



Welcome to [E-XFL.COM](https://www.e-xfl.com)

What is "[Embedded - Microcontrollers](#)"?

"[Embedded - Microcontrollers](#)" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "[Embedded - Microcontrollers](#)"

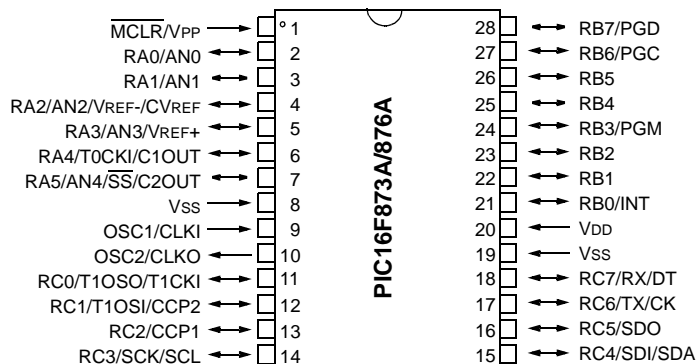
Details

Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	10MHz
Connectivity	I ² C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	22
Program Memory Size	14KB (8K x 14)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	368 x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 5.5V
Data Converters	A/D 5x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic16lf876a-i-so

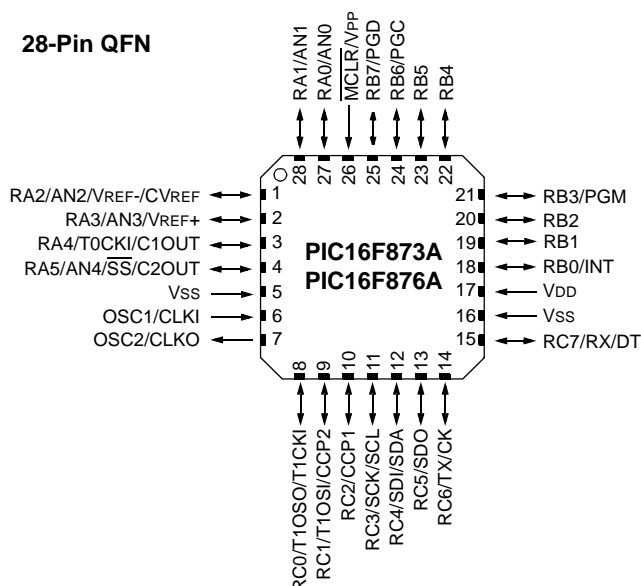
PIC16F87XA

Pin Diagrams

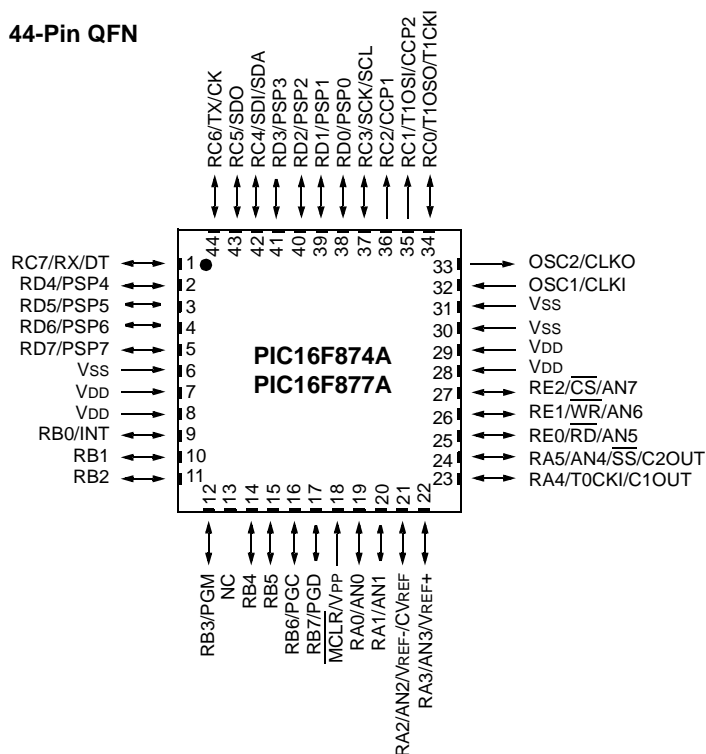
28-Pin PDIP, SOIC, SSOP



28-Pin QFN



44-Pin QFN



PIC16F87XA

2.2 Data Memory Organization

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (Status<6>) and RP0 (Status<5>) are the bank select bits.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some frequently used Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

Note: The EEPROM data memory description can be found in Section 3.0 “Data EEPROM and Flash Program Memory” of this data sheet.

2.2.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly, or indirectly, through the File Select Register (FSR).

3.3 Reading Data EEPROM Memory

To read a data memory location, the user must write the address to the EEADR register, clear the EEPGD control bit (EECON1<7>) and then set control bit RD (EECON1<0>). The data is available in the very next cycle in the EEDATA register; therefore, it can be read in the next instruction (see Example 3-1). EEDATA will hold this value until another read or until it is written to by the user (during a write operation).

The steps to reading the EEPROM data memory are:

1. Write the address to EEADR. Make sure that the address is not larger than the memory size of the device.
2. Clear the EEPGD bit to point to EEPROM data memory.
3. Set the RD bit to start the read operation.
4. Read the data from the EEDATA register.

EXAMPLE 3-1: DATA EEPROM READ

```
BSF    STATUS,RP1    ;
BCF    STATUS,RP0    ; Bank 2
MOVF   DATA_EE_ADDR,W ; Data Memory
MOVWF  EEADR         ; Address to read
BSF    STATUS,RP0    ; Bank 3
BCF    EECON1,EEPGD  ; Point to Data
                     ; memory
BSF    EECON1,RD      ; EE Read
BCF    STATUS,RP0    ; Bank 2
MOVF   EEDATA,W      ; W = EEDATA
```

3.4 Writing to Data EEPROM Memory

To write an EEPROM data location, the user must first write the address to the EEADR register and the data to the EEDATA register. Then the user must follow a specific write sequence to initiate the write for each byte.

The write will not initiate if the write sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. We strongly recommend that interrupts be disabled during this code segment (see Example 3-2).

Additionally, the WREN bit in EECON1 must be set to enable write. This mechanism prevents accidental writes to data EEPROM due to errant (unexpected) code execution (i.e., lost programs). The user should keep the WREN bit clear at all times, except when updating EEPROM. The WREN bit is not cleared by hardware

After a write sequence has been initiated, clearing the WREN bit will not affect this write cycle. The WR bit will be inhibited from being set unless the WREN bit is set. At the completion of the write cycle, the WR bit is cleared in hardware and the EE Write Complete Interrupt Flag bit (EEIF) is set. The user can either enable this interrupt or poll this bit. EEIF must be cleared by software.

The steps to write to EEPROM data memory are:

1. If step 10 is not implemented, check the WR bit to see if a write is in progress.
2. Write the address to EEADR. Make sure that the address is not larger than the memory size of the device.
3. Write the 8-bit data value to be programmed in the EEDATA register.
4. Clear the EEPGD bit to point to EEPROM data memory.
5. Set the WREN bit to enable program operations.
6. Disable interrupts (if enabled).
7. Execute the special five instruction sequence:
 - Write 55h to EECON2 in two steps (first to W, then to EECON2)
 - Write AAh to EECON2 in two steps (first to W, then to EECON2)
 - Set the WR bit
8. Enable interrupts (if using interrupts).
9. Clear the WREN bit to disable program operations.
10. At the completion of the write cycle, the WR bit is cleared and the EEIF interrupt flag bit is set. (EEIF must be cleared by firmware.) If step 1 is not implemented, then firmware should check for EEIF to be set, or WR to clear, to indicate the end of the program cycle.

