AT89S8252 Primer

Introduction

The Atmel AT89S8252 microcontroller is a low-power, high-performance device featuring 8K bytes of Flash memory (CMOS PEROM), 2K bytes of EEPROM, and a Serial Peripheral Interface (SPI). The Flash and EEPROM memories may be reprogrammed in-system via the SPI. The EEPROM provides applications with re-writable, nonvolatile data storage. These features, and others, are described in the text which follows. Code samples are provided. Additional information on the AT89S8252 microcontroller may be found in the data sheet and relevant sections in the Atmel AT89-series Microcontroller Databook.

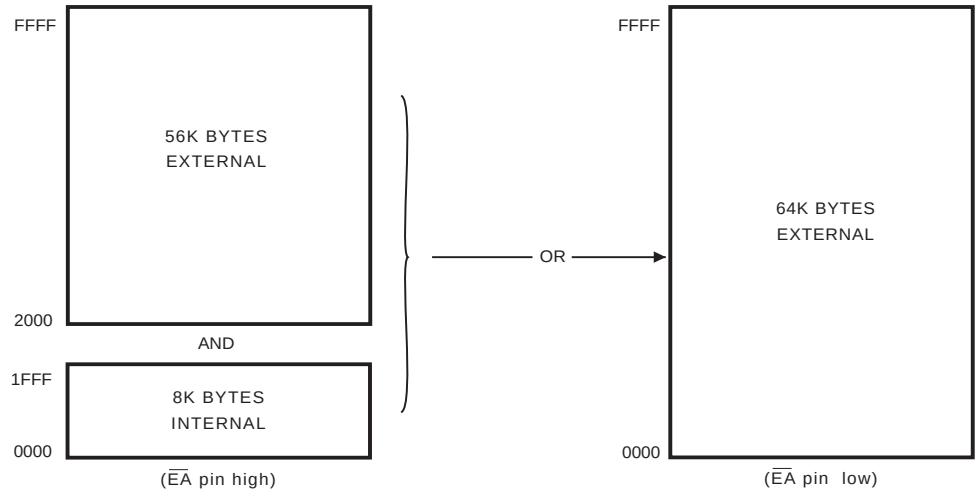
AT89S8252 Memory Organization

Program Memory

The AT89S8252 has separate address spaces for program memory and data memory. Figure 1 shows two alternate maps of program memory.

Program memory is read-only: the microcontroller generates no write signals for program memory. Depending on the state of the $\overline{EA}$ pin, program memory may consist of 8K bytes of internal Flash memory supplemented by up to 56K bytes of external memory, or may consist entirely of up to 64K bytes of external memory. The 8K bytes of internal Flash memory are accessed at addresses 0000H-1FFFH. Program memory accesses at addresses 2000H-FFFFH always access external memory.

Figure 1. The AT89S8252 Program Memory



AT89S8252 Primer

Application Note

Rev. 1018A–03/98

Data Memory

Figure 2 shows a map of AT89S8252 data memory, which consists of 256 bytes of internal RAM, the Special Function Registers (SFRs), 2K bytes of on-chip EEPROM and, optionally, up to 64K bytes of external memory.

