Chapter 11
PIC18F CCP AND SERIAL I/O

Chien-Chung Ho (何建忠)

PIC18F CCP Mode

- The CCP module is to provide measurement and control of time based pulse signals. Timer1 through Timer3 are used for this purpose.
  - Timer0 is not used by the CCP module.

- The CCP1 module of the PIC18F4321 is implemented as a standard CCP with enhanced PWM capabilities for better DC motor control. Hence, the CCP1 module in the PIC18F4321 is also called ECCP (Enhanced CCP). The CCP2 module is provided with standard capture, compare, and PWM features.
  - CCP1 and CCP2 are two independent modules. Hence, one can have CCP1 set up for PWM and CCP2 for capture or compare (or vice versa).
  - It may not be possible to use one CCP module (such as the CCP1) to do capture (or compare) and PWM at the same time. This is because, although the timers may be different (Timer1, Timer3), other logic functions are shared.

PIC18F CCP Mode

- The CCP module can be programmed to one of the following modes:
  - Capture mode (CCPx pin to be configured as input via programming)
  - Compare mode (CCPx pin to be configured as output via programming)
  - PWM mode (CCPx pin to be configured as output via programming)

- Both polled I/O and interrupt I/O can be used with the CCP module.
  - Note that the CCPxIF flag bit in the PIR1 register is monitored in a wait loop for polled I/O for the Capture and Compare modes while the TMR2IF flag bit in the T2CON register is checked in wait loop for the PWM mode.

PIC18F CCP Mode

- The PIC18F **Capture mode** causes the contents of a 16-bit timer (Timer1 or Timer3) to be written in a 16-bit register (CCPRxH:CCPRxL provided with the CCPx module) upon detecting an *n*th rising or falling edge of a pulse.
- Capture mode can be used to measure the length of time elapsed between two events.
  - Typical events include detecting every falling edge or every rising edge of a periodic waveform connected to the CCPx pin.
  - By capturing two consecutive rising edges, and then subtracting their values, the period of a waveform can be obtained.
  - Thus, the capture mode can be used to measure the period of an incoming periodic waveform.

PIC18F CCP Mode

- The PIC18F **Compare mode** is very similar to the timer function of a stopwatch (秒錶).
  - Upon starting the stopwatch, it counts down from the preset value to 0.
  - The PIC18F Compare mode works in a similar manner except that it counts from 0 to the preset value.
- The PIC18F Compare mode can be used to turn ON or OFF a device connected to the CCPx pin after a specified amount of time.
  - This can be accomplished by outputting a signal (e.g., a HIGH or a LOW by programming the CCPxCON rgister) as soon as the specified timer value (Timer1 or Timer3) matches with the 16-bit register value (CCPRxH:CCPRxL).
  - Hence, using the Compare mode, the PIC18F can be programmed to output a signal on the CCPx pin after a certain time delay.

PIC18F CCP Mode

- The PIC18F **PWM (Pulse Width Modulation) mode** can be used to generate a square wave on the CCPx pin with a user-specified frequency and duty cycle.
  - *"Duty cycle" is defined as the percentage of the time the pulse is high in a clock period*.
  - Also, as the duty cycle of a PWM signal increases, the average voltage and power provided by the PWM will increase.

- PWM mode can be used in applications such as dimming LEDs and controlling the speed of a brushless DC electric motor.
  - LEDs will be brighter as the PWM duty cycle increases while the speed of the motor will increase as the PWM duty cycle is increased.
  - LEDs will be dimmer with lower PWM duty cycle while the motor will run at a slower speed with lower PWM duty cycle.

CCP Registers

- Each CCP module is provided with an 8-bit control register (CCPxCON) shown in Figure 11.1. The CCPxCON (x = 1 or 2) can be used to select one of the three modes.
- Each CCP module also contains a 16-bit data register (CCPRx).
  - The 16-bit data register, in turn, is comprised of two 8-bit registers: CCPRxL (low byte) and CCPRxH (high byte).
  - This 16-bit data register can operate as a 16-bit Capture register, a 16-bit Compare register or an 8-bit PWM register (CCPRxL) holding the 8-bit decimal part of the duty cycle.

PIC18F4321 Capture mode

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **DCxB1:DCxB0**: PWM Duty Cycle bit 1 and bit 0 for CCP Module x
Capture mode:
Unused.
Compare mode:
Unused.
PWM mode:
These bits are the two Lower two bits (bit 1 and bit 0) of the 10-bit PWM duty cycle. The higher eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

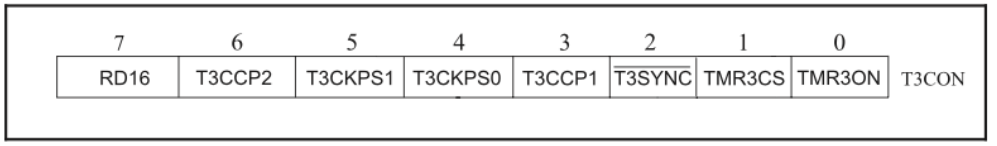
bit 3-0 **CCPxM3:CCPxM0**: CCPx Module Mode Select bits
0000 = Capture/Compare/PWM disabled (resets CCP module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode: initialize CCP pin low; on compare match, force CCP pin high (CCPxIF bit is set)
1001 = Compare mode: initialize CCP pin high; on compare match, force CCP pin low (CCPxIF bit is set)
1010 = Compare mode: generate software interrupt on compare match (CCPxIF bit is set, CCP pin reflects I/O state)
1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCPx match (CCPxIF bit is set)
11xx = PWM mode

FIGURE 11.1 CCPxCON Register

CCP modules and associated timers

- The CCP modules utilize Timers 1, 2 or 3, depending on the mode selected.
  - Timer1 and Timer3 are available to modules in Capture and Compare modes, while
  - Timer2 is available for modules in PWM mode.

- The assignment of a particular timer to a module is determined by the Timer to CCP enable bits in the T3CON register.



7	6	5	4	3	2	1	0	
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	T3CON

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for the CCP modules
01 = Timer3 is the capture/compare clock source for CCP2;
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for the CCP modules

bit 5-4 **T3CKPS1:T3CKPS0**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **T3SYNC:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CKI (on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

FIGURE 10.11 T3CON (Timer3 Control) Register

PIC18F4321 Capture mode

- Table 11.1 summarizes the timer assignments to each one of these modes in the CCP1 module.

TABLE 11.1 Assignment of Timers for the PIC18F4321 CCP1 module

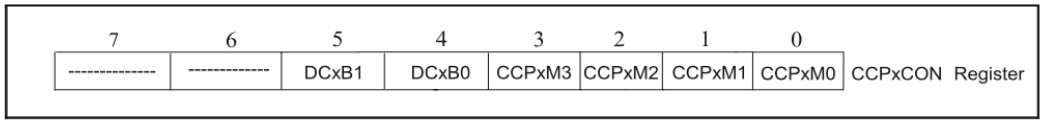
CCP mode selected	Timer
Capture mode	Timer1 or Timer3
Compare mode	Timer1 or Timer3
PWM mode	Timer2

PIC18F4321 Capture mode

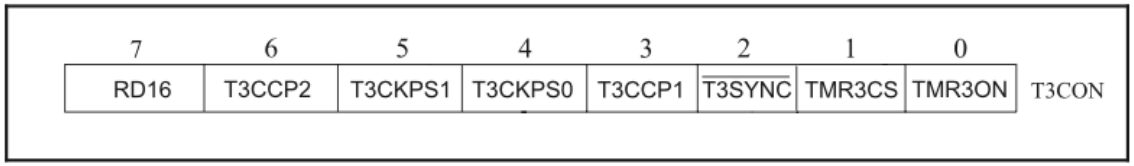
- In Capture mode, the CCPRxH:CCPRxL register pair captures the 16-bit value of the TMR1 or TMR3 registers when an event (such as every rising or falling edge) occurs on the corresponding CCPx pin. The event is selected by the mode select bits, CCPxM3:CCPxM0 (bits 3-0 of CCPxCON).
  - When a capture is made, the CCPx interrupt flag bit such as CCP1IF in the PIR1 register is set to one; it must be cleared to zero in software.
  - If another capture occurs before the value in register CCPRx is read, the old captured value is overwritten by the new captured value; hence, the old captured value should be saved.
- In Capture mode, the CCPx pin should be configured as an input by setting the corresponding TRISx direction bit. Also, the timers that are to be used with the capture mode (Timer1 or Timer3) must be running in Timer mode.
  - The timer to be used with each CCP module is selected using the T3CON register.

PIC18F4321 Capture mode

- The following steps can be used to program the PIC18F4321 in capture mode to determine the period of a waveform (assume CCP1; similar procedure for CCP2):
  1. Load the CCP1CON register with appropriate data for capture mode.



  2. Configure CCP1 as an input pin using the TRISC register. Note that the CCP1 pin is multiplexed with the RC2 pin.
  3. Select Timer1 or Timer3 for the capture mode by loading appropriate data into T3CON register. T1CON is not used for Timer1 selection. Instead, only T3CON is used for selecting Timer1 or Timer3.



  4. Clear the CCP1 flag: bit CCP1IF for CCP1 (in PIR1) or bit CCP2IF for CCP2 (in PIR2), after a capture so that the next capture can be made.

PIC18F4321 Capture mode

5. Clear CCPR1H and CCPR1L to 0.
6. Check CCP1IF flag in PIR1 and wait in a loop until CCP1IF is 1 for the first rising edge. As soon as the first rising edge is detected, start Timer1 using T1CON (or Timer3 using T3CON).
7. Save CCPR1H and CCPR1L in data memory such as REGX and REGY.
8. Clear CCP1IF to 0.
9. Check CCP1IF flag in PIR1 and wait in a loop until CCP1IF is 1 for the second rising edge. As soon as the second rising edge is detected, stop Timer1 (or Timer3).
10. Disable capture by clearing CCP1CON register.
11. Perform 16-bit subtraction: [CCPR1H:CCPR1L] - [REGX:REGY] and store the result in [REGX:REGY].
12. A 16-bit result in register pair [REGX:REGY] will contain the period of the incoming waveform in terms of the number of clock cycles.

PIC18F4321 Capture mode

- Typical applications of the capture mode include:
  - Measurement of the pulse width of an unknown periodic signal by capturing the subsequent leading (rising) or trailing (falling) edges of a pulse.
  - Measurement of the period of a signal by capturing two subsequent leading or trailing edges.
  - Measurement of duty cycle. Note that the duty cycle is defined as (t1/T) x 100 where t1 is the fraction of the time the signal is HIGH in a period, T.

PIC18F4321 Capture mode

Example 11.1 Assume PIC18F4321.

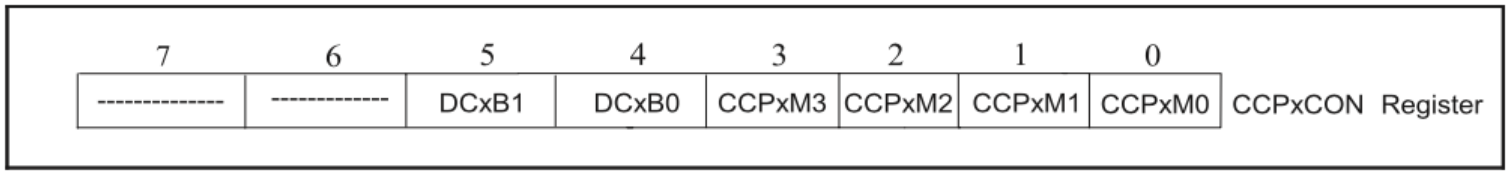
(a) Write a PIC18F assembly language program at address 0x200 to measure the period (in terms of the number of clock cycles) of an incoming periodic waveform connected at CCP1 pin. Store the result in registers 0x21 (high byte) and 0x20 (low byte). Use Timer3, and capture mode of CCP1.

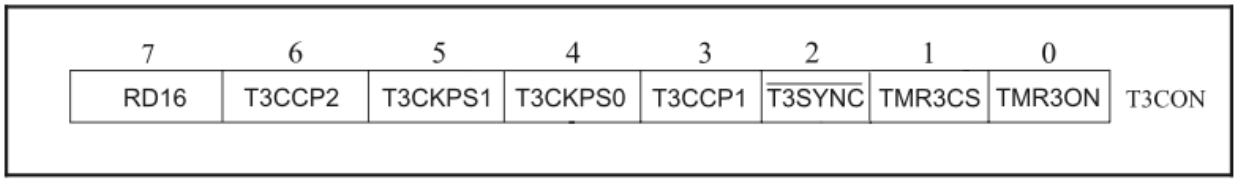
(b) Write a C language program to measure the period (in terms of the number of clock cycles) of an incoming periodic waveform connected to the CCP1 pin. Use Timer3, and the capture mode of CCP1.

PIC18F4321 Capture mode

The PIC18F assembly language program is provided below:

```
INCLUDE   <P18F4321.INC>
ORG       0x200
MOVLW     B'00000101'      ; Select capture mode rising edge
MOVWF     CCP1CON
BSF       TRISC, CCP1      ; Configure CCP1 pin as input
MOVLW     B'01000000'      ; Select TIMER3 as clock source for capture
MOVWF     T3CON            ; Select TIMER3 internal clock, 1:1 prescale
                           ; TIMER3 OFF
CLRF      CCPR1H           ; Clear CCPR1H to 0
CLRF      CCPR1L           ; Clear CCPR1L to 0
BCF       PIR1, CCP1IF     ; Clear CCP1IF
```





PIC18F4321 Capture mode

```
WAIT    BTFSS   PIR1, CCP1IF        ; Wait for the first rising edge
        GOTO    WAIT
        BSF     T3CON, TMR3ON       ; Turn Timer3 ON
        MOVFF   CCPR1L, 0x20        ; Save CCPR1L in 0x20 at 1st rising edge
        MOVFF   CCPR1H, 0x21        ; Save  CCPR1H in 0x21 at 1st rising edge
        BCF     PIR1, CCP1IF        ; Clear CCP1IF
WAIT1   BTFSS   PIR1, CCP1IF        ; Wait for next rising edge
        GOTO    WAIT1
        BCF     T3CON, TMR3ON       ; Turn OFF  Timer3
        CLRF    CCP1CON             ; Disable capture
        MOVF    0x20, W             ; Move 1st Low byte to WREG
        SUBWF   CCPR1L, F           ; Subtract WREG from 2nd low byte
                                    ; Result in 0x20
        MOVF    0x21, W             ; Move 1st HIGH byte to WREG
        SUBWFB  CCPR1H, F           ; Subtract WREG with borrow
                                    ; from 2nd high byte, result in 0x21
        SLEEP                       ; Halt
        END
```

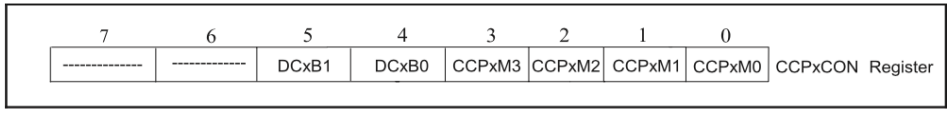
PIC18F4321 Capture mode

(b)

```
#include<p18f4321.h>
void main(void)
{
    unsigned char FIRST_CCPR1L, FIRST_CCPR1H, HIGH_BYTE, LOW_
BYTE;
    CCP1CON=0x05;              // Select capture mode rising edge
    TRISCbits.TRISC2=1;        // Configure RC2/CCP1/P1A pin as input
    T3CON=0x40;                // Select TIMER3 as clock source for capture
    CCPR1H=0x00;               // Clear CCPR1H to 0
    CCPR1L=0x00;               // Clear CCPR1L to 0
    PIR1bits.CCP1IF=0;         // Clear CCP1IF

    while(PIR1bits.CCP1IF==0); // Wait for the first rising edge

    T3CONbits.TMR3ON=1;        // Turn Timer3 ON
    FIRST_CCPR1L=CCPR1L;       // Save CCPR1L in FIRST_CCPR1L at 1st
                               // rising edge
    FIRST_CCPR1H=CCPR1H;       // Save CCPR1H in FIRST_ at 1st rising edge
    PIR1bits.CCP1IF=0;         // Clear CCP1IF

    while(PIR1bits.CCP1IF==0); // Wait for next rising edge

    T3CONbits.TMR3ON=0;        // Turn OFF  Timer3
    CCP1CON=0x00;              // Disable capture
    LOW_BYTE=CCPR1L–FIRST_CCPR1L;        // Low byte of result
    HIGH_BYTE=CCPR1H–FIRST_CCPR1H;       // High byte of result

    while(1);                  // Halt
}
```

PIC18F4321 Compare mode

- In Compare mode, the 16-bit CCPRx (CCPR1H:CCPR1L for CCP1 or CCPR2H: CCPR2L for CCP2) register value is constantly compared to the value in either the TMR1 or TMR3 register. When a match occurs, the CCPx pin can be:
  - driven high, driven low, toggled (high-to-low or low-to-high; complemented), remain unchanged
- The action on the pin is based on the value of the mode select bits (CCPxM3:CCPxM0) in CCPxCON register.
  - As soon as a match occurs, the CCPx flag bit, CCPxIF, is set to one.
  - The user must configure the CCPx pin as an output by clearing the appropriate TRIS bit. Timer1 or Timer3 must be running in Timer mode.
- Typical applications of the compare mode include activating an event after a certain time delay, generating a pulse train or a waveform with a specific duty cycle.



