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	64MHz
Connectivity	I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, HLVD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	16KB (8K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.3V ~ 5.5V
Data Converters	A/D 24x12b; D/A 1x5b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SSOP (0.209", 5.30mm Width)
Supplier Device Package	28-SSOP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f24k42-e-ss

11.2 Interrupt Vector Table (IVT)

The interrupt controller supports an Interrupt Vector Table (IVT) that contains the vector address location for each interrupt request source.

The Interrupt Vector Table (IVT) resides in program memory, starting at address location determined by the IVTBASE registers; refer to Registers 11-36 through 11-38 for details. The IVT contains 68 vectors, one for each source of interrupt. Each interrupt vector location contains the starting address of the associated Interrupt Service Routine (ISR).

The MVECEN bit in Configuration Word 2L controls the availability of the vector table.

11.2.1 INTERRUPT VECTOR TABLE BASE ADDRESS (IVTBASE)

The start address of the vector table is user programmable through the IVTBASE registers. The user must ensure the start address is such that it can encompass the entire vector table inside the program memory.

Each vector address is a 16-bit word (or two address locations on PIC18 devices). So for n interrupt sources, there are 2n address locations necessary to hold the table starting from IVTBASE as the first location. So the staring address of IVTBASE should be chosen such that the address range form IVTBASE to (IVTBASE +2n-1) can be encompassed inside the program flash memory.

For example, the PIC18(L)F24/25K42 devices the highest vector number is 81. So IVTBASE should be chosen such that (IVTBASE + 0xA1) is less than the last memory location in program flash memory.

A programmable vector table base address is useful in situations to switch between different sets of vector tables, depending on the application. It can also be used when the application program needs to update the existing vector table (vector address values).

Note:	It is required that the user assign an even address to the IVTBASE register for correct operation.

11.2.2 INTERRUPT VECTOR TABLE CONTENTS

MVECEN = 0

When MVECEN = 0, the address location pointed by the IVTBASE registers has a GOTO instruction for a high priority interrupt. Similarly, the corresponding low priority vector location also has a GOTO instruction, which is executed in case of a low priority interrupt.

MVECEN = 1

When MVECEN = 1, the value in the vector table of each interrupt, points to the address location of the first instruction of the interrupt service routine.

ISR Location = Interrupt Vector Table entry << 2.

11.2.3 INTERRUPT VECTOR TABLE (IVT) ADDRESS CALCULATION

MVECEN = 0

When the MVECEN bit in Configuration Word 2L (Register 5-3) is cleared, the address pointed by IVTBASE registers is used as the high priority interrupt vector address. The low priority interrupt vector address is offset eight instruction words from the address in IVTBASE registers.

For PIC18 devices the IVTBASE registers default to 00 0008h, the high priority interrupt vector address will be 00 0008h and the low priority interrupt vector address will be 00 0018h.

MVECEN = 1

Each interrupt has a unique vector number associated with it as defined in Table 11-2. This vector number is used for calculating the location of the interrupt vector for a particular interrupt source.

Interrupt Vector Address = IVTBASE + (2*Vector Number).

This calculated Interrupt Vector Address value is stored in the IVTAD<20:0> registers when an interrupt is received (Registers 11-39 through 11-41).

User-assigned software priority assigned using the IPRx registers does not affect address calculation and is only used to resolve concurrent interrupts.

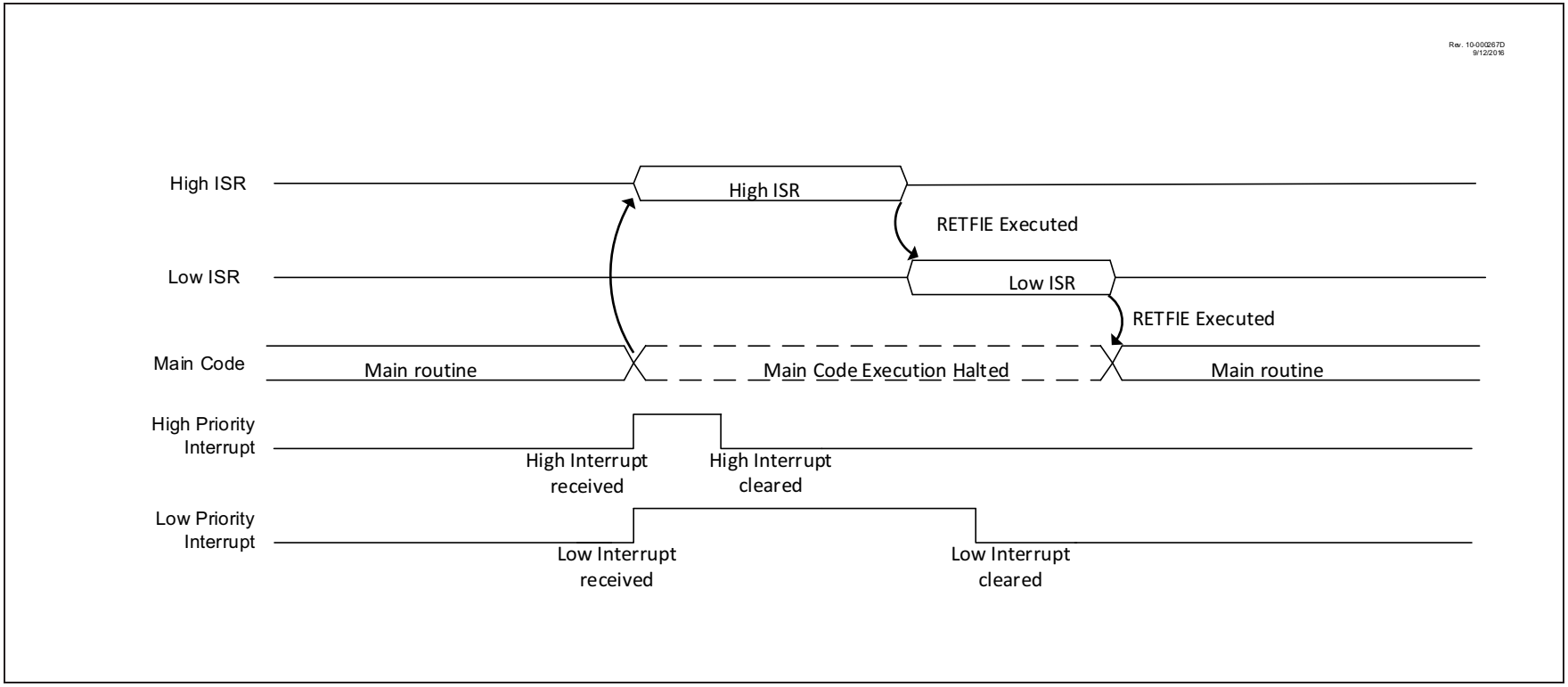
If for any reason the address of the ISR could not be fetched from the vector table, it will cause the system to reset and clear the memory execution violation flag (MEMV bit) in PCON1 register (Register 8-3). This occurs due to any one of the following:

