Using the Serial Peripheral Interface to Communicate Between Multiple Microcomputers

As the complexity of user applications increases, many designers find themselves needing multiple microprocessors to provide necessary functionality in a circuit. Communication between multiple processors can often be difficult, especially when differing processors are used. A possible solution to this problem is usage of the serial peripheral interface (SPI), an interface intended for communication between integrated circuits on the same printed wire board. The MC68HC05C4 is one of the first single-chip microcomputers to incorporate SPI into hardware. One advantage of the SPI is that it can be provided in software, allowing communication between two microcomputers where one has SPI hardware and one does not. Special interfacing is necessary when using the hardware SPI to communicate with a microcomputer that does not include SPI hardware. This interface can be illustrated with a circuit used to display either temperature or time, that incorporates both a MC68HC05C4 and a MC68705R3. The MC68HC05C4 monitors inputs from a keypad and controls the SPI data exchange, while the MC68705R3 determines temperature by performing an analog-to-digital conversion on inputs from a temperature sensor and controls the LED display. Communication between the microcomputers is handled via SPI, with the MC68HC05C4 handling exchanges in hardware, and the MC68705R3 handling them in software.

Usage of software SPI can be expanded to include circuits where the single-chip implementing the SPI in software controls the data exchange, and those in which neither single-chip has hardware SPI capability. Minor modifications to the SPI code are necessary when data exchanges are controlled by the software.

Debugging designs including multiple processors can often be confusing. Some of the confusion can be alleviated by careful planning of both the physical debugging environment and the order in which software is checked.

**SERIAL PERIPHERAL INTERFACE**

Communication between the two processors is handled via the serial peripheral interface (SPI). Every SPI system consists of one master and one or more slaves, where a master is defined as the microcomputer that provides the SPI clock, and a slave is any integrated circuit that receives the SPI clock from the master. It is possible to have a system where more than one IC can be master, but there can only be one master at any given time. In this design, the MC68HC05C4 is the master and the MC68705R3 is the slave. Four basic signals, master-out/slave-in (MOSI), master-in/slave-out (MISO), serial clock (SCK), and slave select (SS), are needed for an SPI. These four signals are provided on the MC68HC05C4 on port D, pins 2-5.

**SIGNALS**

The MOSI pin is configured as a data output on the master and a data input on the slave. This pin is used to transfer data serially from the master to a slave, in this case the MC68HC05C4 to the MC68705R3. Data is transferred most significant bit first.

Data transfer from slave to master is carried out across the MISO, master-in/slave-out, line. The MISO pin is configured as an input on the master device and an output on the slave device. As with data transfers across the MOSI line, data is transmitted most significant bit first.

All data transfers are synchronized by the serial clock. One bit of data is transferred every clock pulse, and one byte can be exchanged in eight clock cycles. Since the serial clock is generated by the master, it is an input on the slave. The serial clock is derived from the master’s internal processor clock, and clock rate is selected by setting bits 0 and 1 of the serial peripheral control register to choose one of four divide-by values. Values for the MCUs crystal oscillators and the SPI divide-by must be chosen so that the SPI clock is no faster than the internal processor clock on the slave.

The last of the four SPI signals is the slave select (SS). Slave select is an active low signal, and the SS pin is a fixed input which is used to enable a slave to receive data. A master will become a slave when it detects a low level on its SS line. In this design, the MC68HC05C4 is always the master, so its SS line is tied to VDD through a pull-up resistor.

**REGISTERS**

Three registers unique to the serial peripheral interface provide control, status, and data storage.

The Serial Peripheral Control Register (SPCR), shown below, provides control for the SPI.

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0A</td>
<td>SPIE</td>
<td>SPE</td>
<td>MSTR</td>
<td>CPOL</td>
<td>CPHA</td>
<td>SPR1</td>
<td>SPR0</td>
</tr>
</tbody>
</table>

**SPICE—Serial Peripheral Interrupt Enable**

0 = SPIF interrupts disabled
1 = SPI interrupt if SPIF = 1

**SPE—Serial Peripheral System Enable**

0 = SPI system off
1 = SPI system on

**MSTR—Master Mode Select**

0 = Slave mode
1 = Master mode

©MOTOROLA INC., 1987
CPOL — Clock Polarity
When the clock polarity bit is cleared and data is not being transferred, a steady state low value is produced at the SCK pin of the master device. Conversely, if this bit is set, the SCK pin will idle high. This bit is also used in conjunction with the clock phase control bit to produce the desired clock-data relationship between master and slave. See Figure 1.

