AT89S8253 Primer

1. Introduction

The Atmel® AT89S8253 microcontroller is a low-power, high-performance device featuring 12K bytes of Flash memory, 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 this application note. Code samples are provided. Additional information on the AT89S8253 microcontroller can be found in the datasheet. The AT89S8253 device is meant as a replacement for the AT89S8252 and AT89S53 devices.

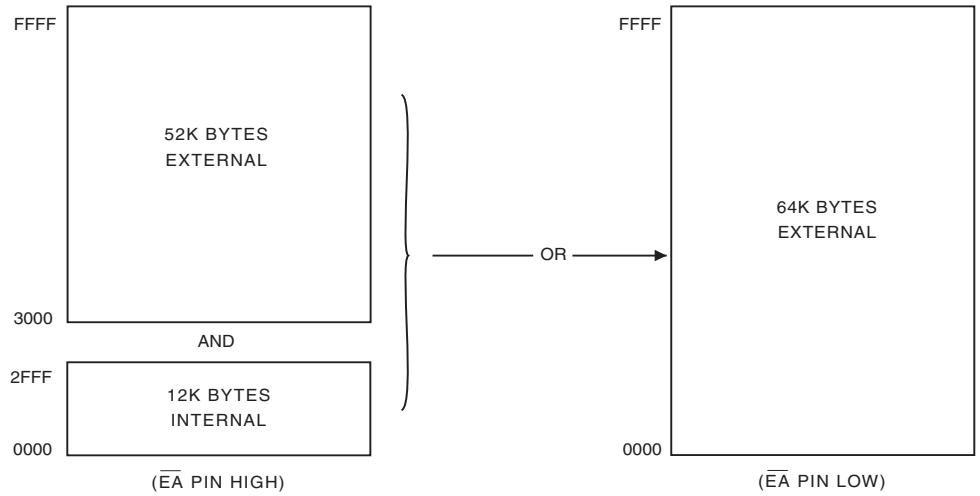
2. Memory Organization

2.1 Program Memory

The AT89S8253 has separate address spaces for program memory and data memory. Figure 2-1 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 12K bytes of internal Flash memory supplemented by up to 52K bytes of external memory, or may consist entirely of up to 64K bytes of external memory. The 12K bytes of internal Flash memory are accessed at addresses 0000H - 2FFFH. Program memory accesses at addresses 3000H - FFFFH always access external memory.

Figure 2-1. AT89S8253 Program Memory



Flash Microcontrollers

AT89S8253 Primer

Application Note

3655B–MICRO–3/07

2.2 Data Memory

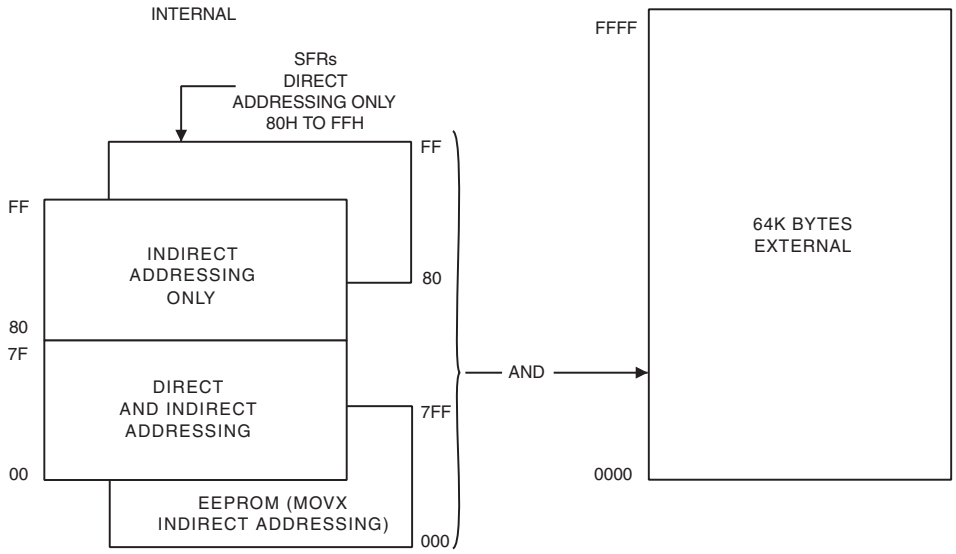
Figure 2-2 shows a map of AT89S8253 data memory, which consists of 256 bytes of internal RAM, the Special Function Registers (SFRs), 2K bytes of on-chip EEPROM and, optionally, up to 64K bytes of external memory.