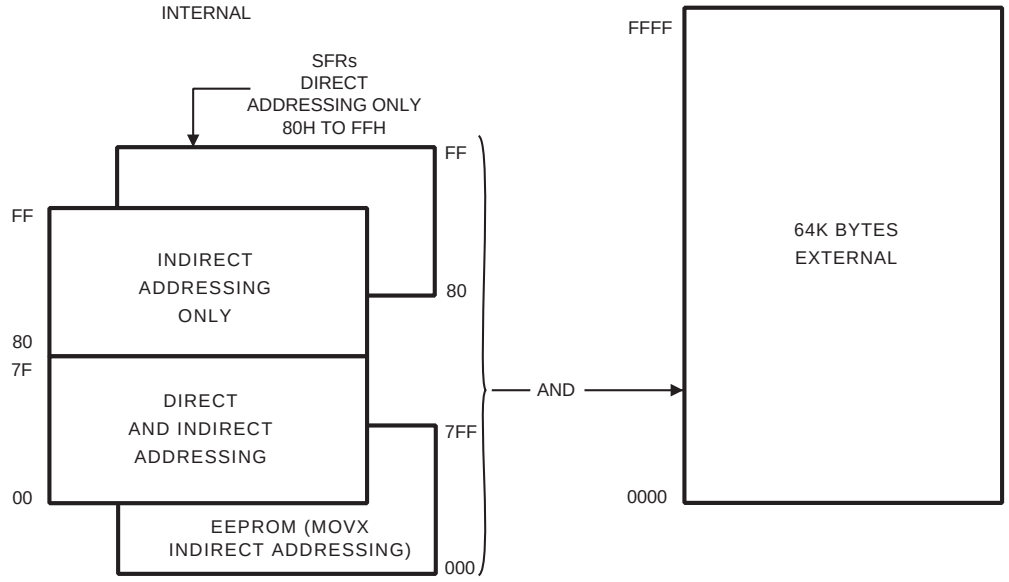
To the left in Figure 2 are shown the 256 bytes of internal RAM and the SFRs, which shadow the upper 128 bytes of internal RAM. The lower 128 bytes (00H-7FH) of internal RAM are accessible by both direct and indirect addressing, while the upper 128 bytes (80H-FFH) are accessible by indirect addressing only. The SFRs (80H-FFH) are accessible by direct addressing only. The addressing mode of an instruction distinguishes accesses to the upper 128 bytes of internal RAM from accesses to the overlapping SFRs.

For an explanation of addressing modes, consult the Architectural Overview section in the Atmel AT89-series Microcontroller Databook.

The stack, which grows upward, may reside anywhere in the 256 bytes of internal RAM.

To the right in Figure 2 are shown the 2K bytes of on-chip EEPROM and the optional 64K bytes of external data memory. Although the EEPROM is internal, it is shown in the diagram shadowing the lower 2K bytes of external data memory because some of the same instructions are used to access EEPROM as are used to access external data memory.

Figure 2. The AT89S8252 Data Memory



EEPROM

EEPROM and external data memory are accessible by indirect addressing only, utilizing the MOVX instructions, which come in two flavors: 8-bit and 16-bit. Only the 16-bit MOVX instructions (those utilizing DPTR) may be used to access internal EEPROM. The 2K bytes of EEPROM are accessed at addresses 000H-7FFH.

Accesses to EEPROM are distinguished from accesses to external data memory by the state of the EEMEN bit in SFR WMCON (96H). To access EEPROM, EEMEN is set; to access external data memory, EEMEN is cleared. Reset clears EEMEN.

To enable write accesses to EEPROM, bit EEMWE in SFR WMCON must also be set. Reset clears this bit, disabling EEPROM writes. It is not necessary to explicitly erase any portion of EEPROM before writing new data.

A write to a location in EEPROM triggers an internal programming cycle, which is guaranteed to last no longer than 10 milliseconds. The completion of an EEPROM programming cycle may be determined by monitoring the RDY/$\overline{\text{BSY}}$ bit in SFR WMCON. RDY/$\overline{\text{BSY}}$ low indicates that programming is in progress; RDY/$\overline{\text{BSY}}$ high indicates that programming is complete. When programming is complete, the contents of the written location may be read back and verified.

The end of an EEPROM programming cycle may also be determined utilizing the $\overline{\text{DATA}}$ Polling method, in which the location written is read repeatedly. During programming, the most significant bit of the data read is the complement of the data bit written. When programming is complete, true data is returned. The return of true data also serves as verification of the write operation.

Sample code showing EEPROM reads and writes is presented in Listing 1.