- The entry for the interrupt in the vector table lies outside the executable PFM area (SAF area is non-executable when SAFEN = 1).
- ISR pointed by the vector table lies outside the executable PFM area (SAF area is non-executable when SAFEN = 1).

11.4.4 SIMULTANEOUS LOW AND HIGH PRIORITY INTERRUPTS

When both high and low interrupts are active in the same instruction cycle (i.e., simultaneous interrupt events), both the high and the low priority requests are generated. The high priority ISR is serviced first before servicing the low priority interrupt see Figure 11-5.

FIGURE 11-5: INTERRUPT EXECUTION: SIMULTANEOUS LOW AND HIGH PRIORITY INTERRUPTS



REGISTER 13-2: WDTCON1: WATCHDOG TIMER CONTROL REGISTER 1

U-0	R/W(3)-q/q(1)	R/W(3)-q/q(1)	R/W(3)-q/q(1)	U-0	R/W(4)-q/q(2)	R/W(4)-q/q(2)	R/W(4)-q/q(2)
—	CS<2:0>			—	WINDOW<2:0>		
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	q = Value depends on condition

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

bit 6-4 **CS<2:0>:** Watchdog Timer Clock Select bits

111 = Reserved
•
•
•
011 = Reserved
010 = SOSC
001 = MFINTOSC 31.25 kHz
000 = LFINTOSC 31 kHz

bit 3 **Unimplemented:** Read as '0'

bit 2-0 **WINDOW<2:0>:** Watchdog Timer Window Select bits

WINDOW<2:0>	Window delay Percent of time	Window opening Percent of time
111	N/A	100
110	12.5	87.5
101	25	75
100	37.5	62.5
011	50	50
010	62.5	37.5
001	75	25
000	87.5	12.5

Note 1: If WDTCCS <2:0> in CONFIG3H = 111, the Reset value of CS<2:0> is 000.

2: The Reset value of WINDOW<2:0> is determined by the value of WDTCWS<2:0> in the CONFIG3H register.

3: If WDTCCS<2:0> in CONFIG3H ≠ 111, these bits are read-only.

4: If WDTCWS<2:0> in CONFIG3H ≠ 111, these bits are read-only.

16.2 CRC Functional Overview

The CRC module can be used to detect bit errors in the program memory using the built-in memory scanner or through user input RAM memory. The CRC module can accept up to a 16-bit polynomial with up to a 16-bit seed value. A CRC calculated check value (or checksum) will then be generated into the CRCACC<15:0> registers for user storage. The CRC module uses an XOR shift register implementation to perform the polynomial division required for the CRC calculation.

EXAMPLE 16-1: CRC EXAMPLE

Rev. 10-000206A
1/8/2014

CRC-16-ANSI

$x^{16} + x^{15} + x^2 + 1$ (17 bits)

Standard 16-bit representation = 0x8005

```
CRCXORH = 0b10000000
CRCXORL = 0b0000010-
```
(1)

Data Sequence:
0x55, 0x66, 0x77, 0x88

```
DLEN = 0b0111
PLEN = 0b1111
```

Data entered into the CRC:
SHIFTM = 0:

```
01010101 01100110 01110111 10001000
```

SHIFTM = 1:

```
10101010 01100110 11101110 00010001
```

Check Value (ACCM = 1):

SHIFTM = 0: 0x32D6

```
CRCACCH = 0b00110010
CRCACCL = 0b11010110
```

SHIFTM = 1: 0x6BA2

```
CRCACCH = 0b01101011
CRCACCL = 0b10100010
```

Note 1: Bit 0 is unimplemented. The LSb of any CRC polynomial is always '1' and will always be treated as a '1' by the CRC for calculating the CRC check value. This bit will be read in software as a '0'.

16.5 CRC Check Value

The CRC check value will be located in the CRCACC registers after the CRC calculation has finished. The check value will depend on two mode settings of the CRCCON0 register: ACCM and SHIFTM. When the ACCM bit is set, the CRC module augments the data with a number of zeros equal to the length of the polynomial to align the final check value. When the ACCM bit is not set, the CRC will stop at the end of the data. A number of zeros equal to the length of the polynomial can then be entered into CRCDAT to find the same check value as augmented mode. Alternatively the expected check value can be entered at this point to make the final result equal '0'.

When the CRC check value is computed with the SHIFTM bit set, selecting LSb first, and the ACCM bit is also set then the final value in the CRCACC registers will be reversed such that the LSb will be in the MSb position and vice versa. This is the expected check value in bit reversed form. If you are creating a check value to be appended to a data stream then a bit reversal must be performed on the final value to achieve the correct checksum. You can use the CRC to do this reversal by the following method:

- Save the CRCACC value in user RAM space
- Clear the CRCACC registers
- Clear the CRCXOR registers
- Write the saved CRCACC value to the CRCDAT input.

The properly oriented check value will be in the CRCACC registers as the result.

16.6 CRC Interrupt

The CRC will generate an interrupt when the BUSY bit transitions from 1 to 0. The CRCIF Interrupt Flag is set every time the BUSY bit transitions, regardless of whether or not the CRC interrupt is enabled. The CRCIF bit can only be cleared in software.

16.7 Configuring the CRC

The following steps illustrate how to properly configure the CRC.

