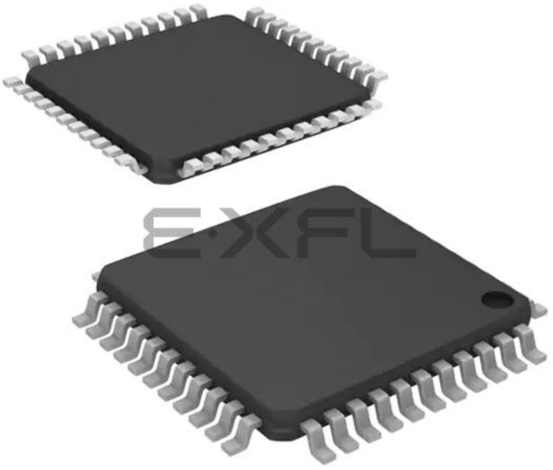
Microchip Technology - PIC18F46K80T-I/PT Datasheet



What is "Embedded - Microcontrollers"?

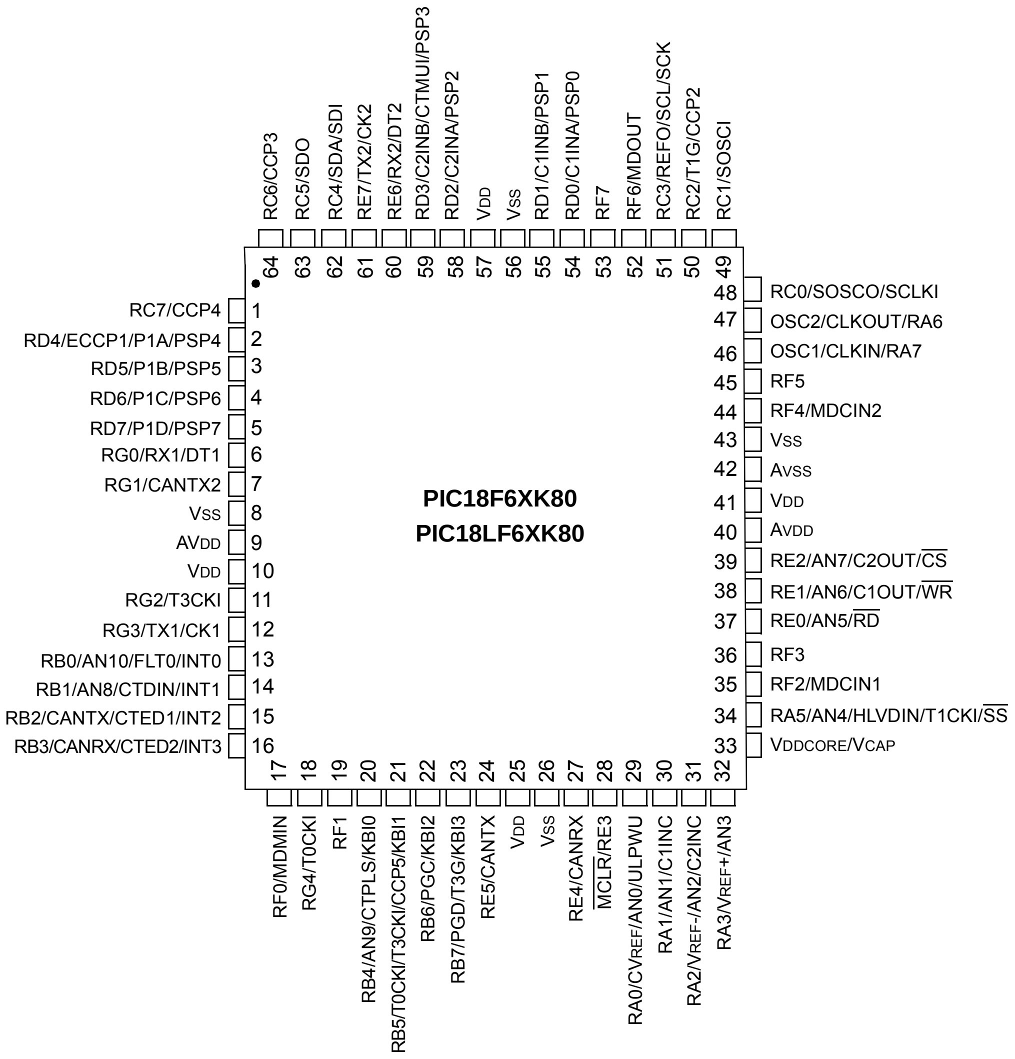
"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" 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	ECANbus, I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	35
Program Memory Size	64KB (32K x 16)
Program Memory Type	FLASH
EEPROM Size	1K x 8
RAM Size	3.6K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 5.5V
Data Converters	A/D 11x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	44-TQFP
Supplier Device Package	44-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f46k80t-i-pt

