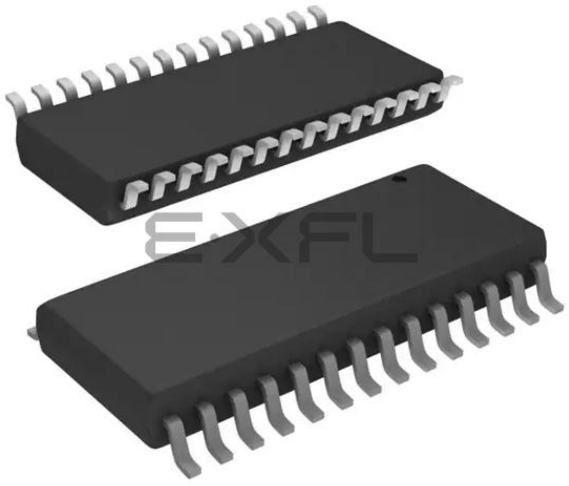
Microchip Technology - PIC18LF258-I/SO Datasheet



Welcome to 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	40MHz
Connectivity	CANbus, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	22
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	1.5K 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/pic18lf258-i-so

FIGURE 1-1: PIC18F248/258 BLOCK DIAGRAM

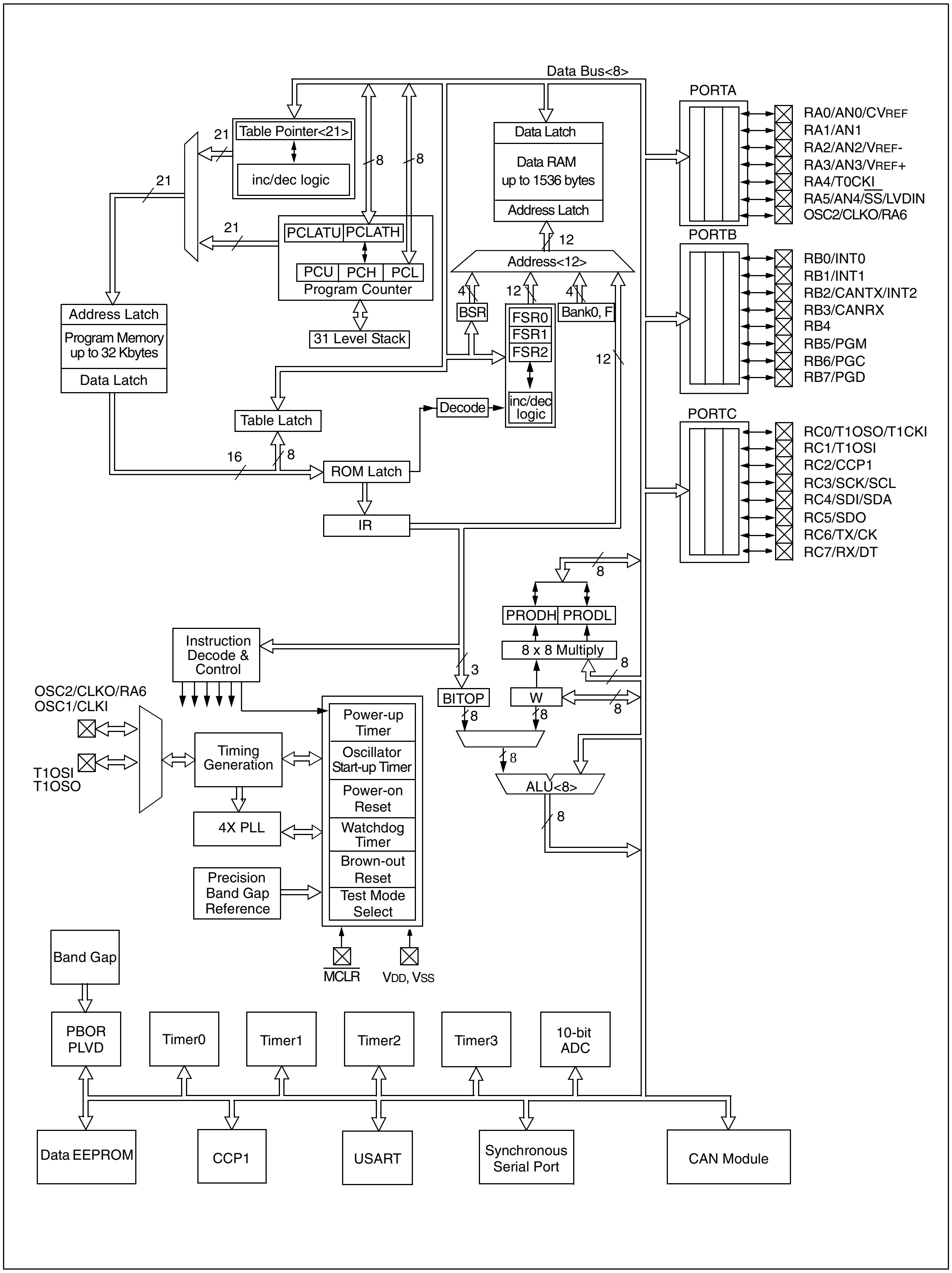


TABLE 2-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Osc Type	Crystal Freq	Cap. Range C1	Cap. Range C2
LP	32.0 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1.0 MHz	15 pF	15 pF
	4.0 MHz	15 pF	15 pF
HS	4.0 MHz	15 pF	15 pF
	8.0 MHz	15-33 pF	15-33 pF
	20.0 MHz	15-33 pF	15-33 pF
	25.0 MHz	15-33 pF	15-33 pF
These values are for design guidance only. See notes on this page.			
Crystals Used			
32.0 kHz	Epson C-001R32.768K-A	±20 PPM	
200 kHz	STD XTL 200.000KHz	±20 PPM	
1.0 MHz	ECS ECS-10-13-1	±50 PPM	
4.0 MHz	ECS ECS-40-20-1	±50 PPM	
8.0 MHz	EPSON CA-301 8.000M-C	±30 PPM	
20.0 MHz	EPSON CA-301 20.000M-C	±30 PPM	

Note 1: Recommended values of C1 and C2 are identical to the ranges tested (Table 2-1).

2: Higher capacitance increases the stability of the oscillator but also increases the start-up time.

3: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

4: Rs may be required in HS mode, as well as XT mode, to avoid overdriving crystals with low drive level specification.

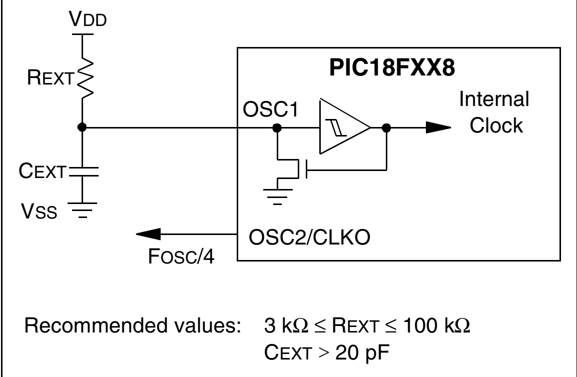
2.3 RC Oscillator

For timing insensitive applications, the "RC" and "RCIO" device options offer additional cost savings. The RC oscillator frequency is a function of the supply voltage, the resistor (REXT) and capacitor (CEXT) values and the operating temperature. In addition to this, the oscillator frequency will vary from unit to unit due to normal process parameter variation. Furthermore, the difference in lead frame capacitance between package types will also affect the oscillation frequency, especially for low CEXT values. The user also needs to take into account variation due to tolerance of external R and C components used. Figure 2-2 shows how the RC combination is connected.

In the RC Oscillator mode, the oscillator frequency divided by 4 is available on the OSC2 pin. This signal may be used for test purposes or to synchronize other logic.

