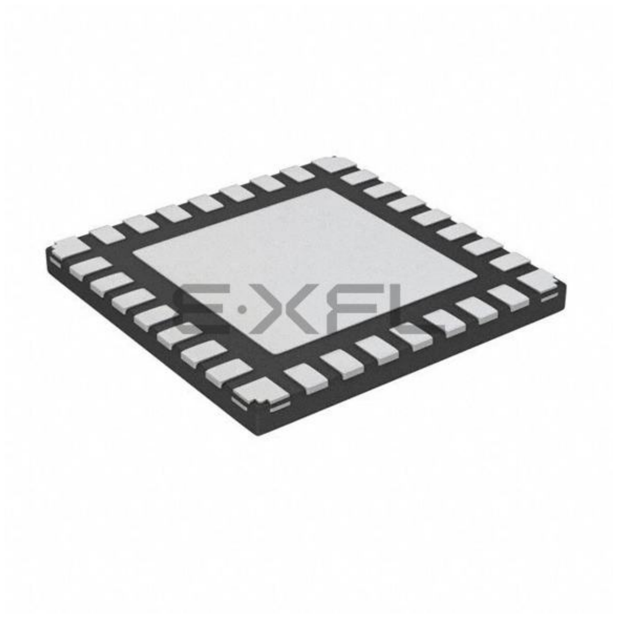
Microchip Technology - PIC18LF26K83T-I/MX Datasheet



Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	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)	1.8V ~ 3.6V
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-UQFN Exposed Pad
Supplier Device Package	28-UQFN (6x6)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lf26k83t-i-mx

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.

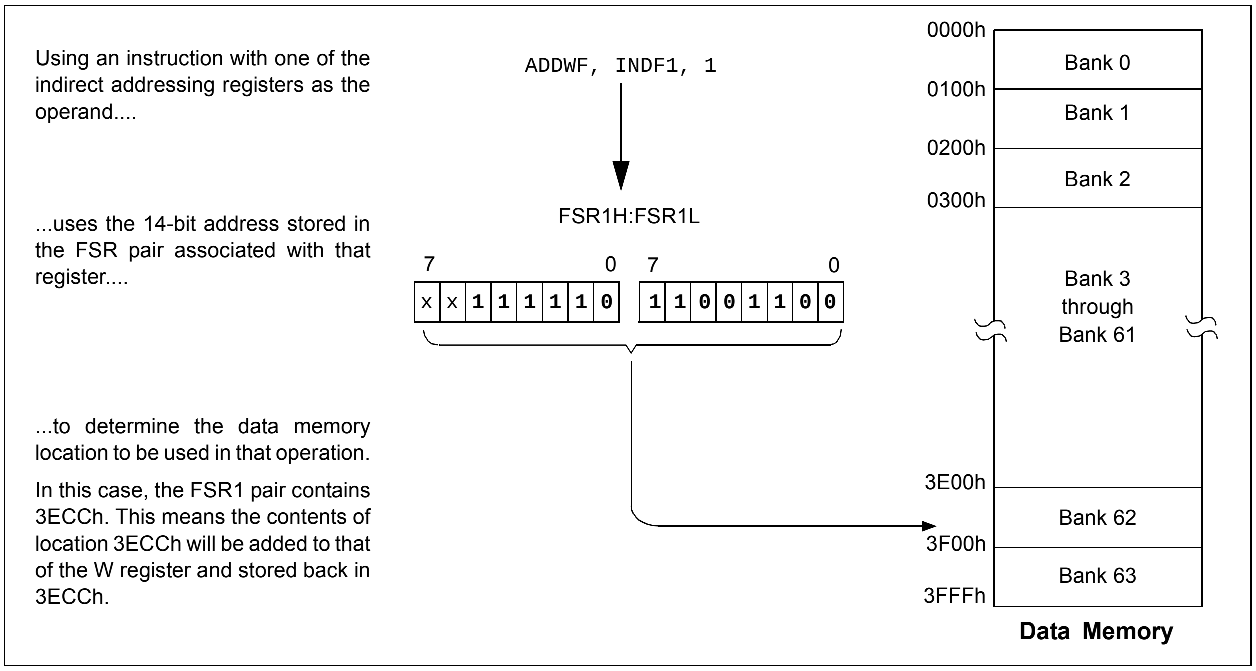
4.7.3.2 FSR Registers, POSTINC, POSTDEC, PREINC and PLUSW

In addition to the INDF operand, each FSR register pair also has four additional indirect operands. Like INDF, these are “virtual” registers which cannot be directly read or written. Accessing these registers actually accesses the location to which the associated FSR register pair points, and also performs a specific action on the FSR value. They are:

- POSTDEC: accesses the location to which the FSR points, then automatically decrements the FSR by 1 afterwards
- POSTINC: accesses the location to which the FSR points, then automatically increments the FSR by 1 afterwards
- PREINC: automatically increments the FSR by 1, then uses the location to which the FSR points in the operation
- PLUSW: adds the signed value of the W register (range of -127 to 128) to that of the FSR and uses the location to which the result points in the operation.

In this context, accessing an INDF register uses the value in the associated FSR register without changing it. Similarly, accessing a PLUSW register gives the FSR value an offset by that in the W register; however, neither W nor the FSR is actually changed in the operation. Accessing the other virtual registers changes the value of the FSR register.

FIGURE 4-6: INDIRECT ADDRESSING


Using an instruction with one of the indirect addressing registers as the operand....
ADDWF, INDF1, 1
FSR1H:FSR1L
...uses the 14-bit address stored in the FSR pair associated with that register....
7 0 7 0
x x 1 1 1 1 1 0 1 1 0 0 1 1 0 0
...to determine the data memory location to be used in that operation.
In this case, the FSR1 pair contains 3ECCh. This means the contents of location 3ECCh will be added to that of the W register and stored back in 3ECCh.
0000h
Bank 0
0100h
Bank 1
0200h
Bank 2
0300h
Bank 3 through Bank 61
3E00h
Bank 62
3F00h
Bank 63
3FFFh
Data Memory

8.0 REFERENCE CLOCK OUTPUT MODULE

The reference clock output module provides the ability to send a clock signal to the clock reference output pin (CLKR). The reference clock output can also be used as a signal for other peripherals, such as the Data Signal Modulator (DSM), Memory Scanner and Timer module.