Pin Diagrams (Continued)

64-Pin QFN(1)/TQFP



Note 1: For the QFN package, it is recommended that the bottom pad be connected to VSS.

TABLE 1-3: DEVICE FEATURES FOR THE PIC18F6XK80 (64-PIN DEVICES)

Features	PIC18F65K80	PIC18F66K80
Operating Frequency	DC – 64 MHz	
Program Memory (Bytes)	32K	64K
Program Memory (Instructions)	16,384	32,768
Data Memory (Bytes)	3.6K	
Interrupt Sources	32	
I/O Ports	Ports A, B, C, D, E, F, G	
Parallel Communications	Parallel Slave Port (PSP)	
Timers	Five	
Comparators	Two	
CTMU	Yes	
Capture/Compare/PWM (CCP) Modules	Four	
Enhanced CCP (ECCP) Modules	One	
DSM	Yes	Yes
Serial Communications	One MSSP and Two Enhanced USARTs (EUSART)	
12-Bit Analog-to-Digital Module	Eleven Input Channels	
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow, $\overline{\text{MCLR}}$, WDT (PWRT, OST)	
Instruction Set	75 Instructions, 83 with Extended Instruction Set Enabled	
Packages	64-Pin QFN and TQFP	

4.1.3 CLOCK TRANSITIONS AND STATUS INDICATORS

The length of the transition between clock sources is the sum of two cycles of the old clock source and three to four cycles of the new clock source. This formula assumes that the new clock source is stable. The HF-INTOSC and MF-INTOSC are termed as INTOSC in this chapter.

Three bits indicate the current clock source and its status, as shown in Table 4-2. The three bits are:

- OSTS (OSCCON<3>)
- HFIOFS (OSCCON<2>)
- SOSCRUN (OSCCON2<6>)

TABLE 4-2: SYSTEM CLOCK INDICATOR

Main Clock Source	OSTS	HFIOFS or MFIOFS	SOSCRUN
Primary Oscillator	1	0	0
INTOSC (HF-INTOSC or MF-INTOSC)	0	1	0
Secondary Oscillator	0	0	1
MF-INTOSC or HF-INTOSC as Primary Clock Source	1	1	0
LF-INTOSC is Running or INTOSC is Not Yet Stable	0	0	0

When the OSTS bit is set, the primary clock is providing the device clock. When the HFIOFS or MFIOFS bit is set, the INTOSC output is providing a stable clock source to a divider that actually drives the device clock. When the SOSCRUN bit is set, the SOSC oscillator is providing the clock. If none of these bits are set, either the LF-INTOSC clock source is clocking the device or the INTOSC source is not yet stable.

If the internal oscillator block is configured as the primary clock source by the FOSC<3:0> Configuration bits (CONFIG1H<3:0>). Then, the OSTS and HFIOFS or MFIOFS bits can be set when in PRI_RUN or PRI_IDLE mode. This indicates that the primary clock (INTOSC output) is generating a stable output. Entering another INTOSC power-managed mode at the same frequency would clear the OSTS bit.