1. Determine if the automatic program memory scan will be used with the scanner or manual calculation through the SFR interface and perform the actions specified in **Section 16.4 "CRC Data Sources"**, depending on which decision was made.
2. If desired, seed a starting CRC value into the CRCACCH/L registers.
3. Program the CRCXORH/L registers with the desired generator polynomial.
4. Program the DLEN<3:0> bits of the CRCCON1 register with the length of the data word - 1 (refer to Example 16-1). This determines how many times the shifter will shift into the accumulator for each data word.
5. Program the PLEN<3:0> bits of the CRCCON1 register with the length of the polynomial -2 (refer to Example 16-1).
6. Determine whether shifting in trailing zeros is desired and set the ACCM bit of the CRCCON0 register appropriately.
7. Likewise, determine whether the MSb or LSb should be shifted first and write the SHIFTM bit of the CRCCON0 register appropriately.
8. Write the GO bit of the CRCCON0 register to begin the shifting process.

9a. If manual SFR entry is used, monitor the FULL bit of the CRCCON0 register. When FULL = 0, another word of data can be written to the CRCDATH/L registers, keeping in mind that CRCDATH should be written first if the data has more than eight bits, as the shifter will begin upon the CRCDATL register being written.

9b. If the scanner is used, the scanner will automatically load words into the CRCDATH/L registers as needed, as long as the GO bit is set.

10a. If manual entry is used, monitor the CRCIF (and BUSY bit to determine when the completed CRC calculation can be read from CRCACCH/L registers.

10b. If using the memory scanner, monitor the SCANIF (or the GO bit) for the scanner to finish pushing information into the CRCDAT registers. After the scanner is completed, monitor the BUSY bit to determine that the CRC has been completed and the check value can be read from the CRCACC registers. If both the interrupt flags are set and the BUSY and GO bits are cleared, the completed CRC calculation can be read from the CRCACCH/L registers.

17.8.4 OVERRUN INTERRUPT

When the DMA receives a trigger to start a new message before the current message is completed, then the DMAxORIF Overrun interrupt flag is set.

This condition indicates that the DMA is being requested before its current transaction is finished. This implies that the active DMA may not be able to keep up with the demands from the peripheral module being serviced, which may result in data loss.

The DMAxORIF flag being set does not cause the current DMA transfer to terminate.

The Overrun interrupt is only available for trigger sources that are edge based and not available for sources that are level-based. Therefore a level-based interrupt source does not trigger a DMA overrun error due to the potential latency issues in the system.

An example of an interrupt that could use the overrun interrupt would be a timer overflow (or period match) interrupt. This event only happens every time the timer rolls over and is not dependent on any other system conditions.

An example of an interrupt that does not allow the overrun interrupt would be the UARTTX buffer. The UART will continue to assert the interrupt until the DMA is able to process the MSG. Due to latency issues, the DMA may not be able to service an empty buffer immediately, but the UART continues to assert its transmit interrupt until it is serviced. If overrun was allowed in this case, the overrun would occur almost immediately as the module samples the interrupt sources every instruction cycle.

17.9 DMA Setup and Operation

The following steps illustrate how to configure the DMA for data transfer:

1. Program the appropriate Source and Destination addresses for the transaction into the DMAxSSA and DMAxDSA registers
2. Select the source memory region that is being addressed by DMAxSSA register, using the SMR<1:0> bits.
3. Program the SMODE and DMODE bits to select the addressing mode.
4. Program the Source size DMAxSSZ and Destination size DMAxDSZ registers with the number of bytes to be transferred. It is recommended for proper operation that the size registers be a multiple of each other.
5. If the user desires to disable data transfers once the message has completed, then the SSTP and DSTP bits in DMAxCON0 register need to be set.(see **Section 17.5.3.2 "Source/Destination Stop"**).
6. If using hardware triggers for data transfer, setup the hardware trigger interrupt sources for the starting and aborting DMA transfers (DMAxSIRQ and DMAxAIRQ), and set the corresponding interrupt request enable bits (SIRQEN and AIRQEN).
7. Select the priority level for the DMA (see **Section 3.1 "System Arbitration"**) and lock the priorities (see **Section 3.1.1 "Priority Lock"**)
8. Enable the DMA (DMAxCON1bits. EN = 1)
9. If using software control for data transfer, set the DGO bit, else this bit will be set by the hardware trigger.

Once the DMA is set up, the following flow chart describes the sequence of operation when the DMA uses hardware triggers and utilizes the unused CPU cycles (bubble) for DMA transfers.

20.6 Register Definitions: Interrupt-on-Change Control

REGISTER 20-1: IOCxP: INTERRUPT-ON-CHANGE POSITIVE EDGE REGISTER EXAMPLE

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
IOCxP7	IOCxP6	IOCxP5	IOCxP4	IOCxP3	IOCxP2	IOCxP1	IOCxP0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-0 **IOCxP<7:0>:** Interrupt-on-Change Positive Edge Enable bits
1 = Interrupt-on-Change enabled on the IOCx pin for a positive-going edge. Associated Status bit and interrupt flag will be set upon detecting an edge.
0 = Interrupt-on-Change disabled for the associated pin.

REGISTER 20-2: IOCxN: INTERRUPT-ON-CHANGE NEGATIVE EDGE REGISTER EXAMPLE

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
IOCxN7	IOCxN6	IOCxN5	IOCxN4	IOCxN3	IOCxN2	IOCxN1	IOCxN0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-0 **IOCxN<7:0>:** Interrupt-on-Change Negative Edge Enable bits
1 = Interrupt-on-Change enabled on the IOCx pin for a negative-going edge. Associated Status bit and interrupt flag will be set upon detecting an edge.
0 = Interrupt-on-Change disabled for the associated pin

REGISTER 20-3: IOCxF: INTERRUPT-ON-CHANGE FLAG REGISTER EXAMPLE

R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0
IOCxF7	IOCxF6	IOCxF5	IOCxF4	IOCxF3	IOCxF2	IOCxF1	IOCxF0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	HS - Bit is set in hardware

bit 7-0 **IOCxF<7:0>:** Interrupt-on-Change Flag bits
1 = A enabled change was detected on the associated pin. Set when IOCP[n] = 1 and a positive edge was detected on the IOCn pin, or when IOCN[n] = 1 and a negative edge was detected on the IOCn pin
0 = No change was detected, or the user cleared the detected change

27.6.2 GATED TIMER MODE

Gated Timer mode uses the SMTSIGx input to control whether or not the SMT1TMR will increment. Upon a falling edge of the external signal, the SMT1CPW register will update to the current value of the SMT1TMR. Example waveforms for both repeated and single acquisitions are provided in Figure 27-4 and Figure 27-5.

