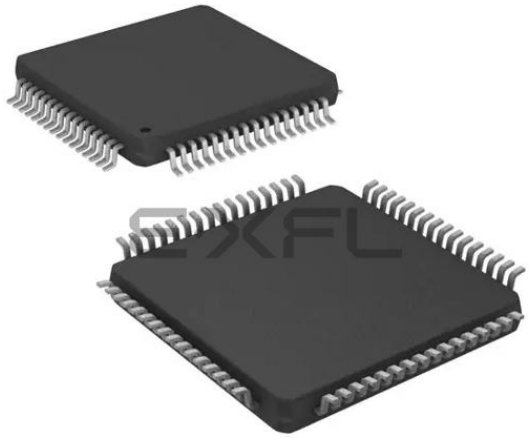


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	40MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	50
Program Memory Size	128KB (64K x 16)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	3.8K x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 3.6V
Data Converters	A/D 11x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	64-TQFP
Supplier Device Package	64-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f67j10t-i-pt

TABLE 1-4: PIC18F8XJ10/8XJ15 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number	Pin Type	Buffer Type	Description
	TQFP			
				PORTE is a bidirectional I/O port.
RE0/AD8/$\overline{RD}$/P2D	4			
RE0		I/O	ST	Digital I/O.
AD8		I/O	TTL	External memory address/data 8.
$\overline{RD}$		I	TTL	Read control for Parallel Slave Port.
P2D		O	—	ECCP2 PWM output D.
RE1/AD9/$\overline{WR}$/P2C	3			
RE1		I/O	ST	Digital I/O.
AD9		I/O	TTL	External memory address/data 9.
$\overline{WR}$		I	TTL	Write control for Parallel Slave Port.
P2C		O	—	ECCP2 PWM output C.
RE2/AD10/$\overline{CS}$/P2B	78			
RE2		I/O	ST	Digital I/O.
AD10		I/O	TTL	External memory address/data 10.
$\overline{CS}$		I	TTL	Chip select control for Parallel Slave Port.
P2B		O	—	ECCP2 PWM output B.
RE3/AD11/P3C	77			
RE3		I/O	ST	Digital I/O.
AD11		I/O	TTL	External memory address/data 11.
P3C(3)		O	—	ECCP3 PWM output C.
RE4/AD12/P3B	76			
RE4		I/O	ST	Digital I/O.
AD12		I/O	TTL	External memory address/data 12.
P3B(3)		O	—	ECCP3 PWM output B.
RE5/AD13/P1C	75			
RE5		I/O	ST	Digital I/O.
AD13		I/O	TTL	External memory address/data 13.
P1C(3)		O	—	ECCP1 PWM output C.
RE6/AD14/P1B	74			
RE6		I/O	ST	Digital I/O.
AD14		I/O	TTL	External memory address/data 14.
P1B(3)		O	—	ECCP1 PWM output B.
RE7/AD15/ECCP2/P2A	73			
RE7		I/O	ST	Digital I/O.
AD15		I/O	TTL	External memory address/data 15.
ECCP2(4)		I/O	ST	Capture 2 input/Compare 2 output/PWM 2 output.
P2A(4)		O	—	ECCP2 PWM output A.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; Analog = Analog input; I = Input; O = Output; P = Power; OD = Open-Drain (no P diode to V_{DD}); I^2C/SMB = I^2C™/SMBus input buffer

Note 1: Alternate assignment for ECCP2/P2A when Configuration bit, CCP2MX, is cleared (Extended Microcontroller mode).

2: Default assignment for ECCP2/P2A for all devices in all operating modes (CCP2MX is set).

3: Default assignments for P1B/P1C/P3B/P3C (ECCPMX Configuration bit is set).

4: Alternate assignment for ECCP2/P2A when CCP2MX is cleared (Microcontroller mode).

5: Alternate assignments for P1B/P1C/P3B/P3C (ECCPMX Configuration bit is cleared).

TABLE 1-4: PIC18F8XJ10/8XJ15 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number	Pin Type	Buffer Type	Description
	TQFP			
				PORTH is a bidirectional I/O port.
RH0/A16	79			
RH0		I/O	ST	Digital I/O.
A16		I/O	TTL	External memory address/data 16.
RH1/A17	80			
RH1		I/O	ST	Digital I/O.
A17		I/O	TTL	External memory address/data 17.
RH2/A18	1			
RH2		I/O	ST	Digital I/O.
A18		I/O	TTL	External memory address/data 18.
RH3/A19	2			
RH3		I/O	ST	Digital I/O.
A19		I/O	TTL	External memory address/data 19.
RH4/AN12/P3C	22			
RH4		I/O	ST	Digital I/O.
AN12		I	Analog	Analog input 12.
P3C(5)		O	—	ECCP3 PWM output C.
RH5/AN13/P3B	21			
RH5		I/O	ST	Digital I/O.
AN13		I	Analog	Analog input 13.
P3B(5)		O	—	ECCP3 PWM output B.
RH6/AN14/P1C	20			
RH6		I/O	ST	Digital I/O.
AN14		I	Analog	Analog input 14.
P1C(5)		O	—	ECCP1 PWM output C.
RH7/AN15/P1B	19			
RH7		I/O	ST	Digital I/O.
AN15		I	Analog	Analog input 15.
P1B(5)		O	—	ECCP1 PWM output B.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; Analog = Analog input; I = Input; O = Output; P = Power; OD = Open-Drain (no P diode to VDD); I^2C/SMB = I^2C™/SMBus input buffer

Note 1: Alternate assignment for ECCP2/P2A when Configuration bit, CCP2MX, is cleared (Extended Microcontroller mode).