0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode: initialize CCP pin low; on compare match, force CCP pin high (CCPxIF bit is set)
1001 = Compare mode: initialize CCP pin high; on compare match, force CCP pin low (CCPxIF bit is set)
1010 = Compare mode: generate software interrupt on compare match (CCPxIF bit is set, CCP pin reflects I/O state)
1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCPx match (CCPxIF bit is set)

PIC18F4321 Compare mode

- The following steps can be used in compare mode:
  1. Load the CCPxCON with appropriate data for compare mode. Configure the CCPx pin of PORTC as an output.
  2. Load the CCPR1H:CCPRIL (or CCPR2H:CCPR2L) register pair with appropriate values.
  3. Select Timer1 (or Timer3) as the clock source in the Compare mode by loading appropriate data into T3CON register. T1CON is not used to select the clock source for Compare mode. Instead, T3CON is used to select Timer1 or Timer3 as the clock source.
  4. Initialize Timer1H:Timer1L (or Timer3H:Timer3L) to 0.
  5. Clear CCP1IF in PIR1 (or CCP2IF in PIR2) to 0.
  6. Start Timer1 using T1CON (or Timer3 using T3CON).
  7. Wait in a loop until the CCP1IF (or CCP2IF) is HIGH.
  8. As soon as a match occurs (CCP1IF or CCP2IF HIGH), stop timer1 using T1CON (or Timer3 using T3CON), and the programmed bit value (HIGH or LOW or Toggled or Unchanged) will be outputted to the CCPx pin.

PIC18F4321 Compare mode

Example 11.2 Assume PIC18F4321.

(a) Write a PIC18F assembly language program at address 0x100 that will toggle the CCP1 pin after a time delay of one ms. Use Timer3, and compare mode of CCP1. Use 8MHz internal clock.

(b) Write a C language program that will toggle the CCP1 pin after a time delay of one ms. Use Timer3 and Compare mode of CCP1. Use 8MHz internal clock.

Solution

With an 8-MHz internal crystal, Fosc = 8 MHz. Since Timer3 uses Fosc/4. Timer clock frequency = Fosc/4 = 2 MHz. Hence, clock period of Timer3 = 0.5 μsec. Counter value = (1 ms)/(0.5 μsec) = 2000_{10} = 0x07D0. Hence, CCPR1H :CCPR1L should be loaded with 0x07D0 for the PIC18F4321 compare mode.

PIC18F4321 Compare m

7	6	5	4	3	2	1	0	
----------	----------	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0	CCPxCON Register

0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode: initialize CCP pin low; on compare match, force CCP pin high (CCPxIF bit is set)
1001 = Compare mode: initialize CCP pin high; on compare match, force CCP pin low (CCPxIF bit is set)
1010 = Compare mode: generate software interrupt on compare match (CCPxIF bit is set, CCP pin reflects I/O state)
1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCPx match (CCPxIF bit is set)

(a) The PIC18F assembly language program is provided below

```
        INCLUDE   <P18F4321.INC>
        ORG       0x100
        MOVLW     0x70          ; 8MHz internal clock
        MOVWF     OSCCON
        MOVLW     0x02          ; Select compare mode, toggle CCP1 pin
        MOVWF     CCP1CON       ; on match
        BCF       TRISC, CCP1   ; Configure CCP1 pin as output
        MOVLW     0x40          ; Select TIMER3 as clock source for
                                ; compare
        MOVWF     T3CON         ; Select TIMER3 internal clock, 1:1 prescale
                                ; TIMER3 OFF
        MOVLW     0x07          ; Load CCPR1H with 0x07
        MOVWF     CCPR1H
        MOVLW     0xD0          ; Load CCPR1L  with 0xD0
        MOVWF     CCPR1L
        CLRF      TMR3H         ; Initialize TMR3H to 0
        CLRF      TMR3L         ; Initialize TMR3L to 0
        BCF       PIR1, CCP1IF  ; Clear CCP1IF
        BSF       T3CON, TMR3ON ; Start Timer3
WAIT    BTFSS     PIR1, CCP1IF  ; Wait in a loop until CCP1IF is 1. CCP1 pin
        BRA       WAIT          ; toggles when match occurs
        BCF       T3CON, TMR3ON ; Stop Timer3
HERE    BRA       HERE          ; Halt
        END
```

PIC18F4321 Compare mode

The C language program is provided below:

```
#include<p18f4321.h>
void main(void)
{
        OSCCON = 0x70;              // 8MHz internal clock
        CCP1CON=0x02;               // Select compare mode, toggle CCP1 pin
                                    // on match
        TRISCbits.TRISC2=0;         // Configure CCP1 pin as output
        T3CON=0x40;                 // Select TIMER3 as clock for compare,
                                    // 1:1 prescale
        CCPR1H=0x07;                // Load CCPR1H with 0x07
        CCPR1L=0xD0;                // Load CCPR1L  with 0xD0
        TMR3H=0;                    // Initialize TMR3H to 0
        TMR3L=0;                    // Initialize TMR3L to 0
        PIR1bits.CCP1IF=0;          // Clear CCP1IF
        T3CONbits.TMR3ON=1;         // Start Timer3

        while(PIR1bits.CCP1IF==0); // Wait in a loop until CCP1IF is 1

        T3CONbits.TMR3ON=0;         // Stop Timer3

        while(1);                   // Halt
}
```

PIC18F4321 Compare mode

Example 11.3 Write a C-program to generate a waveform with a 100 ms period and a 75% duty cycle on the CCP1 pin of the PIC18F4321. Use Compare mode, Timer3, and one MHz crystal.

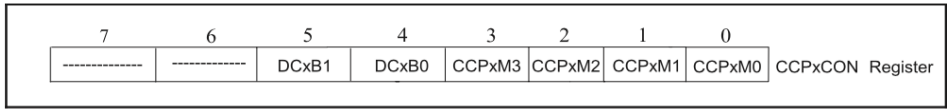
Solution

With 1-MHz OSC, Frequency = Fosc/4 = 0.25MHz
Period = 1/0.25MHz = 4 microseconds, Waveform period is 100 milliseconds

With 75% HIGH during a period, the waveform will be HIGH for 75 ms, LOW for 25 ms.

Hence, with 1:1 prescaler, Counter value = (75milliseconds)/(4 microseconds) = 18,750 (decimal) or 0x493E. Therefore, 0x493E should be loaded into CCPR1H:CCPR1L for 75% HIGH during a period.

Also, with 25% LOW during a period and with a 1:1 prescaler, Counter value = (25milliseconds)/(4 microseconds) = 6250 (decimal) or 0x186A. Hence, 0x186A should be loaded into CCPR1H:CCPR1L for 25% LOW during a period.



PIC18F4321 Compare mode

```
void main(void)
{
    CCP1CON=0x02;              // Select compare mode, toggle CCP1 pin on match
    TRISCbits.TRISC2=0;        // Configure CCP1 pin as output
    T3CON=0x40;                // Select TIMER3 as clock for compare, 1:1 prescale
    while(1)
    {
       //--LOW:
     CCPR1H=0x18;              // Load CCPR1H
     CCPR1L=0x6A;              // Load CCPR1L
     TMR3H=0;                  // Initialize TMR3H to 0
     TMR3L=0;                  // Initialize TMR3L to 0
     PIR1bits.CCP1IF=0;        // Clear CCP1IF
     T3CONbits.TMR3ON=1;       // Start Timer3
     while(PIR1bits.CCP1IF==0);          // Wait in a loop until CCP1IF is 1
     T3CONbits.TMR3ON=0;       // Stop Timer3
     //--HIGH:
     CCPR1H=0x49;              // Load CCPR1H
     CCPR1L=0x3E;              // Load CCPR1L
     TMR3H=0;                  // Initialize TMR3H to 0
     TMR3L=0;                  // Initialize TMR3L to 0
     PIR1bits.CCP1IF=0;        // Clear CCP1IF
     T3CONbits.TMR3ON=1;       // Start Timer3
     while(PIR1bits.CCP1IF==0);          // Wait in a loop until CCP1IF is 1
     T3CONbits.TMR3ON=0;       // Stop Timer3
    }
}
```

0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode: initialize CCP pin low; on compare match, force CCP pin high (CCPxIF bit is set)
1001 = Compare mode: initialize CCP pin high; on compare match, force CCP pin low (CCPxIF bit is set)
1010 = Compare mode: generate software interrupt on compare match (CCPxIF bit is set, CCP pin reflects I/O state)
1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCPx match (CCPxIF bit is set)

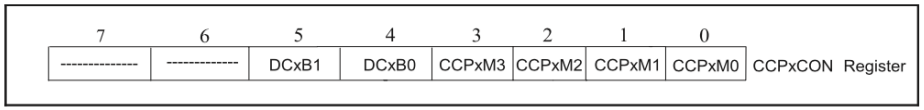
PIC18F4321 PWM mode

- In Pulse-Width Modulation (PWM) mode, the CCPx pin can be configured as an output to generate a periodic waveform with a specified frequency, and a 10-bit (8-bit integer part and 2-bit fractional part) duty cycle.
  - The PWM duty cycle is specified by writing the upper eight bits (integer part) into the CCPRxL register, and the lower two bits (fractional part) of the CCPxCON register (bits 5 and 4).

- Timer2 is used for the PWM mode. The PWM period is specified by writing to the 8-bit PR2 register in the CCP module.
- The PWM period can be calculated using the following formula:
  - PWM Period = [(PR2) + 1] x 4 x Tosc x (TMR2 Prescale Value)
  - Tosc = (1/Fosc), Fosc is the crystal frequency, TMR2 Prescale Value can be initialized as 1, 4, or 16 using the T2CON register.
  - Hence, PR2 = [(Fosc)/(4 x Fpwm x TMR2 Prescale Value)] – 1
  - Note that PWM frequency (Fpwm) is defined as 1/[PWM period].

PIC18F4321 PWM mode

- The following equation is used to calculate the PWM duty cycle in time:
  - PWM Duty Cycle = (CCPxL:CCPxCON<5:4>) × Tosc × (TMR2 Prescale Value).
  - The duty cycle is defined as the percentage of the time the pulse is high in a clock period.
  - Consider a 25% duty cycle. Since duty cycle is a fraction of the PR2 register value, decimal value for the duty cycle with a PR2 value of 30 is 7.5 (0.25 x 30).
    - 8-bit 00000111 must be loaded into CCPRxL
    - 2-bit 10 (0.5 decimal) must be loaded for DCxB1 and DCxB0 bits in the CCPxCON register
    - This will assign a duty cycle of 7.5 to the PWM waveform to be generated at the CCPx pin.



7	6	5	4	3	2	1	0	
----------	----------	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0	CCPxCON Register

bit 5-4 **DCxB1:DCxB0**: PWM Duty Cycle bit 1 and bit 0 for CCP Module x
Capture mode:
Unused.
Compare mode:
Unused.
PWM mode:
These bits are the two Lower two bits (bit 1 and bit 0) of the 10-bit PWM duty cycle. The higher eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

PIC18F4321 PWM mode

- The CCP1 PWM output waveform with the specified duty cycle in CCPR1L:CCP1CON register is generated as follows:
  1. CCPR1L is copied to CCPR1H. Two bits of CCP1CON <5:4> are latched internally to provide a 10-bit duty cycle.
  2. Also, an 8-bit TMR2 value is concatenated with a two-bit internal latch to create a 10-bit duty cycle.
  3. After TMR2 is started from 0, the CCP1 pin goes to HIGH to indicate the start of a cycle. The CCPR1H and two-bit latch values are compared with TMR2 and two-bit latch values for a match. Once the match occurs, the CCP1 pin goes to LOW. The waveform will be HIGH for the duration specified by the duty cycle.
  4. TMR2 keeps incrementing. As soon as the contents of TMR2 and PR2 match, CCP1 pin is driven to HIGH, and TMR2 is cleared to 0. This completes a cycle, and another cycle is then started. The same process continues for subsequent cycles.

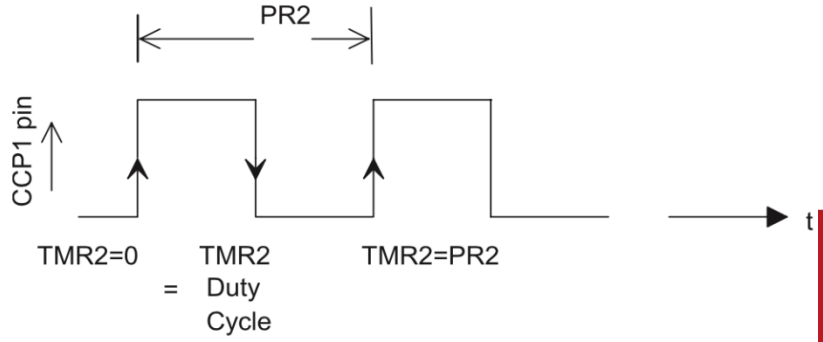


FIGURE 11.2 CCP1 waveform in PWM mode

PIC18F4321 PWM mode

- The following procedure should be followed when configuring the CCP module for PWM operation:
  1. The PR2 register should be initialized with the PWM period.
  2. Load the PWM duty cycle by writing the 8-bit integer value into the CCPRxL register, and the two-bit fractional part into bits 5, 4 of CCPxCON (Figure 11.1). Also, disable PWM mode by programming the CCPxCON register.
  3. Configure the CCPx pin an output by clearing the appropriate TRISCx bit.
  4. Select the Timer2 prescale value, then enable Timer2 by writing to T2CON.
  5. Enable PWM mode by writing appropriate data into CCPxCON register.
  6. Initialize TMR2 register to 0.
  7. Turn Timer2 ON by writing appropriate data into T2CON register.
  8. Clear TMR2IF flag in the PIR1 register to 0.
  9. Wait in a loop for the TMR2IF flag to become HIGH. As soon as TMR2IF is HIGH, the PWM waveform is generated at the CCPx pin with the specified duty cycle.
  10. This will complete a cycle.

