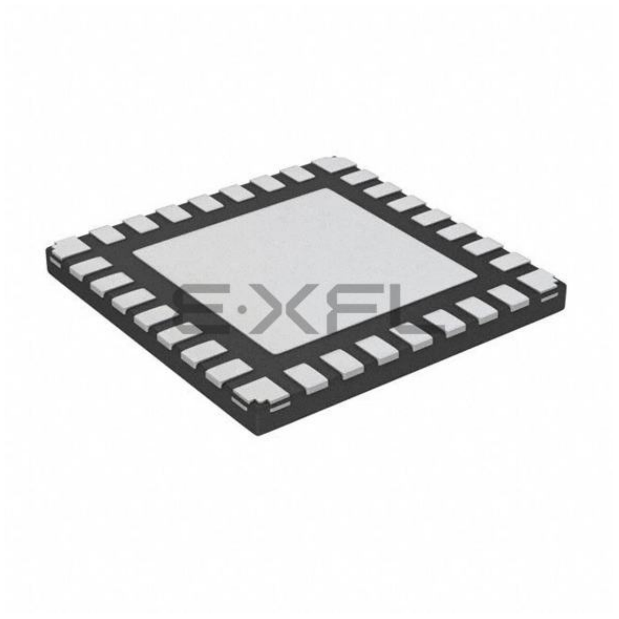


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

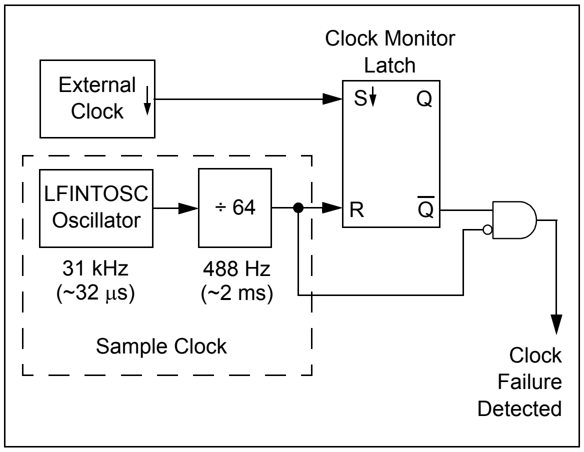
Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	64MHz
Connectivity	I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, HLVD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	64KB (64K x 8)
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 ~ 125°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/pic18lf26k42-e-mx

7.4 Fail-Safe Clock Monitor

The Fail-Safe Clock Monitor (FSCM) allows the device to continue operating should the external oscillator fail. The FSCM is enabled by setting the FCMEN bit in the Configuration Words. The FSCM is applicable to all external Oscillator modes (LP, XT, HS, ECL/M/H and Secondary Oscillator).

FIGURE 7-9: FSCM BLOCK DIAGRAM



7.4.1 FAIL-SAFE DETECTION

The FSCM module detects a failed oscillator by comparing the external oscillator to the FSCM sample clock. The sample clock is generated by dividing the LFINTOSC by 64. See Figure 7-9. Inside the fail detector block is a latch. The external clock sets the latch on each falling edge of the external clock. The sample clock clears the latch on each rising edge of the sample clock. A failure is detected when an entire half-cycle of the sample clock elapses before the external clock goes low.

7.4.2 FAIL-SAFE OPERATION

When the external clock fails, the FSCM overwrites the COSC bits to select HFINTOSC (`3'b110`). The frequency of HFINTOSC would be determined by the previous state of the FRQ bits and the NDIV/CDIV bits. The bit flag OSFIF of the respective PIR register is set. Setting this flag will generate an interrupt if the OSFIE bit of the respective PIR register is also set. The device firmware can then take steps to mitigate the problems that may arise from a failed clock. The system clock will continue to be sourced from the internal clock source until the device firmware successfully restarts the external oscillator and switches back to external operation, by writing to the NOSC and NDIV bits of the OSCCON1 register.

7.4.3 FAIL-SAFE CONDITION CLEARING

The Fail-Safe condition is cleared after a Reset, executing a `SLEEP` instruction or changing the NOSC and NDIV bits of the OSCCON1 register. When switching to the external oscillator or PLL, the OST is restarted. While the OST is running, the device continues to operate from the INTOSC selected in OSCCON1. When the OST times out, the Fail-Safe condition is cleared after successfully switching to the external clock source. The OSCFIF bit should be cleared prior to switching to the external clock source. If the Fail-Safe condition still exists, the OSCFIF flag will again become set by hardware.

9.6 Returning from Interrupt Service Routine (ISR)

The "Return from Interrupt" instruction (`RETFIE`) is used to mark the end of an ISR.

When `RETFIE 1` instruction is executed, the PC is loaded with the saved PC value from the top of the PC stack. Saved context is also restored with the execution of this instruction. Thus, execution returns to the previous state of operation that existed before the interrupt occurred.

When `RETFIE 0` instruction is executed, the saved context is not restored back to the registers.

9.7 Interrupt Latency

By assigning each interrupt with a vector address/ number (MVECEN = `1`), scanning of all interrupts is not necessary to determine the source of the interrupt.

When MVECEN = `1`, Vectored interrupt controller requires three clock cycles to vector to the ISR from main routine, thereby removing dependency of interrupt timing on compiled code.

There is a fixed latency of three instruction cycles between the completion of the instruction active when the interrupt occurred and the first instruction of the Interrupt Service Routine. Figure 9-7, Figure 9-8 and Figure 9-9 illustrate the sequence of events when a peripheral interrupt is asserted when the last executed instruction is one-cycle, two-cycle and three-cycle respectively, when MVECEN = `1`.

After the Interrupt Flag Status bit is set, the current instruction completes executing. In the first latency cycle, the contents of the PC, STATUS, WREG, BSR, FSR0/1/2, PRODL/H and PCLATH/U registers are context saved and the IVTBASE+ Vector number is calculated. In the second latency cycle, the PC is loaded with the calculated vector table address for the interrupt source and the starting address of the ISR is fetched. In the third latency cycle, the PC is loaded with the ISR address. All the latency cycles are executed as a `FNOP` instruction.

When MVECEN = `0`, Vectored interrupt controller requires two clock cycles to vector to the ISR from main routine. There is a latency of two instruction cycles plus the software latency between the completion of the instruction active when the interrupt occurred and the first instruction of the Interrupt Service Routine.