Note: If the oscillator frequency divided by 4 signal is not required in the application, it is recommended to use RCIO mode to save current.

FIGURE 2-2: RC OSCILLATOR MODE



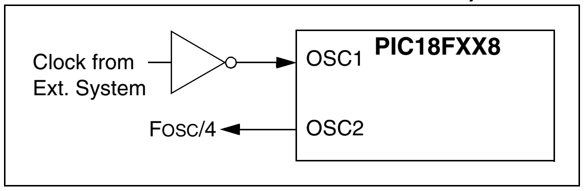
The RCIO Oscillator mode functions like the RC mode, except that the OSC2 pin becomes an additional general purpose I/O pin. The I/O pin becomes bit 6 of PORTA (RA6).

2.4 External Clock Input

The EC and ECIO Oscillator modes require an external clock source to be connected to the OSC1 pin. The feedback device between OSC1 and OSC2 is turned off in these modes to save current. There is no oscillator start-up time required after a Power-on Reset or after a recovery from Sleep mode.

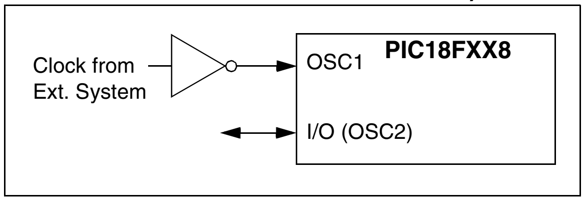
In the EC Oscillator mode, the oscillator frequency divided by 4 is available on the OSC2 pin. This signal may be used for test purposes or to synchronize other logic. Figure 2-3 shows the pin connections for the EC Oscillator mode.

FIGURE 2-3: EXTERNAL CLOCK INPUT OPERATION (EC OSC CONFIGURATION)



The ECIO Oscillator mode functions like the EC mode, except that the OSC2 pin becomes an additional general purpose I/O pin. Figure 2-4 shows the pin connections for the ECIO Oscillator mode.

FIGURE 2-4: EXTERNAL CLOCK INPUT OPERATION (ECIO CONFIGURATION)



2.5 HS4 (PLL)

A Phase Locked Loop circuit is provided as a programmable option for users that want to multiply the frequency of the incoming crystal oscillator signal by 4. For an input clock frequency of 10 MHz, the internal clock frequency will be multiplied to 40 MHz. This is useful for customers who are concerned with EMI due to high-frequency crystals.

The PLL can only be enabled when the oscillator configuration bits are programmed for HS mode. If they are programmed for any other mode, the PLL is not enabled and the system clock will come directly from OSC1.

The PLL is one of the modes of the FOSC2:FOSC0 configuration bits. The oscillator mode is specified during device programming.

A PLL lock timer is used to ensure that the PLL has locked before device execution starts. The PLL lock timer has a time-out referred to as TPLL.

FIGURE 2-5: PLL BLOCK DIAGRAM

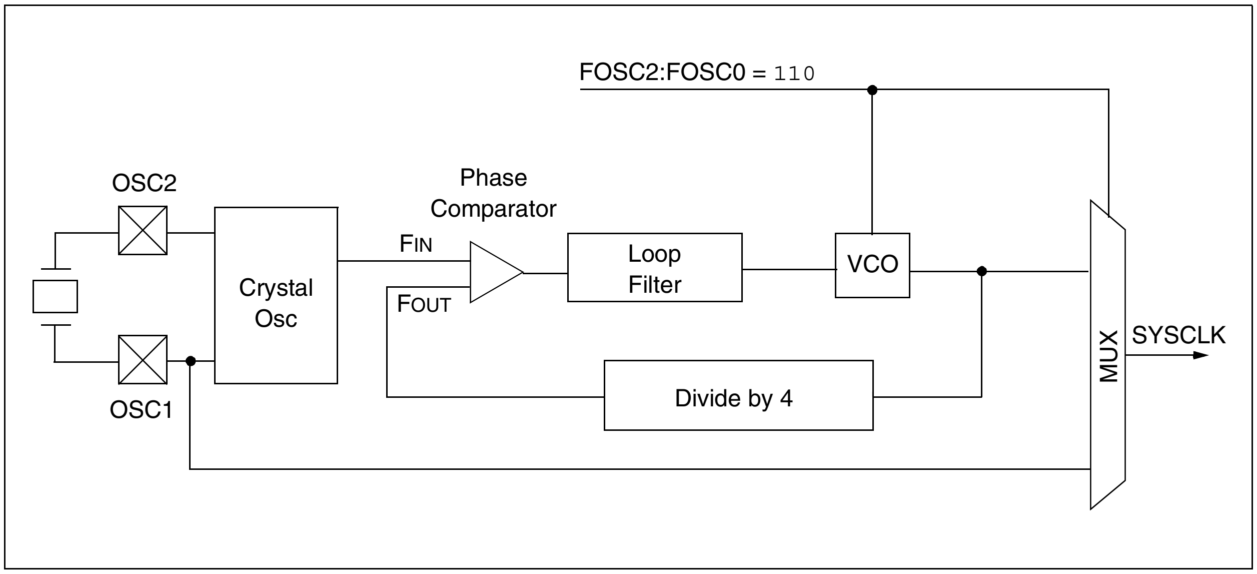


TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices		Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Reset WDT Reset `RESET` Instruction Stack Resets	Wake-up via WDT or Interrupt
ADCON1	PIC18F2X8	PIC18F4X8	`00-- 0000`	`00-- 0000`	`uu-- uuuu`
CCPR1H	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
CCPR1L	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
CCP1CON	PIC18F2X8	PIC18F4X8	`--00 0000`	`--00 0000`	`--uu uuuu`
ECCPR1H	PIC18F2X8 ▨	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
ECCPR1L	PIC18F2X8 ▨	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
ECCP1CON	PIC18F2X8 ▨	PIC18F4X8	`0000 0000`	`0000 0000`	`0000 0000`
ECCP1DEL	PIC18F2X8 ▨	PIC18F4X8	`0000 0000`	`0000 0000`	`0000 0000`
ECCPAS	PIC18F2X8 ▨	PIC18F4X8	`0000 0000`	`0000 0000`	`0000 0000`
CVRCON	PIC18F2X8 ▨	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
CMCON	PIC18F2X8 ▨	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
TMR3H	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
TMR3L	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
T3CON	PIC18F2X8	PIC18F4X8	`0000 0000`	`uuuu uuuu`	`uuuu uuuu`
SPBRG	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
RCREG	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
TXREG	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
TXSTA	PIC18F2X8	PIC18F4X8	`0000 -010`	`0000 -010`	`uuuu -uuu`
RCSTA	PIC18F2X8	PIC18F4X8	`0000 000x`	`0000 000u`	`uuuu uuuu`
EEADR	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
EEDATA	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
EECON2	PIC18F2X8	PIC18F4X8	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
EECON1	PIC18F2X8	PIC18F4X8	`xx-0 x000`	`uu-0 u000`	`uu-0 u000`
IPR3	PIC18F2X8	PIC18F4X8	`1111 1111`	`1111 1111`	`uuuu uuuu`
PIR3	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
PIE3	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`
IPR2	PIC18F2X8	PIC18F4X8	`-1-1 1111`	`-1-1 1111`	`-u-u uuuu`
PIR2	PIC18F2X8	PIC18F4X8	`-0-0 0000`	`-0-0 0000`	`-u-u uuuu`(1)
PIE2	PIC18F2X8	PIC18F4X8	`-0-0 0000`	`-0-0 0000`	`-u-u uuuu`
IPR1	PIC18F2X8	PIC18F4X8	`1111 1111`	`1111 1111`	`uuuu uuuu`
PIR1	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`(1)
PIE1	PIC18F2X8	PIC18F4X8	`0000 0000`	`0000 0000`	`uuuu uuuu`

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

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: See Table 3-2 for Reset value for specific condition.