EXAMPLE 3-2: DATA EEPROM WRITE

```
BSF    STATUS,RP1    ;
BSF    STATUS,RP0    ;
BTFSC  EECON1,WR      ;Wait for write
GOTO   $-1           ;to complete
BCF    STATUS, RP0    ;Bank 2
MOVF   DATA_EE_ADDR,W ;Data Memory
MOVWF  EEADR         ;Address to write
MOVF   DATA_EE_DATA,W ;Data Memory Value
MOVWF  EEDATA        ;to write
BSF    STATUS,RP0    ;Bank 3
BCF    EECON1,EEPGD  ;Point to DATA
                     ;memory
BSF    EECON1,WREN    ;Enable writes

BSF    INTCON,GIE     ;Disable INTs.
MOVLW  55h           ;
MOVWF  EECON2         ;Write 55h
MOVLW  AAh           ;
MOVWF  EECON2         ;Write AAh
BSF    EECON1,WR      ;Set WR bit to
                     ;begin write
BSF    INTCON,GIE     ;Enable INTs.
BCF    EECON1,WREN    ;Disable writes
```

PIC16F87XA

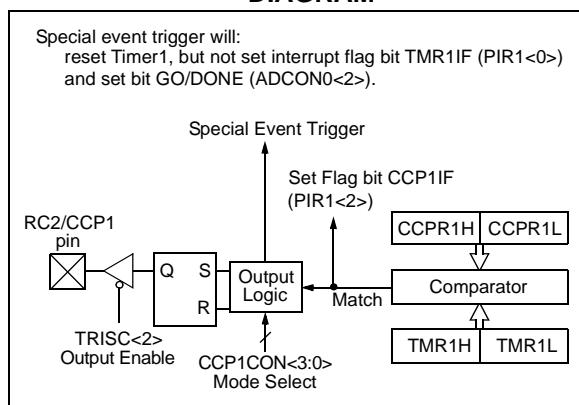
8.2 Compare Mode

In Compare mode, the 16-bit CCPR1 register value is constantly compared against the TMR1 register pair value. When a match occurs, the RC2/CCP1 pin is:

- Driven high
- Driven low
- Remains unchanged

The action on the pin is based on the value of control bits, CCP1M3:CCP1M0 (CCP1CON<3:0>). At the same time, interrupt flag bit CCP1IF is set.

FIGURE 8-2: COMPARE MODE OPERATION BLOCK DIAGRAM



8.2.1 CCP PIN CONFIGURATION

The user must configure the RC2/CCP1 pin as an output by clearing the TRISC<2> bit.

Note: Clearing the CCP1CON register will force the RC2/CCP1 compare output latch to the default low level. This is not the PORTC I/O data latch.

8.2.2 TIMER1 MODE SELECTION

Timer1 must be running in Timer mode, or Synchronized Counter mode, if the CCP module is using the compare feature. In Asynchronous Counter mode, the compare operation may not work.

8.2.3 SOFTWARE INTERRUPT MODE

When Generate Software Interrupt mode is chosen, the CCP1 pin is not affected. The CCPIF bit is set, causing a CCP interrupt (if enabled).

8.2.4 SPECIAL EVENT TRIGGER

In this mode, an internal hardware trigger is generated which may be used to initiate an action.

The special event trigger output of CCP1 resets the TMR1 register pair. This allows the CCPR1 register to effectively be a 16-bit programmable period register for Timer1.

The special event trigger output of CCP2 resets the TMR1 register pair and starts an A/D conversion (if the A/D module is enabled).

Note: The special event trigger from the CCP1 and CCP2 modules will not set interrupt flag bit TMR1IF (PIR1<0>).

9.3.3 ENABLING SPI I/O

To enable the serial port, SSP Enable bit, SSPEN (SSPCON<5>), must be set. To reset or reconfigure SPI mode, clear the SSPEN bit, re-initialize the SSPCON registers and then set the SSPEN bit. This configures the SDI, SDO, SCK and \overline{SS} pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed. That is:

- SDI is automatically controlled by the SPI module
- SDO must have TRISC<5> bit cleared
- SCK (Master mode) must have TRISC<3> bit cleared
- SCK (Slave mode) must have TRISC<3> bit set
- \overline{SS} must have TRISC<4> bit set

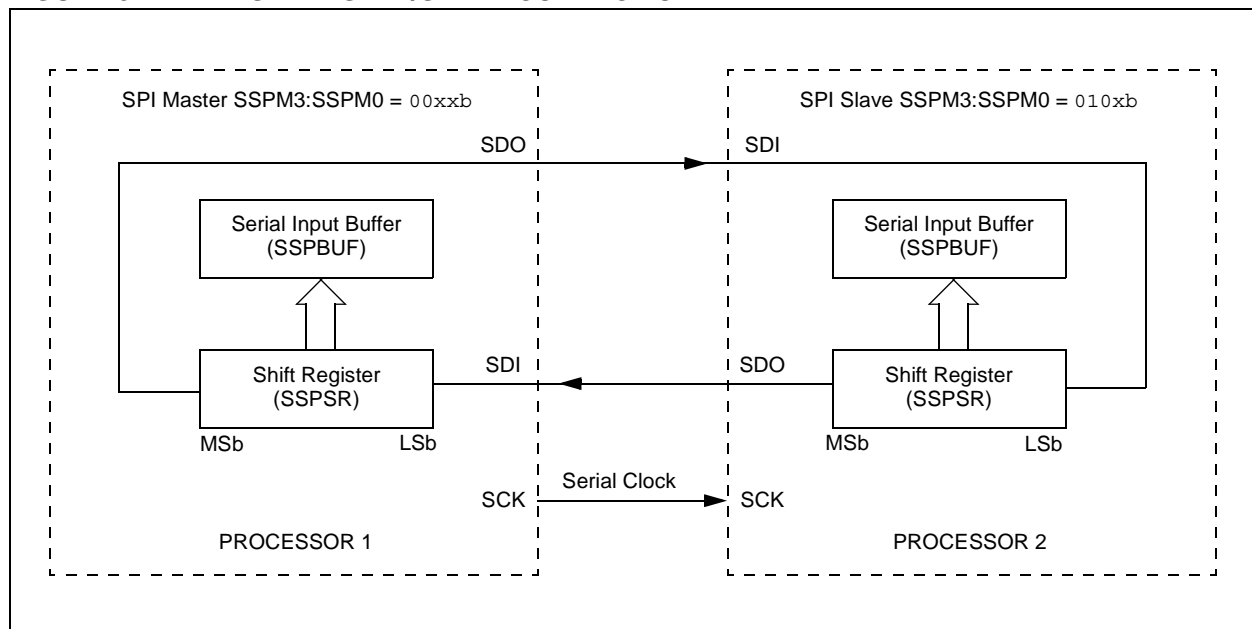
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

9.3.4 TYPICAL CONNECTION

Figure 9-2 shows a typical connection between two microcontrollers. The master controller (Processor 1) initiates the data transfer by sending the SCK signal. 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. Whether the data is meaningful (or dummy data) depends on the application software. This leads to three scenarios for data transmission:

- Master sends data – Slave sends dummy data
- Master sends data – Slave sends data
- Master sends dummy data – Slave sends data

FIGURE 9-2: SPI MASTER/SLAVE CONNECTION



PIC16F87XA

FIGURE 9-5: SPI MODE WAVEFORM (SLAVE MODE WITH CKE = 0)

