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	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	32
Program Memory Size	24KB (12K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	1408 x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Through Hole
Package / Case	40-DIP (0.600", 15.24mm)
Supplier Device Package	40-PDIP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lf4539-i-p

PIC18FXX39

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
RB0/INT0 RB0 INT0	21	21	 I/O I	 TTL ST	PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs. Digital I/O. External interrupt 0.
RB1/INT1 RB1 INT1	22	22	 I/O I	 TTL ST	 Digital I/O. External interrupt 1.
RB2/INT2 RB2 INT2	23	23	 I/O I	 TTL ST	 Digital I/O. External interrupt 2.
RB3	24	24	I/O	TTL	Digital I/O.
RB4	25	25	I/O	TTL	Digital I/O. Interrupt-on-change pin.
RB5/PGM RB5 PGM	26	26	 I/O I/O	 TTL ST	 Digital I/O. Interrupt-on-change pin. Low Voltage ICSP programming enable pin.
RB6/PGC RB6 PGC	27	27	 I/O I/O	 TTL ST	 Digital I/O. Interrupt-on-change pin. In-Circuit Debugger and ICSP programming clock pin.
RB7/PGD RB7 PGD	28	28	 I/O I/O	 TTL ST	 Digital I/O. Interrupt-on-change pin. In-Circuit Debugger and ICSP programming data pin.

Legend: TTL = TTL compatible input CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels I = Input
O   = Output P = Power
OD = Open Drain (no P diode to VDD)

Preliminary

6.3 Reading the 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>), clear the CFGS control bit (EECON1<6>), and then set control bit RD (EECON1<0>). The data is available for the very next instruction cycle; therefore, the EEDATA register can be read by the next instruction. EEDATA will hold this value until another read operation, or until it is written to by the user (during a write operation).

EXAMPLE 6-1:   DATA EEPROM READ

```
        MOVLW   DATA_EE_ADDR   ;
        MOVWF   EEADR          ; Data Memory Address to read
        BCF     EECON1, EEPGD  ; Point to DATA memory
        BCF     EECON1, CFGS   ; Access program FLASH or Data EEPROM memory
        BSF     EECON1, RD     ; EEPROM Read
        MOVF    EEDATA, W      ; W = EEDATA
```

6.4 Writing to the Data EEPROM Memory

To write an EEPROM data location, the address must first be written to the EEADR register and the data written to the EEDATA register. Then, the sequence in Example 6-2 must be followed to initiate the write cycle.

The write will not initiate if the above sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. It is strongly recommended that interrupts be disabled during this code segment.

Additionally, the WREN bit in EECON1 must be set to enable writes. This mechanism prevents accidental writes to data EEPROM due to unexpected code exe-cution (i.e., runaway programs). The WREN bit should be kept clear at all times, except when updating the EEPROM. The WREN bit is not cleared by hardware.

After a write sequence has been initiated, EECON1, EEADR and EDATA cannot be modified. The WR bit will be inhibited from being set unless the WREN bit is set. The WREN bit must be set on a previous instruction. Both WR and WREN cannot be set with the same instruction.

At the completion of the write cycle, the WR bit is cleared in hardware and the EEPROM Write Complete Interrupt Flag bit (EEIF) is set. The user may either enable this interrupt, or poll this bit. EEIF must be cleared by software.

EXAMPLE 6-2:   DATA EEPROM WRITE

```
                MOVLW   DATA_EE_ADDR   ;
                MOVWF   EEADR          ; Data Memory Address to read
                MOVLW   DATA_EE_DATA   ;
                MOVWF   EEDATA         ; Data Memory Value to write
                BCF     EECON1, EEPGD  ; Point to DATA memory
                BCF     EECON1, CFGS   ; Access program FLASH or Data EEPROM memory
                BSF     EECON1, WREN   ; Enable writes


                BCF     INTCON, GIE    ; Disable interrupts
Required        MOVLW   55h            ;
Sequence        MOVWF   EECON2         ; Write 55h
                MOVLW   AAh            ;
                MOVWF   EECON2         ; Write AAh
                BSF     EECON1, WR     ; Set WR bit to begin write
                BSF     INTCON, GIE    ; Enable interrupts


                .                      ; user code execution
                .
                .
                BCF     EECON1, WREN   ; Disable writes on write complete (EEIF set)
```

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**

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

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

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 pairs Most Significant bit (MSb) is tested and the appropriate subtractions are done.

EQUATION 7-2: **16 x 16 SIGNED MULTIPLICATION ALGORITHM**

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

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

Preliminary

8.4 IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Priority registers (IPR1, IPR2). The operation of the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

For PIC18FXX39 devices, the Motor Control kernel requires that the Timer2 to PR2 match interrupt be the only high priority interrupt. Failure to do this may result in unpredictable operation of the kernel or the entire microcontroller.

In practical terms, this means:

- Interrupt priority levels are enabled (IPEN = 1);
- High priority interrupts are enabled (INTCON<7> = 1);
- Timer2 interrupt is enabled and set as high priority (PIE1<1> and IPR<1> = 1); and
- all other interrupts are disabled (INTCON or PIR bits = 0), or set as low priority (IPR bits = 0).

> **Note:** Configuring the interrupts is automatically done by the API method `void ProMPT_Init(PWMfrequency)`. It is the user's responsibility to make certain that this method is called at the very beginning of the application.

REGISTER 8-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	U-1	R/W-1	R/W-1
PSPIP[1,2]	ADIP[2]	RCIP[2]	TXIP[2]	SSPIP[2]	—	TMR2IP[3]	TMR1IP[2]
bit 7							bit 0

bit 7 **PSPIP[1,2]:** Parallel Slave Port Read/Write Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **ADIP[2]:** A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP[2]:** USART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP[2]:** USART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP[2]:** Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **Unimplemented:** Read as '1'

bit 1 **TMR2IP[3]:** TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP[2]:** TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: This bit is reserved on PIC18F2X39 devices.
   2: Maintain this bit cleared (= 0).
   3: This bit is reserved for use by the ProMPT kernel; always maintain this bit set (= 1).

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

PIC18FXX39

FIGURE 9-2: **BLOCK DIAGRAM OF RA4/T0CKI PIN**

Note 1:   I/O pin has protection diode to VSS only.

FIGURE 9-3: **BLOCK DIAGRAM OF RA6 PIN**

Note 1:   I/O pins have protection diodes to VDD and VSS.