To the left in Figure 2-2 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.

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

To the right in Figure 2-2 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 2-2. AT89S8253 Data Memory



3. EEPROM

The AT89S8253 includes 2K bytes of on-chip EEPROM for nonvolatile data storage. EEPROM and external data memory are accessible by indirect addressing only, utilizing the MOVX instructions, which come in two modes: 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 EECON (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 EECON 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.

Sample code showing EEPROM reads and writes is presented in Section 3.3.

Table 3-1. EECON–EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
96h	–	–	EELD	EEMWE	EEMEN	DPS	RDY/$\overline{BSY}$	WRTINH	EECON
Reset Value	X	X	0	0	0	0	X	X	

3.1 Page vs. Byte Programming

The EEPROM on the AT89S8253 allows for page-based programming. One or more bytes in a page, up to 32 bytes total, may be written at one time. This feature reduces the amount of time required to write multiple bytes to the EEPROM. The EELD bit in EECON allows multiple data bytes to be loaded to a temporary page buffer. While EELD = 1, MOVX @DPTR,A instructions will load data to the page buffer, but will not start a write sequence. Note that a previously loaded byte must not be reloaded prior to the write sequence. To write the page into the memory, EELD must first be cleared and then a MOVX @DPTR,A with the final data byte is issued. The address of the final MOVX determines which page will be written. Figures 3-1 and 3-2 show the difference between byte writes and page writes.

Figure 3-1. EEPROM Byte Write

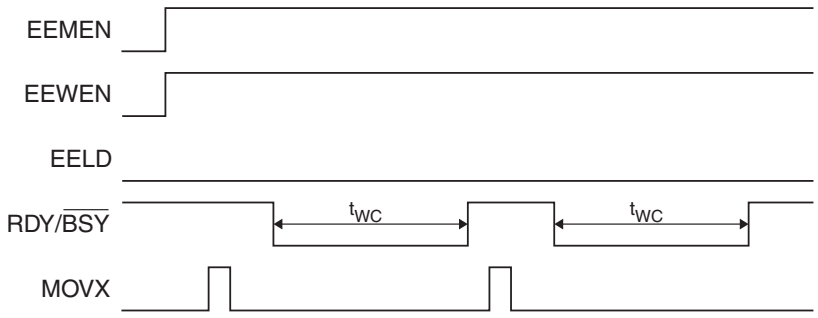
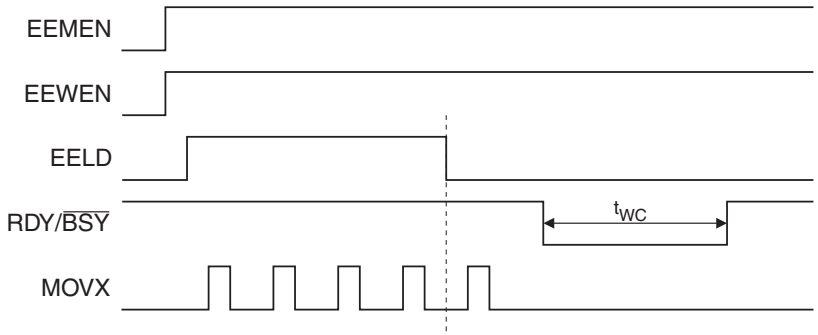


Figure 3-2. EEPROM Page Write



3.2 EEPROM Status

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/$\overline{BSY}$ bit in SFR EECON. The RDY/$\overline{BSY}$ low indicates that programming is in progress; RDY/$\overline{BSY}$ high indicates that programming is complete. It may take several microseconds for RDY/$\overline{BSY}$ to go low after a programming cycle is initiated. Polling of the RDY/$\overline{BSY}$ bit should always check for both the low and high condition. When programming is complete, the contents of the written location may be read back and verified.

3.2.1 $\overline{\text{DATA}}$ Polling

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.

3.2.2 Low-Voltage Write Inhibit

An EEPROM write sequence will not occur if the supply voltage is not sufficiently high enough for programming to be successful. The AT89S8253 includes a Low-Voltage Detector that inhibits writing of the EEPROM when the supply voltage drops below a predefined threshold. The user should check the write inhibit status by reading the $\overline{\text{WRTINH}}$ bit in EECON. $\overline{\text{WRTINH}}$ is driven low by hardware when V_{CC} drops below the write voltage threshold. $\overline{\text{WRTINH}}$ will be set high approximately 2 ms after V_{CC} is restored to a safe level. The EEPROM can always be read regardless of the state of $\overline{\text{WRTINH}}$.