PIC18F4321 PWM mode

- PWM waveform can be generated by polling the TMR2IF in the PIR1 register. The following PIC18F assembly language instruction sequence and C-code will poll the TMR2IF flag bit in the PIR1 register by clearing TMR2IF to 0, turning the TMR2 On, and then waiting in a loop for the TMR2IF bit to become HIGH indicating completion of a cycle:

In PIC18F assembly:

```
REPEAT  BCF     PIR1, TMR2IF        ; Clear TMR2IF flag to 0
BACK    BSF     T2CON, TMR2ON       ; Turn Timer2 ON
        BTFSS   PIR1, TMR2IF        ; wait until completion of a cycle
        GOTO    BACK
        GOTO    REPEAT              ; Repeat
```

In C:

```
PIR1bits.TMR2IF=0;                  // Clear TMR2IF flag to 0
T2CONbits.TMR2ON=1;                 // Turn Timer2 ON
while(PIR1bits.TMR2IF==0);          // Wait until TMR2IF = 1
```

PIC18F4321 PWM mode

Example 11.4

(a) Write a PIC18F assembly language program at address 0x100 to generate a 4 KHz PWM with a 50% duty cycle on the CCP1 pin of the PIC18F4321. Assume 4 MHz crystal.

(b) Write a C language program to generate a 4 KHz PWM with a 50% duty cycle on the CCP1 pin of the PIC18F4321. Assume 4 MHz crystal.

Solution

PR2 = [(Fosc)/(4 x Fpwm x TMR2 Prescale Value)] - 1
PR2 = [(4 MHz)/(4 x 4 KHz x 1)] - 1 assuming Prescale value of 1
PR2 = 249. With 50% duty cycle = 0.5 x 249 = 124.5. Hence, the CCPR1L register will be loaded with 124, and bits DC1B1:DC0B0 (CCP1CON register) with 10 (binary).

PIC18F4321 PWM mode

(a) The PIC18F assembly language program is provided below:

```
        INCLUDE  <P18F4321.INC>
        ORG      0x100
        MOVLW    0x60            ; 4MHz internal clock
        MOVWF    OSCCON
        MOVLW    D'249'          ; Initialize PR2 register
        MOVWF    PR2
        MOVLW    D'124'          ; Initialize CCPR1L
        MOVWF    CCPR1L
        MOVLW    0x20            ; CCP1 OFF,
        MOVWF    CCP1CON         ; DC1B1:DC0B0=10
        BCF      TRISC, CCP1     ; Configure CCP1 pin as output
        CLRF     T2CON           ; 1:1 prescale, Timer2 OFF, no postscale
        MOVLW    0x2C            ; PWM mode
        MOVWF    CCP1CON
        CLRF     TMR2            ; Clear Timer2 to 0
BACK    BCF      PIR1, TMR2IF    ; Clear TMR2IF to 0
        BSF      T2CON, TMR2ON   ; Turn Timer2 ON
WAIT    BTFSS    PIR1, TMR2IF    ; Wait until TMR2IF is HIGH
        GOTO     WAIT
        BRA      BACK
        END
```

PIC18F4321 PWM mode

- In the program, the value of TMR2 is compared to the period register, PR2, on each clock cycle. When the two values match, a comparator generates a match signal as the timer output. This signal also resets the value of TMR2 to 0x00 on the next cycle. In the above program, the last instruction BRA BACK branches to the label where the TMR2IF flag in the PIR1 is cleared to 0. The program does not have to go back to clear TMR2 to 0 since the TMR2 is automatically cleared after each match.

PIC18F4321 PWM mode

The C language program is provided below:

```
#include<p18f4321.h>
void main(void)
{
    OSCCON =0x60;                   // 4MHz internal clock
    PR2=249;                        // Initialize PR2 register
    CCPR1L=124;                     // Initialize CCPR1L
    CCP1CON=0x20;                   // CCP1 OFF, DC1B1:DC0B0=10
    TRISCbits.TRISC2=0;             // Configure CCP1 pin as output
    T2CON =0;                       // 1:1 prescale, Timer2 OFF, no postscale
    CCP1CON=0x2C;                   // PWM mode
    TMR2=0;                         // Clear Timer2 to 0

    while(1)
    {
    PIR1bits.TMR2IF=0;              // Clear TMR2IF to 0
    T2CONbits.TMR2ON=1;             // Turn Timer2 ON
    while(PIR1bits.TMR2IF==0);      // Wait until TMR2IF is HIGH
    }

}
```

DC Motor Control

- The duty cycle of a PWM signal is directly proportional to the average voltage and power provided by the PWM.
- Because of this feature, the PWM mode can be used in applications such as dimming LEDs and controlling the speed of a brushless DC electric motor.
  - This means that LEDs will be brighter as the PWM duty cycle is increased.
  - Similarly, the speed of a DC motor will increase as the PWM duty cycle is increased.
  - On the other hand, LEDs will be dimmer with a lower PWM duty cycle while the motor will run at a slower speed.
- In earlier days, voltage regulator circuits were used to control the speed of a DC motor. But voltage regulators dissipate (浪費) a lot of power.
  - PIC18F in the PWM mode is used to control the speed of a DC motor. In this scheme, power dissipation is significantly reduced by turning the driving voltage to the motor ON and OFF.

DC Motor Control

- Sometimes, it is desirable to change direction of rotation of the DC motor. This can be accomplished by reversing the direction of the motor via software by interfacing a device called H-Bridge to an I/O port of the PIC18F.
  - Note that the speed of the motor, on the other hand, can be controlled using the PWM mode and by connecting the DC motor to the CCP1 pin of the PIC18F4321.
- Microcontrollers such as the PIC18F4321 are not capable of outputting the required large current and voltage to control a typical DC motor. Hence, a driver such as the CNY17F Optocoupler is needed to amplify the current and voltage provided by the PIC18F's output, and provide appropriate levels for the DC motor.

DC Motor Control

Potentiometer: 電位計、分壓計

Example 11.5 It is desired to change the speed of a DC motor by dynamically changing its pulse width using a potentiometer connected to bit 0 of PORTB (Figure 11.3 (a)). Note that the PWM duty cycle is controlled by the potentiometer. The PIC18F4321 will input the potentiometer voltage via its on-chip A/D converter as eight bits, generate an 800-Hz PWM waveform on the CCP1 pin, and then change the speed of the motor as the potentiometer voltage is varied. Use a 4 a MHz crystal and a TMR2 prescaler value of 16. Ignore the fractional part of the duty cycle. Note that the circuit in Figure 11.3 (b) can be used instead of the optocoupler CNY17F.

Using interrupt-driven ADC:

(a) write a PIC18F assembly language program to accomplish this.

(b) write a C language program to accomplish this.

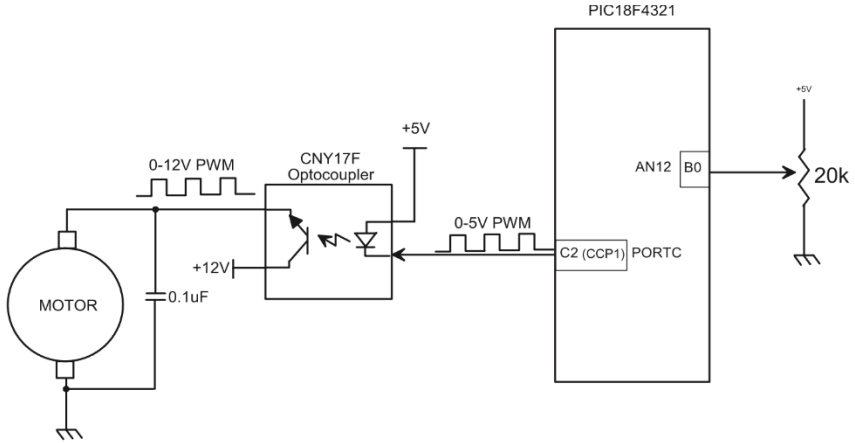


FIGURE 11.3(a) Figure for Example 11.5

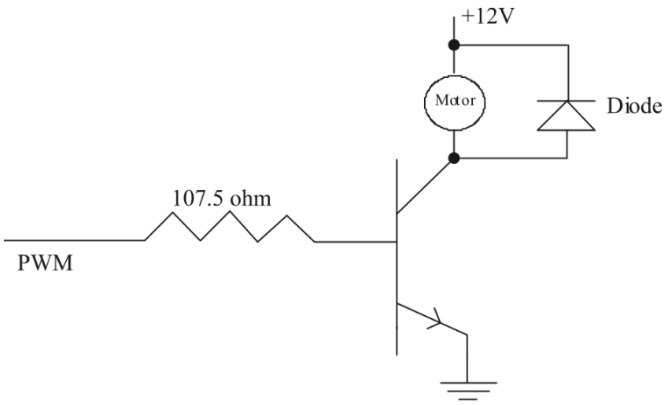


FIGURE 11.3(b) Circuit to be used instead of the Optocoupler

DC Motor Control

PR2 = [(Fosc)/(4 x Fpwm x TMR2 Prescale Value)] - 1
= [(4 MHz)/(4 x 800x 16)] - 1
= 77.125
Hence, PR2 will be 77 approximately.

After converting the potentiometer voltage into 8-bit binary, the ADRESH will contain the converted data. The contents of ADRESH can be moved to CCPR1L to represent the decimal portion of the duty cycle. In order for the duty cycle to be in the range of 0 to 77, the contents of ADRESH must be divided by 3 (255/77 = 3 approximately). This will ensure that the decimal portion of the duty cycle is between 0 and 77. The higher the voltage across the potentiometer, the higher the value of ADRESH/3.

DC Motor Control

```
INCLUDE <P18F4321.INC>
ORG         0x000
GOTO        MAIN
ORG         0x0008              ; Program will go to CHECK_INT
                                ; at interrupt
GOTO        ADIF_ISR            ; Go to Interrupt Service Routine
ORG         0x70                ; Start of the main program
```

```
MAIN    MOVLW   0x60
        MOVWF   OSCCON      ; Setting the internal clock to 4MHz
        MOVLW   0x10        ; Initialize STKPTR
        MOVWF   STKPTR      ; since interrupt is used
        MOVLW   0x00
        MOVWF   TRISC       ; Make PORTC output
        MOVLW   0x02
        MOVWF   T2CON       ; Configure Timer2 with prescale 16 and
                            ; no postscale
        CLR     TMR2
        BCF     PIR1, TMR2IF ; Clears Timer2  flag
        MOVLW   0x31        ; Use AN12 as ADC
        MOVWF   ADCON0
        MOVLW   0x00
        MOVWF   ADCON1      ; Enable pins for analog input
        MOVLW   0x08
        MOVWF   ADCON2      ; 2 TAD and Fosc/2, Left justified
        MOVLW   0x0C
        MOVWF   CCP1CON     ; Select PWM mode
        MOVLW   D'77'       ; Set period of PWM signal
        MOVWF   PR2
```

```
START       BSF     ADCON0,GO          ; Start the ADC
WAIT        BRA     WAIT               ; Wait for ADC to interrupt
            BRA     START              ; Start ADC again
            ORG     0x200              ; start of the Service routine
ADIF_ISR    MOVFF   ADRESH, 0x20       ; Move value in ADRESH to 0x20
                                       ; Divide by 3 using repeated addition
            MOVLW   0x03               ; Move 3 into the WREG
            CLRF    0x21               ; Clear value in 0x21
DIVIDE      CPFSGT  0x20               ; Compare the value to 3 skip if greater than
            BRA     FINISHED           ; Division is done
            INCF    0x21, F            ; Increment 0x21
            SUBWF   0x20               ; Subtract 3 from 0x20
            BRA     DIVIDE             ; Subtract again
FINISHED    MOVFF   0x21, CCPRIL       ; Move final value into CCPRIL
            BCF     PIR1, TMR2IF       ; Clear Timer2 flag
            BSF     T2CON, TMR2ON      ; Turn on Timer2
AGAIN       BTFSS   PIR1, TMR2IF       ; Wait until cycle is over
            BRA     AGAIN
            RETFIE                     ; Return to start ADC again
            END
```

DC Motor Control

(b) The C-program is provided below:

```
#include <p18f4321.h>
void ADINT_ISR (void);
#pragma interrupt check_int
void check_int(void)
{     ADINT_ISR();
```

DC Motor Control

```
}
#pragma code ADC_INT=0x08    // At interrupt, code jumps here
void ADC_Int (void)
{     _asm
     GOTO check_int
     _endasm
}
#pragma code                 // End of Code
void main ()
  {      OSCCON=0x60;        // 4 MHz
         PIE1bits.ADIE = 1;  // enable ADC interrupt
         PIR1bits.ADIF = 0;  // clear ADC interrupt flag
         INTCONbits.PEIE = 1;// enable peripheral interrupt
         INTCONbits.GIE = 1; // enable global interrupt
         TRISC=0;            // CCP1 output
         T2CON=0x02;         // Configure Timer2 with prescale 16
         PIR1bits.TMR2IF=0;  // Clear Timer2 interrupt flag
```

DC Motor Control

```
    ADCON0=0x31;            // AN12
    ADCON1=0x00;            // Enable analog pins
    ADCON2=0x08;            // 2 TAD and Fosc/2, Left justified8-bit in ADRESH
    CCP1CON=0x0C;           // PWM enable
    PR2=77;
    TMR2 = 0;
    ADCON0bits.GO = 1;

   while (1);                    // Wait for interrupt
}
   void ADINT_ISR (void)
   {
        PIR1bits.ADIF = 0;
        CCPR1L=ADRESH/3;
        PIR1bits.TMR2IF=0;          // Clear Timer2 interrupt
        T2CONbits.TMR2ON=1;

    while(PIR1bits.TMR2IF==0);
    ADCONbits.GO=1;
}
```

Serial Interface

- In various instances, it is desirable to transmit binary data from one microcontroller to another. In such situations, data can be transmitted using either parallel or serial transmission techniques. In parallel transmission, each bit of the binary data is transmitted over a separate line. This means that an 8-bit data will be transmitted over eight lines. Parallel data transmission is feasible for short distances. It is not cost effective for transferring data between two systems located at hundreds of feet. Hence, serial transmission is normally used between two systems such as two computers located at long distances.

Serial Interface

- For long distances, telephone lines can be used for data transmission. A peripheral device called “modem” can be used for this purpose. Note that modem stands for modulator/ demodulator. The modem at the transmitting end transforms (modulates) binary data at the transmitting digital device, such as a microcontroller into audio signals, while another modem for the receiving end transforms (demodulates) audio signals to binary data for the receiving digital device, such as another microcontroller.