2: Default assignment for ECCP2/P2A for all devices in all operating modes (CCP2MX is set).

3: Default assignments for P1B/P1C/P3B/P3C (ECCPMX Configuration bit is set).

4: Alternate assignment for ECCP2/P2A when CCP2MX is cleared (Microcontroller mode).

5: Alternate assignments for P1B/P1C/P3B/P3C (ECCPMX Configuration bit is cleared).

6.2.3 INSTRUCTIONS IN PROGRAM MEMORY

The program memory is addressed in bytes. Instructions are stored as two bytes or four bytes in program memory. The Least Significant Byte of an instruction word is always stored in a program memory location with an even address (LSB = 0). To maintain alignment with instruction boundaries, the PC increments in steps of 2 and the LSB will always read '0' (see **Section 6.1.5 "Program Counter"**).

Figure 6-6 shows an example of how instruction words are stored in the program memory.

The CALL and GOTO instructions have the absolute program memory address embedded into the instruction. Since instructions are always stored on word boundaries, the data contained in the instruction is a word address. The word address is written to PC<20:1> which accesses the desired byte address in program memory. Instruction #2 in Figure 6-6 shows how the instruction, GOTO 0006h, is encoded in the program memory. Program branch instructions, which encode a relative address offset, operate in the same manner. The offset value stored in a branch instruction represents the number of single-word instructions that the PC will be offset by. **Section 25.0 "Instruction Set Summary"** provides further details of the instruction set.

FIGURE 6-6: INSTRUCTIONS IN PROGRAM MEMORY

Program Memory Byte Locations →

			LSB = 1	LSB = 0	Word Address ↓
					000000h
					000002h
					000004h
					000006h
Instruction 1:	MOVLW	055h	0Fh	55h	000008h
Instruction 2:	GOTO	0006h	EFh	03h	00000Ah
			F0h	00h	00000Ch
Instruction 3:	MOVFF	123h, 456h	C1h	23h	00000Eh
			F4h	56h	000010h
					000012h
					000014h

6.2.4 TWO-WORD INSTRUCTIONS

The standard PIC18 instruction set has four two-word instructions: CALL, MOVFF, GOTO and LSFR. In all cases, the second word of the instructions always has '1111' as its four Most Significant bits; the other 12 bits are literal data, usually a data memory address.

The use of '1111' in the 4 MSbs of an instruction specifies a special form of NOP. If the instruction is executed in proper sequence – immediately after the first word – the data in the second word is accessed and used by the instruction sequence. If the first word is skipped for some reason and the second word is executed by itself, a NOP is executed instead. This is necessary for cases when the two-word instruction is preceded by a conditional instruction that changes the PC. Example 6-4 shows how this works.

Note:	See **Section 6.5 "Program Memory and the Extended Instruction Set"** for information on two-word instructions in the extended instruction set.

EXAMPLE 6-4: TWO-WORD INSTRUCTIONS

CASE 1:

Object Code	Source Code		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; No, skip this word
1111 0100 0101 0110			; Execute this word as a NOP
0010 0100 0000 0000	ADDWF	REG3	; continue code

CASE 2:

Object Code	Source Code		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; Yes, execute this word
1111 0100 0101 0110			; 2nd word of instruction
0010 0100 0000 0000	ADDWF	REG3	; continue code

6.4.3.2 FSR Registers and 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 that cannot be indirectly read or written to. Accessing these registers actually accesses the associated FSR register pair, but also performs a specific action on its stored value. They are:

- POSTDEC: accesses the FSR value, then automatically decrements it by ‘1’ afterwards
- POSTINC: accesses the FSR value, then automatically increments it by ‘1’ afterwards
- PREINC: increments the FSR value by ‘1’, then uses it 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 new value in the operation

In this context, accessing an INDF register uses the value in the FSR registers without changing them. Similarly, accessing a PLUSW register gives the FSR value offset by the value in the W register; neither value is actually changed in the operation. Accessing the other virtual registers changes the value of the FSR registers.

Operations on the FSRs with POSTDEC, POSTINC and PREINC affect the entire register pair; that is, roll-overs of the FSRnL register from FFh to 00h carry over to the FSRnH register. On the other hand, results of these operations do not change the value of any flags in the STATUS register (e.g., Z, N, OV, etc.).

The PLUSW register can be used to implement a form of Indexed Addressing in the data memory space. By manipulating the value in the W register, users can reach addresses that are fixed offsets from pointer addresses. In some applications, this can be used to implement some powerful program control structure, such as software stacks, inside of data memory.

6.4.3.3 Operations by FSRs on FSRs

Indirect Addressing operations that target other FSRs or virtual registers represent special cases. For example, using an FSR to point to one of the virtual registers will not result in successful operations. As a specific case, assume that FSR0H:FSR0L contains FE7h, the address of INDF1. Attempts to read the value of the INDF1, using INDF0 as an operand, will return 00h. Attempts to write to INDF1, using INDF0 as the operand, will result in a NOP.

