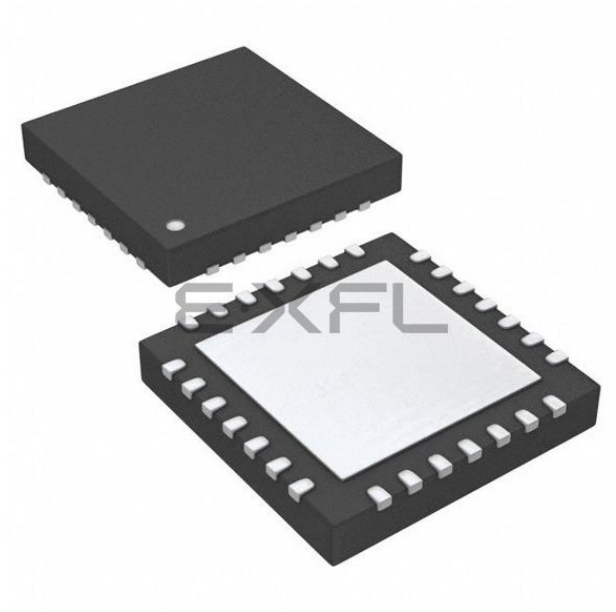


Welcome to E-XFL.COM

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	CANbus, I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	25
Program Memory Size	64KB (32K x 16)
Program Memory Type	FLASH
EEPROM Size	1K x 8
RAM Size	4K 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 ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-VQFN Exposed Pad
Supplier Device Package	28-QFN (6x6)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f26k83-i-ml

3.1.1 PRIORITY LOCK

The System arbiter grants memory access to the peripheral selections (DMAx, Scanner) when the PRLOCKED bit (PRLOCK Register) is set.

Priority selections are locked by setting the PRLOCKED bit of the PRLOCK register. Setting and clearing this bit requires a special sequence as an extra precaution against inadvertent changes. Examples of setting and clearing the PRLOCKED bit are shown in Example 3-1 and Example 3-2.

EXAMPLE 3-1: PRIORITY LOCK SEQUENCE

```
; Disable interrupts
BCF INTCON0,GIE

; Bank to PRLOCK register
BANKSEL PRLOCK
MOVLW 55h

; Required sequence, next 4
instructions
MOVWF PRLOCK
MOVLW AAh
MOVWF PRLOCK
; Set PRLOCKED bit to grant memory
access to peripherals
BSF PRLOCK,0

; Enable Interrupts
BSF INTCON0,GIE
```

EXAMPLE 3-2: PRIORITY UNLOCK SEQUENCE

```
; Disable interrupts
BCF INTCON0,GIE

; Bank to PRLOCK register
BANKSEL PRLOCK
MOVLW 55h

; Required sequence, next 4
instructions
MOVWF PRLOCK
MOVLW AAh
MOVWF PRLOCK
; Clear PRLOCKED bit to allow changing
priority settings
BCF PRLOCK,0

; Enable Interrupts
BSF INTCON0,GIE
```

3.2 Memory Access Scheme

The user can assign priorities to both system level and peripheral selections based on which the system arbiter grants memory access. Let us consider the following priority scenarios between ISR, MAIN, and Peripherals.

Note:	It is always required that the ISR priority be higher than Main priority.

3.2.1 ISR PRIORITY > MAIN PRIORITY > PERIPHERAL PRIORITY

When the Peripheral Priority (DMAx, Scanner) is lower than ISR and MAIN Priority, and the peripheral requires:

1. Access to the Program Flash Memory, then the peripheral waits for an instruction cycle in which the CPU does not need to access the PFM (such as a branch instruction) and uses that cycle to do its own Program Flash Memory access, unless a PFM Read/Write operation is in progress.
2. Access to the SFR/GPR, then the peripheral waits for an instruction cycle in which the CPU does not need to access the SFR/GPR (such as `MOVLW`, `CALL`, `NOP`) and uses that cycle to do its own SFR/GPR access.
3. Access to the Data EEPROM, then the peripheral has access to Data EEPROM unless a Data EEPROM Read/Write operation is being performed.

This results in the lowest throughput for the peripheral to access the memory, and does so without any impact on execution times.

3.2.2 PERIPHERAL PRIORITY > ISR PRIORITY > MAIN PRIORITY

When the Peripheral Priority (DMAx, Scanner) is higher than ISR and MAIN Priority, the CPU operation is stalled when the peripheral requests memory.

The CPU is held in its current state until the peripheral completes its operation. Since the peripheral requests access to the bus, the peripheral cannot be disabled until it completes its operation.

This results in the highest throughput for the peripheral to access the memory, but has the cost of stalling other execution while it occurs.

