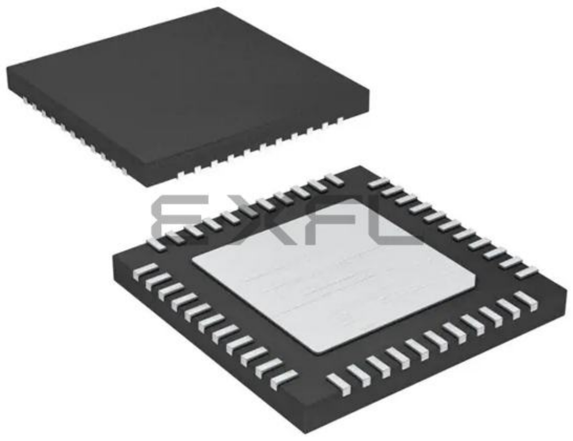


What is "Embedded - Microcontrollers"?

"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	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	40MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	32
Program Memory Size	12KB (6K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	640 x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	44-VQFN Exposed Pad
Supplier Device Package	44-QFN (8x8)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lf4439t-i-ml

NOTES:

5.5 Writing to FLASH Program Memory

The minimum programming block is 4 words or 8 bytes. Word or byte programming is not supported.

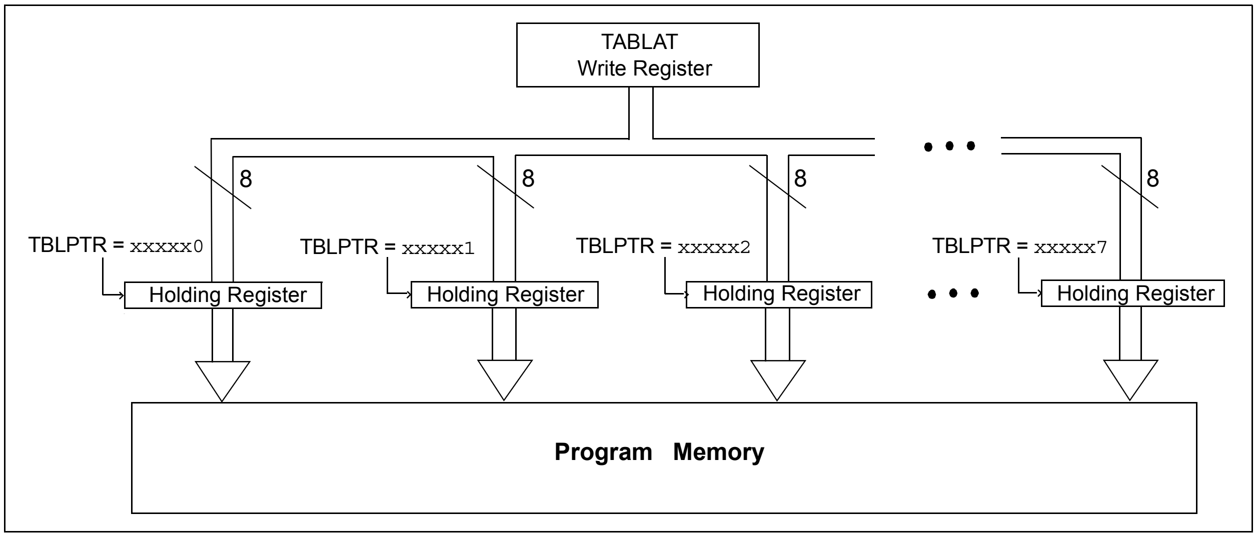
Table Writes are used internally to load the holding registers needed to program the FLASH memory. There are 8 holding registers used by the Table Writes for programming.

Since the Table Latch (TABLAT) is only a single byte, the `TBLWT` instruction has to be executed 8 times for each programming operation. All of the Table Write operations will essentially be short writes, because only the holding registers are written. At the end of updating 8 registers, the EECON1 register must be written to, to start the programming operation with a long write.

The long write is necessary for programming the internal FLASH. Instruction execution is halted while in a long write cycle. The long write will be terminated by the internal programming timer.

The EEPROM on-chip 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 for byte or word operations.

FIGURE 5-5: TABLE WRITES TO FLASH PROGRAM MEMORY



5.5.1 FLASH PROGRAM MEMORY WRITE SEQUENCE

The sequence of events for programming an internal program memory location should be:

1. Read 64 bytes into RAM.
2. Update data values in RAM as necessary.
3. Load Table Pointer with address being erased.
4. Do the row erase procedure.
5. Load Table Pointer with address of first byte being written.
6. Write the first 8 bytes into the holding registers with auto-increment (`TBLWT*+` or `TBLWT+*`).
7. Set EEPGD bit to point to program memory, clear the CFGS bit to access program memory, and set WREN to enable byte writes.
8. Disable interrupts.
9. Write 55h to EECON2.
10. Write AAh to EECON2.
11. Set the WR bit. This will begin the write cycle.
12. The CPU will stall for duration of the write (about 2 ms using internal timer).
13. Re-enable interrupts.
14. Repeat steps 6-14 seven times, to write 64 bytes.
15. Verify the memory (Table Read).

This procedure will require about 18 ms to update one row of 64 bytes of memory. An example of the required code is given in Example 5-3.

Note:	Before setting the WR bit, the table pointer address needs to be within the intended address range of the 8 bytes in the holding registers.