On the other hand, using the virtual registers to write to an FSR pair may not occur as planned. In these cases, the value will be written to the FSR pair but without any incrementing or decrementing. Thus, writing to INDF2 or POSTDEC2 will write the same value to the FSR2H:FSR2L.

Since the FSRs are physical registers mapped in the SFR space, they can be manipulated through all direct operations. Users should proceed cautiously when working on these registers, particularly if their code uses Indirect Addressing.

Similarly, operations by Indirect Addressing are generally permitted on all other SFRs. Users should exercise the appropriate caution that they do not inadvertently change settings that might affect the operation of the device.

7.5.2 WRITE VERIFY

Depending on the application, good programming practice may dictate that the value written to the memory should be verified against the original value. This should be used in applications where excessive writes can stress bits near the specification limit.

7.5.3 UNEXPECTED TERMINATION OF WRITE OPERATION

If a write is terminated by an unplanned event, such as loss of power or an unexpected Reset, the memory location just programmed should be verified and reprogrammed if needed. If the write operation is interrupted by a $\overline{\text{MCLR}}$ Reset or a WDT Time-out Reset during normal operation, the user can check the WRERR bit and rewrite the location(s) as needed.

7.6 Flash Program Operation During Code Protection

See **Section 24.6 "Program Verification and Code Protection"** for details on code protection of Flash program memory.

TABLE 7-2: REGISTERS ASSOCIATED WITH PROGRAM FLASH MEMORY

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TBLPTRU	—	—	bit 21	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					53
TBPLTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								53
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								53
TABLAT	Program Memory Table Latch								53
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	53
EECON2	Program Memory Control Register 2 (not a physical register)								55
EECON1	—	—	—	FREE	WRERR	WREN	WR	—	55

Legend: — = unimplemented, read as '0'. Shaded cells are not used during program memory access.

8.1 External Memory Bus Control

The operation of the interface is controlled by the MEMCON register (Register 8-1). This register is available in all program memory operating modes except Microcontroller mode. In this mode, the register is disabled and cannot be written to.

The EBDIS bit (MEMCON<7>) controls the operation of the bus and related port functions. Clearing EBDIS enables the interface and disables the I/O functions of the ports, as well as any other functions multiplexed to those pins. Setting the bit enables the I/O ports and other functions, but allows the interface to override everything else on the pins when an external memory operation is required. By default, the external bus is always enabled and disables all other I/O.

The operation of the EBDIS bit is also influenced by the program memory mode being used. This is discussed in more detail in **Section 8.5 "Program Memory Modes and the External Memory Bus"**.

The WAIT bits allow for the addition of wait states to external memory operations. The use of these bits is discussed in **Section 8.3 "Wait States"**.

The WM bits select the particular operating mode used when the bus is operating in 16-Bit Data Width mode. These are discussed in more detail in **Section 8.6 "16-Bit Data Width Modes"**. These bits have no effect when an 8-Bit Data Width mode is selected.

REGISTER 8-1: MEMCON: EXTERNAL MEMORY BUS CONTROL REGISTER

R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0
EBDIS	—	WAIT1	WAIT0	—	—	WM1	WM0
bit 7							bit 0

Legend:	S = Settable bit		
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 **EBDIS**: External Bus Disable bit
- 1 = External bus enabled when microcontroller accesses external memory; otherwise, all external bus drivers are mapped as I/O ports
- 0 = External bus always enabled, I/O ports are disabled

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

bit 5-4 **WAIT<1:0>:** Table Reads and Writes Bus Cycle Wait Count bits
- 11 = Table reads and writes will wait 0 TCY
- 10 = Table reads and writes will wait 1 TCY
- 01 = Table reads and writes will wait 2 TCY
- 00 = Table reads and writes will wait 3 TCY

bit 3-2 **Unimplemented:** Read as '0

bit 1-0 **WM<1:0>:** TBLWT Operation with 16-Bit Data Bus Width Select bits
- 1x = Word Write mode: TABLAT0 and TABLAT1 word output; $\overline{WRH}$ active when TABLAT1 written
- 01 = Byte Select mode: TABLAT data copied on both MSB and LSB; $\overline{WRH}$ and ($\overline{UB}$ or $\overline{LB}$) will activate
- 00 = Byte Write mode: TABLAT data copied on both MSB and LSB; $\overline{WRH}$ or $\overline{WRL}$ will activate

11.4 PORTC, TRISC and LATC Registers

PORTC is an 8-bit wide, bidirectional port. The corresponding Data Direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin). Only PORTC pins, RC2 through RC7, are digital only pins and can tolerate input voltages up to 5.5V.

The Output Latch register (LATC) is also memory mapped. Read-modify-write operations on the LATC register read and write the latched output value for PORTC.

PORTC is multiplexed with several peripheral functions (Table 11-7). The pins have Schmitt Trigger input buffers. RC1 is normally configured by Configuration bit, CCP2MX, as the default peripheral pin for the ECCP2 module and enhanced PWM output, P2A (default state, CCP2MX = 1).

When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

Note:	These pins are configured as digital inputs on any device Reset.

The contents of the TRISC register are affected by peripheral overrides. Reading TRISC always returns the current contents, even though a peripheral device may be overriding one or more of the pins.