Listing 1: EEPROM Read/Write Examples.

```
    ; The WMCON register is not bit-addressable, so Boolean operations are used
    ; to control functions and test bits.

    WMCON     DATA      96h             ; watchdog and memory control register
    EEMEN     EQU       00001000b       ; EEPROM access enable bit
    EEMWE     EQU       00010000b       ; EEPROM write enable bit
    WDTRST    EQU       00000010b       ; EEPROM RDY/BSY bit

    ; EEPROM read example.

              orl       WMCON, #EEMEN   ; enable EEPROM accesses
              mov       dptr, #ADDRESS  ; address to read
              movx      a, @dptr        ; read EEPROM
              xrl       WMCON, #EEMEN   ; disable EEPROM accesses

    ; EEPROM write example, utilizing fixed delay for write cycle.
    ; Delay is worst case (10 ms). Code for delay is not shown.
    ; Write is followed by verify (read and compare), but code to handle
    ; verification failure is not shown.

              orl       WMCON, #EEMEN   ; enable EEPROM accesses
              orl       WMCON, #EEMWE   ; enable EEPROM writes

              mov       dptr, #ADDRESS  ; address to write
              mov       a, #DATA        ; data to write
              movx      @dptr, a        ; write EEPROM

              call      DELAY_10_MS     ; wait 10 ms

              movx      a, @dptr        ; read EEPROM
              cjne      a, #DATA, ERROR; jump if data compare fails

              xrl       WMCON, #EEMWE   ; disable EEPROM writes
              xrl       WMCON, #EEMEN   ; disable EEPROM accesses

    ; EEPROM write example, utilizing RDY/BSY to determine the end of
    ; the write cycle. Write is followed by verify (read and compare),
    ; but code to handle verification failure is not shown.
    ; Needs timeout to prevent write error from causing an infinite loop.

              orl       WMCON, #EEMEN   ; enable EEPROM accesses
              orl       WMCON, #EEMWE   ; enable EEPROM writes

              mov       dptr, #ADDRESS  ; address to write
              mov       a, #DATA        ; data to write
              movx      @dptr, a        ; write EEPROM
        loop:
              mov       a, WMCON        ; get EEPROM write status
              anl       a, #WDTRST      ; check RDY/BSY
              jz        loop            ;jump if busy
```

```
        movx    a, @dptr        ; read EEPROM
        cjne    a, #DATA, ERROR; jump if data compare fails

        xrl     WMCON, #EEMWE   ; disable EEPROM writes
        xrl     WMCON, #EEMEN   ; disable EEPROM accesses

; EEPROM write example, utilizing DATA Polling to determine the end of
; the write cycle. After data is loaded, the code loops on read until
; data is returned true. Write verification is implicit in this method.
; Needs timeout to prevent write error from causing an infinite loop.

        orl     WMCON, #EEMEN   ; enable EEPROM accesses
        orl     WMCON, #EEMWE   ; enable EEPROM writes

        mov     dptr, #ADDRESS ; address to write
        mov     a, #DATA        ; data to write
        movx    @dptr, a        ; write EEPROM
    loop:
        movx    a, @dptr        ; read EEPROM
        cjne    a, #DATA, loop ; jump if data compare fails (busy)

        xrl     WMCON, #EEMWE   ; disable EEPROM writes
        xrl     WMCON, #EEMEN   ; disable EEPROM accesses
```

Dual Data Pointers

The AT89S8252 features two 16-bit data pointers (DP0 and DP1) for accessing data in program memory, external data memory, and on-chip EEPROM. The low and high bytes of DP0 are stored in SFRs DP0L (82H) and DP0H (83H), respectively. The low and high bytes of DP1 are stored in SFRs DP1L (84H) and DP1H (85H), respectively. Note that DP0 occupies the same SFRs as the single data pointer in conventional 8051 microcontrollers.

In the AT89S8252, the DPS bit in SFR WMCON (96H) selects the active data pointer (DP0 or DP1). All instructions which reference DPTR utilize the data pointer which is currently selected. To select DP0, DPS is cleared; to select DP1, DPS is set. Reset clears DPS.

The two data pointers may be used to expedite the transfer of data between program memory, external data memory, and on-chip EEPROM, as shown in Listing 2.

