MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

Order this document
by AN991/D

AN991

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 V_{DD} 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.

	7	6	5	4	3	2	1	0	
$0A	SPIE	SPE	–	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
RESET	0	0	0	0	0	1	U	U	

SPIE—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

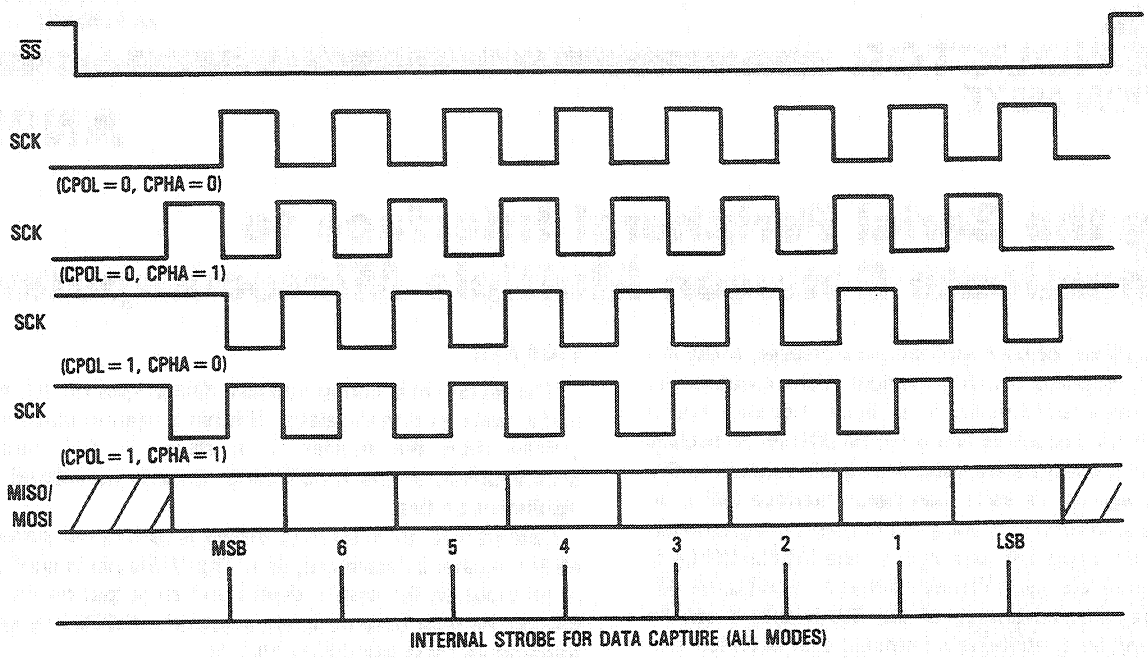

Figure 1. Data Clock Timing Diagram

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 $\overline{\text{SS}}$. As soon as $\overline{\text{SS}}$ goes low the transaction begins and the first edge on SCK involves the first data sample. When CPHA = 1, the $\overline{\text{SS}}$ 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 1. Serial Peripheral Rate Selection

SPR1	SPR0	Internal Processor Clock Divide By
0	0	2
0	1	4
1	0	16
1	1	32

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 farenheit to the MC68HC05C4, where the value is converted to a celcius 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 farenheit. 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 farenheit. Using a V_{CC} of 5 volts, the maximum differential amplifier output is 3.8 volts, resulting in a temperature sensing range from −40 degrees to +140 degrees farenheit.

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 convertor returns a value of \$30 when given an input of zero degrees farenheit, so \$30 must be subtracted from the result before converting to celcius. 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 result 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.

```
DATA IN     PORTC pin 5
SCK         PORTC pin 4

Cycles              Instruction

  2           LDA     #$08
  5           STA     BITCT            Set bit counter
 10     NXT   BRSET   4,PORTC,*        Wait for clock transition
 10           BRSET   5,PORTC,STR      Check data status
  6     STR   ROL     RESULT           Store in result
  6           DEC     BITCT            Check for end of byte
  4           BNE     NXT              Get next bit
 43
```

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.