REGISTER 9-2: INTCON1: INTERRUPT CONTROL REGISTER 1

R-0/0	R-0/0	U-0	U-0	U-0	U-0	U-0	U-0
STAT<1:0>		—	—	—	—	—	—
bit 7							bit 0

Legend:		
HC = Bit is cleared by hardware		
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 7-6	**STAT<1:0>:** Interrupt State Status bits `11` = High priority ISR executing, high priority interrupt was received while a low priority ISR was executing `10` = High priority ISR executing, high priority interrupt was received in main routine `01` = Low priority ISR executing, low priority interrupt was received in main routine `00` = Main routine executing
bit 5-0	**Unimplemented**: Read as ‘`0`’

REGISTER 9-13: PIR10: PERIPHERAL INTERRUPT REGISTER 10(1)

U-0	U-0	U-0	U-0	U-0	U-0	R/W/HS-0/0	R/W/HS-0/0
—	—	—	—	—	—	CLC4IF	CCP4IF
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 7-2 **Unimplemented:** Read as ‘0’

bit 1 **CLC4IF:** CLC4 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

bit 0 **CCP4IF:** CCP4 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
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.

REGISTER 9-25: IPR0: PERIPHERAL INTERRUPT PRIORITY REGISTER 0

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
IOCIP	CRCIP	SCANIP	NVMIP	CSWIP	OSFIP	HLVDIP	SWIP
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 **IOCIP:** Interrupt-on-Change Priority bit
1 = High priority
0 = Low priority

bit 6 **CRCIP:** CRC Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **SCANIP:** Memory Scanner Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **NVMIP:** NVM Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **CSWIP:** Clock Switch Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **OSFIP:** Oscillator Fail Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **HLVDIP:** HLVD Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **SWIP:** Software Interrupt Priority bit
1 = High priority
0 = Low priority

TABLE 9-3: SUMMARY OF REGISTERS ASSOCIATED WITH INTERRUPTS

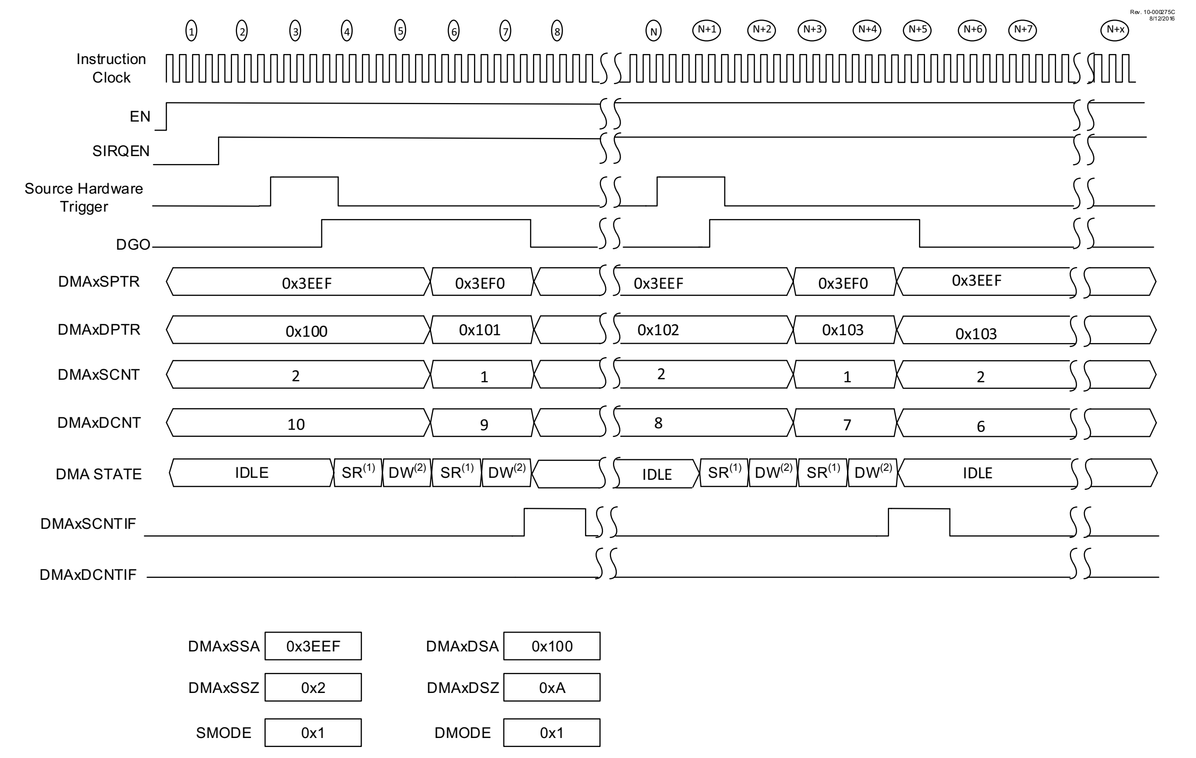
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
INTCON0	GIE/GIEH	GIEL	IPEN	—	—	INT2EDG	INT1EDG	INT0EDG	135
INTCON1	STAT<1:0>		—	—	—	—	—	—	136
PIE0	IOCIE	CRCIE	SCANIE	NVMIE	CSWIE	OSFIE	HLVDIE	SWIE	147
PIE1	SMT1PWAIE	SMT1PRAIE	SMT1IE	C1IE	ADTIE	ADIE	ZCDIE	INT0IE	148
PIE2	I2C1RXIE	SPI1IE	SPI1TXIE	SPI1RXIE	DMA1AIE	DMA1ORIE	DMA1DCNTIE	DMA1SCNTIE	149
PIE3	TMR0IE	U1IE	U1EIE	U1TXIE	U1RXIE	I2C1EIE	I2C1IE	I2C1TXIE	150
PIE4	CLC1IE	CWG1IE	NCO1IE	—	CCP1IE	TMR2IE	TMR1GIE	TMR1IE	151
PIE5	I2C2TXIE	I2C2RXIE	DMA2AIE	DMA2ORIE	DMA2DCNTIE	DMA2SCNTIE	C2IE	INT1IE	152
PIE6	TMR3GIE	TMR3IE	U2IE	U2EIE	U2TXIE	U2RXIE	I2C2EIE	I2C2IE	153
PIE7	—	—	INT2IE	CLC2IE	CWG2IE	—	CCP2IE	TMR4IE	154
PIE8	TMR5GIE	TMR5IE	—	—	—	—	—	—	155
PIE9	—	—	—	—	CLC3IE	CWG3IE	CCP3IE	TMR6IE	155
PIE10	—	—	—	—	—	—	CLC4IE	CCP4IE	156
PIR0	IOCIF	CRCIF	SCANIF	NVMIF	CSWIF	OSFIF	HLVDIF	SWIF	137
PIR1	SMT1PWAIF	SMT1PRAIF	SMT1IF	C1IF	ADTIF	ADIF	ZCDIF	INT0IF	138
PIR2	I2C1RXIF	SPI1IF	SPI1TXIF	SPI1RXIF	DMA1AIF	DMA1ORIF	DMA1DCNTIF	DMA1SCNTIF	139
PIR3	TMR0IF	U1IF	U1EIF	U1TXIF	U1RXIF	I2C1EIF	I2C1IF	I2C1TXIF	140
PIR4	CLC1IF	CWG1IF	NCO1IF	—	CCP1IF	TMR2IF	TMR1GIF	TMR1IF	141
PIR5	I2C2TXF	I2C2RXF	DMA2AIF	DMA2ORIF	DMA2DCNTIF	DMA2SCNTIF	C2IF	INT1IF	142
PIR6	TMR3GIF	TMR3IF	U2IF	U2EIF	U2TXIF	U2RXIF	I2C2EIF	I2C2IF	143
PIR7	—	—	INT2IF	CLC2IF	CWG2IF	—	CCP2IF	TMR4IF	144
PIR8	TMR5GIF	TMR5IF	—	—	—	—	—	—	145
PIR9	—	—	—	—	CLC3IF	CWG3IF	CCP3IF	TMR6IF	145
PIR10	—	—	—	—	—	—	CLC4IF	CCP4IF	146
IPR0	IOCIP	CRCIP	SCANIP	NVMIP	CSWIP	OSFIP	HLVDIP	SWIP	157
IPR1	SMT1PWAIP	SMT1PRAIP	SMT1IP	C1IP	ADTIP	ADIP	ZCDIP	INT0IP	158
IPR2	I2C1RIP	SPI1IP	SPI1TIP	SPI1RIP	DMA1AIP	DMA1ORIP	DMA1DCNTIP	DMA1SCNTIP	159
IPR3	TMR0IP	U1IP	U1EIP	U1TXIP	U1RXIP	I2C1EIP	I2C1IP	I2C1TXIP	160
IPR4	CLC1IP	CWG1IP	NCO1IP	—	CCP1IP	TMR2IP	TMR1GIP	TMR1IP	161
IPR5	I2C2TXP	I2C2RXP	DMA2AIP	DMA2ORIP	DMA2DCNTIP	DMA2SCNTIP	C2IP	INT1IP	162
IPR6	TMR3GIP	TMR3IP	U2IP	U2EIP	U2TXIP	U2RXIP	I2C2EIP	I2C2IP	163
IPR7	—	—	INT2IP	CLC2IP	CWG2IP	—	CCP2IP	TMR4IP	164
IPR8	TMR5GIP	TMR5IP	—	—	—	—	—	—	164
IPR9	—	—	—	—	CLC3IP	CWG3IP	CCP3IP	TMR6IP	165
IPR10	—	—	—	—	—	—	CLC4IP	CCP4IP	165
IVTBASEU	—	—	—	BASE<20:16>					166
IVTBASEH	BASE<15:8>								166
IVTBASEL	BASE<7:0>								166
IVTADU				AD<20:16>					167
IVTADH	AD<15:8>								167
IVTADL	AD<7:0>								167
IVTLOCK	—	—	—	—	—	—	—	IVTLOCKED	168