3.3 Listing 1: EEPROM Read/Write Examples

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

EECON   DATA    96H             ; watchdog and memory control register
EEMEN   EQU     00001000B       ; EEPROM access enable bit
EEMWE   EQU     00010000B       ; EEPROM write enable bit
EELD    EQU     00100000B       ; EEPROM page load enable bit
WRTINH  EQU     00000001B       ; EEPROM WRTINH bit
RDY     EQU     00000010B       ; EEPROM RDY/BSY bit

; EEPROM read example.

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

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

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

        mov     a, EECON      ; get EEPROM status
        anl     a, #WRTINH    ; check WRTINH
        jz      ERROR         ; jump if inhibited

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

; EEPROM byte write example, utilizing RDY/$\overline{BSY}$ to determine the end of
; the write cycle. Write is preceded by a write inhibit check, but code
; to handle an inhibit condition is not shown. 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     EECON, #EEMEN ; enable EEPROM accesses
        orl     EECON, #EEMWE ; enable EEPROM writes

        mov     a, EECON      ; get EEPROM status
        anl     a, #WRTINH    ; check WRTINH
        jz      ERROR         ; jump if inhibited

        mov     dptr, #ADDRESS ; address to write
        mov     a, #DATA      ; data to write
        movx    @dptr, a      ; write EEPROM
   loop1:
        mov     a, EECON      ; get EEPROM write status
        anl     a, #RDY       ; check RDY/BSY
        jnz     loop1         ; jump if not busy yet
   loop2:
        mov     a, EECON      ; get EEPROM write status
        anl     a, #RDY       ; check RDY/BSY
        jz      loop2         ; jump if busy
        movx    a, @dptr      ; read EEPROM
        cjne    a, #DATA, ERROR; jump if data compare fails

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

; EEPROM byte write example, utilizing $\overline{DATA}$ Polling to determine the end of
; the write cycle. Write is preceded by a write inhibit check, but code to
; handle an inhibit condition is not shown. 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     EECON, #EEMEN ; enable EEPROM accesses
        orl     EECON, #EEMWE ; enable EEPROM writes

        mov     a, EECON      ; get EEPROM status
        anl     a, #WRTINH    ; check WRTINH
        jz      ERROR         ; jump if inhibited
```

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

; EEPROM page write example, utilizing RDY/BSY to determine the end of
; the write cycle. Write is preceded by a write inhibit check, but code
; to handle an inhibit condition is not shown. Data is copied from the
; internal RAM into a single page of the EEPROM.
; Needs timeout to prevent write error from causing an infinite loop.

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

        mov     a, EECON      ; get EEPROM status
        anl     a, #WRTINH    ; check WRTINH
        jz      ERROR         ; jump if inhibited

        mov     dptr, #ADDRESS ; starting address to write
        mov     r0, #DATA      ; starting address of data to write
        mov     r1, #(PAGESIZE-1); amount of data to write

        orl    EECON, #EELD   ; set EELD
    loop0:
        mov     a, @r0         ; get data
        inc     r0             ; next source address
        movx    @dptr, a       ; load EEPROM
        inc     dptr           ; next destination address
        djnz    r1, loop0      ; jump if not done yet

        xrl     EECON, #EELD ; clear EELD
        mov     a, @r0         ; get last data
        movx    @dptr, a       ; write EEPROM
    loop1:
        mov     a, EECON       ; get EEPROM write status
        anl     a, #RDY        ; check RDY/BSY
        jnz     loop1          ; jump if not busy yet
    loop2:
        mov     a, EECON       ; get EEPROM write status
        anl     a, #RDY        ; check RDY/BSY
        jz      loop2          ; jump if busy
        movx    a, @dptr       ; read EEPROM
        cjne    a, #DATA, ERROR; jump if data compare fails

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

4. Dual Data Pointers

The AT89S8253 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 AT89S8253, the DPS bit in SFR EECON (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. Note that DP0 is **always** accessed at SFRs DP0L (82H) and DP0H (83H) and DP1 is **always** accessed at SFRs DP1L (84H) and DP1H (85H), **regardless of the state of DPS**. This behavior is the same as AT89S52, but is different than AT89S8252.

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 Section 4.1.

4.1 Listing 2: Dual Data Pointer Examples

