr>
<td>2</td>
<td>Green Video</td>
<td>Green Video</td>
<td>Green Video</td>
<td>Green Video</td>
</tr>
<tr>
<td>3</td>
<td>Blue Video</td>
<td>Blue Video</td>
<td>Blue Video</td>
<td>Blue Video</td>
</tr>
<tr>
<td>4</td>
<td>Monitor ID Bit2</td>
<td>Monitor ID Bit2</td>
<td>Monitor ID Bit2</td>
<td>Return</td>
</tr>
<tr>
<td>5</td>
<td>Test (Ground)</td>
<td>Return</td>
<td>Return</td>
<td>Return</td>
</tr>
<tr>
<td>6</td>
<td>Red Video Return</td>
<td>Red Video Return</td>
<td>Red Video Return</td>
<td>Red Video Return</td>
</tr>
<tr>
<td>7</td>
<td>Green Video Return</td>
<td>Green Video Return</td>
<td>Green Video Return</td>
<td>Green Video Return</td>
</tr>
<tr>
<td>8</td>
<td>Blue Video Return</td>
<td>Blue Video Return</td>
<td>Blue Video Return</td>
<td>Blue Video Return</td>
</tr>
<tr>
<td>9</td>
<td>No Connection</td>
<td>+5V Supply (optional)</td>
<td>+5V Supply (optional)</td>
<td>+5V Supply (optional)</td>
</tr>
<tr>
<td>10</td>
<td>Sync Return</td>
<td>Sync Return</td>
<td>Sync Return</td>
<td>Sync Return</td>
</tr>
<tr>
<td>11</td>
<td>Monitor ID Bit0</td>
<td>Monitor ID Bit0</td>
<td>Monitor ID Bit0</td>
<td>Optional</td>
</tr>
<tr>
<td>12</td>
<td>Monitor ID Bit1</td>
<td>Data from Display (SDA)</td>
<td>Bi-directional Data (SDA)</td>
<td>Bi-directional Data (SDA)</td>
</tr>
<tr>
<td>13</td>
<td>Horizontal Sync</td>
<td>Horizontal Sync</td>
<td>Horizontal Sync</td>
<td>Horizontal Sync</td>
</tr>
<tr>
<td>14</td>
<td>Vertical Sync</td>
<td>Vertical Sync (VCLK)</td>
<td>Vertical Sync</td>
<td>Vertical Sync (VCLK)</td>
</tr>
<tr>
<td>15</td>
<td>Monitor ID Bit3</td>
<td>Monitor ID Bit3</td>
<td>Data Clock (SCL)</td>
<td>Data Clock (SCL)</td>
</tr>
</tbody>
</table>

Programming of the 24LC21 can be accomplished via Microchip Technology's SEEVAL programming and evaluation system or by any final test system at the customer site which can communicate over the I2C bus.
SYSTEM CONFIGURATION

A typical system configuration is shown below. The DDC specification states that a 47 kΩ pull-up resistor is required on the SDA line at the monitor end. It also states that a 15K pullup resistor is needed on both the SCL and SDA lines at the video controller end.

FIGURE 3: USE OF 24LC21 IN VIDEO SYSTEM

* 47 kΩ resistor is recommended on SCL line at the monitor end although it is not required by the VESA specification.
POTENTIAL PROBLEMS CAUSED BY
NOISE IN A VIDEO SYSTEM

Because the typical application for the 24LC21 is in a
computer monitor where electronic noise is prevalent,
some precautions may need to be made in order for
this device (or any other CMOS device) to work prop-
erly. The diagram below (Figure 4) shows a filter circuit
that can be used to reduce the amount of noise seen
by the device on the SCL and VCLK pins.

FIGURE 4: RECOMMENDED FILTER CIRCUIT FOR MONITOR APPLICATIONS

---

**Diagram Description**

- **Vcc**: Power supply
- **CBP**: Bypass capacitor
- **VCLK**: Clock input
- **24LC21**: Memory chip
- **SCL**: Serial clock output
- **Vsyc**: Sync output
- **RS**: Series resistor
- **RPD**: Pull-down resistor
- **RPu**: Pull-up resistor
- **Cs**: Series capacitor
- **CR1**: Current regulator

**Components**

- **CBP**: 1 µF
- **RS**: 100 - 300 Ω
- **RPD**: 4.7 kΩ
- **Cs**: 100 - 1000 pF
- **RPu**: 47 kΩ

**Description of Components**

- **CBP**: Bypass capacitor
- **RPD**: Can be as a termination resistor on VGA
cable. Also will discharge the series capaci-
tor going to the MCU and horizontal/vertical
processor.
- **Cs and Rs on VCLK**: Acts as low pass filter to clean-up noise on
Vsyc line
- **Cs and Rs on SCL**: Acts as low pass filter to clean-up noise and
dampen power transient spikes that may
cause accidental mode switching from
DDC1 to DDC2
- **RPu on SCL**: Keeps SCL pulled high, although a high
enough value is used that the host will not
power the 24LC21. Lets the 24LC21 reset
when the monitor power is turned off
- **CR1**: Eliminates undershoot on Vsyc
INTRODUCTION

Systems requiring embedded control are becoming more and more sophisticated, with microcontrollers required to control these systems increasing in complexity. Many microcontrollers today are being designed with built-in serial communication capability to be able to easily access other features that are not built in to the microcontroller itself. Devices like EEPROMs, A/Ds, D/A, LCDs etc. are all being built with a serial interface to reduce cost, size, pin count, and board area. There are many different serial interfaces on the market used to interface peripherals (I²C™, Microwire®, SPI, for example). One of the serial interfaces that is gaining in popularity is SPI (Serial Peripheral Interface). It is becoming more popular because of its communication speed, simultaneous full-duplex communication, and ease of programming.

Microchip PIC16C64/74 microcontrollers have a built-in serial port that can be configured as an SPI port. Currently, the Microchip Serial EEPROM product line does not support SPI interface Serial EEPROMs, however it is possible to use the 93 series devices on the SPI port. Any version of Microchip's 93 series devices can be communicated with via the SPI port of a PIC16C64/74. The code for this application note is written for a Microchip 93LC56/66, but talking to other 93 series devices can be accommodated with minor code changes. See the Theory of Operation section of this application note for more details on how this is accomplished. This code was verified by downloading to Microchip's PICMASTER™ in-circuit emulator (run at full speed with a 10 MHz crystal) and tested to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic is included in this application note to describe exactly how the 93LC56/66 was connected to the microcontroller.

THEORY OF OPERATION

To use an SPI port to communicate with Microchip's 93 series Serial EEPROMs, the bytes to be output to the 93XXXX must be aligned such that the LSB of the address is the 8th bit (LSB) of a byte to be output. From there, the bits should fill the byte from right to left consecutively. If more than 8 bits are required, then two bytes will be required to be output. This same method will work for any 93 series device. A 93LC66 was chosen as the device to write this application note code for, so the following example will be for that particular device. The theory explained below will work for any 93 series device.

Since more than 8 bits are required to control a 93LC66, two consecutive bytes are required.

**High Byte (where the start bit, op code bits, and address MSB reside)**

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
\end{array}
\]

The High Byte is configured in the following format: SB is the start bit, OP1 is op code MSB, OPO is op code LSB, and AS is the 9th address bit that is required to address 512 bytes. The CS can be set before the byte is output because the leading 0's output to the 93xxxx prevent a start bit from being recognized by the 93xxxx until the first high bit is sent.

**Low Byte (8 address LSBs)**

\[
\begin{array}{cccccccc}
A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0 \\
\end{array}
\]

The Low Byte contains A7-A0, which are the rest of the address bits required to access 512 bytes.

Data output from master MUST be set up on the falling edge of the clock so that it can be read from the 93XXXX on the next rising edge. Receiving data from the 93XXXX MUST also happen on the falling edge of the clock because the data is output from the 93XXXX on the rising edge of the clock. THIS REQUIRES THE CLOCK PHASE BIT OF THE SPI PORT TO BE OPPOSITE FOR RECEIVING THAN IT IS FOR TRANSMITTING. The clock phase needs to be toggled between 0 for transmitting data and 1 for receiving data. See source code for the exact spot where the clock phase bit needs to be changed.
FIGURE 1: PIC16C74 TO 93LC56/66 SCHEMATIC
FIGURE 2: MOTOROLA 68HC11 TO MICROCHIP 93LC56/66 SCHEMATIC

Note: A pull down resistor is included on the CS pin. It deselects the 93LC56/66, always, except for when CS is driven high by the 68HC11.
APPENDIX A: PIC16C64/74 SOURCE CODE

To use the SPI port to communicate with 3-wire devices, the bytes to be output must be aligned such that the LSB of the address is the 8th bit (LSB) of a byte to be output. From there, the bits should fill the byte from right to left consecutively. This same method will work for any 93xxxx device. A 93LC66 was chosen as the device to write this application note code for, so the following example will be for that particular device. The theory explained below will work for any 93 series device.

Since more than 8 bits are required to control a 93LC66, two consecutive bytes are required.

High byte (where start bit, op code bits and address MSB reside)

| 0 | 0 | 0 | O | SB | OP1 | OP0 | A8 |

The High Byte is configured in the following format:

- SB is the start bit, OPl is op code MSB, OP0 is op code LSB, and A8 is the 9th address bit that is required to address 512 bytes. The CS can be set before the byte is output because the leading 0's output to the 93xxxx prevent a start bit from being recognized by the 93xxxx until the first high bit is sent.

Low byte (8 address LSBs)

| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |

The Low Byte contains A8-A0, which are address bits required to access 512 bytes.

The chip select is set high before sending the first byte out because the 93LC66 will not recognize a start bit until both CS and DI are high on the rising edge of a clock.

This code is written to use the 16C64/74 SPI port to communicate with a Microchip 93LC66. The only other I/O line required besides the SPI port pins is a chip select. The ORG pin will be grounded to set up the part in x8 mode.

PIN DESCRIPTIONS:

- SCK (serial clock) port C, bit 3
- SDO (serial data out) port C, bit 5
- SDI (serial data in) port C, bit 4
- CS (chip select) port C, bit 0

Transmits from the master MUST happen on the falling edge of the clock so they can be read by the 93xxxx on the next rising edge of the clock.

Receiving from the 93xxx MUST also happen on the falling edge of the clock because on the rising edge of the clock, the 93xxxx outputs its bit on its DO pin. This requires the CKP bit in the SSPCON register to be set to 1 for transmitting data and CKP=0 for receiving data.

This code was written by Keith Pazul on 10/5/94

Ram Register Definitions
; received bytes from EEPROM memory locations 10h to 13h
; will be stored in RAM registers 20h to 23h.
rxdata equ 24h
txdata equ 25h
addr equ 26h
loops equ 27h
loops2 equ 28h
hibyte equ 29h
lobyte equ 2Ah
datbyt equ 2Bh

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

;****************************************************
; Other Definitions
;****************************************************
;****************************************************

;****************************************************
; Bit Definitions
;****************************************************
; cs equ 0
sdi equ 4

;****************************************************
include "c:\mpasm\include\p16cxx.inc" ; register map for PIC16CXX devices
org 0x000 ; Reset Vector
goto Start

;***********************************************************
This is the transmit/receive routine. The received bytes
are don't cares until reading back from the array.
;***********************************************************
output movwf SSPBUF place data in buffer so it
can be output
loop1 bcf STATUS, RPO specify bank 1
btfss SSPSTAT, BF has data been received (xmit done)?
goto loop1 not done yet, keep trying
bcf STATUS, RPO specify bank 0
movf SSPBUF, W empty receive buffer, even if we
don't need received data
movwf rxdata put received byte into location
retlw 0 return from subroutine

;***********************************************************
;***********************************************************
EWEN routine
;***********************************************************
EWEN
bcf STATUS, RPO need to set bank 0
bsf PORTC, cs set chip select line high
movlw b'00001001' start bit is bit 3, 00 is
retlw 0 return from subroutine

; remember CS must be low for at least 250 nsec
;***********************************************************
This routine outputs the two bytes required to send the start bit, op code bits, and address bits.

```
WRITE
bcf STATUS, RP0 ; need to set bank 0
bsf PORTC, cs ; set chip select line high
movf hibyte, 0 ; put hibyte in w reg
call output ;
movf FSR, 0 ; put addr pointed to by FSR into w reg
call output ;
movf datbyt, 0 ; get ready to output data in datbyt
call output ;
bcf PORTC, cs ; bring chip select low to begin EEPROMs internal write cycle
incf FSR, 1 ; point to next location to write to
retlw 0
```

This is the module that reads one byte of data.

```
READ
bcf STATUS, RP0 ; need to set bank 0
bsf PORTC, cs ; set chip select line high
bsf SSPCON, CKP ; make sure CKP is 1 to output next instruction and addr
movf hibyte, 0 ; move data from hibyte to w
call output ;
movf lobyte, 0 ; get ready to send next byte which is the 8 lsb address bits
call output ;
```

This is where CKP bit is reset for receiving data.

```
bcf SSPCON, CKP ; change clock polarity to 0
movlw 0x00 ; The byte transmitted here is a don't care
call output ;
bcf PORTC, cs ; bring chip select low to terminate read command
clrf INDF ; clr location pointed to by FSR
movf rxd data, 0 ; move received data to w reg
movwf INDF ; point received data in location pointed to by FSR
incf FSR, 1 ; point to next location to write to
incf lobyte, 1 ; next addr to read from
retlw 0 ;
```

Start
```
bcf STATUS, RP0 ; need to set bank 0
clrf PORTC ; initialize port c
bsf STATUS, RP0 ; need to set bank 1
movlw 0x10 ; all bits are outputs except SDI
movwf TRISC ; for SPI input
clrf PIE1 ; disables all peripheral ints
clrf INTCON ; disables all interrupts
```
bcf STATUS, RP0 ; need to set bank 0
clrfrf SSPCON ; clear SSP control register
movlw 0x31 ; SPI master, clk/16, ckp=1
movwf SSPCON ; SSPEN enabled
call EWEN ; output EWEN for enabling writes

; The next thing we will do is to write 0x5A to locations
; 10h through 13h.

movlw b'00001010' ; start bit is bit 3, 01 is
movwf hibyte ; load into hibyte

movlw 0x10 ; put beginning address in FSR
movwf FSR ; for later use
movlw b'01011010' ; load 0x5A as data to be sent out
movwf datbyt ;

wrnext call WRITE ; call write subroutine

; Ready/Busy poll to decide when write is complete
; and the 93LC66 is available for writing the next
; byte.

nop ; cs must be low for > 250 ns
bsf PORTC, cs ; and then be brought high
rbusy btfsr PORTC, sdi ; test ready/busy status
; if 1, internal write is done
goto rbusy ; part still writing, stay in
; loop
bcf PORTC, cs ; bring cs back low
btfsr FSR, 2 ; have we written all 4 locations?
goto wrnext ; no, then write next byte

; Now, lets read back 10h through 13h (non-sequentially) and
; store it in ram locations 20h through 23h in the 16C74.
; With the Picmaster, I can read those memory locations
; and see that it was read in properly. This is how
; I can quickly verify that the read function is working
; properly.

movlw 0x20 ; where in RAM to begin storing
movwf FSR ; data read back from EEPROM
movlw 0x10 ; addr of where to begin reading
movwf lobyte ; EEPROM from
movlw b'00001100' ; start bit is bit 3, 10 is
movwf hibyte ; op code for read

rdnext call READ ;
btfsr FSR, 2 ; have we 20h thru 23?
goto rdnext ; no, then read next location

; While program is in limbo routine, it is possible
; to halt the processor with Picmaster and look at
; the data contained 16C74 RAM locations 20h through 23h.

limbo nop

; ;
end
APPENDIX B: MOTOROLA 68HC11 SOURCE CODE

;*************************************************************************
; 3-Wire byte write and byte read using SPI port (220 bytes)
; This program (hcllsp66.*) writes 4 bytes of $5A to a Microchip 93LC66
; using the SPI port of the Motorola 68HC11 microcontroller. Ready/busy
; polling is used to determine when the current byte is done writing and
; the next byte is ready to be written to the 93LC66. After the 4 bytes
; are written, they are read back and stored in RAM locations $100-$103.
; The evaluation board can be stopped after receiving the 4 bytes to
; view RAM locations and verify that what was written to the 93LC66 is
; what is read back.
; This code was written by Keith Pazul on 3/7/95.
; This is the way the bytes will be aligned to be sent to the 93LC56/66:
; First byte (where start bit, op codes reside)
; | 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |
; where SB is start bit, OP1 is op code msb, OP0 is op code lsb,
; and A8 is address msb.
; Next byte (address)
; | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
; where A8-A0 are address bits required to address 4K bits of memory.
; This code will work for any 93 series device. The only difference
; is that the number of address bits is adjusted and the start bit
; and op code bits are adjusted to follow directly after the address
; MSB.
; THE 93LC65 or 66 IS ASSUMED TO BE IN x8 MODE. IT requires
; THE USER TO TIE THE ORG PIN TO Vss.
; This program was written using a 68CH11EVBU evaluation board.
; This board has a monitor program in firmware of the 68HC11 which
; allows single stepping, register viewing, modifying, etc. Since this
; is the case, the program code will be loaded into the on-chip EEPROM.
; This EEPROM begins at $8600. The control registers are left to thier
; default location of $1000 and the RAM is left to its default location.
; RAM locations $48-$ff are used by the monitor program and are not
; available for program use. Therefore, the stack pointer is set a $47
; and will be able to use all of the RAM to $00, and the RAM variables
; begin at location $100 and go up from there. I cannot program the
; reset vector since it is ROM space (at $fffe), so the way I run this
; program is to use the monitor program that comes with the evaluation
; board to set the program counter to the starting address of my user
; program ($8600) and begin from there. For users who do have access
; to the reset vector, the label of the beginning program (in my case, it
; is called START) should be loaded at location $FFFE. This program was
; not assembled using a Motorola assembler, but was assembled using
; Universal Cross-Assemblers Cross-32 Meta-Assembler. It has the
; ability to assemble just about any microcontroller code. There are
; certain commands that are unique to the cross-assembler. These
; commands will be commented differently than other comments to
; be recognizable. They will look like this:
;*************************************************************************
I do not know the exact assembler commands required to accomplish assembly using a Motorola assembler.

The crystal that comes with the evaluation board is 8 MHz.

The SPI port is on port D. The bits are defined as follows:

MISO - PORT D, PIN 2 (pin 22 on package)
MOSI - PORT D, PIN 3 (pin 23 on package)
SCK - PORT D, PIN 4 (pin 24 on package)
SS - PORT D, PIN 5 (pin 25 on package)
CS - PORT C, PIN 6

Note that only the CS pin resides on port C.

CPU "C:\WINC32\68HC11.TBL" ; LOAD TABLE
HOF "MOT9"; Hex output is Motorola S-records

68HC11 control register locations

REG BS EQU 1000H BEGINNING OF REGISTERS
RAMBS EQU 1000H BEGINNING OF RAM VARIABLES
DDRC EQU REGBS+07H DATA DIRECTION REG FOR PORT C
PORTC EQU REGBS+03H PORT C DATA REGISTER
PCOFF EQU 03H OFFSET FROM CONTROL REG BEG.
PORTD EQU 1008H PORT D DATA REGISTER
DDRD EQU 1009H PORT D DATA DIRECTION REGISTER
SPCR EQU 1028H SPI CONTROL REGISTER
SPSR EQU 1029H SPI STATUS REGISTER
SPDR EQU 102AH SPE DATA REGISTER

User defined constants

CSMASK EQU 01000000B ; BIT MASK FOR CHIP SELECT
SDIMASK EQU 00000100B ; BIT MASK FOR MISO PIN

LDS #0047H ; STACK POINTER BEGINS AT $47
ORG 100H ; RAM variables begin at 100h

DFS is a Universal Cross-Assembler directive that stands for define storage. 1 is for byte, 2 is for word, 4 is for long word. These are the user defined RAM variables.

RXARAY DFS 1 * (4) ; 100H-103H
RXDATA DFS 1 ; 104H
DATBYT DFS 1 ; 105H
HIBYTE DFS 1 ; 106H
LOBYTE DFS 1 ; 107H
RBTEST DFS 1 ; 108H
ADDRDF DFS 1 ; 109H

Program code cannot be placed in ROM for eval board because
; eval board firmware is loaded in ROM. Program code will be
; loaded in EEPROM (which is 512 bytes and begins at B600h).
;
ORG 0B600H
;******************************************************************************
; This is the main portion of the code. It is where the reset
; vector should set program counter to.
;******************************************************************************

START
LDX #REGBS LOADS BEG OF REGISTERS INTO X
BCLR PCOFF,X,CSMASK ; MASK SURE CS IS CLEARED
LDAA #10110000B ; SETS UP BITS 0-3 AND 6 AS INPUTS, BITS 4, 5, AND 7 AS OUTPUTS
STAA DDRC
CLR PORTC ; CLEAR ALL PORT C BITS
LDAA #11111011B ; ALL BITS ARE OUTPUTS EXCEPT MISO
STAA DDRD
LDAA #01010010B ; SPIE=0, SPE=1, DWOM=0, MSTR=1,
STAA SPCR ; CPOL=0, CPHA=0, CLK/16
LDAA #00000101B ; STORE READY/BUSY MASK IN
STAA RBTEST ; LOCATION RBTEST
LDAA #10H
STAA LOBYTE ; LOAD 8 LSB'S OF ADDRESS
CLR ADDROFF ; SET ADDRESS OFFSET TO 0 FOR
; READ COMMANDS
JSR EWEN ; SEND EWEN COMMAND

;******************************************************************************
; Now lets write SAh out to address 10h
;******************************************************************************

LDAA #00001010B ; START BIT IS BIT 3, 01 IS
STAA HIBYTE ; OP CODE FOR WRITE, LSB IS
; ADDRESS
LDAA #01011010B ; LOAD 0x5A IN DATA LOCATION
STAA DATBYT
WRNEXT JSR WRITE ; CALL WRITE ROUTINE

;******************************************************************************
; ready/busy poll input pin to see when internal write
; cycle is complete.
;******************************************************************************

BSET PCOFF,X,CSMASK ; RAISE CS FOR READY/BUSY POLLING
RBUSY
LDAA PORTD ; INPUT PORT D
ANDA RBTEST ; MASK OFF ALL BITS EXCEPT MISO
BEQ RBUSY ; IF MISO IS LO, PART STILL BUSY,
; ELSE WRITE IS COMPLETE
BCLR PCOFF,X,CSMASK ; POLLING COMPLETE, END CHIP SELECT
LDAA  LOBYTE  ; GET ADDRESS OFFSET
BITA  #0000100B  ; ARE ALL BYTES WRITTEN YET?
BEQ  WRNEXT  ; ALL BYTES HAVE NOT BEEN WRITTEN,
             ; WRITE NEXT BYTE

;***************************************************************************************
; Now lets read back the location of rxdata to see if we
; read back what we wrote to that location.
;***************************************************************************************

LDAA  #0001100B  ; STRT BIT IS BIT 3, 10 IS
STAA  HIBYTE  ; OP CODE FOR READ, LSB IS
             ; ADDRESS
LDAA  #10H  ; RESET BEGINNING ADDRESS
STAA  LOBYTE  ; FOR READ OPERATIONS
LDY  #$RAMBS  ; RX BUFF IS AT BEG OF RAM
RDNEXT  JSR  READ  ; CALL READ SUBROUTINE

LDAA  LOBYTE  ; LOAD ACC WITH MASK THAT
             ; CHECKS TO SEE IF 4 BYTES
BITA  #00000100B  ; HAVE BEEN WRITTEN
BEQ  RDNEXT  ; CHECK TO SEE IF 4 BYTES HAVE
             ; BEEN WRITTEN. IF NOT, READ
             ; NEXT BYTE.
LDAA  #$1011000B  ; DISABLES CS TO PROHIBIT
STAA  DDRC  ; POSSIBILITY OF INADVERTANT
             ; WRITES

LIMBO  NOP
JMP  LIMBO

;***************************************************************************************
; This subroutine outputs the data stored in TXBUFF
;***************************************************************************************

OUTPUT

STAA  SPDR  ; MOVE DATA FROM ACC A TO SPDR
LOOP1
LDAB  SPSR  ; SEE IF XFER COMPLETE FLAG SET
BPL  LOOP1  ; IF NOT, LOOP UNTIL SET
LDA  SPDR  ; MOVES RECEIVED BYTE FROM DATA REG
STAA  RXDATA  ; TO RECEIVE DATA LOCATION

RTS

;***************************************************************************************
;
; EWEN routine
;
;***************************************************************************************

EWEN

LDAA  #$11110000B  ; SET CS FROM IN TO OUt. ALL
STAA  DDRC  ; OTHER BITS ARE THE SAME
BSET  PCOFF,X,CSMASK  ; DROP CS TO BEGIN COMMAND
LDAA  #00000100B  ; STRT BIT IS BIT 3, 00 IS
             ; OP CODE FOR EWEN
JSR  OUTPUT  ;
AN613

LDAA #10000000B ; 1 IS REQ FOR EWEN. REST
JSR OUTPUT
BCLR PCOFF,X,CSMASK ; SET CS TO COMPLETE EWEN

RTS

;***********************************************************
;***********************************************************
; WRITE routine
;***********************************************************
WRITE
BSET PCOFF,X,CSMASK ; BEGIN WRITE COMMAND
LDAA HIBYTE ; GET DATA FROM HIBYTE TO OUTPUT
JSR OUTPUT
LDAA LOBYTE ; 8 LSBs OF ADDRESS POINTED TO
JSR OUTPUT ; BY ADDROFF OFFSET TO Y REG
LDAA DATBYT ; OUTPUT DATA BYTE
JSR OUTPUT
BCLR PCOFF,X,CSMASK ; DROP CS TO BEGIN COMMAND
INC LOBYTE ; INC ADDRESS FOR NEXT LOCATION
RTS

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

READ
LDAA SPCR ; READ IN SPI CONTROL REG
ANDA #11111011B ; SET CPHA TO 0 FOR CONTROL
STA SPCR ; PORTION OF READ SEQUENCE
BSET PCOFF,X,CSMASK ; DROP CS TO BEGIN COMMAND
LDAA HIBYTE ; OUTPUT HI BYTE FOR READ CMD
JSR OUTPUT
LDAA LOBYTE ; 8 LSBs OF ADDRESS
JSR OUTPUT

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

; To read a byte, a byte must be transmitted. Here we xmit a byte
; of don't care bits to receive a byte. WE MUST CHANGE THE CPHA
; FROM 0 TO 1 TO CHANGE BEFORE DATA IS RECEIVED. IF NOT, THEN THE
; MICRO WILL TRY TO RECEIVE AT THE SAME TIME THE 93XX66 WILL BE
; TRANSMITTING. THE SYMPTOM OF THIS NOT SET UP PROPERLY IS THAT
; THE DATA READ WILL BE SHIFTED RIGHT ONE BIT WITH MSB BEING 0.
;***********************************************************
;***********************************************************
;***********************************************************
LDAA SPCR ; READ IN SPI CONTROL REG
ORAA #0000100B ; SET CPHA TO 1
STAA SPCR

JSR OUTPUT ; XMIT A BYTE OF DON'T CARE BITS
; TO RECEIVE A BYTE

BCLR PCOFF,X,CSMASK ; CLR CS TO END COMMAND

LDAA RXDATA ; GET DATA RECEIVED IN RXDATA
STAA 0,Y ; AND STORE IN LOCATION POINTED
; TO BY Y INDEX REG + 0 OFFSET

INY
INC LOBYTE ; INC ADDR OF NEXT BYTE TO READ
; FROM

RTS

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

; This command tells the cross-assembler that code starts
; at the location of START

END START

;************************************************
INTERFACING MICROCHIP SERIAL EEPROMS TO THE INTEL 8051 FAMILY OF MICROCONTROLLERS

Many designers today are implementing embedded systems that require low cost non-volatile memory. Microchip has addressed this need with a line of serial EEPROMs, in a variety of memory configurations, using the industry-standard 2- or 3-wire communication protocols. The theory and application of these protocols are addressed in detail in Microchip's application note AN536.