5: Bit 6 of PORTA, LATA and TRISA are enabled in ECIO and RCIO Oscillator modes only. In all other oscillator modes, they are disabled and read '`0`'.

6: Values for CANSTAT also apply to its other instances (CANSTATRO1 through CANSTATRO4).

Example 7-3 shows the sequence to do a 16 x 16 unsigned multiply. Equation 7-1 shows the algorithm that is used. The 32-bit result is stored in four registers, RES3:RES0.

EQUATION 7-1: 16 x 16 UNSIGNED MULTIPLICATION ALGORITHM

$$
\begin{aligned}
\text{RES3:RES0} &= \text{ARG1H:ARG1L} \bullet \text{ARG2H:ARG2L}\\
&= (\text{ARG1H} \bullet \text{ARG2H} \bullet 2^{16}) +\\
&\quad (\text{ARG1H} \bullet \text{ARG2L} \bullet 2^{8}) +\\
&\quad (\text{ARG1L} \bullet \text{ARG2H} \bullet 2^{8}) +\\
&\quad (\text{ARG1L} \bullet \text{ARG2L})
\end{aligned}
$$

EXAMPLE 7-3: 16 x 16 UNSIGNED MULTIPLY ROUTINE

```
    MOVF    ARG1L, W
    MULWF   ARG2L         ; ARG1L * ARG2L ->
                          ; PRODH:PRODL
    MOVFF   PRODH, RES1   ;
    MOVFF   PRODL, RES0   ;
;
    MOVF    ARG1H, W
    MULWF   ARG2H         ; ARG1H * ARG2H ->
                          ; PRODH:PRODL
    MOVFF   PRODH, RES3   ;
    MOVFF   PRODL, RES2   ;
;
    MOVF    ARG1L, W
    MULWF   ARG2H         ; ARG1L * ARG2H ->
                          ; PRODH:PRODL
    MOVF    PRODL, W      ;
    ADDWF   RES1          ; Add cross
    MOVF    PRODH, W      ; products
    ADDWFC  RES2          ;
    CLRF    WREG          ;
    ADDWFC  RES3          ;
;
    MOVF    ARG1H, W      ;
    MULWF   ARG2L         ; ARG1H * ARG2L ->
                          ; PRODH:PRODL
    MOVF    PRODL, W      ;
    ADDWF   RES1          ; Add cross
    MOVF    PRODH, W      ; products
    ADDWFC  RES2          ;
    CLRF    WREG          ;
    ADDWFC  RES3          ;
```

Example 7-4 shows the sequence to do a 16 x 16 signed multiply. Equation 7-2 shows the algorithm used. The 32-bit result is stored in four registers, RES3:RES0. To account for the sign bits of the arguments, each argument pair's Most Significant bit (MSb) is tested and the appropriate subtractions are done.

EQUATION 7-2: 16 x 16 SIGNED MULTIPLICATION ALGORITHM

$$
\begin{aligned}
&\text{RES3:RES0}\\
&= \text{ARG1H:ARG1L} \bullet \text{ARG2H:ARG2L}\\
&= (\text{ARG1H} \bullet \text{ARG2H} \bullet 2^{16}) +\\
&\quad (\text{ARG1H} \bullet \text{ARG2L} \bullet 2^{8}) +\\
&\quad (\text{ARG1L} \bullet \text{ARG2H} \bullet 2^{8}) +\\
&\quad (\text{ARG1L} \bullet \text{ARG2L})+\\
&\quad (-1 \bullet \text{ARG2H<7>} \bullet \text{ARG1H:ARG1L} \bullet 2^{16}) +\\
&\quad (-1 \bullet \text{ARG1H<7>} \bullet \text{ARG2H:ARG2L} \bullet 2^{16})
\end{aligned}
$$

EXAMPLE 7-4: 16 x 16 SIGNED MULTIPLY ROUTINE

```
    MOVF    ARG1L, W
    MULWF   ARG2L         ; ARG1L * ARG2L ->
                          ; PRODH:PRODL
    MOVFF   PRODH, RES1   ;
    MOVFF   PRODL, RES0   ;
;
    MOVF    ARG1H, W
    MULWF   ARG2H         ; ARG1H * ARG2H ->
                          ; PRODH:PRODL
    MOVFF   PRODH, RES3   ;
    MOVFF   PRODL, RES2   ;
;
    MOVF    ARG1L, W
    MULWF   ARG2H         ; ARG1L * ARG2H ->
                          ; PRODH:PRODL
    MOVF    PRODL, W      ;
    ADDWF   RES1          ; Add cross
    MOVF    PRODH, W      ; products
    ADDWFC  RES2          ;
    CLRF    WREG          ;
    ADDWFC  RES3          ;
;
    MOVF    ARG1H, W      ;
    MULWF   ARG2L         ; ARG1H * ARG2L ->
                          ; PRODH:PRODL
    MOVF    PRODL, W      ;
    ADDWF   RES1          ; Add cross
    MOVF    PRODH, W      ; products
    ADDWFC  RES2          ;
    CLRF    WREG          ;
    ADDWFC  RES3          ;
;
    BTFSS   ARG2H, 7      ; ARG2H:ARG2L neg?
    BRA     SIGN_ARG1     ; no, check ARG1
    MOVF    ARG1L, W      ;
    SUBWF   RES2          ;
    MOVF    ARG1H, W      ;
    SUBWFB  RES3
;
SIGN_ARG1
    BTFSS   ARG1H, 7      ; ARG1H:ARG1L neg?
    BRA     CONT_CODE     ; no, done
    MOVF    ARG2L, W      ;
    SUBWF   RES2          ;
    MOVF    ARG2H, W      ;
    SUBWFB  RES3
;
CONT_CODE
      :
```

REGISTER 17-2: SSPCON1: MSSP CONTROL REGISTER 1 (SPI MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

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

<u>SPI Slave mode:</u>
`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 $\overline{\text{SS}}$ 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, $\overline{\text{SS}}$ pin control disabled, $\overline{\text{SS}}$ can be used as I/O pin
`0100` = SPI Slave mode, clock = SCK pin, $\overline{\text{SS}}$ 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 I^{2}C mode only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

REGISTER 17-4: SSPCON1: MSSP CONTROL REGISTER 1 (I^2C MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:
1 = A write to the SSPBUF register was attempted while the I^2C conditions were not valid for a transmission 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 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:** Receive Overflow Indicator bit

In Receive mode:
1 = A byte is received while the SSPBUF register is still holding the previous byte (must be cleared in software)
0 = No overflow

In Transmit mode:
This is a “don’t care” bit in Transmit mode.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit

1 = Enables the serial port and configures the 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 input or output.

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:** Synchronous Serial Port Mode Select bits

1111 = I^2C Slave mode, 10-bit address with Start and Stop bit interrupts enabled
1110 = I^2C Slave mode, 7-bit address with Start and Stop bit interrupts enabled
1011 = I^2C Firmware Controlled Master mode (Slave Idle)
1000 = I^2C Master mode, clock = FOSC/(4 * (SSPADD + 1))
0111 = I^2C Slave mode, 10-bit address
0110 = I^2C Slave mode, 7-bit address

Note: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

FIGURE 17-9: I^{2}C™ SLAVE MODE TIMING (TRANSMISSION, 7-BIT ADDRESS)