The reference clock output module has the following features:

- Selectable clock source using the CLKRCLK register
- Programmable clock divider
- Selectable duty cycle

FIGURE 8-1: CLOCK REFERENCE BLOCK DIAGRAM

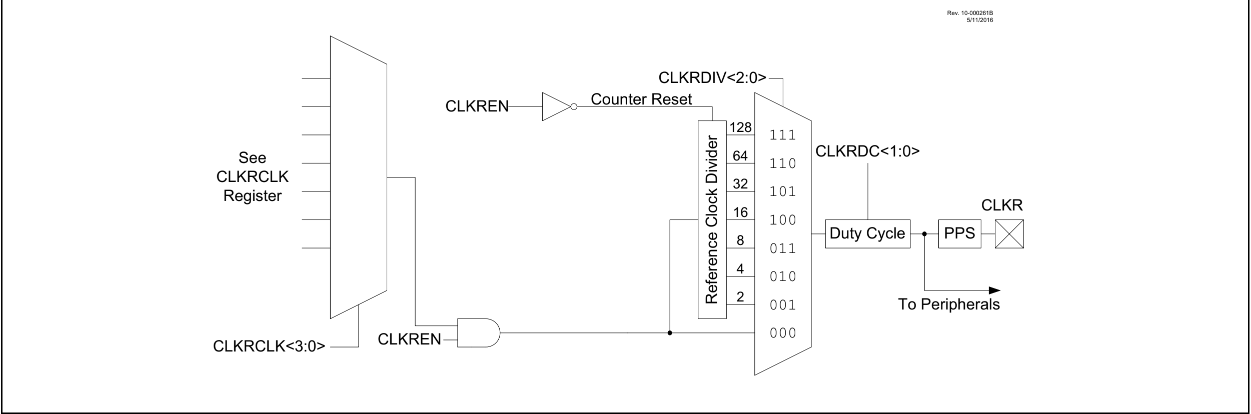
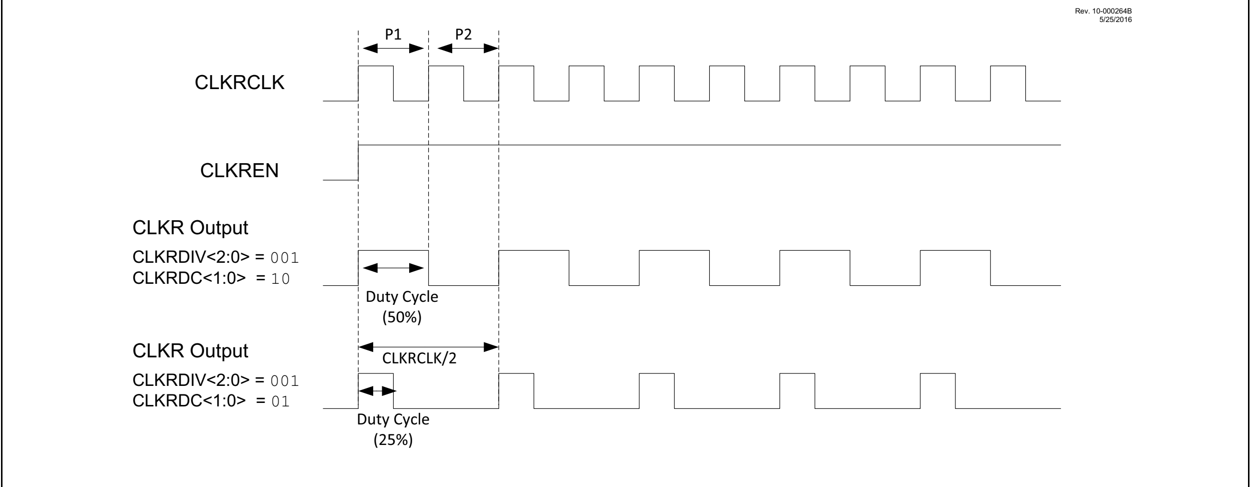


FIGURE 8-2: CLOCK REFERENCE TIMING



REGISTER 9-6: PIR3: PERIPHERAL INTERRUPT REGISTER 3(1)

R/W/HS-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0
TMR0IF	U1IF(2)	U1EIF(3)	U1TXIF(4)	U1RXIF(4)	I2C1EIF(5)	I2C1IF(6)	I2C1TXIF(7)
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 **TMR0IF:** TMR0 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

bit 6 **U1IF:** UART1 Interrupt Flag bit(2)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

bit 5 **U1EIF:** UART1 Framing Error Interrupt Flag bit(3)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

bit 4 **U1TXIF:** UART1 Transmit Interrupt Flag bit(4)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

bit 3 **U1RXIF:** UART1 Receive Interrupt Flag bit(4)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

bit 2 **I2C1EIF:** I^{2}C1 Error Interrupt Flag bit(5)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

bit 1 **I2C1IF:** I^{2}C1 Interrupt Flag bit(6)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

bit 0 **I2C1TXIF:** I^{2}C1 Transmit Interrupt Flag bit(7)
1 = Interrupt has occurred
0 = Interrupt event has not occurred

Note 1: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

2: UxIF is a read-only bit. To clear the interrupt condition, all bits in the UxUIR register must be cleared.

3: UxEIF is a read-only bit. To clear the interrupt condition, all bits in the UxERRIR register must be cleared.