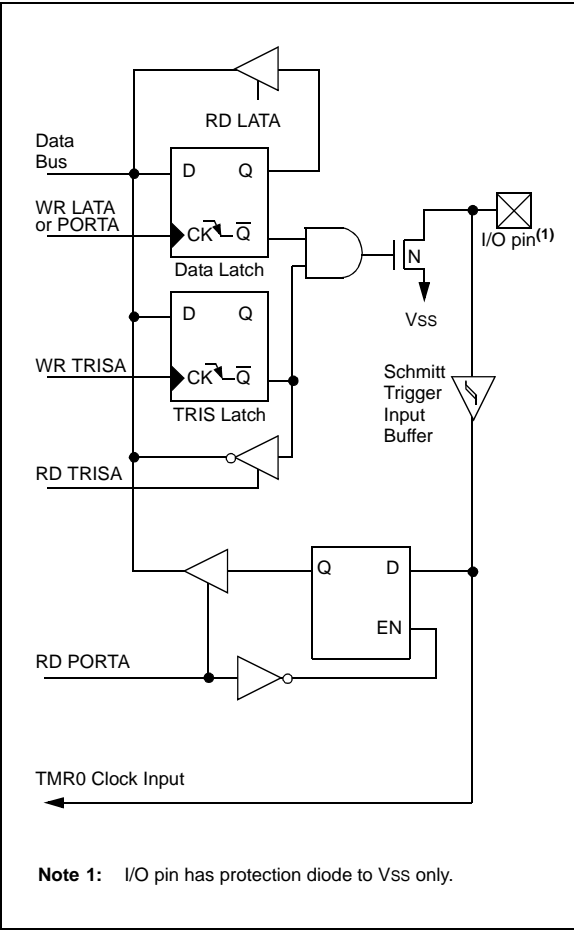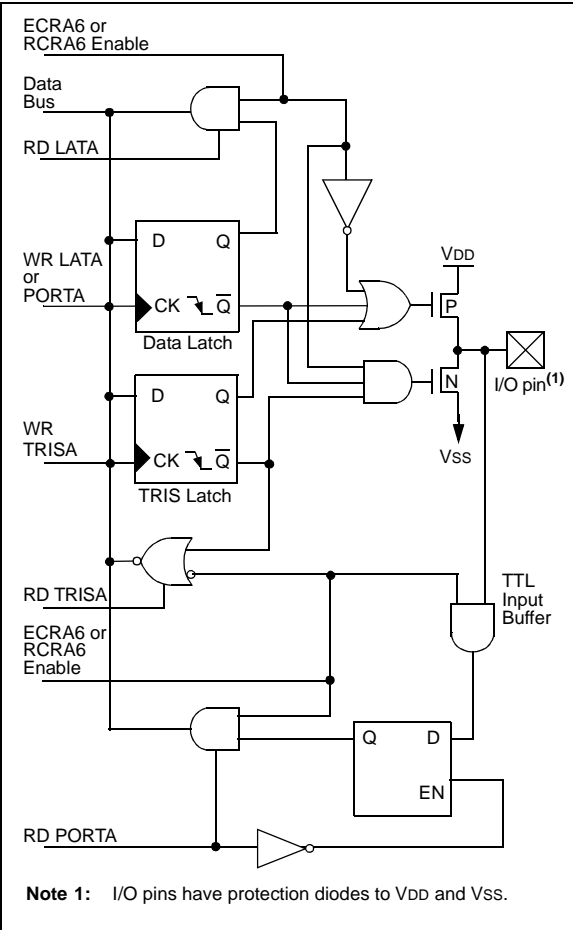**Preliminary**

16.4.4 CLOCK STRETCHING

Both 7- and 10-bit Slave modes implement automatic clock stretching during a transmit sequence.

The SEN bit (SSPCON2<0>) allows clock stretching to be enabled during receives. Setting SEN will cause the SCL pin to be held low at the end of each data receive sequence.

16.4.4.1 Clock Stretching for 7-bit Slave Receive Mode (SEN = 1)

In 7-bit Slave Receive mode, on the falling edge of the ninth clock at the end of the $\overline{\text{ACK}}$ sequence, if the BF bit is set, the CKP bit in the SSPCON1 register is automatically cleared, forcing the SCL output to be held low. The CKP being cleared to '0' will assert the SCL line low. The CKP bit must be set in the user's ISR before reception is allowed to continue. By holding the SCL line low, the user has time to service the ISR and read the contents of the SSPBUF before the master device can initiate another receive sequence. This will prevent buffer overruns from occurring (see Figure 16-13).

> **Note 1:** If the user reads the contents of the SSPBUF before the falling edge of the ninth clock, thus clearing the BF bit, the CKP bit will not be cleared and clock stretching will not occur.
>
> **2:** The CKP bit can be set in software, regardless of the state of the BF bit. The user should be careful to clear the BF bit in the ISR before the next receive sequence, in order to prevent an overflow condition.

16.4.4.2 Clock Stretching for 10-bit Slave Receive Mode (SEN = 1)

In 10-bit Slave Receive mode, during the address sequence, clock stretching automatically takes place but CKP is not cleared. During this time, if the UA bit is set after the ninth clock, clock stretching is initiated. The UA bit is set after receiving the upper byte of the 10-bit address, and following the receive of the second byte of the 10-bit address with the R/$\overline{\text{W}}$ bit cleared to '0'. The release of the clock line occurs upon updating SSPADD. Clock stretching will occur on each data receive sequence, as described in 7-bit mode.

> **Note:** If the user polls the UA bit and clears it by updating the SSPADD register before the falling edge of the ninth clock occurs, and if the user hasn't cleared the BF bit by reading the SSPBUF register before that time, then the CKP bit will still NOT be asserted low. Clock stretching on the basis of the state of the BF bit only occurs during a data sequence, not an address sequence.

16.4.4.3 Clock Stretching for 7-bit Slave Transmit Mode

7-bit Slave Transmit mode implements clock stretching by clearing the CKP bit after the falling edge of the ninth clock, if the BF bit is clear. This occurs, regardless of the state of the SEN bit.

The user's ISR must set the CKP bit before transmission is allowed to continue. By holding the SCL line low, the user has time to service the ISR and load the contents of the SSPBUF before the master device can initiate another transmit sequence (see Figure 16-9).

> **Note 1:** If the user loads the contents of SSPBUF, setting the BF bit before the falling edge of the ninth clock, the CKP bit will not be cleared and clock stretching will not occur.
>
> **2:** The CKP bit can be set in software, regardless of the state of the BF bit.

16.4.4.4 Clock Stretching for 10-bit Slave Transmit Mode

In 10-bit Slave Transmit mode, clock stretching is controlled during the first two address sequences by the state of the UA bit, just as it is in 10-bit Slave Receive mode. The first two addresses are followed by a third address sequence, which contains the high order bits of the 10-bit address and the R/$\overline{\text{W}}$ bit set to '1'. After the third address sequence is performed, the UA bit is not set, the module is now configured in Transmit mode, and clock stretching is controlled by the BF flag, as in 7-bit Slave Transmit mode (see Figure 16-11).

Preliminary © 2002-2013 Microchip Technology Inc.