Listing 2: Dual Data Pointer Examples.

```
; The WMCON register is not bit-addressable, so Boolean operations are used.
WMCON     DATA      96h             ; watchdog and memory control register
EEMEN     EQU       00001000b       ; EEPROM access enable bit
EEMWE     EQU       00010000b       ; EEPROM write enable bit
WDTRST    EQU       00000010b       ; EEPROM RDY/BSY bit
DPS       EQU       00000100b       ; data pointer select bit

; Copy block from program memory to external data memory.

          mov       r7, #COUNT      ; block byte count
          mov       dptr, #PGM_ADDR; pointer to program memory
          xrl       WMCON, #DPS     ; switch data pointers
          mov       dptr, #XD_ADDR ; pointer to external data memory
    loop:
          xrl       WMCON, #DPS     ; switch data pointers
          clr       a               ; read program memory
          movc      a, @a+dptr      ;
          inc       dptr            ; advance program memory pointer
          xrl       WMCON, #DPS     ; switch data pointers
          movx      @dptr, a        ; write external data memory
          inc       dptr            ; advance external data memory pointer
          djnz      r7, loop        ; continue until done

; Copy block from external data memory to on-chip EEPROM.
; Utilizes RDY/BSY to determine the end of the EEPROM write cycle.
; Needs timeout to prevent write error from causing an infinite loop.

          orl       WMCON, #EEMEN ; enable EEPROM accesses
          orl       WMCON, #EEMWE ; enable EEPROM writes

          mov       r7, #COUNT      ; block byte count
          mov       dptr, #EE_ADDR ; pointer to EEPROM
          xrl       WMCON, #DPS     ; switch data pointers
          mov       dptr, #XD_ADDR ; pointer to external data memory
    copy:
          movx      a, @dptr        ; read external data memory
          inc       dptr            ; advance external data memory pointer
          xrl       WMCON, #DPS     ; switch data pointers
          movx      @dptr, a        ; write EEPROM
          inc       dptr            ; advance EEPROM pointer
          xrl       WMCON, #DPS     ; switch data pointers
    wait:
          mov       a, WMCON        ; get EEPROM write status
          anl       a, #WDTRST      ; check RDY/BSY
          jz        wait            ; jump if busy
          djnz      r7, copy        ; continue until done

          xrl       WMCON, #EEMWE; disable EEPROM writes
          xrl       WMCON, #EEMEN; disable EEPROM accesses
```

IMPORTANT: The state of DPS affects ALL accesses to the data pointer SFRs (82H, 83H, 84H, 85H). Any machine instruction whose operand is one of the data pointer SFRs may produce unexpected results, as shown below.

Examples:

```
; Define the new data pointer SFRs for a generic 8051 assembler.

DP0L     DATA     82h            ; data pointer 0
DP0H     DATA     83h            ;
DP1L     DATA     84h            ; data pointer 1
DP1H     DATA     85h            ;

         orl      WMCON, #DPS    ; set DPS

         push     DP0L           ; PUSHES DP1L!!!
         mov      83h, a         ; COPIES ACCUMULATOR TO 85H!!!

         xrl      WMCON, #DPS    ; clear DPS

         push     DP1H           ; PUSHES DP0H!!!
         mov      84h, a         ; COPIES ACCUMULATOR TO 82H!!!
```

The user must exercise caution to avoid accessing the wrong SFRs. The solution to the problem demonstrated above is to clear DPS before any accesses to SFRs 82H and 83H and to set DPS before any accesses to SFRs 84H and 85H.

Watchdog Timer

The AT89S8252 features a watchdog timer which allows control of the microcontroller to be regained, should it be lost. When enabled, the timer will reset the microcontroller after a specified period has elapsed, unless prevented from doing so by the intervention of the firmware.

To enable the watchdog timer, the WDTEN bit in SFR WMCON (96H) must be set; to disable the timer, WDTEN should be cleared. Once the timer is enabled, the firmware must set the WDRST bit in SFR WMCON (or disable the timer) before the reset period elapses to prevent the timer from resetting the microcontroller. Each time WDRST is set, a new reset period begins, requiring another response from the firmware. The firmware does not need to clear WDRST after setting it; WDRST is automatically cleared by the microcontroller.

The watchdog timer reset period varies from 16 to 2048 milliseconds, as specified by bits PS0, PS1 and PS2 in SFR WMCON. Refer to the AT89S8252 data sheet for the nominal reset periods corresponding to the bit settings. The timer reset period is independent of the frequency of the clock source driving the microcontroller and may deviate from the documented nominal value by a huge percentage.

The watchdog timer continues to operate even when the microcontroller is in Idle mode, but is disabled during Power Down mode. The elapsed time between a watchdog timer reset and the execution of the first instruction is approximately 16 ms. Reset (including reset generated by the watchdog timer) clears WDTEN, WDRST, SP0, SP1 and SP2, disabling the watchdog timer.

A typical application of the watchdog timer is outlined in Listing 3.