8.2 PIR Registers

The PIR registers contain the individual flag bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Flag registers (PIR1, PIR2).

Note 1: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

2: User software should ensure the appropriate interrupt flag bits are cleared prior to enabling an interrupt, and after servicing that interrupt.

REGISTER 8-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0
PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF[(2)]	TMR1IF
bit 7							bit 0

bit 7 **PSPIF[(1)]:** Parallel Slave Port Read/Write Interrupt Flag bit
`1` = A read or a write operation has taken place (must be cleared in software)
`0` = No read or write has occurred

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
`1` = An A/D conversion completed (must be cleared in software)
`0` = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
`1` = The USART receive buffer, RCREG, is full (cleared when RCREG is read)
`0` = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit (see Section 17.0 for details on TXIF functionality)
`1` = The USART transmit buffer, TXREG, is empty (cleared when TXREG is written)
`0` = The USART transmit buffer is full

bit 3 **SSPIF**: Master Synchronous Serial Port Interrupt Flag bit
`1` = The transmission/reception is complete (must be cleared in software)
`0` = Waiting to transmit/receive

bit 2 **Unimplemented:** Read as ‘0’

bit 1 **TMR2IF[(2)]:** TMR2 to PR2 Match Interrupt Flag bit
`1` = TMR2 to PR2 match occurred (must be cleared in software)
`0` = No TMR2 to PR2 match occurred

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
`1` = TMR1 register overflowed (must be cleared in software)
`0` = MR1 register did not overflow

Note 1: This bit is reserved on PIC18F2X39 devices; always maintain this bit clear.
2: This bit is reserved for use by the ProMPT kernel; do not alter its value.

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

REGISTER 8-5: PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—
bit 7							bit 0

bit 7-5	**Unimplemented:** Read as '0'
bit 4	**EEIF**: Data EEPROM/FLASH Write Operation Interrupt Flag bit 1 = The write operation is complete (must be cleared in software) 0 = The write operation is not complete, or has not been started
bit 3	**BCLIF**: Bus Collision Interrupt Flag bit 1 = A bus collision occurred (must be cleared in software) 0 = No bus collision occurred
bit 2	**LVDIF**: Low Voltage Detect Interrupt Flag bit 1 = A low voltage condition occurred (must be cleared in software) 0 = The device voltage is above the Low Voltage Detect trip point
bit 1	**TMR3IF**: TMR3 Overflow Interrupt Flag bit 1 = TMR3 register overflowed (must be cleared in software) 0 = TMR3 register did not overflow
bit 0	**Unimplemented:** Read as ‘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

8.6 INT0 Interrupt

External interrupts on the RB0/INT0, RB1/INT1 and RB2/INT2 pins are edge triggered: either rising, if the corresponding INTEDGx bit is set in the INTCON2 register, or falling, if the INTEDGx bit is clear. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit INTxF is set. This interrupt can be disabled by clearing the corresponding enable bit INTxE. Flag bit INTxF must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. All external interrupts (INT0, INT1 and INT2) can wake-up the processor from SLEEP, if bit INTxE was set prior to going into SLEEP. If the global interrupt enable bit GIE is set, the processor will branch to the interrupt vector following wake-up.

The INT0 interrupt is always configured as a high priority interrupt, and cannot be reconfigured. Interrupt priority for INT1 and INT2 is determined by the value contained in the interrupt priority bits, INT1IP (INTCON3<6>) and INT2IP (INTCON3<7>).

Because it is always configured as a high priority interrupt, INT0 cannot be used in conjunction with the ProMPT kernel; it must always be disabled (INTCON<4> = 0). Failure to do this may result in erratic operation of the motor control.

8.7 TMR0 Interrupt

In 8-bit mode (which is the default), an overflow in the TMR0 register (FFh $\rightarrow$ 00h) will set flag bit TMR0IF. In 16-bit mode, an overflow in the TMR0H:TMR0L register pair (FFFFh $\rightarrow$ 0000h) will set flag bit TMR0IF. The interrupt can be enabled or disabled by setting or clearing enable bit TMR0IE (INTCON<5>). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit TMR0IP (INTCON2<2>). See Section 10.0 for further details on the Timer0 module.

8.8 PORTB Interrupt-on-Change

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled or disabled by setting or clearing the enable bit RBIE (INTCON<3>). Interrupt priority for PORTB interrupt-on-change is determined by the value contained in the interrupt priority bit RBIP (INTCON2<0>).