Serial Interface

- In serial transmission, only one line is used to transmit the complete binary data bit by bit. Hence, the transmitting device must convert parallel data into a string of serial bits using a parallel-in-serial-out shift register. The receiving device must convert data from serial to parallel using a serial-in-parallel-out shift register. Data are usually sent starting with the least significant bit. In order to differentiate among various bits, a clock signal is used. Serial data transmission can be divided into two types: synchronous and asynchronous.

Synchronous Serial Data Transmission

- Data are transmitted or received based on a clock signal. After deciding on a specific rate of data transmission, commonly known as “baud rate” (bits per second), the transmitting device sends a data bit at each clock pulse. In order to interpret data correctly, the receiving device must know the start and end of each data unit. Therefore, in synchronous serial data transmission, the receiver must know the number of data units to be transferred. Also, the receiver must be synchronized with data boundaries.

Synchronous Serial Data Transmission

- Usually, one or two SYNC characters (a string of bits) are used to indicate the start of each synchronous data stream. The data unit normally contains error bits such as parity. In some transmissions, the least significant bit is used as a parity bit. The synchronous usually waits in a “hunt” mode while looking for data. As soon as it matches one or more SYNC characters based on the number of SYNC characters used, the receiver starts interpreting the data.

Synchronous Serial Data Transmission

- In synchronous serial transmission, the transmitting device needs to send data continuously to the receiving device. However, if data are not ready to be transmitted, the transmitter will be padded with SYNC characters until data are available.
- In synchronous serial transfer, the receiver must know the number of SYNC characters used and the number of data units to be transferred. Once the receiver matches the SYNC characters, it receives the specified number of data units, and then goes into a “hunt” mode for matching the SYNC pattern for the next data.

Asynchronous Serial Data Transmission

- In this type of data transfer, the transmitting device does not need to be synchronized to the receiving device. The transmitting device can send one or more data units when it has data ready to be sent. Each data unit must be formatted. In other words, each data unit must contain “start” and “stop” bits, indicating the beginning and the end of each data unit.

Asynchronous Serial Data Transmission

- The interface circuits between the transmitting device and the receiving device must perform the following functions:
- 1. Convert an 8-bit parallel data unit from the transmitting device into serial data for transmitting them to the receiving device.
- 2. Convert serial data from the receiving device into parallel data for sending them back to the transmitting device assuming two-way (full duplex) transmission.

Basics of SPI and I^2C

- Serial I/O is typically fabricated as an on-chip module in microcontrollers. This will facilitate interfacing microcontrollers with other microcontrollers or peripheral devices. Several protocol (rules) standards for serial data transmission have been introduced over the years. Two such standards include SPI (Serial Peripheral Interface) developed by Motorola in the late 80's, and I^2C (Inter-Integrated Circuit) developed by Philips in 1982. Both protocols are based on synchronous serial data transmission. Also, both SPI and I^2C protocols are used for communication between ICs (Integrated Circuits) with on-board peripherals. They are intended for data transmission from a master to or from one or more slave devices over short distances.

Basics of SPI and I^2C

- The main purpose of the SPI is to replace parallel interfaces by avoiding routing of parallel buses in a PCB (Printed Circuit Board). The SPI protocol is based on the principle that a bit from an 8-bit shift register can be shifted out on a single pin and a bit can be shifted into another pin. SPI protocol can also be used for data transfer between the CPU (master) and slave devices such as flash memory and ADC.

Basics of SPI and I^2C

- The original purpose of the I^2C protocol was to easily interface a CPU to peripheral chips in a TV. Peripheral devices in an embedded system are typically connected to a microcontroller using memory-mapped I/O. This requires a lot of wiring on the PCB (Printed Circuit Board). In order to refrain from using microcontroller pins and avoid additional circuits, I^2C protocol can be used to interface the microcontroller to all the peripherals. This will also reduce wiring and thus, implementation cost.

Basics of SPI and I^2C

- SPI uses four wires; one for clock, one for data out, one for data in, and one for SS or CS (Slave Select or Chip Select to enable a slave device). I^2C, on the other hand, uses two wires; one for clock and one for data. Data transfer using SPI is faster (up to 25MHz) while data transfer using I^2C is slower (100KHz to 400KHz). In both protocols, the master generates the clock and data read or written at the rising or falling edges of the clock. However, SPI protocol is based on a single master and one or more slaves while I^2C protocol is based on multi-master with a single or multiple slaves.

PIC18F Serial I/O

- The PIC18F4321 contains an on-chip Master Synchronous Serial Port (MSSP) module which is a serial interface, useful for communicating with other peripheral or microcontroller devices. The MSSP module can operate in either SPI or I^2C mode.

PIC18F SPI mode

- The PIC18F SPI primarily uses four pins of the PIC18F4321. They are SCK (Serial Clock), SDI (Serial Data Input) , SDO (Serial Data Output) and SS (Slave Select).
- The SS pin is provided for applications requiring multiple slave devices. The SS pin may not be used for a single slave in the system. In summary, SCK, SDI, SDO, and SS pins are provided on the PIC18F to support SPI.

PIC18F SPI mode

- Figure 11.4 shows a block diagram interfacing a PIC18F master with a single PIC18F slave. Since SPI is a master/slave protocol, the master is the only device which will control the clock (SCK). Data transfers between the master and the slave are controlled by the SCK clock.

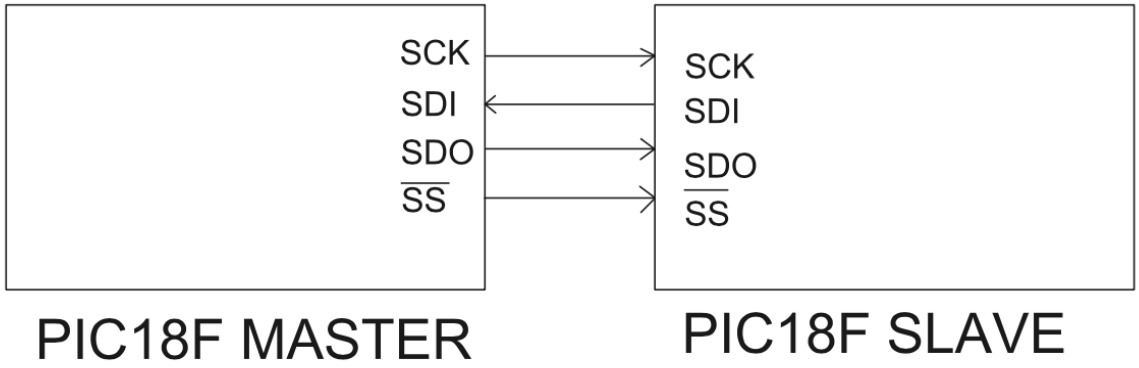


FIGURE 11. 4 PIC18F Master interface to a single slave PIC18F

PIC18F SPI mode

- SDI carries the data into the Master while SDO carries the data sent out from the Master. SS is similar to a chip select signal. SS must be used when more than one slave is present. However, the pin can be disabled via programming for a single slave.
- For simplicity, a master with a single slave will be covered in this section. Coverage of a master with multiple slaves is beyond the scope of this book.

PIC18F SPI mode

- Figure 11.5 shows a simplified diagram of SPI Master/Slave connection between two PIC18F4321's along with relevant registers. SSPSR is the shift register of the SPI module. It shifts data in and data out of the master or the slave. Data travels in a loop to the next shift register. Data is shifted out on the SDO pin of one device and into the SDI pin of the other device. Once a byte of data has been transferred between the two devices, it is copied to the SSPBUF register. The SSPBUF is then read by the user program.

PIC18F SPI mode

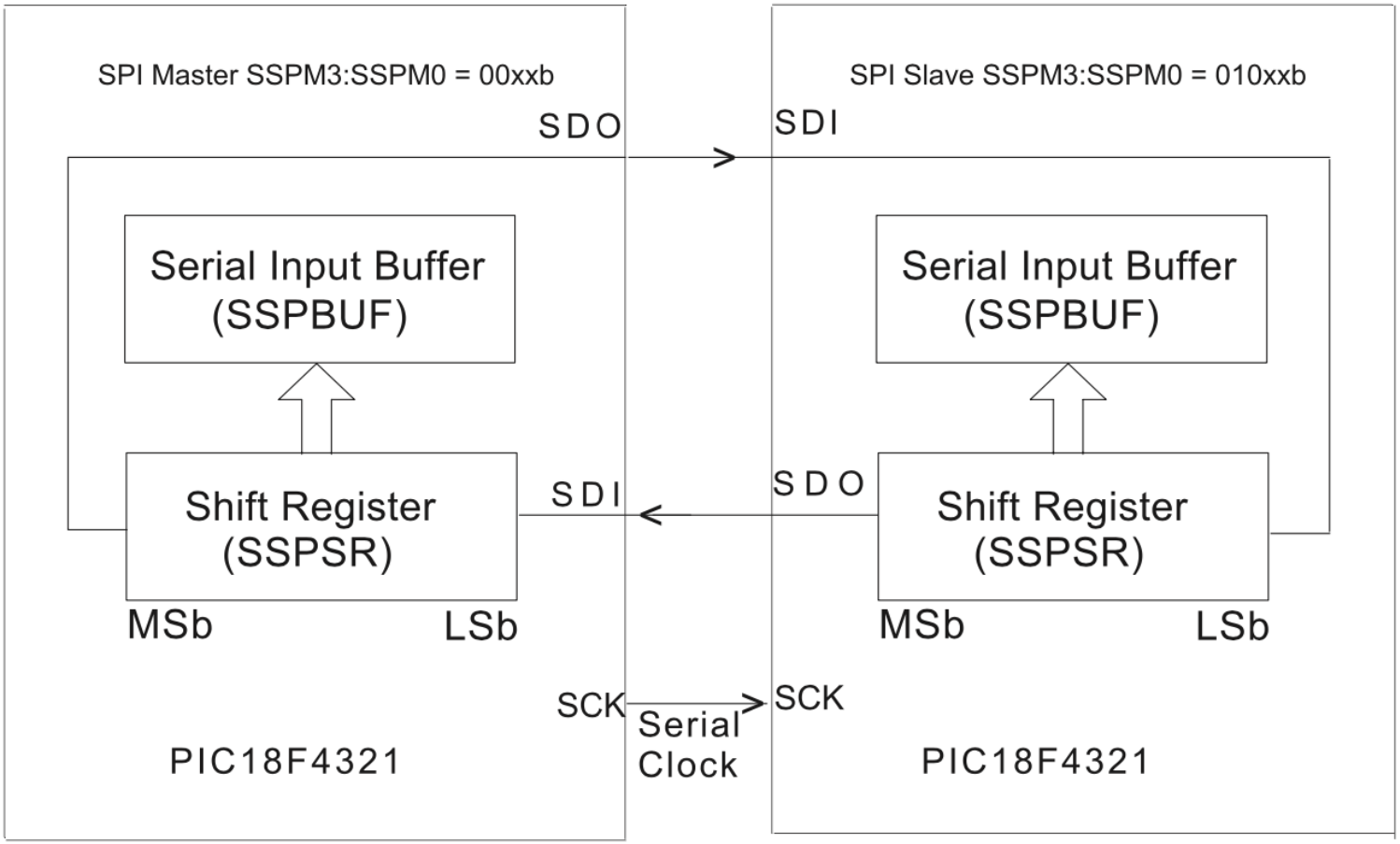


FIGURE 11.5 SPI Master/Slave interface between two PIC18F4321's along with relevant registers

PIC18F SPI mode

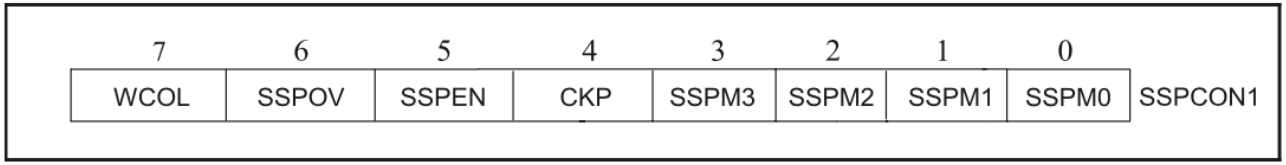
- The master initiates the data transfer while the slave waits for the clock from the master before the transfer occurs. The master determines when an SPI transfer can take place. In order to accomplish this, the master transmits SCK. The slave waits for the SCK signal and uses it when processing the SPI data.

PIC18F SPI mode

- SPI is implemented microcontroller using a hardware module called the Master Synchronous Serial Port (MSSP).
- The MSSP module uses four registers for SPI mode operation. These are:
  - MSSP Control Register 1 (SSPCON1)
  - MSSP Status Register (SSPSTAT)
  - Serial Receive/Transmit Buffer Register (SSPBUF)
  - MSSP Shift Register (SSPSR) – Not directly accessible

PIC18F SPI mode

- Figures 11.6 and 11.7 show the SSPCON1 and SSPSTAT registers respectively. The SSPCON1 and SSPSAT are control and status registers in SPI mode of operation. The SSPCON1 can be used to select the master or slave mode using the SSPM3-SSPM0 bits. The SSPEN bit of the SSPCON1 register is set to one to turn on the SSP module for using it for SPI. The SSP module must be left ON for the entire time the SSP module is in use. SSPEN can be cleared to 0 to disable or reset the SSP module.



bit 7 **WCOL:** Write Collision Detect bit (Transmit mode only)
1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

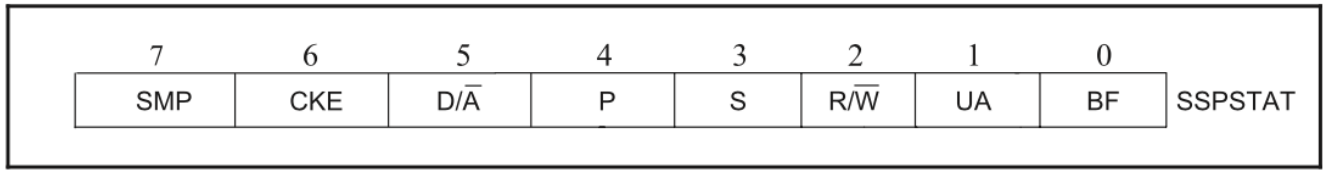
bit 6 **SSPOV:** Receive Overflow Indicator bit
SPI Slave mode:
1 = A new byte is received while the SSPBUF register is still holding the previous data. In case of overflow, the data in SSPSR is lost. Overflow can only occur in Slave mode. The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow (must be cleared in software).
0 = No overflow
Note: In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
1 = Enables serial port and configures SCK, SDO, SDI and SS as serial port pins
0 = Disables serial port and configures these pins as I/O port pins
Note: When enabled, these pins must be properly configured as input or output.

bit 4 **CKP:** Clock Polarity Select bit
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits
0101 = SPI Slave mode, clock = SCK pin, SS pin control disabled, SS can be used as I/O pin
0100 = SPI Slave mode, clock = SCK pin, SS pin control enabled
0011 = SPI Master mode, clock = TMR2 output/2
0010 = SPI Master mode, clock = FOSC/64
0001 = SPI Master mode, clock = FOSC/16
0000 = SPI Master mode, clock = FOSC/4
Note: Bit combinations not specifically listed here are either reserved or implemented in I2C™ mode only.