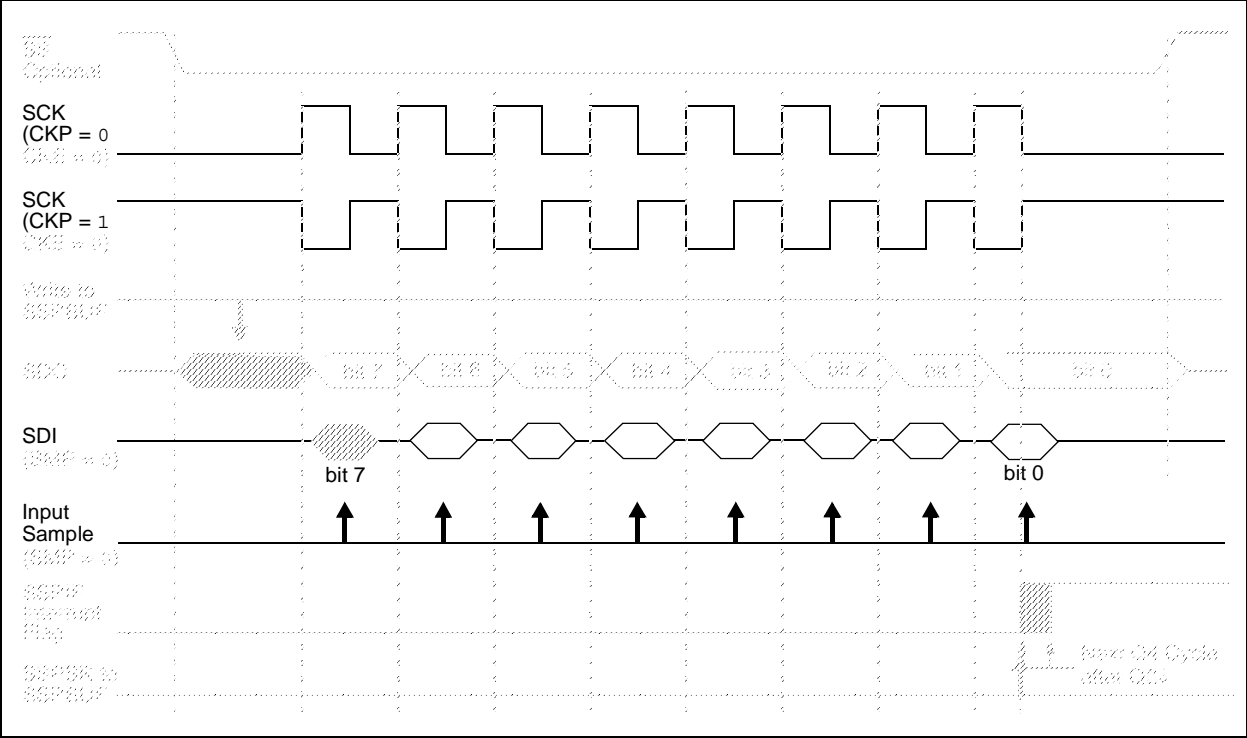


FIGURE 9-6: SPI MODE WAVEFORM (SLAVE MODE WITH CKE = 1)

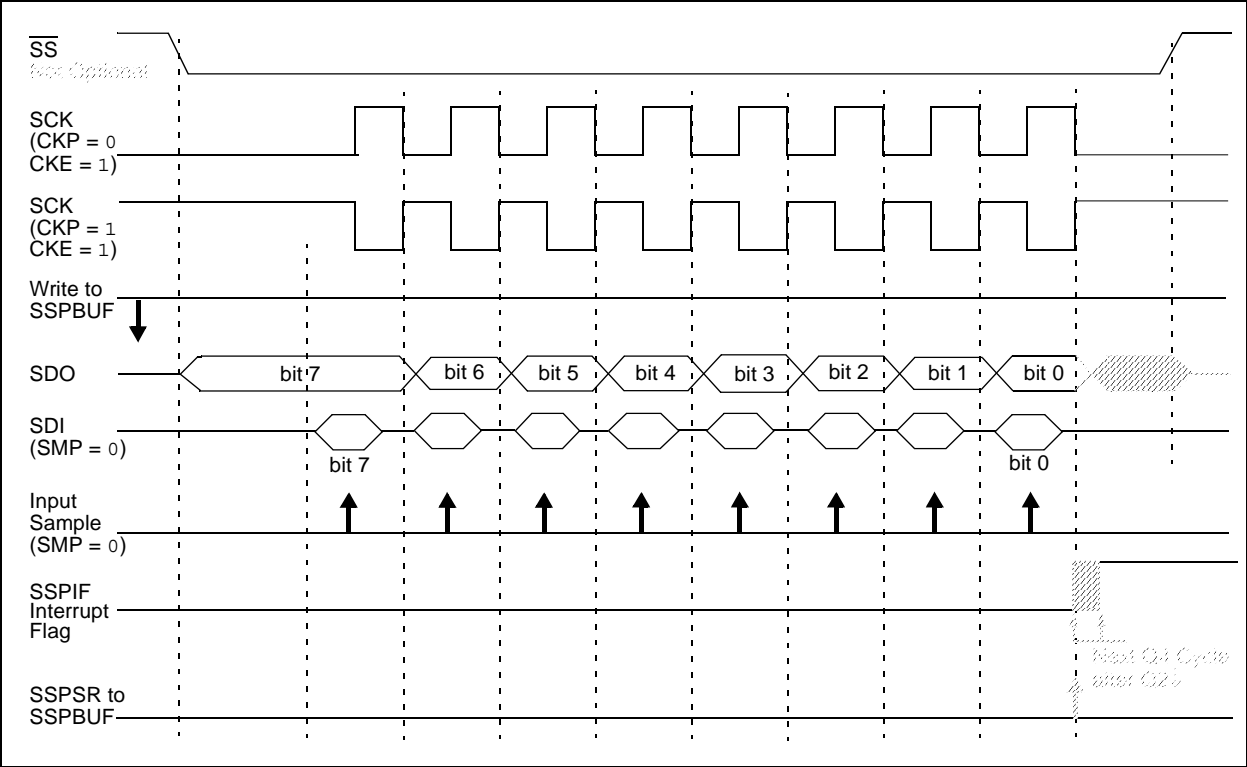


FIGURE 9-9: I²C SLAVE MODE TIMING (TRANSMISSION, 7-BIT ADDRESS)