8.9 Context Saving During Interrupts

During an interrupt, the return PC value is saved on the stack. Additionally, the WREG, STATUS and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (see Section 4.3), the user may need to save the WREG, STATUS and BSR registers in software. Depending on the user's application, other registers may also need to be saved. Example 8-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 8-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF   W_TEMP                         ; W_TEMP is in virtual bank
MOVFF   STATUS, STATUS_TEMP            ; STATUS_TEMP located anywhere
MOVFF   BSR,    BSR_TEMP               ; BSR located anywhere
;
; USER ISR CODE
;
MOVFF   BSR_TEMP,  BSR                 ; Restore BSR
MOVF    W_TEMP,    W                   ; Restore WREG
MOVFF   STATUS_TEMP,STATUS             ; Restore STATUS
```

REGISTER 9-1: TRISE REGISTER

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0
bit 7							bit 0

bit 7	**IBF:** Input Buffer Full Status bit 1 = A word has been received and waiting to be read by the CPU 0 = No word has been received
bit 6	**OBF**: Output Buffer Full Status bit 1 = The output buffer still holds a previously written word 0 = The output buffer has been read
bit 5	**IBOV**: Input Buffer Overflow Detect bit (in Microprocessor mode) 1 = A write occurred when a previously input word has not been read (must be cleared in software) 0 = No overflow occurred
bit 4	**PSPMODE**: Parallel Slave Port Mode Select bit 1 = Parallel Slave Port mode 0 = General Purpose I/O mode
bit 3	**Unimplemented:** Read as '0'
bit 2	**TRISE2**: RE2 Direction Control bit 1 = Input 0 = Output
bit 1	**TRISE1**: RE1 Direction Control bit 1 = Input 0 = Output
bit 0	**TRISE0**: RE0 Direction Control bit 1 = Input 0 = Output

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

NOTES:

14.3 Software Interface

A sine table, stored in the ProMPT kernel, is used as the basis for synthesizing the DC bus using the PWM modules. The table values are accessed in sequence and scaled based on the frequency or the speed at which the motor is intended to run. The intended frequency input can be from an A/D channel or a digital value.

Parameters in the ProMPT modules can be accessed using the pre-defined Application Program Interface (API) methods. A list of the APIs is given in Section 14.3.3.

For example, to run the motor at 40 Hz, the user would invoke the `PromMPT_SetFrequency` API:

```
i = ProMPT_SetFrequency(40);
```

where `i` is an unsigned character variable. In this case, if `i` = 0 on return, the command has been successfully executed. If the frequency input is out of range, or if there is an error in setting the frequency, `i` is returned with a value of FFh.

Similarly, to check the frequency set by the ProMPT kernel, use the `ProMPT_GetFrequency` API:

```
i = ProMPT_GetFrequency(void);
```

where `i` is an unsigned character variable. Upon return from the ProMPT kernel, `i` will contain the frequency value in the ProMPT kernel.

14.3.1 THE V/F CURVE

The ProMPT kernel contains a default V/F curve stored in memory. The default curve is linear, as shown in Figure 14-2. Table 14-1 shows the data points used to construct the curve.

Users may require a different V/F curve for their application, based on the load on the motor, or based on the characteristics of the motor used. The curve can be changed in the application program using the API method `SetVFCurve(X,Y)`, where `X` is the frequency and `Y` is the level of modulation of the DC bus voltage. As a rule, in customizing the curve, the input frequency corresponding to the point on the V/F curve that gives 100% modulation should match the motor's rated frequency. Similarly, full modulation should occur at the motor's rated input voltage. (See Figure 14-2 for details.)

Examples of the characteristics for V/F curves for typical motor applications are shown in Section 14-2 (page 115).

14.3.2 PARAMETERS DEFINED BY THE ProMPT API METHODS

Frequency: The frequency (in Hz) of the supply current for steady state motor operation.

Modulation: The level of modulation (in percentage) applied to the DC supply voltage by the PWM through the H-bridge to produce AC drive current.

Acceleration rate: The rate of increase of motor speed, achieved by ramping up the supply frequency. Expressed in Hz/s.

Deceleration rate: The rate of decrease of motor speed, achieved by ramping down the supply frequency. Expressed in Hz/s.

Boost: The mode for starting a stopped motor by varying the supply current frequency and modulation until steady state speed is reached. Boost is defined in terms of a frequency, a starting and ending modulation, and a time interval for the transition between the two.

PWM Frequency: The sampling rate (in kHz) at which the PWM module operates.

FIGURE 14-2: DEFAULT V/F CURVE FOR THE ProMPT KERNEL

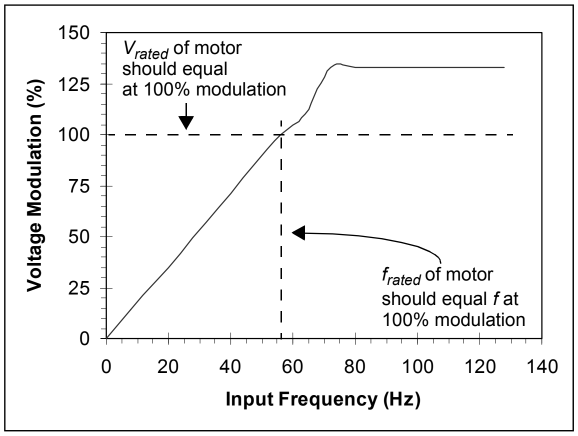


TABLE 14-1: DATA POINTS FOR THE DEFAULT V/F CURVE

Frequency (Hz)	% Modulation
0	0
8	14
16	28
24	42
32	57
40	71
48	86
56	100
64	110
72	133
80	133
88	133
96	133
104	133
112	133
120	133
128	133

`unsigned char ProMPT_GetBoostTime()`

Resources used: 1 stack level

Range of values: 0 to 255

Description: Returns the time in seconds for Boost mode.

`unsigned char ProMPT_GetDecelRate()`

Resources used: 1 stack level

Range of values: 0 to 255

Description: Returns the current deceleration rate in Hz/second.

`unsigned char ProMPT_GetFrequency(void)`

Resources used: 1 stack level

Range of values: 0 to 127

Description: Returns the current output frequency in Hz. This may not be the frequency commanded due to Boost or Accel/Decel logic.

`unsigned char ProMPT_GetModulation(void)`

Resources used: Hardware Multiplier; 1 stack level

Range of values: 0 to 200

Description: Returns the current output modulation in %.

`unsigned char ProMPT_GetParameter(unsigned char parameter)`

Resources used: 1 stack level

Description: In addition to its pre-defined API methods, the ProMPT kernel allows the user to custom define up to 16 functions for control or communication purposes not covered by the ProMPT APIs. These parameters are used to communicate with motor control GUI evaluation tools, such as Microchip's DashDriveMP™. This method returns the current value of any one of the parameters.

`unsigned char ProMPT_GetVFCurve(unsigned char point)`

Resources used: Hardware Multiplier; 1 stack level

Description: This function returns one of the 17 modulation values (in %) of the V/F curve. Each point represents a frequency increment of 8 Hz, ranging from point 0 (0 Hz) to point 16 (128 Hz).

`void ProMPT_Init(unsigned char PWMfrequency)`

Resources used: 64 Bytes RAM; Timer2; PWM1 and PWM2; High Priority Interrupt Vector; Hardware Multiplier; fast call/return; FSR 0; TBLPTR; 2 stack levels

`PWMfrequency` values: 0 or 1

Description: This function must be called before all other ProMPT methods, and it must be called only once. This routine configures Timer2 and the PWM outputs.

When `PWMfrequency` is '0', the module's operating frequency is 9.75 kHz. When `PWMfrequency` is '1', the module's operating frequency is 19.53 kHz.

Note: Since the high priority interrupt is used, the fast call/return cannot be used by other routines.

`void ProMPT_SetLineVoltage(unsigned char voltage)`

Resources used: Hardware Multiplier; 0 stack levels

`voltage` range: 0 to 255

Description: Sets the line voltage for Automatic Voltage Compensation. The units for `SetLineVoltage` and `SetMotorVoltage` must be the same for accurate operation. The values passed to `SetMotorVoltage` and `SetLineVoltage` can be the same to disable voltage compensation.

`void ProMPT_SetMotorVoltage(unsigned char voltage)`