FIGURE 11.6 SSPCON1 (MSSP CONTROL) Register 1 in SPI mode



bit 7 **SMP:** Sample bit
SPI Master mode:
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in Slave mode.

bit 6 **CKE:** SPI Clock Select bit
1 = Transmit occurs on transition from active to Idle clock state
0 = Transmit occurs on transition from Idle to active clock state
Note: Polarity of clock state is set by the CKP bit (SSPCON1<4>).

bit 5 **D/A:** Data/Address bit
Used in I2C™ mode only.

bit 4 **P:** Stop bit
Used in I2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.

bit 3 **S:** Start bit
Used in I2C mode only.

bit 2 **R/W:** Read/Write Information bit
Used in I2C mode only.

bit 1 **UA:** Update Address bit
Used in I2C mode only.

bit 0 **BF:** Buffer Full Status bit (Receive mode only)
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

FIGURE 11.7 SSPSTAT (MSSP Status Register) in SPI mode

PIC18F SPI mode

- The SSMP3-SSMP0 (bits 3-0) can be used to select the master or slave mode. The clock polarity is selected using the CKP bit.
- Both SPI and I^2C modes use the SSPSTAT register. This register can be used to select the SPI mode (master or slave) using the SMP (data SaMPle Timing) bit, SPI clock using the CKE (ClocK Edge select) bit, and the Buffer Full (BF) bit. These three bits in the SSPSTAT control SPI.

PIC18F SPI mode

- SMP must be maintained clear for the slave. In master mode, SMP = 1 means a sample occurs at the end of data output while SMP = 0 indicates a sample occurs in the middle of data output. The SMP bit controls when data received from the SDI line is sampled, relative to when it is sent out on the SDO line.

PIC18F SPI mode

- CKP along with CKE controls one of four SPI modes used for all SPI transfers. Four CKP CKE modes are 00, 01, 10, and 11. Most common modes are 00 and 11. Note that CKP = 0 and CKE = 0 mean that data transfer occurs at the LOW to HIGH clock. On the other hand, when CKP = 1 and CKE = 1, data transfer occurs at the HIGH to LOW clock.

PIC18F SPI mode

- BF (Buffer Full) = 1 means that the SSPBUF register contains data that has not yet been read. SSBUF register contains data received via SPI. The data should be read before any more data is written or received. This is true whether the device is a master or a slave.

PIC18F SPI mode

- The SSPOV (SSP Overflow) bit is set to one if SSPBUF is not read before another byte of data is transferred. After an occurrence of SSPOV = 1, indicating an overflow, the module must be reset to clear this condition. Toggling the SSPEN bit will reset the SSP module.
- Bits 1-5 of the SSPSTAT are used in the I^2C mode.

PIC18F SPI mode

- When initializing the SPI, several options need to be specified.

  This is done by programming the appropriate control bits.

  - Master mode (SCK is the clock output)
  - Slave mode (SCK is the clock input)
  - Clock Polarity (Idle state of SCK)
  - Data Input Sample Phase (middle or end of data output time)
  - Clock Edge (output data on rising/falling edge of SCK)
  - Clock Rate (Master mode only)
  - Slave Select mode (Slave mode only)

PIC18F SPI mode

- The MSSP consists of a transmit/receive shift register (SSPSR) and a buffer register (SSPBUF). The SSPSR shifts the data in and out of the device, the most significant bit (MSB) first. The SSPBUF holds the data that was written to the SSPSR until the received data is ready. Once the 8 bits of data have been received, that byte is moved to the SSPBUF register. Then, the Buffer Full detect bit, BF (bit 0 of SSPSTAT), and an interrupt flag bit (not discussed here) are set to 1. Data transfer can be performed by polling BF or via interrupt.

PIC18F SPI mode

- User software must clear the WCOL bit so that it can be determined if the following write(s) to the SSPBUF register completed successfully. When the software is expecting to receive valid data, the SSPBUF should be read before the next byte of data to transfer is written to the SSPBUF. The Buffer Full bit, BF (bit 0 of SSPSTAT), indicates when SSPBUF has been loaded with the received data (transmission is complete).

PIC18F SPI mode

- When the SSPBUF is read, the BF bit is cleared. Generally, the MSSP interrupt is used to determine when the transmission/reception has completed. The SSPBUF must be read and/or written. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur.

PIC18F SPI mode

- **Enabling SPI I/O** To enable the serial port, MSSP Enable bit, SSPEN (bit 5 of SSPCON1) must be set to one. To reset or reconfigure SPI mode, clear the SSPEN bit to 0, reinitialize the SSPCON registers, and then set the SSPEN bit to one. This configures the SDI, SDO, and SCK pins as serial port pins. For the pins to behave as the serial port function, the data direction bits (in the TRIS register) must be appropriately programmed as follows:

PIC18F SPI mode

- SDI is automatically controlled by the SPI module.
- SDO pin is multiplexed with bit 5 of Port C. Hence, in order to configure SDO as an output pin, bit 5 of TRISC must be cleared to 0.
- SCK (Master mode) pin is multiplexed with bit 3 of Port C. Hence, in order to configure SCK as an output pin, bit 3 of TRISC must be cleared to 0.
- SCK (Slave mode) pin must be configured as an output pin. Hence, bit 3 of TRISC must be set to one.

PIC18F SPI mode

- In Figure 11.5, the master PIC18F4321 initiates the data transfer by sending the SCK signal. The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave will broadcast data by the software protocol. Data is shifted out of both shift registers on their programmed clock edge and latched on the opposite edge of the clock. Both processors should be programmed to the same Clock Polarity (CKP), then both controllers would send and receive data at the same time.

PIC18F SPI mode

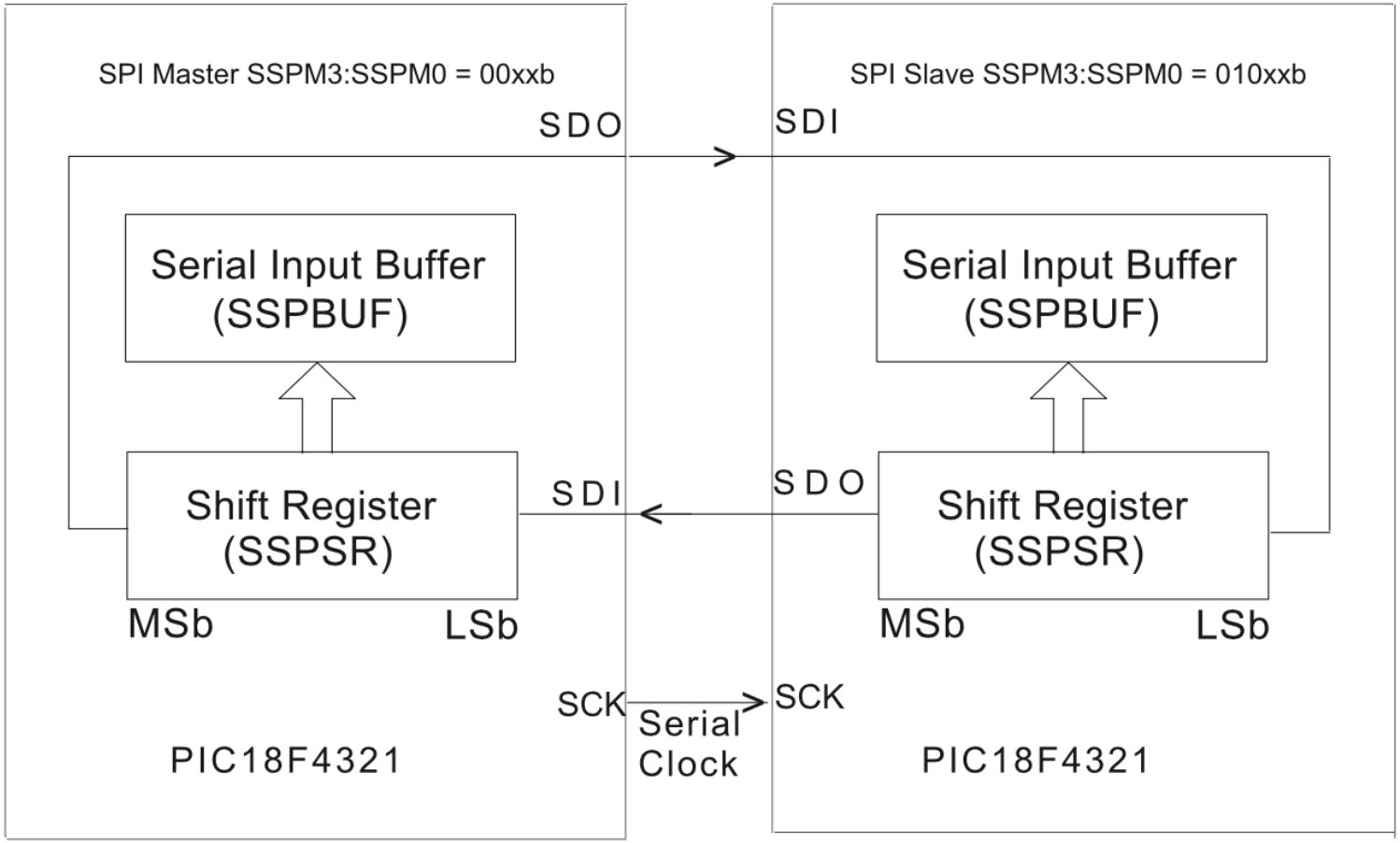


FIGURE 11.5 SPI Master/Slave interface between two PIC18F4321's along with relevant registers

PIC18F SPI mode

- Note that the SSPCON1 of the master, when initialized with 20H, will enable the SSP module for SPI mode (SSPEN = 1), select LOW idle state for the clock (CKP = 0), and select SPI master mode with the clock of FOSC/4 (SSPM3:SSPM0 = 0000).
- For the slave SSPCON1, when initialized with 25H, keep the SSP module enabled, select the same LOW idle state as the master, and select SPI slave mode with the SS pin disabled for a single slave application.

PIC18F SPI mode

- As an example, initializing master SSPSTAT with 40H will select the SPI master mode, data transmission from HIGH to LOW clock (CKE = 1), and SSPBUF empty (BF = 0). During data transmission, the BF bit in the SSPSTAT register (master or slave) can then be polled by waiting in a loop until BF = 1. This will ensure proper data transfer between the master and the slave.

PIC18F SPI mode

Example 11.6 Figure 11.8 shows a block diagram for interfacing two PIC18F4321 in SPI mode. One of the microcontrollers is the master while the other is the slave. The master PIC18F4321 will input four switches via bits 0-3 of PORTB (Switch SW0 connected to bit 0 of Port B, Switch SW1 to bit 1, SW2 to bit 2, and SW3 to bit 3), and then transmit the 4-bit data using its SDO pin to the slave's SDI pin. The slave PIC18F4321 will output this data to four LED's (LED0 connected to bit 0 of Port D, LED1 to bit 1, LED 2 to bit 2, and LED 3 to bit 3), and turn them ON or OFF based on the switch inputs. For example, LED0 will be turned ON if SW0 is one (Open) and will be turned OFF if SW0 is zero (closed), LED1 will be turned ON if SW1 is one (Open) and will be turned OFF if SW1 is zero (closed), and so on.

PIC18F SPI mode

(a) Write a PIC18F assembly language program at 0x70 for the master PIC18F4321 that will configure PORTB, initialize STKPTR to 5, initialize SSPSTAT and SSPCON1, input switches, and call a subroutine called SERIAL_WRITE to place this data into its SSPBUF register.

Also, write a PIC18F assembly language program at 0x100 for the slave PIC18F4321 that will configure PORTD, initialize SSPSTAT and SSPCON1 registers, input data from its SDI pin, place it in the slave's SSPBUF, and then output to the LED's.

(b) Repeat (a) using C language. The master PIC18F4321 will configure PORTB, initialize SSPSTAT and SSPCON1, input switches, and place this data into its SSPBUF register.

Also, write a C language program for the slave PIC18F4321 that will configure PORTD, initialize SSPSTAT and SSPCON1 registers, input data from its SDI pin, place it in the slave's SSPBUF, and then output to the LED's.

PIC18F SPI mode

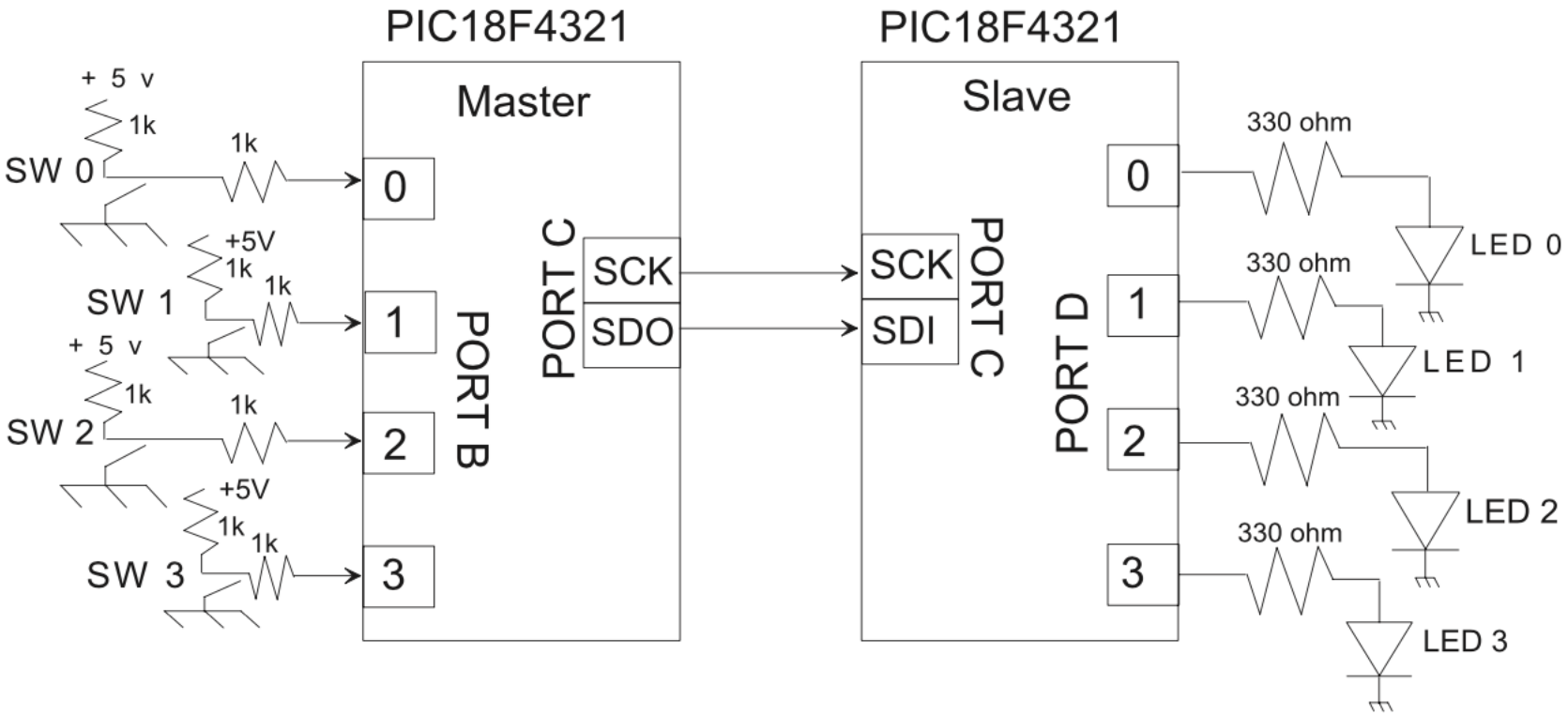


FIGURE 11.8 Figure for Example 11.6 using SPI mode

PIC18F SPI mode

- Master PIC18F4321
- 1. Configure PORTB as input with SDO and SCK as outputs.
- 2. Select CKE (SPI clock select bit) using the master’s SSPSTAT register.
- 3. Enable serial functions and select the master mode with a clock such as fosc/4 using the CCPCON1 register.
- 4. Input switches into WREG, and then CALL a subroutine called SERIAL_WRITE to move switch data into the SSPBUF register.
- 5. Wait in a loop, and check whether the BF bit in the master’s SSPSTAT register is 1, indicating completion of transmission.
- 6. As soon as BF = 1, the program returns from the subroutine and branches to Step 4.