EXAMPLE 11-3: INITIALIZING PORTC

```
CLRF    PORTC   ; Initialize PORTC by
                ; clearing output
                ; data latches
CLRF    LATC    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISC   ; Set RC<3:0> as inputs
                ; RC<5:4> as outputs
                ; RC<7:6> as inputs
```

11.9 PORTH, LATH and TRISH Registers

Note:	PORTH is available only on 80-pin devices.

PORTH is an 8-bit wide, bidirectional I/O port. The corresponding Data Direction register is TRISH. Setting a TRISH bit (= 1) will make the corresponding PORTH pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISH bit (= 0) will make the corresponding PORTH pin an output (i.e., put the contents of the output latch on the selected pin). PORTH<3:0> pins are digital only and tolerate voltages up to 5.5V.

The Output Latch register (LATH) is also memory mapped. Read-modify-write operations on the LATH register read and write the latched output value for PORTH.

All pins on PORTH are implemented with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

When the external memory interface is enabled, four of the PORTH pins function as the high-order address lines for the interface. The address output from the interface takes priority over other digital I/O. The corresponding TRISH bits are also overridden.

PORTH pins, RH4 through RH7, are multiplexed with analog converter inputs. The operation of these pins as analog inputs is selected by clearing or setting the PCFG<3:0> control bits in the ADCON1 register.

PORTH can also be configured as the alternate Enhanced PWM output Channels B and C for the ECCP1 and ECCP3 modules. This is done by clearing the ECCPMX Configuration bit.

EXAMPLE 11-8: INITIALIZING PORTH

```
CLRF    PORTH       ; Initialize PORTH by
                    ; clearing output
                    ; data latches
CLRF    LATH        ; Alternate method
                    ; to clear output
                    ; data latches
MOVLW   0Fh         ; Configure PORTH as
MOVWF   ADCON1      ; digital I/O
MOVLW   0CFh        ; Value used to
                    ; initialize data
                    ; direction
MOVWF   TRISH       ; Set RH3:RH0 as inputs
                    ; RH5:RH4 as outputs
                    ; RH7:RH6 as inputs
```

FIGURE 11-4: PARALLEL SLAVE PORT READ WAVEFORMS

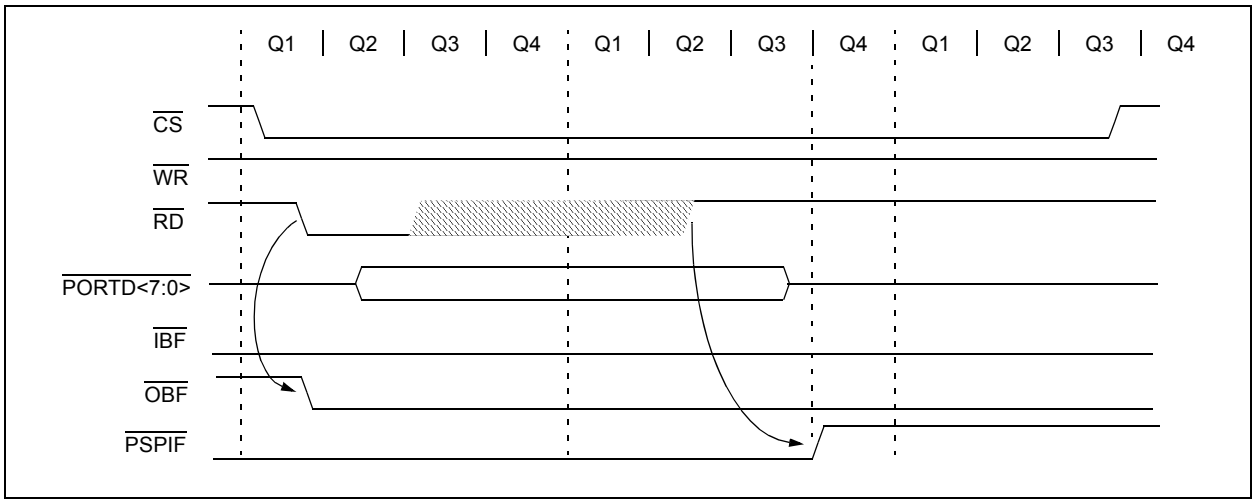


TABLE 11-21: REGISTERS ASSOCIATED WITH PARALLEL SLAVE PORT

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	56
LATD	LATD7	LATD6	LATD5	LATD4	LATD3	LATD2	LATD1	LATD0	56
TRISD	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	56
PORTE	RE7	RE6	RE5	RE4	RE3	RE2	RE1	RE0	56
LATE	LATE7	LATE6	LATE5	LATE4	LATE3	LATE2	LATE1	LATE0	56
TRISE	TRISE7	TRISE6	TRISE5	TRISE4	TRISE3	TRISE2	TRISE1	TRISE0	56
PSPCON	IBF	OBF	IBOV	PSPMODE	—	—	—	—	55
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	53
PIR1	PSPIF	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF	55
PIE1	PSPIE	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE	55
IPR1	PSPIP	ADIP	RC1IP	TX1IP	SSP1IP	CCP1IP	TMR2IP	TMR1IP	55

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the Parallel Slave Port.