Resources used: Hardware Multiplier; 0 stack levels

`voltage` range: 0 to 255

Description: Sets the motor rating for Automatic Voltage Compensation. The units for `SetLineVoltage` and `SetMotorVoltage` must be the same for accurate operation. The values passed to `SetMotorVoltage` and `SetLineVoltage` can be the same to disable voltage compensation.

`void ProMPT_SetParameter(unsigned char parameter, unsigned char value)`

Resources used: 0 stack levels

`parameter` range:

Description: In addition to its pre-defined API methods, the ProMPT kernel allows the user to custom define up to 16 functions for control or communication purposes not covered by the ProMPT APIs. This function sets the value of the specified user defined function.

`void ProMPT_SetPWMfrequency(unsigned char PWMfrequency)`

`PWMfrequency` values: 0 or 1

Resources used: Timer2; 1 stack level

Description: This sets and changes the PWM switching frequency. Typically, this is set with the `Init()` function. When `PWMfrequency` is ‘0’, the module’s operating frequency is 9.75 kHz. When `PWMfrequency` is ‘1’, the module’s operating frequency is 19.53 kHz.

`void ProMPT_SetVFCurve(unsigned char point, unsigned char value)`

Resources used: Hardware Multiplier; 0 stack level

`point` range: 0 to 16 (0 = 0 Hz, 1 = 8 Hz, 2 = 16 Hz……. 17 = 128 Hz)

`value` range: 0 to 200

Description: This sets one of the 17 modulation values (in %) for the V/F curve. Each point represents a frequency increment of 8 Hz, ranging from point 0 (0 Hz) to point 16 (128 Hz).

`unsigned char ProMPT_Tick(void)`

Resources used: 1 stack level

Description: The value of the Tick timer flag becomes ‘1’ every 62.5 ms (1/16 second). This can be used for timing applications. `clearTick` must be called in the timing routine when this is serviced.

16.3.3 ENABLING SPI I/O

To enable the serial port, SSP Enable bit, SSPEN (SSPCON1<5>), must be set. To reset or reconfigure SPI mode, clear the SSPEN bit, re-initialize the SSPCON registers, and then set the SSPEN bit. This configures the SDI, SDO, SCK, and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed. That is:

- SDI is automatically controlled by the SPI module
- SDO must have TRISC<5> bit cleared
- SCK (Master mode) must have TRISC<3> bit cleared
- SCK (Slave mode) must have TRISC<3> bit set
- $\overline{SS}$ must have TRISC<4> bit set

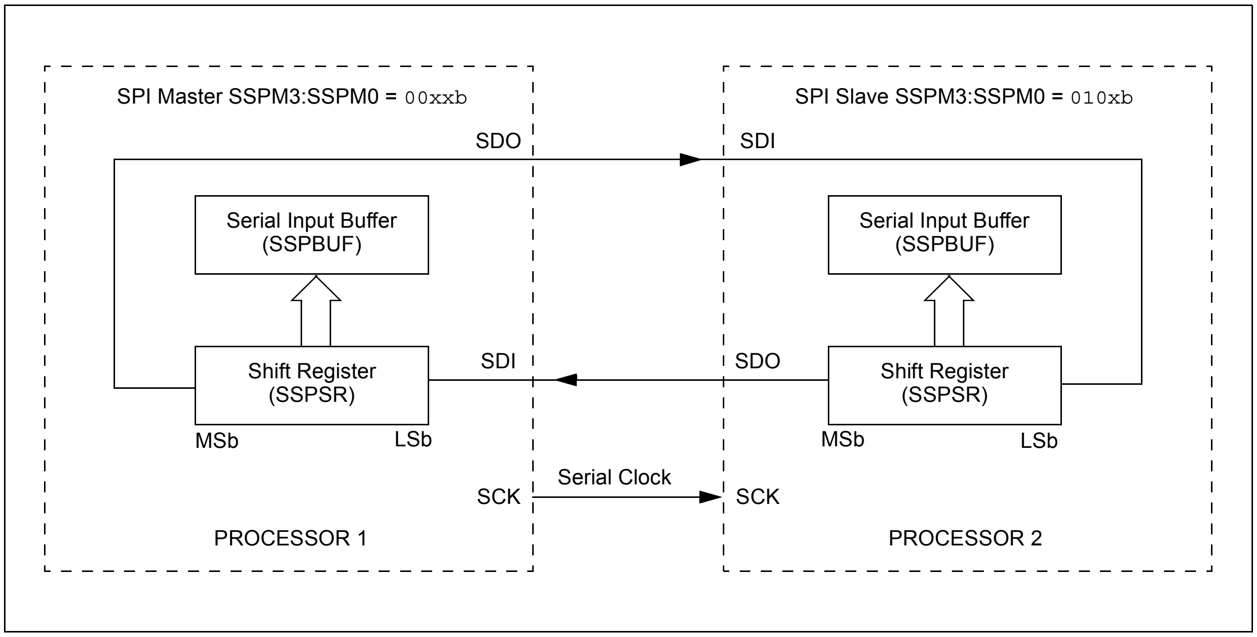
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

16.3.4 TYPICAL CONNECTION

Figure 16-2 shows a typical connection between two microcontrollers. The master controller (Processor 1) initiates the data transfer by sending the SCK signal. Data is shifted out of both shift registers on their programmed clock edge, and latched on the opposite edge of the clock. Both processors should be programmed to the same Clock Polarity (CKP), then both controllers would send and receive data at the same time. Whether the data is meaningful (or dummy data) depends on the application software. This leads to three scenarios for data transmission:

- Master sends data — Slave sends dummy data
- Master sends data — Slave sends data
- Master sends dummy data — Slave sends data

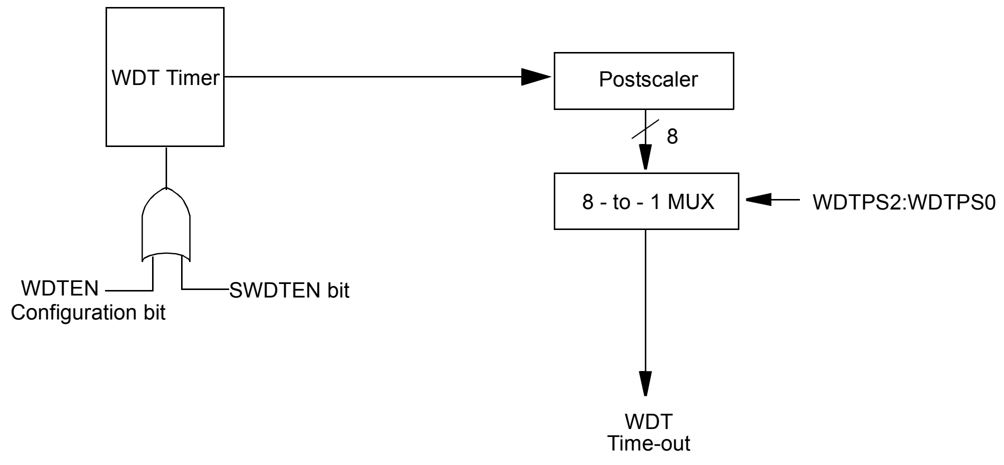
FIGURE 16-2: SPI MASTER/SLAVE CONNECTION


SPI Master SSPM3:SSPM0 = 00xxb
SDO
Serial Input Buffer (SSPBUF)
Shift Register (SSPSR)
SDI
MSb
LSb
SCK
PROCESSOR 1
Serial Clock
SPI Slave SSPM3:SSPM0 = 010xb
SDI
Serial Input Buffer (SSPBUF)
Shift Register (SSPSR)
SDO
MSb
LSb
SCK
PROCESSOR 2

20.2.2 WDT POSTSCALER

The WDT has a postscaler that can extend the WDT Reset period. The postscaler is selected at the time of the device programming, by the value written to the CONFIG2H configuration register.

FIGURE 20-1: WATCHDOG TIMER BLOCK DIAGRAM



Note: WDPS2:WDPS0 are bits in register CONFIG2H.

TABLE 20-2: SUMMARY OF WATCHDOG TIMER REGISTERS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CONFIG2H	—	—	—	—	WDTPS2	WDTPS2	WDTPS0	WDTEN
RCON	IPEN	—	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
WDTCON	—	—	—	—	—	—	—	SWDTEN

Legend: Shaded cells are not used by the Watchdog Timer.

21.0 INSTRUCTION SET SUMMARY

The PIC18FXXX instruction set adds many enhancements to the previous PICmicro instruction sets, while maintaining an easy migration from these PICmicro instruction sets.

Most instructions are a single program memory word (16-bits), but there are three instructions that require two program memory locations.

Each single word instruction is a 16-bit word divided into an OPCODE, which specifies the instruction type and one or more operands, which further specify the operation of the instruction.

The instruction set is highly orthogonal and is grouped into four basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal** operations
- **Control** operations

The PIC18FXXX instruction set summary in Table 21-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 21-1 shows the opcode field descriptions.