4.8.3 MAPPING THE ACCESS BANK IN INDEXED LITERAL OFFSET MODE

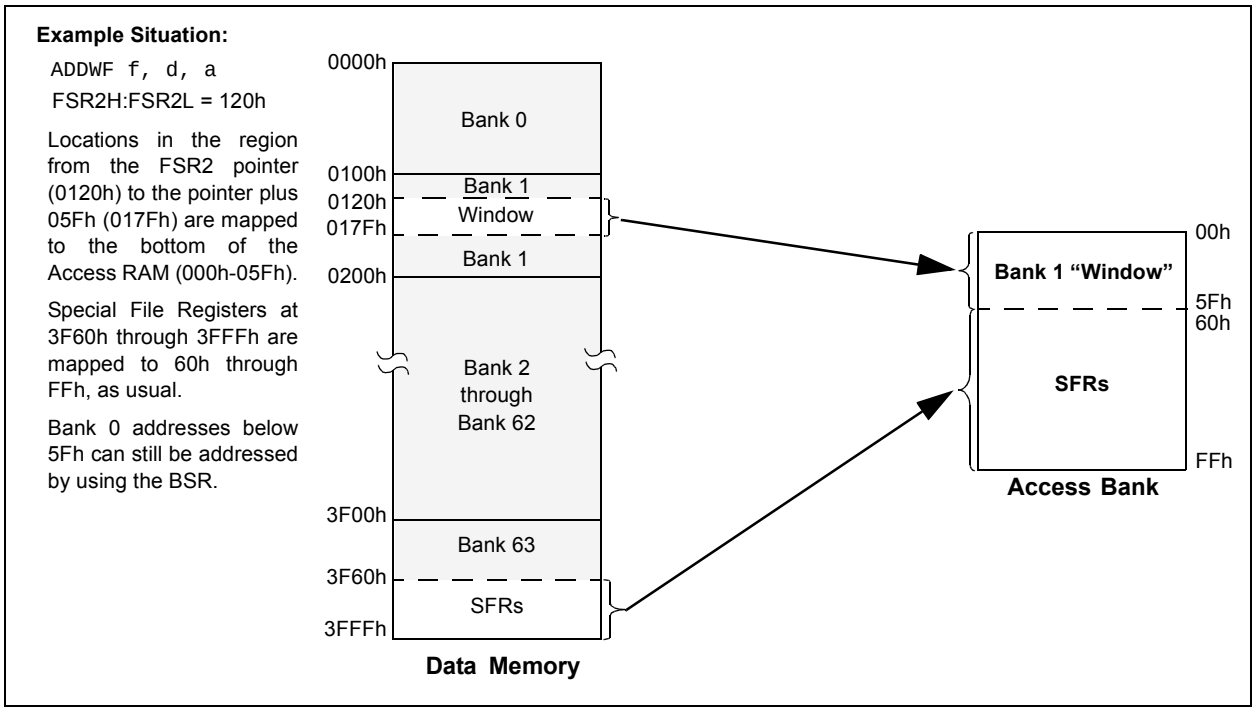
The use of Indexed Literal Offset Addressing mode effectively changes how the first 96 locations of Access RAM (00h to 5Fh) are mapped. Rather than containing just the contents of the bottom section of Bank 0, this mode maps the contents from a user defined “window” that can be located anywhere in the data memory space. The value of FSR2 establishes the lower boundary of the addresses mapped into the window, while the upper boundary is defined by FSR2 plus 95 (5Fh). Addresses in the Access RAM above 5Fh are mapped as previously described (see **Section 4.5.4 “Access Bank”**). An example of Access Bank remapping in this addressing mode is shown in Figure 4-8.

Remapping of the Access Bank applies *only* to operations using the Indexed Literal Offset mode. Operations that use the BSR (Access RAM bit is ‘1’) will continue to use direct addressing as before.

4.9 PIC18 Instruction Execution and the Extended Instruction Set

Enabling the extended instruction set adds eight additional commands to the existing PIC18 instruction set. These instructions are executed as described in **Section 42.2 “Extended Instruction Set”**.

FIGURE 4-8: REMAPPING THE ACCESS BANK WITH INDEXED LITERAL OFFSET ADDRESSING



9.3.2 NATURAL ORDER (HARDWARE) PRIORITY

When more than one interrupt with the same user specified priority level are requested, the priority conflict is resolved by using a method called “Natural Order Priority”. Natural order priority is a fixed priority scheme that is based on the Interrupt Vector Table. Table 9-2 shows the natural order priority and the interrupt vector number assigned for each source.

TABLE 9-2: INTERRUPT VECTOR PRIORITY TABLE

Vector Number	Interrupt Source
0	Software Interrupt
1	HLVD
2	OSF
3	CSW
4	NVM
5	SCAN
6	CRC
7	IOC
8	INT0
9	ZCD
10	AD
11	ADT
12	C1
13	SMT1
14	SMT1PRA
15	SMT1PWA
16	DMA1SCNT
17	DMA1DCNT
18	DMA1OR
19	DMA1A
20	SPI1RX
21	SPI1TX
22	SPI1
23	I2C1RX
24	I2C1TX
25	I2C1
26	I2C1E
27	U1RX
28	U1TX
29	U1E
30	U1
31	TMR0
32	TMR1
33	TMR1G
34	TMR2
35	CCP1
36	NCO
37	CWG1
38	CLC1
39	INT1
40	RXB0IF/FIFOIF
41	RXB1IF/RXBnIF

Vector Number	Interrupt Source
42	TXB0IF
43	TXB1IF
44	TXB2IF/TXBnIF
45	ERRIF
46	WAKIF
47	IRXIF
48	C2
49	SMT2
50	SMT2PRA
51	SMT2PWA
52	DMA2SCNT
53	DMA2DCNT
54	DMA2OR
55	DMA2A
56	I2C2RX
57	I2C2TX
58	I2C2
59	I2C2E
60	U2RX
61	U2TX
62	U2E
63	U2
64	TMR3
65	TMR3G
66	TMR4
67	CCP2
68	CWG2
69	CLC2
70	INT2
71	TMR5
72	TMR5G
73	TMR6
74	CCP3
75	CWG3
76	CLC3
77	CCP4
78	CLC4
79	—
80	—
81	—

The natural order priority scheme has vector interrupt 0 as the highest priority and vector interrupt 81 as the lowest priority.

For example, when two concurrently occurring interrupt sources that are both designated high priority using the IPRx register will be resolved using the natural order priority (i.e., the interrupt with a lower corresponding vector number will preempt the interrupt with the higher vector number).

The ability for the user to assign every interrupt source to high or low priority levels means that the user program can give an interrupt with a low natural order priority a higher overall priority level.

9.4 Interrupt Operation

All pending interrupts are indicated by the flag bit being equal to a ‘1’ in the PIRx register. All pending interrupts are resolved using the priority scheme explained in **Section 9.3 “Interrupt Priority”**.