17.4 PWM Mode

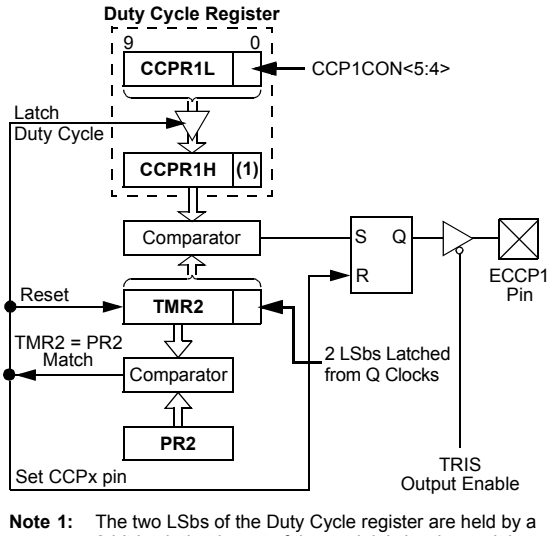
In Pulse-Width Modulation (PWM) mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP4 and CCP5 pins are multiplexed with a PORTG data latch, the appropriate TRISG bit must be cleared to make the CCP4 or CCP5 pin an output.

Note:	Clearing the CCP4CON or CCP5CON register will force the RG3 or RG4 output latch (depending on device configuration) to the default low level. This is not the PORTG I/O data latch.

Figure 17-4 shows a simplified block diagram of the CCP module in PWM mode.

For a step-by-step procedure on how to set up a CCP module for PWM operation, see **Section 17.4.3 "Setup for PWM Operation"**.

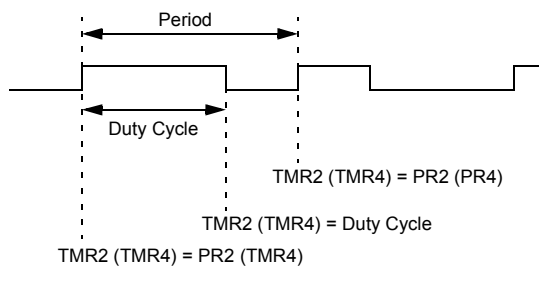
FIGURE 17-4: SIMPLIFIED PWM BLOCK DIAGRAM



Note 1: The two LSbs of the Duty Cycle register are held by a 2-bit latch that is part of the module's hardware. It is physically separate from the CCPR registers.

A PWM output (Figure 17-5) has a time base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

FIGURE 17-5: PWM OUTPUT



17.4.1 PWM PERIOD

The PWM period is specified by writing to the PR2 (PR4) register. The PWM period can be calculated using Equation 17-1:

EQUATION 17-1:

$$\text{PWM Period} = [(\text{PR2}) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

PWM frequency is defined as 1/[PWM period].

When TMR2 (TMR4) is equal to PR2 (PR4), the following three events occur on the next increment cycle:

- TMR2 (TMR4) is cleared
- The CCPx pin is set (exception: if PWM duty cycle = 0%, the CCPx pin will not be set)
- The PWM duty cycle is latched from CCPRxL into CCPRxH

Note:	The Timer2 and Timer 4 postscalers (see **Section 14.0 "Timer2 Module"** and **Section 16.0 "Timer4 Module"**) are not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

17.4.2 PWM DUTY CYCLE

The PWM duty cycle is specified by writing to the CCPRxL register and to the CCPxCON<5:4> bits. Up to 10-bit resolution is available. The CCPRxL contains the eight MSbs and the CCPxCON<5:4> contains the two LSbs. This 10-bit value is represented by CCPRxL:CCPxCON<5:4>. Equation 17-2 is used to calculate the PWM duty cycle in time.

EQUATION 17-2:

$$\text{PWM Duty Cycle} = (\text{CCPRxL:CCPxCON<5:4>}) \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

CCPRxL and CCPxCON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPRxH until after a match between PR2 (PR4) and TMR2 (TMR4) occurs (i.e., the period is complete). In PWM mode, CCPRxH is a read-only register.

19.4.3.3 Reception

When the R/$\overline{W}$ bit of the address byte is clear and an address match occurs, the R/$\overline{W}$ bit of the SSPxSTAT register is cleared. The received address is loaded into the SSPxBUF register and the SDAx line is held low ($\overline{ACK}$).

When the address byte overflow condition exists, then the no Acknowledge ($\overline{ACK}$) pulse is given. An overflow condition is defined as either bit, BF (SSPxSTAT<0>), is set, or bit, SSPOV (SSPxCON1<6>), is set.

An MSSP interrupt is generated for each data transfer byte. The interrupt flag bit, SSPxIF, must be cleared in software. The SSPxSTAT register is used to determine the status of the byte.

If SEN is enabled (SSPxCON2<0> = 1), SCLx will be held low (clock stretch) following each data transfer. The clock must be released by setting bit, CKP (SSPxCON1<4>). See **Section 19.4.4 "Clock Stretching"** for more detail.

19.4.3.4 Transmission

When the R/$\overline{W}$ bit of the incoming address byte is set and an address match occurs, the R/$\overline{W}$ bit of the SSPxSTAT register is set. The received address is loaded into the SSPxBUF register. The $\overline{ACK}$ pulse will be sent on the ninth bit and the SCLx pin is held low regardless of SEN (see **Section 19.4.4 "Clock Stretching"** for more detail). By stretching the clock, the master will be unable to assert another clock pulse until the slave is done preparing the transmit data. The transmit data must be loaded into the SSPxBUF register which also loads the SSPxSR register. Then pin, SCLx, should be enabled by setting bit, CKP (SSPxCON1<4>). The eight data bits are shifted out on the falling edge of the SCLx input. This ensures that the SDAx signal is valid during the SCLx high time (Figure 19-10).