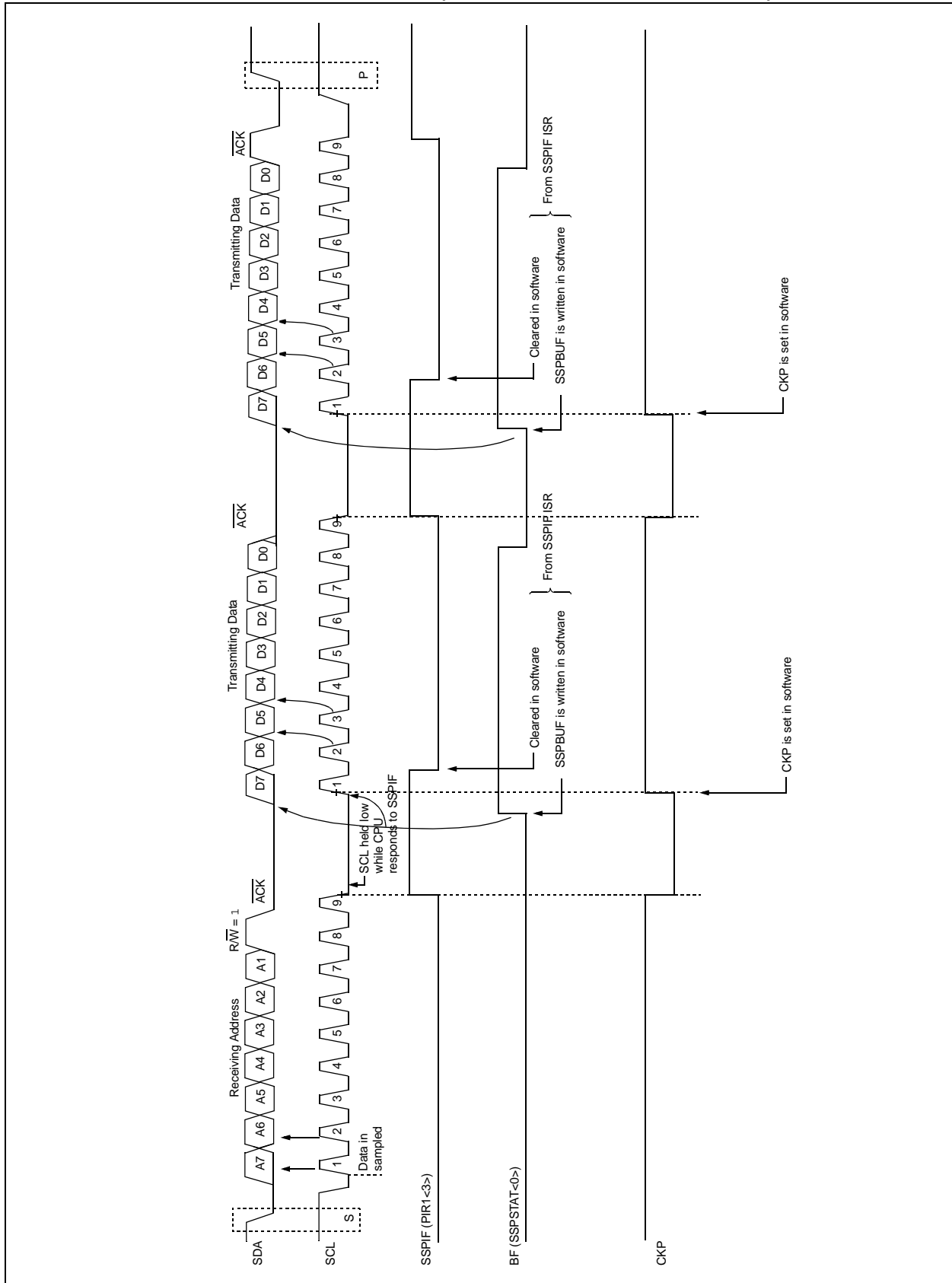
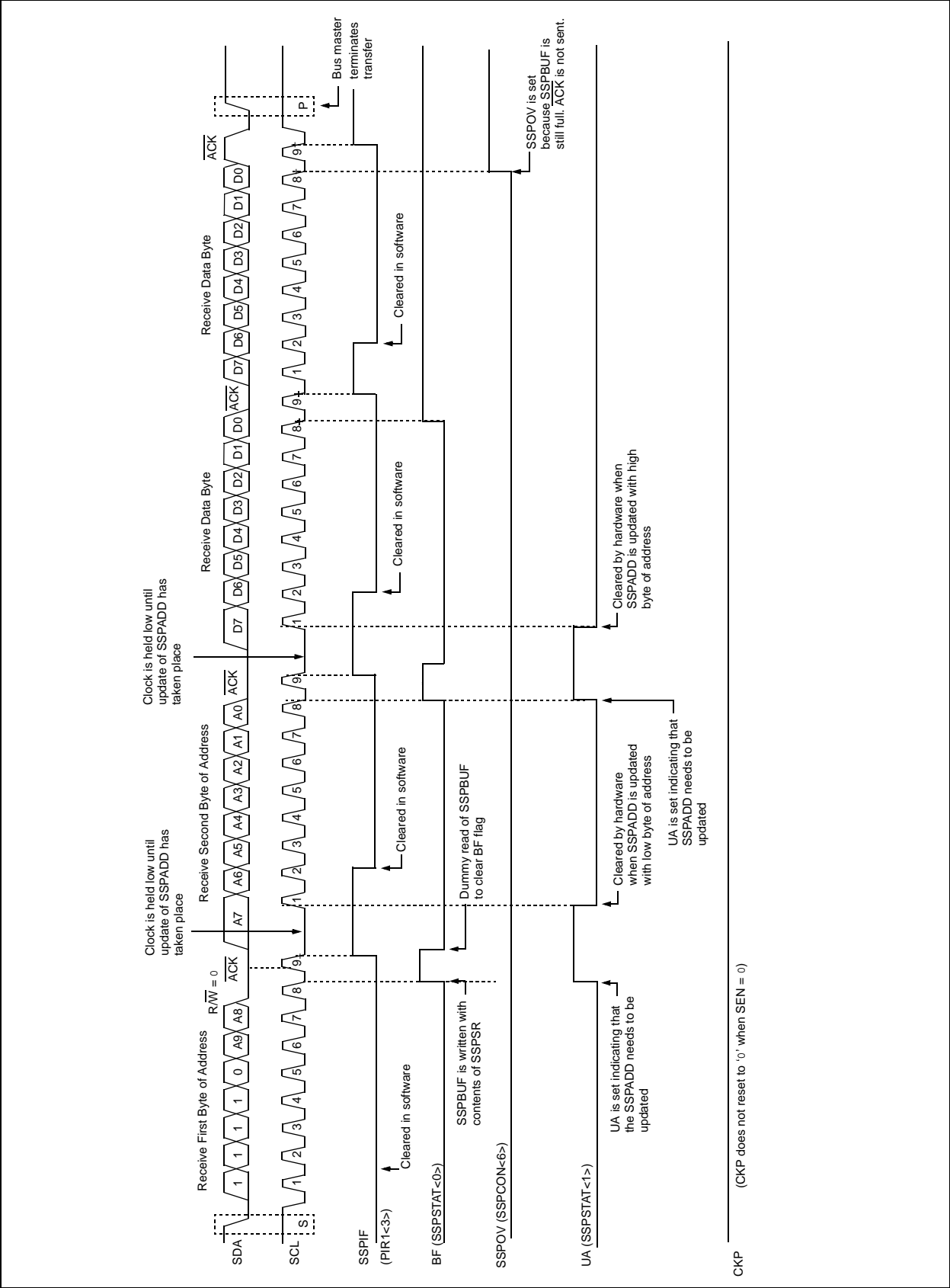


FIGURE 9-10: I²C SLAVE MODE TIMING WITH SEN = 0 (RECEPTION, 10-BIT ADDRESS)



PIC16F87XA

9.4.9 I²C MASTER MODE REPEATED START CONDITION TIMING

A Repeated Start condition occurs when the RSEN bit (SSPCON2<1>) is programmed high and the I²C logic module is in the Idle state. When the RSEN bit is set, the SCL pin is asserted low. When the SCL pin is sampled low, the Baud Rate Generator is loaded with the contents of SSPADD<5:0> and begins counting. The SDA pin is released (brought high) for one Baud Rate Generator count (TBRG). When the Baud Rate Generator times out, if SDA is sampled high, the SCL pin will be deasserted (brought high). When SCL is sampled high, the Baud Rate Generator is reloaded with the contents of SSPADD<6:0> and begins counting. SDA and SCL must be sampled high for one TBRG. This action is then followed by assertion of the SDA pin (SDA = 0) for one TBRG while SCL is high. Following this, the RSEN bit (SSPCON2<1>) will be automatically cleared and the Baud Rate Generator will not be reloaded, leaving the SDA pin held low. As soon as a Start condition is detected on the SDA and SCL pins, the S bit (SSPSTAT<3>) will be set. The SSPIF bit will not be set until the Baud Rate Generator has timed out.

Note 1: If RSEN is programmed while any other event is in progress, it will not take effect.

2: A bus collision during the Repeated Start condition occurs if:

- SDA is sampled low when SCL goes from low to high.
- SCL goes low before SDA is asserted low. This may indicate that another master is attempting to transmit a data '1'.

Immediately following the SSPIF bit getting set, the user may write the SSPBUF with the 7-bit address in 7-bit mode or the default first address in 10-bit mode. After the first eight bits are transmitted and an ACK is received, the user may then transmit an additional eight bits of address (10-bit mode) or eight bits of data (7-bit mode).

9.4.9.1 WCOL Status Flag

If the user writes the SSPBUF when a Repeated Start sequence is in progress, the WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note: Because queueing of events is not allowed, writing of the lower 5 bits of SSPCON2 is disabled until the Repeated Start condition is complete.