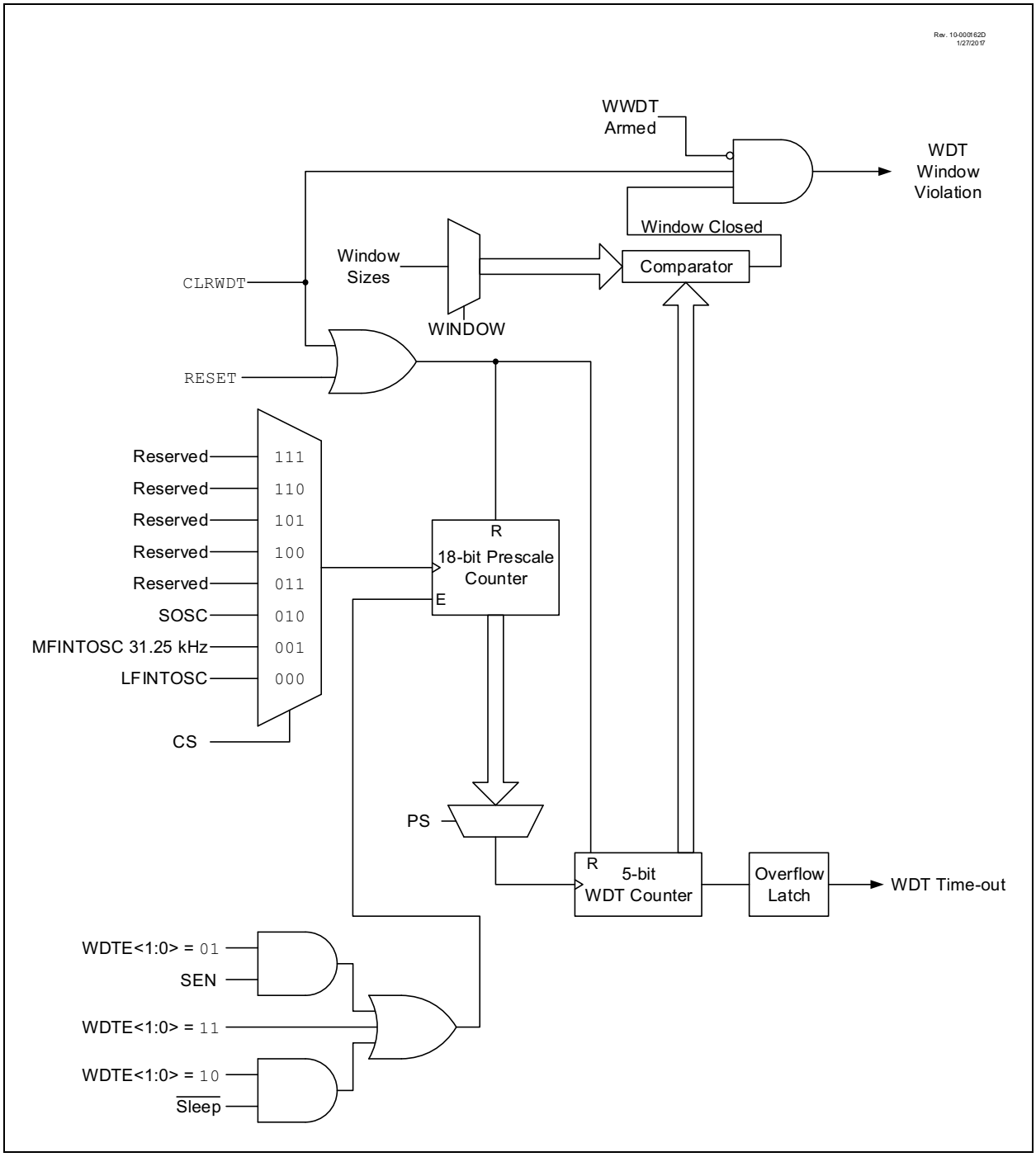
Once the interrupt source to be serviced is resolved, the program execution vectors to the resolved interrupt vector addresses, as explained in **Section 9.2 “Interrupt Vector Table (IVT)”**. The vector number is also stored in the WREG register. Most of the flag bits are required to be cleared by the application software, but in some cases, device hardware clears the interrupt automatically. Some flag bits are read-only in the PIRx registers, these flags are a summary of the source interrupts and the corresponding interrupt flags of the source must be cleared.

A valid interrupt can be either a high or low priority interrupt when in main routine or a high priority interrupt when in low priority Interrupt Service Routine. Depending on order of interrupt requests received and their relative timing, the CPU will be in the state of execution indicated by the STAT bits of the INTCON1 register (Register 9-2).

The State machine shown in Figure 9-1 and the subsequent sections detail the execution of interrupts when received in different orders.

Note:	The state of GIEH/L is not changed by the hardware when servicing an interrupt. The internal state machine is used to keep track of execution states. These bits can be manipulated in the user code resulting in transferring execution to the main routine and ignoring existing interrupts.

FIGURE 11-1: WINDOWED WATCHDOG TIMER BLOCK DIAGRAM



EXAMPLE 13-4: WRITING TO PROGRAM FLASH MEMORY (CONTINUED)

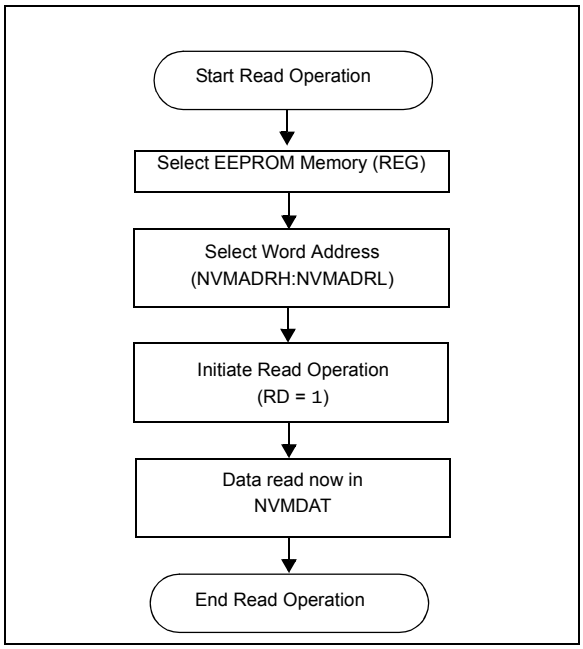
```
WRITE_BYTE_TO_HREGS
              MOVF      POSTINC0, W              ; get low byte of buffer data
              MOVWF     TABLAT                   ; present data to table latch
              TBLWT+*                            ; write data, perform a short write
                                                 ; to internal TBLWT holding register.
              DECFSZ    COUNTER                  ; loop until holding registers are full
              BRA       WRITE_WORD_TO_HREGS
PROGRAM_MEMORY
              BCF       NVMCON1, REG0            ; point to Program Flash Memory
              BSF       NVMCON1, REG1            ; point to Program Flash Memory
              BSF       NVMCON1, WREN            ; enable write to memory
              BCF       NVMCON1, FREE            ; enable write to memory
              BCF       INTCON0, GIE             ; disable interrupts
              MOVLW     55h
Required      MOVWF     NVMCON2                  ; write 55h
Sequence      MOVLW     0AAh
              MOVWF     NVMCON2                  ; write 0AAh
              BSF       NVMCON1, WR              ; start program (CPU stall)
              DCFSZ     COUNTER2                 ; repeat for remaining write blocks
              BRA       WRITE_BYTE_TO_HREGS
              BSF       INTCON0, GIE             ; re-enable interrupts
              BCF       NVMCON1, WREN            ; disable write to memory
```