The $\overline{ACK}$ pulse from the master-receiver is latched on the rising edge of the ninth SCLx input pulse. If the SDAx line is high (not $\overline{ACK}$), then the data transfer is complete. In this case, when the $\overline{ACK}$ is latched by the slave, the slave logic is reset and the slave monitors for another occurrence of the Start bit. If the SDAx line was low ($\overline{ACK}$), the next transmit data must be loaded into the SSPxBUF register. Again, pin, SCLx, must be enabled by setting bit, CKP.

An MSSP interrupt is generated for each data transfer byte. The SSPxIF bit must be cleared in software and the SSPxSTAT register is used to determine the status of the byte. The SSPxIF bit is set on the falling edge of the ninth clock pulse.

19.4.17.1 Bus Collision During a Start Condition

During a Start condition, a bus collision occurs if:

a) SDAx or SCLx are sampled low at the beginning of the Start condition (Figure 19-28).

b) SCLx is sampled low before SDAx is asserted low (Figure 19-29).

During a Start condition, both the SDAx and the SCLx pins are monitored.

If the SDAx pin is already low, or the SCLx pin is already low, then all of the following occur:

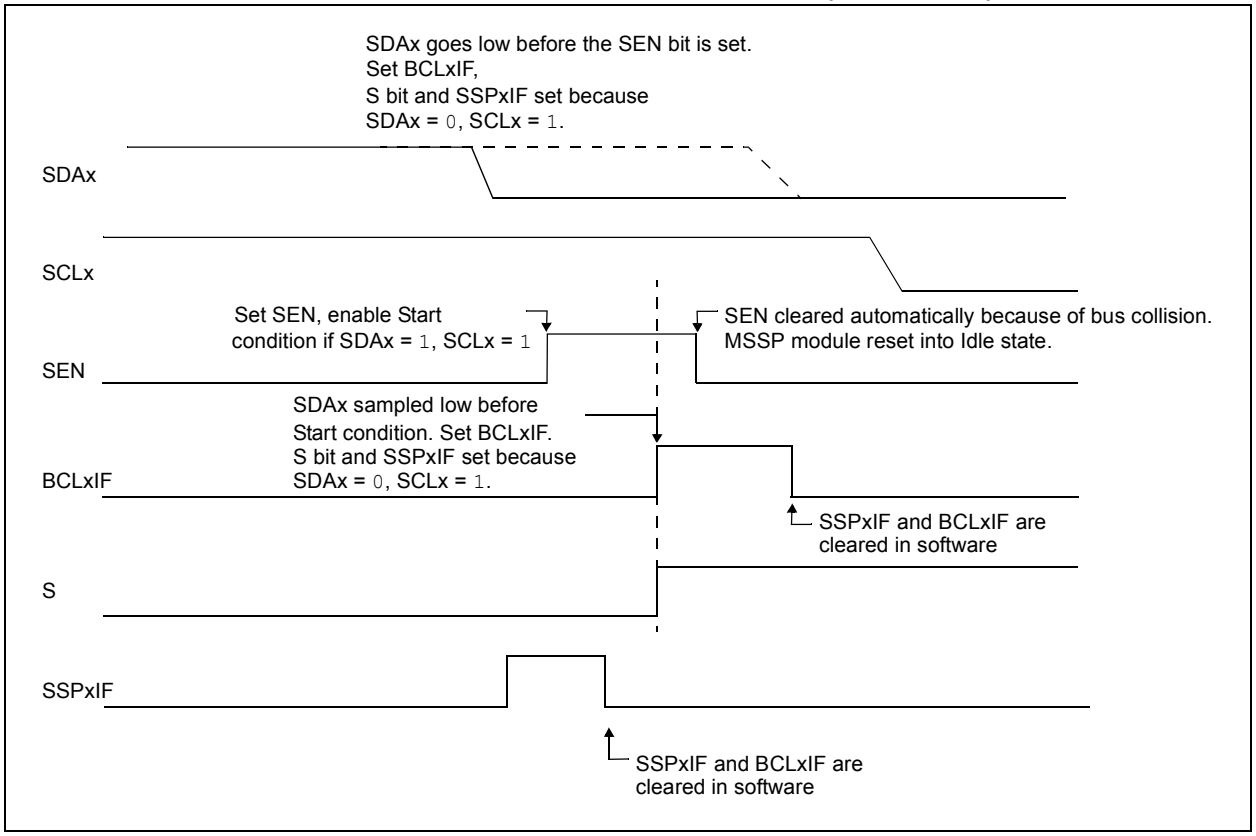
- the Start condition is aborted,
- the BCLxIF flag is set and
- the MSSP module is reset to its inactive state (Figure 19-28).

The Start condition begins with the SDAx and SCLx pins deasserted. When the SDAx pin is sampled high, the Baud Rate Generator is loaded from SSPxADD<6:0> and counts down to 0. If the SCLx pin is sampled low while SDAx is high, a bus collision occurs because it is assumed that another master is attempting to drive a data '1' during the Start condition.