16.4.6   MASTER MODE

Master mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON1 and by setting the SSPEN bit. In Master mode, the SCL and SDA lines are manipulated by the MSSP hardware.

Master mode of operation is supported by interrupt generation on the detection of the START and STOP conditions. The STOP (P) and START (S) bits are cleared from a RESET or when the MSSP module is disabled. Control of the I^{2}C bus may be taken when the P bit is set or the bus is IDLE, with both the S and P bits clear.

In Firmware Controlled Master mode, user code conducts all I^{2}C bus operations based on START and STOP bit conditions.

Once Master mode is enabled, the user has six options.

1. Assert a START condition on SDA and SCL.
2. Assert a Repeated START condition on SDA and SCL.
3. Write to the SSPBUF register initiating transmission of data/address.
4. Configure the I^{2}C port to receive data.
5. Generate an Acknowledge condition at the end of a received byte of data.
6. Generate a STOP condition on SDA and SCL.

> **Note:**   The MSSP Module, when configured in I^{2}C Master mode, does not allow queueing of events. For instance, the user is not allowed to initiate a START condition and immediately write the SSPBUF register to initiate transmission before the START condition is complete. In this case, the SSPBUF will not be written to and the WCOL bit will be set, indicating that a write to the SSPBUF did not occur.

The following events will cause SSP interrupt flag bit, SSPIF, to be set (SSP interrupt if enabled):

• START condition
• STOP condition
• Data transfer byte transmitted/received
• Acknowledge Transmit
• Repeated START

FIGURE 16-16:   MSSP BLOCK DIAGRAM (I^{2}C MASTER MODE)
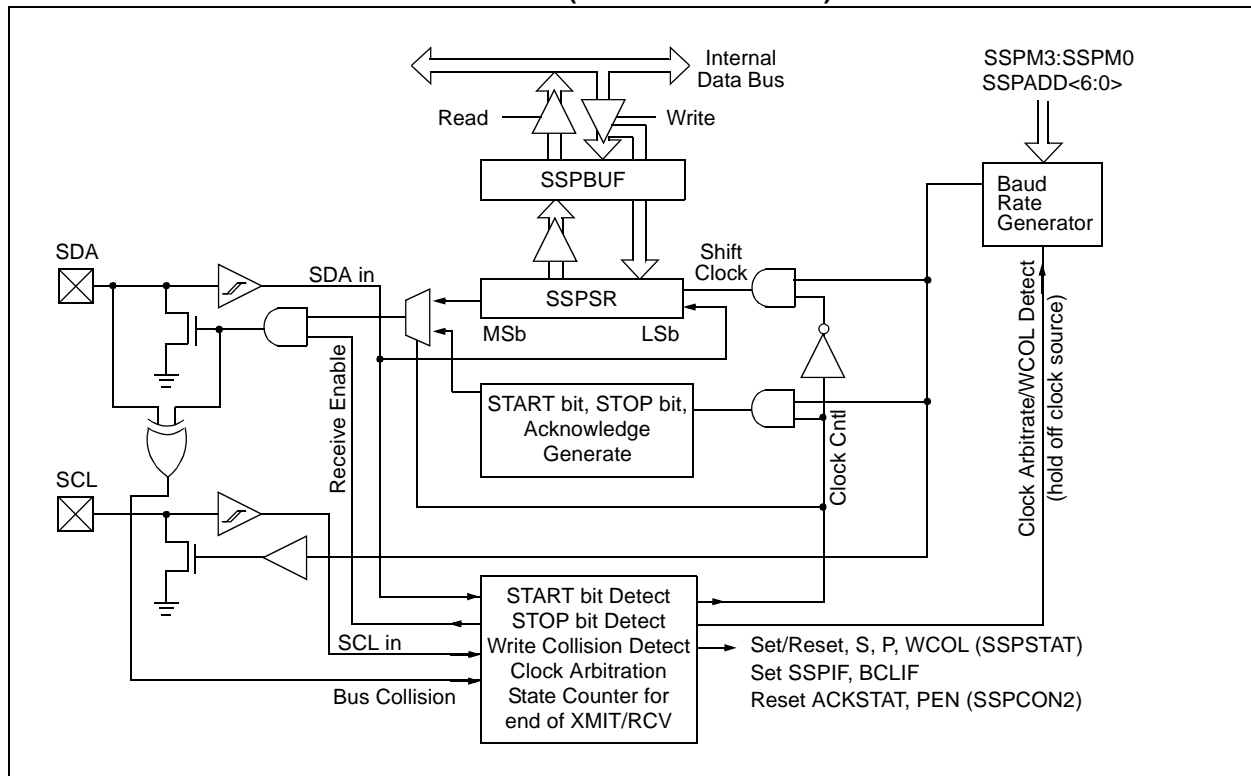
16.4.6.1   I²C Master Mode Operation

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 I²C 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/Write (R/$\overline{W}$) bit. In this case, the R/$\overline{W}$ 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/$\overline{W}$ bit. In this case, the R/$\overline{W}$ bit will be logic '1'. Thus, the first byte transmitted is a 7-bit slave address followed by a '1' to indicate the 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 I²C operation. See Section 16.4.7 ("Baud Rate Generator"), for more detail.

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.

16.4.9 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 (T$_{BRG}$). When the baud rate generator times out, if SDA is sampled high, the SCL pin will be de-asserted (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 T$_{BRG}$. This action is then followed by assertion of the SDA pin (SDA = 0) for one T$_{BRG}$, 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).

16.4.9.1 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 16-20: REPEAT START CONDITION WAVEFORM
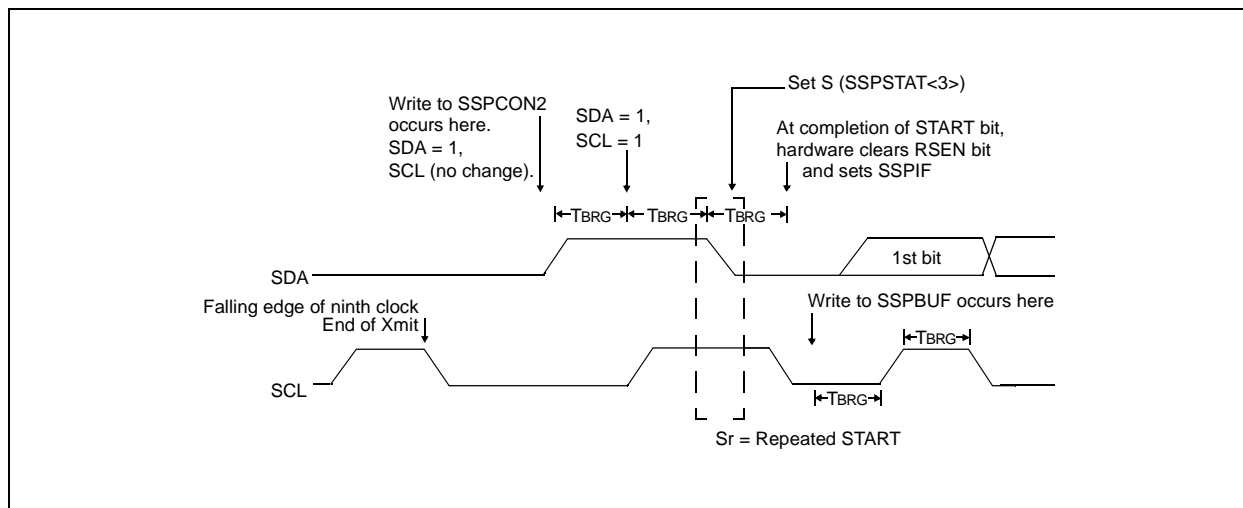
16.4.17.3 Bus Collision During a STOP Condition