Legend: — = unimplemented locations, read as '0'. Shaded bits are not used for interrupts.

15.9.4 TRANSFER FROM SFR TO GPR

The following visual reference describes the sequence of events when copying ADC results to a GPR location. The ADC Interrupt Flag can be chosen as the Source Hardware trigger, the Source address can be set to point to the ADC Result registers at 3EEF, the Destination address can be set to point to any GPR location of our choice (Example 0x100).

FIGURE 15-8: SFR SPACE TO GPR SPACE TRANSFER



Note 1: SR - Source Read

2: DW - Destination Write

REGISTER 16-2: TRISx: TRI-STATE CONTROL REGISTER

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0
bit 7							bit 0

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

bit 7-0 **TRISx<7:0>:** TRISx Port I/O Tri-state Control bits
1 = Port output driver is disabled
0 = Port output driver is enabled

TABLE 16-3: TRIS REGISTERS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
TRISB	TRISB7(1)	TRISB6(1)	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
TRISD(2)	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
TRISE(2)	—	—	—	—	—	TRISE2	TRISE1	TRISE0
TRISF(3)	TRISF7	TRISF6	TRISF5	TRISF4	TRISF3	TRISF2	TRISF1	TRISF0

Note 1: Bits RB6 and RB7 read ‘1’ while in Debug mode.
2: Unimplemented in PIC18(L)F26/27K42.
3: Unimplemented in PIC18(L)F26/45/46/47K42.