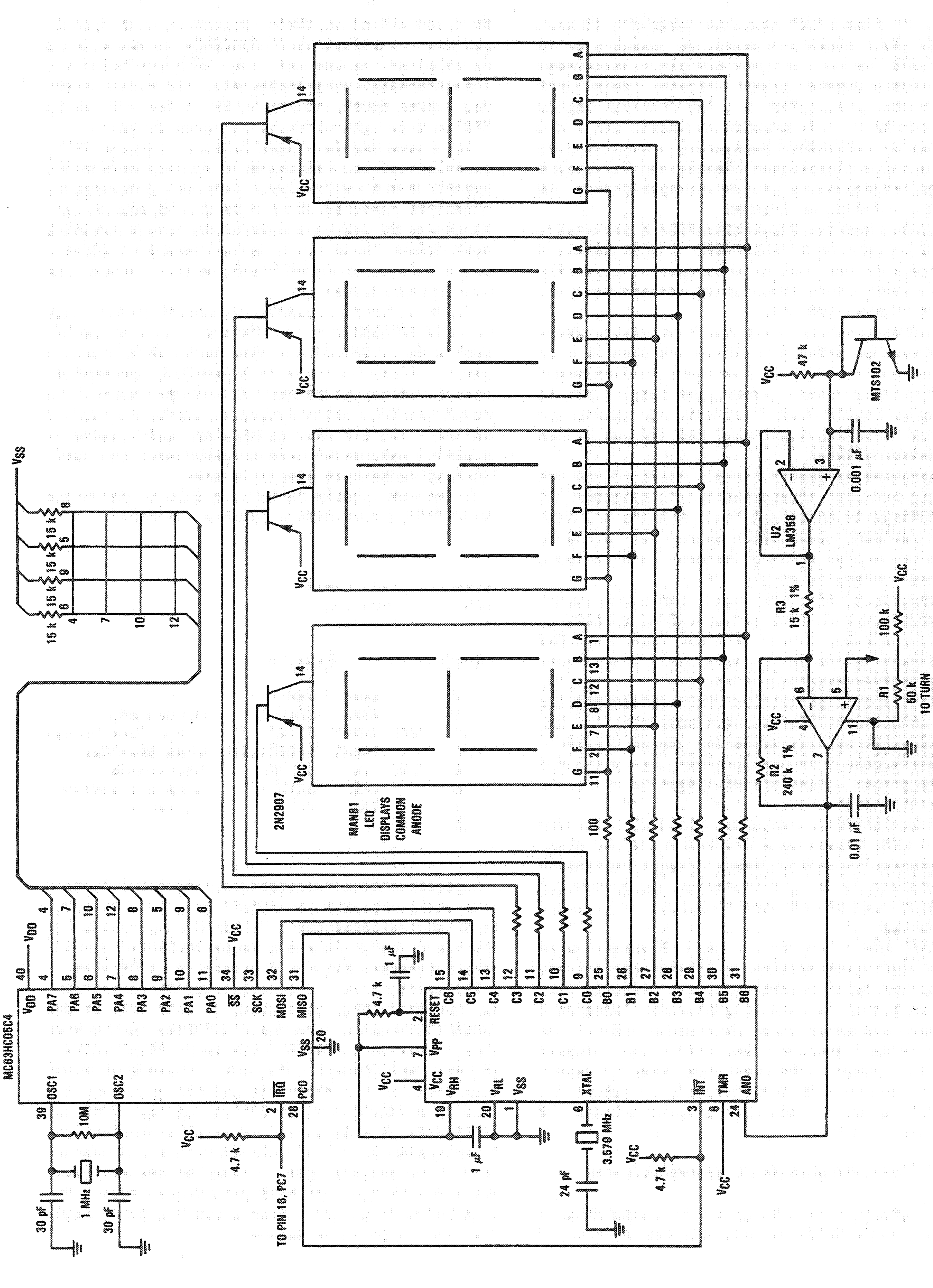

Figure 2. Serial Peripheral Interface Demonstration Schematic

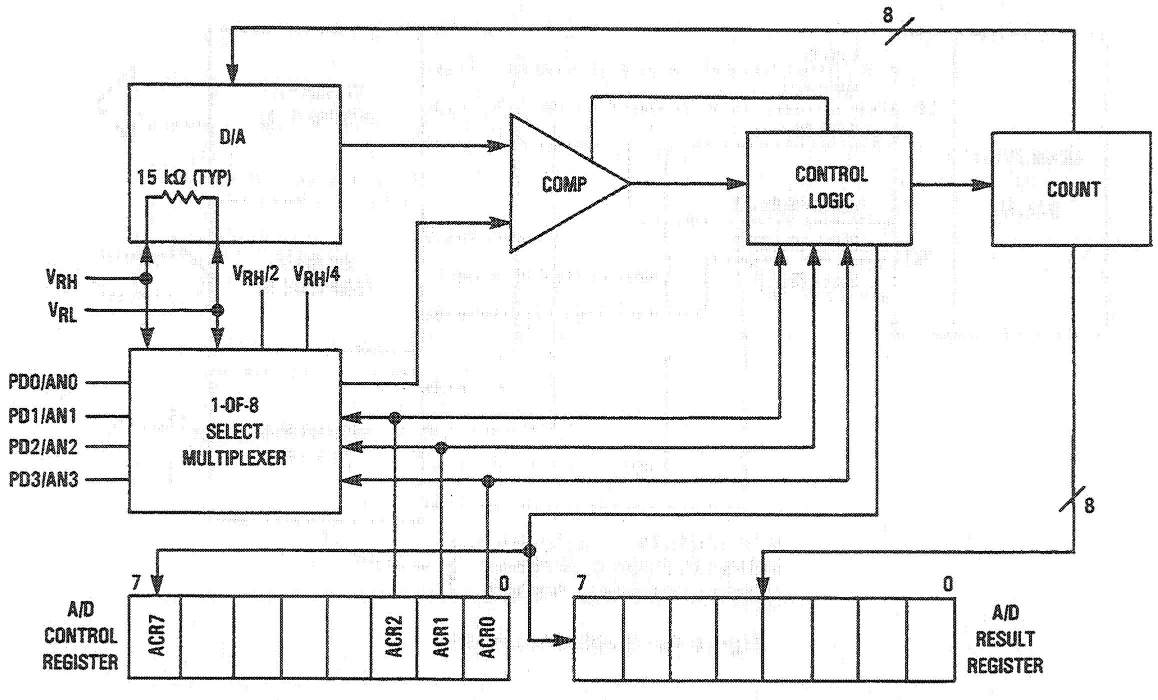

Figure 3. A/D Block Diagram

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
                  BCLR 1,PORTC      Toggle clock pin
                  DECX              Check for end of byte
                  BNE RPT           If not, repeat