13.3.3 READING THE DATA EEPROM MEMORY

To read a data memory location, the user must write the address to the NVMADRL and NVMADRH register pair, clear REG<1:0> control bit in NVMCON1 register to access Data EEPROM locations and then set control bit, RD. The data is available on the very next instruction cycle; therefore, the NVMDAT register can be read by the next instruction. NVMDAT will hold this value until another read operation, or until it is written to by the user (during a write operation).

The basic process is shown in Example 13-5.

FIGURE 13-11: DATA EEPROM READ FLOWCHART



13.3.4 WRITING TO THE DATA EEPROM MEMORY

To write an EEPROM data location, the address must first be written to the NVMADRL and NVMADRH register pair and the data written to the NVMDAT register. The sequence in Example 13-6 must be followed to initiate the write cycle.

The write will not begin if NVM Unlock sequence, described in **Section 13.1.4 "NVM Unlock Sequence"**, is not exactly followed for each byte. It is strongly recommended that interrupts be disabled during this code segment.

Additionally, the WREN bit in NVMCON1 must be set to enable writes. This mechanism prevents accidental writes to data EEPROM due to unexpected code execution (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, NVMCON1, NVMADRL, NVMADRH and NVMDAT cannot be modified. The WR bit will be inhibited from being set unless the WREN bit is set. Both WR and WREN cannot be set with the same instruction.

After a write sequence has been initiated, clearing the WREN bit will not affect this write cycle. A single Data EEPROM word is written and the operation includes an implicit erase cycle for that word (it is not necessary to set FREE). CPU execution continues in parallel and at the completion of the write cycle, the WR bit is cleared in hardware and the NVM Interrupt Flag bit (NVMIF) is set. The user can either enable this interrupt or poll this bit. NVMIF must be cleared by software.

EXAMPLE 15-1: SETUP DMA1 TO MOVE DATA FROM PROGRAM FLASH MEMORY TO UART1 TRANSMIT BUFFER USING HARDWARE TRIGGERS

```
//This code example illustrates using DMA1 to transfer
//10 bytes of data from 0x1000 in PFM to U1TXB 0x3DEA

void main() {
    //System Initialize
    initializeSystem();

    //Setup UART1
    initializeUART1();

    //Setup DMA1
    //DMA1CON1 - DPTR remains, Source Memory Region PFM, SPTR increments, SSTP
    DMA1CON1 = 0x0B;

    //Source registers
    //Source size
    DMA1SSZH = 0x00;
    DMA1SSZL = 0x0A;

    //Source start address, 0x1000
    DMA1SSAU = 0x00;
    DMA1SSAH = 0x10;
    DMA1SSAL = 0x00;

    //Destination registers
    //Destination size
    DMA1DSZH = 0x00;
    DMA1DSZL = 0x01;

    //Destination start address, 0x3DEA
    DMA1DSAH = 0x3D;
    DMA1DSAL = 0xEA;

    //Start trigger source U1TX
    DMA1SIRQ = 0x1C;

    //Enable & Start DMA transfer
    DMA1CON0 = 0xC0;

    while (1) {
        doSomething();
    }
}
```

15.13 Register definitions: DMA

Long bit name prefixes for the DMA peripherals are shown in Table 15-7. Refer to **Section 1.3 "Register and Bit naming conventions"** for more information.

TABLE 15-7: REGISTER AND BIT NAMING

Peripheral	Bit Name Prefix
DMA 1	DMA1
DMA 2	DMA2

REGISTER 15-18: DMAxDSZL: DMAx DESTINATION SIZE 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
DSZ<7:0>							
bit 7							bit 0

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

bit 7-0 **DSZ<7:0>:** Destination Message Size bits

REGISTER 15-19: DMAxDSZH: DMAx DESTINATION SIZE HIGH REGISTER

U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	DSZ<11:8>			
bit 7							bit 0

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

bit 7-4 **Unimplemented:** Read as '0'

bit 3-0 **DSZ<11:8>:** Destination Message Size bits

REGISTER 15-20: DMAxDCNTL: DMAx DESTINATION COUNT LOW REGISTER

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DCNT<7:0>							
bit 7							bit 0

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

bit 7-0 **DCNT<7:0>:** Current Destination Byte Count

24.2 Register Definitions: PWM Control

Long bit name prefixes for the PWM peripherals are shown below. Refer to **Section 1.3.2.2 "Long Bit Names"** for more information.