28.9 Dead-Band Jitter

When the rising and falling edges of the input source are asynchronous to the CWG clock, it creates jitter in the dead-band time delay. The maximum jitter is equal to one CWG clock period. Refer to Equation 28-1 for more details.

EQUATION 28-1: DEAD-BAND DELAY TIME CALCULATION

$$T_{DEAD-BAND\_MIN} = \frac{1}{F_{CWG\ CLOCK}} \bullet DBx<4:0>$$

$$T_{DEAD-BANDMAX} = \frac{1}{F_{CWG\ CLOCK}} \bullet DBx<4:0>+1$$

$$T_{JITTER} = T_{DEAD-BAND\_MAX} - T_{DEAD-BAND\_MIN}$$

$$T_{JITTER} = \frac{1}{F_{CWG\_CLOCK}}$$

$$T_{DEAD-BAND\_MAX} = T_{DEAD-BAND\_MIN} + T_{JITTER}$$

EXAMPLE

$DBR<4:0> = 0x0A = 10$

$F_{CWG\_CLOCK} = 8\ MHz$

$T_{JITTER} = \frac{1}{8MHz} = 125\ ns$

$T_{DEAD-BAND\_MIN} = 125\ ns * 10 = 125\ \mu s$

$T_{DEAD-BAND\_MAX} = 1.25\ \mu s + 0.125 \mu s = 1.37 \mu s$

29.2 CLCx Interrupts

An interrupt will be generated upon a change in the output value of the CLCx when the appropriate interrupt enables are set. A rising edge detector and a falling edge detector are present in each CLC for this purpose.

The CLCxIF bit of the associated PIR5 register will be set when either edge detector is triggered and its associated enable bit is set. The INTP enables rising edge interrupts and the INTN bit enables falling edge interrupts. Both are located in the CLCxCON register.

To fully enable the interrupt, set the following bits:

- CLCxIE bit of the respective PIE register
- INTP bit of the CLCxCON register (for a rising edge detection)
- INTN bit of the CLCxCON register (for a falling edge detection)
- GIE bits of the INTCON0 register

The CLCxIF bit of the respective PIR register, must be cleared in software as part of the interrupt service. If another edge is detected while this flag is being cleared, the flag will still be set at the end of the sequence.

29.3 Output Mirror Copies

Mirror copies of all CON output bits are contained in the CLCxDATA register. Reading this register reads the outputs of all CLCs simultaneously. This prevents any reading skew introduced by testing or reading the OUT bits in the individual CLCxCON registers.

29.4 Effects of a Reset

The CLCxCON register is cleared to zero as the result of a Reset. All other selection and gating values remain unchanged.

29.5 Operation During Sleep

The CLC module operates independently from the system clock and will continue to run during Sleep, provided that the input sources selected remain active.

The HFINTOSC remains active during Sleep when the CLC module is enabled and the HFINTOSC is selected as an input source, regardless of the system clock source selected.

In other words, if the HFINTOSC is simultaneously selected as the system clock and as a CLC input source, when the CLC is enabled, the CPU will go idle during Sleep, but the CLC will continue to operate and the HFINTOSC will remain active.

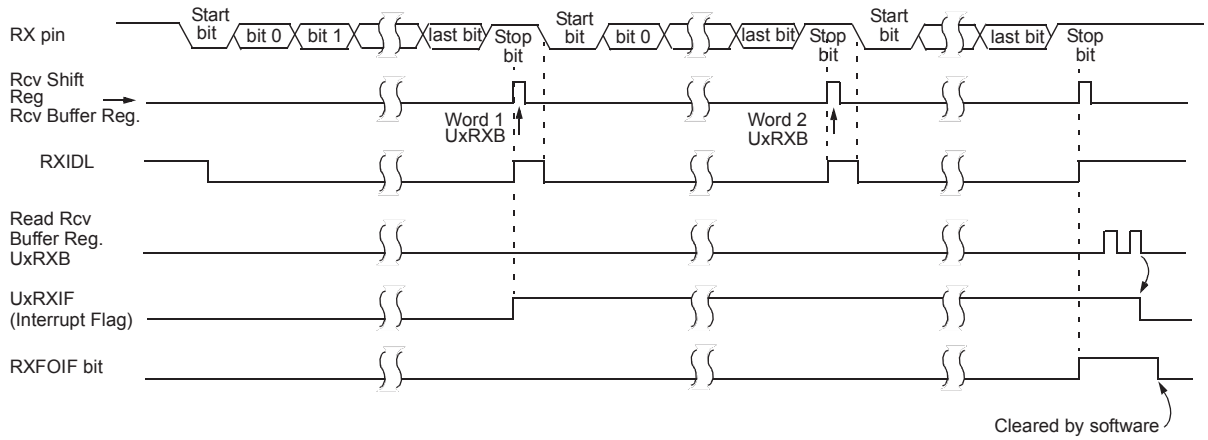
This will have a direct effect on the Sleep mode current.

29.6 CLCx Setup Steps

The following steps should be followed when setting up the CLCx:

- Disable CLCx by clearing the EN bit.
- Select desired inputs using CLCxSEL0 through CLCxSEL3 registers (See Table 29-1).
- Clear any associated ANSEL bits.
- Set all TRIS bits associated with inputs.
- Clear all TRIS bits associated with outputs.
- Enable the chosen inputs through the four gates using CLCxGLS0, CLCxGLS1, CLCxGLS2, and CLCxGLS3 registers.
- Select the gate output polarities with the GyPOL bits of the CLCxPOL register.
- Select the desired logic function with the MODE<2:0> bits of the CLCxCON register.
- Select the desired polarity of the logic output with the POL bit of the CLCxPOL register. (This step may be combined with the previous gate output polarity step).
- If driving a device pin, set the desired pin PPS control register and also clear the TRIS bit corresponding to that output.
- If interrupts are desired, configure the following bits:
  - Set the INTP bit in the CLCxCON register for rising event.
  - Set the INTN bit in the CLCxCON register for falling event.
  - Set the CLCxIE bit of the respective PIE register.
  - Set the GIE bits of the INTCON0 register.
- Enable the CLCx by setting the EN bit of the CLCxCON register.