FIGURE 9-20: REPEAT START CONDITION WAVEFORM

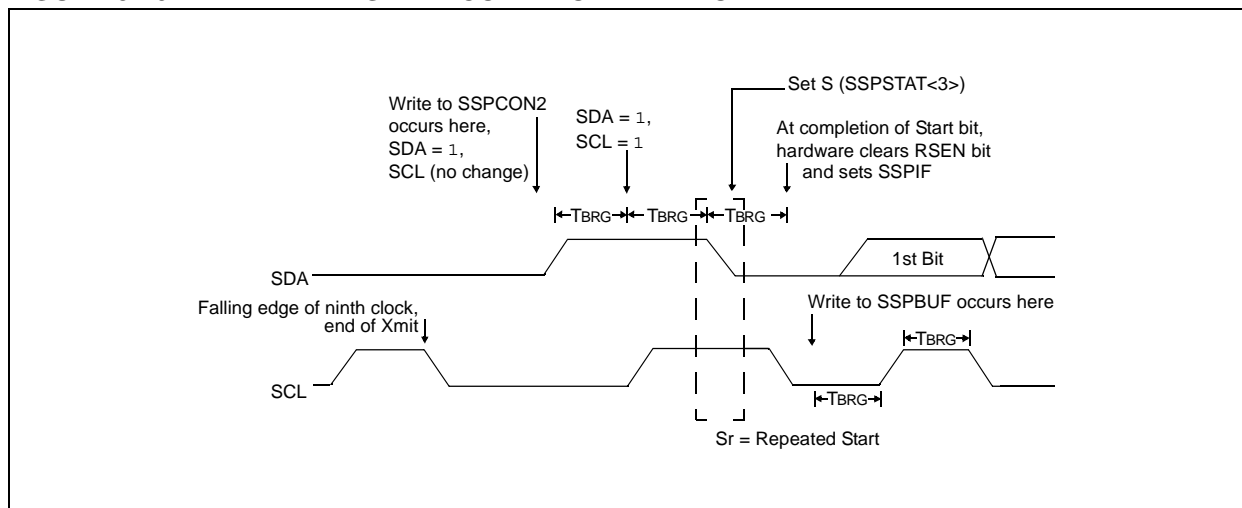
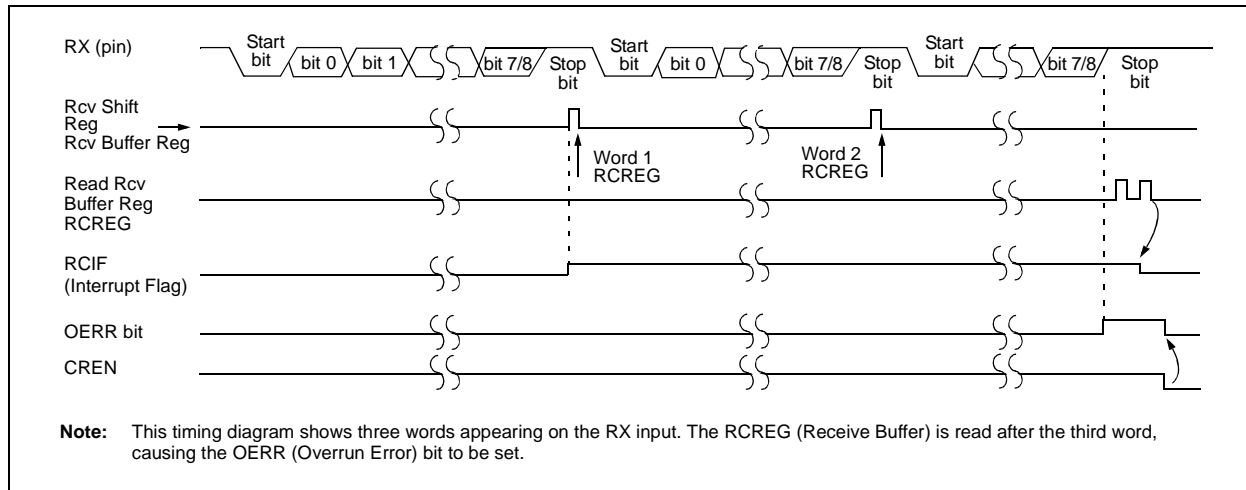


FIGURE 10-5: ASYNCHRONOUS RECEPTION



When setting up an Asynchronous Reception, follow these steps:

1. Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH (**Section 10.1 “USART Baud Rate Generator (BRG)”**).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set enable bit RCIE.
4. If 9-bit reception is desired, then set bit RX9.
5. Enable the reception by setting bit CREN.
6. Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE is set.
7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
8. Read the 8-bit received data by reading the RCREG register.
9. If any error occurred, clear the error by clearing enable bit CREN.
10. If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

TABLE 10-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.

Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

PIC16F87XA

TABLE 14-6: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Devices				Power-on Reset, Brown-out Reset	MCLR Resets, WDT Reset	Wake-up via WDT or Interrupt
W	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	73A	74A	76A	77A	N/A	N/A	N/A
TMR0	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	73A	74A	76A	77A	0000 0000	0000 0000	PC + 1 ⁽²⁾
STATUS	73A	74A	76A	77A	0001 1xxx	000q quuu ⁽³⁾	uuuq quuu ⁽³⁾
FSR	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA	73A	74A	76A	77A	--0x 0000	--0u 0000	--uu uuuu
PORTB	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTC	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTD	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTE	73A	74A	76A	77A	---- -xxx	---- -uuu	---- -uuu
PCLATH	73A	74A	76A	77A	---0 0000	---0 0000	---u uuuu
INTCON	73A	74A	76A	77A	0000 000x	0000 000u	uuuu uuuu ⁽¹⁾
PIR1	73A	74A	76A	77A	r000 0000	r000 0000	ruuu uuuu ⁽¹⁾
	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu ⁽¹⁾
PIR2	73A	74A	76A	77A	-0-0 0--0	-0-0 0--0	-u-u u--u ⁽¹⁾
TMR1L	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR1H	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	73A	74A	76A	77A	--00 0000	--uu uuuu	--uu uuuu
TMR2	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
T2CON	73A	74A	76A	77A	-000 0000	-000 0000	-uuu uuuu
SSPBUF	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
SSPCON	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
CCPR1L	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR1H	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP1CON	73A	74A	76A	77A	--00 0000	--00 0000	--uu uuuu
RCSTA	73A	74A	76A	77A	0000 000x	0000 000x	uuuu uuuu
TXREG	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
RCREG	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
CCPR2L	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR2H	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP2CON	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
ADRESH	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON0	73A	74A	76A	77A	0000 00-0	0000 00-0	uuuu uu-u
OPTION_REG	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu
TRISA	73A	74A	76A	77A	--11 1111	--11 1111	--uu uuuu
TRISB	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu
TRISC	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition, r = reserved, maintain clear. Shaded cells indicate conditions do not apply for the designated device.

Note 1: One or more bits in INTCON, PIR1 and/or PIR2 will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

3: See Table 14-5 for Reset value for specific condition.

14.14 Power-down Mode (Sleep)

Power-down mode is entered by executing a `SLEEP` instruction.

If enabled, the Watchdog Timer will be cleared but keeps running, the `PD` bit (`Status<3>`) is cleared, the `TO` (`Status<4>`) bit is set and the oscillator driver is turned off. The I/O ports maintain the status they had before the `SLEEP` instruction was executed (driving high, low or high-impedance).

For lowest current consumption in this mode, place all I/O pins at either V_{DD} or V_{SS} , ensure no external circuitry is drawing current from the I/O pin, power-down the A/D and disable external clocks. Pull all I/O pins that are high-impedance inputs, high or low externally, to avoid switching currents caused by floating inputs. The `T0CKI` input should also be at V_{DD} or V_{SS} for lowest current consumption. The contribution from on-chip pull-ups on `PORTB` should also be considered.

The `MCLR` pin must be at a logic high level (V_{IHMC}).

14.14.1 WAKE-UP FROM SLEEP

The device can wake-up from Sleep through one of the following events:

1. External Reset input on `MCLR` pin.
2. Watchdog Timer wake-up (if `WDT` was enabled).
3. Interrupt from `INT` pin, `RB` port change or peripheral interrupt.