CPHA — Clock Phase
The clock phase bit, in conjunction with the CPOL bit, controls the clock-data relationship between master and slave. The CPOL bit can be thought of as simply inserting an inverter in series with the SCK line. The CPHA bit selects one of two fundamentally different clocking protocols. When CPHA = 0, the shift clock is the OR of SCK with SS. As soon as SS goes low the transaction begins and the first edge on SCK involves the first data sample. When CPHA = 1, the SS pin may be thought of as a simple output enable control. Refer to Figure 1.

SPR1 and SPR0 — SPI Clock Rate Selects
These two serial peripheral rate bits select one of four baud rates (Table 1) to be used as SCK if the device is a master; however, they have no effect in the slave mode.

<table>
<thead>
<tr>
<th>SPR1</th>
<th>SPR0</th>
<th>Internal Processor Clock Divide By</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>

Data for the SPI is transmitted and received via the Serial Peripheral Data Register (SPDR). A data transfer is initiated by the master writing to its SPDR. If the master is sending data to a slave, it first loads the data into the SPDR and then transfers it to the slave. When reading data, the data bits are gathered in the SPDR and then the complete byte can be accessed by reading the SPDR.

DEMONSTRATION BOARD DESCRIPTION
A keypad input from the user is used to choose the output display function. The MC68HC05C4 monitors the keypad, decodes any valid inputs, and sends the data to the MC68705R3. If the user has requested a temperature display, the MC68705R3 sends a binary value of temperature in degrees fahrenheit to the MC68HC05C4, where the value is converted to a Celsius binary coded decimal value and returned to the MC68705R3 to be displayed. The LEDs are common anode displays and are driven directly off of port B on the MC68705R3. If the user desires the circuit to function as a real-time clock, a starting time must be entered and transmitted from the MC68HC05C4 to the MC68705R3. Once the clock has been initialized, the MC68705R3 updates the clock every minute. Clock values are stored in memory, and when the circuit is functioning as a thermometer, the values in memory are updated as required to maintain clock accuracy.

USING THE A/D CONVERTOR TO MONITOR TEMPERATURE
Temperature monitoring is performed by the Motorola MTS102 silicon temperature sensor and the LM358 Dual Low-Power Operational Amplifier, as shown in the schematic in...
Figure 2. Variations in the base-emitter voltage of the Motorola MTS102 silicon temperature sensor are monitored by the MC68705R3, which converts these analog inputs to equivalent digital values in degrees fahrenheit. The sensor voltage is buffered, inverted, and amplified by a dual differential amplifier before entering the A/D converter. An amplifier gain of 16 is used, resulting in 20-millivolt steps per degree fahrenheit. Using a VCC of 5 volts, the maximum differential amplifier output is 3.8 volts, resulting in a temperature sensing range from -40 degrees to +140 degrees fahrenheit.

The output from the differential amplifier is connected to the A/D converter on the MC68705R3. A block diagram of the successive approximation A/D converter is shown in Figure 3. Provision is made for four separate external inputs and four internal analog channels.

Two different registers associated with the converter control channel selection, initiate a conversion, and store the result of a completed conversion. Both the external and the internal input channels are chosen by setting the lower 3 bits of the A/D Control Register (ACR). The internal input channels are connected to the V RH/V RL resistor chain and may be used for calibration purposes.

The converter operates continuously, requiring 30 machine cycles per conversion. Upon completion of a conversion, the digital value of the analog input is placed in the A/D result register (ARR) and the conversion complete flag, bit 7 of the ACR, is set. Another sample of the selected input is taken, and a new conversion is started.

Conversions are performed internally in hardware by a simple bisection algorithm. The D/A converter (DAC) is initially set to $80, the midpoint of the available conversion range. This value is compared with the input value and, if the input value is larger, $80 becomes the new minimum conversion value and the DAC is once again set to the midpoint of the conversion range, which is now $C0. If the input value is less than $80, $80 becomes the maximum conversion value and the DAC is set to the midpoint of the new conversion range, in this case $40. This process is repeated until all eight bits of the conversion are determined.

Quantizing errors are reduced to +1/2 LSB, rather than +0, −1 LSB, through usage of a built-in 1/2 LSB offset. Ignoring errors, the transition between 00 and 01 will occur at 1/2 LSB above the voltage reference low, and the transition between $FE and $FF will occur 1-1/2 LSBs below voltage reference high.