```
; The EECON register is not bit-addressable, so Boolean operations are used.
EECON   DATA    96H          ; Watchdog and memory control register
EEMEN   EQU     00001000B    ; EEPROM access enable bit
EEMWE   EQU     00010000B    ; EEPROM write enable bit
RDY     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     EECON, #DPS  ; switch data pointers
        mov     dptr, #XD_ADDR ; pointer to external data memory
   loop:
        xrl     EECON, #DPS  ; switch data pointers
        clr     a            ; read program memory
        movc    a, @a+dptr   ;
        inc     dptr         ; advance program memory pointer
        xrl     EECON, #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.
; The code could be made more efficient through use of page programming.
; Needs timeout to prevent write error from causing an infinite loop.

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

        mov     r7, #COUNT   ; block byte count
        mov     dptr, #EE_ADDR ; pointer to EEPROM
        xrl     EECON, #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     EECON, #DPS   ; switch data pointers
     movx    @dptr, a      ; write EEPROM
     inc     dptr          ; advance EEPROM pointer
     xrl     EECON, #DPS   ; switch data pointers
wait1:
     mov     a, EECON      ; get EEPROM write status
     anl     a, #RDY       ; check RDY/BSY
     jnz     wait1         ; jump if not busy
wait2:
     mov     a, EECON      ; get EEPROM write status
     anl     a, #RDY       ; check RDY/BSY
     jz      wait2         ; jump if busy
     djnz    r7, copy      ; continue until done

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

5. Four-level Interrupt Controller

The AT89S8253 includes an enhanced interrupt controller with support for four priority levels. The additional priority levels allow greater control over the interrupt response sequence in multiple interrupt systems. Four priority levels require two priority bits per interrupt. The lower order bits are stored in the existing IP (B8H) Sfr. The higher order bits are stored in the additional IPH (B7H) SFR. The priority bits for individual interrupts share the same position in both IP and IPH. The polling order for interrupts of the same priority remains the same as previous devices. The following example configures the interrupts with different priority levels.

5.1 Listing 3: Interrupt Priority Example

```
; The IPH register is not bit-addressable, so Boolean operations are used.
IP       DATA     B8H           ; low priority register
IPH      DATA     B7H           ; high priority register
PX0H     EQU      00000001B     ; external interrupt 0
PT0H     EQU      00000010B     ; timer 0 overflow
PX1H     EQU      00000100B     ; external interrupt 1
PT1H     EQU      00001000B     ; timer 1 overflow
PSH      EQU      00010000B     ; serial interrupts
PT2H     EQU      00100000B     ; timer 2 interrupt

; Set up priority between interrupts
;   priority 3: Timer 2 (Highest)
;   priority 2: INT1, Timer 0
;   priority 1: Serial
;   priority 0: INT0, Timer 1(Lowest)

     setb    ps            ; serial = level 1
     orl     IPH, #PX1H    ; INT1 = level 2
     orl     IPH, #PT0H    ; timer 0 = level 2
     setb    pt2           ; timer 2 = level 1
     orl     IPH, #PT2H    ; timer 2 = level 3
```

6. Watchdog Timer

The AT89S8253 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. The AT89S8253 watchdog has two operating modes: software mode (from AT89S8252 and AT89S53) and hardware mode (from AT89S52). Note that the watchdog on AT89S8253 is controlled by SFR WDTCON at address A7H as compared with WMCON (96H) on AT89S8252 and AT89S53.

To enable the watchdog timer in software mode, the WDTEN bit in SFR WDTCON (A7H) must be set and the HWDT bit must be cleared; to disable the timer, WDTEN should be cleared. Once the timer is enabled, the firmware must set the WSWRST bit in SFR WDTCON (or disable the timer) before the reset period elapses to prevent the timer from resetting the microcontroller. Each time WSWRST is set, a new reset period begins, requiring another response from the firmware. The firmware does not need to clear WSWRST after setting it; WSWRST is automatically cleared by the microcontroller.

To enable the watchdog timer in hardware mode, the HWDT bit in WDTCON must be set and the sequence 1EH/E1H must be written to SFR WDTRST (A6H). In hardware mode, the watchdog can only be disabled by a device reset. The WDTEN bit will be forced high when the hardware watchdog mode is enabled. Once the hardware watchdog timer is enabled, the firmware must write the 1Eh/E1h sequence to WDTRST before the reset period elapses to prevent the timer from resetting the microcontroller. Each time WDTRST is written with the correct sequence, a new reset period begins, requiring another response from the firmware.