External `MCLR` Reset will cause a device Reset. All other events are considered a continuation of program execution and cause a “wake-up”. The `TO` and `PD` bits in the `Status` register can be used to determine the cause of device Reset. The `PD` bit, which is set on power-up, is cleared when Sleep is invoked. The `TO` bit is cleared if a `WDT` time-out occurred and caused wake-up.

The following peripheral interrupts can wake the device from Sleep:

1. `PSP` read or write (PIC16F874/877 only).
2. `TMR1` interrupt. `Timer1` must be operating as an asynchronous counter.
3. `CCP` Capture mode interrupt.
4. Special event trigger (`Timer1` in Asynchronous mode using an external clock).
5. `SSP` (Start/Stop) bit detect interrupt.
6. `SSP` transmit or receive in Slave mode (`SPI/I2C`).
7. `USART` RX or TX (Synchronous Slave mode).
8. A/D conversion (when A/D clock source is `RC`).
9. `EEPROM` write operation completion.
10. Comparator output changes state.

Other peripherals cannot generate interrupts since during Sleep, no on-chip clocks are present.

When the `SLEEP` instruction is being executed, the next instruction (`PC + 1`) is prefetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up is regardless of the state of the `GIE` bit. If the `GIE` bit is clear (disabled), the device continues execution at the instruction after the `SLEEP` instruction. If the `GIE` bit is set (enabled), the device executes the instruction after the `SLEEP` instruction and then branches to the interrupt address (0004h). In cases where the execution of the instruction following `SLEEP` is not desirable, the user should have a `NOP` after the `SLEEP` instruction.

14.14.2 WAKE-UP USING INTERRUPTS

When global interrupts are disabled (`GIE` cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following will occur:

- If the interrupt occurs **before** the execution of a `SLEEP` instruction, the `SLEEP` instruction will complete as a `NOP`. Therefore, the `WDT` and `WDT` postscaler will not be cleared, the `TO` bit will not be set and `PD` bits will not be cleared.
- If the interrupt occurs **during or after** the execution of a `SLEEP` instruction, the device will immediately wake-up from Sleep. The `SLEEP` instruction will be completely executed before the wake-up. Therefore, the `WDT` and `WDT` postscaler will be cleared, the `TO` bit will be set and the `PD` bit will be cleared.

Even if the flag bits were checked before executing a `SLEEP` instruction, it may be possible for flag bits to become set before the `SLEEP` instruction completes. To determine whether a `SLEEP` instruction executed, test the `PD` bit. If the `PD` bit is set, the `SLEEP` instruction was executed as a `NOP`.

To ensure that the `WDT` is cleared, a `CLRWDT` instruction should be executed before a `SLEEP` instruction.

15.2 Instruction Descriptions

ADDLW Add Literal and W

Syntax: [*label*] ADDLW *k*

Operands: $0 \leq k \leq 255$

Operation: $(W) + k \rightarrow (W)$

Status Affected: C, DC, Z

Description: The contents of the W register are added to the eight-bit literal 'k' and the result is placed in the W register.

ADDWF Add W and f

Syntax: [*label*] ADDWF *f*,*d*

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(W) + (f) \rightarrow (\text{destination})$

Status Affected: C, DC, Z

Description: Add the contents of the W register with register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

ANDLW AND Literal with W

Syntax: [*label*] ANDLW *k*

Operands: $0 \leq k \leq 255$

Operation: $(W) .\text{AND.} (k) \rightarrow (W)$

Status Affected: Z

Description: The contents of W register are AND'ed with the eight-bit literal 'k'. The result is placed in the W register.

ANDWF AND W with f

Syntax: [*label*] ANDWF *f*,*d*

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(W) .\text{AND.} (f) \rightarrow (\text{destination})$

Status Affected: Z

Description: AND the W register with register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

BCF Bit Clear f

Syntax: [*label*] BCF *f*,*b*

Operands: $0 \leq f \leq 127$
 $0 \leq b \leq 7$

Operation: $0 \rightarrow (f)$

Status Affected: None

Description: Bit 'b' in register 'f' is cleared.

BSF Bit Set f

Syntax: [*label*] BSF *f*,*b*

Operands: $0 \leq f \leq 127$
 $0 \leq b \leq 7$

Operation: $1 \rightarrow (f)$

Status Affected: None

Description: Bit 'b' in register 'f' is set.

BTFSS Bit Test f, Skip if Set

Syntax: [*label*] BTFSS *f*,*b*

Operands: $0 \leq f \leq 127$
 $0 \leq b < 7$

Operation: skip if $(f) = 1$

Status Affected: None

Description: If bit 'b' in register 'f' is '0', the next instruction is executed. If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2 Tcy instruction.

BTFSC Bit Test, Skip if Clear

Syntax: [*label*] BTFSC *f*,*b*

Operands: $0 \leq f \leq 127$
 $0 \leq b \leq 7$

Operation: skip if $(f) = 0$

Status Affected: None

Description: If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b' in register 'f' is '0', the next instruction is discarded and a NOP is executed instead, making this a 2 Tcy instruction.

DECFSZ Decrement f, Skip if 0

Syntax: [*label*] DECFSZ f,d

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(f) - 1 \rightarrow (\text{destination});$
skip if result = 0

Status Affected: None

Description: The contents of register 'f' are decremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.
If the result is '1', the next instruction is executed. If the result is '0', then a NOP is executed instead, making it a 2 Tcy instruction.

INCFSZ Increment f, Skip if 0

Syntax: [*label*] INCFSZ f,d

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(f) + 1 \rightarrow (\text{destination});$
skip if result = 0

Status Affected: None

Description: The contents of register 'f' are incremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.
If the result is '1', the next instruction is executed. If the result is '0', a NOP is executed instead, making it a 2 Tcy instruction.

GOTO Unconditional Branch

Syntax: [*label*] GOTO k

Operands: $0 \leq k \leq 2047$

Operation: $k \rightarrow \text{PC}<10:0>$
 $\text{PCLATH}<4:3> \rightarrow \text{PC}<12:11>$

Status Affected: None

Description: GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two-cycle instruction.

IORLW Inclusive OR Literal with W

Syntax: [*label*] IORLW k

Operands: $0 \leq k \leq 255$

Operation: $(W) .OR. k \rightarrow (W)$

Status Affected: Z

Description: The contents of the W register are OR'ed with the eight-bit literal 'k'. The result is placed in the W register.

INCF Increment f

Syntax: [*label*] INCF f,d

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(f) + 1 \rightarrow (\text{destination})$

Status Affected: Z

Description: The contents of register 'f' are incremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.

IORWF Inclusive OR W with f

Syntax: [*label*] IORWF f,d

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(W) .OR. (f) \rightarrow (\text{destination})$

Status Affected: Z

Description: Inclusive OR the W register with register 'f'. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.

16.3 MPLAB C17 and MPLAB C18 C Compilers

The MPLAB C17 and MPLAB C18 Code Development Systems are complete ANSI C compilers for Microchip's PIC17CXXX and PIC18CXXX family of microcontrollers. These compilers provide powerful integration capabilities, superior code optimization and ease of use not found with other compilers.

For easy source level debugging, the compilers provide symbol information that is optimized to the MPLAB IDE debugger.

16.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK object linker combines relocatable objects created by the MPASM assembler and the MPLAB C17 and MPLAB C18 C compilers. It can link relocatable objects from precompiled libraries, using directives from a linker script.

The MPLIB object librarian manages the creation and modification of library files of pre-compiled code. When a routine from a library is called from a source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications.

The object linker/library features include:

- Efficient linking of single libraries instead of many smaller files
- Enhanced code maintainability by grouping related modules together
- Flexible creation of libraries with easy module listing, replacement, deletion and extraction

16.5 MPLAB C30 C Compiler

The MPLAB C30 C compiler is a full-featured, ANSI compliant, optimizing compiler that translates standard ANSI C programs into dsPIC30F assembly language source. The compiler also supports many command-line options and language extensions to take full advantage of the dsPIC30F device hardware capabilities, and afford fine control of the compiler code generator.