PIC18F SPI mode

- Slave PIC18F4321
- 1. Initialize SDI and SCK pins as inputs and PORTD as an output. Note that the SCK is controlled by the master, and therefore, is configured as an input by the slave.
- 2. Select CKE - same as the master CKE (high to low clock in this example) - using the slave's SSPSTAT register.
- 3. Enable serial functions, disable the SS pin, and select slave mode using the slave's SSPCON1 register.
- 4. Wait in a loop and check whether BF = 1 in the slave's SSPSTAT register.
- 5. If BF = 0, wait. However, if BF = 1, output the contents of the slave's Serial Buffer
- register (SSPBUF) to slave's PORTD.
- 6. Go to Step 5.

PIC18F SPI mode

```
                ;Program for  the master PIC18F4321

                INCLUDE <P18F4321.INC>
                ORG         0x00            ; Reset
                GOTO        MAIN
                ORG         0x70
MAIN            BCF         TRISC, RC5      ; Configure RC5/SD0  as output
                BCF         TRISC, RC3      ; Configure RC3/SCK as output
                MOVLW       0x0F
                MOVWF       ADCON1          ; Make PORTB digital input
```

PIC18F SPI mode

```
                MOVLW   5               ; Initialize STKPTR to 5 since subroutine
                MOVWF   STKPTR          ; called SERIAL_WRITE is used in the
                                        ; program
                MOVLW   0x40
                MOVWF   SSPSTAT         ; Set data transmission on high to
                                        ; low clock
                MOVLW    0x20
                MOVWF    SSPCON1        ; Enable serial functions and set to
                                        ; master device, and  Fosc/4
GET_DATA        MOVF    PORTB,W         ; Move switch value to WREG
                CALL    SERIAL_WRITE    ; Call SERIAL_WRITE function
                BRA     GET_DATA
SERIAL_WRITE    MOVWF   SSPBUF          ; Move switch value to serial buffer
WAIT            BTFSS   SSPSTAT, BF     ; Wait until transmission is complete
                BRA     WAIT
                RETURN
                END
```

```
; Program for the slave PIC18F4321
        INCLUDE <P18F4321.INC>
        ORG     0x00            ; Reset
        GOTO    MAIN
        ORG     0x100
MAIN    BSF     TRISC, RC4      ; Configure RC4/SDI  as input
        BSF     TRISC, RC3      ; Configure RC3/SCK as input
        CLRF    TRISD           ; Configure PORTD as output
        MOVLW   0x40
        MOVWF   SSPSTAT         ; Set data transmission on high to
                                ; low clock
        MOVLW   0x25
        MOVWF   SSPCON1         ; Enable serial functions and
                                ; set to the slave
WAIT    BTFSS   SSPSTAT, BF     ; Wait until transmission is
                                ; complete (BF=1)
        BRA     WAIT            ; If BF=0, wait
        MOVFF   SSPBUF, PORTD   ; Output serial buffer data to
                                ; PORTD LEDs
        BRA     WAIT
        END
```

PIC18F SPI mode

- First, consider the master PIC18F4321. The CKE bit (bit 6) in the SSPSTAT is set to one so that data transmission will occur from an active to an idle (HIGH to LOW) clock. Next, the register SSPCON1 is configured in order to set up the parameters for serial transmission. The bit SSPEN (bit 5) in the SSPCON1 is set to HIGH in order to enable the three pins, namely SCK, SDO, and SDI. Writing 0000 to bits 3-0 of the SSPCON1 register define the master mode operation with a clock of Fosc/4.

PIC18F SPI mode

- The following PIC18F instructions accomplish this:

```
MOVLW    0x20
MOVWF    SSPCON1     ; Enable serial functions and set to master
                     ; and Fosc/4
```

- Next, consider the PIC18F assembly language program for the slave; the four bits (bits 3-0) of the slave's SSPCON1 are initialized with 0101. This will place the microcontroller in the slave mode, and also, the SS pin will be disabled since there is only one serial device in this example.

PIC18F SPI mode

- Let us now briefly explain the program logic. The assembly language program for the master, the PIC18F4321, will first perform all initializations, input the switches, and place it in the WREG. The program will then call a subroutine called SERIAL_ WRITE. The subroutine moves the switch inputs into the SSPBUF register. As soon as the serial parameters for the master such as the SCK clock pin is set up, data is automatically transmitted to the slave device. Once all the data has been written, the BF bit (bit 0) in the SSPSTAT register of the master microcontroller will go to HIGH indicating completion of transmission.

PIC18F SPI mode

- The program for the slave microcontroller waits in a loop until the BF flag in its SSPSTAT register goes to HIGH, indicating that the transmission is completed. The switch values from the slave's SSPBUF register are outputted to the LEDs connected at PORTD using the MOVFF instruction as follows:

```
MOVFF   SSPBUF, PORTD   ; Move serial buffer value to PORTD.
```

PIC18F SPI mode

- First, the master and the slave microcontrollers need to be downloaded with the above assembled programs using PICKit3. Then, upon hardware reset, the master PIC18F4321 will jump to address 0x70 (arbitrarily chosen address) while the slave PIC18F4321 will jump to address 0x100 (arbitrarily chosen address). Note that both microcontrollers do not need to be reset at the same time. Also, both the master and the slave start executing the respective programs independently.

PIC18F SPI mode

- The master microcontroller performs initializations, moves switch data input continuously, and waits in the GET_ DATA loop. The slave microcontroller also performs initializations, and then waits in the WAIT loop until BF = 1. As soon as the serial communication is established between the master and the slave, the master transmits the contents of SSPBUF via its SDO pin to the slave's SDI pin using the SCK clock. The switch data is transferred to the slave's SSPBUF register. After completion of the transfer, the slave's BF bit in the SSPSTAT register becomes 1. The slave then outputs this data to the LEDs via PORTD.

PIC18F SPI mode

- C language programs for the master PIC18F4321 and the slave PIC18F4321 can be written using the following steps as the guidelines:

```
#include <p18f4321.h>
void SPI_out(unsigned char);

void main (void)
{
unsigned char output;
TRISCbits.TRISC5 = 0;              // Configure SDO as output
TRISCbits.TRISC3 =0;               // Configure SCK as output
ADCON1=0x0F;                       // Configure PORTB to be digital input
SSPSTAT= 0x40;                     // Transmission occurs on high to low clock
SSPCON1 = 0x20;                    // Enable serial functions and set as master device
        while(1){
        output = PORTB;            // Move switch value to output
        SPI_out(output);           // Send variable 'output' to SPI_out
        }
}
void SPI_out(unsigned char SPI_data)
{
        SSPBUF = SPI_data;         // Place switch value into the serial buffer
        while (SSPSTATbits.BF == 0);     // Wait for transmission to finish
}
```

PIC18F SPI mode

// C-code for the slave PIC18F4321 device:

```
#include <p18f4321.h>
void main (void)
{
TRISCbits.TRISC4 = 1;          // Configure SDI as input
TRISCbits.TRISC3 =1;           // Configure SCK  as input
TRISD=0x00;                    // ConfigurePORTD as output for LEDs
SSPSTAT= 0x40;                 // Transmission occurs on high to low clock
SSPCON1 = 0x25;                // Enable serial functions and disable the  slave device
        while(1){
                while (SSPSTATbits.BF == 0);    // Wait for transmission to finish
                PORTD=SSPBUF;                   // Move serial buffer to PORTD
                }
        }
```

PIC18F Inter-Integrated Circuit mode

- I^2C protocol uses two pins: one for clock and one for data. They are SCL (Serial Clock pin for I2C mode multiplexed with bit 3 of Port C) and SDA (Serial Data pin for I2C mode multiplexed with bit 4 of Port C). There is no need for Slave Select (SS) like the SPI mode. Any number of masters and slaves can be connected using these two signal lines, SCL and SDA. Both signals are bidirectional. Figure 11.9 shows the block diagram of a single master interfaced with a single slave.

PIC18F Inter-Integrated Circuit mode

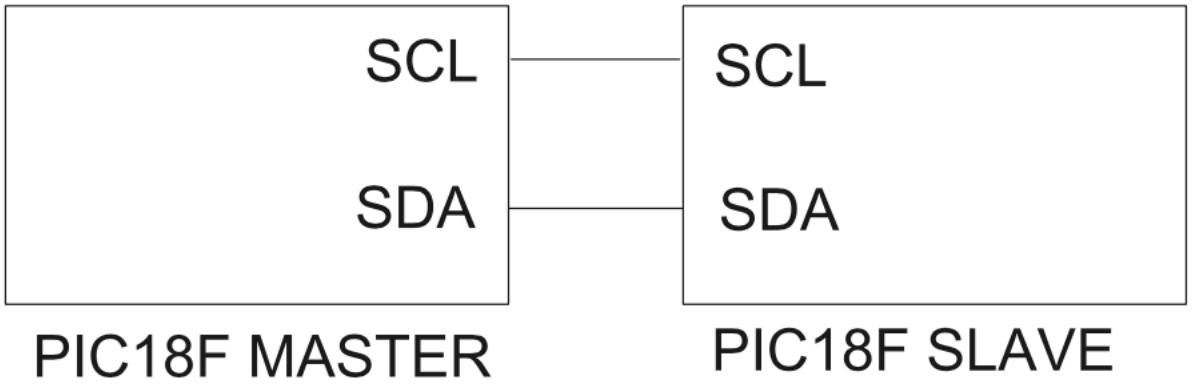


FIGURE 11.9 Master-Slave Interface in I^2C mode

PIC18F Inter-Integrated Circuit mode

- The master that initiates a data transfer on the bus is considered as the bus master with all the slave devices connected to the bus as the slaves.
- Data transfer in I^2C uses a few control bits. They typically include a START , READ/WRITE, and STOP signals. In order to transfer data using I^2C, the active master sends a START condition. All slave devices connected to the bus will receive this signal, and will wait for the incoming data.

PIC18F Inter-Integrated Circuit mode

- Next, the master will send a 7-bit address for the slave (to be selected) along with a read/write signal bit. Upon receipt of this 8-bit information, each slave will compare the master's transmitted 7-bit address with its own 7-bit address. The selected slave with the matched address will respond by sending an ACKNOWLEDGE signal. Upon receipt of this ACKNOWLEDGE signal from the selected slave, the master sends or receives data based on the read/write signal. The master will send a STOP condition and will release the bus. The slaves will then wait for the next data transfer.

PIC18F Inter-Integrated Circuit mode

- As mentioned before, the I^2C bus includes a number of conditions. These conditions typically include “START of a data transfer, “STOP” a transfer, and “ACKNOWLEDGE” a transfer. We will now briefly describe these conditions.

PIC18F Inter-Integrated Circuit mode

- A START condition indicates that a device would like to transfer data on the I^2C bus. In the START condition, SDA is pulled LOW while SCL is HIGH. The PIC18F takes care of the timing details. However, the programmer needs to tell the PIC18F when the START condition is desired, and then check for completion of the transfer.

PIC18F Inter-Integrated Circuit mode

- A STOP condition indicates that a device has completed its transfer on the I^2C bus, and would like to release the bus. Upon releasing the bus, other devices may use the bus for data transmission. The indication of a STOP condition is the release of the SCL line followed by the release of the SDA line. Note that releasing a line means that the line floats HIGH. Once the STOP condition is completed, both SCL and SDA will be HIGH. This is considered to be an “Idle state”. Once a bus is idle, a START condition can be used to transfer data.

PIC18F Inter-Integrated Circuit mode

- Data is sent on the SDA line and the clock is generated on the SCL line. Data is only considered valid when SCL is HIGH. Data is allowed to change when SCL is LOW. Eight bits of data are sent on the bus.

PIC18F Inter-Integrated Circuit mode

- Finally, consider “ACK” (Acknowledge) condition. The ACK condition is used to acknowledge completion of the transfer of a byte by driving the SDA line LOW during the 9th clock pulse of SCL. Nine bits are transferred over the bus as follows. First, eight bits are transferred using the clock, and then the device receiving data will take over the bus for one bit. If this bit is LOW, the device is sending an ACK signal.

PIC18F Inter-Integrated Circuit mode

- The PIC18F I^2C uses the same Master Synchronous Serial Port (MSSP) as SPI. Unlike the SPI, the I^2C protocol uses only two pins. They are pin 18 Serial Clock (SCL) and pin 23 Serial Data (SDA).

PIC18F Inter-Integrated Circuit mode

- Figure 11.10 shows the timing diagram for the "I^2C write transmission sequence" between the PIC18F Master and the Slave.
- The master decides when to start transferring data or when to stop the transmission. However, the slave can hold the clock when more data needs to be sent using a feature "clock stretching".

PIC18F Inter-Integrated Circuit mode

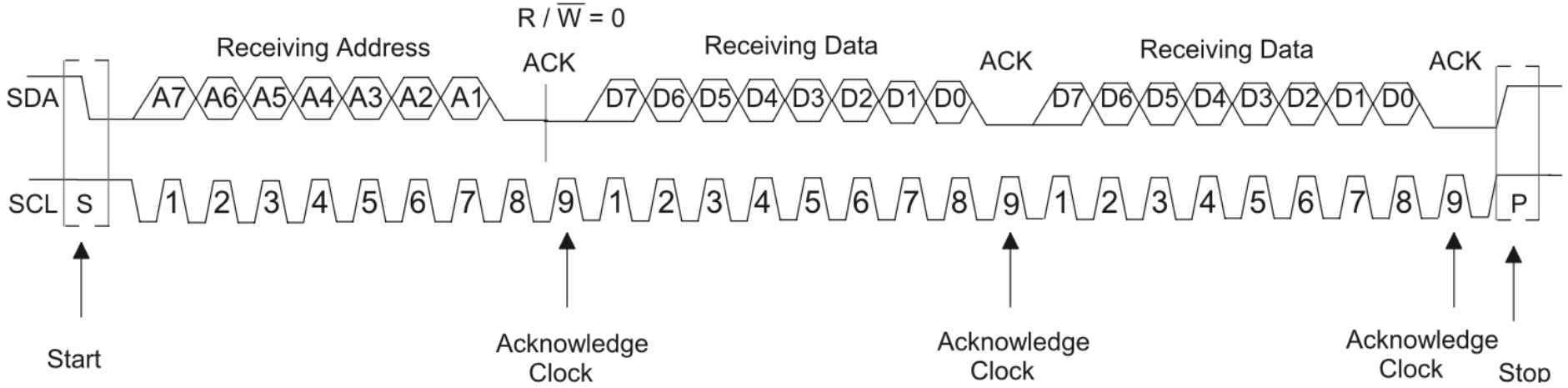


FIGURE 11.10 I²C write transmission sequence

PIC18F Inter-Integrated Circuit mode

- In the figure, the master will drive the SDA line to LOW while maintaining the SCL line HIGH in order to inform the slave of the start of the transmission process. This is called the START condition. For the STOP condition, the SDA line is toggled to HIGH while SCL is maintained HIGH.