            SET   BSET 0,PORTC      Set data out line to 1
                  BRA CLK           Go to clock
```

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
```

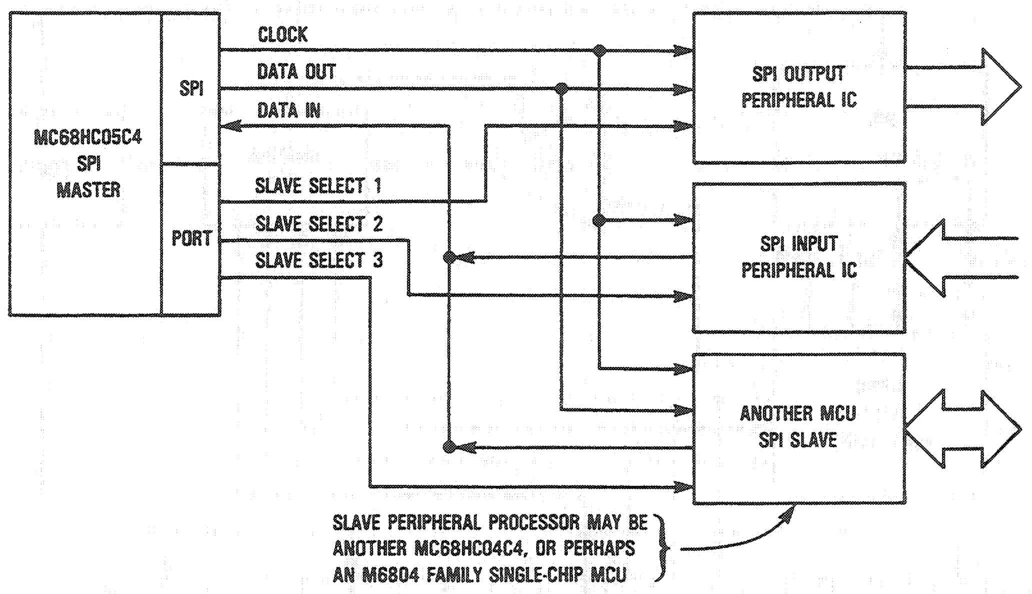

Figure 4a. Single Master SPI

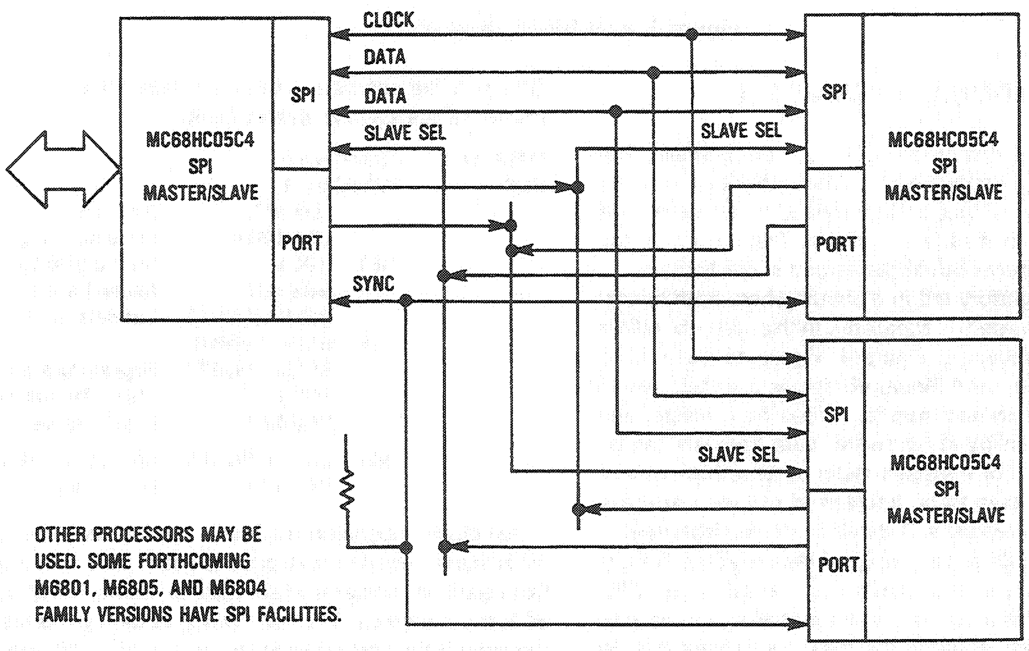

Figure 4b. Multiple Master SPI

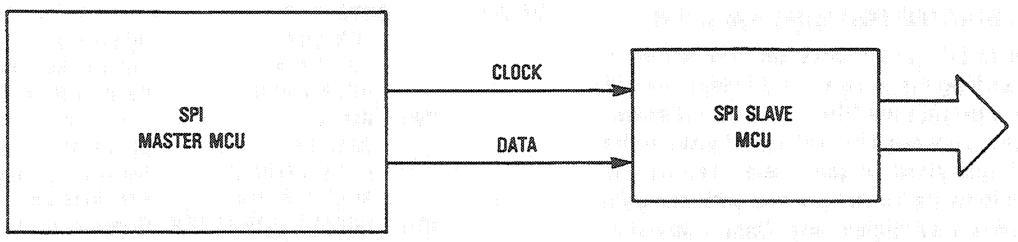

Figure 5. Software SPI

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           Set bit counter
        AGN     BRSET   4,PORTC,*       Wait for clock
                ROL     RES1            Shift data to send
                BCS     SET1            Check data status
                BCLR    6,PORTC         If 0, clear data out
                BRCLR   4,PORTC,*       Wait for clock transition
                BRSET   5,PORTC,STR     Check input data status
        STR     ROL     RESULT          Store in result
                DEC     BITCT           Check for end of byte
                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.

```
                          nam spicnt

                     ********** REGISTER ADDRESS DEFINITION **********

0000                 porta   equ    0
0002                 portc   equ    2
0003                 portd   equ    3
0004                 ddra    equ    4
0006                 ddrc    equ    6
000A                 spcr    equ    $0a
000B                 spsr    equ    $0b
000C                 spdr    equ    $0c
0012                 tcr     equ    $12


00B0                         org $b0

00B0                 rwno    rmb 1
00B1                 tmpa    rmb 1
00B2                 dctr    rmb 1
00B3                 ct1     rmb 1
00B4                 base    rmb 4
00B8                 lsb     rmb 1
00B9                 msb     rmb 1

0020                         org    $20

                     ******* KEYPAD LOOKUP TABLE *******

0020                 kypd    equ    *

0020 07                      fcb    $07
0021 04                      fcb    $04
0022 01                      fcb    $01
0023 00                      fcb    $00
0024 08                      fcb    $08
0025 05                      fcb    $05
0026 02                      fcb    $02
0027 0A                      fcb    $0a       disp. temp.
0028 09                      fcb    $09
0029 06                      fcb    $06
002A 03                      fcb    $03
002B 0E                      fcb    $0e       set time
002C 0D                      fcb    $0d       am
002D 0C                      fcb    $0c       pm
002E 0F                      fcb    $0f       disp. time
002F 0B                      fcb    $0b       blank

0100                         org $100 program start

0100 9C              start    rsp
0101 3F 12                    clr tcr    mask timer interrupts