Bus collision occurs during a STOP condition if:

a) After the SDA pin has been de-asserted and allowed to float high, SDA is sampled low after the BRG has timed out.

b) After the SCL pin is de-asserted, SCL is sampled low before SDA goes high.

The STOP condition begins with SDA asserted low. When SDA is sampled low, the SCL pin is allowed to float. When the pin is sampled high (clock arbitration), the baud rate generator is loaded with SSPADD<6:0> and counts down to '0'. After the BRG times out, SDA is sampled. If SDA is sampled low, a bus collision has occurred. This is due to another master attempting to drive a data '0' (Figure 16-31). If the SCL pin is sampled low before SDA is allowed to float high, a bus collision occurs. This is another case of another master attempting to drive a data '0' (Figure 16-32).

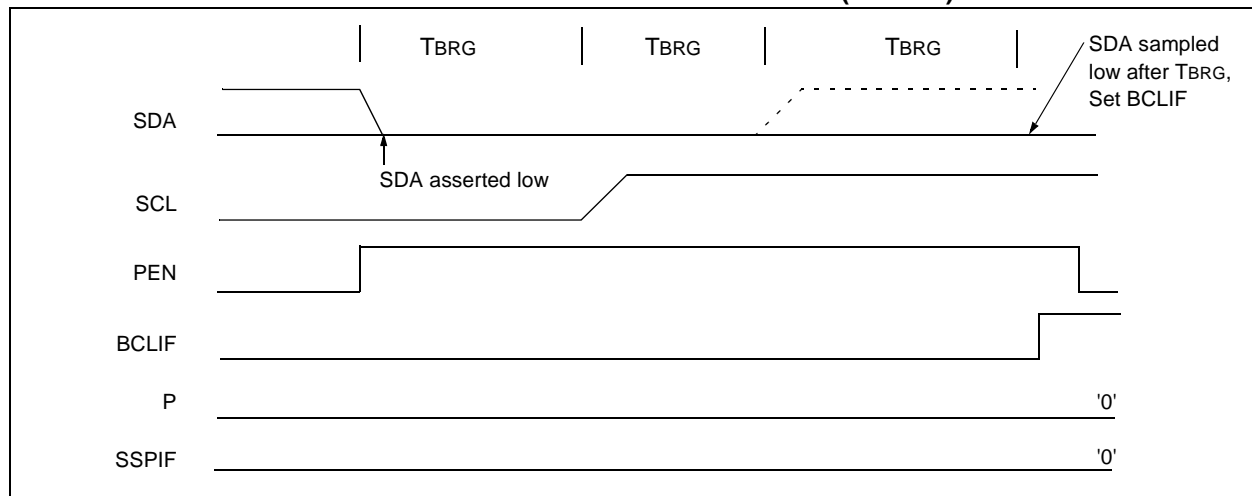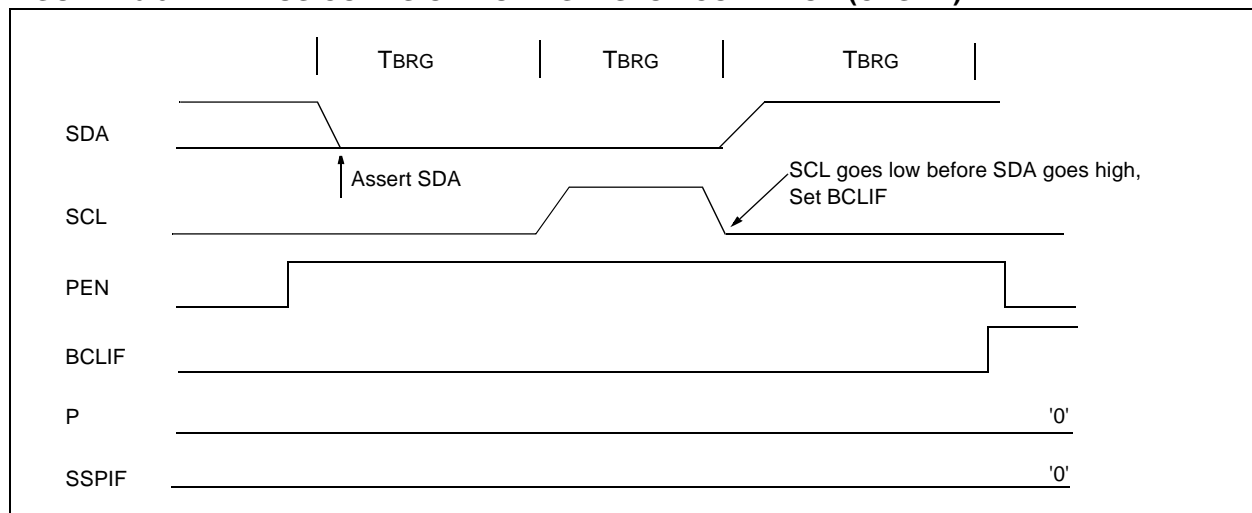**FIGURE 16-31:   BUS COLLISION DURING A STOP CONDITION (CASE 1)**

FIGURE 16-32:   BUS COLLISION DURING A STOP CONDITION (CASE 2)

17.1 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode, bit BRGH is ignored. Table 17-1 shows the formula for computation of the baud rate for different USART modes, which only apply in Master mode (internal clock).

Given the desired baud rate and FOSC, the nearest integer value for the SPBRG register can be calculated using the formula in Table 17-1. From this, the error in baud rate can be determined.

Example 17-1 shows the calculation of the baud rate error for the following conditions:

- FOSC = 16 MHz
- Desired Baud Rate = 9600
- BRGH = 0
- SYNC = 0

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the FOSC/(16(X + 1)) equation can reduce the baud rate error in some cases.

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

17.1.1 SAMPLING

The data on the RC7/RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin.