TABLE 16-11: SUMMARY OF REGISTERS ASSOCIATED WITH I/O

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	263
PORTB	RB7(1)	RB6(1)	RB5	RB4	RB3	RB2	RB1	RB0	263
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	263
PORTD(6)	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	263
PORTE	—	—	—	—	RE3(2)	RE2(6)	RE1(6)	RE0(6)	263
PORTF(7)	RF7	RF6	RF5	RF4	RF3	RF2	RF1	RF0	263
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	264
TRISB	TRISB7(3)	TRISB6(3)	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	264
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	264
TRISD(6)	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	264
TRISE(6)	—	—	—	—	—	TRISE2	TRISE1	TRISE0	264
TRISF(7)	TRISF7	TRISF6	TRISF5	TRISF4	TRISF3	TRISF2	TRISF1	TRISF0	264
LATA	LATA7	LATA6	LATA5	LATA4	LATA3	LATA2	LATA1	LATA0	265
LATB	LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0	265
LATC	LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0	265
LATD(6)	LATD7	LATD6	LATD5	LATD4	LATD3	LATD2	LATD1	LATD0	265
LATE(6)	—	—	—	—	—	LATE2	LATE1	LATE0	265
LATF(7)	LATF7	LATF6	LATF5	LATF4	LATF3	LATF2	LATF1	LATF0	265
ANSELA	ANSELA7	ANSELA6	ANSELA5	ANSELA4	ANSELA3	ANSELA2	ANSELA1	ANSELA0	266
ANSELB	ANSELB7	ANSELB6	ANSELB5	ANSELB4	ANSELB3	ANSELB2	ANSELB1	ANSELB0	266
ANSELC	ANSELC7	ANSELC6	ANSELC5	ANSELC4	ANSELC3	ANSELC2	ANSELC1	ANSELC0	266
ANSELD(6)	ANSELD7	ANSELD6	ANSELD5	ANSELD4	ANSELD3	ANSELD2	ANSELD1	ANSELD0	266
ANSELE(6)	—	—	—	—	—	ANSELE2	ANSELE1	ANSELE0	266
ANSELF(7)	ANSELF7	ANSELF6	ANSELF5	ANSELF4	ANSELF3	ANSELF2	ANSELF1	ANSELF0	266
WPUA	WPUA7	WPUA6	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0	267
WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	267
WPUC	WPUC7	WPUC6	WPUC5	WPUC4	WPUC3	WPUC2	WPUC1	WPUC0	267
WPUD(6)	WPUD7	WPUD6	WPUD5	WPUD4	WPUD3	WPUD2	WPUD1	WPUD0	267
WPUE	—	—	—	—	WPUE3(4)	WPUE2(6)	WPUE1(6)	WPUE0(6)	267
WPUF(6)	WPUF7	WPUF6	WPUF5	WPUF4	WPUF3	WPUF2	WPUF1	WPUF0	267
ODCONA	ODCA7	ODCA6	ODCA5	ODCA4	ODCA3	ODCA2	ODCA1	ODCA0	268
ODCONB	ODCB7	ODCB6	ODCB5	ODCB4	ODCB3	ODCB2	ODCB1	ODCB0	268
ODCONC	ODCC7	ODCC6	ODCC5	ODCC4	ODCC3	ODCC2	ODCC1	ODCC0	268
ODCOND(6)	ODCD7	ODCD6	ODCD5	ODCD4	ODCD3	ODCD2	ODCD1	ODCD0	268
ODCONE(6)	—	—	—	—	—	ODCE2	ODCE1	ODCE0	268
ODCONF(7)	ODCF7	ODCF6	ODCF5	ODCF4	ODCF3	ODCF2	ODCF1	ODCF0	268
SLRCONA	SLRA7	SLRA6	SLRA5	SLRA4	SLRA3	SLRA2	SLRA1	SLRA0	269
SLRCONB	SLRB7	SLRB6	SLRB5	SLRB4	SLRB3	SLRB2	SLRB1	SLRB0	269
SLRCONC	SLRC7	SLRC6	SLRC5	SLRC4	SLRC3	SLRC2	SLRC1	SLRC0	269
SLRCOND(6)	SLRD7	SLRD6	SLRD5	SLRD4	SLRD3	SLRD2	SLRD1	SLRD0	269
SLRCONE(6)	—	—	—	—	—	SLRE2	SLRE1	SLRE0	269
SLRCONF(7)	SLRF7	SLRF6	SLRF5	SLRF4	SLRF3	SLRF2	SLRF1	SLRF0	269
INLVLA	INLVLA7	INLVLA6	INLVLA5	INLVLA4	INLVLA3	INLVLA2	INLVLA1	INLVLA0	270
INLVLB	INLVLB7	INLVLB6	INLVLB5	INLVLB4	INLVLB3	INLVLB2(5)	INLVLB1(5)	INLVLB0	270

Legend: — = unimplemented location, read as ‘0’. Shaded cells are not used by I/O Ports.

Note 1: Bits RB6 and RB7 read ‘1’ while in Debug mode.
2: Bit PORTE3 is read-only, and will read ‘1’ when MCLRE = 1 (Master Clear enabled).
3: Bits RB6 and RB7 read ‘1’ while in Debug mode.
4: If MCLRE = 1, the weak pull-up in RE3 is always enabled; bit WPUE3 is not affected.
5: Any peripheral using the I^2C pins read the I^2C ST inputs when enabled via RxyI2C.
6: Unimplemented in PIC18(L)F26/27K42.
7: Unimplemented in PIC18(L)F26/27/45/46/47K42 parts.

17.8 Register Definitions: PPS Input Selection

REGISTER 17-1: xxxPPS: PERIPHERAL xxx INPUT SELECTION

U-0	U-0	R/W-m/u(1,3)	R/W-m/u(1)	R/W-m/u(1)	R/W-m/u(1)	R/W-m/u(1)	R/W-m/u(1)
—	—	xxxPPS<5:0>					
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	-n/n = Value at POR and BOR/Value at all other Resets
u = Bit is unchanged	x = Bit is unknown	q = value depends on peripheral
‘1’ = Bit is set	U = Unimplemented bit, read as ‘0’	m = value depends on default location for that input
‘0’ = Bit is cleared		

bit 7-6	**Unimplemented:** Read as ‘0’
bit 5-3	**xxxPPS<5:3>:** Peripheral xxx Input PORTx Pin Selection bits See Table 17-1 for the list of available ports and default pin locations. 101 = PORTF(2) 100 = PORTE(3) 011 = PORTD(3) 010 = PORTC 001 = PORTB 000 = PORTA
bit 2-0	**xxxPPS<2:0>:** Peripheral xxx Input PORTx Pin Selection bits 111 = Peripheral input is from PORTx Pin 7 (Rx7) 110 = Peripheral input is from PORTx Pin 6 (Rx6) 101 = Peripheral input is from PORTx Pin 5 (Rx5) 100 = Peripheral input is from PORTx Pin 4 (Rx4) 011 = Peripheral input is from PORTx Pin 3 (Rx3) 010 = Peripheral input is from PORTx Pin 2 (Rx2) 001 = Peripheral input is from PORTx Pin 1 (Rx1) 000 = Peripheral input is from PORTx Pin 0 (Rx0)

Note 1: The Reset value ‘m’ of this register is determined by device default locations for that input.
2: Reserved on PIC18LF26/27/45/46/57K42 parts.
3: Reserved on PIC18LF26/27K42 parts.