If the SDAx pin is sampled low during this count, the BRG is reset and the SDAx line is asserted early (Figure 19-30). If, however, a '1' is sampled on the SDAx pin, the SDAx pin is asserted low at the end of the BRG count. The Baud Rate Generator is then reloaded and counts down to 0. If the SCLx pin is sampled as '0' during this time, a bus collision does not occur. At the end of the BRG count, the SCLx pin is asserted low.

Note:	The reason that bus collision is not a factor during a Start condition is that no two bus masters can assert a Start condition at the exact same time. Therefore, one master will always assert SDAx before the other. This condition does not cause a bus collision because the two masters must be allowed to arbitrate the first address following the Start condition. If the address is the same, arbitration must be allowed to continue into the data portion, Repeated Start or Stop conditions.

FIGURE 19-28: BUS COLLISION DURING START CONDITION (SDAx ONLY)



20.2.4.1 Special Considerations Using Auto-Wake-up

Since auto-wake-up functions by sensing rising edge transitions on RXx/DTx, information with any state changes before the Stop bit may signal a false end-of-character and cause data or framing errors. To work properly, therefore, the initial character in the transmission must be all '0's. This can be 00h (8 bytes) for standard RS-232 devices or 000h (12 bits) for LIN bus.

Oscillator start-up time must also be considered, especially in applications using oscillators with longer start-up intervals (i.e., HS or HSPLL mode). The Sync Break (or Wake-up Signal) character must be of sufficient length and be followed by a sufficient interval to allow enough time for the selected oscillator to start and provide proper initialization of the EUSART.

20.2.4.2 Special Considerations Using the WUE Bit

The timing of WUE and RCxIF events may cause some confusion when it comes to determining the validity of received data. As noted, setting the WUE bit places the EUSART in an Idle mode. The wake-up event causes a receive interrupt by setting the RCxIF bit. The WUE bit is cleared after this when a rising edge is seen on RXx/DTx. The interrupt condition is then cleared by reading the RCREGx register. Ordinarily, the data in RCREGx will be dummy data and should be discarded.

The fact that the WUE bit has been cleared (or is still set) and the RCxIF flag is set should not be used as an indicator of the integrity of the data in RCREGx. Users should consider implementing a parallel method in firmware to verify received data integrity.

To assure that no actual data is lost, check the RCIDL bit to verify that a receive operation is not in process. If a receive operation is not occurring, the WUE bit may then be set just prior to entering the Sleep mode.

FIGURE 20-8: AUTO-WAKE-UP BIT (WUE) TIMINGS DURING NORMAL OPERATION

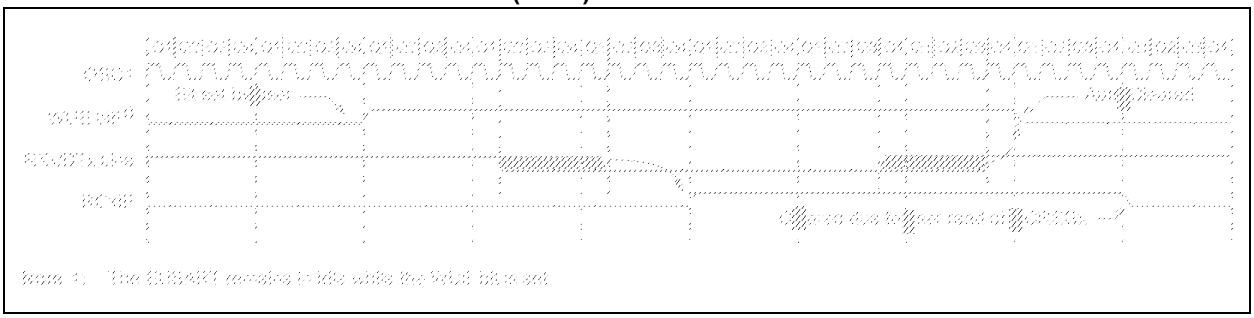


FIGURE 20-9: AUTO-WAKE-UP BIT (WUE) TIMINGS DURING SLEEP

REGISTER 21-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
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 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit (V_{REF-} source)
1 = V_{REF-} (AN2)
0 = AV_{SS}

bit 4 **VCFG0:** Voltage Reference Configuration bit (V_{REF+} source)
1 = V_{REF+} (AN3)
0 = AV_{DD}

bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits:

PCFG<3:0>	AN15(1)	AN14(1)	AN13(1)	AN12(1)	AN11	AN10	AN9	AN8	AN7	AN6	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	D	D	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	D	D	D	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	D	D	D	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	D	D	D	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	D	D	D	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	D	D	D	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	D	D	D	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Note 1: AN12 through AN15 are available only in 80-pin devices.

NOTES:

ANDWF **AND W with f**

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

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

Operation: (W) .AND. (f) → dest

Status Affected: N, Z

Encoding:

0001	01da	ffff	ffff

Description: The contents of W are ANDed 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'.

If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.

If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 25.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ANDWF REG, 0, 0`

Before Instruction
W = 17h
REG = C2h
After Instruction
W = 02h
REG = C2h

BC **Branch if Carry**

Syntax: BC n

Operands: $-128 \le n \le 127$

Operation: if Carry bit is '1',
(PC) + 2 + 2n → 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)

GOTO **Unconditional Branch**

Syntax: GOTO k

Operands: $0 \le k \le 1048575$

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

Status Affected: None

Encoding:

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

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

Words: 2

Cycles: 2

Q Cycle Activity:

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

Example: GOTO THERE

After Instruction
PC = Address (THERE)

INCF **Increment f**

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

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

Operation: (f) + 1 → dest

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

Encoding:

0010	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f'.