Listing 3: Watchdog Timer Example.

```
; Use the watchdog timer to regain control of the microcontroller if an
; operation takes longer than expected. The details of the operation are not
; shown. The operation is expected to take less than 20 ms to complete and
; the reset period chosen is 32 ms. Adequate margin must be allowed between
; the desired reset period and the selected period to allow for the slop
; present in the timer.
; The WMCON register is not bit-addressable, so Boolean operations are used.

WMCON     DATA      96h            ; watchdog and memory control register
WDTEN     EQU       00000001b      ; watchdog timer enable bit
WDTRST    EQU       00000010b      ; watchdog timer reset bit
PS0       EQU       00100000b      ; watchdog timer period select bits
PS1       EQU       01000000b      ;
PS2       EQU       10000000b      ;

          orl       WMCON, #PS0    ; select 32-ms period
          orl       WMCON, #WDTEN  ; enable watchdog
      loop:
          ; Do something which normally takes less than 20 ms.
          .
          .
          .

          orl       WMCON, #WDTRST ; keep watchdog at bay
          jmp       loop
```

Power Off Flag

The Power Off Flag (POF) indicates that power has been removed from the AT89S8252. This allows the firmware to differentiate between reset due to the application of power and reset due to the watchdog timer, or a logic high on the RST pin. POF is set when power is applied to the microcontroller and is not affected by the watchdog timer or by activity on RST. POF is located at bit four in SFR PCON (87H), and may be read, set, or cleared by firmware. Note that PCON is not bit-addressable.

A typical application of the Power Off Flag is outlined in Listing 4.

Listing 4. Power Off Flag Example.

```
; After reset, the microcontroller begins executing code at program memory
; address 0000H. POF is tested to determine if the controller was reset
; by the application of power (cold start) or by the watchdog timer or a
; high on RST (warm start).
; Code for the cold start and warm start routines is not shown.

POF       EQU       00010000b       ; Power Off Flag bit

          CSEG                      ; code segment

          ORG       0000h           ; location of reset vector
          jmp       xreset          ; vector
          .
          .
          .
     xreset:                        ; code for responding to reset
          .
          .
          .
          mov       a, PCON         ; get Power Control register
          anl       a, #POF         ; test Power Off Flag
          jz        WARM_START      ; POF=0 indicates reset from
                                    ;   watchdog timer or RST
          xrl       PCON, #POF      ; clear POF for next time
          jmp       COLD_START      ; POF=1 indicates reset from power
```

Serial Peripheral Interface

The Serial Peripheral Interface (SPI) permits compatible devices to communicate serially over a high-speed, synchronous bus. Devices resident on the bus act as masters or slaves, with only one master and one slave active at any one time. Data transfers are always initiated by a master, and are actually data exchanges, with data flowing from the master to the slave and from the slave to the master simultaneously.

SPI-compatible devices have four pins in common: SCK, MOSI, MISO, and $\overline{SS}$. All devices in a system have their SCK, MOSI, and MISO pins tied together. Data flows from master to slave via MOSI (Master Out Slave In) and from slave to master via MISO (Master In Slave Out). Data transfers are synchronized to a clock generated by the master and output on its SCK pin. SCK is an input for devices configured as slaves. Inactive masters must be reconfigured as slaves to prevent them from driving their SCK and MOSI pins.

The $\overline{SS}$ (Slave Select) pins on the devices in the system are not bussed. Each slave is connected to its master by a select line from its $\overline{SS}$ input to a general purpose output on the master. If a slave has multiple masters, the multiple select lines must be gated to its $\overline{SS}$ input. Masters do not utilize their $\overline{SS}$ pins during SPI data transfers, freeing them for use as general-purpose outputs.

To initiate an SPI data transfer, the active master selects a slave by applying a logic low to the slave's $\overline{SS}$ input. The master starts the serial clock, which it outputs on its SCK pin, and shifts out a byte on its MOSI pin, synchronized to the clock. Simultaneously, the slave shifts out a byte on its MISO pin, synchronized to the clock. When the master and slave have exchanged data, the transfer is complete. The master stops the serial clock and may deselect the slave. Slaves which are not selected ignore their SCK inputs and float their MISO outputs to avoid contention with the active output of the selected slave.