Microchip recognizes that its broad customer base uses a variety of micro-controllers; many firmware related questions have been asked concerning interfacing the 8051 family and its derivatives.

The purpose of this app note is to provide assembly language examples of 8051 code for the various serial EEPROMs available from Microchip. These routines are intended to provide the basic operating kernels for storing data to or retrieving data from a serial EEPROM.

All of the routines in this app note are available, as source code, for downloading from Microchip's BBS. Information on the BBS is available elsewhere in the Embedded Control Handbook. The file to download is AN61437.zip.

This app note covers all of Microchip's 2-wire serial devices. Note that some devices have features not supported in others, and therefore, some sections of the code presented here may not be applicable to a particular part. We have attempted to label those special sections to minimize confusion.

The code includes a simple loop-type shell to enable it to be executed (with an emulator) without the user having to write any other routines. The various address and data pointers must be set to the desired values by hand, before each execution cycle.

TIMING DATA

Clock and data timing is accomplished by software. There are two sets of timing specifications: 100 kHz and 400 kHz. Assuming a 12 MHz 8051 clock, extra NOP's have been added to slow timing down to 100 kHz. See Note 1 in the listing. If a 16 MHz clock is used, additional NOP's are required for 100 kHz operation. See Note 2 in the listing. For 400 kHz operation, the NOP's labelled Note 1 or Note 2 are not needed. If not needed, NOP's may be left out.

Below is the connection diagram used for this app note. Do not forget the pull-up resistor!

WIRING DIAGRAM

* This pin is NC in some parts
Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: SOURCE CODE

MCS-51 MACRO ASSEMBLER 2WIRE

DOS 6.0 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN 2WIRE.OBJ
ASSEMBLER INVOKED BY: C:\ICE5100\ASM51\ASM51.EXE 2WIRE.TXT

LOC OBJ LINE SOURCE

1 $ PAGELENGTH(46) PAGewidth(132) DEBUG NOPAGING XREF
2 ;REGISTER ASSIGNMENTS:
3 ;
4 ;R1 DATA OR DATA POINTER
5 ;R2 LOOP COUNTER REGISTER
6 ;R3 ADDRESS, HI BYTE
7 ;R4 ADDRESS, LOW BYTE
8 ;R5
9 ;R6 BYTE COUNT FOR PAGE OPERATIONS
10
11 ;PIN ASSIGNMENTS:
12 ;Port 1 bit 0 is data
13 ;Port 1 bit 1 is clock
14 ;
15 ; These routines assume chip address = 0
16 ;
17 ; The oscillator frequency assumed for this app note is 12 MHz.
18 ;
19 ; These routines use software timing loops. They may have to be
20 ; adjusted if a different oscillator frequency is used.
21 ;
22 ; NOTE 1 These NOP'S added for timing delays only on 'C' parts, OR 'LC' parts
23 ; where Vcc is less than 4.5 V. and the oscillator frequency is 12 MHz.
24 ; This allows a bit rate of 100kHz.
25 ; NOTE 2 Use these NOP's with a 16 MHz oscillator and 100KHz bit rate.
26 ; For 400KHz bit rate, the NOP's in Note 1 and Note 2 are not required.
27 ;
28 ; The EEPROM will be busy after a write cycle is initiated (by a stop condition)
29 ; for between 1mS to 10 mS per page (or per byte if a byte write). This app note
30 ; assumes the user will program appropriate wait times after a write, or check
31 ; for Busy status. A subroutine is provided to check the Busy Status.
32 ;
; RAM DEFINITIONS

ORG 30H

BYTSTR: DS 20H ; STORAGE FOR READ DATA

; CONSTANTS -- REDEFINE AS NECESSARY

ORG 30A0

WTCMD EQU 10100000B ; WRITE DATA COMMAND Note 3

RDMD EQU 10100001B ; READ DATA COMMAND Note 3

RDEND EQU 01000001B ; READ HIGH-ENDURANCE BLOCK NUMBER COMMAND

ADDRH EQU 0

ADDRL EQU 0

DTA EQU 55H

BYTECNT EQU 8

; Note3 Some chip or byte address bits are embedded in the control byte. Refer to
; the data sheet for exact configuration, which varies from part to part.

; This section contains test loop routines. They form a simple operating shell to
; allow the 2-wire interface code to be tested in a stand-alone mode. Using an
; emulator, change "NONE" to one of the four listed
; routines to test that function. The address and data constants can also be set as
; desired.

; If using a 24XX32 or 24XX65, change the called routines by adding 'L' to the end
; of the name. This is required because these parts use TWO address bytes. The 'L'
; routines send out the extra address byte.

ORG 0

0000 ORG 0

0000 020003 JMP START

0003 7590FF START: MOV P1,#OFFH ; INIT PORT 1

0006 12000B CALL NONE ; TEST LOOP INSERT PROPER ADDRESS HERE

0009 80F8 JMP START

000B 22 NONE: RET

; * WRITE ONE BYTE TO EEPROM

; * The Address Pointer is the address in the EEPROM. The byte to be sent to the
; * EEPROM is stored in the constant 'DTA'

000C 7B00 TESTWR: MOV R3,#ADDRH ; LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY

000E 7C00 MOV R4,#ADDRL ; LOAD ADDRESS POINTER FOR ALL DEVICES

0010 7955 MOV R1,#DTA ; LOAD DATA BYTE

0012 120037 CALL BYTEW ; CALL BYTE WRITE ROUTINE

0015 22 RET
WRITE A BLOCK OF DATA TO EEPROM

The address pointer is the address in EEPROM where data will start. The byte pointer is the starting address of RAM containing the block of data to be sent. The byte count indicates how many bytes to send to the EEPROM.

The number of bytes that can be sent before a STOP command is issued is dependant on EEPROM type. Refer to the data book for specific values.

```
BLKWR: MOV R3,#ADDRH ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65
       MOV R4,#ADDRL ;LOAD ADDRESS POINTER FOR ALL DEVICES
       MOV Rl,#BYTSTR ;LOAD BYTE POINTER
       MOV R6,#BYTCNT ;LOAD BYTE COUNT
       CALL PAGEW ;CALL PAGE WRITE ROUTINE
       RET
```

READ ONE BYTE FROM EEPROM

The address pointer is the address of the desired byte in EEPROM. The byte will be returned in R1.

```
TESTRD: MOV R3,#ADDRH
        MOV R4,#ADDRL
        CALL BYTERD ;CALL BYTE READ ROUTINE.
        MOV Rl,A ;SAVE THE BYTE
        RET
```

READ A BLOCK FROM EEPROM

The address pointer is the starting address of the desired data block in EEPROM. The data pointer is the starting address in RAM where data will be stored. The byte count indicates how many bytes should be read. The entire EEPROM may be read with one READ command this way.

```
BLKRD: MOV R3,#ADDRH
        MOV R4,#ADDRL
        MOV R1,#BYTSTR
        MOV R6,#BYTCNT
        CALL BLKRD ;CALL BLOCK READ ROUTINE
        RET
```

END OF TEST LOOP

This routine writes a byte of data to EEPROM. The EEPROM address is assumed to be in R4. See NOTE 3.
; The DATA to be written is assumed to be in R1

BYT EW: MOV A, #WTCMD ;LOAD WRITE COMMAND
CALL OUTS ;SEND IT
MOV A, R4 ;GET BYTE ADDRESS
CALL OUT ;SEND IT
MOV A, R1 ;GET DATA
CALL OUT ;SEND IT
CALL STOP ;SEND STOP CONDITION
RET

; THIS ROUTINE WRITES A PAGE OF DATA TO EEPROM
; The EEPROM start address is assumed to be in R4. See NOTE 3.
; The DATA pointer is in R1
; The number of bytes that can be transferred depends on the EEPROM used.

PAGEW: MOV A, #WTCMD ;LOAD WRITE COMMAND
CALL OUTS ;SEND IT
MOV A, R4 ;GET LOW BYTE ADDRESS
CALL OUT ;SEND IT
BTLP: MOV A, R1 ;GET DATA
CALL OUT ;SEND IT
INC R1 ;INCREMENT DATA POINTER
DJNZ R6, BTLP ;LOOP TILL DONE
CALL STOP ;SEND STOP CONDITION
RET

; THIS ROUTINE READS A BLOCK OF DATA FROM EEPROM AT A SPECIFIED ADDRESS
; EEPROM address in R4. See NOTE 3.
; Stores data at RAM location pointed to by R1
; Byte count specified in R6. May be 1 to 256 bytes

BLKRD: MOV A, #WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
CALL OUTS ;SEND IT
MOV A, R4 ;GET LOW BYTE ADDRESS
CALL OUT ;SEND IT
MOV A, #RDcmd ;LOAD READ COMMAND
CALL OUTS ;SEND IT
CALL IN ;READ DATA
INC R1 ;INCREMENT DATA POINTER
DJNZ R6, AKLP ;DECREMENT LOOP COUNTER
CALL STOP ;IF DONE, ISSUE STOP CONDITION
RET
BYTERD: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
CALL OUTS ;SEND IT
MOV A,R4 ;GET LOW BYTE ADDRESS
CALL OUT ;SEND IT
CALL CREAD ;GET DATA BYTE
RET

CREAD: MOV A,#RDCMD ;LOAD READ COMMAND
CALL OUTS ;SEND IT
CALL IN ;READ DATA
MOV R1,A ;STORE DATA
CALL STOP ;SEND STOP CONDITION
RET

BYTERD: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
CALL OUTS ;SEND IT
MOV A,R4 ;GET LOW BYTE ADDRESS
CALL OUT ;SEND IT
CALL CREAD ;GET DATA BYTE
RET

This routine reads a byte of data from the EEPROM.

NOTE 1: Returns the data byte in R1.

NOTE 2: The EEPROM address is in R4. See NOTE 3.

NOTE 3: This routine reads a byte of data from the EEPROM.

This routine reads a block of data from EEPROM at a specified address.

This routine is for the 24LC32 or 24LC64.

EEPROM address in R3:R4.

Stores data at RAM location pointed to by R1.

Byte count specified in R6. May be 1 to 256 bytes.
@ 009C 74A0 221 BLKRD: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
009E 120120 222 CALL OUTS ;SEND IT
00A1 EB 223 MOV A,R3 ;GET HI BYTE ADDRESS
00A2 120158 224 CALL OUT ;SEND IT
00A5 EC 225 MOV A,R4 ;GET LOW BYTE ADDRESS
00A6 120158 226 CALL OUT ;SEND IT
00A9 74A1 227 MOV A,#RDCMD ;LOAD READ COMMAND
00AB 120120 228 CALL OUTS ;SEND IT
00AE 808A 229 JMP BRDLP ;CONTINUE WITH DATA READ
00A9 74A0 231 BYTEWL: MOV A,#WTCMD ;LOAD WRITE COMMAND
00B2 120120 232 CALL OUTS ;SEND IT
00B5 EB 233 MOV A,R3 ;GET HI BYTE ADDRESS
00B6 120158 234 CALL OUT ;SEND IT
00B9 EC 235 MOV A,R4 ;GET LOW BYTE ADDRESS
00BA 120158 236 CALL OUT ;SEND IT
00BD E9 237 MOV A,R1 ;GET DATA
00BE 120158 238 CALL OUT ;SEND IT
00C1 120197 239 CALL STOP ;SEND STOP CONDITION
00C4 22 240 RET
00C5 74A0 241 ;*********************************************************************
00C7 120120 242 ;This routine writes a byte of data to EEPROM
00CA EB 243 ;This routine is for the 24LC32 or 24LC64
00CB 120158 244 ;The EEPROM address is assumed to be in R3:R4
00CE EC 245 ;The DATA to be written is assumed to be in R1
00CF 120158 246 ;*********************************************************************
00D2 E7 247 PAGEWL: MOV A,#WTCMD ;LOAD WRITE COMMAND
00D3 120120 248 ;THIS ROUTINE WRITES A PAGE OF DATA TO EEPROM
00DC 22 249 ;This routine is for the 24LC32 or 24LC64
00D6 09 250 ;The EEPROM start address is assumed to be in R3:R4
00D7 DBF9 251 ;The DATA pointer is in R1
00D9 120197 252 ;The BYTE count is in R6
00DC 22 253 ;The number of bytes that can be transferred depends on the EEPROM in use.
00E8 EB 254 ;*********************************************************************
00F1 0000 255
268 ;********************************************************************
269 ; This routine reads a byte of data from the EEPROM
270 ; This routine is for the 24LC32 or 24LC64
271 ; The EEPROM address is in R3:R4
272 ; Returns the data byte in R1
273 ;********************************************************************
00DD 7A40 274 BYTERD: MOV A, #WTCMD ; Load write command to send address
00DF 120120 275 CALL OUTS ; Send it
00E2 EB 276 MOV A, R3 ; Get hi byte address
00E3 120158 277 CALL OUT ; Send it
00E6 EC 278 MOV A, R4 ; Get low byte address
00E7 120158 279 CALL OUT ; Send it
00EA 118F 280 CALL CREAD ; Get data byte
00EC 22 281 RET

;********************************************************************
00ED 7A40 282 ; Subroutines
00EF 7A08 283 ;********************************************************************
00F1 C290 284 This routine tests for WRITE DONE condition
285 ; by testing for an ACK.
286 ; This routine can be run as soon as a STOP condition
287 ; has been generated after the last data byte has been
288 ; sent to the EEPROM. No ACK will be returned until
289 ; the EEPROM is done with the write operation.
290 ;********************************************************************
00F3 00 291 ACRST: MOV A, #WTCMD ; Load write command to send address
00F4 00 292 MOV R2, #8 ; Loop count -- equal to bit count
00F5 00 293 CLR P1.0 ; Start condition -- data = 0
00F6 00 294 NOP ; Note 1
00F7 00 295 NOP ; Note 2
00F8 C291 296 CLR P1.1 ; Clock = 0
00FA 33 297 RLC A ; Shift bit
00FB 5005 298 JNC AKTLS
00FD D290 299 SETB P1.0 ; Data = 1
00FF 020104 300 JMP AKTL1 ; Continue
0102 C290 301 AKTLS: CLR P1.0 ; Data = 0
0104 D291 302 AKTL1: SETB P1.1 ; Clock hi
0106 00 303 NOP ; Note 1
0107 00 304 NOP ; Note 2
0108 00 311 NOP
0109 00 312 NOP
010A 00 313 NOP
010B 00 314 NOP

© 1985 Microchip Technology Inc.
@ OlOB C291 315 CLR P1.1 ;CLOCK LOW
010D DAEB 316 DJNZ R2,ARCLP ;DECREMENT COUNTER
010F D290 317 SETB P1.0 ;TURN PIN INTO INPUT
0111 00 318 NOP ;NOTE 1
0112 00 319 NOP ;NOTE 2
0113 D291 320 SETB P1.1 ;CLOCK ACK
0115 00 321 NOP ;NOTE 1
0116 00 322 NOP
0117 00 323 NOP ;NOTE 2
0118 00 324 NOP
0119 00 325 NOP
011A 309002 326 JNB P1.0,EXIT ;EXIT IF NO ACK (WRITE NOT DONE)
011D C291 327 CLR P1.1 ;SET DONE FLAG
011F 22 328 EXIT: RET

*************************************************************************
THIS ROUTINE SENDS OUT CONTENTS OF THE ACCUMULATOR
; to the EEPROM and includes START condition. Refer to the data sheets
; for discussion of START and STOP conditions.
*************************************************************************

0120 7A08 339 OUTS: MOV R2,#8 ;LOOP COUNT -- EQUAL TO BIT COUNT
0122 D290 340 SETB P1.0 ;INSURE DATA IS HI
0124 D291 341 SETB P1.1 ;INSURE CLOCK IS HI
0126 00 342 NOP ;NOTE 1
0127 00 343 NOP
0128 00 344 NOP ;NOTE 2
0129 00 345 NOP
012A 00 346 NOP
012B C290 347 CLR P1.0 ;START CONDITION -- DATA = 0
012D 00 348 NOP ;NOTE 1
012E 00 349 NOP
012F 00 350 NOP
0130 00 351 NOP ;NOTE 2
0131 00 352 NOP
0132 C291 353 CLR P1.1 ;CLOCK = 0
0134 33 354 OTSLP: RLC A ;SHIFT BIT
0135 5005 355 JNC BITLS
0137 D290 356 SETB P1.0 ;DATA = 1
0139 02013E 357 JMP OTSL1 ;CONTINUE
013C C290 358 BITLS: CLR P1.0 ;DATA = 0
013E D291 359 OTSL1: SETB P1.1 ;CLOCK HI
0140 00 360 NOP ;NOTE 1
0141 00 361 NOP
;***************************************************************************
; THIS ROUTINE SENDS OUT CONTENTS OF ACCUMULATOR TO EEPROM
; without sending a START condition.
;***************************************************************************

0142 00 362 NOP ;NOTE 2
0143 00 363 NOP
0144 00 364 NOP
0145 C291 365 CLR P1.1 ;CLOCK LOW
0147 DAEB 366 DJNZ R2,0TSLP ;DECREMENT COUNTER
0149 D290 367 SETB P1.0 ;TURN PIN INTO INPUT
014B 00 368 NOP ;NOTE 1
014C 00 369 NOP ;NOTE 2
014D 00 370 NOP
014E D291 371 SETB P1.1 ;CLOCK ACK
0150 00 372 NOP ;NOTE 1
0151 00 373 NOP
0152 00 374 NOP
0153 00 375 NOP ;NOTE 2
0154 00 376 NOP
0155 C291 377 CLR P1.1
0157 22 378 RET

379

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

0158 7A08 385 OUT: MOV R2,#8 ;LOOP COUNT -- EQUAL TO BIT COUNT
015A 33 386 OTLP: RLC A ;SHIFT BIT
015B 5005 387 JNC BITL
015D D290 388 SETB P1.0 ;DATA = 1
015F 020164 389 JMP OTL1 ;CONTINUE
0162 C290 390 BITL: CLR P1.0 ;DATA = 0
0164 D291 391 OTL1: SETB P1.1 ;CLOCK HI
0166 00 392 NOP ;NOTE 1
0167 00 393 NOP
0168 00 394 NOP
0169 00 395 NOP ;NOTE 2
016A 00 396 NOP
016B C291 397 CLR P1.1 ;CLOCK LOW
016D DAEB 398 DJNZ R2,0TSLP ;DECREMENT COUNTER
016F D290 399 SETB P1.0 ;TURN PIN INTO INPUT
0171 00 400 NOP ;NOTE 1
0172 00 401 NOP ;NOTE 2
0173 00 402 NOP
0174 D291 403 SETB P1.1 ;CLOCK ACK
0176 00 404 NOP ;NOTE 1
0177 00 405 NOP
0178 00 406 NOP
0179 00 407 NOP ;NOTE 2
017A 00 408 NOP
THIS ROUTINE READS IN A BYTE FROM THE EEP-ROM
and stores it in the accumulator

IN:  MOV  R2,#8 ;LOOP COUNT
SETB  P1.0 ;SET DATA BIT HIGH FOR INPUT
INLP:  CLR  P1.1 ;CLOCK LOW
NOP  ;NOTE 1
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP  ;NOTE 2
SETB  P1.1 ;CLOCK HIGH
CLR  C ;CLEAR CARRY
JNB  P1.0,INL1 ;JUMP IF DATA =0
CPL  C ;SET CARRY IF DATA =1
RLC  A ;ROTATE DATA INTO ACCUMULATOR
DJNZ  R2,INLP ;DECREMENT COUNTER
CLR  P1.1 ;CLOCK LOW
STOP:  CLR  P1.0 ;STOP CONDITION SET DATA LOW
NOP  ;NOTE 1
NOP
NOP
NOP
NOP  ;NOTE 2
NOP
NOP
NOP
NOP
NOP
SETB  P1.1 ;SET CLOCK HI
NOP  ;NOTE 1
NOP
NOP
NOP  ;NOTE 2
NOP
NOP
SETB  P1.0 ;SET DATA HI
RET

These routines contain special commands only for the 24LC65 SMART SERIAL EEP-ROM
* SET SECURE BLOCK
ASSUMES START BLOCK 0 & BLOCK LENGTH OF 1
The numbers are implicit in the commands. Refer to the data sheet for details.

; SETSEC: MOV R3,#80H ;LOAD COMMAND AND STARTING BLOCK NUMBER
01A8 7B80

01AA 7C00

01AC 7981

01AE 1137

01B0 22

01B1 74A0

01B3 3120

01B5 7480

01B7 3158

01B9 7400

01BB 3158

01BD 74C0

01BF 3158

01C1 317E

01C3 F9

01C4 00

01C5 00

01C6 00

01C7 00

01C8 00

01C9 C290

01CB D291

01CD 00

01CE 00

01CF 00

01D0 00

01D1 00

01D2 C291

01D4 317E

01D6 FA

01D7 3197

01D9 22

01B2 7400

01B4 7480

01B6 3158

01B8 7400

01BB 3158

01BD 74C0

01BF 3158

01C1 317E

01C3 F9

01C4 00

01C5 00

01C6 00

01C7 00

01C8 00

01C9 C290

01CB D291

01CD 00

01CE 00

01CF 00

01D0 00

01D1 00

01D2 C291

01D4 317E

01D6 FA

01D7 3197

01D9 22

01D2 7400

01D4 7480

01D6 3158

01D8 7400

01DA 3158

01DB 74C0

01DF 3158

01E1 317E

01E3 F9

01E4 00

01E5 00

01E6 00

01E7 00

01E8 00

01E9 C290

01EA D291

01EB 00

01EC 00

01ED 00

01EE 00

01EF 00

01F0 00

01F1 00

01F2 C291

01F4 317E

01F6 FA

01F7 3197

01F9 22

01F2 7400

01F4 7480

01F6 3158

01F8 7400

01FA 3158

01FB 74C0

01FD 3158

01FE 317E

01FF F9

500  SET HIGH-ENDURANCE BLOCK NUMBER
501  ; ASSUMES BLOCK 0
502  ;
01DA 7880 503  SETHI: MOV R3,#80H ;LOAD command and block number
01DC 7C00 504 MOV R4,#0
01DE 7900 505 MOV R1,#0 ;Set data =0
01E0 1137 506 CALL BYTEW ;Execute
01E2 22  507 RET  
01E3 7880 508 ;*  
01E5 7C00 509 ;Read high-endurance block number  
01E7 1201EB 510 ;Returns block number in R1 (upper nibble will be 1's)
01EA 22  511 ;*  
01EB 7490 512  READHI: MOV R3,#80H ;Load command  
01EF 3120 513  CALL HIEND ;Execute  
01F0 3158 514  MOV A,#WTCMD ;Load write command to send address  
01FC 3197 515  CALL STOP ;Send stop condition  
01FE 22  516 RET  
01ED 3120 517  CALL OUTS ;Send it  
01FF 22  518  CALL IN ;Read data  
01FB F9  519  MOV R1,A ;Store data  
01FC 3197 520  CALL STOP ;Send stop condition  
01FE 22  521 RET  
01FF 22  522  CALL OUTS ;Send it  
01FB F9  523  MOV R1,A ;Store data  
01FC 3197 524  CALL STOP ;Send stop condition  
01FE 22  525 RET  
01FF 22  526  CALL IN ;Read data  
01FC 3197 527  CALL STOP ;Send stop condition  
01FF 22  528 RET  
01FF 22  529 RET  
01FF 22  530 ;End of 24LC65 Routines  
01FF 22  531 ;***************************************************************  
01FF 22  532 ASSEMBLY COMPLETE, NO ERRORS FOUND  
01FF 22  533 END
SECTION 5
QUALITY & RELIABILITY APPLICATION NOTES

AN595  Improving the Susceptibility of an Application to ESD...........................................5-1
AN598  Plastic Packaging and the Effects of Surface Mount Soldering Techniques ...............5-19
AN603  Continuous Improvement............................................................................................5-27
Improving the Susceptibility of an Application to ESD

Author: David Wilkie
Reliability Engineering

INDUCED LATCH-UP

All semiconductor devices are sensitive to electrostatic discharge (ESD) damage to varying degrees. This is true whether they are soldered to a PC board in an application, or whether they are unattached in the shipping or application assembly process. Good handling techniques such as groundstraps, static free work stations and ionizers can reduce the risk of static build up during assembly. Often more attention is paid to reducing ESD during assembly than is paid to reducing ESD risk during the lifetime of the application.