The A/D converter returns a value of $30 when given an input of zero degrees fahrenheit, so $30 must be subtracted from the result before converting to celsius. This offset must also be considered when calibrating the sensor. Calibration of the temperature sensor can be performed by adjusting the variable resistor to produce a display of $00 after a piece of ice has been placed on the temperature sensor for approximately one minute. A 00 display results from a value of $50 in the ARR, so the variable resistor should be adjusted until this value is reached.

**COMMUNICATION CONSIDERATIONS**

In this application, an SPI read or write is initiated via an interrupt from the MCU desiring to write data. When any of the three function keys, display temperature, set time, or display time, are pressed, the MC68HC05C4, as master, sends the MC68705R3 an interrupt on the MC68705R3's INT pin. The MC68HC05C4 writes the key value to its serial peripheral data register, thereby initiating the SPI. It then waits for the SPIF bit to go high and returns to scanning the keypad.

At the same time the MC68HC05C4 is writing to its SPDR, the MC68705R3 sets a bit counter to eight and waits for the first SCK from the MC68HC05C4. After each clock pulse, the MC68705R3 checks the status of the data bit, sets the carry bit equal to the data bit, and rotates the carry bit left into a register. The bit counter is decremented, compared to zero, and if not zero, the MC68705R3 waits for the next clock pulse and repeats the cycle.

To ensure proper data transfers, the internal processor clock of the MC68705R3 must be sufficiently faster than the SPI clock of the MC68HC05C4 to allow the MC68705R3 time to complete this routine before the MC68HC05C4 can send another bit. This requires the user to first write the code to handle the software SPI, count machine cycles, and then choose MCU oscillator values that allow the additional machine cycles required in a software SPI to be completed before the master can send another clock pulse to the slave.

For example, consider the following piece of code for the MC68705R3, a slave receiving data from the master.

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Instruction</th>
<th>Data In</th>
<th>SCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>STA BITCT</td>
<td>PORTC pin 5</td>
<td>PORTC pin 4</td>
</tr>
<tr>
<td>10</td>
<td>BRSET 4, PORTC,*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>BRSET 5, PORTC, STR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>STR ROL RESULT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>DEC BITCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BNE NXT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Execution of this code requires 43 machine cycles. The maximum oscillator speed for an MC68705R3 is 1 MHz, requiring an SPI clock no greater than 1/43 MHz. One way of obtaining this rate for the SPI clock is to run the MC68HC05C4 at 0.5 MHz and choose a divide-by-32 to generate the SPI clock. If the user has selected a temperature display, it is necessary for the MC68705R3, as a slave, to send data to the MC68HC05C4 master. When the MC68705R3 is ready to send data, it interrupts the MC68HC05C4 via the MC68HC05C4's IRQ line. The MC68HC05C4 then writes to its serial peripheral data register to initiate the transfer and shifts in data bits sent from the MC68705R3 until the SPIF bit goes high. While the MC68HC05C4 is writing to its SPDR, the MC68705R3 program is setting a bit counter to 8. When it detects a clock pulse on the SCK pin, the data register is rotated left one bit, placing the MSB in the carry. The MOSI pin is then set equal to the carry bit, the bit counter is decremented and, if it is greater than zero, the process is repeated.**
ADDITIONAL USES OF SPI

Many variations of this usage of the SPI are possible. The three possibilities are hardware SPI at both master and slave, software SPI at the master and hardware at the slave, and software SPI at both master and slave. Table 1 shows the various MCUs that have SPI implemented in hardware.

SPI is fairly straightforward in a circuit where both master and slave have hardware SPI capability. In this case, the MCUs are connected as shown in Figure 4. Figure 4a illustrates a single master system, and Figure 4b shows a system where either MCU can be system master. When both master and slave have SPI capability in hardware, data transfers can be handled full duplex. For a single master system, both master and slave write the data to be transferred to their respective serial peripheral data registers. A data transfer is initiated when the master writes to its serial peripheral data register. A slave device can shift data at a maximum rate equal to the CPU clock, so clock values must be chosen that allow the slave to transfer data at a rate equal to the master’s transfer rate. In a multiple master system, the master must pull the slave’s SS line low prior to writing to its serial peripheral data register and initiating the transfer.

PROGRAMMING A MASTER FOR SOFTWARE SPI

When the master in an SPI system does not have hardware SPI capabilities, the resulting system is quite different. An SPI system with a master providing the SPI in software is shown in Figure 5. This system only requires two lines between the microcomputers; data and clock. A slave select line can be added for use with multiple slaves. If operated with one data line, the SPI will function half-duplex only. Data is stored in a register, rotated left one bit at a time, and a port pin is set equal to the data bit. The master then provides the serial clock by toggling a different port pin. A bit counter must also be used to count the eight bits in the byte. Bit manipulation instructions are very useful for implementing SPI in software.