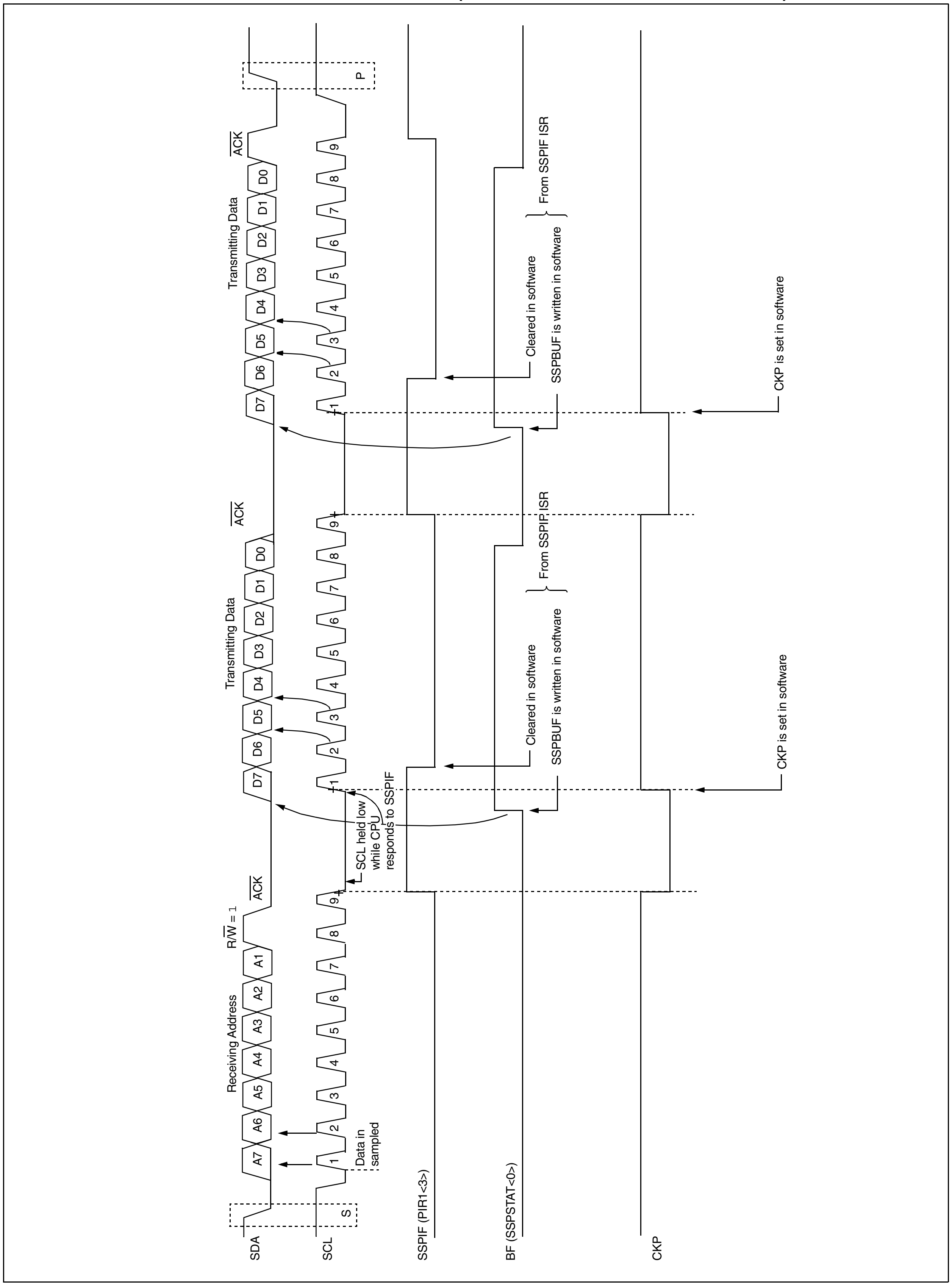
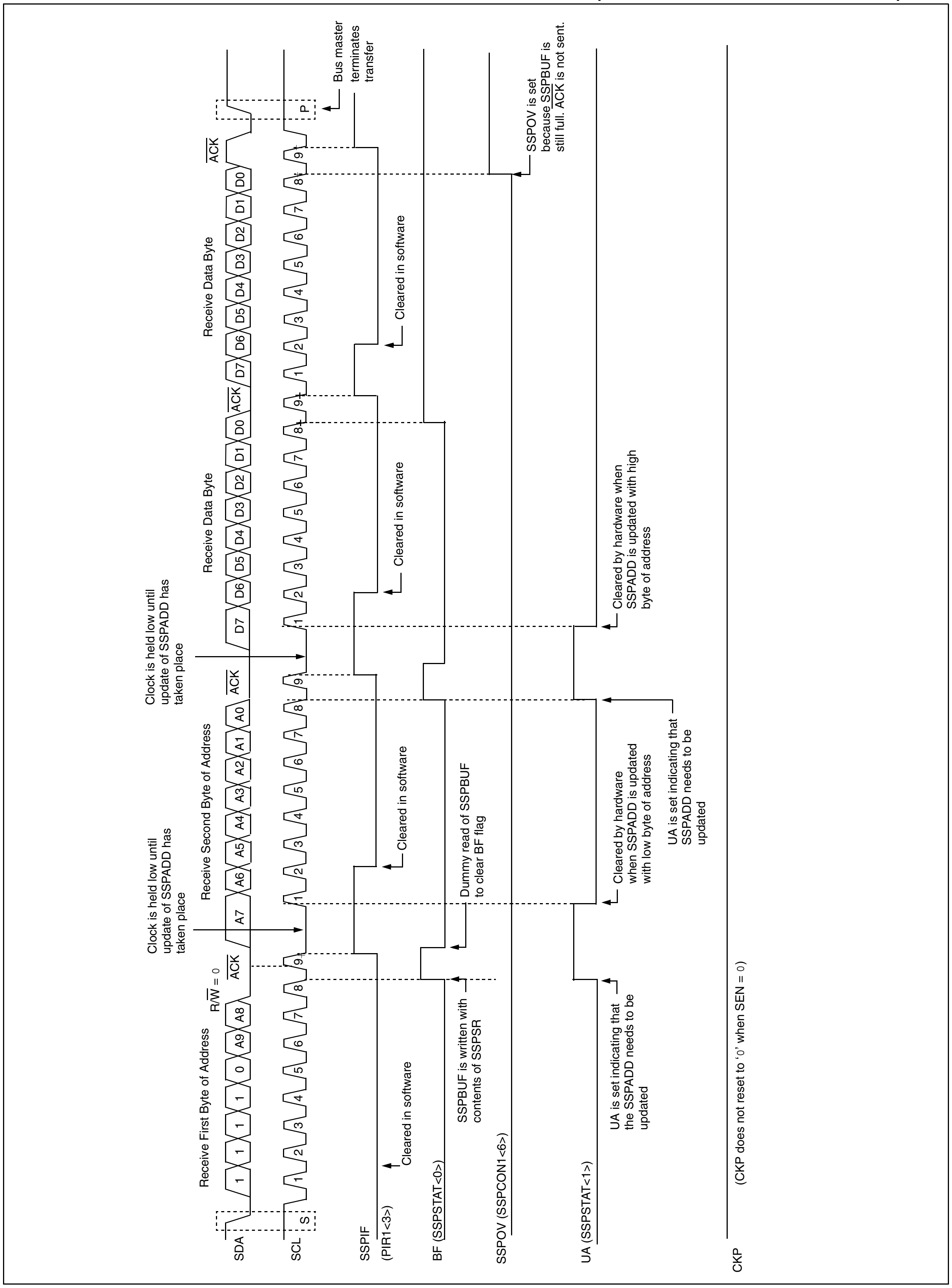


FIGURE 17-10: I²C™ SLAVE MODE TIMING WITH SEN = 0 (RECEPTION, 10-BIT ADDRESS)


SDA
SCL
SSPIF
(PIR1<3>)
BF (SSPSTAT<0>)
SSPOV (SSPCON1<6>)
UA (SSPSTAT<1>)
CKP
Receive First Byte of Address
1 1 1 1 0 A9 A8
R/W = 0
ACK
Receive Second Byte of Address
A7 A6 A5 A4 A3 A2 A1 A0
ACK
Receive Data Byte
D7 D6 D5 D4 D3 D2 D1 D0
ACK
Receive Data Byte
D7 D6 D5 D4 D3 D2 D1 D0
ACK
S
P
1 2 3 4 5 6 7 8 9
Clock is held low until update of SSPADD has taken place
Clock is held low until update of SSPADD has taken place
Bus master terminates transfer
Cleared in software
Cleared in software
Cleared in software
Cleared in software
Cleared in software
SSPBUF is written with contents of SSPSR
Dummy read of SSPBUF to clear BF flag
SSPOV is set because SSPBUF is still full. ACK is not sent.
UA is set indicating that the SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with low byte of address
UA is set indicating that SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with high byte of address
(CKP does not reset to '0' when SEN = 0)

17.4.5 GENERAL CALL ADDRESS SUPPORT

The addressing procedure for the I^{2}C bus is such that the first byte after the Start condition usually determines which device will be the slave addressed by the master. The exception is the general call address which can address all devices. When this address is used, all devices should, in theory, respond with an Acknowledge.