Note 1: Caution should be used when modifying a single IRCF bit. At a lower VDD, it is possible to select a higher clock speed than is supportable by that VDD. Improper device operation may result if the VDD/FOSC specifications are violated.

2: Executing a SLEEP instruction does not necessarily place the device into Sleep mode. It acts as the trigger to place the controller into either the Sleep mode, or one of the Idle modes, depending on the setting of the IDLEN bit.

4.1.4 MULTIPLE SLEEP COMMANDS

The power-managed mode that is invoked with the SLEEP instruction is determined by the setting of the IDLEN bit at the time the instruction is executed. If another SLEEP instruction is executed, the device will enter the power-managed mode specified by IDLEN at that time. If IDLEN has changed, the device will enter the new power-managed mode specified by the new setting.

4.2 Run Modes

In the Run modes, clocks to both the core and peripherals are active. The difference between these modes is the clock source.

4.2.1 PRI_RUN MODE

The PRI_RUN mode is the normal, full-power execution mode of the microcontroller. This is also the default mode upon a device Reset, unless Two-Speed Start-up is enabled. (For details, see **Section 28.4 “Two-Speed Start-up”**.) In this mode, the OSTS bit is set. The HFIOFS or MFIOFS bit may be set if the internal oscillator block is the primary clock source. (See **Section 3.2 “Control Registers”**.)

4.2.2 SEC_RUN MODE

The SEC_RUN mode is the compatible mode to the “clock-switching” feature offered in other PIC18 devices. In this mode, the CPU and peripherals are clocked from the SOSC oscillator. This enables lower power consumption while retaining a high-accuracy clock source.

SEC_RUN mode is entered by setting the SCS<1:0> bits to ‘01’. The device clock source is switched to the SOSC oscillator (see Figure 4-1), the primary oscillator is shut down, the SOSCRUN bit (OSCCON2<6>) is set and the OSTS bit is cleared.

Note: The SOSC oscillator can be enabled by setting the SOSCGO bit (OSCCON2<3>). If this bit is set, the clock switch to the SEC_RUN mode can switch immediately once SCS<1:0> are set to ‘01’.

On transitions from SEC_RUN mode to PRI_RUN mode, the peripherals and CPU continue to be clocked from the SOSC oscillator while the primary clock is started. When the primary clock becomes ready, a clock switch back to the primary clock occurs (see Figure 4-2). When the clock switch is complete, the SOSCRUN bit is cleared, the OSTS bit is set and the primary clock is providing the clock. The IDLEN and SCSx bits are not affected by the wake-up and the SOSC oscillator continues to run.

6.1.2 PROGRAM COUNTER

The Program Counter (PC) specifies the address of the instruction to fetch for execution. The PC is 21 bits wide and contained in three separate 8-bit registers.

The low byte, known as the PCL register, is both readable and writable. The high byte, or PCH register, contains the PC<15:8> bits and is not directly readable or writable. Updates to the PCH register are performed through the PCLATH register. The upper byte is called PCU. This register contains the PC<20:16> bits; it is also not directly readable or writable. Updates to the PCU register are performed through the PCLATU register.

The contents of PCLATH and PCLATU are transferred to the Program Counter by any operation that writes PCL. Similarly, the upper two bytes of the Program Counter are transferred to PCLATH and PCLATU by an operation that reads PCL. This is useful for computed offsets to the PC (see **Section 6.1.5.1 “Computed GOTO”**).

The PC addresses bytes in the program memory. To prevent the PC from becoming misaligned with word instructions, the Least Significant bit (LSb) of PCL is fixed to a value of ‘0’. The PC increments by two to address sequential instructions in the program memory.

The CALL, RCALL, GOTO and program branch instructions write to the Program Counter directly. For these instructions, the contents of PCLATH and PCLATU are not transferred to the Program Counter.