The watchdog timer reset period varies from 16K to 2048K system clock cycles, as specified by bits PS0, PS1 and PS2 in WDTCON. Refer to the AT89S8253 datasheet for the nominal reset periods corresponding to the bit settings. The timer reset period of the AT89S8253 depends on the frequency of the clock source driving the microcontroller. If the watchdog timer times out, it will generate a reset pulse lasting 96 clock periods. The RST pin will also be pulled high for the duration of the reset, unless the DISRTO bit in WDTCON was set prior to the time-out. Reset (including reset generated by the watchdog timer) clears WDTEN, WSWRST, HWDT, WDIDLE, DISRTO, SP0, SP1 and SP2, disabling the watchdog timer.

When WDIDLE = 0, the watchdog timer continues to operate even when the microcontroller is in Idle mode. To prevent the watchdog from timing out during Idle, an interrupt such as a timer overflow must be used to wake up the device periodically and reset the watchdog, or the WDIDLE bit may be set to disable the watchdog during Idle. The watchdog is always disabled during Power-down mode. To prevent the watchdog from timing out immediately upon exit from Power-down (or Idle with WDIDLE set), the watchdog should be reset just before entering Power-down (or Idle).

Table 6-1. WDTCON– Watchdog Control Register

Bit	7	6	5	4	3	2	1	0
A7H	PS2	PS1	PS0	WDIDLE	DISRTO	HWDT	WSWRST	WDTEN
Reset Value	0	0	0	0	0	0	0	0

A typical application of the watchdog timer is outlined in Section 6.1.

6.1 Listing 4: Watchdog Timer Example

```
; Use the software 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 EECON register is not bit-addressable, so Boolean operations are used.

WDTCON   DATA    A7H          ; watchdog control register
WDTRST   DATA    A6H          ; watchdog reset register
WDTEN    EQU     00000001B    ; watchdog timer enable bit
WSWRST   EQU     00000010B    ; watchdog timer software reset bit
HWDT     EQU     00000100B    ; watchdog timer hardware mode bit
PS0      EQU     00100000B    ; watchdog timer period select bits
PS1      EQU     01000000B    ;
PS2      EQU     10000000B    ;

         orl     WDTCON, #PS0  ; select 32-ms period at 12MHz
         orl     WDTCON, #WDTEN; enable watchdog
    loop:
         ; Do something which normally takes less than 20 ms.
         .
         .
         .

         orl     WDTCON, #WSWRST ; keep watchdog at bay
         jmp     loop

; Use the hardware 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 EECON register is not bit-addressable, so Boolean operations are used.

         orl     WDTCON, #PS0  ; select 32-ms period at 12MHz
         orl     WDTCON, #HWDT ; enable hardware watchdog
         mov     WDTRST, #01EH ; enable watchdog sequence 1
         mov     WDTRST, #0E1H ; enable watchdog sequence 2

    loop:
         ; Do something which normally takes less than 20 ms.
         .
         .
         .

         ; keep watchdog at bay
         mov     WDTRST, #01EH ; feed watchdog sequence 1
         mov     WDTRST, #0E1H ; feed watchdog sequence 2
         jmp     loop
```

7. Power Off Flag

The Power Off Flag (POF) indicates that power has been removed from the AT89S8253. 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 Section 7.1.