Peripheral	Bit Name Prefix
PWM3	PWM3
PWM4	PWM4

REGISTER 24-1: PWMxCON: PWM CONTROL REGISTER

R/W-0/0	U-0	R-0/0	R/W-0/0	U-0	U-0	U-0	U-0
EN	—	OUT	POL	—	—	—	—
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 **EN:** PWM Module Enable bit
1 = PWM module is enabled
0 = PWM module is disabled

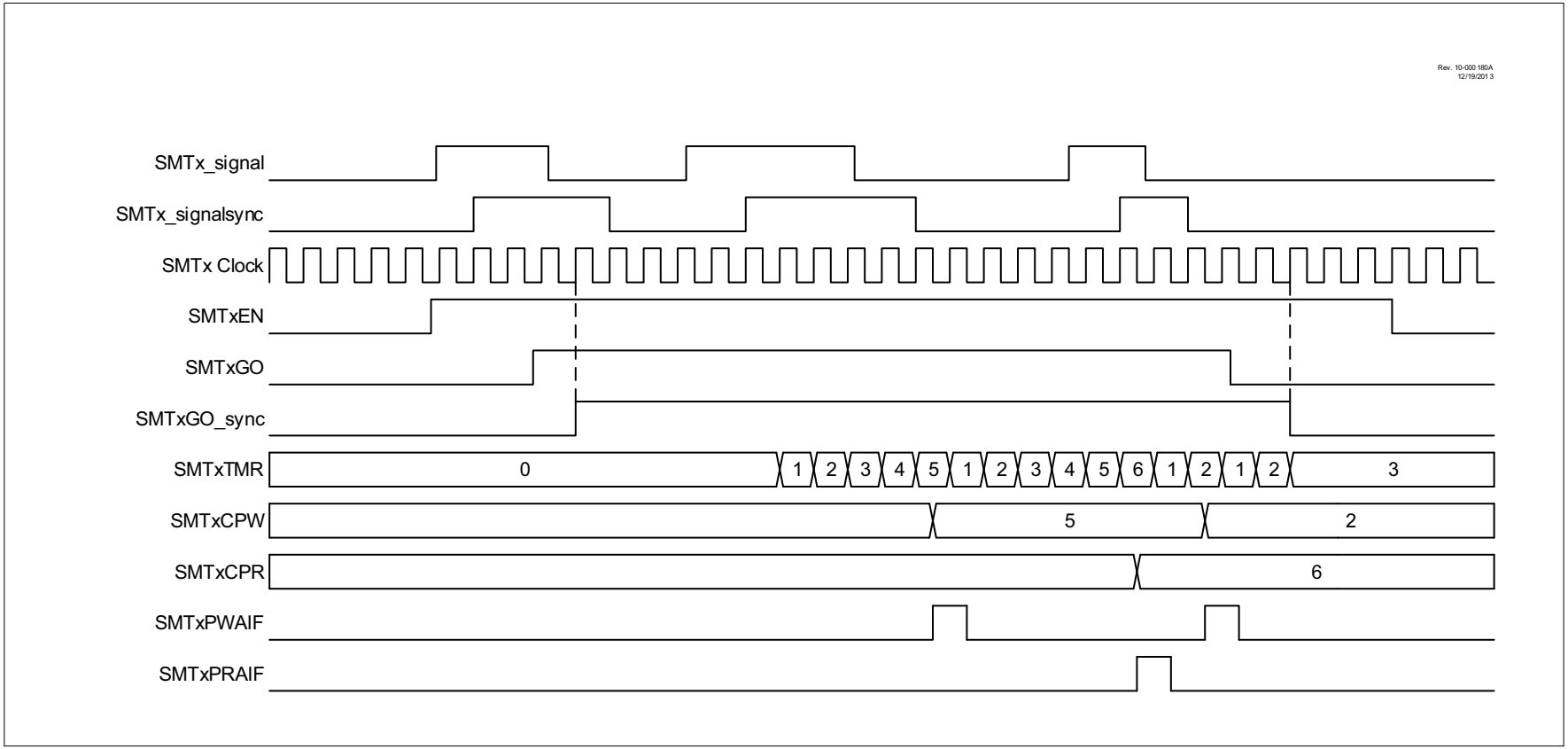
bit 6 **Unimplemented:** Read as '0'

bit 5 **OUT:** PWM Module Output Level When Bit is Read

bit 4 **POL:** PWM Output Polarity Select bit
1 = PWM output is inverted
0 = PWM output is normal

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

FIGURE 25-8: HIGH AND LOW MEASURE MODE REPEAT ACQUISITION TIMING DIAGRAM



REGISTER 25-13: SMTxCPWL: SMT CAPTURED PULSE WIDTH REGISTER – LOW BYTE

R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x
SMTxCPW<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 **SMTxCPW<7:0>**: Significant bits of the SMT PW Latch – Low Byte

REGISTER 25-14: SMTxCPWH: SMT CAPTURED PULSE WIDTH REGISTER – HIGH BYTE

R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x
SMTxCPW<15:8>							
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 7-0 **SMTxCPW<15:8>**: Significant bits of the SMT PW Latch – High Byte

REGISTER 25-15: SMTxCPWU: SMT CAPTURED PULSE WIDTH REGISTER – UPPER BYTE

R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x	R-x/x
SMTxCPW<23:16>							
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 **SMTxCPW<23:16>**: Significant bits of the SMT PW Latch – Upper Byte

31.11 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.

31.11.1 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.

31.11.2 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.

31.12 Flow Control

This section does not apply to the LIN, DALI, or DMX modes.

Flow control is the means by which a sending UART data stream can be suspended by a receiving UART. Flow control prevents input buffers from overflowing without software intervention. The UART supports both hardware and XON/XOFF methods of flow control.

The flow control method is selected with the FLO<1:0> bits in the UxCON2 register. Flow control is disabled when are both bits are cleared.

31.12.1 HARDWARE FLOW CONTROL

Hardware flow control is selected by setting the FLO<1:0> bits to '10'.

Hardware flow control consists of three lines. The RS-232 signal names for two of these are $\overline{\text{RTS}}$, and $\overline{\text{CTS}}$. Both are low true. The third line may be used to control an RS-485 transceiver. The signal name for this is TXDE for transmit drive enable. This output is high when the TX output is actively sending a character and low at all other times. The UART is configured as DTE (computer) equipment which means $\overline{\text{RTS}}$ is an output and $\overline{\text{CTS}}$ is an input.