The general call address is one of eight addresses reserved for specific purposes by the I^{2}C protocol. It consists of all '0's with R/$\overline{W}$ = 0.

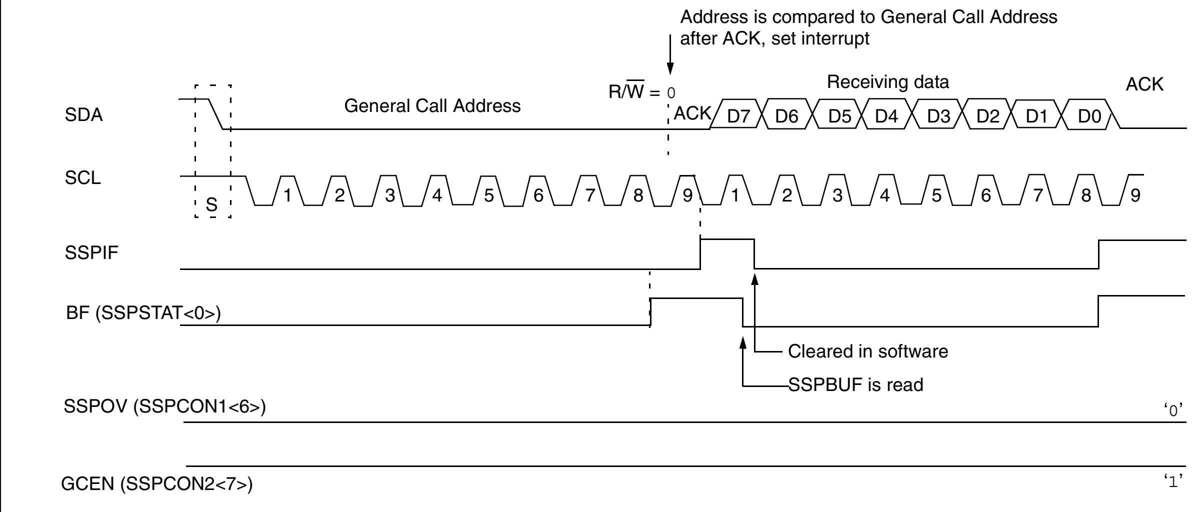
The general call address is recognized when the General Call Enable bit (GCEN) is enabled (SSPCON2<7> set). Following a Start bit detect, 8 bits are shifted into the SSPSR and the address is compared against the SSPADD. It is also compared to the general call address and fixed in hardware.

If the general call address matches, the SSPSR is transferred to the SSPBUF, the BF flag bit is set (eighth bit) and on the falling edge of the ninth bit ($\overline{ACK}$ bit), the SSPIF interrupt flag bit is set.

When the interrupt is serviced, the source for the interrupt can be checked by reading the contents of the SSPBUF. The value can be used to determine if the address was device specific or a general call address.

In 10-bit mode, the SSPADD is required to be updated for the second half of the address to match and the UA bit is set (SSPSTAT<1>). If the general call address is sampled when the GCEN bit is set, while the slave is configured in 10-bit Address mode, then the second half of the address is not necessary, the UA bit will not be set and the slave will begin receiving data after the Acknowledge (Figure 17-15).

FIGURE 17-15: SLAVE MODE GENERAL CALL ADDRESS SEQUENCE (7 OR 10-BIT ADDRESS MODE)



17.4.6.1 I^{2}C Master Mode Operation

The master device generates all of the serial clock pulses and the Start and Stop conditions. A transfer is ended with a Stop condition, or with a Repeated Start condition. Since the Repeated Start condition is also the beginning of the next serial transfer, the I^{2}C bus will not be released.

In Master Transmitter mode, serial data is output through SDA while SCL outputs the serial clock. The first byte transmitted contains the slave address of the receiving device (7 bits) and the Read/$\overline{\text{Write}}$ (R/$\overline{\text{W}}$) bit. In this case, the R/$\overline{\text{W}}$ bit will be logic '0'. Serial data is transmitted 8 bits at a time. After each byte is transmitted, an Acknowledge bit is received. Start and Stop conditions are output to indicate the beginning and the end of a serial transfer.

In Master Receive mode, the first byte transmitted contains the slave address of the transmitting device (7 bits) and the R/$\overline{\text{W}}$ bit. In this case, the R/$\overline{\text{W}}$ bit will be logic '1'. Thus, the first byte transmitted is a 7-bit slave address followed by a '1' to indicate receive bit. Serial data is received via SDA while SCL outputs the serial clock. Serial data is received 8 bits at a time. After each byte is received, an Acknowledge bit is transmitted. Start and Stop conditions indicate the beginning and end of transmission.

The Baud Rate Generator used for the SPI mode operation is used to set the SCL clock frequency for either 100 kHz, 400 kHz or 1 MHz I^{2}C operation. See **Section 17.4.7 "Baud Rate Generator"** for more details.

A typical transmit sequence would go as follows:

1. The user generates a Start condition by setting the Start Enable bit, SEN (SSPCON2<0>).
2. SSPIF is set. The MSSP module will wait the required start time before any other operation takes place.
3. The user loads the SSPBUF with the slave address to transmit.
4. Address is shifted out the SDA pin until all 8 bits are transmitted.
5. The MSSP module shifts in the ACK bit from the slave device and writes its value into the SSPCON2 register (SSPCON2<6>).
6. The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
7. The user loads the SSPBUF with eight bits of data.
8. Data is shifted out the SDA pin until all 8 bits are transmitted.
9. The MSSP module shifts in the ACK bit from the slave device and writes its value into the SSPCON2 register (SSPCON2<6>).
10. The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
11. The user generates a Stop condition by setting the Stop Enable bit PEN (SSPCON2<2>).
12. Interrupt is generated once the Stop condition is complete.

17.4.17.2 Bus Collision During a Repeated Start Condition

During a Repeated Start condition, a bus collision occurs if:

a) A low level is sampled on SDA when SCL goes from low level to high level.

b) SCL goes low before SDA is asserted low, indicating that another master is attempting to transmit a data '1'.

When the user deasserts SDA and the pin is allowed to float high, the BRG is loaded with SSPADD<6:0> and counts down to 0. The SCL pin is then deasserted and when sampled high, the SDA pin is sampled.

If SDA is low, a bus collision has occurred (i.e., another master is attempting to transmit a data '0', Figure 17-29). If SDA is sampled high, the BRG is reloaded and begins counting. If SDA goes from high-to-low before the BRG times out, no bus collision occurs because no two masters can assert SDA at exactly the same time.

If SCL goes from high-to-low before the BRG times out and SDA has not already been asserted, a bus collision occurs. In this case, another master is attempting to transmit a data '1' during the Repeated Start condition (Figure 17-30).

If, at the end of the BRG time-out, both SCL and SDA are still high, the SDA pin is driven low and the BRG is reloaded and begins counting. At the end of the count, regardless of the status of the SCL pin, the SCL pin is driven low and the Repeated Start condition is complete.