EXAMPLE 17-1: CALCULATING BAUD RATE ERROR

Desired Baud Rate	=	FOSC / (64 (X + 1))
Solving for X:		
X	=	((FOSC / Desired Baud Rate) / 64) – 1
X	=	((16000000 / 9600) / 64) – 1
X	=	[25.042] = 25
Calculated Baud Rate	=	16000000 / (64 (25 + 1))
	=	9615
Error	=	(Calculated Baud Rate – Desired Baud Rate) / Desired Baud Rate
	=	(9615 – 9600) / 9600
	=	0.16%

TABLE 17-1: BAUD RATE FORMULA

| SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64(X+1))	Baud Rate = FOSC/(16(X+1))
1	(Synchronous) Baud Rate = FOSC/(4(X+1))	N/A

Legend: X = value in SPBRG (0 to 255)

TABLE 17-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

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
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used by the BRG.

FIGURE 19-2: **LOW VOLTAGE DETECT (LVD) BLOCK DIAGRAM**
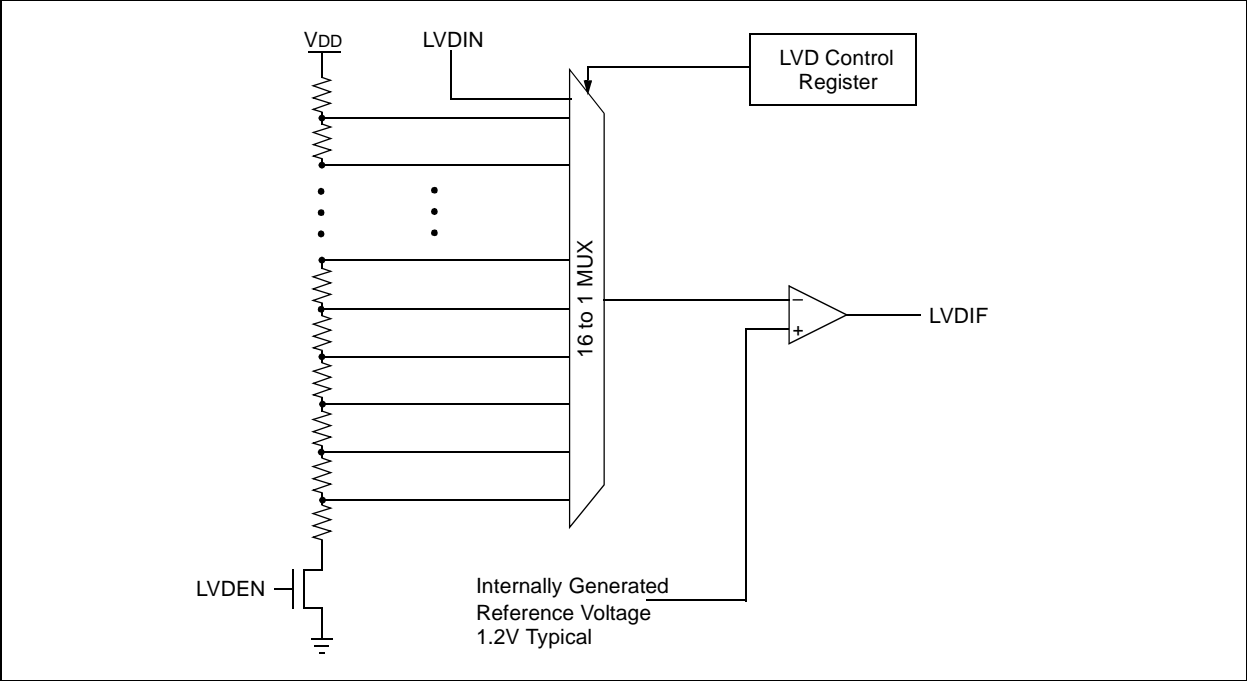
The LVD module has an additional feature that allows the user to supply the trip voltage to the module from an external source. This mode is enabled when bits LVDL3:LVDL0 are set to '1111'. In this state, the comparator input is multiplexed from the external input pin, LVDIN (Figure 19-3). This gives users flexibility, because it allows them to configure the Low Voltage Detect interrupt to occur at any voltage in the valid operating range.

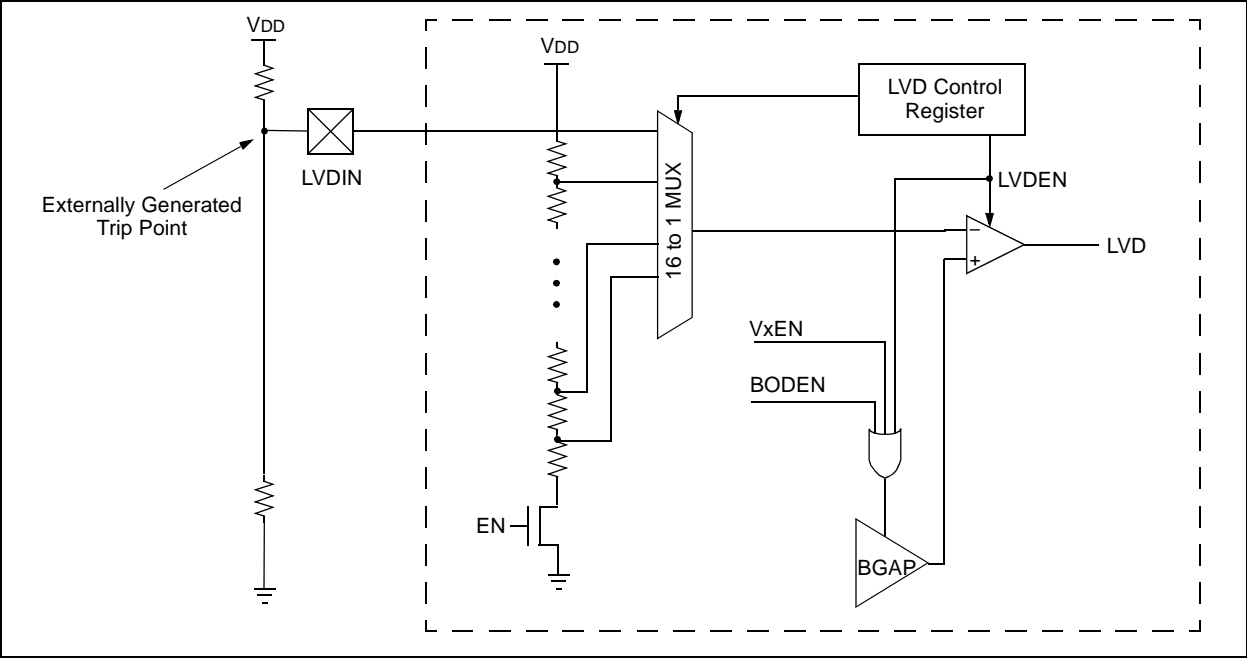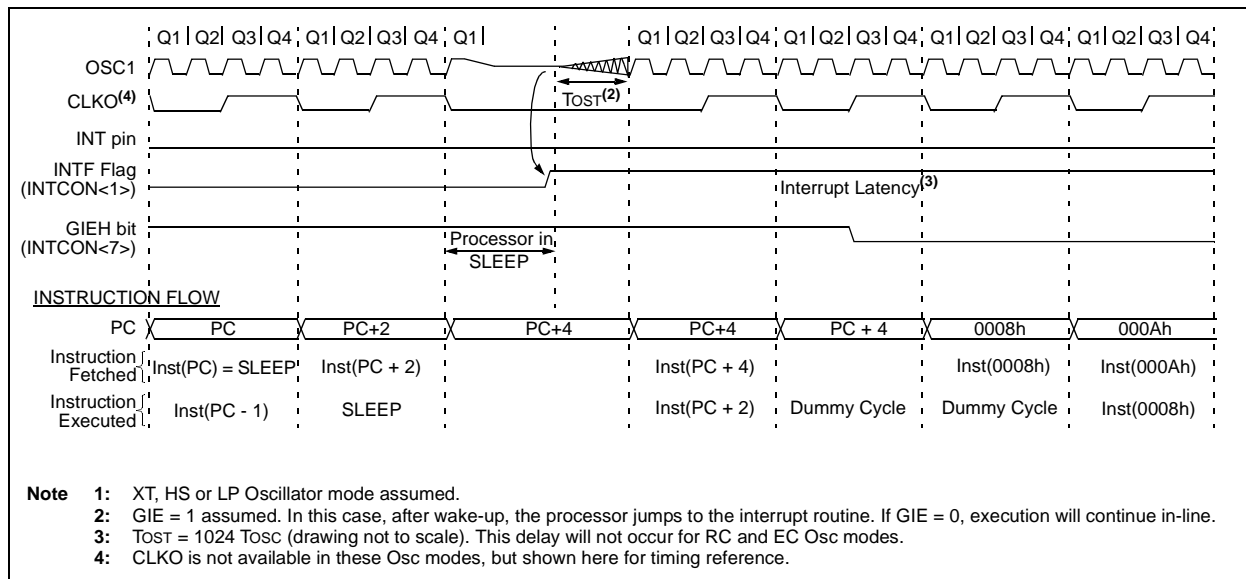**FIGURE 19-3:** **LOW VOLTAGE DETECT (LVD) WITH EXTERNAL INPUT BLOCK DIAGRAM**