7.1 Listing 5. 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
```

8. Enhanced UART

The serial port (UART) of the AT89S8253 includes the enhanced features of framing error detection and automatic address recognition.

8.1 Framing Error Detection

Invalid stop bits can flag errors in the serial communication stream of the UART. The UART looks for missing stop bits in the communication. A missing stop bit will set the FE bit in the SCON register. The FE bit shares the SCON.7 bit position with SM0 and the function of SCON.7 is determined by SMOD0 (PCON.6). SCON.7 functions as SM0 when SMOD0 is cleared. If SMOD0 is set then SCON.7 functions as FE. FE will be set by missing stop bits regardless of the state of SMOD0. FE can only be cleared by software, i.e. a framing error will be remembered, even if subsequent communication is valid, until the FE bit is cleared by the software.

To use framing error detection with Modes 2 or 3, first set SM0 while SMOD0 = 0 and then set SMOD0 to map FE into SCON.

8.2 Automatic Address Recognition

Automatic Address Recognition is a feature which allows the UART to recognize certain addresses in the serial bit stream by using hardware to make the comparisons. This feature saves a great deal of software overhead by eliminating the need for the software to examine every serial address which passes by the serial port. This feature is enabled by default when setting the SM2 bit in SCON to use multiprocessor communication mode. In the 9-bit UART modes, Mode 2 and Mode 3, the Receive Interrupt flag (RI) will be automatically set when the received byte contains either the “Given” address or the “Broadcast” address. The 9-bit mode requires that the 9th information bit is a 1 to indicate that the received information is an address and not data.

In 8-bit Mode 1 the RI flag will be set if SM2 is enabled and the information received has a valid stop bit following the 8 address bits and the information is either a Given or Broadcast address.

Using the Automatic Address Recognition feature allows a master to selectively communicate with one or more slaves by invoking the given slave address or addresses. All of the slaves may be contacted by using the Broadcast address. Two special function registers are used to define the slave’s address, SADDR (A9H), and the address mask, SADEN (B9H). SADEN is used to define which bits in SADDR are to be used and which bits are “don’t care”. The SADEN mask can be logically ANDed with the SADDR to create the “Given” address which the master will use for addressing each of the slaves. Use of the Given address allows multiple slaves to be recognized while excluding others. The following examples will help to show the versatility of this scheme:

Slave 0:
SADDR = 1100 0000
SADEN = 1111 1101
Given = 1100 00X0

Slave 1:
SADDR = 1100 0000
SADEN = 1111 1110
Given = 1100 000X

In the previous example SADDR is the same and the SADEN data is used to differentiate between the two slaves. Slave 0 requires a 0 in bit 0 and it ignores bit 1. Slave 1 requires a 0 in bit 1 and bit 0 is ignored. A unique address for slave 0 would be 1100 0010 since slave 1 requires a 0 in bit 1. A unique address for slave 1 would be 1100 0001 since a 1 in bit 0 will exclude slave 0. Both slaves can be selected at the same time by an address which has bit 0 = 0 (for slave 0) and bit 1 = 0 (for slave 1). Thus, both could be addressed with 1100 0000.