One possible software implementation for a write from the master to the slave is shown below.

DATA OUT PORTC pin 0
SCK PORTC pin 1
LDX #08 Bit counter
LDA DATA Put data in register A
RPT ROLA Shift a data bit into carry
BCS SET Check for a 1
BCLR 0,PORTC Set data out line to 0
CLK BSET 1,PORTC Toggle clock pin
BCLR 1,PORTC Check for end of byte
BNE RPT If not, repeat
 BCEX Set data out line to 1
SET BSET 0,PORTC Go to clock
BRA CLK

Full duplex operation requires a second data line. One port pin is then devoted to data-out and one to date-in. Data transfer from slave to master is accomplished immediately before the SCK pin is toggled. The state of the data-in pin is tested, and the carry is then set equal to the data-in pin. This value is then rotated in to a result register. The modified code is shown below.

DATA OUT PORTC pin 0
SCK PORTC pin 1
DATA IN PORTC pin 2
LDX #08 Bit counter
LDA DATA Put data in register A
BCLR 1,PORTC Clear clock pin
RPT ROLA Shift a data bit into carry
BCS SET Check for a 1
BCLR 0,PORTC Set data out line to 0
BSET 1,PORTC Set clock pin
DIN BRCLR 3,PORTC,CLK Check state of data
CLK ROL DATAIN Rotate input data one bit
DECX Check for end of byte
BNE RPT If not, repeat
SET BSET 0,PORTC Set data out line to 1
BRA DIN Go to data input

Figure 3. A/D Block Diagram
Figure 4a. Single Master SPI

Figure 4b. Multiple Master SPI

Figure 5. Software SPI
PROGRAMMING A SLAVE FOR SOFTWARE SPI

If the slave in the system is a MCU with hardware SPI capability, the data transfer will happen automatically, one bit per clock pulse. If the slave is a MCU that does not have SPI implemented in hardware, a read requires the following actions. A bit counter is set to eight, the slave polls its SCK pin waiting for a clock transition, once it perceives a clock it checks its data-in pin, sets the carry equal to the data and rotates the carry into a results register. One possible code implementation is shown in the previous timing example.

Converting this to full duplex operation requires the addition of a write from slave to master. The slave rolls a data register to place the data bit to be sent into the carry, and the data-out pin is set equal to the carry. These actions occur prior to the read of data from the master. With these modifications, the code looks as shown below.

```
DATA OUT PORTC pin 6
DATA IN PORTC pin 5
SCK PORTC pin 4
LDA #$08
STA BITCT
AGN BRSET 4, PORTC,* Set bit counter
ROI RES1
BCS SET1 Shift data to send
BCLR 6, PORTC
BCLR 4, PORTC,* Shift data to send
BRSET 5, PORTC, STR
STR ROL RESULT Check input data status
DEC BITCT
BNE AGN
```

DEBUGGING TIPS

Debugging a circuit containing two microcomputers presents various problems not evident when working with a single microcomputer circuit. The first problem is simultaneously providing emulation for both microcomputers. Once emulation capability is arranged, the designer needs to keep track of the progress of each single-chip, and monitor how the actions of one affects the actions of the other.

One of the easiest methods to debug a circuit of this type is to use two emulator stations, complete with separate terminals. Any emulators can be used, but user confusion is reduced if the emulators have similar commands and syntax. Physical separation also helps reduce confusion. It is somewhat easier to keep track of the concurrent operations if one side of the prototype board is devoted to each single-chip and the majority of peripherals they each must interface with, and the emulator for that microcomputer is placed to that side of the printed circuit board.

Before starting simultaneous debugging, it is best to individually debug the code for each microcomputer wherever possible. Once it becomes necessary for the microcomputers to communicate with one another, halt one of the microcomputers anytime they are not actually talking and work with the remaining microcomputer. As the debugging progresses, keep in mind that an error in the function of one single-chip does not necessarily indicate an error in the corresponding code for that single-chip, but rather, the error may have been caused by an incorrect or unintended transmission from the other single-chip.

Although the aforementioned suggestions reduce debugging problems, some will remain. Long periods of debug can result in an obscuring of the separation of the functions of the two programs. It helps to take periodic breaks to get away from the system and clear the thought processes. Expect to occasionally be confused, be willing to retrace sections of code multiple numbers of times, and the debugging will proceed fairly smoothly.