If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.

If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 25.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: INCF CNT, 1, 0

Before Instruction
CNT = FFh
Z = 0
C = ?
DC = ?
After Instruction
CNT = 00h
Z = 1
C = 1
DC = 1

MOVFF **Move f to f**

Syntax: MOVFF f_s,f_d

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

Operation: $(f_s) \rightarrow f_d$

Status Affected: None

Encoding:

1st word (source)	1100	ffff	ffff	$ffff_s$
2nd word (destin.)	1111	ffff	ffff	$ffff_d$

Description: The contents of source register 'f_s' are moved to destination register 'f_d'. Location of source 'f_s' can be anywhere in the 4096-byte data space (000h to FFFh) and location of destination 'f_d' can also be anywhere from 000h to FFFh.

Either source or destination can be W (a useful special situation).

MOVFF is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

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

Words: 2

Cycles: 2

Q Cycle Activity:

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

Example: MOVFF REG1, REG2

Before Instruction
REG1 = 33h
REG2 = 11h
After Instruction
REG1 = 33h
REG2 = 33h

MOVLB **Move Literal to Low Nibble in BSR**

Syntax: MOVLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow BSR$

Status Affected: None

Encoding:

0000	0001	kkkk	kkkk

Description: The eight-bit literal 'k' is loaded into the Bank Select Register (BSR). The value of BSR<7:4> always remains '0' regardless of the value of k_7:k_4.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: MOVLB 5

Before Instruction
BSR Register = 02h
After Instruction
BSR Register = 05h

CALLW **Subroutine Call using WREG**

Syntax: CALLW

Operands: None

Operation: (PC + 2) → TOS,
(W) → PCL,
(PCLATH) → PCH,
(PCLATU) → PCU

Status Affected: None

Encoding:

0000	0000	0001	0100

Description: First, the return address (PC + 2) is pushed onto the return stack. Next, the contents of W are written to PCL; the existing value is discarded. Then, the contents of PCLATH and PCLATU are latched into PCH and PCU, respectively. The second cycle is executed as a NOP instruction while the new next instruction is fetched.

Unlike CALL, there is no option to update W, STATUS or BSR.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read WREG	Push PC to stack	No operation
No operation	No operation	No operation	No operation

Example: HERE CALLW

Before Instruction
PC = address (HERE)
PCLATH = 10h
PCLATU = 00h
W = 06h

After Instruction
PC = 001006h
TOS = address (HERE + 2)
PCLATH = 10h
PCLATU = 00h
W = 06h

MOVSF **Move Indexed to f**

Syntax: MOVSF [z_s], f_d

Operands: $0 \le z_s \le 127$
$0 \le f_d \le 4095$

Operation: ((FSR2) + z_s) → f_d

Status Affected: None

Encoding:

1st word (source)	1110	1011	0zzz	$zzzz_s$
2nd word (destin.)	1111	ffff	ffff	$ffff_d$

Description: The contents of the source register are moved to destination register 'f_d'. The actual address of the source register is determined by adding the 7-bit literal offset 'z_s', in the first word, to the value of FSR2. The address of the destination register is specified by the 12-bit literal 'f_d' in the second word. Both addresses can be anywhere in the 4096-byte data space (000h to FFFh).

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

If the resultant source address points to an Indirect Addressing register, the value returned will be 00h.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Determine source addr	Determine source addr	Read source reg
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: MOVSF [05h], REG2

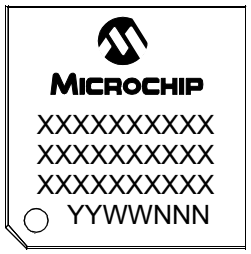
Before Instruction
FSR2 = 80h
Contents of 85h = 33h
REG2 = 11h

After Instruction
FSR2 = 80h
Contents of 85h = 33h
REG2 = 33h

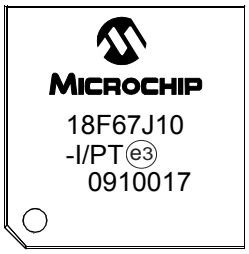
28.0 PACKAGING INFORMATION

28.1 Package Marking Information

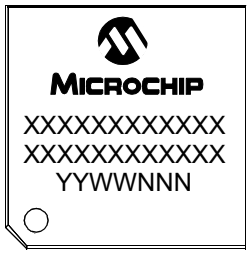
64-Lead TQFP



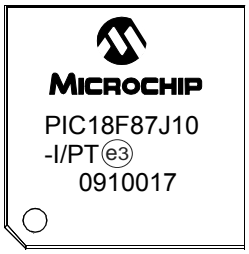
Example



80-Lead TQFP



Example



Legend:	XX...X	Customer-specific information
	Y	Year code (last digit of calendar year)
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week '01')
	NNN	Alphanumeric traceability code
	(e3)	Pb-free JEDEC designator for Matte Tin (Sn)
	*	This package is Pb-free. The Pb-free JEDEC designator ((e3)) can be found on the outer packaging for this package.

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line, thus limiting the number of available characters for customer-specific information.