MPLAB C30 is distributed with a complete ANSI C standard library. All library functions have been validated and conform to the ANSI C library standard. The library includes functions for string manipulation, dynamic memory allocation, data conversion, time-keeping, and math functions (trigonometric, exponential and hyperbolic). The compiler provides symbolic information for high level source debugging with the MPLAB IDE.

16.6 MPLAB ASM30 Assembler, Linker, and Librarian

MPLAB ASM30 assembler produces relocatable machine code from symbolic assembly language for dsPIC30F devices. MPLAB C30 compiler uses the assembler to produce its object file. The assembler generates relocatable object files that can then be archived or linked with other relocatable object files and archives to create an executable file. Notable features of the assembler include:

- Support for the entire dsPIC30F instruction set
- Support for fixed-point and floating-point data
- Command line interface
- Rich directive set
- Flexible macro language
- MPLAB IDE compatibility

16.7 MPLAB SIM Software Simulator

The MPLAB SIM software simulator allows code development in a PC hosted environment by simulating the PIC series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user defined key press, to any pin. The execution can be performed in Single-Step, Execute Until Break, or Trace mode.

The MPLAB SIM simulator fully supports symbolic debugging using the MPLAB C17 and MPLAB C18 C Compilers, as well as the MPASM assembler. The software simulator offers the flexibility to develop and debug code outside of the laboratory environment, making it an excellent, economical software development tool.

16.8 MPLAB SIM30 Software Simulator

The MPLAB SIM30 software simulator allows code development in a PC hosted environment by simulating the dsPIC30F series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user defined key press, to any of the pins.

The MPLAB SIM30 simulator fully supports symbolic debugging using the MPLAB C30 C Compiler and MPLAB ASM30 assembler. The simulator runs in either a Command Line mode for automated tasks, or from MPLAB IDE. This high speed simulator is designed to debug, analyze and optimize time intensive DSP routines.

17.3 Timing Parameter Symbology

The timing parameter symbols have been created following one of the following formats:

1. TppS2ppS
2. TppS
3. TCC:ST (I²C specifications only)
4. Ts (I²C specifications only)

T			
F	Frequency	T	Time

Lowercase letters (pp) and their meanings:

pp			
cc	CCP1	osc	OSC1
ck	CLKO	rd	\overline{RD}
cs	\overline{CS}	rw	\overline{RD} or \overline{WR}
di	SDI	sc	SCK
do	SDO	ss	\overline{SS}
dt	Data in	t0	T0CKI
io	I/O port	t1	T1CKI
mc	\overline{MCLR}	wr	\overline{WR}

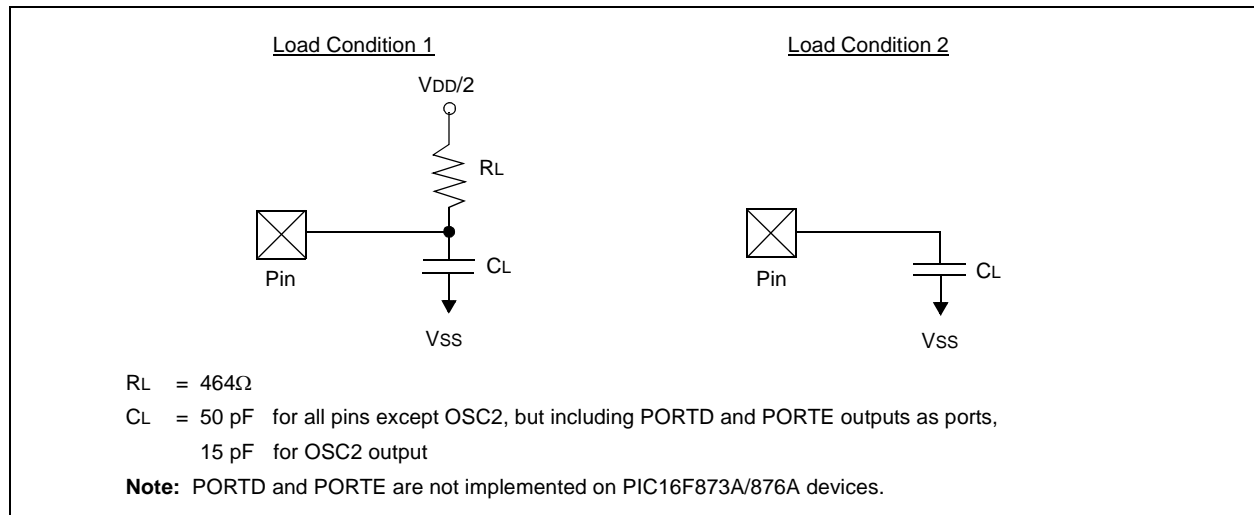
Uppercase letters and their meanings:

S			
F	Fall	P	Period
H	High	R	Rise
I	Invalid (High-impedance)	V	Valid
L	Low	Z	High-impedance
I²C only			
AA	output access	High	High
BUF	Bus free	Low	Low

TCC:ST (I²C specifications only)

CC			
HD	Hold	SU	Setup
ST			
DAT	Data input hold	STO	Stop condition
STA	Start condition		

FIGURE 17-3: LOAD CONDITIONS



PIC16F87XA

FIGURE 17-11: SPI MASTER MODE TIMING (CKE = 0, SMP = 0)

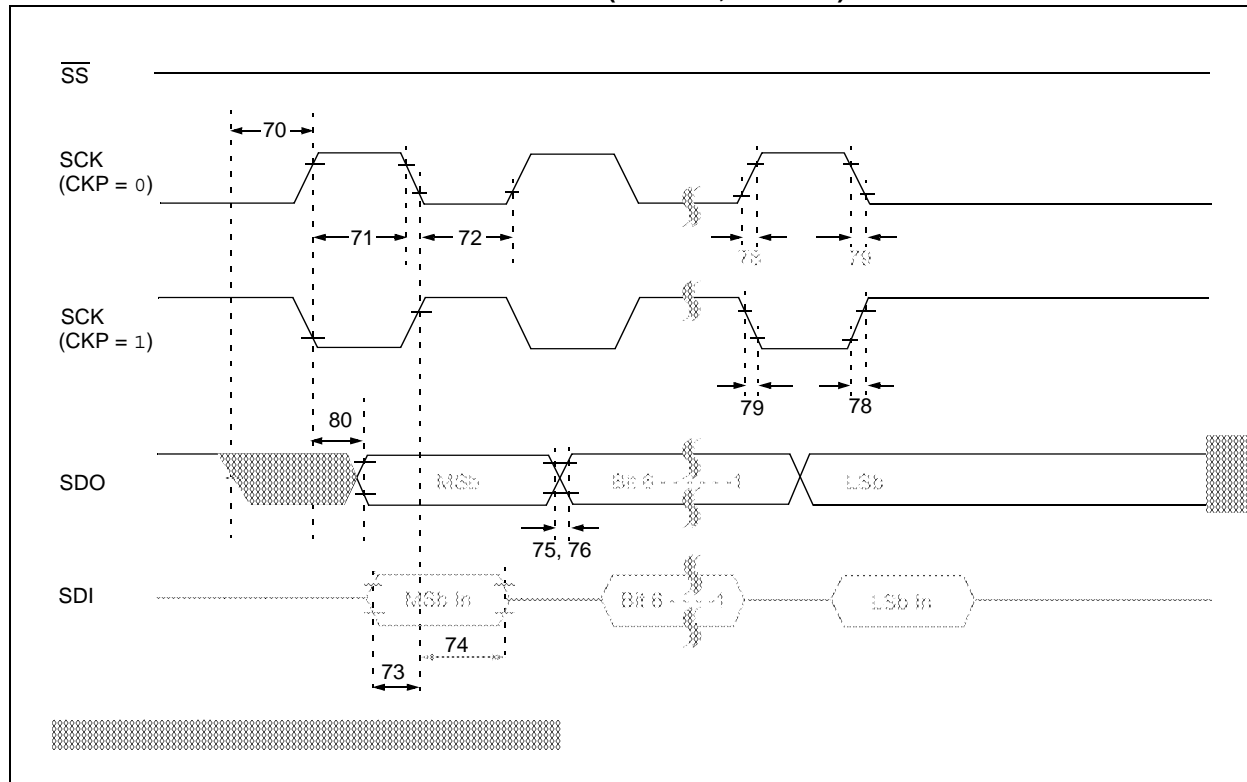


FIGURE 17-12: SPI MASTER MODE TIMING (CKE = 1, SMP = 1)