0103 AE 7B                    ldx #$7b
0105 BF 02                    stx portc     initialize port c
0107 AE 7F                    ldx #$7f
0109 BF 0A                    stx spcr   set spi cont. reg.
010B BF 06                    stx ddrc     set c0 as output
010D 3F 00                    clr porta    clear keypad inputs
010F A6 F0                    lda #$f0     set up port a
0111 B7 04                    sta ddra     a7-a4 out., a0-a3 in
0113 9B                       sei
```

```


                     ** check keypad **

0114 CD 01 67        key     jsr keypad
0117 A1 0A                   cmp #$0a     check for disp. temp
0119 27 0E                   beq dtmp
011B A1 0E                   cmp #$0e     check for set time
011D 27 2B                   beq sttm
011F A1 0F                   cmp #$0f     check for disp. time
0121 27 06                   beq dtmp
0123 A1 0B                   cmp #$0b     check for disp. sec
0125 27 02                   beq dtmp
0127 20 EB                   bra key      wait for next input

                     ** display temp **

0129 11 02           dtmp    bclr 0,portc send interrupt for spi
012B CD 01 59                jsr spiwr    send byte
012E A1 0A                   cmp #$0a     check for disp. temp.
0130 27 02                   beq clr
0132 20 E0                   bra key

0134 9A              clr     cli
0135 20 DD                   bra key

                     ** set time **
0137 CD 01 67        nudig   jsr keypad
013A A1 0A                   cmp #$0a
013C 27 F9                   beq nudig
013E A1 0B                   cmp #$0b
0140 27 F5                   beq nudig    look for valid digit
0142 A1 0E                   cmp #$0e
0144 27 F1                   beq nudig
0146 A1 0F                   cmp #$0f
0148 27 ED                   beq nudig
014A 11 02           sttm    bclr 0,portc send int. for spi
014C CD 01 59                jsr spiwr    send value
014F A1 0C                   cmp #$0c     check for pm
0151 27 C1                   beq key      yes, wait for next input
0153 A1 0D                   cmp #$0d     check for am
0155 27 BD                   beq key      yes, wait for next input
0157 20 DE                   bra nudig    get next time digit

                     ** spi write subroutine **

0159                 spiwr   equ *
0159 B7 0C                   sta spdr     put data in data reg.
015B 0F 0B FD                brclr 7,spsr,*   wait for end of byte
015E 10 02                   bset 0,portc
0160 81              spiflg  rts          done

                     ** spi read subroutine **

0161                 spird   equ *
0161 BF 0C                   stx spdr     initiate transfer
0163 0F 0B FD                brclr 7,spsr,*  wait for end of byte
0166 81              rdend   rts