PIC18FXX39

FIGURE 20-2: **WAKE-UP FROM SLEEP THROUGH INTERRUPT[1,2]**

Note 1: XT, HS or LP Oscillator mode assumed.

2: GIE = 1 assumed. In this case, after wake-up, the processor jumps to the interrupt routine. If GIE = 0, execution will continue in-line.

3: T_{OST} = 1024 T_{OSC} (drawing not to scale). This delay will not occur for RC and EC Osc modes.

4: CLKO is not available in these Osc modes, but shown here for timing reference.

20.4 Program Verification and Code Protection

The overall structure of the code protection on the PIC18 FLASH devices differs significantly from other PIC devices. The user program memory is divided on binary boundaries into individual blocks, each of which has three separate code protection bits associated with it:

- Code Protect bit (CPn)
- Write Protect bit (WRTn)
- External Block Table Read bit (EBTRn)

The code protection bits are located in Configuration Registers 5L through 7H. Their locations within the registers are summarized in Table 20-3.

In the PIC18FXX39 family, program memory is divided into segments of 8 Kbytes. The first block in turn divided into a boot block of 512 bytes and a separately protected remainder (Block 0) of 7.5 Kbytes. This means for PIC18FXX39 devices, that there may be up to five blocks, depending on the program memory size. The organization of the blocks and their associated code protection bits are shown in Figure 20-3.

For PIC18FX439 devices, program memory is divided into three blocks: a boot block, Block 0 (7.5 Kbytes) and Block 1 (8 Kbytes). Block 1 is further divided in half; the upper portion above 3000h is reserved, and unavailable to user applications. The entire block can be protected as a whole by bits CP1, WRT1 and EBTR1. By default, Block 1 is not code protected.

For PIC18FX539 devices, program memory is divided into five blocks: the boot block, Block 0 (7.5 Kbytes), and Blocks 1 through 3 (8 Kbytes). Code protection is implemented for the boot block and Blocks 0 through 2. There is no provision for code protection for Block 3.

> **Note:** The reserved segments of the program memory space are used by the Motor Control kernel. For the kernel to function properly, this area must not be write protected. If users are developing applications that require code protection for PIC18FX439 devices, they should restrict program code (or at least those sections requiring protection) to below the 1FFFh memory boundary.

CPFSGT	Compare f with W, skip if f > W
Syntax:	[*label*] CPFSGT f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(f) − (W), skip if (f) > (W) (unsigned comparison)
Status Affected:	None

Encoding:

0110	010a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of the W by performing an unsigned subtraction.
If the contents of 'f' are greater than the contents of WREG, then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 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        CPFSGT REG, 0
NGREATER    :
GREATER     :
```

Before Instruction

PC	=	Address (HERE)
W	=	?

After Instruction

If REG	>	W;
PC	=	Address (GREATER)
If REG	≤	W;
PC	=	Address (NGREATER)

CPFSLT	Compare f with W, skip if f < W
Syntax:	[*label*] CPFSLT f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(f) − (W), skip if (f) < (W) (unsigned comparison)
Status Affected:	None

Encoding:

0110	000a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of W by performing an unsigned subtraction.
If the contents of 'f' are less than the contents of W, then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR will not be overridden (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    CPFSLT REG, 1
NLESS   :
LESS    :
```

Before Instruction

PC	=	Address (HERE)
W	=	?

After Instruction

If REG	<	W;
PC	=	Address (LESS)
If REG	≥	W;
PC	=	Address (NLESS)

PIC18FXX39

GOTO **Unconditional Branch**

Syntax: [*label*] GOTO k

Operands: $0 \leq k \leq 1048575$

Operation: $k \rightarrow PC<20:1>$

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	1111	k_7kkk	$kkkk_0$
2nd word(k<19:8>)	1111	$k_{19}kkk$	kkkk	$kkkk_8$

Description: GOTO allows an unconditional branch anywhere within entire 2 Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

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

Example: GOTO THERE

After Instruction
PC = Address (THERE)

INCF **Increment f**

Syntax: [*label*] INCF f [,d [,a]

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

Operation: $(f) + 1 \rightarrow dest$

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

Encoding:

0010	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: INCF CNT, 1, 0

Before Instruction
CNT = 0xFF
Z = 0
C = ?
DC = ?
After Instruction
CNT = 0x00
Z = 1
C = 1
DC = 1

MOVFF	Move f to f

Syntax: [*label*] MOVFF f_s,f_d

Operands: $0 \leq f_s \leq 4095$
$0 \leq f_d \leq 4095$

Operation: $(f_s) \rightarrow f_d$

Status Affected: None

Encoding:
1st word (source)

1100	ffff	ffff	ffff$_s$

2nd word (destin.)