PIC18F Inter-Integrated Circuit mode

- The I^2C interface can have more than one slave device. Therefore, in the data transfer sequence, the first 8 bit of information includes a 7-bit address of the slave along with one bit for the R/W bit. The master identifies the selected slave from this information. After receiving this 8-bit information, the slave sends an ACK signal to the master acknowledging receipt of this information.

PIC18F Inter-Integrated Circuit mode

- The master then sends the actual data to the slave. Upon receipt of this 8-bit data, the slave will send ACK signal to the master, and the process continues until the slave receives the STOP condition. The programmer can generate the START, STOP, and ACK signals by programming certain registers in the PIC18F.
- Figure 11.11 shows the MSSP block diagram for the I^2C.

PIC18F Inter-Integrated Circuit mode

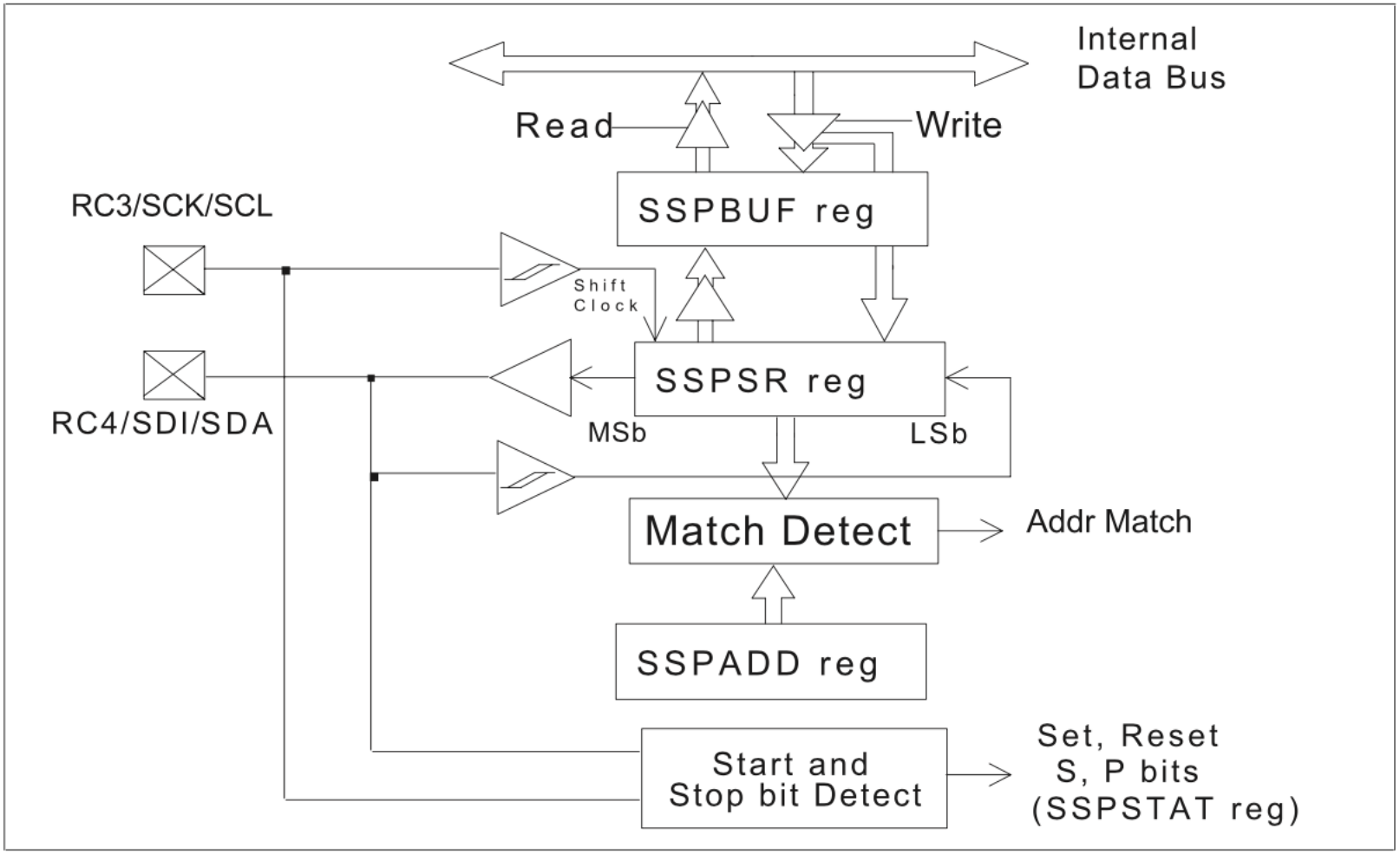


FIGURE 11.11 MSSP Block Diagram (I^2C)

PIC18F Inter-Integrated Circuit mode

- SSPSR is the shift register used for shifting data in or out. SSPBUF is the register to which data bytes are written to or read from. SSPADD register holds the slave device address when the MSSP is configured in I^2C Slave mode. When the MSSP is configured in Master mode, the lower seven bits of SSPADD act as the Baud Rate Generator reload value. When SSPSR receives a complete byte, it is transferred to SSPBUF and an interrupt is set. A write to SSPBUF will write to both SSPBUF and SSPSR.

PIC18F Inter-Integrated Circuit mode

- PIC18F MSSP Registers for I^2C mode The following registers are used for the I2C mode:
  - SSPSTAT (Figure 11.12)
  - SSPCON1 (Figure 11.13)
  - SSPCON2 (Figure 11.14)
  - SSPADD (Figure 11.15)

PIC18F Inter-Integrated Circuit mode

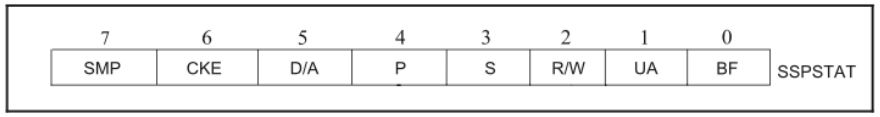
- SSPSTAT register (Figure 11.12) Upper two bits (SMP and CKE) are read/write and other bits are read-only.
- BF (Buffer Full): If this is 1, it means the SSPBUF is loaded with 8-bit. This bit will be cleared if the SSPBUF is read.
- R/W (Read/Write): This bit tells when the Master is reading from or writing to the slave device.
- S (Start): If we set this bit, the START condition will occur.
- P (Stop): If we set this bit, the STOP condition will occur.

PIC18F Inter-Integrated Circuit mode

- CKE (Clock Edge) enables (synchronizes with) or disables the SMBus. Note that the SMBus (System Management Bus) is a serial protocol based on I2C. The SMBus is a two-wire bus commonly used in computer motherboards for monitoring voltage and temperature. SMBus only goes up to 100kHz.

PIC18F Inter-Integrated Circuit mode

- SMP (Slew rate control): Enabled (SMP cleared to 0) for 400kHz and disabled (SMP set to 1) for 100kHz. Note that slew rate control is primarily an EMI (ElectroMagnetic Interference) control feature. When using slow-speed along with long-wire IC setups, enabling the slew rate control can provide additional distance of bus length with appropriate pull-ups. In our case, SMP will be set to one to disable.



bit 7 **SMP:** Slew Rate Control bit
In Master of Slave mode:
1 = Slew rate control disabled for Standard Speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for High-Spee mode (400 kHz)

bit 6 **CKE:** SMBus Select bit
In Master or Slave mode:
1 = Enable SMBus specific inputs
0 = Disabel SMBus specific inputs

bit 5 **D/A:** Data/Address bit
In Master mode:
Reserved.
In Slave mode:
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit
1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last
Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 3 **S:** Start bit
1 = Indicates that a Start bit has been detected last
0 = Start bit was not detected las
Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 2 **R/W**: Read/Write Information bit (I^2C mode only)
In Slave mode:
1 = Read
0 = Write
Note: This bit hold the R/W bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not ACK bit.
In Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress
Note: ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Active mode.

bit 1 **UA:** Update Adrress bit (10-bit Slave mode only)
1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit
In Transmit mode:
1 = SSPBUF is full
0 = SSPBUF is empty
In Receive mode:
1 = SSPBUF is full (does not include the ACK and Stop bits)
0 = SSPBUF is empty (does not include the ACK and Stop bits)

FIGURE 11.12 SSPSTAT: MSSP STATUS REGISTER (I^2C^{TM} MODE)

PIC18F Inter-Integrated Circuit mode

- SSPCON1 register (FIGURE 11.13)
- The device will be selected as Master or Slave using the SSPCON1 register. This register also enables or disables the MSSP module for I2C and releases or holds SCK for the slave.

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

bit 7 bit 0

bit 7 **WCOL:** Write Collision Detect bit
In Master Trransmit mode:
1 = A write tothe SSPBUF register was attemted while the I²C conditions were not valid for trasmission to be started (must be cleared in software
0 = No collision
In Slave Transmit mode:
1 = The SSPBUF register is written while it is still trasmittting the previous word (must be cleared in software)
0 = No collision
In Slave Transmit mode:
1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision
In Receive mode (Master or Slave modes):
This is a "don't care" bit.

bit 6 **SSPOV:** Recieive Overflow Indicator bit
In Receive mode:
1 = A byte is reeived while the SSPBUF register is still holding the previous byte (must be cleared in software)
0 = No overflow
In Transmit mode:
This a "don't care" bit in Transmit mode.

bit 5 **SSPEN:** Master Synchronous Serial Port Enable bit
1 = Enables the serial port and configures th SDA and SCL pins as the serial port pins
0 = Disables serial port and configures these pins as I/O port pins
Note: When enabled, the SDA and SCL pins must be properly configured as inputs.

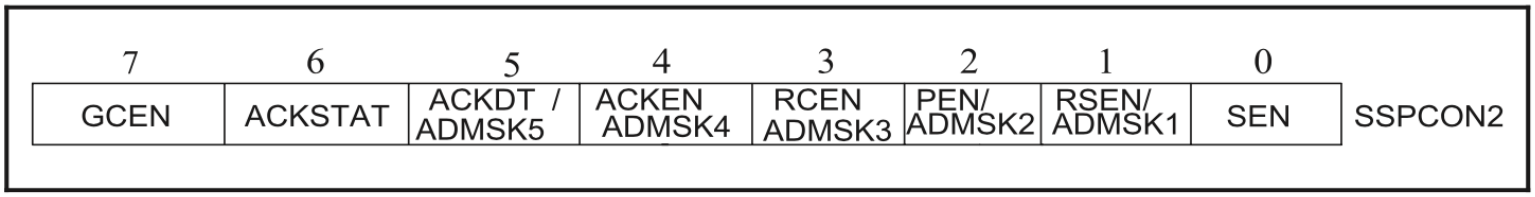
bit 4 **CKP:** SCK Release Control bit
In Slave mode:
1 = Release clock
0 = Holds clock low (clock stretch), used to ensure data setup time
In Master mode:
Unused in this mode.

bit 3-0 **SSPM3:SSPM0:** Master Synchronous Serial Port Mode Select bits
1111 = I²C Slave mode, 10-bit address with Start and Stop bit interrups enabled
1110 = I²C Slave mode, 7-bit address with Start and Stop bit interrups enabled
1011 = I²C Firmware Controlled Master mode (slve Idle)
1000 = I²C Master mode, clock = Fosc/(4 * (SSPADD + 1))
0111 = I²C Slave mode, 10-bit address
0110 =I²C Slave mode, 7-bit address

FIGURE 11.13 SSPCON1: MSSP CONTROL REGISTER 1 (I2C™ MODE)

PIC18F Inter-Integrated Circuit mode

- SSPCON2 register (FIGURE 11.14) provides acknowledge status bit for the master (received or not received) for the master initiates START or STOP conditions for the master provides clock stretching.
- If SEN is enabled (SSPCON2<0> = 1), RC3/SCK/SCL will be held low (clock stretch) following each data transfer. The clock must be released by setting bit, CKP (SSPCON1<4>).

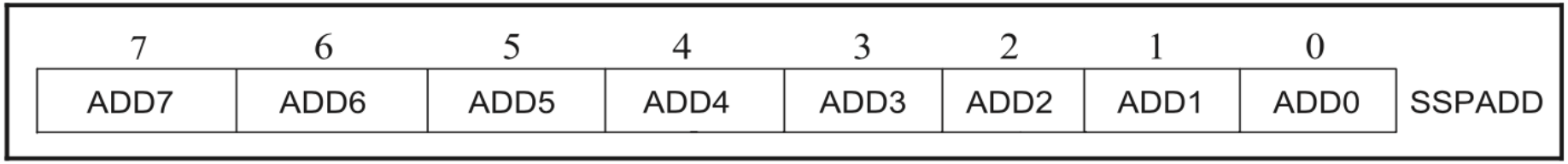


bit 7 **GCEN:** General Call Enable bit (Slave mode on)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled
bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)
1 = Acknowledge was not received from slave 0 = Acknowledge was received from slave
bit 5 **ACKDT/ADMSK5:** Acknowledge Data bit
In Master Receive mode: 1 = Not Acknowledge 0 = Acknowledge
Note: Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive. In Slave mode: 1 = Address masking of ADD5 enabled 0 = Address masking of ADD5 disabled
bit 4 **ACKEN/ADMSK4:** Acknowledge Sequence Enable bit
In Master Receive mode:(1) 1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.0 = Acknowledge sequence Idle In Slave mode:
1 = Address masking of ADD4 enabled 0 = Address masking of ADD4 disabled
bit 3 **RCEN/ADMSK3:** Receive Enable bit
In Master Receive mode:(1) 1 = Enables Receive mode for I^2C 0 = Receive Idle In Slave mode:
1 = Address masking of ADD3 enabled 0 = Address masking of ADD3 disabled
bit 2 **PEN/ADMSK2:** Stop Condition Enable bit
In Master mode:(1)
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware. 0 = Stop condition Idle
In Slave mode:1 = Address masking of ADD2 enabled 0 = Address masking of ADD2 disabled
bit 1 **RSEN/ADMSK1:** Repeated Start Condition Enable bit
In Master mode:1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition In Slave mode (7-bit Address mode): 1 = Address masking of ADD1 enabled 0 = Address masking of ADD1 disabled
In Slave mode (10-bit Address mode): 1 = Address masking of ADD1 and ADD0 enabled
0 = Address masking of ADD1 and ADD0 disabled
bit 0 **SEN:** Start Condition Enable/Stretch Enable bit (1)
In Master mode:
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware. 0 = Start condition Idle
In Slave mode:1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is disabled Note that I2C devices can slow down communication by clock stretching (that is, by stretching SCL). During an SCL low phase, any I2C device on the bus may additionally hold down SCL clock to prevent it to rise high again, enabling them to slow down the SCL clock rate or to stop I2C

FIGURE 11.14 SSPCON2: MSSP CONTROL REGISTER 2 (I2C™ MODE)

PIC18F Inter-Integrated Circuit mode

- SSPADD register (FIGURE 11.15): For the slave device, SSPADD = slave address. For the master device, the 8-bit value of SSPADD will determine the clock frequency in the SCK line. This is called Baud rate. The baud rate formula is:
  - Baud rate = FOSC/(4*(SSPADD +1))
  - SSPADD = (FOSC/ (4*Baud rate))-1



7	6	5	4	3	2	1	0	
ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0	SSPADD

bit 7-0 **ADD<7:0>:** MSSP Address bits

Note 1: MSSP Address register in I^2C Slave mode. MSSP Baud Rate register in I^2C Master mode.