When a device is installed in an application, it is still susceptible to damage due to ESD. This can take on a different form when the application is powered up and running. If power is supplied to a CMOS device such as a memory product or a microcontroller when an ESD event occurs, the device can be triggered into a "latch-up" condition. This is a high current mode where internal circuitry can be disturbed into making a short circuit (or a circuit with very low resistance) between power (Vcc) and ground (Vss) on a device. This condition is self-sustaining; it does not require subsequent ESD events to continue the latch-up condition.

This short circuit will tend to reduce the voltage level on the application (particularly if the application is battery powered) and will do a great deal of damage to the device which has latched-up. The only way to halt this condition is to remove power from the device.

Microchip uses careful design practices to reduce the susceptibility of all products (microcontrollers or memories) to ESD events. However, the protection level varies from pin to pin, reflecting the different functions of each pin. Certain types of pins (notably supply pins) are much more susceptible to latch-up caused by ESD pulses than other pins. This is due to the different design and layout considerations that reduce the effectiveness of ESD protection.

There is a great deal that the system designer can do to improve (by up to an order of magnitude) the level of protection of a device from latch-up inducing ESD events. This tutorial is intended as a guide for helping designers choose protection. This type of protection is application dependent, so consideration should be made of the type of environment that the application, or the device, will be in.

FIGURE 1: CIRCUIT DIAGRAM FOR ESD-INDUCED LATCH-UP

![Circuit Diagram](image-url)
ELECTROSTATIC DISCHARGE

Electrostatic discharges can come from a variety of sources. The traditional ESD pulse is caused by a body at very high potential coming into contact, or near contact, with a grounded object. This could be a human body, a piece of electrical equipment, or even a piece of furniture.

In a dry environment, where static dissipation is low, a human body can develop tens of thousands of volts of potential. Almost everyone has experienced the shock of walking across a new carpet and touching a door handle. An audible snap can be heard when the potential difference between the person and the door handle is around 5,000V.

Any piece of equipment which has metal components moving against other metal components can develop a charge. An automated device handler will generally have a metal tray where devices move around, with pins in occasional contact with the tray. Static build up can reach hundreds of volts and can damage the devices in the same way as an ungrounded human handler can.

Incorrectly placed ionizers, meant to improve static dissipation, can build large potentials on office or laboratory furniture. Any person touching such a piece of furniture might feel a shock. Any devices being placed on a table with a large potential can suffer damage.

All these situations can be avoided by careful handling procedures. Groundstraps for manual handling of devices is essential, as are grounded tables and work surfaces with anti-static surfaces such as metal or specially designed plastic mats. Equipment can be carefully grounded wherever the potential exists for static build up.

ELECTROSTATIC DISCHARGE IN A POWERED APPLICATION

Once the device has been mounted to a board or installed in an application, most types of ESD events will no longer occur. For example, unless the PC board is out in the open, it is very unlikely to be touched by the user, and so a direct ESD pulse will not be a concern.

However, there are several sources of indirect ESD pulses, or noise pulses which are very similar in nature and magnitude to ESD pulses. ESD pulses can induce currents in nearby wiring. So, for example, if the user creates an ESD event on the casing of an application, the magnetic field of the pulse can induce currents inside the casing, in the wiring of an application. This electromagnetic interference can occasionally be seen in other ways, such as a noisy TV picture when a vacuum cleaner is being used, or a "click" heard on the radio when a light switch is turned on. This type of event is often called Radio Frequency Interference (RFI) or Electromagnetic Interference (EMI).

Often the application itself can induce noise spikes during operation. If one component in that application is a high speed, high current transistor or other type of...
switch, the sudden change in current can induce a noise spike which could be seen by that component, or other components in the system. This switching noise is endemic in systems with metal wires, and can not be removed completely. Large magnitude pulses of switching noise can induce latchup on sensitive CMOS devices.

The effects of switching noise or EMI can be catastrophic to an application with sensitive components. Even components which are not particularly sensitive or already have some built-in protection (such as Microchip Technology Inc.'s products) can still be latched-up by noise pulses if they are of a large enough magnitude.

PROTECTING AGAINST ESD PULSES

Basic protection can be provided by a simple decoupling capacitor, placed as close as possible to the power and ground pins of components. Each component should have its own capacitor; simply decoupling the whole application at the supply points will not be sufficient if a component in the application is producing noise.

If the value of the capacitor is chosen to match the device, then non-supply pins can also benefit from a decoupling capacitor between power and ground. A single capacitor can not filter out all the frequencies associated with a noise spike, but it can still offer very effective and low cost protection.

We used a simple test circuit to accurately model an ESD-induced noise spike. The test circuit is shown in Figure 1. A 200 pF capacitor was charged to various voltages and then switched to discharge directly into the device being tested. The waveform is a high frequency (=14 MHz) and short period (=100 ns) decaying oscillation. The resulting waveform, measured through a Tektronix CT-1 current probe with a 10X attenuator, is shown in Figure 2. The testing was conducted at 25°C.

The period of the oscillation is governed by the relationship:

$$\omega = \sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}$$

where L, C, and R are the inductance, capacitance, and resistance of the oscillating circuit, and w is the frequency of the oscillation. Since R and L are very small, and C is 200pF, the oscillation period is very fast.

Two devices were tested. A typical serial EEPROM, the 24LC04B, was tested on all function and supply pins. The results, shown in graphical form (Figure 3 through Figure 7) show that, even with a capacitor placed between power (Vcc) and ground (Vss) other pins, such as SDA can have increased protection. From the figures it is clear that the best all-round protection can be obtained from a 10,000 pF capacitor.

Particular applications may be different. For example, the designer may know that there is a better chance for a noise spike on the SDA signal, and so may want to use a 1,000 pF capacitor to improve the protection level of SDA.

A PIC16C54 microcontroller in XT mode was tested on all functional and supply pins. The results are shown in Figure 8 through Figure 17. From these results, it is clear that the particular application environment will be much more important in determining which capacitor value to use. Some pins, such as MCLR, do not respond at all to a decoupling capacitor. Others, such as the RA ports, respond strongly to a capacitor. Note that for the RA and RB graphs, the pin with the lowest latch-up threshold was used. All other pins are higher.

It is important for the designer to be aware of the potential problems associated with noise in a powered application. A combination of using sound design techniques to reduce causes of noise and an awareness of the protection levels of the components being used can make the problems of ESD manageable.
FIGURE 3: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vss. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 4: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON SDA. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 5: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON SCL. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 6: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON WP. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 7: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vcc. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 8: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RTCC. PROTECTION CAPACITOR BETWEEN VSS AND VCC.
FIGURE 9: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON MCLR. PROTECTION CAPACITOR BETWEEN VSS AND VCC.
FIGURE 10: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vss. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 11: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RB PORTS. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 12: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RA PORTS. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 13: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC1. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 14: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC2. PROTECTION CAPACITOR BETWEEN VSS AND VCC.
FIGURE 15: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC1. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 16: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RA PORTS. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
FIGURE 17: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vcc. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.
Plastic Packaging and the Effects of Surface Mount Soldering Techniques

Author: John Barber
Surface Mount Technology Team

PURPOSE

This application note is intended to inform and assist the customers of Microchip Technology Inc. with Surface Mount Devices (SMD's). The process of packaging a semiconductor in plastic brings to pass a somewhat unlikely marriage of different materials. In order to minimize potential adverse effects of surface mount solder techniques, it is worthwhile to understand the interaction of the package materials during the time they are subjected to thermal stress. Understanding both the limits of thermal stressing that SMD's can withstand and how those stresses interact to produce failures are crucial to successfully maintaining reliability in the finished product. A recommended Infrared (IR) solder profile is provided as a reference later.

The electronics industry has moved to smaller and thinner surface mount packaging in the progress toward miniaturization of circuits. This trend has necessitated the use of lower profile and smaller footprint packages. There has been an increase in reliability problems corresponding with the shrinking size of plastic SMD's. These problems manifest themselves in such ways as moisture sensitivity, cracked packages, open bond wires and intermittent continuity failures. Problems of this type are not present in the devices prior to assembly onto printed circuit boards but are the result of thermally induced stressing to the part during assembly or any rework such as desoldering.

WHAT CAN HAPPEN DURING THE SOLDER PROCESS

In surface mount soldering both the body and leads are intentionally heated. This direct heating of the reduced sized device package is at the heart of the problems experienced with Surface Mount Technology (SMT). Older techniques were concerned with the heating of the leads only. Some degree of heating was present in the body due to the thermal conductivity of the leadframe but this did not produce the same level of stress that devices are now subject to with SMT.

The heat from soldering causes a buildup of additional stresses within the device that were not present from the manufacturing process. Board level solder processes, such as IR reflow and Vapor phase reflow, are well-known areas where temperatures can reach levels sufficient to cause failure of the package integrity. A plastic semiconductor package forms a rigid system where the various components are locked together. Differences in the physical expansion rate of materials will result in internal package stresses because the constituent parts cannot move. When a package is heated, the stresses in the device are applied to the die in such a way that the maximum areas of stress are at the corners. Forces can build to the point where the areas of adhesion between different components of the package give way causing device failure. Failure modes associated with excess stress include delamination of surfaces, fracturing of bond pads, die cracking, cratering of bond pads and package cracking.

Moisture sensitivity of plastic packages has been a concern for semiconductor manufacturers. Moisture can and will permeate any molding compound. The rate of permeation will vary with package compound thickness and type. Relative humidity will also play a role in the time required to saturate a device. Moisture content will affect the ability of a device to withstand the stresses of surface mount soldering. If sufficient moisture is present inside of a device during soldering, high temperatures can cause steam which has the potential to crack the package. This type of damage is commonly called "pop-coming". Moisture can lead to corrosion of exposed aluminum metallization inside the device. Fortunately, a film of water is required for corrosion to take place. Water vapor alone is not sufficient to produce the onset of corrosion. It is difficult to collect a film of water inside a package where there is no defect.

Chemical compatibility is important to control corrosion of aluminum (especially in the presence of moisture). Most molding compounds and die attach epoxies contain free ions that can lead to corrosion under conditions where moisture is available to support the chemical reaction(s). Careful selection and handling of materials minimize the number of chemical impurities in the device that could lead to corrosion.
MATERIAL INTERACTIONS

To understand the significance of what is happening to a device during solder reflow, it is necessary to understand something about a few specific material properties and how those properties interact. Physical properties of interest are listed for reference in Table 2. These properties will be defined as needed to explain various concepts.

There are five major components used in a plastic package. Basic package components are: (1) epoxy molding compound, (2) leadframe made of copper or Alloy 42, (3) die attach epoxy, (4) silicon die and (5) gold bond wires. Molding compounds have several significant contributing factors that define their performance. These must be considered by the manufacturer when a selection is made. These items are Temperature of Glass Transition (Tg), Coefficient of Thermal Expansion (CTE), moisture absorption characteristics, flexural modulus and strength, and thermal conductivity. In reality, molding compound manufacturers have published data suggesting independent experimentation in this area. Leadframes used by Microchip are copper with a silver plated area for die attach and wire bonding. Die attach material is typically a silver filled epoxy. The silver is added for thermal and electrical conduction. Plastic devices have gold bond wires.

Thermally induced transients generated during surface mount soldering can have a significant impact on the reliability of plastic encapsulated devices in the field. In most cases, thermal transients below 125°C are not of sufficient magnitude to cause damage to the part. As a device experiences a significant thermal transient, such as would result from IR reflow soldering, the differing materials expand and contract at unequal rates. The CTE describes the behavior of a material as it expands or contracts when subjected to a temperature change. Materials with similar expansion coefficients will have similar thermal behavior if the phase boundaries are not approached. The CTE characterizes performance over a given temperature range. Table 2 shows that there is a change in the coefficient (ΔCTE) of roughly 12.9 (16 - 3.1 = 12.9) between a copper leadframe and silicon, and a ΔCTE of 4.4 (7.5 - 3.1 = 4.4) between silicon and lowest stress molding compounds available. The rate of expansion and contraction is not a simple linear relationship with temperature change. The rate can vary dramatically with the phase of the material. For example, molding compounds will greatly increase the rate of expansion when temperatures above the Tg is reached.

In general terms, Tg is the temperature where the material changes from a solid to something more like a plastic or vice versa. More precisely, the temperature of glass transition would be the temperature at which atoms, in chains of 30 to 40 atom groups, start to move.

Generally, Tg is in the area of 170°C for molding compounds. Several factors affect Tg such as the basic formulation of the resin from the supplier, cure time and the temperature used in the manufacturing process. A common misconception when trying to relate to these concepts is to assume that the entire package is at the same temperature at any given moment in time. This is not the case.

To aid in understanding the effects of CTE mismatching, take, for example, a bi-metallic strip. They can be found in every automobile and are used in turn signal flasher units for controlling the flashing action of the lights used as turn indicators. Inside the flasher unit the contact arm is formed by use of a bi-metallic strip to make and break electrical contact.

The metals used in the construction of a bi-metallic strip have different CTE's and are specifically chosen to produce a desired effect. These two materials are bonded so that they move together. In the automobile example, current flowing through the bi-metallic strip causes local heating of the strip. Due to the unequal CTE's, the strip will bend away from the contact as it is heated which causes the circuit to open. When current stops flowing in the strip, heating also stops. The bending action of the strip is produced by unequal expansion on one side of the strip. As the strip cools, it will move back thus making contact again allowing current induced heating which starts the cycle over again. In similar fashion as bi-metallic strip, the die in a plastic package will be stressed when subjected to a thermal transient.

Adhesion of materials is a matter of importance since it is in areas of delamination that moisture can collect. The collecting of moisture can lead to corrosion problems.

Differing rates of expansion and contraction can make joining two materials a distinct challenge. Especially if the materials are significantly different in other characteristics that affect surface adhesion. Some materials that are not suitable for use in the packaging system require special adhesion promoting modifications of the surfaces. The topic of adhesion is beyond the scope of this work, but is an important factor and should be given careful attention.
The property known as "Fracture Toughness" is the ability of the material to resist the propagation of a fracture once the defect has been initiated. Silicon has a very low fracture toughness, therefore, a fracture will readily propagate in silicon. An example of a material at the other end of the spectrum would be Gold, one of the ductile metals, which has a high fracture toughness. This property is responsible for the tendency for glass or tungsten carbide to shatter rather than simply chip or crack. In semiconductor devices, fracture toughness should be considered when cratering and die cracking are a problem.

The closest point to a zero stress state in a plastic package is at the temperature used to cure the molding compound (≤175°C). Present plastic molding compounds are thermosetting polymers. Thermosetting means that the compound sets up and becomes hard as a result of being heated. A cooling cycle after the set up period does not undo the process. The molding compound is held at an elevated temperature for the time period required to harden or cure. During the subsequent cooling, after cure, stresses are trapped in the package because of the differing CTE's. This trapping of stress results in a net compressive force, at room temperature, on the die surface in a plastic molded device. As external stresses such as thermal stress from soldering are presented, if stresses are of sufficient magnitude, the material strength will be exceeded resulting in package damage. Manufacturing processes tend to leave stresses trapped inside the devices. Thickness of the die attach material is regulated to control stressing levels due to its presence. A thin die attach will result in higher tensile stress on the die surface after die attach. The interaction of the die attach material and the molding compound result in the compressive stress on the die surface after packaging. Low stress molding compounds are used on SMD's to minimize the thermal stresses generated during soldering operations.

FAILURE MODES AND MECHANISMS

Let us now review the types of damage that may be seen as a result of SMT soldering. Package damage may be manifested in several ways and may not always result in immediate device failure. Following is a list of failure types and their morphology:

1. **Delamination** of the molding compound along the leadframe interface and or die surface can take place. This delamination, or separation, can provide a path for moisture and contaminant ingress and pooling along the interfaces where the materials are no longer adhering to each other. This condition may lead to corrosion related problems.

2. **Cracking of the mold compound** can produce immediate failure if the crack crosses a bond wire or it can allow similar moisture effects as in delamination. It can also produce intermittent contact problems.

3. **Cracking of the die** is generally seen as a functional failure but can be temperature sensitive if the crack is in a more benign area of the die.

4. **Cratering of wire bonds** is characterized as a phenomenon where portions of the silicon below the ball bond are fractured. The ball bonds pull up "plugs" or "chunks" of silicon, thereby, leaving crater shaped damaged spots in the bulk silicon below the bond pad. Cratering is a possible result of lateral stress on ball bonds. A cratering failure will typically but not always show up during electrical testing. However, it can lay dormant until another temperature excursion comes along which causes the Al metal conductor (bond pad) to open. This is a result of the soft nature of the aluminum used for interconnects in semiconductor devices. The silicon below a bond pad can be damaged without breaking contact in the aluminum. Intermittent or thermally sensitive continuity failures may be produced.

5. **Moisture** inside the device may collect in the die attach, along material interfaces (primarily leadframe to mold compound and leadframe to die surface) or in the molding compound. Rapid heating causes pressure build up as the moisture expands. This results in delamination and cracking of the package along with the failure modes associated with those phenomena.

6. **Corrosion** is heavily related to moisture effects and is intensified by delamination and cracks. The typical failure mode for corrosion is loss of continuity since the area most often affected is in the bond pads where there is no passivation layer to give additional protection to the metal.

HOW TO CORRECT THIS PROBLEM: A CASE STUDY

After the device leaves the factory there are two things that need to be done to maintain reliability.

1. The exposure to thermal stressing needs to be minimized. This is not always an easy task in light of the varied components that may go onto a single board. Some will require different conditions to solder due to there bulk size.

   Microchip carried out a partial factorial experiment on 32-lead PLCC devices with the intent of verifying an industry standard profile and providing a specification for our various customers. The devices were subjected to IR solder reflow profiles with ramp rates ranging from 1.5°C/second to 4°C/second and maximum temperatures from 220°C to 310°C (Table 1).
TABLE 1: 32-LEAD PLCC PACKAGE PARTIAL FACTORIAL EXPERIMENT

<table>
<thead>
<tr>
<th>Variables used in the analysis</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = Maximum Temperature (°C)</td>
<td>220 235 250 265 280 295 310</td>
</tr>
<tr>
<td>R = Ramp Rate (°C/second)</td>
<td>1.5 2 3 4</td>
</tr>
<tr>
<td>P = Number of Passes</td>
<td>1 2 56 Total Conditions</td>
</tr>
</tbody>
</table>

EXAMPLE 1: IR REFLOW PROFILE

No drypack state was employed and the parts were allowed to sit on the shelf in excess of 30 days prior to temperature exposure. The RH of the room was not recorded but is estimated to be approximately 60%. This condition was employed to insure that worst case conditions were evaluated.

Our experiments, to determine optimum soldering temperatures, indicated that a maximum temperature of 220°C should be observed during solder processing. The temperature ramp rate was found to be a less significant factor but should be kept in the range of 2 to 5°C per second. Very slow ramp rates, i.e. those of less than 2°C per second, show a slight decrease in performance. Ramp rates of 3°C showed the best performance and should be used whenever possible. Results from the experiment independently confirm the exposure recommendations set forth in IPC-SM-786. A sample profile for IR reflow is illustrated by the graph in Example 1.

Prior to exposure to the IR temperature profiles all parts were electrically good and were examined by Scanning Acoustic Microscopy (SAM) to verify a defect free state and establish a baseline. After exposure to the IR profiles, all parts were again tested electrically and examined via SAM. Delamination can be easily detected by use of SAM. No package cracks were observed after temperature exposure. Only limited electrical failures occurred and all were at temperatures above 220°C.

In the experiments a thermocouple was interfaced to the exterior of a device and passed through an IR reflow oven. Both the maximum temperature and ramp rate were recorded in this manner. General trends for temperature sensitivity in relationship to maximum temperature and ramp rate are presented in the graphs shown in Example 2 and Example 3.

Pass/fail criteria for the experiments was primarily the extent of delamination, observed by Scanning Acoustic Microscopy, after exposure to two passes through an IR reflow oven. Top side die delamination and delamination in areas of the leadframe where the bonds are made are classed as unacceptable. Some delamination along non critical areas are classed as acceptable (i.e., no delamination extending to the exterior of the leadframe).
package or covering more than minor portions of the leadframe). Specific information on pass/fail criteria can be found in IPC - SM - 786.

<table>
<thead>
<tr>
<th>Maximum temperature should not exceed 220°C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature ramp rate should be between 2 to 5°C per second.</td>
</tr>
</tbody>
</table>

Use of the above temperature profile will increase the reliability of the device following board level assembly.

2. The second item that must be observed is the dry packing of SM devices. Dry packing is a standard product packaging procedure for surface mount devices. This practice guarantees to the customer that devices are shipped in a dry state. All devices are sealed inside plastic bags with a moisture indicator to monitor moisture when they are shipped. Customers should be aware of this practice and maintain the dry packed state until the time of use. Shelf life for opened parts is a function of package style and ambient conditions. A small increase in failure rate will be experienced if the drypack condition is violated.

EXAMPLE 2: **PERCENT DELAMINATION BY MAXIMUM TEMPERATURE - ALL AREAS AVERAGED**

![Graph showing percent delamination by maximum temperature - all areas averaged.](image)

EXAMPLE 3: **DELAMINATION BY RAMP RATE AT 220°C**

![Graph showing delamination by ramp rate at 220°C.](image)
CONCLUSION

Significant thermal stressing can cause a diverse list of package or device failures. To minimize any damage caused by high temperature exposure, moisture content needs to be controlled in electronic devices.

Experiments conducted by Microchip validate the recommendation of IPC - SM - 786 to limit maximum temperature exposure during surface mount soldering to 220°C. A temperature ramp rate of 2°C/second to 5°C/second will also serve to safeguard the reliability of the device.

Microchip is committed to the principles of continuous improvement. Product reliability and customer satisfaction are a primary focus. For this reason, new packaging technology enhancements are continually evaluated to improve the performance of Microchip devices.

TABLE 2: PHYSICAL PROPERTIES OF COMMON MATERIALS USED IN PLASTIC PACKAGING OF SEMICONDUCTOR DEVICES

<table>
<thead>
<tr>
<th>Material</th>
<th>Coefficient of Thermal Expansion in 10^6/°C (CTE)</th>
<th>Temp. of Glass Transition in °C (Tg)</th>
<th>Modulus of Elasticity in 10^6 psi (10^5 MPa.) (E)</th>
<th>Tensile Strength in 10^3 psi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicon (Si)</td>
<td>2.3-4.2</td>
<td>19-29</td>
<td>16-25</td>
<td></td>
</tr>
<tr>
<td>Alloy 42 Leadframe</td>
<td>4.0- 4.7</td>
<td>21</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>Alumina (Al2O3)</td>
<td>6.5-8.8</td>
<td>37</td>
<td>25-50</td>
<td></td>
</tr>
<tr>
<td>Gold Eutectic Die Attach</td>
<td>14.2</td>
<td>12.5</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Copper Leadframe</td>
<td>16-18</td>
<td>17.3</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Pb -- Sn Die Attach</td>
<td>29</td>
<td>1.2-5</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>Epoxy Mold Compounds (α₁ is for Temps below Tg, α₂ is for Temps above Tg)</td>
<td>7.5-28, α₁ 20-22 ppm/°C, α₂ 60-65 ppm/°C^12</td>
<td>145-170</td>
<td>1.8-3</td>
<td>10-20</td>
</tr>
<tr>
<td>Epoxy Die Attach</td>
<td>20-70</td>
<td>0.4</td>
<td>1-8</td>
<td></td>
</tr>
<tr>
<td>Polymide Die Attach</td>
<td>35-60</td>
<td>0.6</td>
<td>1-8</td>
<td></td>
</tr>
<tr>
<td>Unfilled Epoxies</td>
<td>60-70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unfilled Silicones</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ENDNOTES


10. IPC - SM - 786A is obtainable from the Institute for Interconnecting and Packaging of Electronic Circuits, 7830 N. Lincoln Ave., Lincolnwood, ILL. 60646.