In a more complex system the following could be used to select slaves 1 and 2 while excluding slave 0:

Slave 0:
SADDR = 1100 0000
SADEN = 1111 1001
Given = 1100 0XX0

Slave 1:
SADDR = 1110 0000
SADEN = 1111 1010
Given = 1110 0X0X

Slave 2: SADDR = 1110 0000
SADEN = 1111 1100
Given = 1110 00XX

In the previous example the differentiation among the 3 slaves is in the lower 3 address bits. Slave 0 requires that bit 0 = 0 and it can be uniquely addressed by 1110 0110. Slave 1 requires that bit 1 = 0 and it can be uniquely addressed by 1110 and 0101. Slave 2 requires that bit 2 = 0 and its unique address is 1110 0011. To select Slaves 0 and 1 and exclude Slave 2, use address 1110 0100, since it is necessary to make bit 2 = 1 to exclude slave 2.

The Broadcast Address for each slave is created by taking the logical OR of SADDR and SADEN. Zeros in this result are trended as don't-cares. In most cases, interpreting the don't-cares as ones, the broadcast address will be FFH.

Upon reset SADDR and SADEN are loaded with 0s. This produces a given address of all "don't cares" as well as a Broadcast address of all "don't cares". This effectively disables the Automatic Addressing mode and allows the microcontroller to use standard 80C51-type UART drivers which do not make use of this feature.

9. 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. A master device will always drive its SCK and MOSI pins, even when not currently transmitting. 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 bused. 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 AT89S8253, 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. Note that because P1.7 starts high, enabling the SPI in master mode with CPOL = 0 will result in a falling edge on SCK and clearing either MSTR or SPE will generate a rising edge on SCK. To prevent the slaves from being clocked by these edges, all slaves should be disabled with either $\overline{SS}$ = 1 or SPE = 0 when enabling/disabling the master.

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 AT89S8253 datasheet 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 AT89S8253 datasheet.

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. Examples of SPI serial clock phase and polarity are shown in the diagrams in the AT89S8253 datasheet.

Only an AT89S8253 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. The SPIF and WCOL bits cannot be cleared directly by software. To clear bits SPIF and WCOL, first read SPSR and then read or write SPDR. This operation generally occurs within the normal operation of the SPI routine, e.g., polling the status of SPIF reads the SPSR register and then reading the received value from SPDR or sending the next value by writing SPDR automatically clears the flags.

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 are the EA and ES bits in SFR IE (A8H). EA is the global interrupt enable bit. The SPI shares an interrupt vector with the UART, so the UART enable ES must also be set. 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.

9.1 Buffered Mode

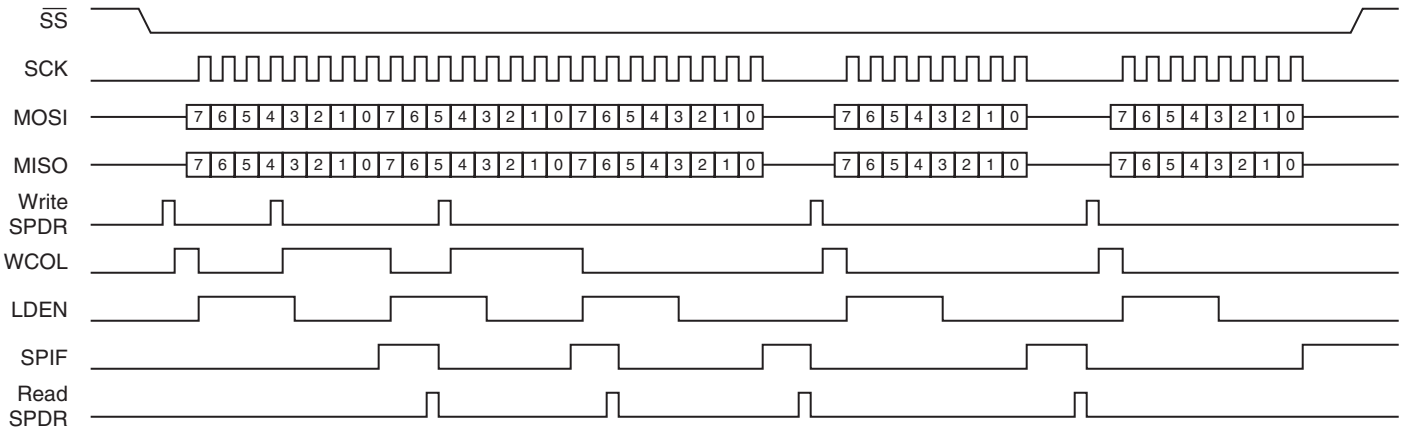