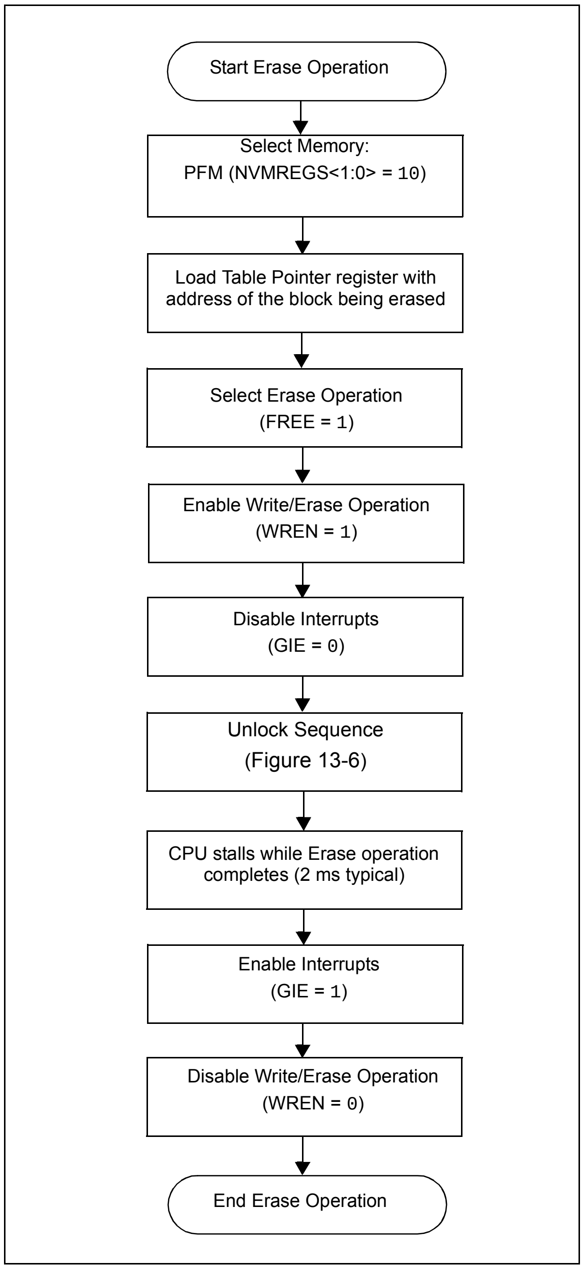
4: UxTXIF and UxRXIF are read-only bits and cannot be set/cleared by the software.

5: I2CxEIF is a read-only bit. To clear the interrupt condition, all bits in the I2CxERR register must be cleared.

6: I2CxIF is a read-only bit. To clear the interrupt condition, all bits in the I2CxPIR register must be cleared.

7: I2CxTXIF and I2CxRXIF are read-only bits. To clear the interrupt condition, the CLRBF bit in I2CxSTAT1 register must be set.

FIGURE 13-7: PFM ROW ERASE FLOWCHART


Start Erase Operation
Select Memory: PFM (NVMREGS<1:0> = 10)
Load Table Pointer register with address of the block being erased
Select Erase Operation (FREE = 1)
Enable Write/Erase Operation (WREN = 1)
Disable Interrupts (GIE = 0)
Unlock Sequence (Figure 13-6)
CPU stalls while Erase operation completes (2 ms typical)
Enable Interrupts (GIE = 1)
Disable Write/Erase Operation (WREN = 0)
End Erase Operation

13.1.6 WRITING TO PROGRAM FLASH MEMORY

The programming write block size is described in Table 5-4. Word or byte programming is not supported. Table writes are used internally to load the holding registers needed to program the memory. There are only as many holding registers as there are bytes in a write block. Refer to Table 5-4 for write latch size.

Since the table latch (TABLAT) is only a single byte, the TBLWT instruction needs to be executed multiple times for each programming operation. The write protection state is ignored for this operation. All of the table write operations will essentially be short writes because only the holding registers are written. NVMIF is not affected while writing to the holding registers.

After all the holding registers have been written, the programming operation of that block of memory is started by configuring the NVMCON1 register for a program memory write and performing the long write sequence.

If the PFM address in the TBLPTR is write-protected or if TBLPTR points to an invalid location, the WR bit is cleared without any effect and the WREER is signaled.

The long write is necessary for programming the program memory. CPU operation is suspended during a long write cycle and resumes when the operation is complete. The long write operation completes in one instruction cycle. When complete, WR is cleared in hardware and NVMIF is set and an interrupt will occur if NVMIE is also set. The latched data is reset to all '1s'. WREN is not changed.

The internal programming timer controls the write time. The write/erase voltages are generated by an on-chip charge pump, rated to operate over the voltage range of the device.

Note: The default value of the holding registers on device Resets and after write operations is FFh. A write of FFh to a holding register does not modify that byte. This means that individual bytes of program memory may be modified, provided that the change does not attempt to change any bit from a '0' to a '1'. When modifying individual bytes, it is not necessary to load all holding registers before executing a long write operation.

15.5.1.2 Hardware Trigger, SIRQ

A Hardware trigger is an interrupt request from another module sent to the DMA with the purpose of starting a DMA message. The DMA start trigger source is user selectable using the DMAxSIRQ register.

The SIRQEN bit (DMAxCON0 register) is used to enable sampling of external interrupt triggers by which a DMA transfer can be started. When set the DMA will sample the selected Interrupt source and when cleared, the DMA will ignore the selected Interrupt source. Clearing SIRQEN does not stop a DMA transaction currently progress, it only stops more hardware request signals from being received.

15.5.2 STOPPING DMA MESSAGE TRANSFERS

The DMA controller can stop data transactions by either of the following two conditions:

1. Clearing the DGO bit
2. Hardware trigger, AIRQ
3. Source Count reload
4. Destination Count reload
5. Clearing the Enable bit

15.5.2.1 User Software Control

If the user clears the DGO bit, the message will be stopped and the DMA will remain in the current configuration.

For example, if the user clears the DGO bit after source data has been read but before it is written to the destination, then the data in DMAxBUF will not reach its destination.

This is also referred to as a soft-stop as the operation can resume if desired by setting DGO bit again.

15.5.2.2 Hardware Trigger, AIRQ

The AIRQEN bit (DMAxCON0 register) is used to enable sampling of external interrupt triggers by which a DMA transaction can be aborted.

Once an Abort interrupt request has been received, the DMA will perform a soft-stop by clearing the DGO bit as well as clearing the SIRQEN bit so overruns do not occur. The AIRQEN bit is also cleared to prevent additional abort signals from triggering false aborts.

If desired, the DGO bit can be set again and the DMA will resume operation from where it left off after the soft-stop had occurred as none of the DMA state information is changed in the event of an abort.

15.5.2.3 Source Count Reload

A DMA message is considered to be complete when the Source count register is decremented from 1 and then reloaded (i.e., once the last byte from either the source read or destination write has occurred). When the SSTP bit is set (DMAxCON1 register) and the source count register is reloaded then further message transfer is stopped.

15.5.2.4 Destination Count Reload

A DMA message is considered to be complete when the Destination count register is decremented from 1 and then reloaded (i.e., once the last byte from either the source read or destination write has occurred). When the DSTP bit is set (DMAxCON1) and the destination count register is reloaded then further message transfer is stopped.