```

```
                    ** keypad scanning routine **

0167                keypad  equ *

                    ** 32 msec delay **

0167 A6 20          wtlp    lda #$20      set up outer loop
0169 B7 B3                  sta ct1       counter
016B A6 32          otlp    lda #$32      set up inner loop
016D 4A             inlp    deca    dec.  inner loop
016E 26 FD                  bne inlp      when 0,
0170 3A B3                  dec ct1       decrement outer loop
0172 26 F7                  bne otlp
0174 5F                     clrx          set up row counter
0175 A6 80                  lda #$80      check first row
0177 B7 00                  sta porta
0179 B6 00          nxtr    lda porta     check for key
017B A4 0F                  and #$0f      mask upper nibble
017D A1 00                  cmp #$00      look for zero
017F 26 09                  bne debnc     branch if have a key
0181 37 00                  asr porta     try next row
0183 5C                     incx          decrement row counter
0184 A3 03                  cpx #$03      check for zero
0186 23 F1                  bls nxtr      test next row
0188 20 DD          inval   bra wtlp      no key pressed

                    ** debounce key input **

018A B7 B1          debnc   sta tmpa      save value
018C BF B0                  stx rwno      save row number
018E A6 FF                  lda #$ff      set up delay
0190 4A             loop    deca
0191 26 FD                  bne loop      wait
0193 B6 00                  lda porta     check row again
0195 A4 0F                  and #$0f      mask upper nibble
0197 B1 B1                  cmp tmpa      check for same key
0199 26 CC                  bne wtlp      return if invalid

                    ** wait for key release **

019B B6 00          wtr     lda porta     check value
019D A4 0F                  and #$0f      mask upper nibble
019F A1 00                  cmp #$00      look for zero
01A1 26 F8                  bne wtr       wait for release

                    ** decode key value **

01A3 B6 B1                  lda tmpa      restore value
01A5 5F                     clrx          set up column ctr.
01A6 44             nxtc    lsra          shift columns
01A7 25 03                  bcs col       branch if have column
01A9 5C                     incx
01AA 20 FA                  bra nxtc      try next column
01AC 58             col     lslx
01AD 58                     lslx          x=4*col. no.
01AE 9F                     txa           place x in a
01AF BB B0                  add rwno      key value =4*col + row
01B1 97                     tax           place a in x
01B2 E6 20                  lda kypd,x    convert to decimal
01B4 B7 B1                  sta tmpa
01B6 81                     rts