CONCLUSION

The Serial Peripheral Interface can be used as a tool to innerconnect to MCU with various other MCUs or peripherals, and can be used with any microcomputer. A special case occurs when one, or more, of the MCUs in a circuit do not have SPI capability in hardware. In this case, a simple software routine can be written to perform the SPI. Used in this manner, the SPI eliminates the need for costly, inconvenient parallel expansion buses and Universal Asynchronous Receiver/Transmitters (UARTs) and simplifies the design effort.
******* REGISTER ADDRESS DEFINITION *******

0006 0000 porta equ 0
0007 0002 portc equ 2
0008 0003 portd equ 3
0009 0004 ddra equ 4
0010 0006 ddrb equ 6
0011 000A spcr equ $0A
0012 000B spsr equ $0B
0013 000C spdr equ $0C
0014 0012 tcr equ $12

0017 0080 org $B0
0018
0019 0080 rwno rmb 1
0020 0081 tmpa rmb 1
0021 0082 dctr rmb 1
0022 0083 c1 rmb 1
0023 0084 base rmb 4
0024 0088 lsb rmb 1
0025 0089 msb rmb 1

0027 0020 org $20
0028
0029 ******* KEYPAD LOOKUP TABLE *******

0031 0020 kypd equ *

0032 0020 07 fcb $07
0033 0021 04 fcb $04
0034 0022 01 fcb $01
0035 0023 00 fcb $00
0036 0024 08 fcb $08
0037 0025 05 fcb $05
0038 0026 02 fcb $02
0039 0027 0A fcb $0A disp. temp.
0040 0028 09 fcb $09
0041 0029 06 fcb $06
0042 002A 03 fcb $03
0043 002B 0E fcb $0E set time
0044 002C 0D fcb $0D am
0045 002D 0C fcb $0C pm
0046 002E 0F fcb $0F disp. time
0047 002F 0B fcb $0B blank

0050 0100 org $100 program start
0051
0052 0100 9C start rsp
0053 0101 3F 12 clr tcr mask timer interrupts
0054 0103 AE 7B ldx #$7B
0055 0105 8F 02 stx portc initialize port c
0056 0107 AE 7F ldx #$7F
0057 0109 8F 0A stx spcr set spi cont. reg.
0058 010B 8F 06 stx ddrb set C0 as output
0059 010D 3F 00 clrrt porta clear keypad inputs
0060 010F A6 F0 lda #$F0 set up port a
0061 0111 B7 04 sta ddra a7-a4 out., a0-a3 in
0062 0113 9B sei

MOTOROLA AN991
** check keypad **

    key
    jsr keypad
    cmp #$0a  check for disp. temp
    beq dtmp
    cmp #$0e  check for set time
    beq sttm
    cmp #$0f  check for disp. time
    beq dtmp
    cmp #$0b  check for disp. sec
    beq dtmp
    bra key  wait for next input

** display temp **

    dtmp
    bclr $0, portc send interrupt for spi
    jsr spiwr  send byte
    cmp #$0a  check for disp. temp.
    beq cir
    bra key

    cir
    cli
    bra key

** set time **

    nudig
    jsr keypad
    cmp #$0a  check for valid digit
    beq nudig
    cmp #$0b  look for valid digit
    beq nudig
    cmp #$0e  check for pm
    beq nudig
    cmp #$0f  check for am
    beq nudig
    bra nudig

    sttm
    bclr $0, portc send int. for spi
    jsr spiwr  send value
    cmp #$0c  check for pm
    beq key  yes, wait for next input
    cmp #$0d  check for am
    beq key  yes, wait for next input
    bra nudig  get next time digit

** spi write subroutine **

    spiwr  equ *
    sta spdr  put data in data reg.
    brcr 7, spsr,*  wait for end of byte
    bset 0, portc
    spiflg  rts  done

** spi read subroutine **

    spird  equ *
    stx spdr  initiate transfer
    brcr 7, spsr,*  wait for end of byte
    rdend  rts
** keypad scanning routine **

keypad equ *

** 32 msec delay **

wtlp lda $20     set up outer loop
sta ct1         counter
otlp lda $32     set up inner loop
inlp deca dec. inner loop
           bne inlp when 0,
           dec ct1 decrement outer loop
           bne otlp
           cirx set up row counter
           lda $80 check first row
           sta porta

nxtr lda porta check for key
           and #$0f mask upper nibble
           cmp #$00 look for zero
           bne debnc branch if have a key
           asr porta try next row
           incx decrement row counter
           cpx #$03 check for zero
           bls nxtr test next row

invalid bra wt lp no key pressed

** debounce key input **

debnc sta tmpa save value
       stx rwno save row number
       lda #$ff set up delay

loop deca

           bne loop wait
           lda porta check row again
           and #$0f mask upper nibble
           cmp tmpa check for same key
           bne wt lp return if invalid