Note:	Reading the DMAxSCNT or DMAxDCNT registers will never return zero. When either register is decremented from '1' it is immediately reloaded from the corresponding size register.

15.5.2.5 Clearing the Enable bit

If the User clears the EN bit, the message will be stopped and the DMA will return to its default configuration. This is also referred to as a hard-stop as the DMA cannot resume operation from where it was stopped.

Note:	After the DMA message transfer is stopped, it requires an extra instruction cycle before the Stop condition takes effect. Thus, after the Stop condition has occurred, a Source read or a Destination write can occur depending on the Source or Destination Bus availability.

15.5.3 DISABLE DMA MESSAGES TRANSFERS UPON COMPLETION

Once the DMA message is complete it may be desirable to disable the trigger source to prevent overrun or under run of data. This can be done by either of the following methods:

1. Clearing the SIRQEN bit
2. Setting the SSTP bit
3. Setting the DSTP bit

REGISTER 19-3: PMD2: PMD CONTROL REGISTER 2

U-0	R/W-0/0	R/W-0/0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
—	DACMD	ADCMD	—	—	CMP2MD	CMP1MD	ZCDMD(1)
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 **DACMD:** Disable DAC bit
1 = DAC module disabled
0 = DAC module enabled

bit 5 **ADCMD:** Disable ADCC bit
1 = ADCC module disabled
0 = ADCC module enabled

bit 4-3 **Unimplemented:** Read as ‘0’

bit 2 **CMP2MD:** Disable Comparator CMP2 bit
1 = CMP2 module disabled
0 = CMP2 module enabled

bit 1 **CMP1MD:** Disable Comparator CMP1 bit
1 = CMP1 module disabled
0 = CMP1 module enabled

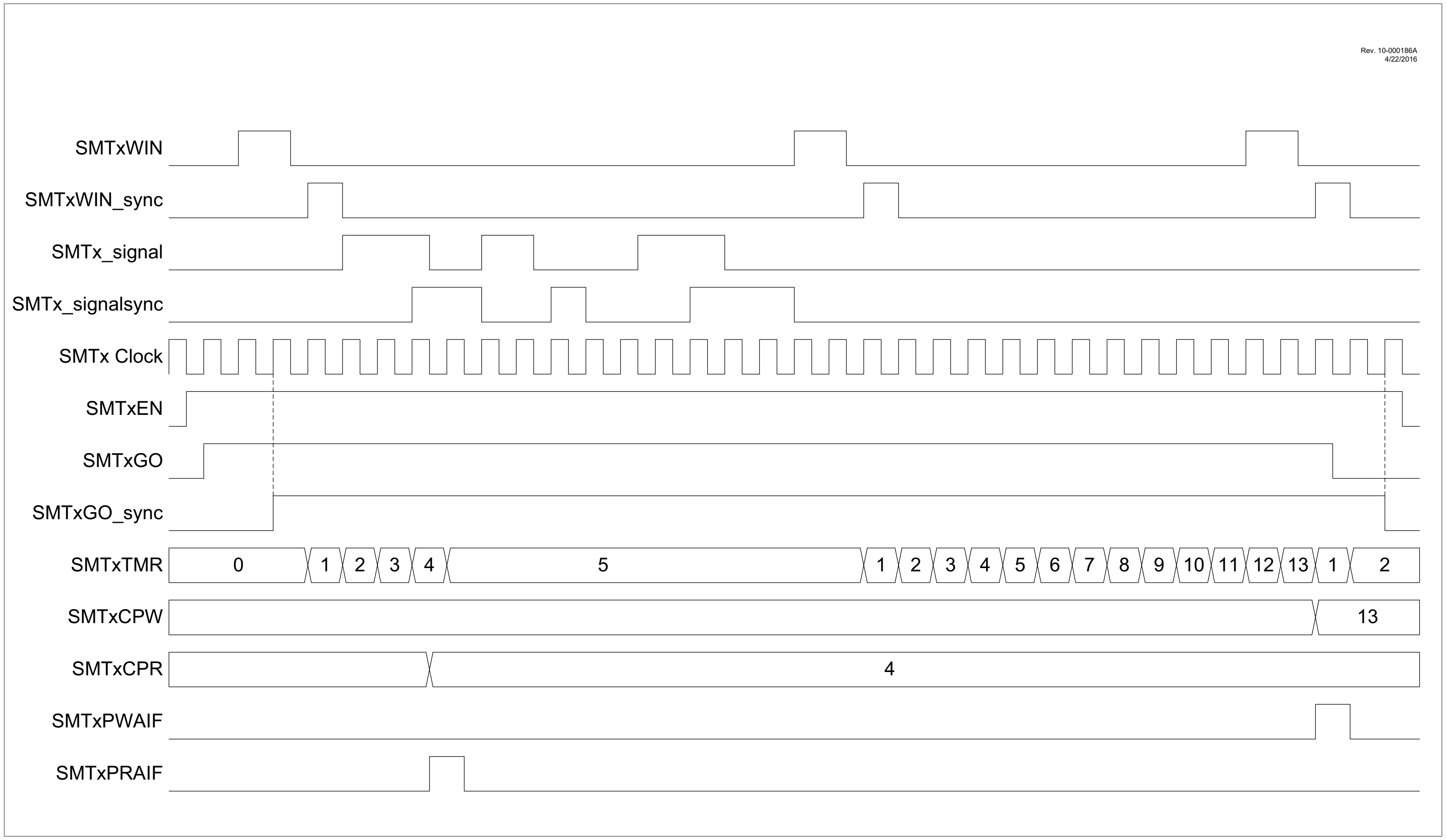
bit 0 **ZCDMD:** Disable Zero-Cross Detect module bit(1)
1 = ZCD module disabled
0 = ZCD module enabled

Note 1: Subject to $\overline{\text{ZCD}}$ bit in CONFIG2H.

25.6.6 GATED WINDOWED MEASURE MODE

This mode measures the duty cycle of the SMTx_signal input over a known input window. It does so by incrementing the timer on each pulse of the clock signal while the SMTx_signal input is high, updating the SMTxCPR register and resetting the timer on every rising edge of the SMTWINx input after the first. See Figure 25-12 and Figure 25-13.