```

```
                        ****** temperature conversion routine ********
                        *
                        * farenheit value is received from 705r3 via
                        * spi and received in the a register.  the value
                        * is converted to celcius and the leftmost
                        * led is blanked.

01B7                    r3int   equ *
01B7 CD 01 61                   jsr spird      read value
01BA B6 0C                      lda spdr       transfer value to register
01BC A0 20                      sub #$20       subtract 32
01BE 24 05                      bhs conv       if pos. convert
01C0 40                         nega           negate
01C1 AE 0A                      ldx #$0a
01C3 BF B6                      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.

01C5 3F B9              conv    clr msb        clear counters
01C7 3F B8                      clr lsb
01C9 B7 B8                      sta lsb
01CB 48                         lsla           multiply by 2
01CC 39 B9                      rol msb        load overflow into msb
01CE 48                         lsla           multiply by 2
01CF 39 B9                      rol msb        load overflow into msb
01D1 BB B8                      add lsb        a contains value x5
01D3 24 02                      bcc div
01D5 3C B9                      inc msb        if overflow, inc msb
01D7 3F B2              div     clr dctr
01D9 B7 B8                      sta lsb
01DB 98                         clc
01DC 3C B2              nxt9    inc dctr
01DE B7 B8                      sta lsb
01E0 B6 B9                      lda msb
01E2 A2 00                      sbc #$00       subtract borrow from msb
01E4 B7 B9                      sta msb
01E6 B6 B8                      lda lsb        count factors of 9
01E8 A0 09                      sub #$09
01EA 24 F0                      bcc nxt9       if no borrow, repeat
01EC 3D B9                      tst msb        if borrow, check for end
01EE 26 EC                      bne nxt9       repeat if not end


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

01F0 AB 09                      add #$09       find remainder
01F2 A1 04                      cmp #$04       if greater
01F4 22 02                      bhi done       than 4, round up
01F6 3A B2                      dec dctr
01F8 B6 B2              done    lda dctr

01FA AE 0B              pos     ldx #$0b       blank pattern
01FC BF B7                      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.

01FE 97                          tax            place a into x
01FF 4F                          clra
0200 3F B5                       clr base+1     clear values
0202 3F B6                       clr base+2
0204 5D                  st      tstx           check for end
0205 27 17                       beq send       if complete, send to r3
0207 5A                          decx           decrement hex number
0208 4C                          inca           increment decimal number
0209 A1 0A                       cmp #$0a       equal to 10?
020B 26 F7                       bne st         no, keep going
020D 3C B5                       inc base+1     increment tens
020F B6 B5                       lda base+1     test for 10
0211 A1 0A                       cmp #$0a
0213 27 03                       beq hund       if equal, set hundreds
0215 4F                  zero    clra           clear ones
0216 20 EC                       bra st         count next 10
0218 3F B5               hund    clr base+1     clear tens
021A 3C B6                       inc base+2     increment hundreds
021C 20 F7                       bra zero

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

021E B7 B4               send    sta base       store ones
0220 B6 B6                       lda base+2
0222 27 0B                       beq blk
0224 E6 B4               nxtdg   lda base,x     start at base
0226 CD 01 59                    jsr spiwr      send to r3
0229 5C                          incx
022A A3 03                       cpx #$03       look for end
022C 26 F6                       bne nxtdg      if no, next digit
022E 80                          rti
022F A6 0B               blk     lda #$0b
0231 B7 B6                       sta base+2
0233 20 EF                       bra nxtdg


                         *** initialize interrupt vectors ***

1FF4                             org $1ff4

1FF4 01 00               spivec  fdb start
1FF6 01 00               scivec  fdb start
1FF8 01 00               tmrvec  fdb start
1FFA 01 B7               irqvec  fdb r3int
1FFC 01 00               swivec  fdb start
1FFE 01 00               reset   fdb start
```

```

                         nam r3disp

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

0001                    portb  equ   1
0002                    portc  equ   2
0005                    ddrb   equ   5
0006                    ddrc   equ   6
0008                    tdr    equ   8
0009                    tcr    equ   9
000E                    acr    equ  14
000F                    arr    equ  15


0040                           org $40


0040                    wrdat  rmb  1
0041                    timtmp rmb  1
0042                    ct     rmb  1
0043                    ct1    rmb  1
0044                    result rmb  1
0045                    res1   rmb  1
0046                    bitct  rmb  1
0047                    sec    rmb  1
0048                    segmnt rmb  1
0049                    pm     rmb  1
004A                    base   rmb  4

0080                           org $80

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

0080                    segtab equ *
0080 01                        fcb %00000001 0
0081 4F                        fcb %01001111 1
0082 12                        fcb %00010010 2
0083 06                        fcb %00000110 3
0084 4C                        fcb %01001100 4
0085 24                        fcb %00100100 5
0086 20                        fcb %00100000 6
0087 0F                        fcb %00001111 7
0088 00                        fcb %00000000 8
0089 0C                        fcb %00001100 9
008A 7E                        fcb %01111110 -
008B 7F                        fcb %01111111 blank
008C 7F                        fcb %01111111 pm
008D 18                        fcb %00011000 p

0090                           org $90   program start

                        *** initialize variables ***

0090                    start  equ *
0090 A6 07                     lda #$07
0092 C7 0F 38                  sta $f38        set MOR
0095 A6 FF                     lda #$ff
0097 B7 05                     sta ddrb        set up port b as output
0099 B7 08                     sta tdr         set timer for prescale of 128
009B B7 4A                     sta base
009D B7 4B                     sta base+1      blank time display
```

```
009F B7 4C                  sta base+2
00A1 B7 4D                  sta base+3
00A3 A6 EF                  lda #$ef
00A5 B7 02                  sta portc     set portc to choose msd
00A7 B7 48                  sta segmnt
00A9 A6 CF                  lda #$cf
00AB B7 06                  sta ddrc      set up c0-3,c6,c7 as outputs
00AD A6 0F                  lda #$0f
00AF B7 09                  sta tcr       unmask timer interrupt
00B1 3F 41                  clr timtmp    start with time disp.