FIGURE 17-29: BUS COLLISION DURING A REPEATED START CONDITION (CASE 1)

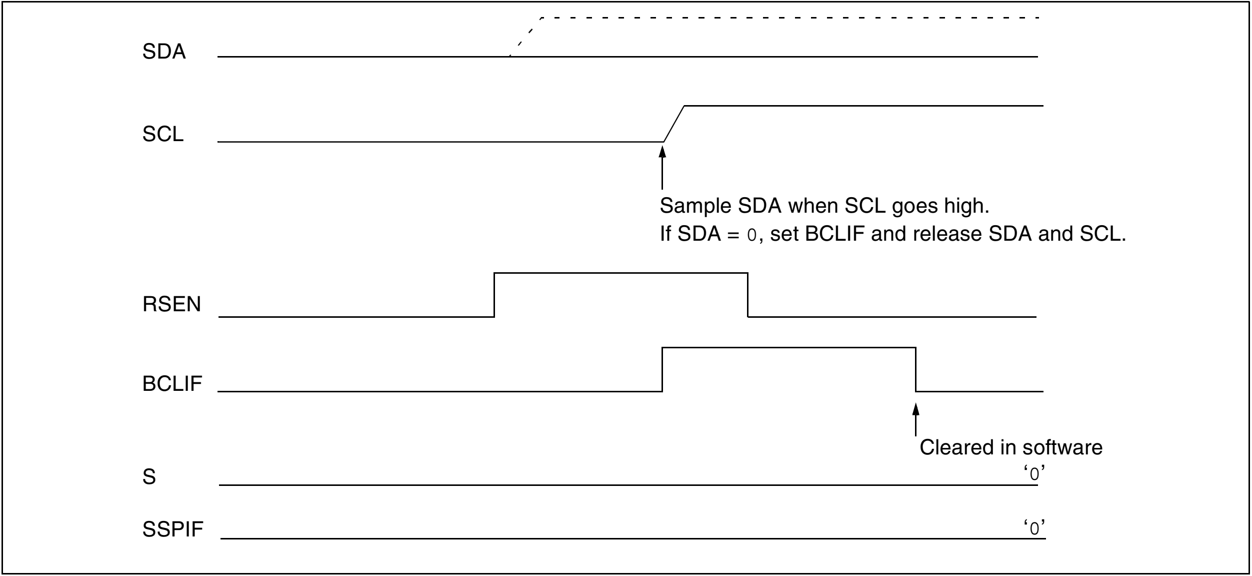
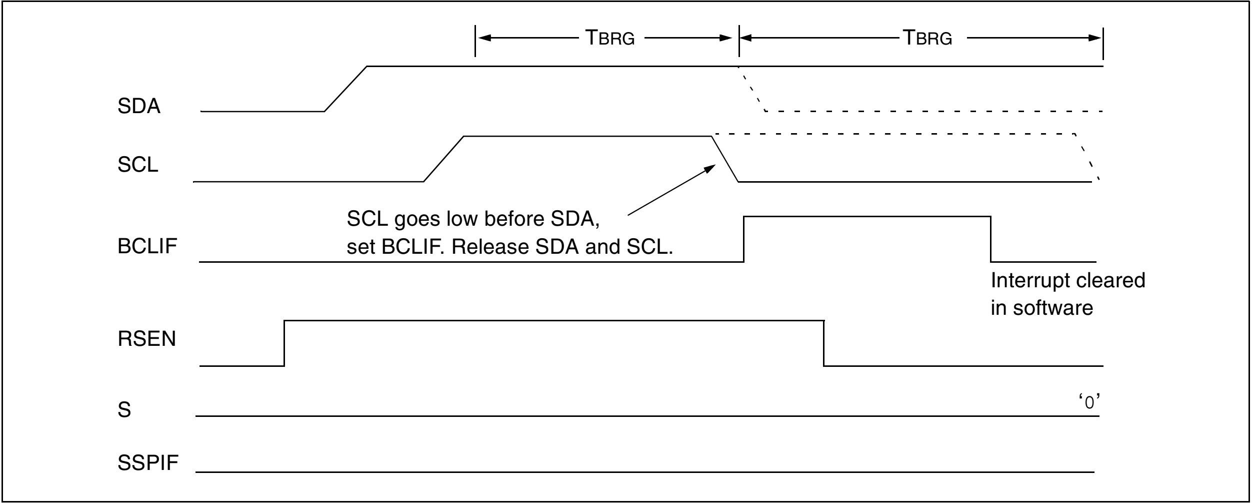

SDA
SCL
Sample SDA when SCL goes high.
If SDA = 0, set BCLIF and release SDA and SCL.
RSEN
BCLIF
Cleared in software
S
'0'
SSPIF
'0'

FIGURE 17-30: BUS COLLISION DURING A REPEATED START CONDITION (CASE 2)


TBRG
TBRG
SDA
SCL
SCL goes low before SDA, set BCLIF. Release SDA and SCL.
BCLIF
Interrupt cleared in software
RSEN
S
'0'
SSPIF

19.2.6 CAN INTERRUPT REGISTERS

The registers in this section are the same as described in **Section 8.0 "Interrupts"**. They are duplicated here for convenience.

REGISTER 19-33: PIR3: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRXIF	WAKIF	ERRIF	TXB2IF	TXB1IF	TXB0IF	RXB1IF	RXB0IF
bit 7							bit 0

bit 7	**IRXIF:** CAN Invalid Received Message Interrupt Flag bit 1 = An invalid message has occurred on the CAN bus 0 = No invalid message on CAN bus
bit 6	**WAKIF:** CAN bus Activity Wake-up Interrupt Flag bit 1 = Activity on CAN bus has occurred 0 = No activity on CAN bus
bit 5	**ERRIF:** CAN bus Error Interrupt Flag bit 1 = An error has occurred in the CAN module (multiple sources) 0 = No CAN module errors
bit 4	**TXB2IF:** CAN Transmit Buffer 2 Interrupt Flag bit 1 = Transmit Buffer 2 has completed transmission of a message and may be reloaded 0 = Transmit Buffer 2 has not completed transmission of a message
bit 3	**TXB1IF:** CAN Transmit Buffer 1 Interrupt Flag bit 1 = Transmit Buffer 1 has completed transmission of a message and may be reloaded 0 = Transmit Buffer 1 has not completed transmission of a message
bit 2	**TXB0IF:** CAN Transmit Buffer 0 Interrupt Flag bit 1 = Transmit Buffer 0 has completed transmission of a message and may be reloaded 0 = Transmit Buffer 0 has not completed transmission of a message
bit 1	**RXB1IF:** CAN Receive Buffer 1 Interrupt Flag bit 1 = Receive Buffer 1 has received a new message 0 = Receive Buffer 1 has not received a new message
bit 0	**RXB0IF:** CAN Receive Buffer 0 Interrupt Flag bit 1 = Receive Buffer 0 has received a new message 0 = Receive Buffer 0 has not received a new message

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

24.0 SPECIAL FEATURES OF THE CPU

There are several features intended to maximize system reliability, minimize cost through elimination of external components, provide power-saving operating modes and offer code protection. These are:

- Oscillator Selection
- Reset
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Brown-out Reset (BOR)
- Interrupts
- Watchdog Timer (WDT)
- Sleep
- Code Protection
- ID Locations
- In-Circuit Serial Programming

All PIC18FXX8 devices have a Watchdog Timer which is permanently enabled via the configuration bits or software controlled. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in Reset until the crystal oscillator is stable. The other is the Power-up Timer (PWRT) which provides a fixed delay on power-up only, designed to keep the part in Reset while the power supply stabilizes. With these two timers on-chip, most applications need no external Reset circuitry.