The $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$ signals work as a pair to control the transmission flow. A DTE-to-DTE configuration connects the $\overline{\text{RTS}}$ output of the receiving UART to the $\overline{\text{CTS}}$ input of the sending UART. Refer to Figure 31-10.

The UART receiving data asserts the $\overline{\text{RTS}}$ output low when the input FIFO is empty. When a character is received, the $\overline{\text{RTS}}$ output goes high until the UxRXB is read to free up both FIFO locations.

When the $\overline{\text{CTS}}$ input goes high after a byte has started to transmit, the transmission will complete normally. The receiver accommodates this by accepting the character in the second FIFO location even when the $\overline{\text{CTS}}$ input is high.

REGISTER 31-18: UxTXCHK: UART TRANSMIT CHECKSUM RESULT 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
TXCHK<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 **TXCHK<7:0>:** Checksum calculated from TX bytes
LIN mode and C0EN = 1:
Sum of all transmitted bytes including PID
LIN mode and C0EN = 0:
Sum of all transmitted bytes except PID
All other modes and C0EN = 1:
Sum of all transmitted bytes since last clear
All other modes and C0EN = 0:
Not used

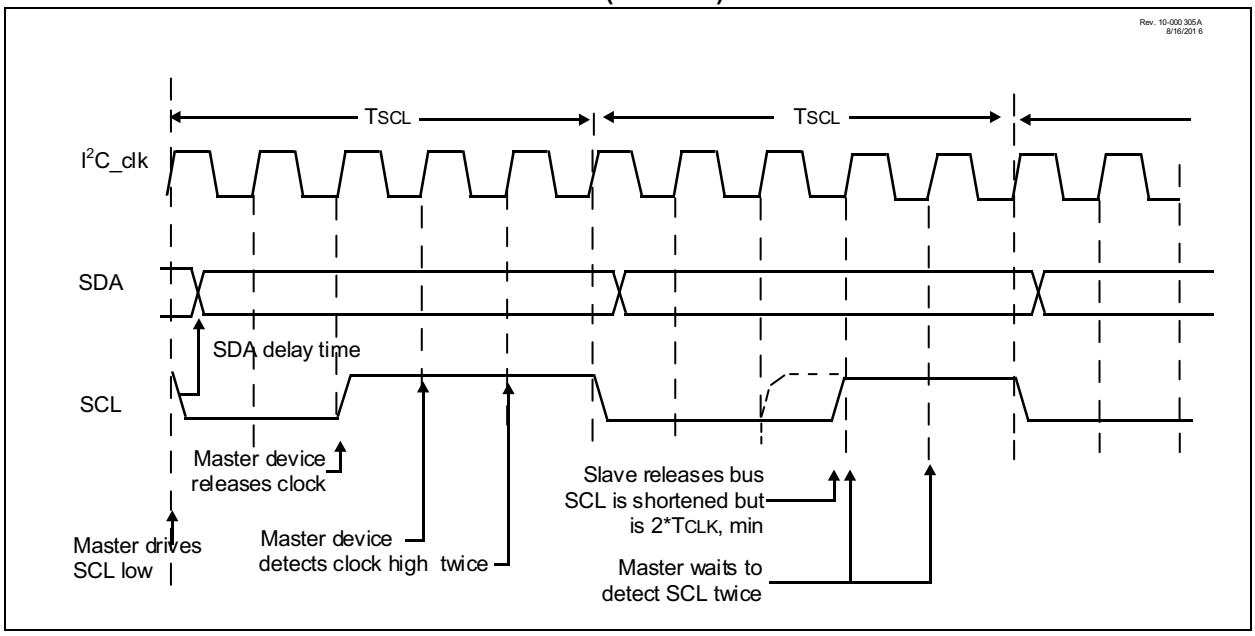
REGISTER 31-19: UxRXCHK: UART RECEIVE CHECKSUM RESULT REGISTER

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
RXCHK<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 **RXCHK<7:0>:** Checksum calculated from RX bytes
LIN mode and C0EN = 1:
Sum of all received bytes including PID
LIN mode and C0EN = 0:
Sum of all received bytes except PID
All other modes and C0EN = 1:
Sum of all received bytes since last clear
All other modes and C0EN = 0:
Not used

FIGURE 33-13: CLOCK SYNTHESIS TIMING (FME = 0)