In the AT89S8252, the SPI is configured via SFR SPCR (D5H), the SPI Control Register. The frequency of the serial clock, the ordering of the serial data, and the relationship between the clock and the shifting and sampling of data are all programmable, as described below.

To enable the SPI feature, the SPE bit in SFR SPCR must be set; to disable the SPI, SPE is cleared. When the SPI is enabled, microcontroller pins P1.4, P1.5, P1.6 and P1.7 become $\overline{SS}$, MOSI, MISO, and SCK, respectively. The SPI may not operate correctly unless pins P1.4-P1.7 are first

programmed high. Reset sets pins P1.4-P1.7 high and clears SPE, disabling the SPI.

The MSTR bit in SFR SPCR configures the microcontroller as a SPI master when set, and as a slave when cleared. Reset clears MSTR. When the microcontroller is configured as a SPI master, $\overline{SS}$ (P1.4) is not utilized and may be used as a general-purpose, programmable output.

When the microcontroller is configured as a SPI master, the frequency of the serial clock is determined by bits SPR0 and SPR1 in SFR SPCR. The frequency of the serial clock is the frequency of the microcontroller's clock source divided by the selected divisor. The divisor must be selected to produce a serial clock frequency which is compatible with the master's slaves. Refer to the AT89S8252 data sheet for the divisors corresponding to the settings of bits SPR0 and SPR1.

The DORD bit in SFR SPCR determines the order in which the bits in the serial data are transferred. Data is transferred least-significant bit (LSB) first when DORD is set; most-significant bit (MSB) first when DORD is cleared. Reset clears DORD. Note that only MSB-first data transfers are shown in the diagrams in the AT89S8252 data sheet.

The polarity of the SPI serial clock is determined by the CPOL bit in SFR SPCR. Setting CPOL specifies serial clock high when idle; clearing CPOL specifies serial clock low when idle. Reset clears CPOL.

The CPHA bit in SFR SPCR controls the phase of the SPI serial clock, which defines the relationship between the clock and the shifting and sampling of serial data. Setting CPHA specifies that data is to be shifted on the leading edge of the clock and sampled on the trailing edge. Clearing CPHA specifies that data is to be sampled on the leading edge of the clock and shifted on the trailing edge. Reset sets CPHA. The state of bit CPHA also affects the slave selects. If CPHA is set, the slave may remain selected between consecutive byte transfers, or may be permanently selected ($\overline{SS}$ tied low). If CPHA is clear, the slave must be deselected ($\overline{SS}$ returned high) after each byte transferred. Examples of SPI serial clock phase and polarity are shown in the diagrams in the AT89S8252 data sheet.

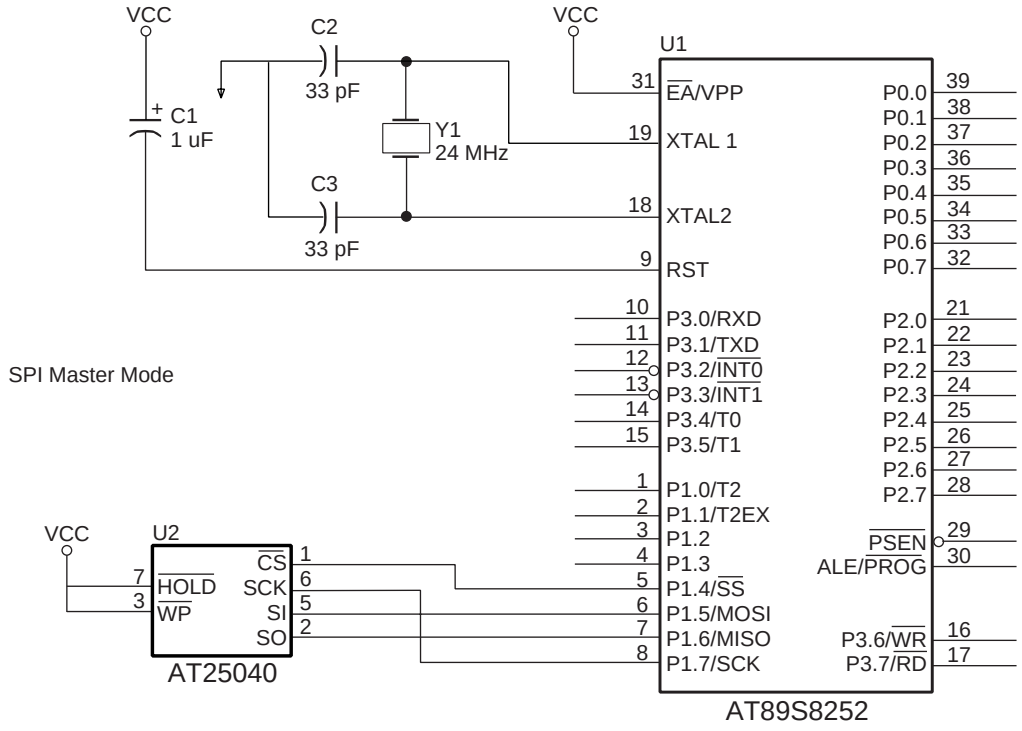
Only an AT89S8252 configured as an SPI master may initiate a data transfer. A data transfer is triggered by a byte written to SFR SPDR (86H), the SPI Data Register. As data is shifted out of the master, data from the selected slave is simultaneously shifted in, replacing the data in SPDR. When a data transfer is complete, the SPIF bit is set in SFR SPSR (AAH), the SPI Status Register. The data received from the slave may then be read from SPDR. Writing SPDR during a data transfer sets the Write Collision bit (WCOL) in SPSR. The progress of the data transfer is not affected by a collision. To clear bits SPIF and WCOL, read SPSR and read or write SPDR.