FIGURE 25-14: TIME OF FLIGHT MODE REPEAT ACQUISITION TIMING DIAGRAM



REGISTER 25-10: SMTxCPRL: SMT CAPTURED PERIOD 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
SMTxCPR<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 **SMTxCPR<7:0>**: Significant bits of the SMT Period Latch – Low Byte

REGISTER 25-11: SMTxCPRH: SMT CAPTURED PERIOD 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
SMTxCPR<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 **SMTxCPR<15:8>**: Significant bits of the SMT Period Latch – High Byte

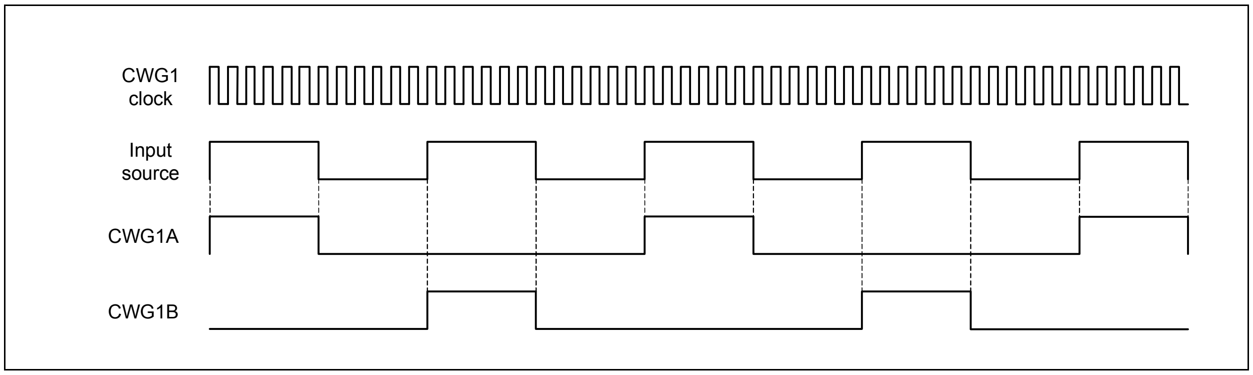
REGISTER 25-12: SMTxCPRU: SMT CAPTURED PERIOD REGISTER – UPPER BYTE

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

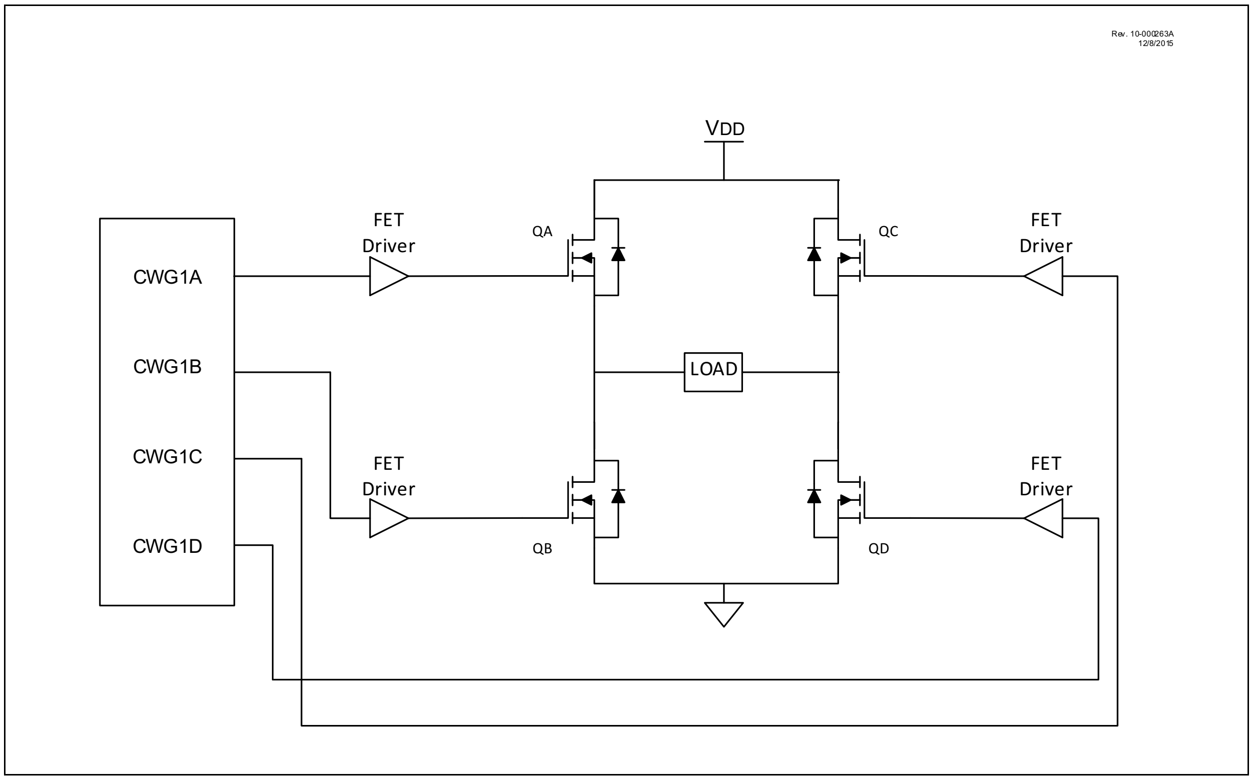
FIGURE 26-4: CWGx PUSH-PULL MODE OPERATION



26.2.3 FULL-BRIDGE MODES

In Forward and Reverse Full-Bridge modes, three outputs drive static values while the fourth is modulated by the input data signal. The mode selection may be toggled between forward and reverse by toggling the MODE<0> bit of the CWGxCON0 while keeping MODE<2:1> static, without disabling the CWG module. When connected as shown in Figure 26-5, the outputs are appropriate for a full-bridge motor driver. Each CWG output signal has independent polarity control, so the circuit can be adapted to high-active and low-active drivers. A simplified block diagram for the Full-Bridge modes is shown in Figure 26-6.

FIGURE 26-5: EXAMPLE OF FULL-BRIDGE APPLICATION



REGISTER 26-3: CWGxCLK: CWGx CLOCK INPUT SELECTION REGISTER

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0/0
—	—	—	—	—	—	—	CS
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 7-1 **Unimplemented:** Read as ‘0’

bit 0 **CS:** CWG Clock Source Selection Select bits

CS	CWG1	CWG2	CWG3
1	HFINTOSC [(1)]	HFINTOSC [(1)]	HFINTOSC [(1)]
0	FOSC	FOSC	FOSC

Note 1: HFINTOSC remains operating during Sleep.