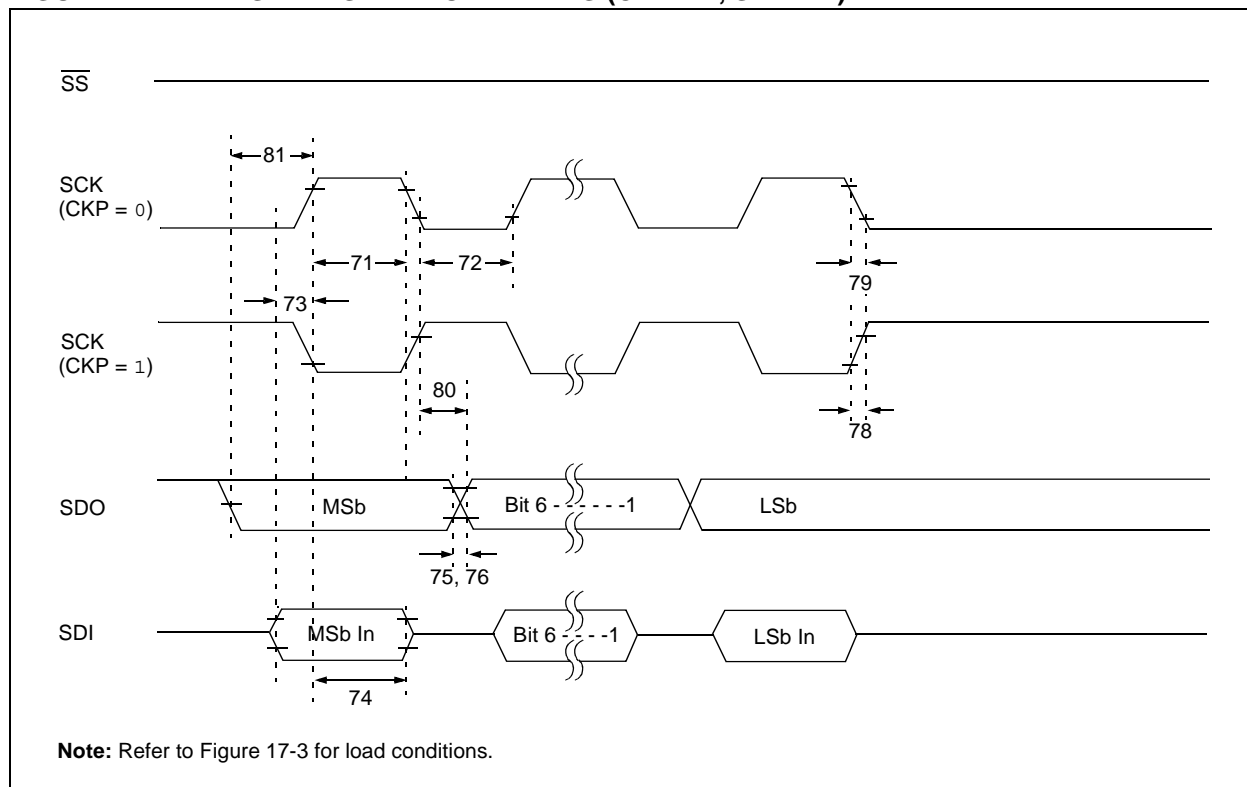
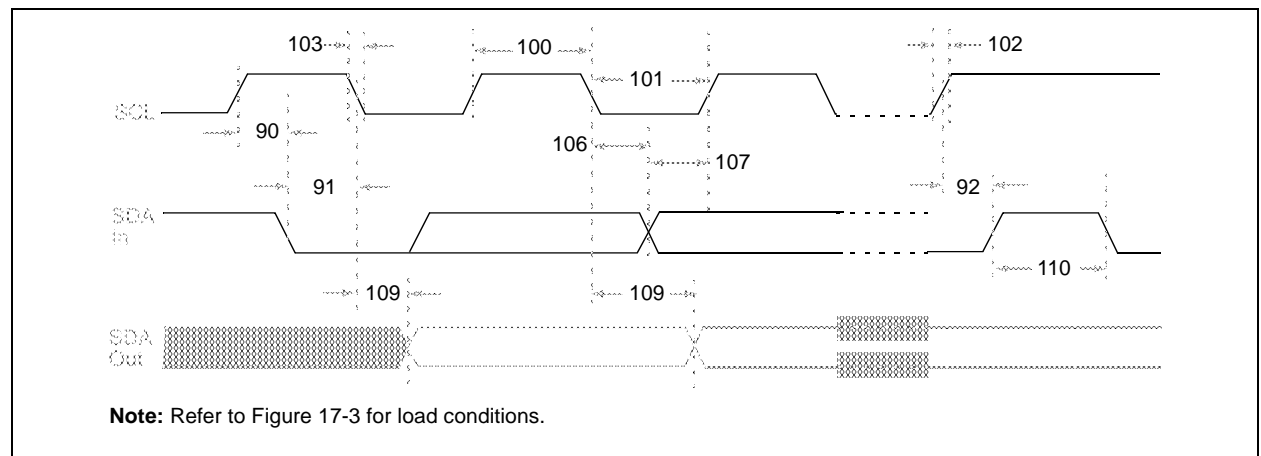


TABLE 17-10: I²C BUS START/STOP BITS REQUIREMENTS

Param No.	Symbol	Characteristic		Min	Typ	Max	Units	Conditions
90	TSU:STA	Start condition	100 kHz mode	4700	—	—	ns	Only relevant for Repeated Start condition
		Setup time	400 kHz mode	600	—	—		
91	THD:STA	Start condition	100 kHz mode	4000	—	—	ns	After this period, the first clock pulse is generated
		Hold time	400 kHz mode	600	—	—		
92	TSU:STO	Stop condition	100 kHz mode	4700	—	—	ns	
		Setup time	400 kHz mode	600	—	—		
93	THD:STO	Stop condition	100 kHz mode	4000	—	—	ns	
		Hold time	400 kHz mode	600	—	—		

FIGURE 17-16: I²C BUS DATA TIMING



PIC16F87XA

NOTES:

PIC16F87XA PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

<u>PART NO.</u>		<u>X</u>	<u>/XX</u>	<u>XXX</u>
Device	Temperature Range	Package	Pattern	
Device	PIC16F87XA ⁽¹⁾ , PIC16F87XAT ⁽²⁾ ; VDD range 4.0V to 5.5V PIC16LF87XA ⁽¹⁾ , PIC16LF87XAT ⁽²⁾ ; VDD range 2.0V to 5.5V			
Temperature Range	I	=	-40°C to +85°C (Industrial)	
Package	ML	=	QFN (Metal Lead Frame)	
	PT	=	TQFP (Thin Quad Flatpack)	
	SO	=	SOIC	
	SP	=	Skinny Plastic DIP	
	P	=	PDIP	
	L	=	PLCC	
	S	=	SSOP	

Examples:

- a) PIC16F873A-I/P 301 = Industrial temp., PDIP package, normal VDD limits, QTP pattern #301.
- b) PIC16LF876A-I/SO = Industrial temp., SOIC package, Extended VDD limits.
- c) PIC16F877A-I/P = Industrial temp., PDIP package, 10 MHz, normal VDD limits.

Note 1: F = CMOS Flash
LF = Low-Power CMOS Flash
2: T = in tape and reel - SOIC, PLCC, TQFP packages only