11. IPC - SM - 786A, Table 4-2 (See endnote above).


An Excellent Reference Article --

INTRODUCTION TO MICROCHIP'S CULTURE

The corporate culture at Microchip Technology Inc. is embodied in our Guiding Values. This culture has been key to our success in business because of its emphasis on customer satisfaction, quality, continuous improvement, empowerment and communication. The synergy of these values has created a very dynamic, very successful culture. They have impacted many aspects of Microchip, including the EEPROM Technology Team. Two of our Guiding Values in particular have steered this group.

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.

Products and Technology Are Our Foundation

We make ongoing investments and advancements in the design and development of our manufacturing process, device, circuit, system, and software technologies to provide timely, innovative, reliable, and cost effective products to support current and future market opportunities.

We recognized in the late 1980's that the industry as a whole did not have great endurance on EEPROMs. We were no exception. And this was at a 10,000 E/W cycle level, which is orders of magnitude less than where we are today. We also recognized that there were many uncertainties in the interaction of design, process and customers applications with respect to endurance performance and expectations. It was also clear that the customers were not educated enough on these interactions to be able to say clearly what they needed in a product. This was clearly an "improvement opportunity" that could allow us "to support current and future market opportunities".

THE EEPROM TECHNOLOGY TEAM

Microchip had been using the team approach to problem solving and continuous improvement activities such as new product introduction in the development groups and defect reduction in the manufacturing area. These had been in place for some time, and were beginning to work extremely well in bringing fast, well thought out and researched solutions to problems that crossed organizational boundaries.

A cross functional team was formed at Microchip that would be the technology management for an entire family of products. The team was comprised of engineers from design, process, test, product, quality, reliability, and yield enhancement, who reported in to different functional groups in the regular organizational structure. This team was charged with setting the technical direction for all EEPROMs at Microchip. This included both areas of development and sustaining, which were always in competition for the same resources.

The Microchip EEPROM Technology team went through the typical early phases of team development. The initial elation of having a great group of technical contributors wore off after a while. The competition for resources lead to some rough areas at the start, but this was overcome with the help of management support and direction. A balance developed, where the development side really understood the need to improve yields today, while the sustaining side understood the need for new products and innovations for the future. The emphasis on the Guiding Values and strong support from management for the team concept greatly helped the group through the early phases.
THE PROCESS

There is a team training program at Microchip that teaches high performance team leadership, and provides training on a set of tools for teams to use. These tools are aimed at improving the teams ability to identify problems, more importantly to identify the real root causes of the problems, to use the Pareto and other techniques to establish the "Vital Few", to brainstorm ideas on solutions, to investigate solutions prior to implementing, to execute implementations smoothly and accurately, to follow-up to ensure that the fix is as expected, and to re-examine the process to see what other improvements are needed. The key elements of this process are shown in the flowchart (Figure 1).

This process lead to improvements in every area of Microchip's EEPROM technology: design, process, product, test, quality, reliability, and yield. Most of these changes are confidential and proprietary, and several have been patented. These can only be discussed here in general terms.

FIGURE 1: MICROCHIP'S QUALITY IMPROVEMENT PROCESS

The actual EEPROM cell design has been the subject of repeated studies. There have been improvements made to the original that have increased process tolerance, increased yield, and of course dramatically increased endurance performance. "Repeated studies" sounds as if we did not get it right the first time, but in fact is just an acknowledgment that continuous improvement is essential. We have the best design in the industry today, but the EEPROM technology team is still working on improving it.

The process also has gone through many iterations of improvement. During some of the early studies, it was determined that new equipment with better control was needed to ensure the reproducibility of the product. This equipment was purchased, installed, and carefully monitored based on the recommendation from the team. We saw the expected improvement as a result.

There have also been ongoing studies to improve control, reproducibility, and defect reduction in many other areas of the process.

Note: TQI = Total Quality Improvement
SPC = Statistical Process Control
Improvements to the product itself have often shown up in the data sheet. These include reduction in current consumption, or improved timings as a result of improved design as verified by characterization on test equipment. We have greatly expanded the package variety for our Serial EEPROMs as a response to market needs. These activities have all been prioritized by the team with management input, evaluated, and implemented by the team.

The example of a new package introduction highlights the value of the team. A new package requires new test hardware, verification of the test program with the package and hardware, qualification of the package with the reliability group, assurance of acceptable yields based on the yield enhancement engineers involvement, and continued monitoring of the final products outgoing quality. Other companies often have these disciplines separated, with communication by memos back and forth. This is a cumbersome way to do business, and often details are overlooked, or the opportunity for synergy between groups to go the extra step is lost.

Testability is a vital concern for microelectronics. As the feature set grows, the ability to test them accurately can be lost. The introduction of the 64K-bit Serial EEPROM, with its security block and programmable endurance options, presented a concern for testability. By having even the initial design objective discussions in the team environment, with a test engineer, the designers were able to ensure that the new features could be tested thoroughly and cost effectively. This greatly improved our time to market, as well as the quality of the part that reached our customers.

The quality group has been a tremendous contribution to the success of the team. There were initial concerns on some members parts that this group would hold back progress by acting as “policeman” for the status quo. Quite the opposite, the quality group brought the statistical ability to measure the results of proposed changes with accuracy and confidence. So changes that really lead to improvement were actually implemented more quickly than they might have been without quality engineering input at the beginning. The team was also able to move on quickly from proposed changes that did not give measurable improvement as a result of having these measurement tools.

**FIGURE 2: ENDURANCE CHARTS**

![Endurance Charts](image)

*PERCENTAGE DEFECTIVE AFTER 100K CYCLES*

*PERCENTAGE DEFECTIVE AFTER 1 MILLION CYCLES*
The quality engineering group is also more directly exposed to customer needs than some of the other groups. They heard first hand what some of the areas for improvement were from the field. These inputs were brought directly into the team environment, where the breadth of engineering expertise we had was able to apply Microchip's Quality Improvement Process to them.

The dramatic improvement to the endurance performance of Microchip's Serial EEPROMs was a direct result of the reliability group's involvement with the team. This group was able to establish standard methods for testing and measuring endurance on products. This was the first step in determining that this was another area for continuous improvement. Inputs from the business unit were that this was going to develop into a critical point of competition in the industry. We found out later that our standards were significantly higher than most competitors. The reliability group set up competitive analysis and benchmarking between vendors that put us on a level playing field. We found that Microchip's endurance was much better than we thought! Even so, the Microchip EEPROM Technology team drove programs for improving endurance that gave impressive results (Figure 2). And the elimination of the root causes of endurance problems gave equally impressive gains in other areas of reliability. This is shown is the reduced failure rates for both dynamic life testing and retention bake (Figure 3 and Figure 4).

The group also kept one eye on the bottom line through the years. Yields have always been a concern, since continuous improvement in that area is also vital to the business. The yield enhancement engineers, like the other groups, brought tools with which to measure and identify problems. As with the others, they were dependent on the rest of the team to help find and implement solutions. They also acted as a conduit to the front end teams that were implementing defect density reduction and process control improvements in the manufacturing process across product families. This coordination between the teams lead to a whole new level of synergy that allowed other products to benefit from improvements from the EEPROM team, and the EEPROM product line to piggyback on improvements aimed primarily at other areas. These other teams were focused on plant wide areas of continuous improvement: oxide integrity, the poly-silicon modules, the metal modules, defect reduction, wafer handling techniques, etc. The result of this cross functional synergy is shown is the yield trend (Figure 5).

As we grew and learned together, each member developed increasing respect and trust for the others. This was critical to the team's development. Not all of the proposed changes were successful. In the stages of implementing, follow-up, and maintenance, some proposals did not hold up. Either they did not really fix a root cause, or the implementation caused other problems to surface. Only by having established respect and trust was the team able to raise and address these issues quickly.

FIGURE 3: EEPROM DYNAMIC LIFE FAILURE RATE
THE RESULTS

The improvements to both areas that this team has made are spectacular. At the time of formation, the industry standard was 10,000 E/W cycles of endurance, and 4K bits of memory. The Microchip lead the world when the efforts of the team lead to introducing:

- The first 10,000,000 E/W cycle guarantee
- The very first 64K bit Serial EEPROM
- The first to put a 64K bit Serial EEPROM in an 8-pin SOIC package
- The first family of Serial EEPROMs to operate at 1.8V
- The first VESA busing compatible Serial EEPROMs

These improvements are orders of magnitude increases in critical areas of market need. They establish Microchip Technology Inc. as the technical leader in EEPROM technology. This is what the EEPROM Technology team set out to do.

The team's efforts continue today to make sure Microchip remains a technical leader. We have the best products in the market today. But that will not be good enough tomorrow. Continuous improvement is essential.

FIGURE 4: EEPROM RETENTION BAKE FAILURE RATE

FIGURE 5: WORLD CLASS MANUFACTURING
SECTION 6
DEVELOPMENT TOOLS

DEVELOPMENT SYSTEMS:

System Support  
Development System Selection Chart ................................................................. 6-1
Microchip BBS  
Microchip Bulletin Board Service ......................................................................... 6-3
Design Kit  
Microchip Serial EEPROM Designer’s Kit .............................................................. 6-5
PICMASTER™  
Universal In-Circuit Emulator System with MPLAB IDE ......................................... 6-7
PRO MATE™  
Universal Device Programmer .................................................................................. 6-13
PICSEE™ Tools  
PICSEE Product Development Tools ...................................................................... 6-17
PICSTART™-16B1  
PIC16CXX Low-Cost Microcontroller Development System ........................................ 6-25
PICSTART-16C  
PIC16CXX Low-Cost Microcontroller Development System ...................................... 6-27
PICDEM-1  
Low-Cost PIC16/17 Demonstration Board ................................................................ 6-29
PICDEM-2  
Low-Cost PIC16CXX Demonstration Board ................................................................. 6-31

SOFTWARE TOOLS:

fuzzyTECH®-MP  
Fuzzy Logic Development System for PIC16/17 ....................................................... 6-33
MPASM  
Universal PIC16/17 Microcontroller Assembler Software ........................................... 6-35
MPSIM  
PIC16/17 Microcontroller Simulator .......................................................................... 6-37
MP-C  
C Compiler ................................................................................................................. 6-39
Total Endurance™  
Microchip Serial EEPROM Endurance Model .......................................................... 6-41
TrueGauge™ Tools  
TrueGauge Intelligent Battery Management Development Tool ................................ 6-43
## Development System Selection Chart

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>PIC16/17 Microcontroller</th>
<th>ASSP</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulator Systems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICMASTER™-16B</td>
<td>PIC16C54</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-16C</td>
<td>PIC16C55</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-16D</td>
<td>PIC16C56</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-16E</td>
<td>PIC16C57</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-16F</td>
<td>PIC16C58A</td>
<td>X X X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-16G</td>
<td>PIC16C621</td>
<td>X X X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-16H</td>
<td>PIC16C622</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-17</td>
<td>PIC16C623</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>PICMASTER-17B</td>
<td>PIC16C624</td>
<td>X X X</td>
<td></td>
</tr>
<tr>
<td>Emulator Probe Kits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16B</td>
<td>PIC16C74</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16C</td>
<td>PIC17C42</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16D</td>
<td>PIC17C44</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16E</td>
<td>PIC17C44</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16F</td>
<td>PIC17C44</td>
<td>X X X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16G</td>
<td>PIC17C44</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-16H</td>
<td>PIC17C44</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-17A</td>
<td>PIC17C44</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICPROBE-17B</td>
<td>PIC17C44</td>
<td>X X X</td>
<td></td>
</tr>
<tr>
<td>Development Kits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fuzzyTECH®-MP</td>
<td>PIC16C54</td>
<td>X X X X X</td>
<td></td>
</tr>
<tr>
<td>PICSTART™-16B1</td>
<td>PIC16C55</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>PICSTART-16C</td>
<td>PIC16C56</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>PRO MATE™</td>
<td>PIC16C57</td>
<td>X X X X</td>
<td></td>
</tr>
<tr>
<td>PICSEEKIT-81A</td>
<td>PIC16C58A</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>PICSEEKSTART-81A</td>
<td>PIC16C621</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICSEEKSTART-85A</td>
<td>PIC16C622</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PICSEEKSTART-85A</td>
<td>PIC16C623</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Serial EEPROM Designer's Kit</td>
<td>PIC16/17 Software Tools:</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TrueGauge™</td>
<td>MPASM/MPSIM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Endurance Software Model</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Demonstration Boards</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICDEM-1</td>
<td>PIC16C54</td>
<td>X X X X X</td>
<td></td>
</tr>
<tr>
<td>PICDEM-2</td>
<td>PIC16C55</td>
<td>X X X X X</td>
<td></td>
</tr>
</tbody>
</table>

© 1995 Microchip Technology Inc.
## Compatible & Universal

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
<th>PIC16C5X</th>
<th>PIC16CXX</th>
<th>PIC17CXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPASM</td>
<td>Universal Assembler</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MPSIM</td>
<td>Software Simulator</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MP-C</td>
<td>C Compiler</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>fuzzyTECH</td>
<td>Fuzzy Logic Development Kit</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PICSTART</td>
<td>Low Cost Development Kit</td>
<td>X</td>
<td>X</td>
<td>—</td>
</tr>
<tr>
<td>PRO MATE</td>
<td>Universal Programmer/Development Kit</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PICMASTER</td>
<td>Universal In-Circuit Emulator</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MPLAB</td>
<td>Integrated Development Environment</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Get current information and help on Microchip's Bulletin Board Service (BBS)! Microchip wants to provide you with the most 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.

SYSTEMS INFORMATION AND UPGRADE HOT LINE

The Systems Information And Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits. The Hot Line Numbers are: 1-800-755-2345 for U.S. and most of Canada, and 1-602-786-7302 for the rest of the world.

These phone numbers are also listed on the "Important Information" sheet that is shipped with all development systems. The hot line message is updated whenever a new software version is added to the Microchip BBS, or when a new upgrade kit becomes available.

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. There is no charge for connecting to the BBS, except for a toll charge to the CompuServe access number, where applicable. You do not need to be a CompuServe member to take advantage of this connection (you never actually log in to CompuServe).

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 service allow multiple users at baud rates up to 14400 bps.

The following connect procedure applies in most locations.

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 access number.
3. Depress <Enter> and a garbage string will appear because CompuServe is expecting a 7E1 setting.
4. Type +, depress <Enter> and Host Name: will appear.
5. Type MCHIPBBS, depress <Enter> and you will be connected to the Microchip BBS.

In the United States, to find CompuServe's phone number closest to you, set your modem to 7E1 and dial (800) 848-4480 for 300-2400 baud or (800) 331-7166 for 9600-14400 baud connection. After the system responds with Host Name:, type NETWORK, depress <Enter> and follow CompuServe's directions.

For voice information (or calling from overseas), you may call (614) 457-1550 for your local CompuServe number.

USING THE BULLETIN BOARD

The bulletin board is a multifaceted tool. It can provide you with information on a number of different topics.

- Special Interest Groups
- Files
- Mail
- Bug Lists

Special Interest Groups

Special Interest Groups, or SIGs as they are commonly referred to, provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. SIGs may provide you with information not available by any other method because of the broad background of the PIC16/17 user community.

There are SIGs for most Microchip systems, including:

- MPASM
- PICMASTER™
- PRO MATE™
- PICSTART™
- Utilities
- Bugs
- MPSIM
- TRUE GAUGE™
- fuzzyTECH®-MP
- ASSP
- MTE1122
Note: The SIGs provide you with the opportunity to discuss issues and exchange ideas. Technical support and urgent questions should be referred to your local distributor, FAE or sales representative. They are your first level of support.

These groups are monitored by the Microchip staff.

Files
Microchip regularly uses the Microchip BBS to distribute technical information, application notes, source code, errata sheets, bug reports, and interim patches for Microchip systems software products. Users can contribute files for distribution on the BBS. For each SIG, a moderator monitors, scans, and approves or disapproves files submitted to the SIG. No executable files are accepted from the user community in general to limit the spread of computer viruses.

Mail
The BBS can be used to distribute mail to other users of the service. This is one way to get answers to your questions and problems from the Microchip staff, as well as keeping in touch with fellow Microchip users worldwide.

Consider mailing the moderator of your SIG, or the SYSOP, if you have ideas or questions about Microchip products, or the operation of the BBS.

Software Releases
Software products released by Microchip are referred to by version numbers. Version numbers use the form:

xx.yy.zz <status>

Where xx is the major release number, yy is the minor number, and zz is the intermediate number. The status field displays one of the following categories:
- Alpha
- Intermediate
- Beta
- Released

Production releases are numbered with major, and minor version numbers like:

3.04 Released

Alpha, Beta and Intermediate releases are numbered with the major, minor and intermediate numbers:

3.04.01 Alpha

Intermediate Release
Intermediate released software represents changes to a released software system and is designated as such by adding an intermediate number to the version number. Intermediate changes are represented by:
- Bug Fixes
- Special Releases
- Feature Experiments

Intermediate released software does not represent our most tested and stable software. Typically, it will not have been subject to a thorough and rigorous test suite, unlike production released versions. Therefore, users should use these versions with care, and only in cases where the features provided by an intermediate release are required.

Intermediate releases are primarily available through the BBS.

Beta Release
Preproduction software is designated as Beta. Beta software is sent to Applications Engineers and Consultants, FAEs, and select customers. The Beta Test period is limited to a few weeks. Software that passes Beta testing without having significant flaws, will be production released. Flawed software will be evaluated, repaired, and updated with a new revision number for a subsequent Beta trial.

Production Release
Production released software is software shipped with tool products. Example products are PROMATE, PICSTART, and PICMASTER. The Major number is advanced when significant feature enhancements are made to the product. The minor version number is advanced for maintenance fixes and minor enhancements. Production released software represents Microchip's most stable and thoroughly tested software.

There will always be a period of time when the Production Released software is not reflected by products being shipped until stocks are rotated. You should always check the BBS for the current production release.

Alpha Release
Alpha designated software is engineering software that has not been submitted to any quality assurance testing. In general, this grade of software is intended for software development team access only, but may be sent to selected individuals for conceptual evaluation.
FEATURES

- Includes everything necessary to begin developing Serial EEPROM-based applications
- Microchip Total Endurance™ software model
- Microchip SEEVAL™ evaluation and programming board
- Microchip SEEVAL software
- Microchip Serial EEPROM handbook
- Microchip Serial EEPROM sample pack
- RS-232 serial cable
- Power supply

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows® 3.1
- VGA monitor
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/85CXX
- Microchip Smart Serial™ 24XX65
- Microchip 3-wire 93CXX/93LCXX series
- Microchip 4-wire 59C11
**Designer’s Kit**

**DESCRIPTION**

Now designers of Serial EEPROM-based applications can enjoy the increased productivity, reduced time to market, and the ability to create a rock-solid design that only a well-thought-out development system can provide. Microchip’s new Serial EEPROM Designer’s Kit includes everything necessary to quickly develop a robust and reliable Serial EEPROM-based design and greatly reduce the time required for system integration and hardware/software debug.

The Total Endurance software model enables designers to quickly choose the best Serial EEPROM for the specific application and perform trade-off analysis with voltage, temperature, write cycle and other system parameters in order to achieve the desired Erase/Write endurance (specific ppm rate) or product lifetime. Total Endurance is the new standard of excellence in understanding and predicting the Erase/Write endurance of Serial EEPROMs. An on-line endurance tutorial is included, along with hypertext help files.

Microchip’s SEEVAL Serial EEPROM evaluation and programming system will accept any Microchip Serial EEPROM in DIP package and enable the designer or system integrator to read, write, or erase any byte or the entire array. SEEVAL also provides the following advanced features to aid in system integration and debug:

- Program special user functions like Smart Serial configurations
- Hexadecimal display of array contents
- Pre-set or user-defined repeating patterns
- User-configurable functions like continuous read/write, programmable delay, etc.
- Upload/download files between the Serial EEPROM and disk

Another industry first, the Microchip Serial EEPROM Handbook provides a plethora of information crucial to the designers of Serial EEPROM-based systems. Along with data sheets on Microchip Serial EEPROMs, this resource provides application notes regarding Erase/Write endurance, interfacing with different protocols and many, many others. A cross-reference and selector guide are also included, plus article reprints and qualification reports on Microchip Serial EEPROMs.

**USING SEEVAL AND TOTAL ENDURANCE**

Both software packages can be loaded from Windows by choosing FILE RUN and entering SETUP.EXE from the Program Manager. The applications will install themselves; then a double mouse-click will start either application. The first step in either program is to select a device from the device list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

In SEEVAL, the user may choose to load a file from disk to program the Serial EEPROM, or read data from the EEPROM and save it to disk. The screen displays the contents of a software buffer. The buffer may be manipulated before programming data to the Serial EEPROM, or data can be written to the Serial EEPROM directly on-line. An area of memory can be highlighted (selected) and programmed with a predefined pattern or user-specified pattern. Alternately, the entire device can be programmed with any repeating pattern.

Both SEEVAL and Total Endurance allow the user to save any configuration as default. This configuration (device and application settings) will then automatically load at boot time.

**Order Information:**

<table>
<thead>
<tr>
<th>Description</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial EEPROM Designer’s Kit</td>
<td>DV243001</td>
</tr>
</tbody>
</table>

© 1995 Microchip Technology Inc.
SYSTEM FEATURES

General:

- Complete Hi-Performance PC-based Microcontroller Development System for the PIC16/17 families.
- For use on PC-compatible 386, 486 and Pentium machines under Microsoft® Windows® 3.X environment (also runs in Windows 95).
- Assembler and Simulator Software, Emulator System, and EPROM Programmer unit, sample kit, and demonstration hardware and software provide a complete microcontroller product development environment.

Windows IDE:

- Built-in full-featured programmer's text editor.
- Project manager keeps track of files, automatically downloads new builds to emulator/simulator and updates symbolic information into break/trace/trigger point settings.
- Customizable tool bar.
- Editor, emulator, simulator modes.
- Status bar shows development mode, editor information, current PC and W register and status register values.
- Extensive on-line help
- Customizable IDE desktop so key mappings, colors, symbol width, and toolbar position can be changed and saved.
Emulator System:
- Hi-Performance In-Circuit Emulation of Microchip Microcontrollers.
- Real-time instruction emulation.
- 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 four PICMASTER emulators can be synchronized on a single PC, for multi-processor development.
- Extended 48-bit stopwatch measures execution time.
- 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).
- Pass counter for break and trace points.

EPROM Programmer System:
- PRO MATE™ Device Programmer unit for all current PIC16/17 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 PIC16/17 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.

Simulator:
MPSIM is a discrete event software simulator designed to imitate operation of PIC16/17 microcontrollers. It allows the user to debug software that will use any of these microcontrollers. At any instruction boundary, MPSIM also allows the user to examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. The simulator has the exact same look and feel as the PICMASTER emulator but provides the most cost-effective debugging tools. Once operation of the system with the simulator has been mastered, upgrading to the PICMASTER emulator is an easy transition, since the operation is virtually identical.

Demo Board:
The PICDEM Demonstration Board provides a user with a simple hardware tool through which software can be exercised and debugged. A step-by-step tutorial enables first-time users of PICMASTER to become familiar with all the features of the emulator. A generous prototype area allows the user to build additional hardware for their project.

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 PIC16/17 family. The PICMASTER system currently supports the PIC16C54, PIC16C54A, PIC16C55, PIC16C56, PIC16C57 and PIC16C58A at clock frequencies of 20MHz; the PIC16C61, PIC16C62, PIC16C620, PIC16C621, PIC16C622, PIC16C64, PIC16C65, PIC16C71, PIC16C73, PIC16C74, PIC16C84 to 10MHz; and the PIC17C42, PIC17C42A, PIC17C43, and PIC17C44 to 16 MHz.

© 1995 Microchip Technology Inc.
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.

Provided with the PICMASTER System is the MPLAB integrated development environment, a high performance, real-time In-Circuit Emulator, a microcontroller programmer unit, a macro assembler program, and an Unusually sophisticated simulator program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC16/17 microcontroller families. The MPLAB IDE user interface features a built-in text editor, project management, and full integration with MPASM and ByteCraft's C compiler, MPC, for rapid product development.

A "Quick Start" Product Sample Pak containing user programmable parts is included for additional convenience (only devices supported by the specified probe header).

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:

MPLAB and PICMASTER have been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The IBM PC-compatible platform and Windows 3.X environment was chosen to best make these features available to you the end user. To properly take advantages of these features, MPLAB and PICMASTER require installation on a system having the following minimum configuration:

- PC-compatible machine: 386DX, 486 or Pentium with ISA or EISA Bus.
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MS-DOS® / PC-DOS version 5.0 or greater.
- Microsoft Windows version 3.10 or greater operating in 386 enhanced mode).
- 4 Mbyte RAM.
- One 3.5" floppy disk drive.
- Approximately 4 Mbytes of free hard disk space.
- 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 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, PIC16CXX family, and the PIC17C4X family microcontrollers are supported. Future microcontroller probes will be available as they are released.