REGISTER 27-9: CLCxGLS2: GATE 2 LOGIC SELECT REGISTER

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
G3D4T	G3D4N	G3D3T	G3D3N	G3D2T	G3D2N	G3D1T	G3D1N
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	**G3D4T:** Gate 2 Data 4 True (noninverted) bit 1 = CLCIN3 (true) is gated into CLCx Gate 2 0 = CLCIN3 (true) is not gated into CLCx Gate 2
bit 6	**G3D4N:** Gate 2 Data 4 Negated (inverted) bit 1 = CLCIN3 (inverted) is gated into CLCx Gate 2 0 = CLCIN3 (inverted) is not gated into CLCx Gate 2
bit 5	**G3D3T:** Gate 2 Data 3 True (noninverted) bit 1 = CLCIN2 (true) is gated into CLCx Gate 2 0 = CLCIN2 (true) is not gated into CLCx Gate 2
bit 4	**G3D3N:** Gate 2 Data 3 Negated (inverted) bit 1 = CLCIN2 (inverted) is gated into CLCx Gate 2 0 = CLCIN2 (inverted) is not gated into CLCx Gate 2
bit 3	**G3D2T:** Gate 2 Data 2 True (noninverted) bit 1 = CLCIN1 (true) is gated into CLCx Gate 2 0 = CLCIN1 (true) is not gated into CLCx Gate 2
bit 2	**G3D2N:** Gate 2 Data 2 Negated (inverted) bit 1 = CLCIN1 (inverted) is gated into CLCx Gate 2 0 = CLCIN1 (inverted) is not gated into CLCx Gate 2
bit 1	**G3D1T:** Gate 2 Data 1 True (noninverted) bit 1 = CLCIN0 (true) is gated into CLCx Gate 2 0 = CLCIN0 (true) is not gated into CLCx Gate 2
bit 0	**G3D1N:** Gate 2 Data 1 Negated (inverted) bit 1 = CLCIN0 (inverted) is gated into CLCx Gate 2 0 = CLCIN0 (inverted) is not gated into CLCx Gate 2

31.17.1 AUTO-BAUD DETECT

The UART module supports automatic detection and calibration of the baud rate in the 8-bit Asynchronous and LIN modes. However, setting ABDEN to start auto-baud detection is neither necessary, nor possible in LIN mode because that mode supports auto-baud detection automatically at the beginning of every data packet. Enabling auto-baud detect with the ABDEN bit applies to the Asynchronous modes only.

When Auto-Baud Detect (ABD) is active, the clock to the BRG is reversed. Rather than the BRG clocking the incoming RX signal, the RX signal is timing the BRG. The Baud Rate Generator is used to time the period of a received 55h (ASCII “U”), which is the Sync character for the LIN bus. The unique feature of this character is that it has five falling edges, including the Start bit edge, five rising edges including the Stop bit edge.

In 8-bit Asynchronous mode, setting the ABDEN bit in the UxCON0 register enables the auto-baud calibration sequence. The first falling edge of the RX input after ABDEN is set will start the auto-baud calibration sequence. While the ABD sequence takes place, the UART state machine is held in idle. On the first falling edge of the receive line, the UxBRG begins counting up using the BRG counter clock as shown in Figure 31-12. The fifth falling edge will occur on the RX pin at the beginning of the bit 7 period. At that time, an accumulated value totaling the proper BRG period is left in the UxBRGH, UxBRGL register pair, the ABDEN bit is automatically cleared and the ABDIF interrupt flag is set. ABDIF must be cleared by software.

RXIDL indicates that the sync input is active. RXIDL will go low on the first falling edge and go high on the fifth rising edge.

The BRG auto-baud clock is determined by the BRGS bit as shown in Table 31-2. During ABD, the internal BRG register is used as a 16-bit counter. However, the UxBRGH and UxBRGL registers retain the previous BRG value until the auto-baud process is successfully completed. While calibrating the baud rate period, the internal BRG register is clocked at 1/8th the BRG base clock rate. The resulting byte measurement is the average bit time when clocked at full speed and is transferred to the UxBRGH and UxBRGL registers when complete.

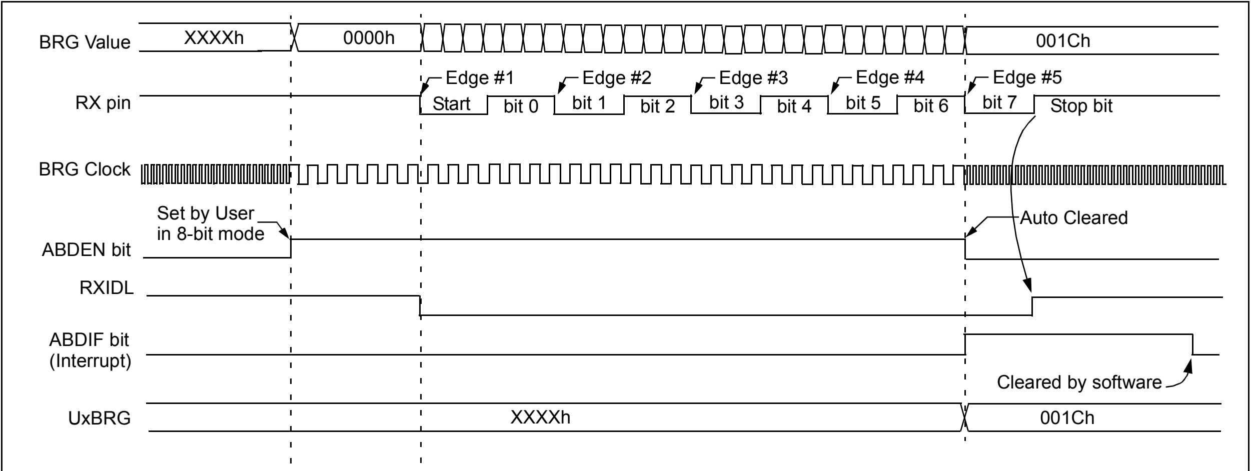
Note 1: If the WUE bit is set with the ABDEN bit, auto-baud detection will occur on the byte following the Break character (see **Section 31.17.3 “Auto-Wake-up on Break”**).