** wait for key release **

wtr lda porta check value
       and #$0f mask upper nibble
       cmp #$00 look for zero
       bne wtr wait for release

** decode key value **

lda tmpa restore value
           cirx set up column ctr.

nxtc isra shift columns
           bcs col branch if have column
           incx
           bra nxtc try next column

col lslx
           lslx x=4*col. no.
tax place x in a
add rwno key value =$4*col + row
       tax place a in x
       lda kypdx convert to decimal
       sta tmpa
       rts
****** temperature conversion routine ******

* Farenheit value is received from 705r3 via
* SPI and received in the A register, the value
* is converted to Celsius and the leftmost
* bit is blanked.

r3 int equ *
jsr spdrd read value
lda spdr transfer value to register
sub $20 subtract 32
bhs conv if pos. convert
nega negate
idx $0a
stx base+2 '/-' pattern

*** temperature conversion ***
** A 16-bit multiply by 5 is performed on the
** value received from the r3, this number
** is then divided by 9.

conv cir msb clear counters
      cir isb
sta isb
islx multiply by 2
rol msb load overflow into msb
islx multiply by 2
rol msb load overflow into msb
add isb a contains value x5
bcc div
inc msb if overflow, inc msb

div cir dctr
sta isb

inc dctr
sta isb

nxt9 inc dctr

sta isb
sbc $00 subtract borrow from msb
sta msb
lda isb count factors of 9
sub $09

bcc nxt9 if no borrow, repeat

stb msb if borrow, check for end

bne nxt9 repeat if not end

*** end of divide, add last 9 back in and
*** check remainder for rounding

add $09 find remainder
cmp $04 if greater
bhi done than 4, round up
dec dctr
done lda dctr

pos idx $0b blank pattern
stx base+3 blank most sig. digit

**** convert binary value to bcd value ****
* The x registers begins with the binary value
* and exits with zero, each digit, units, tens
* and hundreds, is stored separately and checked
* for a value equal to 10.

tax    place a into x
cira
clrbase+1 clear values
clrbase+2
st    tstx check for end
beq send if complete, send to r3
decx decrement hex number
incal increment decimal number
cmp #$0a equal to 10?
bne st no; keep going
incbase+1 increment tens
ldabase+1 test for 10
cmp #$0a
beq hund if equal, set hundreds

* send all digits to 705r3 via spi
* start by interrupting r3 and then
* sequentially sending four values

send    sta base store ones
beq blk
nxtdeg    lda base+x start at base
jsr spiwr send to r3
incx
cmpeq #$03 look for end
bne nxtdeg if no, next digit
rti

blk    lda #$0b
sta base+2
bra nxtdeg

*** initialize interrupt vectors ***

org $1ff4

spivec fdb start
sci vec fdb start
tmrvec fdb start
irqvec fdb r3int
swivec fdb start
reset fdb start
name r3disp

********** REGISTER DEFINITION **********

org $40

org $80

**** display look-up table ****

org $90  program start

*** initialize variables ***

start equ *

lda #$07
sta $f38  set MOR

lda #$ff
sta ddrb  set up port b as output

sta tdr   set timer for prescale of 128
sta base
sta base+1 blank time display
sta base+2
sta base+3
ida #Sef
sta portc  set portc to choose msg
sta segmnt
ida #Scf
sta ddr c  set up c0-3,c6,c7 as outputs
ida #S0f
sta tcr   unmask timer interrupt
clr tmtmp  start with time disp.
clr sec   set seconds to zero
clr pm    start with am
ida #S3b
sta ct    set up timing loops
ida #S08
sta ct1
cli

delay for

* delay for

 indexer #Sff
ida #Sff
deca
bne delay+4
decx
bne delay+2

temperature measurement *

clr acr   clear conv. complete flag
ida acr
bpl #-2   get result
ida arr
sbc #S30  adjust so 0 deg =$30
sta res1  store in spi data register
ida tmtmp
cmp #S07  check for temp. update
bne delay

*** send temperature value to hc05c4 for
conversion into celcius. start by
interrupting the hc05c4 and then transmit
data via the spi.