Sleep mode is designed to offer a very Low-Current Power-Down mode. The user can wake-up from Sleep through external Reset, Watchdog Timer wake-up or through an interrupt. Several oscillator options are also made available to allow the part to fit the application. The RC oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits is used to select various options.

24.1 Configuration Bits

The configuration bits can be programmed (read as '0') or left unprogrammed (read as '1'), to select various device configurations. These bits are mapped starting at program memory location 300000h.

The user will note that address 300000h is beyond the user program memory space. In fact, it belongs to the configuration memory space (300000h-3FFFFFh) which can only be accessed using table reads and table writes.

Programming the Configuration registers is done in a manner similar to programming the Flash memory. The EECON1 register $\overline{\text{WR}}$ bit starts a self-timed write to the Configuration register. In normal operation mode, a `TBLWT` instruction, with the TBLPTR pointed to the Configuration register, sets up the address and the data for the Configuration register write. Setting the $\overline{\text{WR}}$ bit starts a long write to the Configuration register. The Configuration registers are written a byte at a time. To write or erase a configuration cell, a `TBLWT` instruction can write a '1' or a '0' into the cell.

TABLE 24-1: CONFIGURATION BITS AND DEVICE IDS

File Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default/ Unprogrammed Value
300001h	CONFIG1H	—	—	$\overline{\text{OSCSEN}}$	—	—	FOSC2	FOSC1	FOSC0	`--1- -111`
300002h	CONFIG2L	—	—	—	—	BORV1	BORV0	BOREN	$\overline{\text{PWRTEN}}$	`---- 1111`
300003h	CONFIG2H	—	—	—	—	WDTPS2	WDTPS1	WDTPS0	WDTEN	`---- 1111`
300006h	CONFIG4L	$\overline{\text{DEBUG}}$	—	—	—	—	LVP	—	STVREN	`1--- -1-1`
300008h	CONFIG5L	—	—	—	—	CP3	CP2	CP1	CP0	`---- 1111`
300009h	CONFIG5H	CPD	CPB	—	—	—	—	—	—	`11-- ----`
30000Ah	CONFIG6L	—	—	—	—	WRT3	WRT2	WRT1	WRT0	`---- 1111`
30000Bh	CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	—	`111- ----`
30000Ch	CONFIG7L	—	—	—	—	EBTR3	EBTR2	EBTR1	EBTR0	`---- 1111`
30000Dh	CONFIG7H	—	EBTRB	—	—	—	—	—	—	`-1-- ----`
3FFFFEh	DEVID1	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0	**(1)**
3FFFFFh	DEVID2	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3	`0000 1000`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, `q` = value depends on condition. Shaded cells are unimplemented, read as '0'.
Note 1: See Register 24-11 for DEVID1 values.

25.2 Instruction Set

ADDLW **ADD Literal to W**

Syntax:	[*label*] ADDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) + k → W
Status Affected:	N, OV, C, DC, Z
Encoding:	`0000` `1111` `kkkk` `kkkk`
Description:	The contents of W are added to the 8-bit literal 'k' and the result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

<u>Example:</u> `ADDLW 0x15`

Before Instruction
W = 0x10
After Instruction
W = 0x25

ADDWF **ADD W to f**

Syntax:	[*label*] ADDWF f [,d [,a]]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(W) + (f) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0010` `01da` `ffff` `ffff`
Description:	Add W to register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default). If 'a' is '0', the Access Bank will be selected. If 'a' is '1', the BSR is used.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

<u>Example:</u> `ADDWF REG, W`

Before Instruction
W = 0x17
REG = 0xC2
After Instruction
W = 0xD9
REG = 0xC2

BZ — Branch if Zero

Syntax:	[*label*] BZ n
Operands:	$-128 \leq n \leq 127$
Operation:	if Zero bit is '`1`' $(PC) + 2 + 2n \rightarrow PC$
Status Affected:	None
Encoding:	`1110` `0000` `nnnn` `nnnn`
Description:	If the Zero bit is '`1`', then the program will branch. The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC + 2 + 2n. This instruction is then a two-cycle instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE    BZ   Jump`

Before Instruction
    PC = address `(HERE)`
After Instruction
    If Zero = 1;
        PC = address `(Jump)`
    If Zero = 0;
        PC = address `(HERE + 2)`

CALL — Subroutine Call

Syntax:	[*label*] CALL k [,s]
Operands:	$0 \leq k \leq 1048575$ $s \in [0,1]$
Operation:	$(PC) + 4 \rightarrow TOS$, $k \rightarrow PC<20:1>$, if s = `1` $(W) \rightarrow WS$, $(Status) \rightarrow STATUSS$, $(BSR) \rightarrow BSRS$
Status Affected:	None

Encoding:

1st word (k<7:0>)	`1110`	`110s`	k_7`kkk`	`kkkk`$_0$
2nd word(k<19:8>)	`1111`	k_{19}`kkk`	`kkkk`	`kkkk`$_8$

Description:	Subroutine call of entire 2-Mbyte memory range. First, return address (PC + 4) is pushed onto the return stack. If 's' = `1`, the W, Status and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = `0`, no update occurs (default). Then, the 20-bit value 'k' is loaded into PC<20:1>. `CALL` is a two-cycle instruction.
Words:	2
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	Push PC to stack	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `HERE    CALL  THERE,FAST`

Before Instruction
    PC = address `(HERE)`
After Instruction
    PC = address `(THERE)`
    TOS = address `(HERE + 4)`
    WS = W
    BSRS = BSR
    STATUSS= Status

TSTFSZ — Test f, Skip if 0

Syntax:	[*label*] TSTFSZ f [,a]
Operands:	$0 \leq f \leq 255$ $a \in [0,1]$
Operation:	skip if f = `0`
Status Affected:	None

Encoding:

`0110`	`011a`	`ffff`	`ffff`

Description:	If 'f' = `0`, the next instruction fetched during the current instruction execution is discarded and a `NOP` is executed, making this a two-cycle instruction. If 'a' is '`0`', the Access Bank will be selected, overriding the BSR value. If 'a' is '`1`', then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:

```
HERE    TSTFSZ CNT
NZERO     :
ZERO    :
```

Before Instruction
PC = Address `(HERE)`
After Instruction
If CNT = 0x00,
PC = Address `(ZERO)`
If CNT ≠ 0x00,
PC = Address `(NZERO)`

XORLW — Exclusive OR Literal with W

Syntax:	[*label*] XORLW k
Operands:	$0 \leq k \leq 255$
Operation:	(W) .XOR. k $\rightarrow$ W
Status Affected:	N, Z

Encoding:

`0000`	`1010`	`kkkk`	`kkkk`

Description:	The contents of W are XORed with the 8-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `XORLW 0xAF`

Before Instruction
W = 0xB5
After Instruction
W = 0x1A

26.0 DEVELOPMENT SUPPORT

The PICmicro® microcontrollers are supported with a full range of hardware and software development tools:

- Integrated Development Environment
  - MPLAB® IDE Software
- Assemblers/Compilers/Linkers
  - MPASM™ Assembler
  - MPLAB C17 and MPLAB C18 C Compilers
  - MPLINK™ Object Linker/ MPLIB™ Object Librarian
  - MPLAB C30 C Compiler
  - MPLAB ASM30 Assembler/Linker/Library
- Simulators
  - MPLAB SIM Software Simulator
  - MPLAB dsPIC30 Software Simulator
- Emulators
  - MPLAB ICE 2000 In-Circuit Emulator
  - MPLAB ICE 4000 In-Circuit Emulator
- In-Circuit Debugger
  - MPLAB ICD 2
- Device Programmers
  - PRO MATE® II Universal Device Programmer
  - PICSTART® Plus Development Programmer
  - MPLAB PM3 Device Programmer
- Low-Cost Demonstration Boards
  - PICDEM™ 1 Demonstration Board
  - PICDEM.net™ Demonstration Board
  - PICDEM 2 Plus Demonstration Board
  - PICDEM 3 Demonstration Board
  - PICDEM 4 Demonstration Board
  - PICDEM 17 Demonstration Board
  - PICDEM 18R Demonstration Board
  - PICDEM LIN Demonstration Board
  - PICDEM USB Demonstration Board
- Evaluation Kits
  - KEELOQ® Evaluation and Programming Tools
  - PICDEM MSC
  - microID® Developer Kits
  - CAN
  - PowerSmart® Developer Kits
  - Analog