An interrupt may be generated as an alternative to polling SPIF to determine the end of a SPI data transfer. To enable the SPI interrupt, three bits must be set. The first is the SPIE bit in SPCR, which causes an interrupt to be generated when SPIF is set. The second and third bits are ES and EA in SFR IE (A8H). ES is the UART interrupt enable bit, which must be set because the SPI shares an interrupt vector with the UART. EA is the global interrupt enable bit. When an SPI interrupt occurs, the SPI/UART interrupt service routine must determine the source of the interrupt. An SPI interrupt is indicated when the SPIF bit in SPSR is set. Bits SPIF and ES must be cleared by software.

In the application shown below, the AT89S8252 is configured as an SPI master and interfaces to an Atmel AT25040 SPI-compatible EEPROM. The EEPROM provides 512 bytes of re-writable, non-volatile storage while requiring only a four-pin interface to the microcontroller. The microcontroller and EEPROM are wired as shown in Figure 3. Note that the microcontroller's $\overline{SS}$ pin is used as a slave select, since it is unused when the microcontroller is configured as a SPI master. Additional EEPROMs may be connected to the microcontroller's SCK, MISO and MOSI pins, but each device must have its own select line.

Sample code for the application is shown in Listing 5. A SPI master must be configured to meet the requirements of its slaves. The AT25040 data sheet states that the maximum clock rate for the device is 2 MHz. The microcontroller's clock source is a 24-MHz crystal (Figure 3), so a SPI serial clock divisor of 16 was chosen to produce a serial clock of 1.5 MHz. As shown in the AT25040 data sheet, the device's chip select ($\overline{CS}$) input must remain active (low) for the duration of an operation, which may include multiple data transfers. Also, the serial clock must be low when idle and data is transferred most-significant bit first. Therefore, CPHA=1, CPOL=0 and DORD=0. In the example, SPI interrupts are not used.