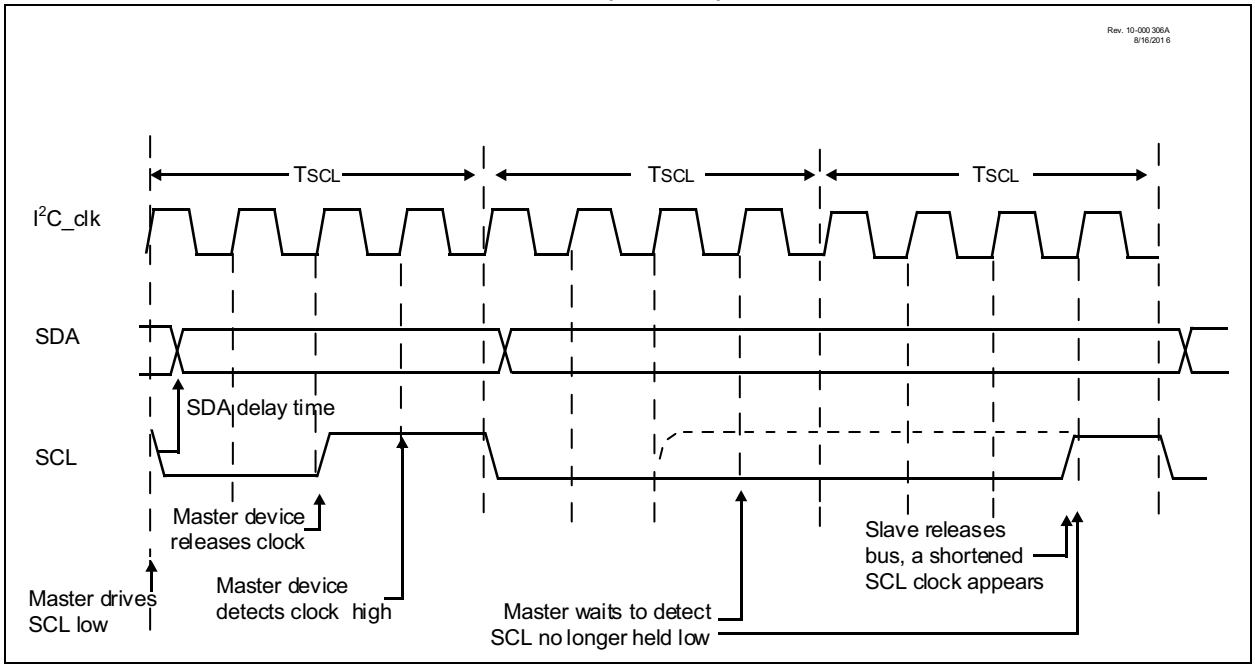


33.5.4.2 Clock Timing with FME = 1

One TSCL, consists of four clocks of the I^2C clock input. The first clock is used to drive SCL low, the third releases SCL high, and the fourth is used to detect if the clock is, in fact, high or being stretched by a slave.

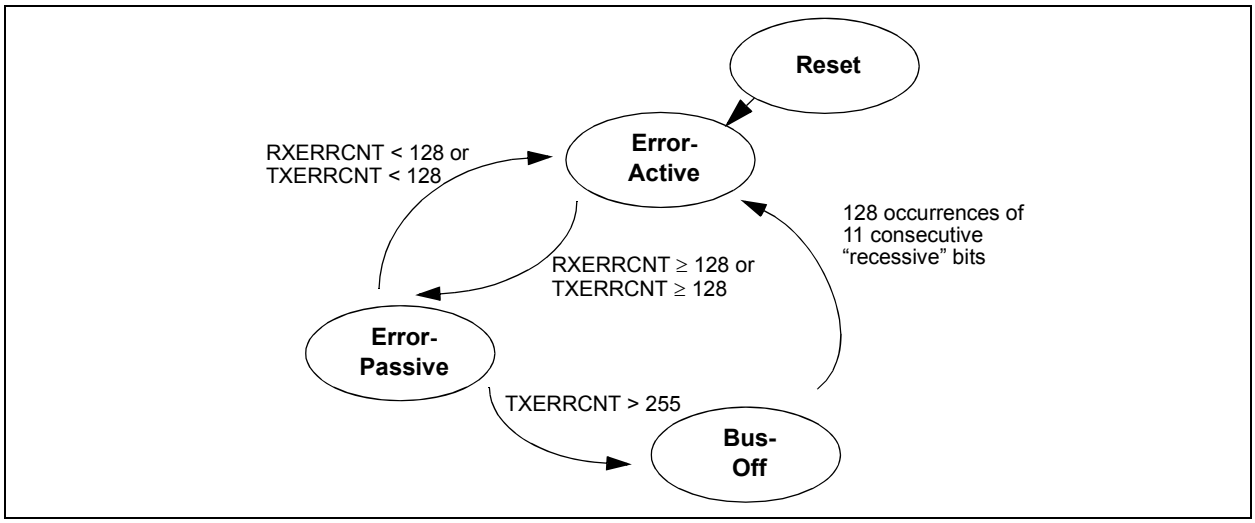
If a slave is clock stretching, the hardware waits; checking SCL on each successive I^2C clock, proceeding only after detecting SCL high. Figure 33-14 shows the clock synthesis timing when FME = 1.

FIGURE 33-14: CLOCK SYNTHESIS TIMING (FME = 1)



Additionally, there is an Error State Warning flag bit, EWARN, which is set if at least one of the error counters equals or exceeds the error warning limit of 96. EWARN is reset if both error counters are less than the error warning limit.

FIGURE 34-8: ERROR MODES STATE DIAGRAM



REGISTER 34-21: RXERRCNT: RECEIVE ERROR COUNT REGISTER

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0
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 7-0 **REC<7:0>:** Receive Error Counter bits

This register contains the receive error value as defined by the CAN specifications. When RXERRCNT > 127, the module will go into an error-passive state. RXERRCNT does not have the ability to put the module in "bus-off" state.

EXAMPLE 34-5: READING A CAN MESSAGE

```
    ; Need to read a pending message from RXB0 buffer.
    ; To receive any message, filter, mask and RXM1:RXM0 bits in RXB0CON registers must be
    ; programmed correctly.
    ;
    ; Make sure that there is a message pending in RXB0.
    BTFSS   RXB0CON, RXFUL                ; Does RXB0 contain a message?
    BRA     NoMessage                     ; No. Handle this situation...
    ; We have verified that a message is pending in RXB0 buffer.
    ; If this buffer can receive both Standard or Extended Identifier messages,
    ; identify type of message received.
    BTFSS   RXB0SIDL, EXID                ; Is this Extended Identifier?
    BRA     StandardMessage               ; No. This is Standard Identifier message.
                                          ; Yes. This is Extended Identifier message.
    ; Read all 29-bits of Extended Identifier message.
    ...
    ; Now read all data bytes
    MOVFF   RXB0DO, MY_DATA_BYTE1
    ...
    ; Once entire message is read, mark the RXB0 that it is read and no longer FULL.
    BCF     RXB0CON, RXFUL                ; This will allow CAN Module to load new messages
                                          ; into this buffer.
    ...
```

REGISTER 34-58: IPR5: PERIPHERAL INTERRUPT PRIORITY REGISTER 5 (CONTINUED)

bit 0 When CAN is in Mode 0:
RXB0IP: CAN Receive Buffer 0 Interrupt Priority bit
1 = High priority
0 = Low priority
When CAN is in Mode 1:
Unimplemented: Read as '0'
When CAN is in Mode 2:
FIFOWMIP: FIFO Watermark Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: In CAN Mode 1 and 2, these bits are forced to '0'.