6.1.3 RETURN ADDRESS STACK

The return address stack enables execution of any combination of up to 31 program calls and interrupts. The PC is pushed onto the stack when a CALL or RCALL instruction is executed or an interrupt is Acknowledged. The PC value is pulled off the stack on a RETURN, RETLW or a RETFIE instruction. The value is also pulled off the stack on ADDULNK and SUBULNK instructions if the extended instruction set is enabled. PCLATU and PCLATH are not affected by any of the RETURN or CALL instructions.

The stack operates as a 31-word by 21-bit RAM and a 5-bit Stack Pointer, STKPTR. The stack space is not part of either program or data space. The Stack Pointer is readable and writable and the address on the top of the stack is readable and writable through the Top-of-Stack (TOS) Special Function Registers. Data can also be pushed to, or popped from the stack, using these registers.

A CALL type instruction causes a push onto the stack. The Stack Pointer is first incremented and the location pointed to by the Stack Pointer is written with the contents of the PC (already pointing to the instruction following the CALL). A RETURN type instruction causes a pop from the stack. The contents of the location pointed to by the STKPTR are transferred to the PC and then the Stack Pointer is decremented.

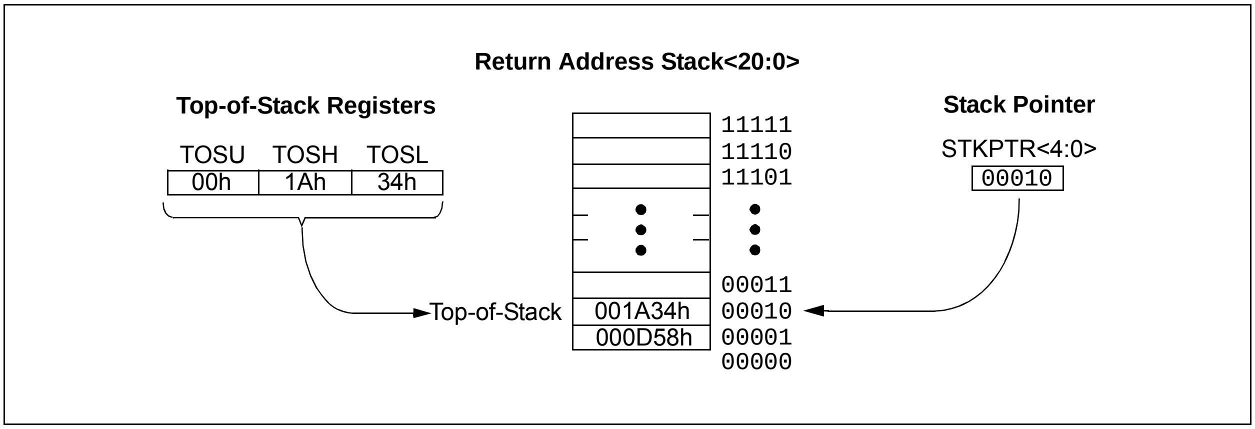
The Stack Pointer is initialized to ‘00000’ after all Resets. There is no RAM associated with the location corresponding to a Stack Pointer value of ‘00000’; this is only a Reset value. Status bits indicate if the stack is full, has overflowed or has underflowed.

6.1.3.1 Top-of-Stack Access

Only the top of the return address stack is readable and writable. A set of three registers, TOSU:TOSH:TOSL, holds the contents of the stack location pointed to by the STKPTR register (Figure 6-3). This allows users to implement a software stack, if necessary. After a CALL, RCALL or interrupt (or ADDULNK and SUBULNK instructions, if the extended instruction set is enabled), the software can read the pushed value by reading the TOSU:TOSH:TOSL registers. These values can be placed on a user-defined software stack. At return time, the software can return these values to TOSU:TOSH:TOSL and do a return.

While accessing the stack, users must disable the Global Interrupt Enable bits to prevent inadvertent stack corruption.