1111	ffff	ffff	ffff$_d$

Description: The contents of source register 'f_s' are moved to destination register 'f_d'. Location of source 'f_s' can be anywhere in the 4096 byte data space (000h to FFFh), and location of destination 'f_d' can also be anywhere from 000h to FFFh.
Either source or destination can be W (a useful special situation).
MOVFF is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

The MOVFF instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.

Note: The MOVFF instruction should not be used to modify interrupt settings while any interrupt is enabled. See Section 8.0 for more information.

Words: 2

Cycles: 2 (3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f' (src)	Process Data	No operation
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: MOVFF REG1, REG2

Before Instruction
  REG1 = 0x33
  REG2 = 0x11
After Instruction
  REG1 = 0x33,
  REG2 = 0x33

MOVLB	Move literal to low nibble in BSR

Syntax: [*label*] MOVLB k

Operands: $0 \leq k \leq 255$

Operation: $k \rightarrow BSR$

Status Affected: None

Encoding:

0000	0001	kkkk	kkkk

Description: The 8-bit literal 'k' is loaded into the Bank Select Register (BSR).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: MOVLB 5

Before Instruction
  BSR register = 0x02
After Instruction
  BSR register = 0x05

PIC18FXX39

RLNCF	Rotate Left f (no carry)
Syntax:	[*label*] RLNCF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f<n>) \to dest<n+1>$, $(f<7>) \to dest<0>$
Status Affected:	N, Z
Encoding:	0100 | 01da | ffff | ffff
Description:	The contents of register 'f' are rotated one bit to the left. If 'd' is 0, the result is placed 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, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
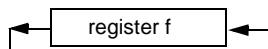
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: RLNCF   REG, 1, 0

Before Instruction
REG   =   1010 1011
After Instruction
REG   =   0101 0111

RRCF	Rotate Right f through Carry
Syntax:	[*label*] RRCF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f<n>) \to dest<n-1>$, $(f<0>) \to C$, $(C) \to dest<7>$
Status Affected:	C, N, Z
Encoding:	0011 | 00da | ffff | ffff
Description:	The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). 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).
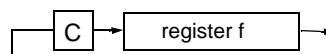
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: RRCF   REG, 0, 0

Before Instruction
REG   =   1110 0110
C   =   0
After Instruction
REG   =   1110 0110
W   =   0111 0011
C   =   0

Preliminary

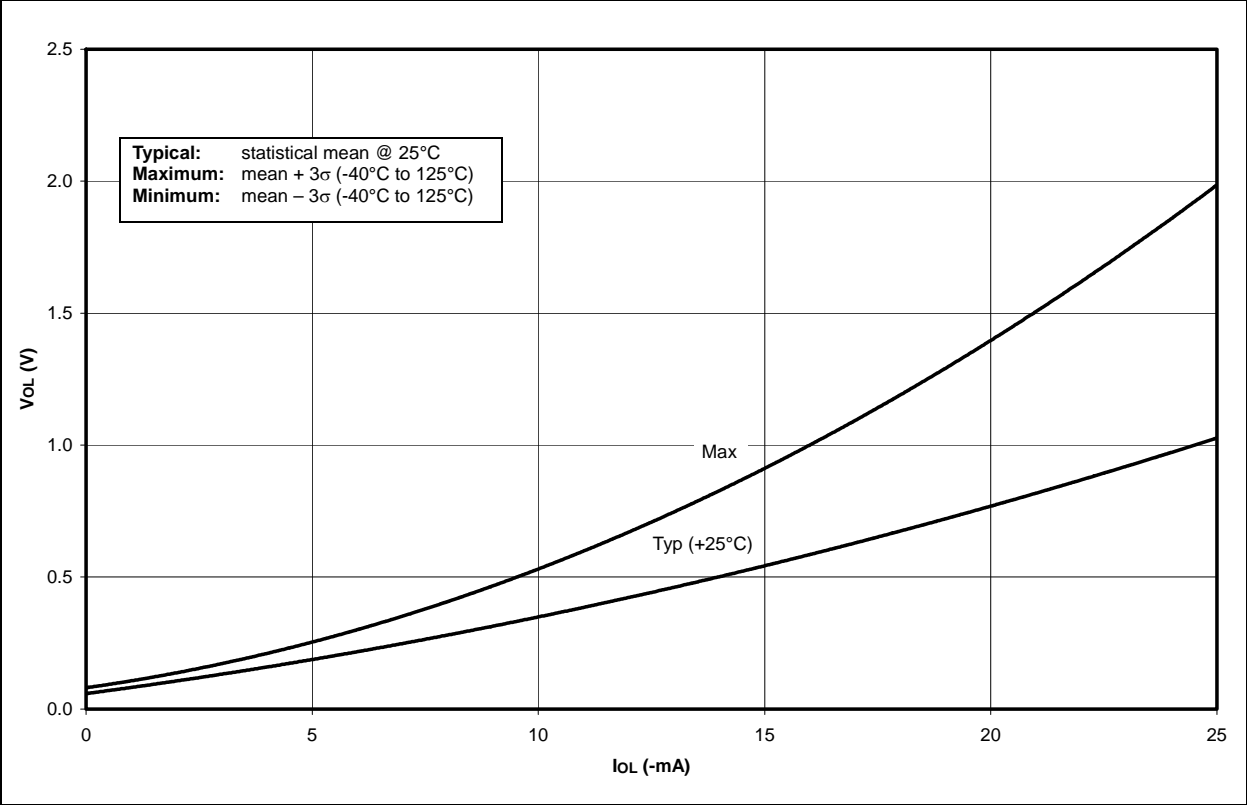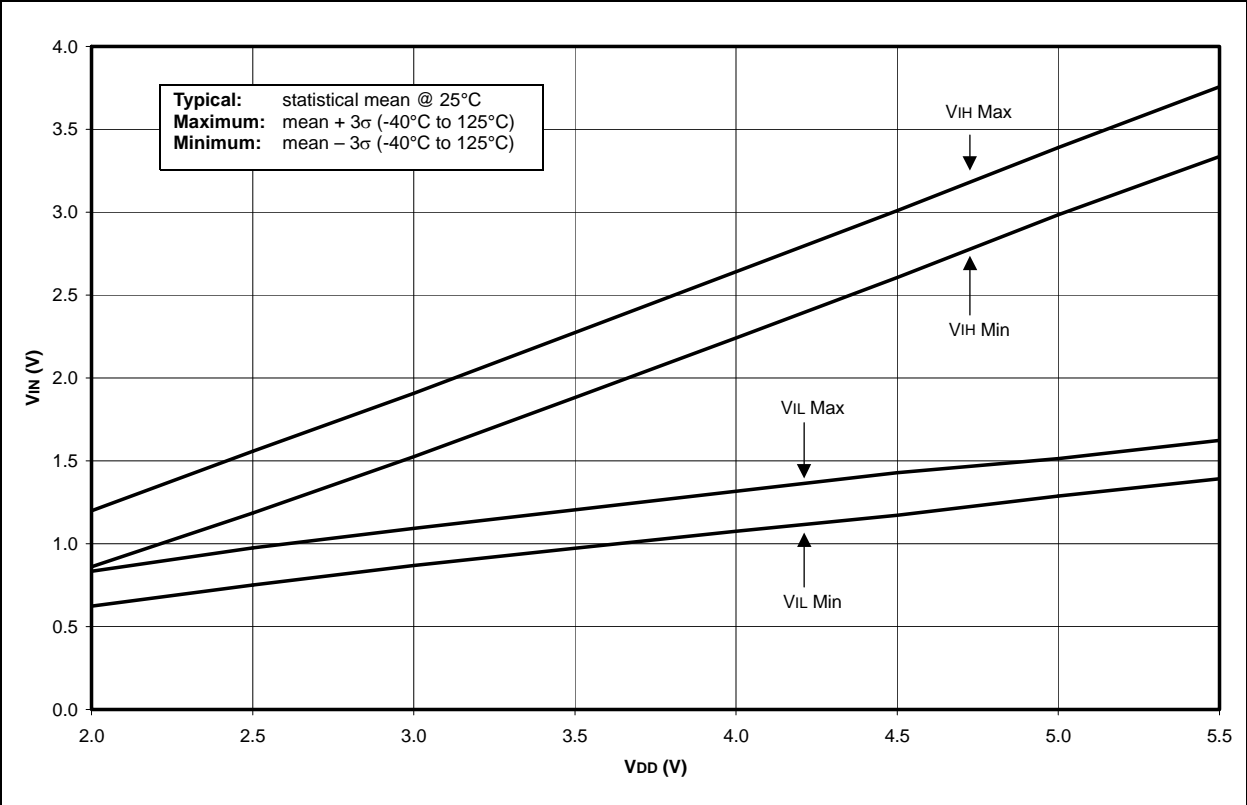**FIGURE 24-15:** TYPICAL AND MAXIMUM V$_{OL}$ vs. I$_{OL}$ (V$_{DD}$ = 3V, -40°C TO +125°C)