2: It is up to the user to determine that the incoming character baud rate is within the range of the selected BRG clock source. Some combinations of oscillator frequency and UART baud rates are not possible.

TABLE 31-2: BRG COUNTER CLOCK RATES

BRGS	BRG Base Clock	BRG ABD Clock
1	FOSC/4	FOSC/32
0	FOSC/16	FOSC/128

FIGURE 31-12: AUTOMATIC BAUD RATE CALIBRATION



Note 1: Auto-baud is supported in LIN and 8-bit Asynchronous modes only.

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

34.0 CAN MODULE

This family of devices contain a Controller Area Network (CAN) module. The CAN module is fully backwards-compatible with the CAN and ECAN modules found in older PIC18 devices.

The Controller Area Network (CAN) module is a serial interface which is useful for communicating with other peripherals or microcontroller devices. This interface, or protocol, was designed to allow communications within noisy environments.

The CAN module is a communication controller, implementing the CAN 2.0A or B protocol as defined in the BOSCH specification. The module will support CAN 1.2, CAN 2.0A, CAN 2.0B Passive and CAN 2.0B Active versions of the protocol. The module implementation is a full CAN system; however, the CAN specification is not covered within this data sheet. Refer to the BOSCH CAN specification for further details.

The module features are as follows:

- Implementation of the CAN protocol, CAN 1.2, CAN 2.0A and CAN 2.0B
- DeviceNet™ data bytes filter support
- Standard and extended data frames
- 0-8 bytes data length
- Programmable bit rate up to 1 Mbit/sec
- Fully backward compatible with CAN modules on older PIC18 devices
- Three modes of operation:
  - Mode 0 – Legacy mode
  - Mode 1 – Enhanced Legacy mode with DeviceNet support
  - Mode 2 – FIFO mode with DeviceNet support
- Support for remote frames with automated handling
- Double-buffered receiver with two prioritized received message storage buffers
- Six buffers programmable as RX and TX message buffers
- 16 full (standard/extended identifier) acceptance filters that can be linked to one of four masks
- Two full acceptance filter masks that can be assigned to any filter
- One full acceptance filter that can be used as either an acceptance filter or acceptance filter mask
- Three dedicated transmit buffers with application specified prioritization and abort capability
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loopback mode supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to timer module for time-stamping and network synchronization
- Low-power Sleep mode

34.1 Module Overview

The CAN bus module consists of a protocol engine and message buffering and control. The CAN protocol engine automatically handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate data registers. Status and errors can be checked by reading the appropriate registers. Any message detected on the CAN bus is checked for errors and then matched against filters to see if it should be received and stored in one of the two receive registers.

The CAN module supports the following frame types:

- Standard Data Frame
- Extended Data Frame
- Remote Frame
- Error Frame
- Overload Frame Reception

The CANRX input pin is selected with the CANRXPPS register. The CANTX output pin is selected with each pin's RxyPPS register.

Note: The CANRX pin defaults to pin RB3, but the CANTX has no default location and must be assigned to a pin before CAN transmissions can occur.

In Normal mode, the user must ensure that the appropriate TRIS bit for CANRX is set and the appropriate TRIS bit for CANRX is cleared. In addition, the appropriate ANSEL bit for CANRX must be cleared to disable the analog input buffer.

Note: Unlike older Microchip devices with CAN functionality, the CAN pins can be mapped to pins with analog functionality. Ensure that the analog functionality on the CANRX pin is disabled, or the CAN module will not properly function.

34.1.1 MODULE FUNCTIONALITY

The CAN bus module consists of a protocol engine, message buffering and control (see Figure 34-1). The protocol engine can best be understood by defining the types of data frames to be transmitted and received by the module.

The following sequence illustrates the necessary initialization steps before the CAN module can be used to transmit or receive a message. Steps can be added or removed depending on the requirements of the application.

1. Use the CANRXPPS and appropriate RxyPPS registers to map the CANRX and CANTX functions to the desired pins of the device.
2. Initialize LAT, TRIS and ANSEL bits for the selected CANRX and CANTX pins.
3. Ensure that the CAN module is in Configuration mode.
4. Select CAN Functional mode.

REGISTER 34-24: BnSIDH: TX/RX BUFFER 'n' STANDARD IDENTIFIER REGISTERS, HIGH BYTE IN RECEIVE MODE [0 ≤ n ≤ 5, TXnEN (BSEL0<n>) = 0](1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
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 **SID<10:3>:** Standard Identifier bits (if EXIDE (BnSIDL<3>) = 0)
Extended Identifier bits, EID<28:21> (if EXIDE = 1).

Note 1: These registers are available in Mode 1 and 2 only.

REGISTER 34-25: BnSIDH: TX/RX BUFFER 'n' STANDARD IDENTIFIER REGISTERS, HIGH BYTE IN TRANSMIT MODE [0 ≤ n ≤ 5, TXnEN (BSEL0<n>) = 1](1)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
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 **SID<10:3>:** Standard Identifier bits (if EXIDE (BnSIDL<3>) = 0)
Extended Identifier bits, EID<28:21> (if EXIDE = 1).

Note 1: These registers are available in Mode 1 and 2 only.

37.0 ANALOG-TO-DIGITAL CONVERTER WITH COMPUTATION (ADC^2) MODULE

The Analog-to-Digital Converter with Computation (ADC^2) allows conversion of an analog input signal to a 12-bit binary representation of that signal. This device uses analog inputs, which are multiplexed into a single sample and hold circuit. The output of the sample and hold is connected to the input of the converter. The converter generates a 12-bit binary result via successive approximation and stores the conversion result into the ADC result registers (ADRESH:ADRESL register pair).

Additionally, the following features are provided within the ADC module:

- 13-bit Acquisition Timer
- Hardware Capacitive Voltage Divider (CVD) support:
  - 13-bit Precharge Timer
  - Adjustable sample and hold capacitor array
  - Guard ring digital output drive
- Automatic repeat and sequencing:
  - Automated double sample conversion for CVD
  - Two sets of result registers (Result and Previous result)
  - Auto-conversion trigger
  - Internal retrigger
- Computation features:
  - Averaging and Low-Pass Filter functions
  - Reference Comparison
  - 2-level Threshold Comparison
  - Selectable Interrupts

Figure 37-1 shows the block diagram of the ADC.