FIGURE 33-5: ASYNCHRONOUS RECEPTION



Note: This timing diagram shows three words appearing on the RX input. The UxRXB (receive buffer) is not read before the third word is received, causing the RXFOIF (FIFO overrun) bit to be set. STPMD = 0, STP<1:0> = 00.

33.9 Stop Bits

The number of Stop bits is user selectable with the STP bits in the UxCON2 register.The STP bits affect all modes of operation.

Stop bits selections include:

- 1 transmit with receive verify on first
- 1.5 transmit with receive verify on first
- 2 transmit with receive verify on both
- 2 transmit with receive verify on first only

In all modes, except DALI, the transmitter is idle for the number of Stop bit periods between each consecutively transmitted word. In DALI, the Stop bits are generated after the last bit in the transmitted data stream.

The input is checked for the idle level in the middle of the first Stop bit, when receive verify on first is selected, as well as in the middle of the second Stop bit, when verify on both is selected. If any Stop bit verification indicates a non-idle level, the framing error FERIF bit is set for the received word.

33.9.1 DELAYED UXRXIF

When operating in Half-Duplex mode, where the microcontroller needs to reverse the transceiver direction after a reception, it may be more convenient to hold off the UxRXIF interrupt until the end of the Stop bits to avoid line contention. The user selects when the UxRXIF interrupt occurs with the STPMD bit in the UxFIFO register. When STPMD is '1', the UxRXIF occurs at the end of the last Stop bit. When STPMD is '0', UxRXIF occurs when the received byte is stored in the receive FIFO. When STP<1:0> = 10, the store operation is performed in the middle of the second Stop bit, otherwise, it is performed in the middle of the first Stop bit. The FERIF and PERIF interrupts are not delayed with STPMD. Only UxRXIF is delayed when STPMD is set and should be the only indicator for reversing transceiver direction.

33.10 Operation after FIFO overflow

The Receive Shift Register (RSR) can be configured to stop or continue running during a receive FIFO overflow condition. Stopped operation is the Legacy mode.

When the RSR continues to run during an overflow condition, the first word received after clearing the overflow will always be valid.

When the RSR is stopped during an overflow condition, synchronization with the Start bits is lost. Therefore, the first word received after the overflow is cleared may start in the middle of a word.

Operation during overflow is selected with the RUNOVF bit in the UxCON2 register. Setting the RUNOVF bit selects the run during overflow method.

33.11 Receive and Transmit Buffers

The UART uses small buffer areas to transmit and receive data. These are sometimes referred to as FIFOs.

The receiver has a Receive Shift Register (RSR) and two buffer registers. The buffer at the top of the FIFO (earliest byte to enter the FIFO) is by retrieved by reading the UxRXB register.

The transmitter has one Transmit Shift Register (TSR) and one buffer register. Writes to UxTXB go to the transmit buffer then immediately to the TSR, if it is empty. When the TSR is not empty, writes to UxTXB are held then transferred to the TSR when it becomes available.

33.11.1 FIFO STATUS

The UxFIFO register contains several status bits for determining the state of the receive and transmit buffers.

The RXBE bit indicates that the receive FIFO is empty. This bit is essentially the inverse of UxRXIF. The RXBF bit indicates that the receive FIFO is full.

The transmitter has only one buffer register so the status bits are essentially a copy and inverse of the UxTXIF bit. The TXBE bit indicates that the buffer is empty (same as UxTXIF) and the TXBF bit indicates that the buffer is full (UxTXIF inverse). A third transmitter status bit, TXWRE (transmit write error), is set whenever a UxTXB write is performed when the TXBF bit is set. This indicates that the write was unsuccessful.

33.11.2 FIFO RESET

All modes support resetting the receive and transmit buffers.

The receive buffer is flushed and all unread data discarded when the RXBE bit in the UxFIFO register is written to '1'. The MOVWF instruction with the TXBE bit cleared should be used to avoid inadvertently clearing a byte pending in the TSR when UxTXB is empty.

Data written to UxTXB when TXEN is low will be held in the Transmit Shift Register (TSR) then sent when TXEN is set. The transmit buffer and inactive TSR are flushed by setting the TXBE bit in the UxFIFO register. Setting TXBE while a character is actively transmitting from the TSR will complete the transmission without being flushed.

Clearing the ON bit will discard all received data and transmit data pending in the TSR and UxTXB.

33.16 Clock Accuracy with Asynchronous Operation

The factory calibrates the internal oscillator block output (INTOSC). However, the INTOSC frequency may drift as VDD or temperature changes, and this directly affects the asynchronous baud rate. Two methods may be used to adjust the baud rate clock, but both require a reference clock source of some kind.

The first (preferred) method uses the OSCTUNE register to adjust the INTOSC output. Adjusting the value of the OSCTUNE register allows for fine resolution changes to the system clock source. See **Section 6.2.2.3 "Internal Oscillator Frequency Adjustment"** for more information.

The other method adjusts the value of the Baud Rate Generator. This can be done automatically with the Auto-Baud Detect feature (see **Section 33.17.1 "Auto-Baud Detect"**). There may not be fine enough resolution when adjusting the Baud Rate Generator to compensate for a gradual change of the peripheral clock frequency.

33.17 UART Baud Rate Generator (BRG)

The Baud Rate Generator (BRG) is a 16-bit timer that is dedicated to the support of the UART operation.

The UxBRGH, UxBRGL register pair determines the period of the free running baud rate timer. The multiplier of the baud rate period is determined by the BRGS bit in the UxCON0 register.

Table 33-1 contains the formulas for determining the baud rate. Example 33-1 provides a sample calculation for determining the baud rate and baud rate error.

The high baud rate range (BRGS = 1) is intended to extend the baud rate range up to a faster rate when the desired baud rate is not possible otherwise. Using the normal baud rate range (BRGS = 0) is recommended when the desired baud rate is achievable with either range.

Writing a new value to the UxBRGH, UxBRGL register pair causes the BRG timer to be reset (or cleared). This ensures that the BRG does not wait for a timer overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receive error or data loss may result. To avoid this problem, check the status of the RXIDL bit to make sure that the receive operation is idle before changing the system clock.