TABLE 22-3: SUMMARY OF REGISTERS ASSOCIATED WITH TIMER2

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
TxPR	Timer2 Module Period Register								320*
TxTMR	Holding Register for the 8-bit T2TMR Register								320*
TxCON	ON	CKPS<2:0>			OUTPS<3:0>				338
TxCLK	—	—	—	—	—	CS<2:0>			335
TxRST	—	—	—	—	RSEL<3:0>				336
TxHLT	$\overline{\text{PSYNC}}$	CPOL	$\overline{\text{CSYNC}}$	MODE<4:0>					339

Legend: — = unimplemented location, read as ‘0’. Shaded cells are not used for Timer2 module.
* Page provides register information.

25.6.6 GATED WINDOWED MEASURE MODE

This mode measures the duty cycle of the SMT1_signal input over a known input window. It does so by incrementing the timer on each pulse of the clock signal while the SMT1_signal input is high, updating the SMT1CPR 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 26-11: SIMPLIFIED CWG BLOCK DIAGRAM (OUTPUT STEERING MODES)

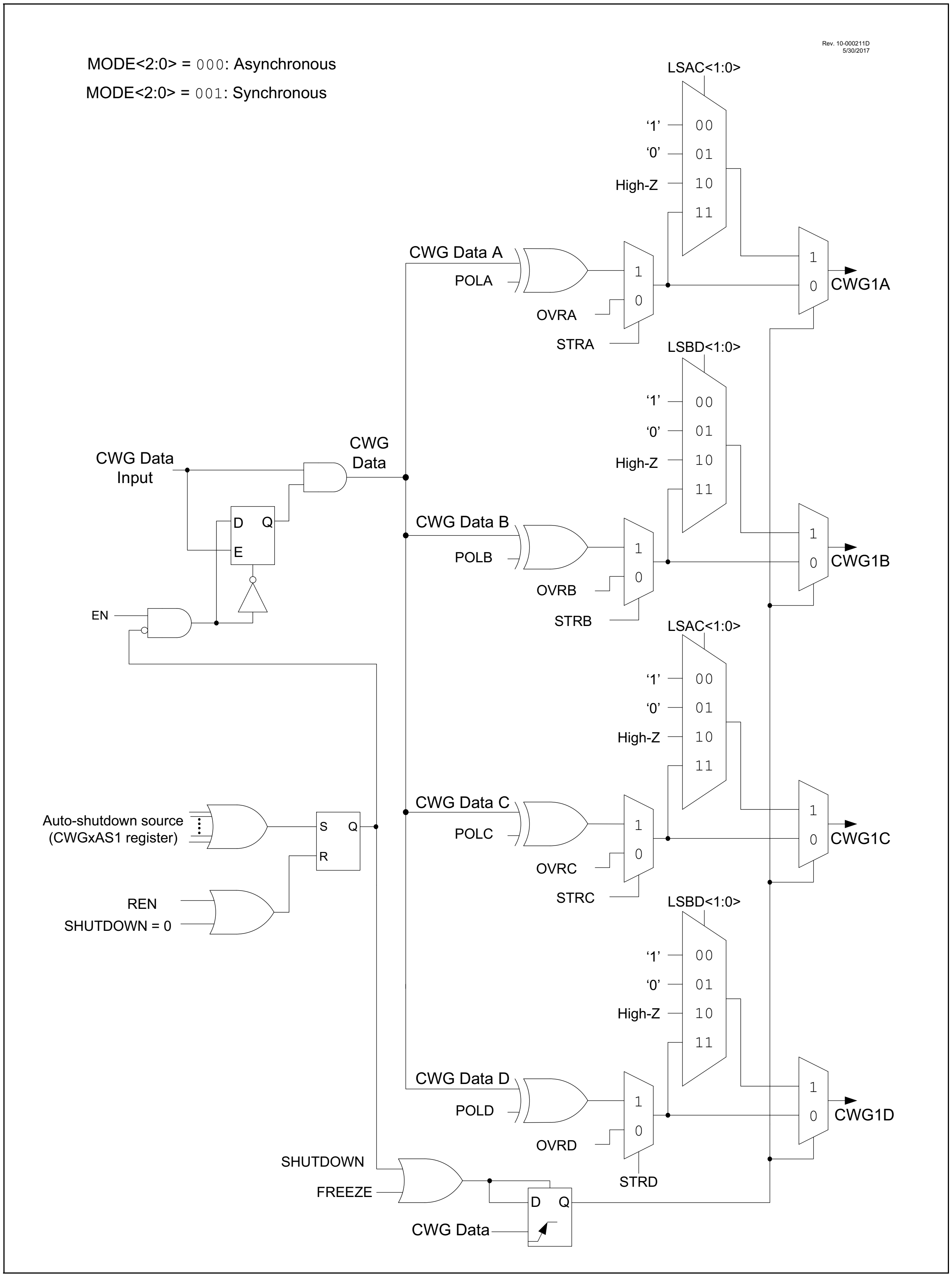


FIGURE 26-12: DEAD-BAND OPERATION, CWGxDBR = 0x01, CWGxDBF = 0x02

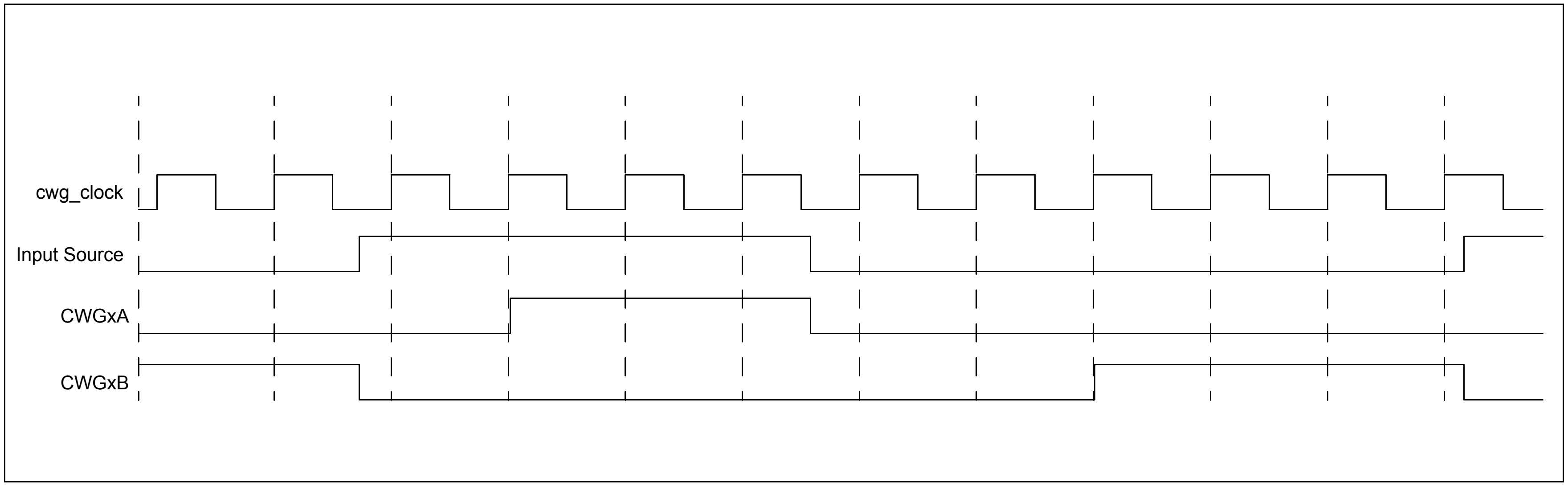


FIGURE 26-13: DEAD-BAND OPERATION, CWGxDBR = 0x03, CWGxDBF = 0x06, SOURCE SHORTER THAN DEAD BAND

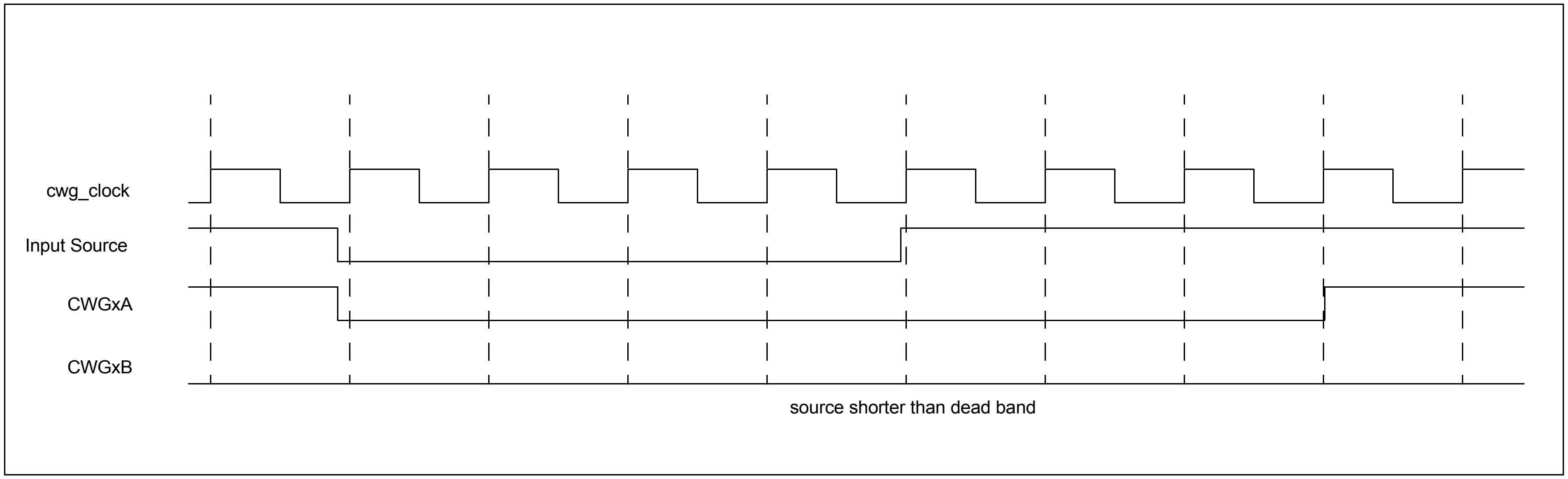
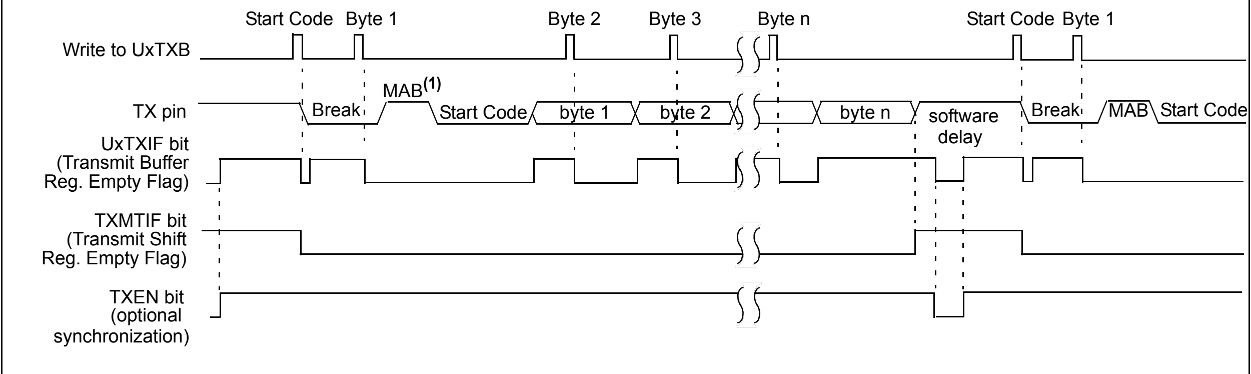


FIGURE 31-6: DMX TRANSMIT SEQUENCE



Note 1: The MAB period is fixed at 3-bits period.

31.5 LIN Modes (UART1 only)

LIN is a protocol used primarily in automotive applications. The LIN network consists of two kinds of software processes: a Master process and a Slave process. Each network has only one Master process and one or more Slave processes.

From a physical layer point of view, the UART on one processor may be driven by both a Master and a Slave process, as long as only one Master process exists on the network.

A LIN transaction consists of a Master process followed by a Slave process. The Slave process may involve more than one Slave where one is transmitting and the other(s) are receiving. The transaction begins by the following Master process transmission sequence:

1. Break
2. Delimiter bit
3. Sync Field
4. PID byte

The PID determines which Slave processes are expected to respond to the Master. When the PID byte is complete, the TX output remains in the Idle state. One or more of the Slave processes may respond to the Master process. If no one responds within the inter-byte period, the Master is free to start another transmission. The inter-byte period is timed by software using a means other than the UART.