00B3 3F 47                  clr sec       set seconds to zero
00B5 3F 49                  clr pm        start with am
00B7 A6 3B                  lda #$3b
00B9 B7 42                  sta ct        set up timing loops
00BB A6 08                  lda #$08
00BD B7 43                  sta ct1
00BF 9A                     cli

                     *       delay for

00C0 AE FF           dlay   ldx #$ff
00C2 A6 FF                  lda #$ff
00C4 4A                     deca
00C5 26 FD                  bne dlay+4
00C7 5A                     decx
00C8 26 F8                  bne dlay+2

                     *       temperature measurement *

00CA 3F 0E                  clr acr        clear conv. complete flag
00CC B6 0E                  lda acr
00CE 2A FC                  bpl *-2
00D0 B6 0F                  lda arr       get result
00D2 A2 30                  sbc #$30        adjust so 0 deg =$30
00D4 B7 45                  sta res1     store in spi data register
00D6 B6 41                  lda timtmp
00D8 A1 07                  cmp #$07     check for temp. update
00DA 26 E4                  bne dlay

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

00DC 9B                     sei
00DD 1F 02                  bclr 7,portc interrupt hc05c4
00DF CD 01 18               jsr spiwr       write data to hc05c4
00E2 AE 04                  ldx #$04

                     *       wait for return data ~ 140 cycles *

00E4 A6 0B                  lda #$0b
00E6 B7 51                  sta base+7
00E8 A6 0E                  lda #$0e
00EA 4A              timlp  deca
00EB 26 FD                  bne timlp

                     *       get decimal values in celcius from
                     *       hc05c4

00ED CD 01 01        nxtdg  jsr spird
00F0 B6 44                  lda result      get value
```

```
00F2 E7 4A                      sta base,x      store
00F4 5C                         incx
00F5 A3 07                      cpx #$07        check for end
00F7 26 F4                      bne nxtdg
00F9 9A                         cli
00FA 20 C4                      bra dlay

                        **       select temperature display **

00FC A6 07              temp    lda #$07
00FE B7 41                      sta timtmp     choose temp. display
0100 80                         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 hc05c4 and sent
                        *       on portc pin 4.  a bit of data is
                        *       transferred on each high-to-low
                        *       transition of the clock.

                        *       spi read *

0101                    spird   equ *
0101 A6 08                      lda #$08
0103 B7 46                      sta bitct       set bit counter
0105 08 02 FD           nxt     brset 4,portc,* wait for clock transition
0108 0A 02 00                   brset 5,portc,str check data status
                        *
                        *       note:  the brset command automatically
                        *       sets the carry bit to be equal to the
                        *       bit under test
                        *
010B 39 44              str     rol result   store in result
010D A6 02                      lda #$02        delay loop
010F 4A                 stall   deca
0110 26 FD                      bne stall
0112 9D                         nop
0113 3A 46                      dec bitct       check for end of byte
0115 26 EE                      bne nxt         get next bit
0117 81                         rts

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

0118                    spiwr   equ *
0118 A6 08                      lda #$08
011A B7 46                      sta bitct       set bit counter
011C 39 45              agn     rol res1        shift data
011E 25 12                      bcs set1        check data status
0120 1D 02                      bclr 6,portc    if 0, clear miso
0122 1E 02                      bset 7,portc    clear interrupt
0124 19 02                      bclr 4,portc
0126 9D                         nop
0127 9D                         nop             timing delay.
0128 08 02 FD           tst     brset 4,portc,* wait for clock trans.
```

```
012B 3A 46                      dec bitct       check for end of byte
012D 26 ED                      bne agn
012F 1E 02                      bset 7,portc    clear interrupt
0131 81                         rts

0132 1C 02              set1    bset 6,portc    if 1, set miso
0134 1E 02                      bset 7,portc    clear interrupt
0136 18 02                      bset 4,portc
0138 20 EE                      bra tst

                        **        initialization of data read via spi **
                        *
                        *        a data read is initiated via an interrupt
                        *        from the hc05c4.  the value received is
                        *        tested to determine which function is
                        *        requested and the processor jumps to the
                        *        proper routine.
                        *

013A CD 01 01           c4int   jsr spird       get value
013D A6 03                      lda #$03
013F B7 41                      sta timtmp      choose time
0141 B6 44                      lda result
0143 A1 0A                      cmp #$0a        check for disp temp
0145 27 B5                      beq temp
0147 A1 0F                      cmp #$0f        check for display time
0149 27 3C                      beq rtry
014B A1 0E                      cmp #$0e        check for set time
014D 27 39                      beq clrtm
014F A1 0D                      cmp #$0d        check for am
0151 27 0C                      beq am
0153 A1 0B                      cmp #$0b        check for secs
0155 27 6A                      beq dsec
0157 A1 0C                      cmp #$0c        check for pm
0159 26 39                      bne dig         no, set digit
015B A6 FF                      lda #$ff        set pm address
015D B7 49                      sta pm

                        **       check for valid input **

015F B6 4D              am      lda base+3      check tens of hours
0161 27 08                      beq blhr        if zero, blank digit
0163 A1 01                      cmp #$01
0165 27 46                      beq twoc
0167 A1 0B                      cmp #$0b        look for blank
0169 26 48                      bne blank       if not, blank display
016B A6 0B              blhr    lda #$0b
016D B7 4D                      sta base+3      blank tens of hours
016F B6 4B              mtn     lda base+1      check tens of minutes
0171 A1 05                      cmp #$05        check against 5
0173 22 3E                      bhi blank       if greater, blank display

                        *        valid input, set timer counter *

0175 A6 0F                      lda #$0f
0177 B7 09                      sta tcr         unmask timer interrupt
0179 A6 43                      lda #$43
017B B7 42                      sta ct          load inner loop counter
017D A6 06                      lda #$06
017F B7 43                      sta ct1         load outer loop counter
0181 3F 47                      clr sec
0183 3F 52                      clr base+8
```

```
0185 3F 53                 clr base+9
0187 80            rtry    rti

                   *        clear displays *