EXAMPLE 33-1: CALCULATING BAUD RATE ERROR

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, BRGS = 0:

$$Desired\ Baud\ Rate = \frac{FOSC}{16([UxBRG] + 1)}$$

$$X = \frac{\frac{FOSC}{Desired\ Baud\ Rate}}{16} - 1$$

$$= \frac{\frac{16000000}{9600}}{16} - 1$$

$$= [103.17] = 103$$

$$Calculated\ Baud\ Rate = \frac{16000000}{16(103 + 1)}$$

$$= 9615$$

$$Error = \frac{Calc.\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate}$$

$$= \frac{(9615 - 9600)}{9600} = 0.16\%$$

TABLE 33-1: BAUD RATE FORMULAS

BRGS	BRG/UART Mode	Baud Rate Formula
1	High Rate	FOSC/[4 (n+1)]
0	Normal Rate	FOSC/[16(n+1)]

Legend: n = value of UxBRGH, UxBRGL register pair.

REGISTER 33-14: UxP2H: UART PARAMETER 2 HIGH REGISTER

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0/0
—	—	—	—	—	—	—	P2<8>
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-6 **Unimplemented:** Read as ‘0’

bit 0 **P2<8>:** Most Significant Bit of Parameter 2
DMX mode:
Most Significant bit of first address of receive block
DALI mode:
Most Significant bit of number of half-bit periods of idle time in Forward Frame detection threshold
Other modes:
Not used

REGISTER 33-15: UxP2L: UART PARAMETER 2 LOW REGISTER

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
P2<7:0>							
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-0 **P2<7:0>:** Least Significant Bits of Parameter 2
DMX mode:
Least Significant Byte of first address of receive block
LIN Slave mode:
Number of data bytes to transmit
DALI mode:
Least Significant Byte of number of half-bit periods of idle time in Forward Frame detection threshold
Asynchronous Address mode:
Receiver address
Other modes:
Not used

34.4.3 TRANSFER COUNTER IN SLAVE MODE

In Slave Mode, the transfer counter will still decrement as data is shifted in and out of the SPI module, but it will not control data transfers. In addition, in slave mode, the BMODE bit along with the transfer counter is used to determine when the device should look for Slave Select faults. If BMODE = 0, the SSFLT bit will be set if Slave Select transitions from its active to inactive state during bytes of data, as well as if it transitions before the last bit sent during the final byte (if SPIx-TWIDTH≠0). If BMODE=1, the SSFLT bit will be set if Slave Select transitions from its active to inactive state before the final bit of each individual transfer is completed. Note that SSFLT does not have an associated interrupt, so it should be checked in software. An ideal time to do this is when the End of Slave Select Interrupt (EOSIF) is triggered (see **Section 34.8.3.3 "Start of Slave Select and End of Slave Select Interrupts"**).

34.5 Master mode

In master mode, the device controls the SCK line, and as such, initiates data transfers and determines when any slaves broadcast data onto the SPI bus.

Master mode of this device can be configured in four different modes, configured by the TXR and RXR bits:

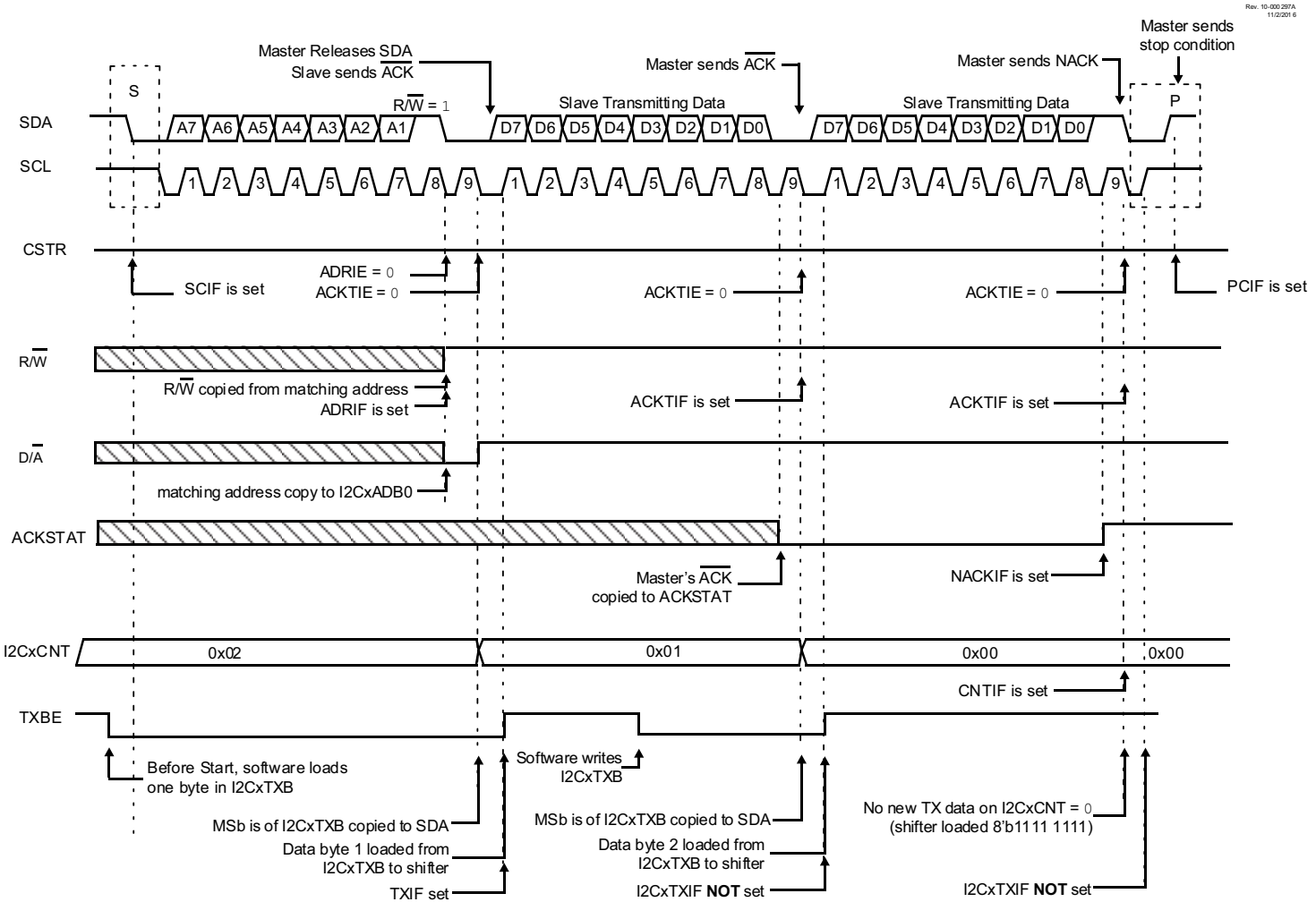
- Full Duplex mode
- Receive Only mode
- Transmit Only mode
- Transfer Off mode