The Slave process follows the Master process. When the Slave software recognizes the PID then that Slave process responds by either transmitting the required response or by receiving the transmitted data. Only Slave processes send data. Therefore, Slave processes receiving data are receiving that of another Slave process.

When a Slave sends data, the Slave UART automatically calculates the checksum for the transmitted bytes as they are sent and appends the inverted checksum byte to the slave response.

When a Slave receives data, the checksum is accumulated on each byte as it is received using the same algorithm as the sending process. The last byte, which is the inverted checksum value calculated by the sending process, is added to the locally calculated checksum by the UART. The check passes when the result is all '1's, otherwise the check fails and the CERIF bit is set.

Two methods for computing the checksum are available: legacy and enhanced. The legacy checksum includes only the data bytes. The enhanced checksum includes the PID and the data. The C0EN control bit in the UxCON2 register determines the checksum method. Setting C0EN to '1' selects the enhanced method. Software must select the appropriate method before the Start bit of the checksum byte is received.

31.5.1 LIN MASTER/SLAVE MODE

The LIN Master mode includes capabilities to generate Slave processes. The Master process stops at the PID transmission. Any data that is transmitted in Master/Slave mode is done as a Slave process. LIN Master/Slave mode is configured by the following settings:

- MODE<3:0> = 1100
- TXEN = 1
- RXEN = 1
- UxBRGH:L = Value to achieve desired baud rate
- TXPOL = 0 (for high Idle state)
- STP = desired Stop bits selection
- C0EN = desired checksum mode
- RxyPPS = TX pin selection code
- TX pin TRIS control = 0
- ON = 1

Note:	The TXEN bit must be set before the Master process is received and remain set while in LIN mode whether or not the slave process is a transmitter.

31.6 DALI Mode (UART1 only)

DALI is a protocol used for intelligent lighting control for building automation. The protocol consists of Control Devices and Control Gear. A Control Device is an application controller that sends out commands to the light fixtures. The light fixture itself is termed as a Control Gear. The communication is done using Manchester encoding, which is performed by the UART hardware.

Manchester encoding consists of the clock and data in a single bit stream. A high-to-low or a low-to-high transition always occurs in the middle of the bit period and is not guaranteed to occur at the bit period boundaries. When the consecutive bits in the bit stream are of the same value i.e. consecutive '1's or consecutive '0's, a transition occurs at the bit boundary. However, when the bit value changes, there is no transition at the bit boundary. According to the standard, a half-bit time is typically 416.7 μs long. A double half-bit time or a single bit is typically 833.3 μs.

The protocol is inherently half-duplex. Communication over the bus occurs in the form of forward and backward frames. Wait times between the frames are defined in the standard to prevent collision between the frames.

A Control Device transmission is termed as the forward frame. In the DALI 2.0 standard, a forward frame can be two or three bytes in length. The two-byte forward frame is used for communication between Control Device and Control Gear whereas the three-byte forward frame is used for communication between Control Devices on the bus. The first byte in the forward frame is the control byte and is followed by either one or two data bytes. The transaction begins when the Control Device starts a transmission. Unlike other protocols, each byte in the frame is transmitted MSB first. Typical frame timing is as shown in Figure 31-8.

During communication between two Control Devices, three bytes are required to be transmitted. In this case, the software must write the third byte to UxTXB as soon as UxTXIF goes True and before the output shifter becomes empty. This ensures that the three bytes of the forward frame are transmitted back-to-back without any interruption.

All Control Gear on the bus receive the forward frame. If the forward frame requires a reply to be sent, one of the Control Gear may respond with a single byte, called the backward frame. The 2.0 standard requires the Control Gear to begin transmission of the backward frame between 5.5 ms to 10.5 ms (~14 to 22 half-bit times) after reception of the forward frame. Once the backward frame is received by the Control Device, it is required to wait a minimum of 2.4 ms (~6 half-bit times). After this wait time, the Control Device is free to transmit another forward frame (see Figure 31-9).

A start bit is used to indicate the start of the forward and backward frames. The receiver bit rate is determined by the BRG register. The low period of the start bit is measured and is used as the timing reference for all data bits in the forward and backward frames. The ABDOVF bit is set if the start bit low period causes the measurement counter to overflow. All the bits following the start bit are data bits. The bit stream terminates when no transition is detected in the middle of a bit period (see Figure 31-7).

Forward and backward frames are terminated by two Idle bit periods or Stop bits. Normally, these start in the first bit period of a byte. If both Stop bits are valid, the byte reception is terminated.

If either of the Stop bits is invalid, the frame is tagged as invalid by saving it as a null byte and setting the framing error in the receive FIFO.

A framing error also occurs when no transition is detected on the bus in the middle of a bit period when the byte reception is not complete. In such a scenario, the byte will be saved with the FERIF bit.

31.6.1 CONTROL DEVICE

Control Device mode is configured with the following settings:

- MODE = `0b1000`
- TXEN = `1`
- RXEN = `1`
- UxP1 = Forward frames are held for transmission with this number of half-bit periods after the completion of a forward or backward frame.
- UxP2 = Forward/backward frame threshold delimiter. Any reception that starts this number of half bit periods after the completion of a forward or backward frame is detected as forward frame and sets the PERIF flag of the corresponding received byte.
- UxBRGH:L = Value to achieve 1200 baud rate
- TXPOL = appropriate polarity for interface circuit
- STP = `0b10` for two Stop bits
- RxyPPS = TX pin selection code
- TX pin TRIS control = `0`
- ON = `1`.

A forward frame is initiated by writing the control byte to the UxTXB register. After sending the control byte, each data byte must be written to the UxTXB register as soon as UxTXIF goes true. It is necessary to perform every write after UxTXIF goes true, to ensure that the transmit buffer is ready to accept the byte. Each write must also occur before the TXMTIF bit goes true, to ensure that the bit stream of forward frame is generated without an interruption.

31.9 Stop Bits

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

Stop bits selections include:

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

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

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

31.9.1 DELAYED UXRXIF

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

31.10 Operation after FIFO overflow

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

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

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

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

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.