The ADC voltage reference is software selectable to be either internally generated or externally supplied.

The ADC can generate an interrupt upon completion of a conversion and upon threshold comparison. These interrupts can be used to wake up the device from Sleep.

REGISTER 37-13: ADCAP: ADC ADDITIONAL SAMPLE CAPACITOR SELECTION REGISTER

U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	ADCAP<4:0>				
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 7-5 **Unimplemented**: Read as ‘0’

bit 4-0 **ADCAP<4:0>**: ADC Additional Sample Capacitor Selection bits
`11111` = 31 pF
`11110` = 30 pF
`11101` = 29 pF
•
•
•
`00011` = 3 pF
`00010` = 2 pF
`00001` = 1 pF
`00000` = No additional capacitance

REGISTER 37-14: ADRPT: ADC REPEAT SETTING 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
RPT<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 **RPT<7:0>**: ADC Repeat Threshold bits
Counts the number of times that the ADC has been triggered and is used along with CNT to determine when the error threshold is checked when the computation is Low-pass Filter, Burst Average, or Average modes. See Table 37-2 for more details.

TABLE 43-1: REGISTER FILE SUMMARY FOR PIC18(L)F25/26K83 DEVICES (CONTINUED)

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
3CFCh	MD1SRC	—	—	—	MS					458
3CFBh	MD1CON1	—	—	CHPOL	CHSYNC	—	—	CLPOL	CLSYNC	456
3CFAh	MD1CON0	EN	—	OUT	OPOL	—	—	—	BIT	455
3CF9h - 3CE7h	—	Unimplemented								—
3CE6h	CLKRCLK	—	—	—	—	CLK				104
3CE5h	CLKRCON	EN	—	—	DC		DIV			103
3CE4h - 3C7Fh	—	Unimplemented								—
3C7Eh	CLCDATA0	—	—	—	—	CLC4OUT	CLC3OUT	CLC2OUT	CLC1OUT	433
3C7Dh	CLC1GLS3	G4D4T	G4D4N	G4D3T	G4D3N	G4D2T	G4D2N	G4D1T	G4D1N	432
3C7Ch	CLC1GLS2	G3D4T	G3D4N	G3D3T	G3D3N	G3D2T	G3D2N	G3D1T	G3D1N	431
3C7Bh	CLC1GLS1	G2D4T	G2D4N	G2D3T	G2D3N	G2D2T	G2D2N	G2D1T	G2D1N	430
3C7Ah	CLC1GLS0	G1D4T	G1D4N	G1D3T	G1D3N	G1D2T	G1D2N	G1D1T	G1D1N	429
3C79h	CLC1SEL3	D4S								428
3C78h	CLC1SEL2	D3S								428
3C77h	CLC1SEL1	D2S								428
3C76h	CLC1SEL0	D1S								428
3C75h	CLC1POL	POL	—	—	—	G4POL	G3POL	G2POL	G1POL	427
3C74h	CLC1CON	EN	OE	OUT	INTP	INTN	MODE			426
3C73h	CLC2GLS3	G4D4T	G4D4N	G4D3T	G4D3N	G4D2T	G4D2N	G4D1T	G4D1N	432
3C72h	CLC2GLS2	G3D4T	G3D4N	G3D3T	G3D3N	G3D2T	G3D2N	G3D1T	G3D1N	431
3C71h	CLC2GLS1	G2D4T	G2D4N	G2D3T	G2D3N	G2D2T	G2D2N	G2D1T	G2D1N	430
3C70h	CLC2GLS0	G1D4T	G1D4N	G1D3T	G1D3N	G1D2T	G1D2N	G1D1T	G1D1N	429
3C6Fh	CLC2SEL3	D4S								428
3C6Eh	CLC2SEL2	D3S								428
3C6Dh	CLC2SEL1	D2S								428
3C6Ch	CLC2SEL0	D1S								428
3C6Bh	CLC2POL	POL	—	—	—	G4POL	G3POL	G2POL	G1POL	427
3C6Ah	CLC2CON	EN	OE	OUT	INTP	INTN	MODE			426
3C69h	CLC3GLS3	G4D4T	G4D4N	G4D3T	G4D3N	G4D2T	G4D2N	G4D1T	G4D1N	432
3C68h	CLC3GLS2	G3D4T	G3D4N	G3D3T	G3D3N	G3D2T	G3D2N	G3D1T	G3D1N	431
3C67h	CLC3GLS1	G2D4T	G2D4N	G2D3T	G2D3N	G2D2T	G2D2N	G2D1T	G2D1N	430
3C66h	CLC3GLS0	G1D4T	G1D4N	G1D3T	G1D3N	G1D2T	G1D2N	G1D1T	G1D1N	429
3C65h	CLC3SEL3	D4S								428
3C64h	CLC3SEL2	D3S								428
3C63h	CLC3SEL1	D2S								428
3C62h	CLC3SEL0	D1S								429
3C61h	CLC3POL	POL	—	—	—	G4POL	G3POL	G2POL	G1POL	427
3C60h	CLC3CON	EN	OE	OUT	INTP	INTN	MODE			426
3C5Fh	CLC4GLS3	G4D4T	G4D4N	G4D3T	G4D3N	G4D2T	G4D2N	G4D1T	G4D1N	432
3C5Eh	CLC4GLS2	G3D4T	G3D4N	G3D3T	G3D3N	G3D2T	G3D2N	G3D1T	G3D1N	431
3C5Dh	CLC4GLS1	G2D4T	G2D4N	G2D3T	G2D3N	G2D2T	G2D2N	G2D1T	G2D1N	430
3C5Ch	CLC4GLS0	G1D4T	G1D4N	G1D3T	G1D3N	G1D2T	G1D2N	G1D1T	G1D1N	429
3C5Bh	CLC4SEL3	D4S								428
3C5Ah	CLC4SEL2	D3S								428
3C59h	CLC4SEL1	D2S								428
3C58h	CLC4SEL0	D1S								429
3C57h	CLC4POL	POL	—	—	—	G4POL	G3POL	G2POL	G1POL	427
3C56h	CLC4CON	EN	OE	OUT	INTP	INTN	MODE			426

Legend: x = unknown, u = unchanged, — = unimplemented, q = value depends on condition
Note 1: Not present in LF devices.