The modes are illustrated in Table 34-1, below:

TABLE 34-1: MASTER MODE TXR/RXR SETTINGS

	TXR = 1	**TXR = 0**
RXR = 1	Full Duplex Mode If BMODE = 1, transfer when RxFIFO is not full and TxFIFO is not empty If BMODE = 0, Transfer when RXFIFO is not full, TXFIFO is not empty, and the Transfer Counter is non-zero	Receive Only mode Transfer when RxFIFO is not full and the Transfer Counter is non-zero Transmitted data is either the top of the FIFO or the most recently received data
RXR = 0	Transmit Only Mode If BMODE = 1, transfer when TxFIFO is not empty If BMODE = 0, Transfer when TXFIFO is not empty and the Transfer Counter is non-zero Received data is not stored	No Transfers

FIGURE 35-10: I^2C SLAVE, 7-BIT ADDRESS, TRANSMISSION (NO CLOCK STRETCHING)



38.6.5 BURST AVERAGE MODE

The Burst Average mode (ADMD = 011) acts the same as the Average mode in most respects. The one way it differs is that it continuously retriggers ADC sampling until the CNT value is greater than or equal to RPT, even if Continuous Sampling mode (see **Section 38.6.8 "Continuous Sampling mode"**) is not enabled. This allows for a threshold comparison on the average of a short burst of ADC samples.

38.6.6 LOW-PASS FILTER MODE

The Low-pass Filter mode (ADMD = 100) acts similarly to the Average mode in how it handles samples (accumulates samples until CNT value greater than or equal to RPT, then triggers threshold comparison), but instead of a simple average, it performs a low-pass filter operation on all of the samples, reducing the effect of high-frequency noise on the average, then performs a threshold comparison on the results. (see Table 38-2 for a more detailed description of the mathematical operation). In this mode, the ADCRS bits determine the cut-off frequency of the low-pass filter (as demonstrated by Table 38-3).

38.6.7 THRESHOLD COMPARISON

At the end of each computation:

- The conversion results are latched and held stable at the end-of-conversion.
- The error is calculated based on a difference calculation which is selected by the ADCALC<2:0> bits in the ADCON3 register. The value can be one of the following calculations (see Register 38-4 for more details):
  - The first derivative of single measurements
  - The CVD result in CVD mode
  - The current result vs. a setpoint
  - The current result vs. the filtered/average result
  - The first derivative of the filtered/average value
  - Filtered/average value vs. a setpoint
- The result of the calculation (ERR) is compared to the upper and lower thresholds, UTH<ADUTHH:ADUTHL> and LTH<ADLTHH:ADLTHL> registers, to set the ADUTHR and ADLTHR flag bits. The threshold logic is selected by ADTMD<2:0> bits in the ADCON3 register. The threshold trigger option can be one of the following:
  - Never interrupt
  - Error is less than lower threshold
  - Error is greater than or equal to lower threshold
  - Error is between thresholds (inclusive)
  - Error is outside of thresholds
  - Error is less than or equal to upper threshold
  - Error is greater than upper threshold
  - Always interrupt regardless of threshold test results
  - If the threshold condition is met, the threshold interrupt flag ADTIF is set.

Note 1: The threshold tests are signed operations.

2: If ADAOV is set, a threshold interrupt is signaled.

38.6.8 CONTINUOUS SAMPLING MODE

Setting the CONT bit in the ADCON0 register automatically retriggers a new conversion cycle after updating the ADACC register. The GO bit remains set and re-triggering occurs automatically.

If ADSOI = 1, a threshold interrupt condition will clear GO and the conversions will stop.

38.6.9 DOUBLE SAMPLE CONVERSION

Double sampling is enabled by setting the ADDSEN bit of the ADCON1 register. When this bit is set, two conversions are required before the module will calculate threshold error (each conversion must still be triggered separately). The first conversion will set the ADMATH bit of the ADSTAT register and update ADACC, but will not calculate ERR or trigger ADTIF. When the second conversion completes, the first value is transferred to PREV (depending on the setting of ADPSIS) and the value of the second conversion is placed into ADRES. Only upon the completion of the second conversion is ERR calculated and ADTIF triggered (depending on the value of ADCALC).

REGISTER 40-2: CMxCON1: COMPARATOR x CONTROL REGISTER 1

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0
—	—	—	—	—	—	INTP	INTN
bit 7							bit 0

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

bit 7-2 Unimplemented: Read as '0'

bit 1 **INTP**: Comparator Interrupt on Positive-Going Edge Enable bit
- 1 = The CxIF interrupt flag will be set upon a positive-going edge of the CxOUT bit
- 0 = No interrupt flag will be set on a positive-going edge of the CxOUT bit

bit 0 **INTN**: Comparator Interrupt on Negative-Going Edge Enable bit
- 1 = The CxIF interrupt flag will be set upon a negative-going edge of the CxOUT bit
- 0 = No interrupt flag will be set on a negative-going edge of the CxOUT bit

REGISTER 40-3: CMxNCH: COMPARATOR x INVERTING CHANNEL SELECT REGISTER

U-0	U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	—	NCH<2:0>		
bit 7							bit 0

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

bit 7-3 **Unimplemented:** Read as '0'

bit 2-0 **NCH<2:0>:** Comparator Inverting Input Channel Select bits
- 111 = Vss
- 110 = FVR_Buffer2
- 101 = NCH not connected
- 100 = NCH not connected
- 011 = CxIN3-
- 010 = CxIN2-
- 001 = CxIN1-
- 000 = CxIN0-

42.0 IN-CIRCUIT SERIAL PROGRAMMING™ (ICSP™)

ICSP™ programming allows customers to manufacture circuit boards with unprogrammed devices. Programming can be done after the assembly process, allowing the device to be programmed with the most recent firmware or a custom firmware. Five pins are needed for ICSP™ programming:

- ICSPCLK
- ICSPDAT
- MCLR/VPP
- VDD
- VSS

In Program/Verify mode the program memory, User IDs and the Configuration Words are programmed through serial communications. The ICSPDAT pin is a bidirectional I/O used for transferring the serial data and the ICSPCLK pin is the clock input. For more information on ICSP™ refer to the "*PIC18(L)F24/25K42 Memory Programming Specification*" (DS40001836).