Most **byte-oriented** instructions have three operands:

1. The file register (specified by ‘f’)
2. The destination of the result (specified by ‘d’)
3. The accessed memory (specified by ‘a’)

The file register designator 'f' specifies which file register is to be used by the instruction.

The destination designator ‘d’ specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the WREG register. If 'd' is one, the result is placed in the file register specified in the instruction.

All **bit-oriented** instructions have three operands:

1. The file register (specified by ‘f’)
2. The bit in the file register (specified by ‘b’)
3. The accessed memory (specified by ‘a’)

The bit field designator 'b' selects the number of the bit affected by the operation, while the file register designator 'f' represents the number of the file in which the bit is located.

The **literal** instructions may use some of the following operands:

- A literal value to be loaded into a file register (specified by ‘k’)
- The desired FSR register to load the literal value into (specified by ‘f’)
- No operand required (specified by ‘—’)

The **control** instructions may use some of the following operands:

- A program memory address (specified by ‘n’)
- The mode of the Call or Return instructions (specified by ‘s’)
- The mode of the Table Read and Table Write instructions (specified by ‘m’)
- No operand required (specified by ‘—’)

All instructions are a single word, except for three double-word instructions. These three instructions were made double-word instructions, so that all the required information is available in these 32 bits. In the second word, the 4 MSbs are ‘1’s. If this second word is executed as an instruction (by itself), it will execute as a `NOP`.

All single word instructions are executed in a single instruction cycle, unless a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles with the additional instruction cycle(s) executed as a `NOP`.

The double-word instructions execute in two instruction cycles.

One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 μs. If a conditional test is true, or the program counter is changed as a result of an instruction, the instruction execution time is 2 μs. Two-word branch instructions (if true) would take 3 μs.

Figure 21-1 shows the general formats that the instructions can have.

All examples use the format ‘`nnh`’ to represent a hexadecimal number, where ‘`h`’ signifies a hexadecimal digit.

The Instruction Set Summary, shown in Table 21-2, lists the instructions recognized by the Microchip Assembler (MPASM™).

Section 21.1 provides a description of each instruction.

TABLE 21-2: PIC18FXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da0	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	0da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word	2	1100	ffff	ffff	ffff	None	
		f_d (destination) 2nd word		1111	ffff	ffff	ffff		
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	1, 2
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	1, 2
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	1, 2
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

ANDWF **AND W with f**

Syntax:	[*label*] ANDWF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) .AND. (f) $\rightarrow$ dest
Status Affected:	N,Z

Encoding:

`0001`	`01da`	`ffff`	`ffff`

Description: The contents of W are AND'ed with register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR will not be overridden (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ANDWF REG, 0, 0`

Before Instruction

W	=	0x17
REG	=	0xC2

After Instruction

W	=	0x02
REG	=	0xC2

BC **Branch if Carry**

Syntax:	[*label*] BC n
Operands:	$-128 \leq n \leq 127$
Operation:	if carry bit is '1' (PC) + 2 + 2n $\rightarrow$ PC
Status Affected:	None

Encoding:

`1110`	`0010`	`nnnn`	`nnnn`

Description: If the Carry bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BC 5`

Before Instruction

PC	=	address `(HERE)`

After Instruction

If Carry	=	1;
PC	=	address `(HERE+12)`
If Carry	=	0;
PC	=	address `(HERE+2)`

COMF — Complement f

Syntax:	[*label*] COMF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(\overline{f}) \rightarrow$ dest
Status Affected:	N, Z
Encoding:	`0001` `11da` `ffff` `ffff`
Description:	The contents of register 'f' are complemented. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `COMF REG, 0, 0`

Before Instruction

REG	=	0x13

After Instruction

REG	=	0x13
W	=	0xEC

CPFSEQ — Compare f with W, skip if f = W

Syntax:	[*label*] CPFSEQ f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(f) – (W), skip if (f) = (W) (unsigned comparison)
Status Affected:	None
Encoding:	`0110` `001a` `ffff` `ffff`
Description:	Compares the contents of data memory location 'f' to the contents of W by performing an unsigned subtraction. If 'f' = W, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:

```
HERE      CPFSEQ REG, 0
NEQUAL    :
EQUAL     :
```

Before Instruction

PC Address	=	`HERE`
W	=	?
REG	=	?

After Instruction

If REG	=	W;
PC	=	Address (`EQUAL`)
If REG	≠	W;
PC	=	Address (`NEQUAL`)

22.0 DEVELOPMENT SUPPORT

The PICmicro® microcontrollers are supported with a full range of hardware and software development tools:

- Integrated Development Environment
  - MPLAB® IDE Software
- Assemblers/Compilers/Linkers
  - MPASM™ Assembler
  - MPLAB C17 and MPLAB C18 C Compilers
  - MPLINK™ Object Linker/ MPLIB™ Object Librarian
- Simulators
  - MPLAB SIM Software Simulator
- Emulators
  - MPLAB ICE 2000 In-Circuit Emulator
  - ICEPIC™ In-Circuit Emulator
- In-Circuit Debugger
  - MPLAB ICD
- Device Programmers
  - PRO MATE® II Universal Device Programmer
  - PICSTART® Plus Entry-Level Development Programmer
- Low Cost Demonstration Boards
  - PICDEM™ 1 Demonstration Board
  - PICDEM 2 Demonstration Board
  - PICDEM 3 Demonstration Board
  - PICDEM 17 Demonstration Board
  - KEELOQ® Demonstration Board

22.1 MPLAB Integrated Development Environment Software

The MPLAB IDE software brings an ease of software development previously unseen in the 8-bit microcontroller market. The MPLAB IDE is a Windows® based application that contains:

- An interface to debugging tools
  - simulator
  - programmer (sold separately)
  - emulator (sold separately)
  - in-circuit debugger (sold separately)
- A full-featured editor
- A project manager
- Customizable toolbar and key mapping
- A status bar
- On-line help

The MPLAB IDE allows you to:

- Edit your source files (either assembly or ‘C’)
- One touch assemble (or compile) and download to PICmicro emulator and simulator tools (automatically updates all project information)
- Debug using:
  - source files
  - absolute listing file
  - machine code

The ability to use MPLAB IDE with multiple debugging tools allows users to easily switch from the cost-effective simulator to a full-featured emulator with minimal retraining.

22.2 MPASM Assembler

The MPASM assembler is a full-featured universal macro assembler for all PICmicro MCU’s.

The MPASM assembler has a command line interface and a Windows shell. It can be used as a stand-alone application on a Windows 3.x or greater system, or it can be used through MPLAB IDE. 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, an absolute LST file that contains source lines and generated machine code, and a COD file for debugging.

The MPASM assembler features include:

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

22.3 MPLAB C17 and MPLAB C18 C Compilers

The MPLAB C17 and MPLAB C18 Code Development Systems are complete ANSI ‘C’ compilers for Microchip’s PIC17CXXX and PIC18CXXX family of microcontrollers, respectively. These compilers provide powerful integration capabilities and ease of use not found with other compilers.

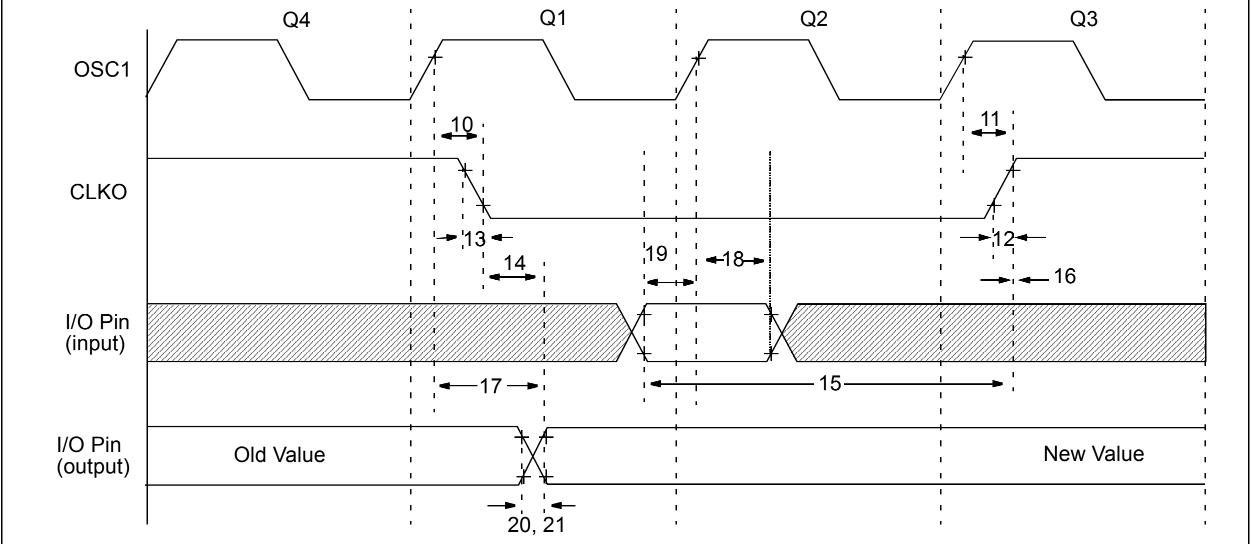
For easier source level debugging, the compilers provide symbol information that is compatible with the MPLAB IDE memory display.

TABLE 22-1: DEVELOPMENT TOOLS FROM MICROCHIP

		PIC12CXXX	PIC14000	PIC16C5X	PIC16C6X	PIC16CXXX	PIC16F62X	PIC16C7X	PIC16C7XX	PIC16C8X/ PIC16F8X	PIC16F8XX	PIC16C9XX	PIC17C4X	PIC17C7XX	PIC18CXX2	PIC18FXXX	24CXX/ 25CXX/ 93CXX	HCSXXX	MCRFXXX	MCP2510
Software Tools	MPLAB® Integrated Development Environment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	MPLAB® C17 C Compiler												✓	✓						
	MPLAB® C18 C Compiler														✓	✓				
	MPASM™ Assembler/ MPLINK™ Object Linker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Emulators	MPLAB® ICE In-Circuit Emulator	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	ICEPIC™ In-Circuit Emulator	✓		✓	✓	✓		✓	✓	✓		✓								
Debugger	MPLAB® ICD In-Circuit Debugger				✓*			✓*			✓					✓				
Programmers	PICSTART® Plus Entry Level Development Programmer	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	PRO MATE® II Universal Device Programmer	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Demo Boards and Eval Kits	PICDEM™ 1 Demonstration Board			✓		✓		✓†		✓			✓							