sei
bcir 7, portc interrupt hc05c4
jsr spiwr  write data to hc05c4
ida #S04

* wait for return data - 140 cycles *

lda #$0b
sta base+7
ida #$0e
tml p deca
bne timlp

* get decimal values in celcius from
* hc05c4

nxtlg jsr spi rd
ida result  get value
sta base,x store
incx
cpx #$07 check for end
bne nxtdg
cli
bra delay

** select temperature display **
temp lda #$07
sta ttmp choose temp. display
rti

****** spi routines ******
* the three pins used for the spi are
* miso bit 6, portc
* mosi bit 5, portc
* sck bit 4, portc
* the r3 waits for a high-to-low
* transition of the spi clock, which
* is provided by the h05c4 and sent
* on portc pin 4. a bit of data is
* transferred on each high-to-low
* transition of the clock.
*
* spi read *

spird equ *
lda #$08
sta bitct set bit counter
nxt brset 4, portc,* wait for clock transition
brset 5, portc,str check data status
*
* note: the brset command automatically
* sets the carry bit to be equal to the
* bit under test
*
str rol result store in result
lda #$02 delay loop
deaca
bne stall
nop
dec bitct check for end of byte
bne nxt get next bit
rts

* spi write *
* data to be sent is in result at
* start of write

spiwr equ *
lda #$08
sta bitct set bit counter
agn rol res1 shift data
bcsl set1 check data status
bclr 6, portc if 0, clear miso
bset 7, portc clear interrupt
bclr 4, portc nop
nop
tst brset 4, portc,* wait for clock trans.

0176 0118
0177 0118 A6 08
0178 011A B7 46
0179 011C 39 45
0180 011E 25 12
0181 0120 10 02
0182 0122 1E 02
0183 0124 19 02
0184 0126 90
0185 0127 90
0186 0128 08 02 FD
0187 012B 3A 46
0188 012D 26 ED
0189 012F 1E 02
0190 0131 81
0191
0192 0132 1C 02
0193 0134 1E 02
0194 0136 18 02
0195 0138 20 EE
0196
0197 ** initialization of data read via SPI **
0198 *
0199 * a data read is initiated via an interrupt
0200 * from the HC05C4, the value received is
0201 * tested to determine which function is
0202 * requested and the processor jumps to the
0203 * proper routine.
0204 *
0205 0206 013A CD 01 01
0207 013D A6 03
0208 013F B7 41
0209 0141 B6 44
0210 0143 A1 0A
0211 0145 27 B5
0212 0147 A1 0F
0213 0149 27 3C
0214 014B A1 0E
0215 014D 27 39
0216 014F A1 0D
0217 0151 27 0C
0218 0153 A1 0B
0219 0155 27 6A
0220 0157 A1 0C
0221 0159 26 39
0222 015B A6 FF
0223 015D B7 49
0224
0225 ** check for valid input **
0226
0227 015F B6 4D
0228 0161 27 0B
0229 0163 A1 01
0230 0165 27 44
0231 0167 A1 0B
0232 0169 26 48
0233 016B A6 0B
0234 016D B7 40
0235 016F B6 4B
0236 0171 A1 05
0237 0173 22 3E
0238
0239
0240
0241 0175 A6 DF
0242 0177 B7 09
0243 0179 A6 43
0244 017B B7 42
0245 017D A6 04
0246 017F B7 43
0247 0181 3F 47
0248 0183 3F 52

Acknowledgments: This document is an excerpt from a larger work. The original content is copyrighted and published by Motorola. For more information, visit their official website or contact the publisher directly.
0249 0185 3F 53
0250 0187 80
0251
0252
0253
0254 0186 A6 0B
0255 018A B7 4A
0256 018C B7 4B
0257 018E B7 4C
0258 0190 B7 4D
0259 0192 20 F3
0260
0261
0262
0263
0264
0265
0266
0267
0268
0269
0270
0271
0272
0273
0274
0275 0194 9A
0276 0195 B6 4C
0277 0197 B7 4D
0278 0199 B6 4B
0279 019B B7 4C
0280 019D B6 4A
0281 019F B7 4B
0282 01A1 B6 44
0283 01A3 B7 4A
0284 01A5 A1 09
0285 01A7 22 DE
0286 01A9 B7 4A
0287 01AB 20 DA
0288
0289
0290
0291
0292 01AD B6 4C
0293 01AF A1 02
0294 01B1 23 BC
0295 01B3 3F 4D
0296 01B5 3F 4C
0297 01B7 A6 0D
0298 01B9 B7 4B
0299 01BB A6 05
0300 01BD B7 4A
0301 01BF 20 C6
0302
0303
0304
0305
0306 01C1 B7 41
0307 01C3 B7 55
0308 01C5 B7 54
0309 01C7 80
0310

clr base+9
rtry rti

* clear displays *
clrtn lda #$0b
sta base
sta base+1
sta base+2
sta base+3
bra rtry