FIGURE 24-16: MINIMUM AND MAXIMUM V$_{IN}$ vs. V$_{DD}$ (ST INPUT, -40°C TO +125°C)
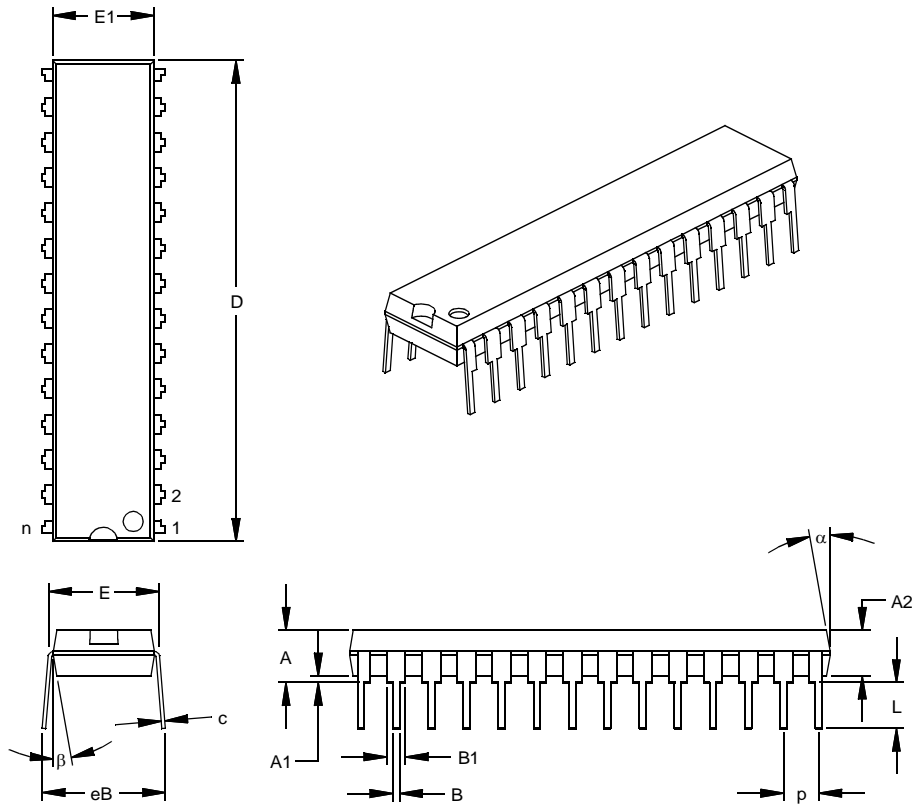
25.2 Package Details

The following sections give the technical details of the packages.

28-Lead Skinny Plastic Dual In-line (SP) – 300 mil (PDIP)

> **Note:** For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging

		Units	INCHES*			MILLIMETERS		
	Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins		n		28			28	
Pitch		p		.100			2.54	
Top to Seating Plane		A	.140	.150	.160	3.56	3.81	4.06
Molded Package Thickness		A2	.125	.130	.135	3.18	3.30	3.43
Base to Seating Plane		A1	.015			0.38		
Shoulder to Shoulder Width		E	.300	.310	.325	7.62	7.87	8.26
Molded Package Width		E1	.275	.285	.295	6.99	7.24	7.49
Overall Length		D	1.345	1.365	1.385	34.16	34.67	35.18
Tip to Seating Plane		L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness		c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width		B1	.040	.053	.065	1.02	1.33	1.65
Lower Lead Width		B	.016	.019	.022	0.41	0.48	0.56
Overall Row Spacing §		eB	.320	.350	.430	8.13	8.89	10.92
Mold Draft Angle Top		α	5	10	15	5	10	15
Mold Draft Angle Bottom		β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic
Notes:
Dimension D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed
.010" (0.254mm) per side.
JEDEC Equivalent: MO-095
Drawing No. C04-070

APPENDIX D: MIGRATION FROM HIGH-END TO ENHANCED DEVICES

A detailed discussion of the migration pathway and differences between the high-end MCU devices (i.e., PIC17CXXX) and the enhanced devices (i.e., PIC18FXXX) is provided in AN726, "PIC17CXXX to PIC18CXXX Migration". This Application Note is available as Literature Number DS00726.