	PICDEM™ 2 Demonstration Board				✓†			✓†							✓	✓				
	PICDEM™ 3 Demonstration Board											✓								
	PICDEM™ 14A Demonstration Board		✓																	
	PICDEM™ 17 Demonstration Board													✓						
	KEELOQ® Evaluation Kit																	✓		
	KEELOQ® Transponder Kit																	✓		
	microID™ Programmer's Kit																		✓	
	125 kHz microID™ Developer's Kit																		✓	
	125 kHz Anticollision microID™ Developer's Kit																		✓	
	13.56 MHz Anticollision microID™ Developer's Kit																		✓	
	MCP2510 CAN Developer's Kit																			✓

* Contact the Microchip Technology Inc. web site at www.microchip.com for information on how to use the MPLAB® ICD In-Circuit Debugger (DV164001) with PIC16C62, 63, 64, 65, 72, 73, 74, 76, 77.
** Contact Microchip Technology Inc. for availability date.
† Development tool is available on select devices.

FIGURE 23-6: CLKO AND I/O TIMING



Note: Refer to Figure 23-4 for load conditions.

TABLE 23-6: CLKO AND I/O TIMING REQUIREMENTS

Param. No.	Symbol	Characteristic		Min	Typ	Max	Units	Conditions
10	TosH2ckL	OSC1↑ to CLKO↓		—	75	200	ns	**(Note 1)**
11	TosH2ckH	OSC1↑ to CLKO↑		—	75	200	ns	**(Note 1)**
12	TckR	CLKO rise time		—	35	100	ns	**(Note 1)**
13	TckF	CLKO fall time		—	35	100	ns	**(Note 1)**
14	TckL2ioV	CLKO↓ to Port out valid		—	—	0.5 TCY + 20	ns	**(Note 1)**
15	TioV2ckH	Port in valid before CLKO ↑		0.25 TCY + 25	—	—	ns	**(Note 1)**
16	TckH2ioI	Port in hold after CLKO ↑		0	—	—	ns	**(Note 1)**
17	TosH2ioV	OSC1↑ (Q1 cycle) to Port out valid		—	50	150	ns	
18	TosH2ioI	OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time)	PIC18**F**XXXX	100	—	—	ns	
18A			PIC18**LF**XXXX	200	—	—	ns	
19	TioV2osH	Port input valid to OSC1↑ (I/O in setup time)		0	—	—	ns	
20	TioR	Port output rise time	PIC18**F**XXXX	—	10	25	ns	
20A			PIC18**LF**XXXX	—	—	60	ns	VDD = 2V
21	TioF	Port output fall time	PIC18**F**XXXX	—	10	25	ns	
21A			PIC18**LF**XXXX	—	—	60	ns	VDD = 2V
22††	TINP	INT pin high or low time		TCY	—	—	ns	
23††	TRBP	RB7:RB4 change INT high or low time		TCY	—	—	ns	
24††	TRCP	RC7:RC4 change INT high or low time		20			ns	

†† These parameters are asynchronous events not related to any internal clock edges.
Note 1: Measurements are taken in RC mode, where CLKO output is 4 x TOSC.

PIC18FXX39 PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	–	X	/XX	XXX
Device		Temperature Range	Package	Pattern

Device	PIC18FXX39[(1)], PIC18FXX39T[(2)]; VDD range 4.2V to 5.5V PIC18LFXX39[(1)], PIC18LFXX39T[(2)]; VDD range 2.0V to 5.5V

Temperature Range	I = -40°C to +85°C (Industrial)
	E = -40°C to +125°C (Extended)

Package	ML = QFN (Quad Flatpack, No Leads)
	P = PDIP
	PT = TQFP (Plastic Thin Quad Flatpack)
	SO = SOIC
	SP = Skinny Plastic DIP

Pattern	QTP, SQTP, Code or Special Requirements (blank otherwise)

Examples:

a) PIC18LF4539 - I/P 301 = Industrial temp., PDIP package, Extended VDD limits, QTP pattern #301.

b) PIC18LF2439 - I/SO = Industrial temp., SOIC package, Extended VDD limits.

c) PIC18F4439 - E/P = Extended temp., PDIP package, normal VDD limits.

Note 1: F = Standard Voltage range
LF = Wide Voltage Range

2: T = in tape and reel - SOIC, QFN, and TQFP packages only.

Sales and Support

Data Sheets
Products supported by a preliminary Data Sheet may have an errata sheet describing minor operational differences and recommended workarounds. To determine if an errata sheet exists for a particular device, please contact one of the following:

1. Your local Microchip sales office
2. The Microchip Corporate Literature Center U.S. FAX: (480) 792-7277
3. The Microchip Worldwide Site (www.microchip.com)

Please specify which device, revision of silicon and Data Sheet (include Literature #) you are using.

New Customer Notification System
Register on our web site (www.microchip.com/cn) to receive the most current information on our products.