- **MPLAB PC Software**: Host software necessary to control and provide an integrated development environment (IDE) working user interface is the last major component of the system. This software contains a built-in text editor, project manager, PICMASTER emulator drivers, MPSIM simulator drivers (available soon), the software runs in the Windows 3.X environment and provides the user with quick editing, debugging, optimization and control of the system under emulation. MPLAB software is universal to all microcontroller families.

Dynamic Data Exchange (DDE), a feature of Windows 3.X, can be utilized in this version to allow data to be dynamically transferred between two or more Windows programs. Data collected with MPLAB can be automatically transferred to a spreadsheet or database program for further analysis.

Up to four 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).

**PRO MATE Device Programmer:**
The PRO MATE Programmer system included in the PICMASTER Development System provides the product developer with the ability to program (transfer) the developer's software into PIC16/17 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, a power supply unit, and host operating software.
The PRO MATE 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 MATE 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 MATE 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:

**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.

**PICMASTER PROBE Specifications**

Table 1 shows the current probe specifications for the PICMASTER In-Circuit Emulator. The devices are supported regardless of program memory type (ROM, EPROM or EEPROM), process technology or voltage range. That is, selecting the PROBE that supports the PIC16C54 (Probe-16D) also supports the PIC16CR54, PIC16C54A and the PIC16LC54A devices. The probe would also support other variations as they become available (such as PIC16CR54A).

**TABLE 1: PICMASTER PROBE SPECIFICATIONS**

<table>
<thead>
<tr>
<th>PICMASTER Probe</th>
<th>Devices Supported</th>
<th>Maximum Frequency</th>
<th>Operating Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBE-16B</td>
<td>PIC16C71</td>
<td>10 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-16C</td>
<td>PIC16C84</td>
<td>10 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-16D</td>
<td>PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C58A</td>
<td>20 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-16E</td>
<td>PIC16C64, PIC16C62</td>
<td>10 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-16F</td>
<td>PIC16C63, PIC16C65, PIC16C74, PIC16C73</td>
<td>10 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-16G</td>
<td>PIC16C61</td>
<td>10 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-16H</td>
<td>PIC16C620, PIC16C621, PIC16C622</td>
<td>10 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>PROBE-17B</td>
<td>PIC17C42, PIC17C43, PIC17C44</td>
<td>16 MHz</td>
<td>4.5V - 5.5V</td>
</tr>
</tbody>
</table>
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>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>EM167015</td>
<td>Complete PICMASTER-16D System for PIC16C5X</td>
</tr>
<tr>
<td>EM167016</td>
<td>Complete PICMASTER-16D System for PIC16C5X 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 PIC16C62, PIC16C64 without Programmer</td>
</tr>
<tr>
<td>EM167019</td>
<td>Complete PICMASTER-16F System for PIC16C65, PIC16C74/C73 without Programmer</td>
</tr>
<tr>
<td>EM167020</td>
<td>Complete PICMASTER-16F System for PIC16C63, PIC16C65, PIC16C74/C73 without Programmer</td>
</tr>
<tr>
<td>EM167021</td>
<td>Complete PICMASTER-16G System for PIC16C61</td>
</tr>
<tr>
<td>EM167022</td>
<td>Complete PICMASTER-16G System for PIC16C61 without Programmer</td>
</tr>
<tr>
<td>EM167023</td>
<td>Complete PICMASTER-16H System for PIC16C62X</td>
</tr>
<tr>
<td>EM167024</td>
<td>Complete PICMASTER-16H System for PIC16C62X without Programmer</td>
</tr>
<tr>
<td>EM177007</td>
<td>Complete PICMASTER-17B System for PIC17C42</td>
</tr>
<tr>
<td>EM177008</td>
<td>Complete PICMASTER-17B System for PIC17C42, PIC17C43, PIC17C44 without Programmer</td>
</tr>
</tbody>
</table>
SYSTEM FEATURES

Device Programmer System:
- PRO MATE Programmer unit for the PIC16C5X, PIC16CXX, 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.
- Communicates with PC via RS-232.
- Modular socket modules provide easy migration from one PIC16/17 microcontroller product to another.

SYSTEM DESCRIPTION

PRO MATE Programmer:
The PRO MATE Programmer system provides the product developer with the ability to program user software into PIC16C5X, PIC16CXX, PIC17CXX CMOS microcontrollers.

PRO MATE is also supplied with a discrete event software simulator (MPSIM) and a Universal PIC16/17 Macro assembler (MPASM).

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 MATE 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.

The modular socket module design allows users to easily migrate between PIC16/17 devices at the lowest possible cost.
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 MATE provides a very user friendly user interface which allows complete control over the programming session.

The PRO MATE 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 MATE 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 MATE 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 PIC16/17 microcontroller. Once a serialization file is generated, it may be used over different programming sessions. Serial numbers are automatically marked as used when a PIC16/17 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.

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16/17 microcontroller product family.
- Macro-Assembly capability.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16/17 Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16/17 microcontroller product family.
- For PC-compatible systems running the MS-DOS® operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.
### PRO MATE SOCKET MODULE CROSS-REFERENCE

<table>
<thead>
<tr>
<th>Device Description</th>
<th>Pin Count</th>
<th>DIP</th>
<th>SOIC</th>
<th>SSOP</th>
<th>PLCC</th>
<th>MQFP</th>
<th>TQFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C54</td>
<td>18/20</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C54A</td>
<td>18/20</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16CR54</td>
<td>18/20</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16CR54A</td>
<td>18/20</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C55</td>
<td>28</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C56</td>
<td>18/20</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C57</td>
<td>28</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16CR57A</td>
<td>28</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C58A</td>
<td>18/20</td>
<td>AC164001</td>
<td>AC164002</td>
<td>AC164015</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C620</td>
<td>18/20</td>
<td>AC164010</td>
<td>AC164010</td>
<td>AC164018</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C621</td>
<td>18/20</td>
<td>AC164010</td>
<td>AC164010</td>
<td>AC164018</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C622</td>
<td>18/20</td>
<td>AC164010</td>
<td>AC164010</td>
<td>AC164018</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C61</td>
<td>18</td>
<td>AC164010</td>
<td>AC164010</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C62</td>
<td>28</td>
<td>AC164012</td>
<td>AC164017</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C63</td>
<td>28</td>
<td>AC164012</td>
<td>AC164017</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C64</td>
<td>40/44</td>
<td>AC164012</td>
<td>-</td>
<td>-</td>
<td>AC164013</td>
<td>AC164014</td>
<td>AC164020</td>
</tr>
<tr>
<td>PIC16C65</td>
<td>40/44</td>
<td>AC164012</td>
<td>-</td>
<td>-</td>
<td>AC164013</td>
<td>AC164014</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C71</td>
<td>18</td>
<td>AC164010</td>
<td>AC164010</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C73</td>
<td>28</td>
<td>AC164012</td>
<td>AC164017</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC16C74</td>
<td>40/44</td>
<td>AC164012</td>
<td>-</td>
<td>-</td>
<td>AC164013</td>
<td>AC164014</td>
<td>AC164020</td>
</tr>
<tr>
<td>PIC16C84</td>
<td>18</td>
<td>AC164010</td>
<td>AC164010</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIC17C42</td>
<td>40/44</td>
<td>AC174001</td>
<td>-</td>
<td>-</td>
<td>AC174002</td>
<td>AC174004</td>
<td>-</td>
</tr>
<tr>
<td>PIC17C43</td>
<td>40/44</td>
<td>AC174001</td>
<td>-</td>
<td>-</td>
<td>AC174002</td>
<td>-</td>
<td>AC174005</td>
</tr>
<tr>
<td>PIC17C44</td>
<td>40/44</td>
<td>AC174001</td>
<td>-</td>
<td>-</td>
<td>AC174002</td>
<td>-</td>
<td>AC174005</td>
</tr>
</tbody>
</table>
## SALES AND SUPPORT

To order or obtain information (e.g., pricing or delivery), please use listed part numbers and refer to listed sales offices.

### Programmer Part Number

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DV007001</td>
<td>Programmer Kit as described above</td>
</tr>
</tbody>
</table>

### Socket Part Number

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC164001</td>
<td>PIC16C54 through C58A 18- &amp; 28-Lead PDIP Socket Module</td>
</tr>
<tr>
<td>AC164002</td>
<td>PIC16C54 through C58A 18- &amp; 28-Lead SOIC Socket Module</td>
</tr>
<tr>
<td>AC164010</td>
<td>PIC16C61, PIC16C62X, PIC16C71, PIC16C84, 18-Lead PDIP/SOIC Socket Module</td>
</tr>
<tr>
<td>AC164012</td>
<td>PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C73, PIC16C74, 40-Lead PDIP Socket Module</td>
</tr>
<tr>
<td>AC164013</td>
<td>PIC16C64, PIC16C65, PIC16C74, 44-Lead PLCC Socket Module</td>
</tr>
<tr>
<td>AC164014</td>
<td>PIC16C64, PIC16C65, PIC16C74, 44-Lead PQFP Socket Module</td>
</tr>
<tr>
<td>AC164015</td>
<td>PIC16C54 through PIC16C58A, 20 and 28-Lead SSOP Socket Module</td>
</tr>
<tr>
<td>AC164017</td>
<td>PIC16C62, PIC16C63, PIC16C73, 28-Lead SOIC Socket Module</td>
</tr>
<tr>
<td>AC164018</td>
<td>PIC16C62X 20-Lead SSOP Socket Module</td>
</tr>
<tr>
<td>AC164020</td>
<td>PIC16C64, PIC16C74, 44-Lead TQFP Socket Module</td>
</tr>
<tr>
<td>AC174001</td>
<td>PIC17C42, PIC17C43, PIC17C44, 40-Lead PDIP Socket Module</td>
</tr>
<tr>
<td>AC174002</td>
<td>PIC17C42, PIC17C43, PIC17C44, 44-Lead PLCC Socket Module</td>
</tr>
<tr>
<td>AC174004</td>
<td>PIC17C42 44-Lead QFP Socket Module</td>
</tr>
<tr>
<td>AC174005</td>
<td>PIC17C43, PIC17C44, 44-Lead TQFP Socket Module</td>
</tr>
</tbody>
</table>

### PRO MATE CROSS REFERENCE BY SOCKET PART NUMBER

<table>
<thead>
<tr>
<th>Device</th>
<th>Pin Count</th>
<th>AC164001</th>
<th>AC164002</th>
<th>AC164010</th>
<th>AC164012</th>
<th>AC164013</th>
<th>AC164014</th>
<th>AC164015</th>
<th>AC164017</th>
<th>AC164018</th>
<th>AC164020</th>
<th>AC174001</th>
<th>AC174002</th>
<th>AC174003</th>
<th>AC174004</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C54</td>
<td>18/20</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>PIC16C54A</td>
<td>18/20</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>PIC16CR54</td>
<td>18/20</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>PIC16C55</td>
<td>28</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>PIC16C56</td>
<td>18/20</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>PIC16C57</td>
<td>28</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>PIC16CR57A</td>
<td>28</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>PIC16C58A</td>
<td>18/20</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>PIC16C61</td>
<td>18</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>PIC16C62</td>
<td>28</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>PIC16C63</td>
<td>28</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>PIC16C64</td>
<td>40/44</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>PIC16C65</td>
<td>40/44</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>PIC16C620</td>
<td>18/20</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>PIC16C621</td>
<td>18/20</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>PIC16C622</td>
<td>18/20</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>PIC16C71</td>
<td>18</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>PIC16C73</td>
<td>28</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>PIC16C74</td>
<td>40/44</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>PIC16C84</td>
<td>18</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>PIC17C42</td>
<td>40/44</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>PIC17C43</td>
<td>40/44</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>PIC17C44</td>
<td>40/44</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>
INTRODUCTION

The PICSEE Development Systems provide the product development engineer with cost effective and timely design tool solutions for the MTA8XXXX family of 8-bit CMOS microcontrollers with Serial EEPROM. They are designed specifically for the MTA8XXXX family. These tools work in conjunction with existing hardware and software design tools for the PIC16CXX microcontroller family. This allows the development engineer to efficiently implement systems utilizing these multichip modules with a minimal learning curve and capital investment.

PICSEEKIT-81A — P/N AC812001

- Supports MTA81010
- Programming Adapters for PDIP and SOIC packages
- Daughter card for PICPROBE-16D
- I²C™ bus Serial Communication Application Software

This kit supports the MTA81010 multichip module. It contains programming adapters, a PICMASTER™ emulator daughter board and MTA81010 product samples in a 28-lead PDIP. The kit also includes an MS-DOS®, 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 its 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 kit. 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-81A — P/N DV813001

- Complete Low-Cost Development Solution for MTA81010
- Combines PICSEEKIT-81A (AC812001) and PICSTART-16B1 (DV163003)
- MPASM Assembler
- MPSIM Simulator
- Low-Cost Programmer
- Programming Adapter Sockets
- I²C Bus Applications Software

This kit combines the PICSEEKIT (P/N AC812001) with a PICSTART™-16B1 (P/N DV163003) 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 DV163003 product descriptions.
This kit supports the MTA85XXX multichip module. It contains a programming adapter, a PICMASTER adapter socket, and product samples. Also included is an MS-DOS, 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
A programming adapter socket is provided to allow the MTA85XXX's internal program EPROM as well as its data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip's PIC16C54 can program the MTA85XXX's internal EPROM. Also, any programmer that supports Microchip's 24LC01B and 24LC02B Serial EEPROM can program the MTA85XXX's internal Serial EEPROM. There is one adapter for the SSOP package for MTA85XXX.

Emulation Support
An adapter socket allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA85XXX Microcontroller with Serial EEPROM. The adapter socket provides the required translation from a PIC16C54A/58A pin out to the MTA85XXX pin out. It also contains a discrete 24LC02B Serial EEPROM to provide the same functions as the MTA85XXX'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 kit. 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.
Description of Contents

1. PICSEE PDIP and SOIC to PIC16C54 or 24LC01B Programming Adapter Sockets
2. Header Interface for PICMASTER-16D and PICPROBE-16D
3. Serial EEPROM Example Software Disk
4. MTA81010 Product Samples
5. 8- and 18-Pin Programming Adapter Plugs
6. Complete Systems Documentation
FIGURE 2: PICSEE START-81A DEVELOPMENT KIT

Description of Contents

1. PICSEE PDIP and SOIC to PIC16C54 or 24LC01B Programming Adapter Sockets
2. Header Interface for PICMASTER-16D and PICPROBE-16D
3. Serial EEPROM Example Software Disk
4. MTA81010/PIC16CXX Product Samples
5. PIC16CXX Device Programmer Board
6. PIC16CXX Assembler, Simulator and Host Software
7. 8- and 18-Pin Programming Adapter plug
8. Power Supply
9. RS-232 Cable
10. Complete Systems Documentation
Description of Contents

1. 20-Lead SSOP Programming Adapter Socket
2. PICMASTER Adapter Socket
3. Serial EEPROM Communications Software Disk
4. Product Samples
5. 8- and 18-pin Programming Adapter Plugs
6. Complete Systems Documentation
Description of Contents

1. 20-lead SSOP Programming Adapter Socket
2. PICMASTER Adapter Socket
3. Serial EEPROM Communications Software Disk
4. Product Samples
5. PIC16CXX Device Programmer Board
6. PIC16/17 Assembler, Simulator, and Host Software
7. 8- and 18-pin Programming Adapter Plugs
8. Power Supply
9. RS-232 Cable
10. Complete Systems Documentation
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-81A FOR MTA81010</td>
</tr>
<tr>
<td>DV813001</td>
<td>PICSEESTART-81A for MTA81010</td>
</tr>
<tr>
<td>AC814003</td>
<td>PDIP PROGRAMMING SOCKET</td>
</tr>
<tr>
<td></td>
<td>SOIC PROGRAMMING SOCKET</td>
</tr>
<tr>
<td>AC852001</td>
<td>PICSEEKIT-85A FOR MTA85XXX</td>
</tr>
<tr>
<td>AC854001</td>
<td>20-LEAD SSOP PROGRAMMING ADAPTER SOCKET AND ADAPTER PLUGS ONLY</td>
</tr>
<tr>
<td>DV853001</td>
<td>PICSEESTART-85A FOR MTA85XXX</td>
</tr>
</tbody>
</table>
SYSTEM FEATURES

EPROM Programmer System:
- Operates with a PC-compatible host system.
- READS, PROGRAMS, VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.
- Universal power supply
- RS-232 interface cable

Macro Assembler:
- Provides translation of Assembler source code to object code for all PIC16CXX microcontroller product family.
- For PC-compatible systems running the MS-DOS® operating system.
- 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 and INHX8M.

Simulator:
- Instruction-level Simulator of the PIC16CXX microcontroller product family.
- For PC-compatible systems running the MS-DOS 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-16B1 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-16B1 system operates on any PC compatible machine running the MS-DOS/PC-DOS operating system.

Provided in the System is an MS-DOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPASM).

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, PIC16C58A, PIC16C61, PIC16C62X, 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-16B1 Development Programmer:

The Microchip device programmer system included in the PICSTART-16B1 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 MPASM 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 PIC16/17 assembler code.

The simulator is meant for use with smaller projects not requiring precise, more extensive development equipment. 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 the pricing or delivery, please use the listed part number, and refer to the listed sales offices.

PART NUMBER DESCRIPTION
DV163003 PICSTART-16B1 DEVELOPMENT SYSTEM
SYSTEM FEATURES

EPROM Programmer System:
- Operates with a PC-compatible host system.
- READS, PROGRAMS, VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.
- Universal power supply
- RS-232 interface cable

Macro Assembler:
- Provides translation of Assembler source code to object code for all PIC16CXX microcontroller product family.
- Macro-Assembly capability.
- For PC-compatible systems running the MS-DOS® operating system.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
- Output formats: INHXS and INHXM.

Simulator:
- Instruction-level Simulator of the PIC16/17 microcontroller product family.
- For PC-compatible systems running the MS-DOS 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 the supported PIC16CXX parts for initial prototype use.
SYSTEM DESCRIPTION

The PICSTART-16C 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-16C system operates on any PC compatible machine running the MS-DOS/PC-DOS operating system.

Provided in the System is an MS-DOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPASM).

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 PIC16C64, PIC16C65, PIC16C73, or the PIC16C74.

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-16C Development Programmer:

The Microchip device programmer system included in the PICSTART-16C 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 MPASM Assembler program are directly loadable into programmer memory.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part number, and refer to the listed sales offices.

<table>
<thead>
<tr>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DV163002</td>
<td>PICSTART-16C DEVELOPMENT SYSTEM</td>
</tr>
</tbody>
</table>

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16/17 assembler code.

The simulator is meant for use with smaller projects not requiring precise more extensive development equipment. 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.
PRODUCT INFORMATION

The PICDEM-1 is a simple board which demonstrates the capabilities of several Microchip microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58), PIC16C62X, PIC16C61, PIC16C62X, PIC16C71, PIC16C84, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. The users can program the samples (one each of PIC17C42, PIC16C71 and PIC16C55) provided with the PICDEM-1, on a PRO MATE™ or PICSTART™ programmer and easily debug/test the sample code, or the user can connect the PICDEM-1 with the PICMASTER™ emulator and download the sample code to the emulator and debug/test the code. Additionally, a generous 200-hole prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s).

FEATURES:

Hardware:
- 40-pin, 28-pin and 18-pin Precision sockets for all supported microcontrollers.
- On board +5V regulator and filter rectifier for direct input from 9V AC/DC wall adapter.
- RS-232 socket and associated hardware for direct connection to RS-232 interface.
- 5K pot to simulate analog input for PIC16C71.
- Three push button Key for external stimulus and RESET.
- Eight bright LEDs connect to PORTB, help in displaying 8-bit binary values on PORTB.
- Socket for "canned" crystal Oscillator.
- Unpopulated holes provided for Xtal connection
- Jumper to disconnect on board RC Oscillator.
- 200-hole prototype area for user's hardware.
Software:
- Program for PIC16C71 to demonstrate on-chip A/D features.
- Program for PIC16C84 to demonstrate on-chip EEPROM.
- Program for PIC17C42 to demonstrate on-chip USART.
- Program for PIC16C5X to demonstrate key input capability.
- All demo programs supplied on 3.5" disk,
- Additional programs available on Microchip's BBS.

DOCUMENTATION
- Schematics for the entire circuit.

SAMPLES
Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:
- PIC17C42
- PIC16C71
- PIC16C55

SALES AND SUPPORT
To order or to obtain information, e.g., on the 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>
</table>
PRODUCT INFORMATION

The PICDEM-2 is a simple board which demonstrates the capabilities of several Microchip microcontrollers, including PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C73 and the PIC16C74. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PROMATE™ or PICSTART™ programmer and easily debug and test the sample code. The PICDEM-2 is also usable with the PICMASTER™ emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, a generous prototype area is available for user hardware.

FEATURES:

Hardware:
- 40- and 28-pin DIP sockets
- On board +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery.
- RS-232C socket and associated hardware for direct connection to RS-232C interface.
- 5K pot for analog inputs for the PIC16C73/74
- Three push button keys for external stimulus and RESET.
- Eight bright LEDs connected to PORTB for displaying 8-bit binary values.
- Socket for "canned" crystal oscillator.
- Unpopulated holes provided for crystal connection
- 128 x 8 Serial EEPROM.
- LCD module header.
- Keyboard header.
- Unpopulated holes for ACCESS.bus™ connector.
Hardware (continued):
- Jacks for connection of 9V battery.
- Jumper to disconnect on-board RC oscillator.
- Prototype area for user hardware.

Software:
- Program for PIC16C74 to demonstrate on-chip A/D feature.
- Program for PIC16C64 to demonstrate I^C™ Serial EEPROM usage.
- All demo programs supplied on 3.5" disk.
- Additional programs available on Microchip’s BBS.

DOCUMENTATION:
- Full schematics.

Samples:
Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:
- PIC16C64
- PIC16C74

SALES AND SUPPORT
To order or to obtain information, e.g., on the 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>DM163002</td>
<td>Low-cost Demonstration Board for PIC16C64, PIC16C65, PIC16C73 AND PIC17C74</td>
</tr>
</tbody>
</table>
fuzzyTECH-MP FOR MICROCHIP
PIC16/17

This product brief describes the technical aspects of the fuzzyTECH-MP Fuzzy Logic Development System for PIC16/17 microcontrollers developed by INFORM Software Corporation specifically for Microchip.

The fuzzyTECH-MP Development System comes in two versions. The first, the Explorer, contains everything you need to gain a comprehensive working knowledge about fuzzy-logic system design. It is easy-to-use, all graphic editors and tools guide you step-by-step through the development phases of fuzzy systems. The Explorer supports two input variables and one output variable.

The full-featured fuzzyTECH-MP Edition offers all of the capabilities of the Explorer, plus it has the additional flexibility of eight input variables and four output variables for designing more complex systems. The full features are enabled with a hardware key lock attached to the parallel port of the PC.

Included in both versions is fuzzyLAB™, a fully functional demonstration board, to give customers hand-on experience with fuzzy logic systems implementation. fuzzyLAB is a simple heating thermostat consisting of a PWM-controlled resistor configured to heat a thermistor to a preset temperature. Using the two fuzzy algorithms provided, a designer can set a target temperature and observe the thermostat response to the set point.

Both systems generate assembly code compatible with the MPASM, Microchip’s Universal Assembler, that can be integrated into your application. Examining this code provides you with further insights into the fabrics of fuzzy logic systems.
fuzzyTECH-MP System Requirements

fuzzyTECH-MP will run on any IBM PC® (386 or higher) or compatible computer, running DOS 4.1 or later, and Microsoft® Windows® 3.0 or later. Because fuzzyTECH-MP makes extensive use of graphics, a color graphic monitor (VGA) is required, and higher resolutions of 800 x 600 or 1024 x 768 are recommended.

What is Fuzzy Logic?

Fuzzy logic is a technology that enhances mode-based system designs using both intuition and engineering heuristics. Fuzzy logic uses elements of everyday language to represent desired system behavior, thus circumventing the need for rigorous mathematical modeling.

It is an efficient way of designing, optimizing and maintaining highly complex systems transparently.

Fuzzy Logic Applications

Fuzzy logic finds its home in unique applications:

• When no adequate mathematical model for a given problem is readily apparent.
• When non-linearities, time constraints or multiple parameters exist.
• When engineering know-how about the given problem is available or can be acquired during the design process.

The fuzzyTECH-MP Implementation

fuzzyTECH-MP provides the following standard features:

• Windows Compatible with full graphical user interface
• 8-Input variables (2 for the Explorer version)
• 4-Output variables (1 for the Explorer version)
• 8-Bit resolution on input and output variables
• 16-Bit computation resolution for the PIC16CXX and PIC17CXX microcontrollers
• No theoretical limit on rules, antecedents and linguistic conjunctions (chip limitations will place a practical limit on these)
• MAX-MIN and MAX-DOT inference methods
• CoM and MoM defuzzification methods
• MPASM Compatible
• PICMASTER™ Compatible

fuzzyTECH-MP is available directly from Microchip Technology and its authorized distributors. Contact your local sales office for more information.

SALES AND SUPPORT

To order or to obtain information, e.g., on the 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>DV005001</td>
<td>fuzzyTECH-MP EXPLORER</td>
</tr>
<tr>
<td>DV005002</td>
<td>fuzzyTECH-MP EDITION</td>
</tr>
</tbody>
</table>
This product brief describes the technical aspects of the PIC16/17 Assembler. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC16C5X, 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/AT® or compatible computer running DOS 5.0 or later.