26.1 MPLAB Integrated Development Environment Software

The MPLAB IDE software brings an ease of software development previously unseen in the 8/16-bit micro-controller market. The MPLAB IDE is a Windows® based application that contains:

- An interface to debugging tools
  - simulator
  - programmer (sold separately)
  - emulator (sold separately)
  - in-circuit debugger (sold separately)
- A full-featured editor with color coded context
- A multiple project manager
- Customizable data windows with direct edit of contents
- High-level source code debugging
- Mouse over variable inspection
- Extensive on-line help

The MPLAB IDE allows you to:

- Edit your source files (either assembly or C)
- One touch assemble (or compile) and download to PICmicro emulator and simulator tools (automatically updates all project information)
- Debug using:
  - source files (assembly or C)
  - mixed assembly and C
  - machine code

MPLAB IDE supports multiple debugging tools in a single development paradigm, from the cost effective simulators, through low-cost in-circuit debuggers, to full-featured emulators. This eliminates the learning curve when upgrading to tools with increasing flexibility and power.

26.2 MPASM Assembler

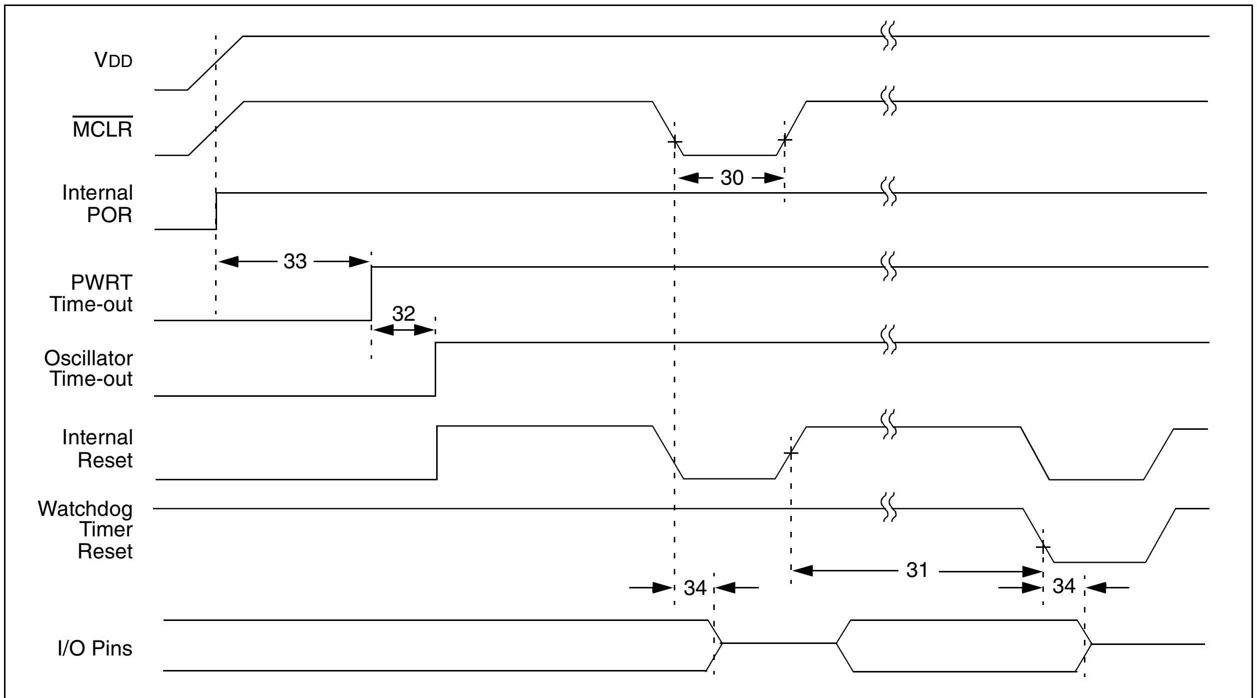
The MPASM assembler is a full-featured, universal macro assembler for all PICmicro MCUs.

The MPASM assembler generates relocatable object files for the MPLINK object linker, Intel® standard HEX files, MAP files to detail memory usage and symbol reference, absolute LST files that contain source lines and generated machine code and COFF files for debugging.

The MPASM assembler features include:

- Integration into MPLAB IDE projects
- User defined macros to streamline assembly code
- Conditional assembly for multi-purpose source files
- Directives that allow complete control over the assembly process

FIGURE 27-8: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER TIMING



Note: Refer to Figure 27-5 for load conditions.

FIGURE 27-9: BROWN-OUT RESET AND LOW-VOLTAGE DETECT TIMING

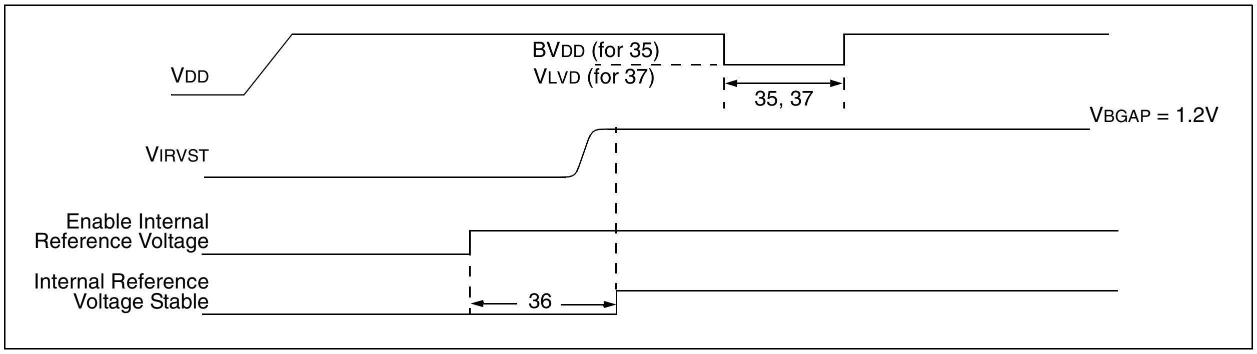

VDD
BVDD (for 35)
VLVD (for 37)
35, 37
VBGAP = 1.2V
VIRVST
Enable Internal Reference Voltage
Internal Reference Voltage Stable
36

TABLE 27-9: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER, POWER-UP TIMER, BROWN-OUT RESET AND LOW-VOLTAGE DETECT REQUIREMENTS

Param No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
30	TmcL	$\overline{\text{MCLR}}$ Pulse Width (low)	2	—	—	μs	
31	TWDT	Watchdog Timer Time-out Period (no prescaler)	7	18	33	ms	
32	TOST	Oscillation Start-up Timer Period	1024 TOSC	—	1024 TOSC	—	TOSC = OSC1 period
33	TPWRT	Power-up Timer Period	28	72	132	ms	
34	TIOZ	I/O High-Impedance from $\overline{\text{MCLR}}$ Low or Watchdog Timer Reset	—	2	—	μs	
35	TBOR	Brown-out Reset Pulse Width	200	—	—	μs	For VDD ≤ BVDD (see D005)
36	TIRVST	Time for Internal Reference Voltage to become stable	—	20	50	μs	
37	TLVD	Low-Voltage Detect Pulse Width	200	—	—	μs	For VDD ≤ VLVD (see D420)