FIGURE 11.15 SSPADD: MSSP ADDRESS REGISTER

PIC18F Inter-Integrated Circuit mode

Example 11.7 Figure 11.16 shows a block diagram for interfacing two PIC18F4321 in I²C mode. One of the microcontrollers is the master while the other is the slave. The master PIC18F4321 will input four switches via bits 0-3 of PORTB (Switch SW0 connected to bit 0 of Port B, Switch SW1 to bit 1, SW2 to bit 2 and SW3 to bit 3), and then transmit the 4-bit data using its SDA pin to the slave's SDA pin. The slave PIC18F4321 will output this data to four LED's (LED0 connected to bit 0 of Port D, LED1 to bit 1, LED 2 to bit 2, and LED 3 to bit 3), and turn them ON or OFF based on the switch inputs. For example, LED0 will be turned ON if SW0 is one (Open) and will be turned OFF if SW0 is zero (closed), LED1 will be turned ON if SW1 is one (Open) and will be turned OFF if SW1 is zero (closed), and so on.

PIC18F Inter-Integrated Circuit mode

(a) Write a PIC18F assembly language program at 0x70 for the master PIC18F4321 that will configure PORTB, initialize STKPTR to 5, initialize SSPSTAT and SSPCON1, input switches, and call a subroutine called SERIAL_WRITE to place this data into its SSPBUF register.
Also, write a PIC18F assembly language program at 0x100 for the slave PIC18F4321 that will configure PORTD, initialize SSPSTAT and SSPCON1 registers, input data from its SDA pin, place it in the slave's SSPBUF, and then output to the LED's.

(b) Repeat (a) using C language. The master PIC18F4321 will configure PORTB, initialize SSPSTAT and SSPCON1, input switches, and place this data into its SSPBUF register.
Also, write a C language program for the slave PIC18F4321 that will configure PORTD, initialize SSPSTAT and SSPCON1 registers, input data from its SDA pin, place it in the slave's SSPBUF, and then output to the LED's.

PIC18F Inter-Integrated Circuit mode

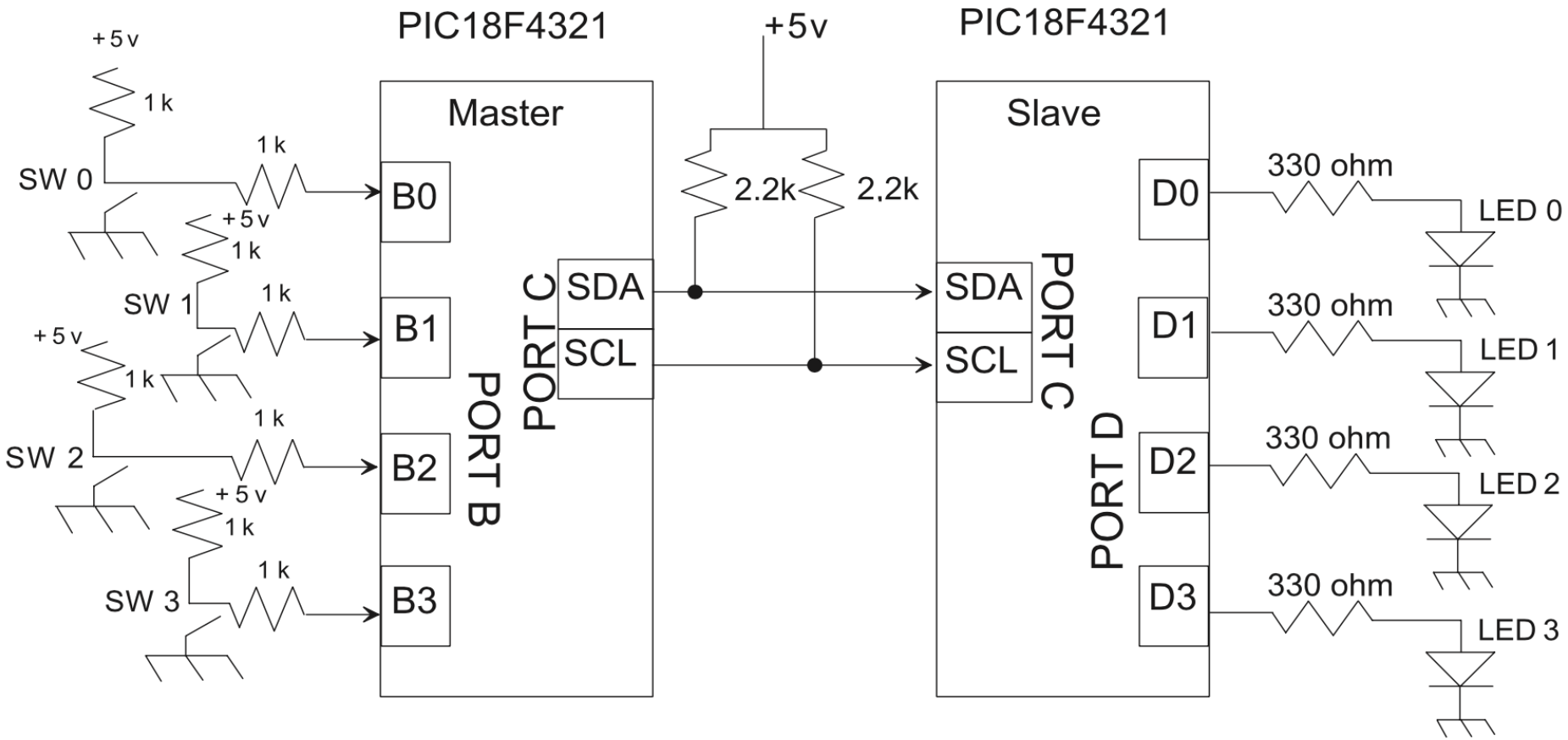


FIGURE 11.16 Figure for Example 11.7

PIC18F Inter-Integrated Circuit mode

- Master:
- 1. Configure PORTB as input for switches. Also, configure Master SDA and SCL as inputs. See “Note” (bit 5, SSPCON1, Figure 11.13)
- 2. Initialize SSPSTAT = 0x80, SSPCON1 = 0x38, SSPADD = 0x01, SSPCON2 = 0x00

PIC18F Inter-Integrated Circuit mode

- SSPSTAT (Figure 11.12) = 0x80 means SMP = 1 (Slew rate control OFF), CKE = 0 (Disable SMBus). Since other bits are read-only, they are don’t cares initially and assumed 0’s.

PIC18F Inter-Integrated Circuit mode

- SSPCON1 (Figure 11.13) = 0x38 means WCOL = 0 (No collision), SSPOV = 0 (Don’t care mode in transmit mode; assumed 0), SSPEN = 1 (Enables serial port and configures SDA and SCL), CKP = 1 (Unused in master mode; assumed 1 arbitrarily), SSPM3:SSPM0 = 1000 (I2C master mode).
- Since SSPEN = 1 in this example, SDA and SCL pins are configured as inputs. Hence, configure SDA (I2C data pin multiplexed with RC4) and SCL (I2C clock multiplexed with RC3) as inputs using PORTC using TRISC register.

PIC18F Inter-Integrated Circuit mode

- SSPADD (Figure 11.15) = 0x01. SSPADD is the baud rate register and its contents are computed from SSPADD = (FOSC/(4*Baud rate))-1. With FOSC = 1MHz and Baud rate = 100 kHz, SSPADD is 1.5. Hence, ‘1’ is loaded into SSPADD.
- SSPCON2 (Figure 11.14) = 0x00. See details in the figure for SSPCON2.

PIC18F Inter-Integrated Circuit mode

- 3. Write functions that will tell the Master to wait for idle, trigger start condition, send stop condition, and write information to the slave.
- 4. In the main, set up ADCON1, TRISB, and run the function in a loop with sequence: start, write address, write data, stop.

PIC18F Inter-Integrated Circuit mode

- Slave
- 1. Configure PORTD as an output for LEDs. Configure SDA and SCL as inputs in PORTC.
- 2. Configure SSPSTAT, SSPCON1, SSPADD. In this case, SSPSTAT = 0x80, SSPCON1 = 0x36, and SSPADD = slave address.

PIC18F Inter-Integrated Circuit mode

- SSPSTAT = 0x80 means SMP = 1 (Slew rate control OFF), CKE = 0 (Disable SMBus). The other bits are read-only.
- SSPCON1 = 0x36 means WCOL = 0 (No collision), SSPOV = 0 (Don’t care mode in transmit mode; assumed 0), SSPEN = 1 (Enables serial port and configures SDA and SCL), CKP = 1 (release clock), SSPM3:SSPM0 = 0110 (I2C Slave mode).
- SSPADD contains slave address.
- SSPCON2 = 0x01 to enable clock stretching.

PIC18F Inter-Integrated Circuit mode

- 3. Check for information sent from the master. To know whether the master is sending address or data, the slave will check at bits S, D/A, R/ W and BF in the SSPSTAT register. Note that S=1, R/W = 0, and BF = 1 will indicate that the master is sending information and the SSPBUF contains address or data that needs to be read or cleared. Next, the D/ A bit can be checked: D/A = 0: The master is sending address; ignore by doing the dummy read to clear the SSPBUF. D/A = 1: The master is sending data; output SSPBUF data to PORTD
- 4. In the main, make infinite loop to keep reading data from Master.

```
; PIC18F assembly language program for the Master

INCLUDE <P18F4321.INC>

SLAVE_ADDR      EQU   0x30              ; 7-bit Slave address (arbitrarily chosen)
                                        ; + Write bit
                ORG 0x00
                GOTO MAIN
                ORG 0x70
MAIN            BSF TRISC, RC3          ; Configure RC3/SCL as input
                BSF TRISC, RC4          ; Configure RC4/SDA as input
                MOVLW  5
                MOVWF  STKPTR           ; Initialize STKPTR since subroutines used
                MOVLW 0x0F
                MOVWF ADCON1            ; ADCON1 = 0x0F, make PORTB as an input
                MOVLW 0x38
                MOVWF SSPCON1           ; I2C master mode
                CLRF SSPCON2            ; Reset/clear SSPCON2
                MOVLW 1
                MOVWF SSPADD            ; Baudrate 100kHz with 1MHz clock
                MOVLW 0x80
                MOVWF SSPSTAT           ; Slew rate control disable
START           CALL IDLE_WAIT          ; Wait for idle
                BSF SSPCON2, SEN        ; Send start condition
WRITE_ADDR      CALL IDLE_WAIT          ; Wait for idle
                MOVLW SLAVE_ADDR        ; Load slave address to WREG
                CALL MASTER_WRITE       ; Send Address to slave
WRITE_DATA      CALL IDLE_WAIT          ; Wait for idle
                MOVF PORTB, W           ; Move switch value to WREG
                CALL MASTER_WRITE
STOP            CALL IDLE_WAIT          ; Wait for idle
                BSF SSPCON2, PEN        ; Send stop condition
                BRA START               ; LOOPING
MASTER_WRITE    MOVWF SSPBUF            ; Move switch value to Serial buffer
                RETURN
IDLE_WAIT       BTFSC  SSPSTAT, R_W     ; Check if the slave is reading
                BRA      IDLE_WAIT
                RETURN
                END
```

```
; PIC18F assembly language for the Slave
<INCLUDE <P18F4321.INC>

SLAVE_ADDR      EQU   0x30
DUMMY           EQU   0x60              ; dummy place to clear SSPBUF
                ORG   0x00
                GOTO  MAIN

                ORG 0x70
MAIN            BSF TRISC, RC3          ; Configure RC3/SCL as input
                BSF TRISC, RC4          ; Configure RC4/SDA as input
                MOVLW 0xF0
                MOVWF TRISD             ; PORT D outputs
                MOVLW SLAVE_ADDR        ; Slave address to WREG
                MOVWF SSPADD            ; SSPSTAT = slave address
                MOVLW 0x36
                MOVWF SSPCON1           ; I2C slave mode
                MOVLW 0x80
                MOVWF SSPSTAT           ; Slew rate control disable
                MOVLW 0x01
                MOVWF SSPCON2           ; Enable clock stretching
CHECK           BTFSS SSPSTAT, S        ; Check if S = 1
                BRA CHECK
                BTFSC SSPSTAT, R_W      ; Check if R/nW = 0
                BRA CHECK
                BTFSS SSPSTAT, BF       ; Check if BF = 1
                BRA CHECK
                BTFSS SSPSTAT, D_A      ; Check if D/nA = 1
                BRA DUMMY_READ
                MOVFF SSPBUF, PORTD     ; PORTD = SSPBUF
                BSF SSPCON1, CKP        ; SSPCON1.CKP = 1, release clock
                BRA CHECK               ; Looping
DUMMY_READ      MOVFF SSPBUF, DUMMY     ; DUMMY = SSPBUF, clear SSPBUF
                BSF SSPCON1, CKP        ; Release clock
                BRA CHECK               ; Looping
                END
```

PIC18F Inter-Integrated Circuit mode

```
//C-program for the Master

#include <P18F4321.h>
#include <stdio.h>
#define SLAVE_ADDRESS 0x30
void I2C_Master_Init(unsigned long bRate)
{
        TRISCbits.RC3 = 1;              // SCL as input
        TRISCbits.RC4 = 1;              // SDA as input
        SSPCON1 = 0x38;                 // I2C master mode
        SSPCON2 = 0x00;
        SSPADD = (1000000/(4*bRate))-1;   // I2C Baud Rate formula with 1MHz OSC
        SSPSTAT = 0x80;                 // slew rate control disable for 100khz i2c BaudRate
}
void I2C_Master_Wait()
{
        while ((SSPSTATbits.R_W)); // wait for idle, not writing
}
void I2C_Master_Start()
```

PIC18F Inter-Integrated Circuit mode

```
{
        I2C_Master_Wait();
        SSPCON2bits.SEN = 1;
}
void I2C_Master_Stop()
{
        I2C_Master_Wait();
        SSPCON2bits.PEN = 1;
}
void I2C_Master_Write(unsigned char writeData)
{
        I2C_Master_Wait();
        SSPBUF = writeData;
}
void main()
{
        ADCON1 = 0x0F;              // PORTB as input
        I2C_Master_Init(100000);    // 100kHz baud rate
        while(1)
        {
        I2C_Master_Start(); //Start condition
        I2C_Master_Write(SLAVE_ADDRESS);    // 7 bit address + Write
        I2C_Master_Write(PORTB);  // Write data
        I2C_Master_Stop();          // Stop condition
        }
```