MPASM ASSEMBLER FEATURES

MPASM supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions 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.
• Output formats: INHX6S, INHX8M, INHX32 and relocatable objects.
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 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.

SALES AND SUPPORT

To order or to obtain information, e.g., on the 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>SW165002</td>
<td>MPSIM/MPASM Simulator and Assembler Software and Documentation</td>
</tr>
</tbody>
</table>
MPSIM is a discrete event simulator software application designed to imitate operation of the PIC16/17 microcontrollers. It allows the user to debug software that will use any of these microcontrollers.

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/AT® or compatible computer running DOS version 5.0 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. Many applications can be developed by using a simulator program alone.

The PIC16CXX and PIC17CXX families support various peripherals and interrupts. MPSIM generally simulates interrupts and most peripheral functions. However, certain peripheral functions are not supported (such as A/D conversion or serial I/O).

The MPSIM Simulator has the following features to assist in the debugging of software / firmware for the user:
**MPSIM Simulator**

**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 works with Microchip's MPASM Universal Assembler.

---

**SALES AND SUPPORT**

To order or to obtain information (e.g., on the 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>SW165002</td>
<td>MPSIM/MPASM Simulator and Assembler Software and Documentation</td>
</tr>
</tbody>
</table>
MP-C C COMPILER FOR PIC16/17

This product brief describes the technical aspects of the MP-C Code Development System for PIC16/17 micro-controllers developed by Byte Craft Limited.

The MP-C Code Development System is a complete 'C' compiler and integrated development environment for Microchip's PIC16/17 family of microcontrollers. The compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the PICMASTER™ Universal Emulator memory display (emulator software versions 1.13 and later).

MP-C is fast and efficient. You can quickly produce stand-alone single-chip microcontroller applications. These, taken with its other advantages make the Byte Craft MP-C Code Development System the first choice in intelligent compiler technology.

MP-C Requirements

The compiler will run on any IBM PC, PC/XT®, PC/AT® or compatible computer, running DOS 5.0 or later.

MP-C Code Development System Features

MP-C supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and 16-bit PIC17CXX cores. It is a rule-based compiler with expert systems tailored to each of these platforms for optimal efficiency.

The compiler generates executable code directly from the compile process. There is no need for an extra step to assemble code generated by the compiler.
MP-C

MP-C has the following features to assist in developing PIC16/17 software for specific user applications:

- Provides Object, Listing, Symbol and special files required for debugging with other Microchip Development systems.
- Supports interrupt routines
- Checks source against target hardware definitions
- Generates efficient, tight object code
- Includes a linker and built-in macro assembler
- 'C' enhancements specific to the PIC16/17 families' instruction sets.
- Output formats: INHX8S, INHX8M, and INHX32.

MP-C Microprocessor Specific Extensions

The MP-C Code Development System includes common 'C' enhancements such as ROM arrays, binary constants and case statements together with functions specific to the PIC16/17 architecture.

- **Binary Constants** of the form 0b0101110 which are logical extensions to the conventional 0x1a3b style of hexadecimal constants. You may also use 0B as leading characters.
- **Case Statements** are supported well by the PIC16/17 instruction set and the compiler provides a superset of the standard 'C' case statement. For example, case 4,5:, case '0'.'9', and complex case statements are allowed.
- **Processor Specific Functions** that are specific to the PIC16/17 family. For example NOP() and SLEEP() produce the equivalent PIC16/17 instruction.
- **"At" or @ Extension** allows you to fix a variable to a specific address in memory, for example: int N @0x0C.

SALES AND SUPPORT

The MP-C system is supported by Microchip via the Microchip technical support hotline.

The MP-C Code Development System is supplied directly by Byte Craft Limited of Waterloo, Ontario, Canada.

If you have any questions please contact your regional Microchip FAE or Microchip technical support personnel at (602) 786-7627.
FEATURES

- IBM® PC compatibility
- Windows® 3.1 or DOS 3.1 compatibility
- Automatic or manual recalculation
- Real-time update of data
- Full-screen or windowed graphical view
- Hypertext on-screen help
- Key or slide-bar entry of parameters
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk Endurance Tutorial

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows 3.1
- 1MB memory
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/24AAXX/85CXX
- Microchip 3-wire 93CXX/93LCXX/93AAXX Series
- Microchip 4-wire 59C11
DESCRIPTION

Microchip's revolutionary Total Endurance Model provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. This advanced software model (with a very friendly user interface) eliminates time and guesswork from Serial EEPROM-based designs by accurately predicting the device's performance and reliability within a user-defined application environment. Design trade-off analysis which formerly consumed days or weeks can now be performed in minutes...with a level of accuracy that delivers a truly robust design.

Users may input the following application parameters:

- Serial EEPROM device type
- Bytes to be written per cycle
- Cycling mode - byte or page
- Data pattern type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application lifetime or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level vs. number of cycles. The model is available in both DOS and Windows versions.

BACKGROUND

Microchip's research into the Erase/Write endurance of Serial EEPROMs has resulted in the conclusion that endurance depends upon three primary effects: the physical properties of the EEPROM cell, the internal error-correction technology employed, and the application environment. EEPROM endurance specified as a "typical" value in device data sheets must therefore be evaluated on a case-by-case basis, taking into account the manner in which the device will be used in the application. The Microchip Total Endurance™ software applies the user-defined application parameters to a complex mathematical model in order to emulate the EEPROM's performance and reliability in the system.

USING THE MODEL

The user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

ACCURACY OF THE MODEL

The accuracy of the Microchip Total Endurance model has been verified against test data to within ten percent of the actual values. However, Microchip makes no warranty as to its accuracy or applicability of the information to any given application. It is intended to be used as a guide to aid designers of Serial EEPROM-based systems in performing trade-off analysis and developing robust and reliable designs.

Order Information:

<table>
<thead>
<tr>
<th>Description</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Endurance Software Disk</td>
<td>SW242001</td>
</tr>
</tbody>
</table>
INTRODUCTION

The MTA11200 TrueGauge Intelligent Battery Management IC is supported by a user friendly tool for system development. The DV114001 operates under Microsoft Windows®. This development tool enables for management of all phases of product development, including inception, debugging and maintenance. System design verification can be accomplished before a hardware prototype needs to be built, thus reducing time and cost. The user interface provides a graphically-oriented development environment. The data logging feature saves measured data into a file that can be imported to Excel®. Battery parameters can be changed, down-loaded to the TrueGauge, then battery performance can be evaluated.

SUMMARY OF FEATURES

The TrueGauge development tool is a tool for system development under Windows. The development tool kit contains the following:

- NiCd battery with TrueGauge module
- NiMH battery with TrueGauge module
- Stand-alone TrueGauge module
- Charger/discharger Interface Board
- Universal Power Supply with power cord
- PC Interface Cable with DB9-DB25 converter
- Design/Verification software on a 3.5" diskette, including calibration routines
- MTA11200B and 24LC01B product samples
- MTA11200B Datasheet
- TrueGauge Development Tool User's Guide
- Version 2.5 supports MTA11200B

FIGURE 1: TRUEGAUGE DEVELOPMENT TOOL KIT
Battery status information is plotted on the computer allowing for real-time monitoring of battery management parameters (Figure 2).

FIGURE 2: CAPACITY/VOLTAGE OVER TIME

Capacity in Percent [red, solid] and Voltage [black, dashed]
Parameters can be changed easily and downloaded to the TrueGauge module (Figure 3 and Figure 4).

**FIGURE 3: CONFIGURATION CONTROL PANEL**

<table>
<thead>
<tr>
<th>MTA11200 Configuration Data</th>
<th>Battery Warning Levels (% of Full Charge)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev: 001</td>
<td>OK</td>
</tr>
<tr>
<td>Fast Charge Termination</td>
<td>Low: 5</td>
</tr>
<tr>
<td></td>
<td>Critical: 3</td>
</tr>
<tr>
<td></td>
<td>Shutdown: 1</td>
</tr>
<tr>
<td>Delta Temp Delta Time</td>
<td>Volt (volts): 12.032</td>
</tr>
<tr>
<td>0.50 degrees C / min</td>
<td>Time (min): 178</td>
</tr>
<tr>
<td>Nominal Battery Capacity</td>
<td>Charge Turn On Threshold (% of Charge Capacity)</td>
</tr>
<tr>
<td>1800 milliamp-hours</td>
<td>Max Fast: 45</td>
</tr>
<tr>
<td></td>
<td>Min Fast: 10</td>
</tr>
<tr>
<td></td>
<td>Max Maint: 50</td>
</tr>
<tr>
<td></td>
<td>Min Maint: 5</td>
</tr>
<tr>
<td>Measured Battery Capacity</td>
<td>End of Discharge Voltage (% of Charge Capacity)</td>
</tr>
<tr>
<td>1800 milliamp-hours</td>
<td>% Charge: 95</td>
</tr>
<tr>
<td></td>
<td>Volts: 05.632</td>
</tr>
</tbody>
</table>
FIGURE 4: ADVANCED CONFIGURATION DISPLAY

### Advanced Configuration

<table>
<thead>
<tr>
<th></th>
<th>Interval</th>
<th>LED Mode</th>
<th>Broadcast Data Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPINTRVL &amp; REP Mode: Broadcast Interval</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interval</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD 10% Levels</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete 20% Levels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Broadcast</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Select</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Byte</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sixteen Byte</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Counts

- TOERRS: Charge Time: 0
- LOTERRS: Low Temp.: 0
- HITERRS: High Temp.: 0
- HIVERRS: High Volt.: 0
- CCCR: Charge Cycle: 16
- TCC: Charge Cycle: 0

#### USER: User Storage

- User Value # 1: 0
- User Value # 2: 0
- User Value # 3: 0
- User Value # 4: 0

FIGURE 5: REV. B CONTROL OPTIONS (BOPT)

### MTA11200 Rev. B Control Options

#### Rev B Control Options (BOPT)

- Fast Charge Termination
  - **Does not affect % capacity gauge**
  - **Forces % Capacity gauge to 100%**

- Primary Fast Charge Termination Method
  - **Stop fast charge when gauge at 100% or dT/dt detect**
  - **Stop fast charge on dT/dt detect**

- Negative Delta Voltage (-dV) control panel
  - **Use current monitor to prevent false -dV detects**
  - **Use voltage monitor to prevent false -dV detects**

- Wake up from Standby if current flow is detected
  - **Enable wake up on current flow detect**
  - **Disable wake up on current flow detect**

- EODV pin function
  - **EODV pin indicates >=0% charge, P0#, (like P20#, P40#, P60#, P80# pins)**
  - **EODV pin indicates battery voltage below EODV limit**

#### Wake up current threshold

- **IWAKE: (mA)**: 0

#### Current monitor -dV detector threshold

- **DIMON: (mA)**: 0

#### Standby current deadzone threshold

- **STBYIDZ: (mA)**: 0
**FIGURE 6: CHARGE EFFICIENCY vs STATE OF CHARGE COMPENSATION**

Charger Discharge Compensation Constants

<table>
<thead>
<tr>
<th>Charge Efficiency vs. State of Charge Compensation</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.6%</td>
</tr>
<tr>
<td>Chg. Eff.</td>
</tr>
<tr>
<td>0%</td>
</tr>
</tbody>
</table>

CESC(15)=65, Retained charge = 25.39% of input charge.

(00)=150  (04)=247  (08)=247  (12)=236
(01)=213  (05)=247  (09)=247  (13)=197
(02)=238  (06)=247  (10)=247  (14)=125
(03)=245  (07)=247  (11)=247  (15)=65

**FIGURE 7: CHARGE EFFICIENCY vs TEMPERATURE COMPENSATION**

Charger Discharge Compensation Constants

<table>
<thead>
<tr>
<th>Charge Efficiency vs. Temperature Compensation</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.6%</td>
</tr>
<tr>
<td>Chg. Eff.</td>
</tr>
<tr>
<td>0%</td>
</tr>
</tbody>
</table>

CESC(15)=159, Retained charge = 62.11% of input charge.

(00)=249  (04)=249  (08)=230  (12)=190
(01)=249  (05)=249  (09)=220  (13)=180
(02)=249  (06)=249  (10)=210  (14)=170
(03)=249  (07)=240  (11)=199  (15)=159
System design verification can be accomplished before hardware implementation (Figure 8, Figure 9, and Figure 10).

FIGURE 8: TRUEGAUGE VOLTAGE AND CAPACITY vs. TIME

FIGURE 9: TEMPERATURE vs. TIME
A data logging feature saves measured data into a file that can be imported into Microsoft Excel (Figure 11)

**FIGURE 11: EXAMPLE OF DATA LOG FILE**

<table>
<thead>
<tr>
<th>Time</th>
<th>Remaining Capacity (%)</th>
<th>Voltage</th>
<th>Temperature (C)</th>
<th>Current (mA)</th>
<th>Total Capacity (mA-Hr)</th>
<th>Flag_byte</th>
<th>Error_byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>22:39:13</td>
<td>9</td>
<td>8.645</td>
<td>23.88281</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:39:15</td>
<td>9</td>
<td>8.645</td>
<td>23.88672</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:39:23</td>
<td>9</td>
<td>8.646</td>
<td>23.90234</td>
<td>648</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:39:34</td>
<td>9</td>
<td>8.646</td>
<td>23.92188</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:39:44</td>
<td>9</td>
<td>8.647</td>
<td>23.9375</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:39:55</td>
<td>9</td>
<td>8.647</td>
<td>23.95313</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:40:04</td>
<td>10</td>
<td>8.647</td>
<td>23.96484</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:40:14</td>
<td>10</td>
<td>8.648</td>
<td>23.96875</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
<tr>
<td>22:40:25</td>
<td>10</td>
<td>8.648</td>
<td>23.98047</td>
<td>647</td>
<td>1200</td>
<td>BB</td>
<td>0</td>
</tr>
</tbody>
</table>
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 office.

<table>
<thead>
<tr>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DV114001</td>
<td>TRUEGAUGE DEVELOPMENT TOOL</td>
</tr>
<tr>
<td>AC113001</td>
<td>TRUEGAUGE UPGRADE KIT</td>
</tr>
<tr>
<td></td>
<td>(3 MODULES and SOFTWARE DISK)</td>
</tr>
</tbody>
</table>
SECTION 7
SALES AND SERVICE LOCATIONS

Factory Representatives
Africa .......................................................... 7-1
Asia/Pacific ...................................................... 7-1
Canada .......................................................... 7-1
Europe ........................................................... 7-2
Mexico ............................................................. 7-2
South America ............................................... 7-3
United States ................................................ 7-4

Distributors/Service
Africa .......................................................... 7-7
Asia/Pacific ...................................................... 7-7
Canada .......................................................... 7-8
Europe ........................................................... 7-10
Mexico ............................................................. 7-12
South America ............................................... 7-12
United States ................................................ 7-13

Factory Sales .................................................. 7-19
<table>
<thead>
<tr>
<th>Factory Representatives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AFRICA</strong></td>
</tr>
<tr>
<td>Tempe Technologies Pty Ltd</td>
</tr>
<tr>
<td>P.O. Box 2480</td>
</tr>
<tr>
<td>Edenvale 1610</td>
</tr>
<tr>
<td>South Africa</td>
</tr>
<tr>
<td>Tel: 27/11-3341427</td>
</tr>
<tr>
<td>Fax: 27/11-3340103</td>
</tr>
<tr>
<td><strong>CANADA</strong></td>
</tr>
<tr>
<td><strong>Alberta</strong></td>
</tr>
<tr>
<td>Enerlec Sales Ltd.</td>
</tr>
<tr>
<td>#103 155 Glendeer Circle S.E.</td>
</tr>
<tr>
<td>Calgary, Alberta T2H 2S8 Canada</td>
</tr>
<tr>
<td>Tel: 403-777-1550</td>
</tr>
<tr>
<td>Fax: 403-777-1553</td>
</tr>
<tr>
<td><strong>British Columbia</strong></td>
</tr>
<tr>
<td>Enerlec Sales Ltd.</td>
</tr>
<tr>
<td>#7 3671 Viking Way</td>
</tr>
<tr>
<td>Richmond, B.C. V6V 1W1 Canada</td>
</tr>
<tr>
<td>Tel: 604-273-0882</td>
</tr>
<tr>
<td>Fax: 604-273-0884</td>
</tr>
<tr>
<td><strong>Manitoba</strong></td>
</tr>
<tr>
<td>Enerlec Sales Ltd.</td>
</tr>
<tr>
<td>#7 3671 Viking Way</td>
</tr>
<tr>
<td>Richmond, B.C. V6V 1W1 Canada</td>
</tr>
<tr>
<td>Tel: 604-273-0882</td>
</tr>
<tr>
<td>Fax: 604-273-0884</td>
</tr>
<tr>
<td><strong>Montreal</strong></td>
</tr>
<tr>
<td>Electronic Sales Professionals</td>
</tr>
<tr>
<td>10690 Peloquin Street</td>
</tr>
<tr>
<td>Suite 210</td>
</tr>
<tr>
<td>Montreal, Quebec, Canada H2C 2K3</td>
</tr>
<tr>
<td>Tel: 514-388-6596</td>
</tr>
<tr>
<td>Fax: 514-388-8402</td>
</tr>
<tr>
<td><strong>Ottawa</strong></td>
</tr>
<tr>
<td>Electronic Sales Professionals</td>
</tr>
<tr>
<td>215 Stafford Road</td>
</tr>
<tr>
<td>Unit 104</td>
</tr>
<tr>
<td>Nepean, Ontario, Canada K2H 9C1</td>
</tr>
<tr>
<td>Tel: 613-828-6881</td>
</tr>
<tr>
<td>Fax: 613-828-5725</td>
</tr>
<tr>
<td><strong>Saskatchewan</strong></td>
</tr>
<tr>
<td>Enerlec Sales Ltd.</td>
</tr>
<tr>
<td>#7 3671 Viking Way</td>
</tr>
<tr>
<td>Richmond, B.C., Canada V6V 1W1</td>
</tr>
<tr>
<td>Tel: 604-273-0882</td>
</tr>
<tr>
<td>Fax: 604-273-0884</td>
</tr>
<tr>
<td><strong>Toronto</strong></td>
</tr>
<tr>
<td>Electronic Sales Professionals</td>
</tr>
<tr>
<td>137 Main Street</td>
</tr>
<tr>
<td>Suite 204</td>
</tr>
<tr>
<td>Markham, Ontario, Canada L3P 1Y2</td>
</tr>
<tr>
<td>Tel: 905-294-3520</td>
</tr>
<tr>
<td>Fax: 905-294-3806</td>
</tr>
<tr>
<td>Country</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Finland</td>
</tr>
<tr>
<td>Germany</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Ireland</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Israel</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Netherlands</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Norway</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Scotland</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Sweden</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Turkey</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
## SOUTH AMERICA

### Argentina
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Bolivia
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Brazil
Aplicacoes Eletronicas  
Artimar Ltda.  
Rua Marques de Itu N° 70-10And.  
CEP 01223  
Sao Paulo, Brazil  
Tel: 55-11-231-0277  
Fax: 55-11-255-0511

### Caribbean
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Chile
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Columbia
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Ecuador
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Paraguay
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Peru
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Uruguay
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763

### Venezuela
Ibars Electronics Corporation  
10020 N.W. 6th Ct.  
Pembroke Pines, Florida 33024  
USA  
Tel: 305-430-3740  
Fax: 305-430-3763
# Factory Representatives

## UNITED STATES

### Alaska
- **Trinity Technologies**
  - Address: 1261 Oakmead Parkway
  - City: Sunnyvale, CA 94086
  - Tel: 408-733-9000
  - Fax: 408-733-9970

### Alabama
- **Concord Component Reps Inc.**
  - Address: 190 Lime Quarry Road, Suite 102
  - City: Madison, AL 35758
  - Tel: 205-772-8883
  - Fax: 205-772-8262

### Arizona
- **Western High Tech Marketing, Inc.**
  - Address: 9414 E. San Salvador, Suite 206
  - City: Scottsdale, AZ 85258
  - Tel: 602-860-2702
  - Fax: 602-860-2712

### Arkansas
- **CompTech Sales, Inc.**
  - Address: 2401 Gateway Drive, Suite 114
  - City: Irving, TX 75063
  - Tel: 214-751-1181
  - Fax: 214-550-8113

### California
- **Costa Mesa**
  - **Competitive Technology, Inc.**
    - Address: 200 Baker Street, Suite 101
    - City: Costa Mesa, CA 92626
    - Tel: 714-540-5501
    - Fax: 714-540-5171

- **Escondido**
  - **Eagle Technical Sales Assoc. Inc.**
    - Address: 1900 Sunset Drive, Suite A
    - City: Escondido, CA 92025
    - Tel: 619-743-6550
    - Tel: 619-743-6585

- **Sunnyvale**
  - **Trinity Technologies**
    - Address: 1261 Oakmead Parkway
    - City: Sunnyvale, CA 94086
    - Tel: 408-733-9000
    - Fax: 408-733-9970

### Colorado
- **Western Region Marketing**
  - Address: 9176 Marshall Place
  - City: Westminster, CO 80030
  - Tel: 303-428-8088
  - Fax: 303-426-8585

### Connecticut
- **Gilford**
  - **S-J Associates, Inc.**
    - Address: 10 Copper Ridge Circle
    - City: Gilford, CT 06437
    - Tel: 203-458-7558
    - Fax: 203-458-1181

### Delaware
- **S-J Mid-Atlantic, Inc.**
  - Address: 131-D Gaither Drive
  - City: Mt. Laurel, NJ 08054
  - Tel: 609-866-1234
  - Fax: 609-866-8627

### District of Columbia
- **S-J Chesapeake**
  - Address: 900 S. Washington Street, Suite 307
  - City: Falls Church, VA 22046
  - Tel: 703-533-2233
  - Fax: 703-533-2236

### Florida
- **Altamonte Springs**
  - **Electramark Florida, Inc.**
    - Address: 401 Whooping Loop, Suite 1565
    - City: Altamonte Springs, FL 32701
    - Tel: 407-830-0844
    - Fax: 407-830-0847

- **Boca Raton**
  - **Electramark Florida, Inc.**
    - Address: 621 NW 53rd Street, Suite 240
    - City: Boca Raton, FL 33487
    - Tel: 407-998-8820
    - Fax: 407-998-8821

### Georgia
- **Concord Component Reps Inc.**
  - Address: 6825 Jimmy Carter Boulevard
  - City: Norcross, GA 30071
  - Tel: 404-416-9597
  - Fax: 404-441-0790

### Hawaii
- **Trinity Technologies**
  - Address: 1261 Oakmead Parkway
  - City: Sunnyvale, CA 94086
  - Tel: 408-733-9000
  - Fax: 408-733-9970

### Idaho
- **Trinity Technologies**
  - Address: 1261 Oakmead Parkway
  - City: Sunnyvale, CA 94086
  - Tel: 408-733-9000
  - Fax: 408-733-9970

### Illinois
- **Janus Incorporated**
  - Address: 650 E. Devon Avenue
  - City: Itasca, IL 60143
  - Tel: 708-250-9650
  - Fax: 708-250-8761

### Indiana
- **Fort Wayne**
  - **Electro Reps, Inc.**
    - Address: 125 Airport North Office Park
    - City: Fort Wayne, IN 46825
    - Tel: 219-489-8205
    - Fax: 219-489-8408

### Indianapolis
- **Electro Reps, Inc.**
  - Address: 7240 Shadeland Station, Suite 275
  - City: Indianapolis, IN 46256
  - Tel: 317-942-7202
  - Fax: 317-941-0230

### Iowa
- **Spectrum Sales**
  - Address: 1364 Elmhurst Drive NE
  - City: Cedar Rapids, IA 52402
  - Tel: 319-366-0575
  - Fax: 319-366-0635

### Kansas
- **Spectrum Sales**
  - Address: 5382 W. 95th Street
  - City: Prairie Village, KS 66207
  - Tel: 913-648-6811
  - Fax: 913-648-6823
### Factory Representatives