The SPI data register is normally double-buffered in the read direction but only single-buffered in the write direction. This means that received data from the previous transfer can be read while the current transfer is in progress, but the next byte to transmit cannot be queued while the current transfer is in progress. The AT89S8253 has the ability to also double-buffer SPDR in the write direction by setting the ENH bit in SPSR. Double-buffered mode is referred to as Enhanced Mode.

In enhanced mode, the Serial Peripheral Interface (SPI) on the AT89S8253 uses a double-buffered transmitter with WCOL as the buffer full flag. When data is written to SPDR, the transmit buffer is loaded with the data and the WCOL flag is set to show that the buffer is full. If the shift register is empty, as is the case for the first byte in a sequence, the contents of the buffer are transferred immediately to the shift register and WCOL is cleared low to flag buffer empty. Afterwards the user may queue data bytes in the transmit buffer by writing to SPDR while WCOL is low. WCOL is set high when SPDR is written, but the data is not transferred to the shift register. When the current byte transfer completes, the queued data in the buffer is moved to the shift register, WCOL is cleared, and the next transfer starts immediately. If a byte transfer completes with the buffer empty, the transmission halts until data is written to SPDR.

If the user waits until SPIF is set before loading the next data byte, gaps may occur between byte transfers. For higher utilization of the bus, the user may poll the status of WCOL to determine when to load the next data byte and thereby keep the transmitter full occupied.

The AT89S8253 includes the LDEN bit in SPSR to flag the first four bit slots of a transfer. The LDEN signal is not required for SPI operation. However, the LDEN signal may be used on slave devices to determine when an SPI transfer starts.

Figure 9-1. SPI in Enhanced Mode

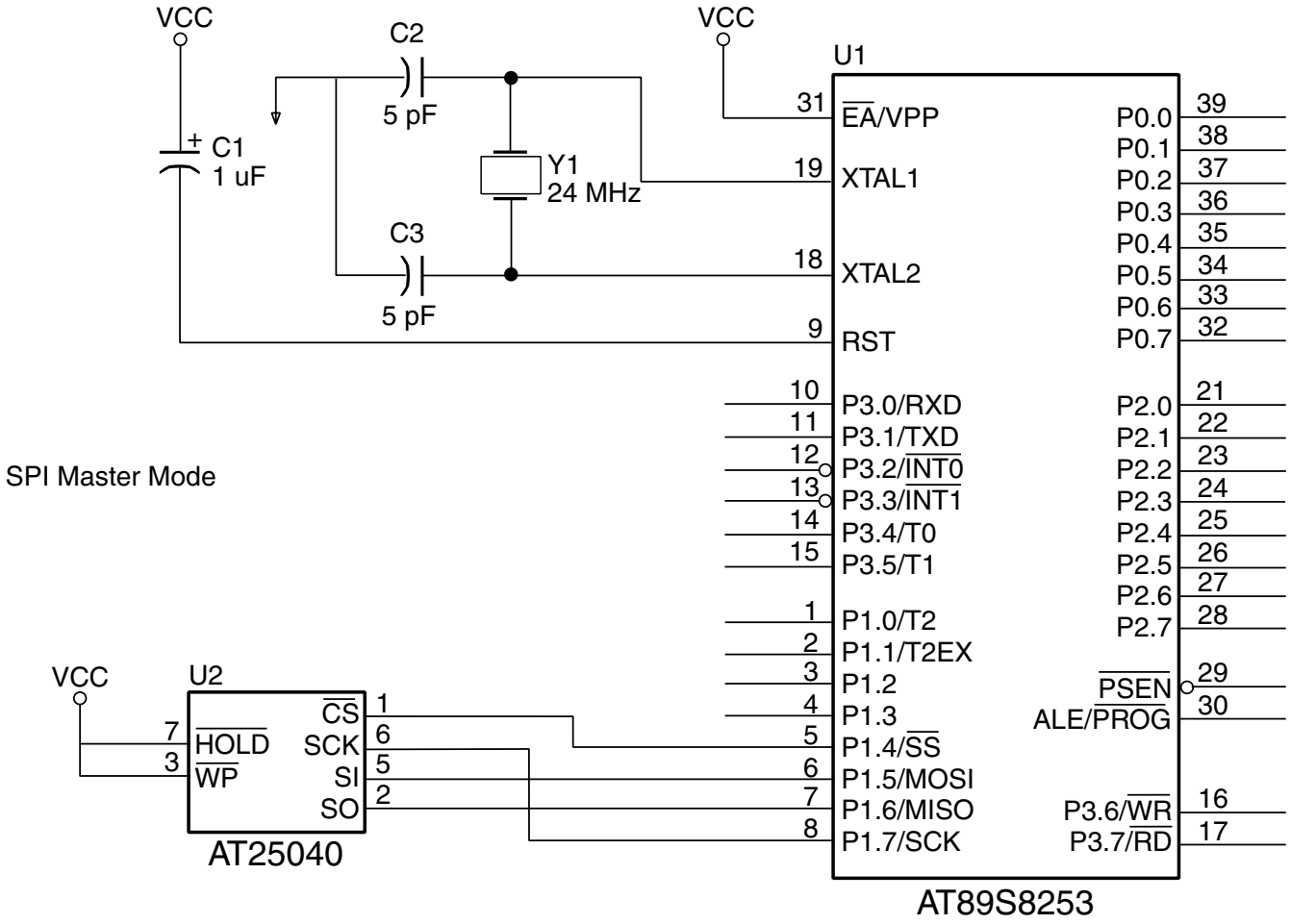


9.2 Listing 6: SPI Example

In the application shown below, the AT89S8253 is configured as an SPI master and interfaces to an Atmel AT25040 SPI-compatible EEPROM. The EEPROM provides 512 bytes of re-writable, nonvolatile storage while requiring only a four-pin interface to the microcontroller. The microcontroller and EEPROM are wired as shown in Figure 9-2. 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 Section 9.2. A SPI master must be configured to meet the requirements of its slaves. The AT25040 datasheet states that the maximum clock rate for the device is 2 MHz. The microcontroller's clock source is a 24-MHz crystal (Figure 9-2), so a SPI serial clock divisor of 16 was chosen to produce a serial clock of 1.5 MHz. As shown in the AT25040 datasheet, 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 9-2. AT89S8253 as an SPI Master



```
; 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
```

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3
France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex
France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR
Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn
Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
Tel: (33) 4-76-58-47-50
Fax: (33) 4-76-58-47-60

Literature Requests
www.atmel.com/literature