Figure 3. AT89S8252 as an SPI Master



Listing 5: SPI Example.

```
; Write/Read AT25C040 EEPROM via the Serial Peripheral Interface (SPI).
; Completion of AT25C040 programming is determined by polling the device.
; SPI interrupt is not used. ; Works with a microcontroller clock of 24 MHz (or slower).
;
; The AT25040 routines (“read_status”, “enable_write”, “read_byte”,
; “write_byte”) are excerpted from code previously made available by Atmel
; for use with the AT89Cx051 microcontrollers. In that code, access to the
; AT25040 was via “bit banging”. The two routines which shifted the serial
; data in/out have been replaced by the single SPI routine “masterIO”.

; Microcontroller registers and bit definitions.

SPCR      DATA      0d5h          ; SPI control register
SPSR      DATA      0aah          ; SPI status register
SPIF      EQU       10000000b     ; interrupt flag
SPDR      DATA      86h           ; SPI data register

; Microcontroller connections to AT25040.

CS_       BIT       p1.4          ; AT25040 slave select
MOSI      BIT       p1.5          ; SPI
MISO      BIT       p1.6          ; SPI
SCK       BIT       p1.7          ; SPI

; AT25040 device command and bit definitions.
```

```
RDSR     EQU      05h            ; Read Status Register
WRSR     EQU      01h            ; Write Status Register
READ     EQU      03h            ; Read Data from Memory
WRITE    EQU      02h            ; Write Data to Memory
WREN     EQU      06h            ; Write Enable
WRDI     EQU      04h            ; Write Disable

A8       BIT      acc.3          ; MSB of address
NRDY     BIT      acc.0          ; high = write cycle in progress

main:
     ; SPI master mode initialization code.

         setb     CS_            ; deselect AT25040

         setb     MOSI           ; initialize SPI pins
         setb     MISO           ;
         setb     SCK            ;

         mov      SPCR, #01010101b      ; initialize SPI master
                                ; interrupt disable, pin enable,
                                ; MSB first, polarity 0, phase 1,
                                ; clock rate /16
; Write one byte to AT25040 and verify (read and compare).
; Code to handle verification failure is not shown.
; Needs timeout to prevent write error from causing an infinite loop.

         call     enable_write     ; must precede each byte write
         mov      a, #DATA         ; data
         mov      dptr, #ADDRESS   ; address
         call     write_byte       ; write
   wchk:
         call     read_status      ; check write status
         jb       NRDY, wchk       ; loop until done

         mov      dptr, #ADDRESS   ; address
         call     read_byte        ; read
         cjne     a, #DATA, ERROR  ; jump if data compare fails
         .
         .
         .
read_status:

   ; Read device status.
   ; Returns status byte in A.

         clr      CS_              ; select device
         mov      a, #RDSR         ; get command
         call     masterIO         ; send command
         call     masterIO         ; get status
```

```
        setb    CS_             ; deselect device
        ret

enable_write:

    ; Enable write.
    ; Does not check for device ready before sending command.
    ; Returns nothing. Destroys A.

        clr     CS_             ; select device
        mov     a, #WREN        ; get command
        call    masterIO        ; send command
        setb    CS_             ; deselect device
        ret

read_byte:

    ; Read one byte of data from specified address.
    ; Does not check for device ready before sending command.
    ; Called with address in DPTR.
    ; Returns data in A.

        clr     CS_             ; select device
        mov     a, dph          ; get high byte of address
        rrc     a               ; move LSB into carry bit
        mov     a, #READ        ; get command
        mov     A8, c           ; combine command and high bit of addr
        call    masterIO        ; send command and high bit of address
        mov     a, dpl          ; get low byte of address
        call    masterIO        ; send low byte of address
        call    masterIO        ; get data
        setb    CS_             ; deselect device
        ret

write_byte:

    ; Write one byte of data to specified address.
    ; Does not check for device ready or write enabled before sending
    ; command. Does not wait for write cycle to complete before returning.
    ; Called with address in DPTR, data in A.
    ; Returns nothing.

        clr     CS_             ; select device
        push    acc             ; save data
        mov     a, dph          ; get high byte of address
        rrc     a               ; move LSB into carry bit
        mov     a, #WRITE       ; get command
        mov     A8, c           ; combine command and high bit of address
        call    masterIO        ; send command and high bit of address
        mov     a, dpl          ; get low byte of address
```

```
        call    masterIO        ; send low byte of address
        pop     acc             ; restore data
        call    masterIO        ; send data
        setb    CS_             ; deselect device
        ret

masterIO:
    ; Send/receive data through the SPI port.
    ; A byte is shifted in as a byte is shifted out,
    ; receiving and sending simultaneously.
    ; Waits for shift out/in complete before returning.
    ; Expects slave already selected.
    ; Called with data to send in A. Returns data received in A.

        mov     SPDR, a         ; write output data
    bbb:
        mov     a, SPSR         ; get status
        anl     a, #SPIF        ; check for done
        jz      bbb             ; loop until done

        move    a, SPDR         ; read input data
        ret
```