REGISTER 34-59: TXBIE: TRANSMIT BUFFERS INTERRUPT ENABLE REGISTER(1)

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	U-0	U-0
—	—	—	TXB2IE(2)	TXB1IE(2)	TXB0IE(2)	—	—
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 7-5 **Unimplemented:** Read as '0'

bit 4-2 **TXB2IE:TXB0IE:** Transmit Buffer 2-0 Interrupt Enable bits(2)
1 = Transmit buffer interrupt is enabled
0 = Transmit buffer interrupt is disabled

bit 1-0 **Unimplemented:** Read as '0'

Note 1: This register is available in Mode 1 and 2 only.
2: TXBnIE in PIE5 register must be set to get an interrupt.

37.2.5 AUTO-CONVERSION TRIGGER

The auto-conversion trigger allows periodic ADC measurements without software intervention. When a rising edge of the selected source occurs, the GO bit is set by hardware.

The auto-conversion trigger source is selected by the ADACT register.

Using the auto-conversion trigger does not assure proper ADC timing. It is the user's responsibility to ensure that the ADC timing requirements are met. See Register 37-33 for auto-conversion sources.

37.2.6 ADC CONVERSION PROCEDURE (BASIC MODE)

This is an example procedure for using the ADC to perform an analog-to-digital conversion:

1. Configure Port:
   - Disable pin output driver (Refer to the TRISx register)
   - Configure pin as analog (Refer to the ANSELx register)
2. Configure the ADC module:
   - Select ADC conversion clock
   - Select voltage reference
   - Select ADC input channel
   - Precharge and acquisition
   - Turn on ADC module
3. Configure ADC interrupt (optional):
   - Clear ADC interrupt flag
   - Enable ADC interrupt
   - Enable global interrupt (GIEL bit)**(1)**
4. If ADACQ = 0, software must wait the required acquisition time**(2)**.
5. Start conversion by setting the GO bit.
6. Wait for ADC conversion to complete by one of the following:
   - Polling the GO bit
   - Polling the ADIF bit
   - Waiting for the ADC interrupt (interrupts enabled)
7. Read ADC Result.
8. Clear the ADC interrupt flag (required if interrupt is enabled).

Note 1: The global interrupt can be disabled if the user is attempting to wake-up from Sleep and resume in-line code execution.

2: Refer to **Section 37.3 "ADC Acquisition Requirements"**.

EXAMPLE 37-1: ADC CONVERSION

```
/*This code block configures the ADC
for polling, VDD and VSS references, FRC
oscillator and AN0 input.
Conversion start & polling for completion
are included.
 */
void main() {
    //System Initialize
    initializeSystem();

    //Setup ADC
    ADCON0bits.FM = 1; //right justify
    ADCON0bits.CS = 1; //FRC Clock
    ADPCH = 0x00; //RA0 is Analog channel
    TRISAbits.TRISA0 = 1; //Set RA0 to input
    ANSELAbits.ANSELA0 = 1; //Set RA0 to analog
    ADCON0bits.ON = 1; //Turn ADC On

    while (1) {
        ADCON0bits.GO = 1; //Start conversion
        while (ADCON0bits.GO); //Wait for conversion done
        resultHigh = ADRESH; //Read result
        resultLow = ADRESL; //Read result
    }
}
```

39.9 CWG1 Auto-Shutdown Source

The output of the comparator module can be used as an auto-shutdown source for the CWG1 module. When the output of the comparator is active and the corresponding WGASxE is enabled, the CWG operation will be suspended immediately (see **Section 26.10.1.2 "External Input Source"**).

39.10 ADC Auto-Trigger Source

The output of the comparator module can be used to trigger an ADC conversion. When the ADACT register is set to trigger on a comparator output, an ADC conversion will trigger when the Comparator output goes high.

39.11 TMR2/4/6 Reset

The output of the comparator module can be used to reset Timer2. When the TxERS register is appropriately set, the timer will reset when the Comparator output goes high.

39.12 Operation in Sleep Mode

The comparator module can operate during Sleep. The comparator clock source is based on the Timer1 clock source. If the Timer1 clock source is either the system clock (FOSC) or the instruction clock (FOSC/4), Timer1 will not operate during Sleep, and synchronized comparator outputs will not operate.

A comparator interrupt will wake the device from Sleep. The CxIE bits of the respective PIE register must be set to enable comparator interrupts.

SUBFWB **Subtract f from W with borrow**

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

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

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

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

Encoding:

0101	01da	ffff	ffff

Description: Subtract register 'f' and CARRY flag (borrow) from W (2's complement method). If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored 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 42.2.3 "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: `SUBFWB REG, 1, 0`

Before Instruction
REG = 3
W = 2
C = 1
After Instruction
REG = FF
W = 2
C = 0
Z = 0
N = 1 ; result is negative

Example 2: `SUBFWB REG, 0, 0`

Before Instruction
REG = 2
W = 5
C = 1
After Instruction
REG = 2
W = 3
C = 1
Z = 0
N = 0 ; result is positive

Example 3: `SUBFWB REG, 1, 0`

Before Instruction
REG = 1
W = 2
C = 0
After Instruction
REG = 0
W = 2
C = 1
Z = 1 ; result is zero
N = 0

SUBLW **Subtract W from literal**

Syntax: SUBLW k

Operands: $0 \le k \le 255$

Operation: k – (W) → W

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

Encoding:

0000	1000	kkkk	kkkk

Description: W is subtracted from 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 1: `SUBLW 02h`

Before Instruction
W = 01h
C = ?
After Instruction
W = 01h
C = 1 ; result is positive
Z = 0
N = 0

Example 2: `SUBLW 02h`

Before Instruction
W = 02h
C = ?
After Instruction
W = 00h
C = 1 ; result is zero
Z = 1
N = 0

Example 3: `SUBLW 02h`

Before Instruction
W = 03h
C = ?
After Instruction
W = FFh ; (2's complement)
C = 0 ; result is negative
Z = 0
N = 1