0188 A6 0B         clrtm    lda #$0b
018A B7 4A                 sta base
018C B7 4B                 sta base+1
018E B7 4C                 sta base+2
0190 B7 4D                 sta base+3
0192 20 F3                 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.
                   *


0194 9A            dig     cli
0195 B6 4C                 lda base+2
0197 B7 4D                 sta base+3
0199 B6 4B                 lda base+1       shift data left one
019B B7 4C                 sta base+2       digit
019D B6 4A                 lda base
019F B7 4B                 sta base+1
01A1 B6 44                 lda result       enter digit 1
01A3 B7 4A                 sta base
01A5 A1 09                 cmp #$09         check for valid digit
01A7 22 DE                 bhi rtry
01A9 B7 4A                 sta base
01AB 20 DA                 bra rtry         get next number

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

01AD B6 4C         twoc    lda base+2       check hours units
01AF A1 02                 cmp #$02
01B1 23 BC                 bls mtn          okay, check tens of min.
01B3 3F 4D         blank   clr base+3
01B5 3F 4C                 clr base+2
01B7 A6 0D                 lda #$0d
01B9 B7 4B                 sta base+1       send error message
01BB A6 05                 lda #$05
01BD B7 4A                 sta base
01BF 20 C6                 bra rtry

                   **** seconds display *****
                   *  blank first two leds
                   *
01C1 B7 41         dsec    sta timtmp       set timtmp to $0b
01C3 B7 55                 sta base+$b      blank 1st two leds
01C5 B7 54                 sta base+$a
01C7 80                    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
                    *

01C8                tmrint equ *

01C8 99                     sec
01C9 BE 41                  ldx timtmp      choose time or temp
01CB A6 FF                  lda #$ff        blank displays
01CD B7 01                  sta portb       send to leds
01CF 36 48                  ror segmnt      select display
01D1 25 04                  bcs min2        look for restart


01D3 A6 F7                  lda #$f7        restart with msd
01D5 B7 48                  sta segmnt
01D7 E6 4A          min2    lda base,x      load a with minutes
01D9 5A                     decx            point to next digit
01DA 02 48 02               brset 1,segmnt,hrs1  check hours units
01DD E6 4A                  lda base,x      load a with tens of min.
01DF 5A             hrs1    decx            point to next digit
01E0 04 48 02               brset 2,segmnt,hrs2 check tens of hrs.
01E3 E6 4A                  lda base,x      load a with hours units
01E5 5A             hrs2    decx            point to next digit
01E6 06 48 02               brset 3,segmnt,disp display value
01E9 E6 4A                  lda base,x      load a with tens of hrs
01EB A4 0F          disp    and #$0f        mask upper nibble
01ED 97                     tax             set x equal to a
01EE EE 80                  ldx segtab,x    display value table
01F0 BF 01                  stx portb       enable display drivers
01F2 B6 48                  lda segmnt
01F4 B7 02                  sta portc       enable display

                    **       count display refreshes.  402 refreshes
                    *       equals one second.  after 402 refreshes,
                    *       update clock
                    *

01F6 A6 10                  lda #$10        set timer to interrupt
01F8 B7 08                  sta tdr         after 2048 cycles
01FA A6 0F                  lda #$0f
01FC B7 09                  sta tcr         reset timer interrupt flag
01FE 3A 42                  dec ct          decrement inner loop
0200 26 08                  bne ret
0202 A6 3B                  lda #$3b        reset inner loop
0204 B7 42                  sta ct
0206 3A 43                  dec ct1         decrement outer loop
0208 27 01                  beq tmchg       if one sec., to time change
020A 80             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.
                    *
```

```

020B            tmchg   equ *
020B 3C 47              inc sec         increase seconds
020D 3C 52              inc base+8      inc. secs. units
020F B6 52              lda base+8
0211 A1 0A              cmp #$0a        look for ten
0213 27 38              beq tens        if yes, inc. tens
0215 A6 3C      minck   lda #$3c        look for a minute
0217 B1 47              cmp sec
0219 26 54              bne ret1        wait for next second
021B 3F 47              clr sec
021D 3F 53              clr base+9      zero display
021F B6 4A              lda base        check min. units
0221 A1 09              cmp #$09        less than 9?
0223 26 20              bne inm1        increase
0225 3F 4A              clr base        min. units = 0
0227 B6 4B              lda base+1      check tens of min.
0229 A1 05              cmp #$05        less than 5?
022B 26 1C              bne inm2        increase
022D 3F 4B              clr base+1      tens of min =0
022F B6 4D              lda base+3      check tens of hrs.
0231 A1 0B              cmp #$0b        look for blank
0233 27 1E              beq hrck        less than 10:00
0235 B6 4C              lda base+2      check hrs. units
0237 A1 02              cmp #$02        less than 2?
0239 26 2A              bne inhr1a      increase
023B A6 0B              lda #$0b
023D B7 4D              sta base+3      set time to 1:00
023F A6 01              lda #$01
0241 B7 4C              sta base+2
0243 20 2A              bra ret1        done
0245 3C 4A      inm1    inc base        increase min. units
0247 20 26              bra ret1        done
0249 3C 4B      inm2    inc base+1      increase tens of min.
024B 20 22              bra ret1

024D 3F 52      tens    clr base+8      zero sec. units
024F 3C 53              inc base+9      inc sec. tens
0251 20 C2              bra minck

                *        increase hours *

0253 B6 4C      hrck    lda base+2      check hours units
0255 A1 09              cmp #$09        less than 9?
0257 26 08              bne inhr1       increase
0259 3F 4C              clr base+2      hours units =0
025B 3F 4D              clr base+3
025D 3C 4D              inc base+3      tens of hours =1
025F 20 0E              bra ret1        done
0261 3C 4C      inhr1   inc base+2      increase hours units
0263 20 0A              bra ret1        done
0265 3C 4C      inhr1a  inc base+2      increase hours units
0267 B6 4C              lda base+2      check value
0269 A1 02              cmp #$02        for 12:00
026B 26 9D              bne ret         no, done
026D 33 49              com pm          switch pm indicator
026F A6 3B      ret1    lda #$3b
0271 B7 42              sta ct          reset inner loop counter
0273 A6 08              lda #$08
0275 B7 43              sta ct1         reset outer loop counter
0277 80                 rti

```

```

                              ***      initialize interrupt vectors ***

0FF8                                  org $ff8

0FF8 01 C8              tmrvec fdb tmrint
0FFA 01 3A              intveq fdb c4int
0FFC 00 90              swiveq fdb start
0FFE 00 90              reset  fdb start
```

Literature Distribution Centers:
USA: 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.
ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MOTOROLA