<table>
<thead>
<tr>
<th>State</th>
<th>Northern</th>
<th>Southern</th>
</tr>
</thead>
</table>
| Kentucky      | Kentucky Technology Marketing Corporation  
                7775 Cooper Road  
                Suite 3  
                Cincinnati, OH 45242  
                Tel: 513-984-6720  
                Fax: 513-984-6874 |
| Minnesota     | Comprehensive Technical Sales Inc.  
                6513 City West Parkway  
                Eden Prairie, MN 55344  
                Tel: 612-941-7181  
                Fax: 612-941-4322 |
| Missouri      | Concord Component Reps Inc.  
                190 Lime Quarry Road, Suite 102  
                Madison, AL 35758  
                Tel: 205-772-8883  
                Fax: 205-772-8262 |
| Montana       | Spectrum Sales  
                5494 Brown Road, Suite 124  
                St. Louis, MO 63042  
                Tel: 314-921-1313  
                Fax: 314-921-0701 |
| Nebraska      | Spectrum Sales  
                5382 W. 95th Street  
                Prairie Village, KS 66207  
                Tel: 913-648-6811  
                Fax: 913-648-6823 |
| Nevada        | Trinity Technologies  
                1261 Oakmead Parkway  
                Sunnyvale, CA 94086  
                Tel: 408-733-9000  
                Fax: 408-733-9970 |
| New Jersey    | Parallax  
                734 Walt Whitman Road  
                Melville, NY 11747  
                Tel: 516-351-1000  
                Fax: 516-351-1606 |
| New Mexico    | Western High Tech Marketing, Inc.  
                9414 E. San Salvador, Suite 206  
                Scottsdale, AZ 85258  
                Tel: 602-860-2702  
                Fax: 602-860-2712 |
| New York      | Parallax  
                734 Walt Whitman Road  
                Melville, NY 11747  
                Tel: 516-351-1000  
                Fax: 516-351-1606 |
| North Carolina| Zucker Associates  
                4070 Barrett Drive  
                Raleigh, NC 27609  
                Tel: 919-782-8433  
                Fax: 919-782-8476 |
| North Dakota  | Comprehensive Technical Sales Inc.  
                6513 City West Parkway  
                Eden Prairie, MN 55344  
                Tel: 612-941-7181  
                Fax: 612-941-4322 |
| North Dakota  | Western High Tech Marketing, Inc.  
                9414 E. San Salvador, Suite 206  
                Scottsdale, AZ 85258  
                Tel: 602-860-2702  
                Fax: 602-860-2712 |
| North Dakota  | Western High Tech Marketing, Inc.  
                9414 E. San Salvador, Suite 206  
                Scottsdale, AZ 85258  
                Tel: 602-860-2702  
                Fax: 602-860-2712 |
| North Dakota  | Western High Tech Marketing, Inc.  
                9414 E. San Salvador, Suite 206  
                Scottsdale, AZ 85258  
                Tel: 602-860-2702  
                Fax: 602-860-2712 |
| North Dakota  | Western High Tech Marketing, Inc.  
                9414 E. San Salvador, Suite 206  
                Scottsdale, AZ 85258  
                Tel: 602-860-2702  
                Fax: 602-860-2712 |
| North Dakota  | Western High Tech Marketing, Inc.  
                9414 E. San Salvador, Suite 206  
                Scottsdale, AZ 85258  
                Tel: 602-860-2702  
                Fax: 602-860-2712 |
<table>
<thead>
<tr>
<th>State</th>
<th>City</th>
<th>Address</th>
<th>Phone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohio</td>
<td>Cincinnati</td>
<td>Technology Marketing Corporation 7775 Cooper Road Suite 3 Cincinnati, OH 45242 Tel: 513-984-6720 Fax: 513-984-6874</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Technology Marketing Corporation One Independence Place 4807 Rockside Road, Suite 360 Independence, OH 44131 Tel: 216-520-0150 Fax: 216-520-0190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oklahoma</td>
<td></td>
<td>CompTech Sales, Inc. 18700 Woodbriar Lane Catoosa, OK 74015 Tel: 918-266-1966 Fax: 918-266-1808</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oregon</td>
<td></td>
<td>Micro Sales 1865 N.W. 169th Pl. #210 Beaverton, OR 97006 Tel: 503-645-2841 Fax: 503-645-3754</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pennsylvania</td>
<td></td>
<td>S-J Mid-Atlantic, Inc. 131-D Gaither Drive Mt. Laurel, NJ 08054 Tel: 609-866-1234 Fax: 609-866-8627</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Western</td>
<td>Technology Marketing Corporation 7775 Cooper Road Suite 3 Cincinnati, OH 45242 Tel: 513-984-6720 Fax: 513-984-6874</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhode Island</td>
<td></td>
<td>S-J New England, Inc. 40 Mall Road Burlington, MA 01803 Tel: 617-272-5552 Fax: 617-272-5515</td>
<td></td>
<td></td>
</tr>
<tr>
<td>South Carolina</td>
<td></td>
<td>Zucker Associates 4070 Barrett Drive Raleigh, NC 27609 Tel: 919-782-8433 Fax: 919-782-8476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>South Dakota</td>
<td></td>
<td>Comprehensive Technical Sales Inc. 6513 City West Parkway Eden Prairie, MN 55344 Tel: 612-941-7181 Fax: 612-941-4322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tennessee</td>
<td>Eastern</td>
<td>Zucker Associates 4070 Barrett Drive Raleigh, NC 27609 Tel: 919-782-8433 Fax: 919-782-8476</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Western</td>
<td>Concord Component Reps Inc. 6825 Jimmy Carter Boulevard Norcross, GA 30071 Tel: 404-416-9597 Fax: 404-441-0790</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Texas</td>
<td>Austin</td>
<td>CompTech Sales, Inc. 1130 Jollyville Road, Suite 200 Austin, TX 78759 Tel: 512-343-0300 Fax: 512-345-2530</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brownsville</td>
<td>CompTech Sales, Inc. 2390 Central Blvd., Suite P Brownsville, TX 78520 Tel: 210-504-9693 Fax: 210-504-9982</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>El Paso</td>
<td>CompTech Sales, Inc. 3120 Wheeling Rd. El Paso, TX 79930 Tel: 915-566-1022 Fax: 915-566-1030</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Houston</td>
<td>CompTech Sales, Inc. 10550 Richmond Avenue Suite 105 Houston, TX 77042 Tel: 713-781-7420 Fax: 713-781-5865</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Irving</td>
<td>CompTech Sales, Inc. 2401 Gateway Drive Suite 114 Irving, TX 75063 Tel: 214-751-1181 Fax: 214-550-8113</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utah</td>
<td></td>
<td>Western Region Marketing 3539 South Main, Suite 210 Salt Lake City, UT 84115 Tel: 801-268-9768 Fax: 801-268-9796</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vermont</td>
<td></td>
<td>S-J New England, Inc. 40 Mall Road Burlington, MA 01803 Tel: 617-272-5552 Fax: 617-272-5515</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virginia</td>
<td></td>
<td>S-J Chesapeake 900 S. Washington Street Suite 307 Falls Church, VA 22046 Tel: 703-533-2233 Fax: 703-533-2236</td>
<td></td>
<td></td>
</tr>
<tr>
<td>West Virginia</td>
<td></td>
<td>S-J Chesapeake 900 S. Washington Street Suite 307 Falls Church, VA 22046 Tel: 703-533-2233 Fax: 703-533-2236</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wisconsin</td>
<td></td>
<td>Janus Incorporated 375 Williamstowe Delafield, WI 53018 Tel: 414-646-5420 Fax: 414-646-2421</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wyoming</td>
<td></td>
<td>Western Region Marketing 9176 Marshall Place Westminster, CO 80030 Tel: 303-428-8088 Fax: 303-426-8585</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AFRICA</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>South Africa</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Pace Electronic Components Ltd.  
Cnr. Vanacht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
Republic of South Africa  
Tel: 27/11-9741211  
Fax: 27/11-9741271 |

| **Asia/Pacific** |
| **Australia** |
| Har-Tee Australia  
205 Middleborough Road
Box Hill, Victoria Australia 3128  
Tel: 61-3-890-0970  
Fax: 61-3-899-5191 |

| **China** |
| Wuhan Liyuan  
107-2 Luo Yu Road
Wuhan 430070  
Peoples Republic of China  
Tel: 86-27-7802986  
Fax: 86-27-7802985 |

| **Hong Kong** |
| Goldenchip Research  
11/F, 1116-7 Kwai on Fty Bldg
103 Tai Lin Pai Road
Kwai Chung, N.T. Hong Kong  
Tel: 852-2426-3968  
Fax: 852-2481-7403 |

| **Japan** (continued) |
| Nippon Precision Device Corp.  
Nichibei Time 24 Bldg.
35 Tansu-Cho, Shinjuku-Ku
Tokyo 162 Japan  
Tel: 81-3-3260-1411  
Fax: 81-3-3260-7100 |

| **New Zealand** |
| Har-Tec NZ Limited  
50 O'Rourke Road
P.O. Box 1205
Penrose, Auckland New Zealand  
Tel: 64-9-525-1096  
Fax: 64-9-525-1097 |
Distributors

**ASIA/PACIFIC (continued)**

**Singapore**
Gates Engineering
1123 Serangoon Road #03-01 UMW Building
Singapore 1232
Tel: 65-299-9937
Fax: 65-299-7636

Excelpoint Systems
12 Tannery Lane #06-01/02
Beam Building
Singapore 1334
Tel: 65-741-8980
Fax: 65-741-8980

**Taiwan, R.O.C.**
Solomon Technology Corp.
5th Floor, No. 293, Sec. 5
Chung Hsiao E. Rd.
Taipei, Taiwan, R.O.C.
Tel: 886-02-788-8989
Fax: 886-02-788-8029

Pinnacle Technologies Co. Ltd.
4F, No. 270, Sec. 3
Nan-Kang Road
Taipei, Taiwan, R.O.C.
Tel: 02 788 4800
Fax: 02 788 5969

**CANADA**

**Vancouver, British Columbia**
Farnell Electronic Services
8525 Baxter Place, Unit 101
Production Court
Burnaby, B.C. V5A 4V7
Canada
Tel: 604-421-6222
Fax: 604-421-0582

Semad Electronics
3700 Gilmore Way, #212
Burnaby, B.C. V5G 4M1
Canada
Tel: 604-451-3444
Fax: 604-451-3445

Pioneer/Zentronics
4455 No. 6 Road
Richmond, B.C. V6V 1P8
Canada
Tel: 604-273-5575
Fax: 604-273-2413

Future Electronics
1695 Boundary Road
Vancouver, B.C. V5K 4X7
Canada
Tel: 604-294-1166
Fax: 604-294-1206

**Calgary, Alberta**
Farnell Electronic Services
3015 - 5th Ave. NE
Suite 210
Calgary, Alberta T2A 6T8
Canada
Tel: 403-273-2780
Fax: 403-273-7458

Future Electronics
2015 32nd Ave. N.E., Unit 1
Calgary, Alberta T2E 6Z3
Canada
Tel: 403-250-5550
Fax: 403-291-7054

Pioneer/Zentronics
#560, 1212-31st Avenue N.E.
Calgary, Alberta T2E 7S8
Canada
Tel: 403-291-1988
Fax: 403-291-0740

Semad Electronics
6815 8th St. N.E.
Suite 175
Calgary, Alberta T2E 7H7
Canada
Tel: 403-252-5664
Fax: 800-565-9779
Edmonton, Alberta
Future Electronics
4606-97th Street
Edmonton, Alberta T6E 5NG
Canada
Tel: 403-438-2858
Fax: 403-434-0812

Pioneer/Zentronics
Plaza 124 #708
10216-124 Street
Edmonton, Alberta T5N 4A3
Canada
Tel: 403-482-3038
Fax: 403-482-1336

London, Ontario
Pioneer/Zentronics
148 York Street, Suite 209
London N6A 1A9
Canada
Tel: 519-672-4666
Fax: 519-672-3528

Toronto, Ontario
Farnell Electronic Services
300 N. Rivermede Road
Concord, Ontario L4K 2Z4
Canada
Tel: 416-798-4884
Fax: 416-798-4889

Semad Electronics
85 Spy Court
Markham, Ontario L3R 4Z4
Canada
Tel: 416-475-3922
Fax: 416-475-4158

Future Electronics
5935 Airport Road, Suite 200
Mississauga, Ontario L4V 1W5
Canada
Tel: 905-612-9200
Fax: 905-612-9185

Pioneer/Zentronics
3415 American Drive
Mississauga, Ontario L4V 1T4
Canada
Tel: 905-405-8300
Fax: 905-405-6423

Ottawa, Ontario
Farnell Electronic Services
39 Robertson Road, Suite 506
Bell Mews
Nepean, Ontario K2H 8R2
Canada
Tel: 613-596-6980
Fax: 613-596-6987

Pioneer/Zentronics
223 Colonnade Road, Suite 112
Nepean, Ontario K2E 7K3
Canada
Tel: 613-226-8840
Fax: 613-226-6352

Future Electronics
1050 Baxter Road
Ottawa, Ontario K2C 3P2
Canada
Tel: 613-820-8313
Fax: 613-820-3271

Semad Electronics
2781 Lancaster
Suite 302
Ottawa, Ontario K1B 1A7
Canada
Tel: 613-526-4866
Fax: 613-523-4372

Montreal, Quebec
Farnell Electronic Services
6600 Trans Canada Highway
Suite 620
Pointe Claire, Quebec H9R 4S2
Canada
Tel: 514-697-8149
Fax: 514-697-1210

Future Electronics/Branch
237 Hymus Boulevard
Pointe Claire, Quebec H9R 5C7
Canada
Tel: 514-694-7710
Fax: 514-695-3707

Semad Electronics
243 Place Frontenac
Pointe Claire, Quebec H9R 427
Canada
Tel: 514-694-0860
Fax: 514-694-0965

Quebec City, Quebec
Future Electronics
1000 Ave. St. Jean Baptiste
Suite 100
Quebec G2E 5G5
Canada
Tel: 418-877-6666
Fax: 418-877-6671

Pioneer/Zentronics
2954 Blvd. Laurier
Suite 100
Ste-Foy, Quebec G1V 4T2
Canada
Tel: 418-654-1077
Fax: 418-654-2958

Winnipeg, Manitoba
Pioneer/Zentronics
540 Marjorie Street
Winnipeg, Manitoba R3H 0S9
Canada
Tel: 204-989-1957
Fax: 204-633-9255

Future Electronics
106 King Edward
Winnipeg, Manitoba R3H 0N8
Canada
Tel: 204-944-1446
Fax: 204-783-8133

Farnell Electronic Services
Unit 250, 1625 Dublin Ave.
Winnipeg, Manitoba R3H 0W3
Canada
Tel: 204-786-2589
Fax: 204-786-2637

© 1995 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Region</th>
<th>Country</th>
<th>Location</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EUROPE</strong></td>
<td>Austria</td>
<td>Elbatex GmbH</td>
<td>Tel: 43/1-86642-0, Fax: 43/1-86642-201</td>
</tr>
<tr>
<td></td>
<td>Austria</td>
<td>Eitnergasse 6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Austria</td>
<td>A-1230 Wien</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Austria</td>
<td>Tel: 43/1-86642-0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Austria</td>
<td>Fax: 43/1-86642-201</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Baltic</td>
<td>Memec Baltic Ltd.</td>
<td>Tel: 371-2-364684, Fax: 371-8-828062</td>
</tr>
<tr>
<td></td>
<td>Baltic</td>
<td>Brivibas Str. 148A / 709</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Baltic</td>
<td>LV-1012 Riga</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Baltic</td>
<td>Lithuania</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Belgium, Netherlands</td>
<td>Sonetech Nederland BV</td>
<td>Tel: 31-40-837075, Fax: 31-40-832300</td>
</tr>
<tr>
<td></td>
<td>Belgium, Netherlands</td>
<td>P.O. Box 259</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Belgium, Netherlands</td>
<td>5670 AG Nuenen</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Belgium, Netherlands</td>
<td>Netherlands</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Belgium, Netherlands</td>
<td>Tel: 31-40-837075</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Belgium, Netherlands</td>
<td>Fax: 31-40-832300</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Finland</td>
<td>Memec Finland</td>
<td>Tel: 358-07001-9830, Fax: 358-07001-9839</td>
</tr>
<tr>
<td></td>
<td>France</td>
<td>Mecodis</td>
<td>Tel: 33 1 43 99 44 00, Fax: 33 1 43 99 98 28</td>
</tr>
<tr>
<td></td>
<td>France</td>
<td>Parc d'Activites</td>
<td></td>
</tr>
<tr>
<td></td>
<td>France</td>
<td>3 Allee des Erables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>France</td>
<td>94042 CRETEIL Cedex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>Arrow Exatec AS</td>
<td>Tel: 45-44-92-7000, Fax: 45-44-92-6020</td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>Mileparken 20E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>2740 Skovlund</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>Postbus 4903</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>NL-5604 CC Eindhoven</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>Netherlands</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>Tel: 31-40-659399</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denmark</td>
<td>Fax: 31-40-659393</td>
<td></td>
</tr>
<tr>
<td><strong>Germany</strong></td>
<td><strong>Israel</strong></td>
<td><strong>Russia</strong></td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------</td>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td>Rutronik RSC-Halbleiter GmbH</td>
<td>Elina Electronics Ltd.</td>
<td>Gamma Ltd.</td>
<td></td>
</tr>
<tr>
<td>Industriestr. 2</td>
<td>14 Raoul Wallenberg St.</td>
<td>Grazdanski PR-T</td>
<td></td>
</tr>
<tr>
<td>D-75228 Ispringen</td>
<td>P.O. Box 13190</td>
<td>111 KOM 427, 429</td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>Tel: 972 3 49 85 43</td>
<td>Elektronmash Building</td>
<td></td>
</tr>
<tr>
<td>Tel: 49/7231-801-0</td>
<td>Fax: 972 3 49 87 45</td>
<td>R-195265 St. Petersburg</td>
<td></td>
</tr>
<tr>
<td>Fax: 49/7231-82282</td>
<td></td>
<td>Russia</td>
<td></td>
</tr>
<tr>
<td>Future Electronics Deutschland GmbH</td>
<td>Eureletronica SPR</td>
<td>Tel: 7/812-5311-180</td>
<td></td>
</tr>
<tr>
<td>Muenchner Str. 18</td>
<td>Via E. Fermi 8</td>
<td>Fax: 7/812-5311-402</td>
<td></td>
</tr>
<tr>
<td>D-85774 Unterfoehring</td>
<td>20090 Assago Milano</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>Italy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tel: 49/89-95727-0</td>
<td>Tel: 39-2-657841</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fax: 49/89-95727-140</td>
<td>Fax: 39-2-26126270</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semitron W. Roeck GmbH</td>
<td>Farnell Spa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Im Gut 1</td>
<td>Viale Milanofiori E/5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D-79790 Kuessaberg</td>
<td>20090 Assago Milano</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>Italy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tel: 49/7742-8001-0</td>
<td>Tel: 39-2-824701</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fax: 49/7742-6901</td>
<td>Fax: 39-2-82470278</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avnet E2000 GmbH</td>
<td>Kevin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stahlgruberring 12</td>
<td>Via Dei Gradenigo, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D-81829 Muenchen</td>
<td>20148 Milano</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>Italy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tel: 49/89-4 5110-01</td>
<td>Tel: 39-2-48706300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fax: 49/89-4 5110-210</td>
<td>Fax: 39-2-48706500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metronik GmbH</td>
<td><strong>Norway</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leonhardsweg 2</td>
<td>Berendesden</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D-82008 Unterhaching</td>
<td>P.O. Box 9376 Gronlund</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>N0135</td>
<td>Tel: 41/56-275100</td>
<td></td>
</tr>
<tr>
<td>Tel: 49/89-61108-0</td>
<td>Oslo Norway</td>
<td>Fax: 41/56-275454</td>
<td></td>
</tr>
<tr>
<td>Fax: 49/89-61108-155</td>
<td>Tel: 47-22-677290</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Greece</strong></td>
<td>Fax: 47-22-677380</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P. Caritato &amp; Associates S.A.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ilia Iliou 31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Athens 11743</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tel: 30 1 902 01 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fax: 30 1 901 70 24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hungary</strong></td>
<td><strong>Portugal</strong></td>
<td><strong>Turkey</strong></td>
<td></td>
</tr>
<tr>
<td>Humansoft Ltd.</td>
<td>Digicontrolte</td>
<td>Inter Muehendislik Danismanlik</td>
<td></td>
</tr>
<tr>
<td>Angol u. 24/b</td>
<td>Dpt Comercial</td>
<td>Ve Ticaret A.S. 1</td>
<td></td>
</tr>
<tr>
<td>H-1149 Budapest</td>
<td>Aven. Eng Arantes E. Oliveira 52d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hungary</td>
<td>1900 LISBOA</td>
<td>Hasircbasi Caddesi No. 55</td>
<td></td>
</tr>
<tr>
<td>Tel: 36/1162-2879</td>
<td>Portugal</td>
<td>81310 Kadikoy</td>
<td></td>
</tr>
<tr>
<td>Fax: 36/1251-3673</td>
<td>Tel: 35 11 80 57 30</td>
<td>Istanbul</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 35 11 849 03 73</td>
<td>Turkey</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: 90 216 349 94 00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fax: 90 216 349 94 31</td>
<td></td>
</tr>
</tbody>
</table>
Distributors

MEXICO

**Mexico, D.F.**
Semiconductores Profesionales
Madrid No. 55
Col. Del Carmen Coyoacan
04100 Mexico, D. F.
Tel: 525-658-60-11
Fax: 525-658-60-44

**Edo de Mexico**
Electronica Seta
Galeana No. 114-20 Piso
La Loma Tlahuapan
54060 Edo de Mexico
Tel: 525-390-77-13
Fax: 525-390-94-68

**Guadalajara**
Future Electronics de Mexico
S.A. de C.V.
Prof. Americas 1600
2 do Piso
Guadalajara, Jalisco
Mexico 44610
Tel: 523-678-9281
Fax: 523-678-9200

SOUTH AMERICA

**All Other Countries Except Brazil:**

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

**Argentina**
CIKA Electronica SRL
Av. de Los Incas, 4821
1427 Buenos Aires
Tel: 521-3188/8108
Fax: 523-6068

Electronica Elemon SA
Franklin D. Roosevelt, 5415
1431 Buenos Aires
Tel: 523-5555/3909
Fax: 522-735

**Brazil**
Aplicacoes Eletronicas
Artimar Ltda.
Rua Marques de Itu N°. 70-10And.
CEP 01223
Sao Paulo, Brazil
Tel: 55-11-231-0277
Fax: 55-11-255-0511

**Chile**
Victronics Ltda.
Ricardo Matte Perez, 0307
Providencia
Santiago
Tel: 225-3788
Fax: 341-5578
UNITED STATES

Alabama
Future Electronics
4825 University Square, Suite 12
Huntsville, AL 35816
Tel: 205-830-2322
Fax: 205-830-6664

Pioneer Technologies
4835 University Square, Suite 5
Huntsville, AL 35816
Tel: 205-837-9300
Fax: 205-837-9358

Arizona
Bell Industries
10611 North Hayden Road
Building D, Suite 103
Scottsdale, AZ 85260
Tel: 602-905-2355
Fax: 602-905-2356

Future Electronics
4636 E. University Dr., Suite 245
Phoenix, AZ 85034
Tel: 602-968-7140
Fax: 602-968-0334

Pioneer Standard
1438 W. Broadwa, Suite B140
Tempe, AZ 85282
Tel: 602-350-9335
Fax: 602-350-9376

California
San Jose Area
Bell Industries
1161 N. Fairoaks Ave.
Sunnyvale, CA 94089
Tel: 408-734-8570
Fax: 408-734-8875

Future Electronics
1024 Marilyn Dr.
Mountain View, CA 94040
Tel: 408-232-1998
Fax: 408-433-0822

Future Electronics
2220 O'Toole Ave.
San Jose, CA 95131
Tel: 408-434-1122
Fax: 408-433-0822

Pioneer Technical Products
333 River Oaks Parkway
San Jose, CA 95134
Tel: 408-954-9100
Fax: 408-954-9113

Roseville
Bell Industries
300 Douglas Blvd., Suite 205
Roseville, CA 95661
Tel: 916-781-8070
Fax: 916-781-2954

Future Electronics
755 Sunrise Avenue, Suite 150
Roseville, CA 95661
Tel: 916-783-7877
Fax: 916-783-7988

Agoura Hills
Bell Industries
30101 Agoura Court, Suite 118
Agoura Hills, CA 91301
Tel: 818-865-7900
Fax: 818-991-7695

Future Electronics
27499 West Agoura Road, Suite 300
Agoura Hills, CA 91301
Tel: 818-865-0040
Fax: 818-865-1340

Pioneer Standard
5126 Clareton Dr., Suite 160
Agoura Hills, CA 91301
Tel: 818-865-5800
Fax: 818-865-5814

Irvine
Bell Industries
220 Technology Dr. #100
Irvine, CA 92718
Tel: 714-727-4500
Fax: 714-453-4610

Future Electronics
258 Technology, Suite 200
Irvine, CA 92718
Tel: 714-453-1515
Fax: 714-453-1226

Pioneer Standard
217 Technology Drive #110
Irvine, CA 92718
Tel: 800-753-5090
Fax: 714-753-5074

San Diego
Aegis Electronic Group, Inc.
1015 Chestnut Ave., Suite G2
Carlsbad, CA 92008
Tel: 619-729-2026
Fax: 619-729-9295

Bell Industries
5520 Ruffin Rd., Suite 209
San Diego, CA 92123
Tel: 619-576-3204
Fax: 800-444-0139

Future Electronics
5151 Shoreham Place, Suite 220
San Diego, CA 92122
Tel: 619-625-2800
Fax: 619-625-2810

Pioneer Standard
9449 Balboa Ave., Suite 114
San Diego, CA 92123
Tel: 619-514-7700
Fax: 619-514-7799