42.1 High-Voltage Programming Entry Mode

The device is placed into High-Voltage Programming Entry mode by holding the ICSPCLK and ICSPDAT pins low then raising the voltage on MCLR/VPP to VIHH.

42.2 Low-Voltage Programming Entry Mode

The Low-Voltage Programming Entry mode allows the PIC® Flash MCUs to be programmed using VDD only, without high voltage. When the LVP bit of Configuration Words is set to '1', the low-voltage ICSP™ programming entry is enabled. To disable the Low-Voltage ICSP mode, the LVP bit must be programmed to '0'.

Entry into the Low-Voltage Programming Entry mode requires the following steps:

1. MCLR is brought to VIL.
2. A 32-bit key sequence is presented on ICSPDAT, while clocking ICSPCLK.

Once the key sequence is complete, MCLR must be held at VIL for as long as Program/Verify mode is to be maintained.

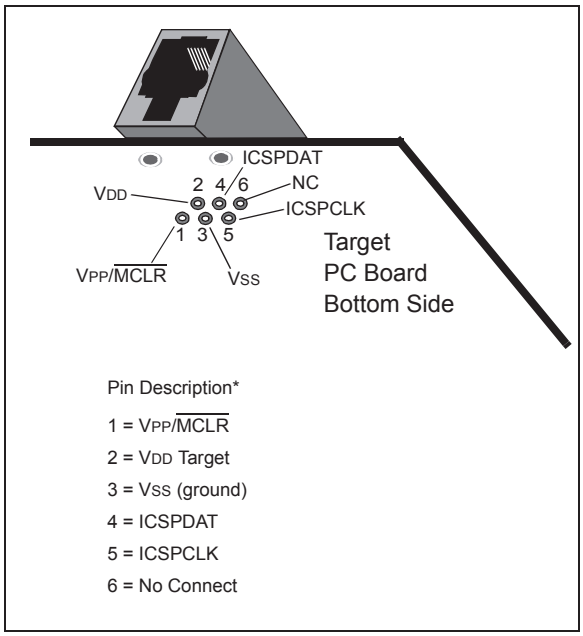
If low-voltage programming is enabled (LVP = 1), the MCLR Reset function is automatically enabled and cannot be disabled. See **Section 8.5 "MCLR"** for more information.

The LVP bit can only be reprogrammed to '0' by using the High-Voltage Programming mode.

42.3 Common Programming Interfaces

Connection to a target device is typically done through an ICSP™ header. A commonly found connector on development tools is the RJ-11 in the 6P6C (6-pin, 6-connector) configuration. See Figure 42-1.

FIGURE 42-1: ICD RJ-11 STYLE CONNECTOR INTERFACE



Another connector often found in use with the PICkit™ programmers is a standard 6-pin header with 0.1 inch spacing. Refer to Figure 42-2.

For additional interface recommendations, refer to your specific device programmer manual prior to PCB design.

It is recommended that isolation devices be used to separate the programming pins from other circuitry. The type of isolation is highly dependent on the specific application and may include devices such as resistors, diodes, or even jumpers. See Figure 42-3 for more information.

SUBWF Subtract W from f

Syntax: SUBWF f {,d {,a}}

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) – (W) → dest

Status Affected: N, OV, C, DC, Z

Encoding:

0101	11da	ffff	ffff

Description: Subtract W from register 'f' (2's complement method). 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank.
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 43.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWF   REG, 1, 0
Before Instruction
    REG =  3
    W   =  2
    C   =  ?
After Instruction
    REG =  1
    W   =  2
    C   =  1    ; result is positive
    Z   =  0
    N   =  0
```

Example 2:
```
SUBWF   REG, 0, 0
Before Instruction
    REG =  2
    W   =  2
    C   =  ?
After Instruction
    REG =  2
    W   =  0
    C   =  1    ; result is zero
    Z   =  1
    N   =  0
```

Example 3:
```
SUBWF   REG, 1, 0
Before Instruction
    REG =  1
    W   =  2
    C   =  ?
After Instruction
    REG =  FFh  ;(2's complement)
    W   =  2
    C   =  0    ; result is negative
    Z   =  0
    N   =  1
```

SUBWFB Subtract W from f with Borrow

Syntax: SUBWFB f {,d {,a}}

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) – (W) – $(\overline{C})$ → dest

Status Affected: N, OV, C, DC, Z

Encoding:

0101	10da	ffff	ffff

Description: Subtract W and the CARRY flag (borrow) from register 'f' (2's complement method). 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank.
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 43.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWFB  REG, 1, 0
Before Instruction
    REG =  19h      (0001 1001)
    W   =  0Dh      (0000 1101)
    C   =  1
After Instruction
    REG =  0Ch      (0000 1100)
    W   =  0Dh      (0000 1101)
    C   =  1
    Z   =  0
    N   =  0        ; result is positive
```

Example 2:
```
SUBWFB  REG, 0, 0
Before Instruction
    REG =  1Bh      (0001 1011)
    W   =  1Ah      (0001 1010)
    C   =  0
After Instruction
    REG =  1Bh      (0001 1011)
    W   =  00h
    C   =  1
    Z   =  1        ; result is zero
    N   =  0
```

Example 3:
```
SUBWFB  REG, 1, 0
Before Instruction
    REG =  03h      (0000 0011)
    W   =  0Eh      (0000 1110)
    C   =  1
After Instruction
    REG =  F5h      (1111 0101)
                    ; [2's comp]
    W   =  0Eh      (0000 1110)
    C   =  0
    Z   =  0
    N   =  1        ; result is negative
```