* input digit *

*** time setting routine ***
* time is inputted left to right
* and the end of input is indicated
* be pressing either the am or pm
* button. pm is denoted on the
* display by lighting the decimal
* point. counters are set to zero out
* after each second.

dig cli
lda base+2
sta base+3
lda base+1
sta base+2
lda base
sta base+1
lda result
sta base
cmp #$09
bhi rtry
sta base
bra rtry

* check if time less than 12 o'clock
* blank display if not

twoc lda base+2
cmp #$02
bcs mtn

blank lcr base+3
cr base+2
lda #$0d
sta base+1
lda #$05
sta base
bra rtry

**** seconds display ****
* blank first two leds

dsec sta timtmp
set timtmp to $0b
sta base+$b
blank 1st two leds
sta base+$a
rti
**** display routine ****
*
* displays are refreshed every msec
* when a timer interrupt occurs. the
* most significant digit is displayed
* first. at the conclusion of each
* minute, the time is updated
*
	tmint equ *

	sec
	ldx timp    choose time or temp 
	lda #$ff    blank displays
	sta portb   send to leds
	ror segmnt  select display
	bcs min2    look for restart

	lda #$f7    restart with msd
	sta segmnt
	min2
	lda base+x  load a with minutes
	decx  point to next digit
	bset 1,segmnt,hrs1 check hours units
	lda base+x  load a with tens of min.
	hrs1
	decx  point to next digit
	bset 2,segmnt,hrs2 check tens of hrs.
	lda base+x  load a with hours units
	hrs2
	decx  point to next digit
	bset 3,segmnt,disp display value
	disp
	lda base+x  load a with tens of hrs
	and #$0f    mask upper nibble
	tax    set x equal to a
	dex segtab,x display value table

tst portb enable display drivers
	lda segmnt
	sta portc enable display

** count display refreshes. 402 refreshes
* equals one second. after 402 refreshes,
* update clock
*
	lda #$10    set timer to interrupt
	sta tdr    after 2048 cycles
	lda #$0f    reset timer interrupt flag

dec ct    decrement inner loop
	nbe ret    reset inner loop
	lda #$3b    decrement outer loop

dec cl    if one sec., to time change


ret
rti

**** time change routine ****
*
* when 60 seconds are counted,
* increase minutes by one, if
* necessary, blank minutes and increase
* hours. change am/pm if needed.
*
tmchg equ *
inc sec increase seconds
inc base+8 inc. secs. units
lda base+8
cmp #00a look for ten
beq tens if yes, inc. tens
minck lda #3c
lda base+9
cmp #09 less than 9?
beq inm1 increase
clr base min. units = 0
lda base+1 check tens of min.
cmp #05 less than 5?
beq inm2 increase
clr base+1 tens of min = 0
lda base+3 check tens of hrs.
cmp #0b look for blank
beq hrck less than 10:00
lda base+2 check hrs. units
cmp #02 less than 2?
beq inhr1a increase
lda base+0
sta base+3 set time to 1:00
lda #01
sta base+2
bra ret1 done
inm1 inc base increase min. units
bra ret1 done
inm2 inc base+1 increase tens of min.
bra ret1

tens clr base+8 zero sec. units
inc base+9 inc sec. tens
bra minck

* increase hours *

hrck lda base+2 check hours units
cmp #09 less than 9?
bne inhr1 increase
clr base+1 hours units = 0
clr base+3
inc base+3 tens of hours = 1
bra ret1 done
inhr1 inc base+2 increase hours units
bra ret1 done
inhr1a inc base+2 increase hours units
lda base+2 check value
cmp #02 for 12:00
bne ret no, done
com pm
ret1 lda #3b
sta ct reset inner loop counter
lda #0b
sta ct1 reset outer loop counter
rtil
** initialize interrupt vectors **

```
0435
0436
0437
0438
0439
0440
0441
0442
0443

.38 OxFFB
.39
.40 OxFFB 01 C8
.41 OxFFA 01 3A
.42 OFFC 00 90
.43 OFFE 00 90

tmrvec fdb tmrint
intvec fdb c4int
swiveq fdb start
reset fdb start
```

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

**Literature Distribution Centers:**

JSA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.
SIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.