© 1995 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Distributors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Colorado</strong></td>
</tr>
</tbody>
</table>
| Bell Industries  
1873 S. Bellaire St.  
Denver, CO 80222  
Tel: 303-691-9270  
Fax: 303-790-4991 |
| Future Electronics  
12600 W. Colfax Avenue  
Suite B110  
Lakewood, CO 80215  
Tel: 303-232-2008  
Fax: 303-232-2009 |
| Pioneer Technologies  
5600 Greenwood Plaza Blvd. Suite 201  
Englewood, CO 80111  
Tel: 303-773-8090  
Fax: 303-773-8194 |
| **Connecticut** |
| Bell Industries  
1064 East Main Street  
Meriden, CT 06450  
Tel: 203-639-6000  
Fax: 203-639-6005 |
| Future Electronics  
700 W. Johnson Ave. Westgate Office Center  
Cheshire, CT 06410  
Tel: 203-250-0083  
Fax: 203-250-0081 |
| Pioneer Standard  
Two Trap Falls #101  
Shelton, CT 06484  
Tel: 203-929-5600  
Fax: 203-929-9791 |
| **Florida** |
| *Altamonte Springs*  
Bell Industries  
650 S. North Lake Blvd. #400  
Altamonte Springs, FL 32701  
Tel: 407-339-0076  
Fax: 407-339-0139 |
| Future Electronics  
237 S. Westmonte Drive, Suite 307  
Altamonte Springs, FL 32714  
Tel: 407-865-7900  
Fax: 407-865-7660 |
| Pioneer Technologies  
337 South-North Lake, Suite 1000  
Altamonte Springs, FL 32701  
Tel: 407-834-9090  
Fax: 407-834-0865 |
| *Deerfield Beach*  
Future Electronics  
1400 E. Newport Center Drive  
Suite 200  
Deerfield Beach, FL 33442  
Tel: 305-426-4043  
Fax: 305-426-3939 |
| Pioneer Technologies  
674 S. Military Trail  
Deerfield Beach, FL 33442  
Tel: 305-426-8877  
Fax: 305-481-2950 |
| *Largo*  
Future Electronics  
2200 Tall Pines Drive, Suite 108  
Largo, FL 34641  
Tel: 813-530-1222  
Fax: 813-538-9598 |
| **Georgia** |
| *Pioneer Technologies*  
4250 C Rivergreen Pkwy.  
Duluth, GA 30013  
Tel: 404-623-1003  
Fax: 404-623-0665 |
| Bell Industries  
3020 Business Park Drive, Suite A  
Norcross, GA 30071  
Tel: 404-446-7167  
Fax: 404-446-7264 |
| Future Electronics  
3150 Holcomb Bridge Rd. Suite 130  
Norcross, GA 30071  
Tel: 404-441-7676  
Fax: 404-441-7580 |
| **Illinois** |
| Bell Industries  
870 Cambridge Drive  
Elk Grove Village, IL 60007  
Tel: 708-640-1910  
Fax: 708-640-1942 |
| Future Electronics  
3150 W. Higgins Rd., Suite 160  
Hoffman Estates, IL 60195  
Tel: 708-882-1255  
Fax: 708-490-9290 |
| *Pioneer Standard*  
2171 Executive Drive #200  
Addison, IL 60101  
Tel: 708-495-9680  
Fax: 708-495-9831 |
Indiana

**Fort Wayne**

Bell Industries  
525 Airport North Office Park  
Fort Wayne, IN 46825  
Tel: 219-490-2100  
Fax: 219-490-2104

Pioneer Standard  
237 Airport N. Office Park  
Fort Wayne, IN 46825  
Tel: 219-489-0283  
Fax: 219-489-6262

**Indianapolis**

Bell Industries  
5230 W. 79th St.  
P.O. Box 6885  
Indianapolis, IN 46268  
Tel: 317-875-8200  
Fax: 317-875-8219

Future Electronics  
8425 Woodfield Crossing  
Suite 175  
Indianapolis, IN 46240  
Tel: 317-469-0447  
Fax: 317-469-0448

Pioneer Standard  
9350 N. Priority Way W. Dr.  
Indianapolis, IN 46240  
Tel: 317-573-0880  
Fax: 317-573-0979

**Kansas**

Future Electronics  
8826 Sante Fe Drive, Suite 150  
Overland Park, KS 66212  
Tel: 913-649-1531  
Fax: 913-649-1786

Maryland

**Fort Wayne**

Bell Industries  
8945 Guilford Rd., Suite 130  
Columbia, MD 21046  
Tel: 800-274-6953  
Fax: 410-290-8006

Future Electronics  
6716 Alexander Bell Drive  
Suite 101  
Columbia, MD 21046  
Tel: 410-290-0600  
Fax: 410-290-0328

**Indianapolis**

Bell Industries  
5230 W. 79th St.  
P.O. Box 6885  
Indianapolis, IN 46268  
Tel: 317-875-8200  
Fax: 317-875-8219

Seymour Electronics  
Columbia Business Center  
6440 Dobbin Road, Suite B  
Columbia, MD 21045  
Tel: 410-992-7474  
Fax: 410-992-7410

Future Electronics  
6716 Alexander Bell Drive  
Suite 101  
Columbia, MD 21046  
Tel: 410-290-0600  
Fax: 410-290-0328

**Kansas**

Future Electronics  
8826 Sante Fe Drive, Suite 150  
Overland Park, KS 66212  
Tel: 913-649-1531  
Fax: 913-649-1786

Future Electronics  
4505 Broadmoor S.E.  
Grand Rapids, MI 49512  
Tel: 616-698-6800  
Fax: 616-698-6821

Pioneer Standard  
4477 Byron Ctr Rd SW  
Grand Rapids, MI 49509  
Tel: 616-534-3145  
Fax: 616-534-3922

**Detroit Area**

Future Electronics  
35200 Schoolcraft Road, Suite 106  
Livonia, MI 48150  
Tel: 313-261-5270  
Fax: 313-261-8125

Pioneer Standard  
4499 Plymouth Oak Blvd.  
Plymouth, MI 48170  
Tel: 313-525-1800  
Fax: 313-427-3720

**Massachusetts**

Bell Industries  
100 Burtt Road #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: 508-779-3050

**Michigan**

**Grand Rapids**

Future Electronics  
4505 Broadmoor S.E.  
Grand Rapids, MI 49512  
Tel: 616-698-6800  
Fax: 616-698-6821

Pioneer Standard  
4477 Byron Ctr Rd SW  
Grand Rapids, MI 49509  
Tel: 616-534-3145  
Fax: 616-534-3922

**Detroit Area**

Future Electronics  
35200 Schoolcraft Road, Suite 106  
Livonia, MI 48150  
Tel: 313-261-5270  
Fax: 313-261-8125

Pioneer Standard  
4499 Plymouth Oak Blvd.  
Plymouth, MI 48170  
Tel: 313-525-1800  
Fax: 313-427-3720

**Massachusetts**

Bell Industries  
100 Burtt Road #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: 508-779-3050

Pioneer Standard  
44 Hartwell Ave.  
Lexington, MA 02173  
Tel: 617-861-9200  
Fax: 617-863-1547

© 1995 Microchip Technology Inc.
## Distributors

### Minnesota

**Bloomington**
- Bell Industries
  - 9400 James Ave. So., #142
  - Bloomington, MN 55431
  - Tel: 612-888-7247
  - Fax: 612-888-7757

**Eden Prairie**
- Future Electronics
  - 10025 Valley View Road, Suite 196
  - Eden Prairie, MN 55344
  - Tel: 612-944-2200
  - Fax: 612-944-2520

- Pioneer Standard
  - 7625 Golden Triangle
  - Eden Prairie, MN 55344
  - Tel: 612-944-3355
  - Fax: 612-944-3794

**Thief River Falls**
- Digi-Key Corporation
  - 701 Brooks Ave. So.
  - P.O. Box 677
  - Thief River Falls, MN 55714
  - Tel: 218-681-6674
  - Fax: 218-681-3380

### New Jersey

**Northern**
- Bell Industries
  - 271 Route 46 West
  - Suites F202-203
  - Fairfield, NJ 07004
  - Tel: 201-227-6060
  - Fax: 201-227-2626

- Pioneer Standard
  - 14A Madison Road
  - Fairfield, NJ 07006
  - Tel: 201-575-3510
  - Fax: 201-575-3454

- Future Electronics
  - 1259 Route 46 East
  - Parsippany, NJ 07054
  - Tel: 201-299-0400
  - Fax: 201-299-1377

**Phase 1**
- Technology
  - 295 Molnar Drive
  - Elmwood Park, NJ 07407
  - Tel: 201-791-2990
  - Fax: 201-791-2552

- Seymour Electronics Corporation
  - 357 Crossways Park Drive
  - Woodbury, NY 11797-2042
  - Tel: 201-465-7474
  - Fax: 201-465-0890

### New Jersey (continued)

**Southern**
- Bell Industries
  - 158 Gaither Drive, Suite 110
  - Mt. Laurel, NJ 08054
  - Tel: 609-439-9009
  - Fax: 609-439-0570

- Phase 1
  - 12-B Ellipse Bldg. #221
  - 4201 Church Road
  - Mount Laurel, NJ 08054
  - Tel: 609-234-3237
  - Fax: 609-234-5012

- Pioneer Technologies
  - 500 Enterprise Road
  - Keith Valley Business Center
  - Horsham, PA 19044
  - Tel: 215-674-4000
  - Fax: 215-674-3107

- Seymour Electronics
  - 520 Fellowship Road, Suite A104
  - Mt. Laurel, NJ 08054
  - Tel: 609-235-7474
  - Fax: 609-235-4992

- Future Electronics
  - 12 East Stow Road, Suite 200
  - Marlton, NJ 08053
  - Tel: 609-596-4080
  - Fax: 609-596-4266

---

© 1995 Microchip Technology Inc.
New Mexico
Bell Industries
11728 Linn N.E.
Albuquerque, NM 87123
Tel: 505-292-2700
Fax: 505-275-2819

New York
Binghamton
Pioneer Standard
1249 Front Street, Suite 201
Binghamton, NY 13901
Tel: 607-722-9300
Fax: 607-722-9562

Long Island
Future Electronics
801 Motor Parkway
Hauppauge, NY 11788
Tel: 516-234-4000
Fax: 516-234-6183

Phase 1 Technology
46 Jefryne Blvd.
Deer Park, NY 11729
Tel: 516-254-2600
Fax: 516-254-2693

Pioneer Standard
60 Crossways Park West
Woodbury, NY 11797
Tel: 516-921-8700
Fax: 516-921-2143

Seymour Electronics
357 Crossways Park Drive
Woodbury, NY 11797-2042
Tel: 516-496-7474
Fax: 516-496-0857

Upstate New York
Pioneer Standard
840 Fairport Park
Fairport, NY 14450
Tel: 716-381-7070
Fax: 716-381-5955

Future Electronics
300 Linden Oaks
Rochester, NY 14625
Tel: 716-387-9550
Fax: 716-387-9563

Future Electronics
200 Salina Meadows Parkway
Suite 130
Syracuse, NY 13212
Tel: 315-451-2371
Fax: 315-451-7258

North Carolina
Future Electronics
8401 University Executive PK.
Suite 108
Charlotte, NC 28262
Tel: 704-547-1107
Fax: 704-547-9650

Future Electronics
5225 Capital Blvd.
1 North Commerce Center
Raleigh, NC 27604
Tel: 919-790-7111
Fax: 919-790-9022

Pioneer Technologies
2200 Gateway Centre Blvd.
Suite 215
Morrisville, NC 27560
Tel: 919-460-1530
Fax: 919-460-1540

Ohio
Cleveland
Future Electronics
6009-E Landerhaven Dr.
Mayfield Heights, OH 44124
Tel: 216-449-6996
Fax: 216-449-8987

Bell Industries
31200 Solon Road Unit 11
Solon, OH 44139
Tel: 216-496-2002
Fax: 216-496-2006

Pioneer Standard
29125 Solon Road
Solon, OH 44139
Tel: 216-248-8710
Fax: 216-248-9166

Dayton
Bell Industries
446 Windsor Park Drive
Dayton, OH 45459
Tel: 513-434-8231
Fax: 513-434-8103

Pioneer Standard
4433 Interpoint Blvd.
Dayton, OH 45424
Tel: 513-236-9900
Fax: 513-236-8133

Future Electronics
1430 Oak Ct., Suite 203
Beavercreek, OH 45430
Tel: 513-426-0090
Fax: 513-426-8490

Worthington
Pioneer Standard
100 Old Wilson Bridge, Suite 105
Worthington, OH 43085
Tel: 614-848-4854
Fax: 614-848-4889

Oklahoma
Pioneer Standard
9717 E. 42nd St., Suite 105
Tulsa, OK 74146
Tel: 918-665-7840
Fax: 918-665-1891

© 1995 Microchip Technology Inc.
<table>
<thead>
<tr>
<th>Oregon</th>
<th>Texas</th>
<th>Washington</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bell Industries</td>
<td>Future Electronics</td>
<td>Bell Industries</td>
</tr>
<tr>
<td>9275 S.W. Nimbus</td>
<td>9020 II Capital of Texas Hwy N</td>
<td>1715 114th Ave SE #208</td>
</tr>
<tr>
<td>Beaverton, OR 97005</td>
<td>Austin</td>
<td>Bellevue, WA 98004</td>
</tr>
<tr>
<td>Tel: 503-644-3444</td>
<td>Tel: 214-690-9096</td>
<td>Tel: 206-646-8750</td>
</tr>
<tr>
<td>Fax: 503-520-1948</td>
<td>Fax: 214-690-0467</td>
<td>Fax: 206-646-8559</td>
</tr>
<tr>
<td>Future Electronics</td>
<td>Future Electronics</td>
<td>Pioneer Technologies</td>
</tr>
<tr>
<td>15236 N.W. Greenbrier Pkwy.</td>
<td>800 E. Campbell Rd., Suite 130</td>
<td>2800 156th Ave., SE, Suite 100</td>
</tr>
<tr>
<td>Beaverton, OR 97006</td>
<td>Richardon, TX 75081</td>
<td>Bellevue, WA 98007</td>
</tr>
<tr>
<td>Tel: 503-645-9454</td>
<td>Tel: 214-437-2437</td>
<td>Tel: 206-644-7500</td>
</tr>
<tr>
<td>Fax: 503-645-1559</td>
<td>Fax: 214-669-2347</td>
<td>Fax: 206-644-7300</td>
</tr>
<tr>
<td>Pioneer Technologies</td>
<td>Pioneer Technologies</td>
<td>Future Electronics</td>
</tr>
<tr>
<td>8905 SW Nimbus, Suite 160</td>
<td>13765 Beta Road</td>
<td>19102 N. Creek Parkway South</td>
</tr>
<tr>
<td>Beaverton, OR 97008</td>
<td>Dallas</td>
<td>Suite 118</td>
</tr>
<tr>
<td>Tel: 503-626-7300</td>
<td>Tel: 214-386-7300</td>
<td>Bothell, WA 98011</td>
</tr>
<tr>
<td>Fax: 503-626-5300</td>
<td>Fax: 214-490-6419</td>
<td>Tel: 206-489-3400</td>
</tr>
<tr>
<td></td>
<td>Houston</td>
<td>Fax: 206-489-3411</td>
</tr>
<tr>
<td></td>
<td>Future Electronics</td>
<td>Pioneer Standard</td>
</tr>
<tr>
<td></td>
<td>10333 Richmond Ave., Suite 970</td>
<td>120 Bishop's Way, Suite 163</td>
</tr>
<tr>
<td></td>
<td>Houston</td>
<td>Brookfield, WI 53005</td>
</tr>
<tr>
<td></td>
<td>Tel: 713-785-1155</td>
<td>Tel: 414-784-3480</td>
</tr>
<tr>
<td></td>
<td>Fax: 713-785-4558</td>
<td>Fax: 414-784-8207</td>
</tr>
<tr>
<td></td>
<td>Pioneer Standard</td>
<td>Bell Industries</td>
</tr>
<tr>
<td></td>
<td>10530 Rockley Rd, Ste 100</td>
<td>W 226 N 900 Eastmound</td>
</tr>
<tr>
<td></td>
<td>Houston</td>
<td>Waukesha, WI 53186</td>
</tr>
<tr>
<td></td>
<td>Tel: 713-495-4700</td>
<td>Tel: 414-547-8879</td>
</tr>
<tr>
<td></td>
<td>Fax: 713-495-5642</td>
<td>Fax: 414-547-6547</td>
</tr>
<tr>
<td></td>
<td>Utah</td>
<td>Authorized Distributor for</td>
</tr>
<tr>
<td></td>
<td>Bell Industries</td>
<td>Obsolete Products</td>
</tr>
<tr>
<td></td>
<td>6912 S. 185 West, Suite B</td>
<td>Rochester Electronics, Inc.</td>
</tr>
<tr>
<td></td>
<td>Midvale, UT 84047</td>
<td>10 Malcolm Hoyt Drive</td>
</tr>
<tr>
<td></td>
<td>Tel: 801-561-9691</td>
<td>Newburyport, MA 01950</td>
</tr>
<tr>
<td></td>
<td>Fax: 801-255-2477</td>
<td>Tel: 508-462-9332</td>
</tr>
<tr>
<td></td>
<td>Future Electronics</td>
<td>Fax: 508-462-9512</td>
</tr>
<tr>
<td></td>
<td>3540 S. Highland Drive #301</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Salt Lake City, UT 84106</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: 801-467-4448</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 801-467-3604</td>
<td></td>
</tr>
</tbody>
</table>

**Pennsylvania**

**Philadelphia Area**

Bell Industries

158 Gaither Drive, Suite 110

Mt. Laurel, NJ 08054

Tel: 609-439-9009

Fax: 609-439-0570

Phase 1

12-B Ellipse Bldg. #221

4201 Church Road

Mount Laurel, NJ 08054

Tel: 609-234-3237

Fax: 609-234-5012

Pioneer Technologies

500 Enterprise Road

Keith Valley Business Center

Horsham, PA 19044

Tel: 215-674-4000

Fax: 215-674-3107

Seymour Electronics

520 Fellowship Road, Suite A104

Mt. Laurel, NJ 08054

Tel: 609-235-7474

Fax: 609-235-4992

Future Electronics

12 East Stow Road, Suite 200

Marlton, NJ 08053

Tel: 609-596-4080

Fax: 609-596-4266

**Pittsburgh**

Pioneer Standard

259 Kappa Drive

Pittsburgh, PA 15238

Tel: 412-782-2300

Fax: 412-963-8255

<table>
<thead>
<tr>
<th>Texas</th>
<th>Washington</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austin</td>
<td>Bell Industries</td>
</tr>
<tr>
<td>Future Electronics</td>
<td>1715 114th Ave SE #208</td>
</tr>
<tr>
<td>9020 II Capital of Texas Hwy N</td>
<td>Bellevue, WA 98004</td>
</tr>
<tr>
<td>Austin, TX 78759</td>
<td>Tel: 206-646-8750</td>
</tr>
<tr>
<td>Tel: 512-502-0991</td>
<td>Fax: 206-646-8559</td>
</tr>
<tr>
<td>Fax: 512-502-0740</td>
<td>Pioneer Technologies</td>
</tr>
<tr>
<td>Future Electronics</td>
<td>2800 156th Ave., SE, Suite 100</td>
</tr>
<tr>
<td>800 E. Campbell Rd., Suite 130</td>
<td>Bellevue, WA 98007</td>
</tr>
<tr>
<td>Richardon, TX 75081</td>
<td>Tel: 206-644-7500</td>
</tr>
<tr>
<td>Tel: 214-437-2437</td>
<td>Fax: 206-644-7300</td>
</tr>
<tr>
<td>Fax: 214-669-2347</td>
<td>Future Electronics</td>
</tr>
<tr>
<td>Houston</td>
<td>19102 N. Creek Parkway South</td>
</tr>
<tr>
<td>Future Electronics</td>
<td>Suite 118</td>
</tr>
<tr>
<td>10333 Richmond Ave., Suite 970</td>
<td>Bothell, WA 98011</td>
</tr>
<tr>
<td>Houston</td>
<td>Tel: 206-489-3400</td>
</tr>
<tr>
<td>Tel: 713-785-1155</td>
<td>Fax: 206-489-3411</td>
</tr>
<tr>
<td>Fax: 713-785-4558</td>
<td></td>
</tr>
<tr>
<td>Pioneer Standard</td>
<td></td>
</tr>
<tr>
<td>10530 Rockley Rd, Ste 100</td>
<td></td>
</tr>
<tr>
<td>Houston, TX 77099</td>
<td></td>
</tr>
<tr>
<td>Tel: 713-495-4700</td>
<td></td>
</tr>
<tr>
<td>Fax: 713-495-5642</td>
<td></td>
</tr>
<tr>
<td>Utah</td>
<td></td>
</tr>
<tr>
<td>Bell Industries</td>
<td></td>
</tr>
<tr>
<td>6912 S. 185 West, Suite B</td>
<td></td>
</tr>
<tr>
<td>Midvale, UT 84047</td>
<td></td>
</tr>
<tr>
<td>Tel: 801-561-9691</td>
<td></td>
</tr>
<tr>
<td>Fax: 801-255-2477</td>
<td></td>
</tr>
<tr>
<td>Future Electronics</td>
<td></td>
</tr>
<tr>
<td>3540 S. Highland Drive #301</td>
<td></td>
</tr>
<tr>
<td>Salt Lake City, UT 84106</td>
<td></td>
</tr>
<tr>
<td>Tel: 801-467-4448</td>
<td></td>
</tr>
<tr>
<td>Fax: 801-467-3604</td>
<td></td>
</tr>
</tbody>
</table>

**Wisconsin**

Future Electronics

250 N. Patrick Boulevard, Suite 170

Brookfield, WI 53005

Tel: 414-879-0244

Fax: 414-879-0250

Pioneer Standard

120 Bishop's Way, Suite 163

Brookfield, WI 53005

Tel: 414-784-3480

Fax: 414-784-8207

Bell Industries

W 226 N 900 Eastmound

Waukesha, WI 53186

Tel: 414-547-8879

Fax: 414-547-6547

**Authorized Distributor for Obsolete Products**

Rochester Electronics, Inc.

10 Malcolm Hoyt Drive

Newburyport, MA 01950

Tel: 508-462-9332

Fax: 508-462-9512

© 1995 Microchip Technology Inc.
# Factory Sales

## AMERICAS

**Corporate Office**
Microchip Technology Inc.  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 602 786-7200 Fax: 602 786-7277  
Technical Support: 602 786-7627  
Web: http://www.mchp.com/biz/mchp

**Atlanta**
Microchip Technology Inc.  
500 Sugar Mill Road, Suite 2008  
Atlanta, GA 30350  
Tel: 770 640-0034 Fax: 770 640-0307

**Boston**
Microchip Technology Inc.  
5 Mount Royal Avenue  
Marlborough, MA 01752  
Tel: 508 480-9990 Fax: 508 480-8575

**Chicago**
Microchip Technology Inc.  
333 Pierce Road, Suite 180  
Itasca, IL 60143  
Tel: 708 285-0071 Fax: 708 285-0075

**Dallas**
Microchip Technology Inc.  
14651 Dallas Parkway, Suite 816  
Dallas, TX 75240-0809  
Tel: 214 991-7177 Fax: 214 991-8588

**Dayton**
Microchip Technology Inc.  
35 Rockridge Road  
Englewood, OH 45322  
Tel: 513 832-2543 Fax: 513 832-2841

**Los Angeles**
Microchip Technology Inc.  
18201 Von Karman, Suite 455  
Irvine, CA 92715  
Tel: 714 263-1888 Fax: 714 263-1338

**New York**
Microchip Technology Inc.  
150 Motor Parkway, Suite 416  
Hauppauge, NY 11788  
Tel: 516 273-5305 Fax: 516 273-5335

**San Jose**
Microchip Technology Inc.  
2107 North First Street, Suite 590  
San Jose, CA 95131  
Tel: 408 436-7950 Fax: 408 436-7955

## ASIA/PACIFIC

**Hong Kong**
Microchip Technology  
Unit No. 3002-3004, Tower 1  
Metropiazza  
223 Hing Fong Road  
Kwai Fong, N.T. Hong Kong  
Tel: 852 2 401 1200 Fax: 852 2 401 3431

**Korea**
Microchip Technology  
168-1, Youngbo Bldg. 3 Floor  
Samsung-Dong, Kangnam-Ku,  
Seoul, Korea  
Tel: 82 2 554 7200 Fax: 82 2 558 5934

**Singapore**
Microchip Technology  
200 Middle Road  
#10-03 Prime Centre  
Singapore 18998  
Tel: 65 334 8670 Fax: 65 334 8650

**Taiwan**
Microchip Technology  
10F-1C 207  
Tung Hua North Road  
Taipei, Taiwan, ROC  
Tel: 886 2 717 7175 Fax: 886 2 545 0139

## EUROPE

**United Kingdom**
Arizona Microchip Technology Ltd.  
Unit 6, The Courtyard  
Meadow Bank, Furlong Road  
Boune End, Buckinghamshire SL8 5AJ  
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

**France**
Arizona Microchip Technology SARL  
2 Rue du Buisson aux Fraises  
91300 Massy - France  
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

**Germany**
Arizona Microchip Technology GmbH  
Gustav-Heinemann-Ring 125  
D-81739 Muenchen, Germany  
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

**Italy**
Arizona Microchip Technology SRL  
Centro Direzionale Colleoni  
Palazzo Pegaso ingresso No. 2  
Via Paracelso 23, 20041  
Agrate Brianza (MI) Italy  
Tel: 39 039 689 9939 Fax: 39 039 689 9883

**Japan**
Microchip Technology Intl. Inc.  
BeneX S-1 6F  
3-12-20, Shin Yokohama  
Kohoku-Ku, Yokohama  
Kanagawa 222 Japan  
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/05/95
Microchip Worldwide Sales and Distribution