36.6.5 BURST AVERAGE MODE

The Burst Average mode (ADMD = `011`) acts the same as the Average mode in most respects. The one way it differs is that it continuously retriggers ADC sampling until the CNT value is greater than or equal to RPT, even if Continuous Sampling mode (see **Section 36.6.8 "Continuous Sampling mode"**) is not enabled. This allows for a threshold comparison on the average of a short burst of ADC samples.

36.6.6 LOW-PASS FILTER MODE

The Low-pass Filter mode (ADMD = `100`) acts similarly to the Average mode in how it handles samples (accumulates samples until CNT value greater than or equal to RPT, then triggers threshold comparison), but instead of a simple average, it performs a low-pass filter operation on all of the samples, reducing the effect of high-frequency noise on the average, then performs a threshold comparison on the results. (see Table 36-2 for a more detailed description of the mathematical operation). In this mode, the ADCRS bits determine the cut-off frequency of the low-pass filter (as demonstrated by Table 36-3).

36.6.7 THRESHOLD COMPARISON

At the end of each computation:

- The conversion results are latched and held stable at the end-of-conversion.
- The error is calculated based on a difference calculation which is selected by the ADCALC<2:0> bits in the ADCON3 register. The value can be one of the following calculations (see Register 36-4 for more details):
  - The first derivative of single measurements
  - The CVD result in CVD mode
  - The current result vs. a setpoint
  - The current result vs. the filtered/average result
  - The first derivative of the filtered/average value
  - Filtered/average value vs. a setpoint
- The result of the calculation (ERR) is compared to the upper and lower thresholds, UTH<ADUTHH:ADUTHL> and LTH<ADLTHH:ADLTHL> registers, to set the ADUTHR and ADLTHR flag bits. The threshold logic is selected by ADTMD<2:0> bits in the ADCON3 register. The threshold trigger option can be one of the following:
  - Never interrupt
  - Error is less than lower threshold
  - Error is greater than or equal to lower threshold
  - Error is between thresholds (inclusive)
  - Error is outside of thresholds
  - Error is less than or equal to upper threshold
  - Error is greater than upper threshold
  - Always interrupt regardless of threshold test results
  - If the threshold condition is met, the threshold interrupt flag ADTIF is set.

Note 1:	The threshold tests are signed operations.
2:	If ADAOV is set, a threshold interrupt is signaled.

36.6.8 CONTINUOUS SAMPLING MODE

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

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

36.6.9 DOUBLE SAMPLE CONVERSION

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

36.7 Register Definitions: ADC Control

REGISTER 36-1: ADCON0: ADC CONTROL REGISTER 0

R/W-0/0	R/W-0/0	U-0	R/W-0/0	U-0	R/W-0/0	U-0	R/W/HC-0
ON	CONT	—	CS	—	FM	—	GO
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	HC = Bit is cleared by hardware

bit 7 **ON:** ADC Enable bit
`1` = ADC is enabled
`0` = ADC is disabled

bit 6 **CONT:** ADC Continuous Operation Enable bit
`1` = GO is retriggered upon completion of each conversion trigger until ADTIF is set (if ADSOI is set) or until GO is cleared (regardless of the value of ADSOI)
`0` = ADC is cleared upon completion of each conversion trigger

bit 5 **Unimplemented:** Read as ‘`0`’

bit 4 **CS:** ADC Clock Selection bit
`1` = Clock supplied from FRC dedicated oscillator
`0` = Clock supplied by FOSC, divided according to ADCLK register

bit 3 **Unimplemented:** Read as ‘`0`’

bit 2 **FM:** ADC results Format/alignment Selection
`1` = ADRES and PREV data are right-justified
`0` = ADRES and PREV data are left-justified, zero-filled

bit 1 **Unimplemented:** Read as ‘`0`’

bit 0 **GO:** ADC Conversion Status bit(1)
`1` = ADC conversion cycle in progress. Setting this bit starts an ADC conversion cycle. The bit is cleared by hardware as determined by the CONT bit
`0` = ADC conversion completed/not in progress

Note 1: This bit requires ON bit to be set.

2: If cleared by software while a conversion is in progress, the results of the conversion up to this point will be transfered to ADRES and the state machine will be reset, but the ADIF interrupt flag bit will not be set; filter and threshold operations will not be performed.

43.2 MPLAB XC Compilers

The MPLAB XC Compilers are complete ANSI C compilers for all of Microchip's 8, 16, and 32-bit MCU and DSC devices. These compilers provide powerful integration capabilities, superior code optimization and ease of use. MPLAB XC Compilers run on Windows, Linux or MAC OS X.

For easy source level debugging, the compilers provide debug information that is optimized to the MPLAB X IDE.

The free MPLAB XC Compiler editions support all devices and commands, with no time or memory restrictions, and offer sufficient code optimization for most applications.

MPLAB XC Compilers include an assembler, linker and utilities. The assembler generates relocatable object files that can then be archived or linked with other relocatable object files and archives to create an executable file. MPLAB XC Compiler uses the assembler to produce its object file. Notable features of the assembler include:

- Support for the entire device instruction set
- Support for fixed-point and floating-point data
- Command-line interface
- Rich directive set
- Flexible macro language
- MPLAB X IDE compatibility

43.3 MPASM Assembler

The MPASM Assembler is a full-featured, universal macro assembler for PIC10/12/16/18 MCUs.

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

The MPASM Assembler features include:

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

43.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK Object Linker combines relocatable objects created by the MPASM Assembler. It can link relocatable objects from precompiled libraries, using directives from a linker script.

The MPLIB Object Librarian manages the creation and modification of library files of precompiled code. When a routine from a library is called from a source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications.

The object linker/library features include:

- Efficient linking of single libraries instead of many smaller files
- Enhanced code maintainability by grouping related modules together
- Flexible creation of libraries with easy module listing, replacement, deletion and extraction

43.5 MPLAB Assembler, Linker and Librarian for Various Device Families

MPLAB Assembler produces relocatable machine code from symbolic assembly language for PIC24, PIC32 and dsPIC DSC devices. MPLAB XC Compiler uses the assembler to produce its object file. The assembler generates relocatable object files that can then be archived or linked with other relocatable object files and archives to create an executable file. Notable features of the assembler include:

- Support for the entire device instruction set
- Support for fixed-point and floating-point data
- Command-line interface
- Rich directive set
- Flexible macro language
- MPLAB X IDE compatibility