FIGURE 6-3: RETURN ADDRESS STACK AND ASSOCIATED REGISTERS



7.5 Writing to Flash Program Memory

The programming blocks are 32 words or 64 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 64 holding registers for programming by the table writes.

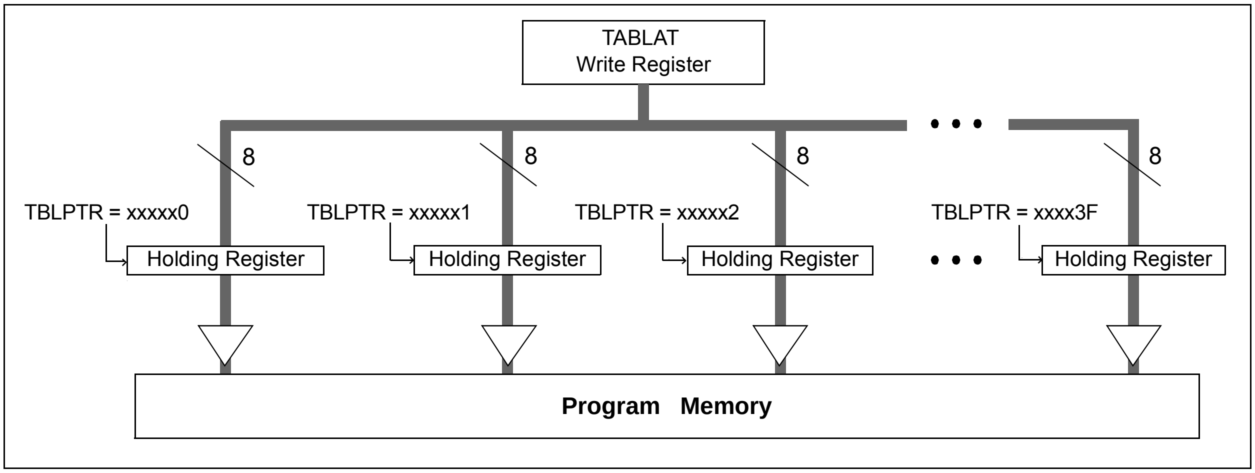
Since the Table Latch (TABLAT) is only a single byte, the TBLWT instruction may need to be executed 64 times for each programming operation. All of the table write operations will essentially be short writes because only the holding registers are written. At the end of updating the 64 or 128 holding registers, the EECON1 register must be written to in order 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 is 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.

Note:	The default value of the holding registers on device Resets and after write operations is FFh. A write of FFh to a holding register does not modify that byte. This means that individual bytes of program memory may be modified, provided that the change does not attempt to change any bit from a '0' to a '1'. When modifying individual bytes, it is not necessary to load all 64 holding registers before executing a write operation.

FIGURE 7-5: TABLE WRITES TO FLASH PROGRAM MEMORY



7.5.1 FLASH PROGRAM MEMORY WRITE SEQUENCE

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

1. Read the 64 bytes into RAM.
2. Update the data values in RAM as necessary.
3. Load the Table Pointer register with the address being erased.
4. Execute the row erase procedure.
5. Load the Table Pointer register with the address of the first byte being written.
6. Write the 64 bytes into the holding registers with auto-increment.
7. Set the EECON1 register for the write operation:
   - Set the EEPGD bit to point to program memory
   - Clear the CFGS bit to access program memory
   - Set the WREN to enable byte writes
8. Disable the interrupts.
9. Write 55h to EECON2.
10. Write 0AAh to EECON2.
11. Set the WR bit. This will begin the write cycle. The CPU will stall for the duration of the write for TIW (see Parameter D133A).
12. Re-enable the interrupts.
13. Verify the memory (table read).

An example of the required code is shown in Example 7-3 on the following page.

Note:	Before setting the WR bit, the Table Pointer address needs to be within the intended address range of the 64 bytes in the holding register.

FIGURE 14-7: TIMER1 GATE SINGLE PULSE AND TOGGLE COMBINED MODE

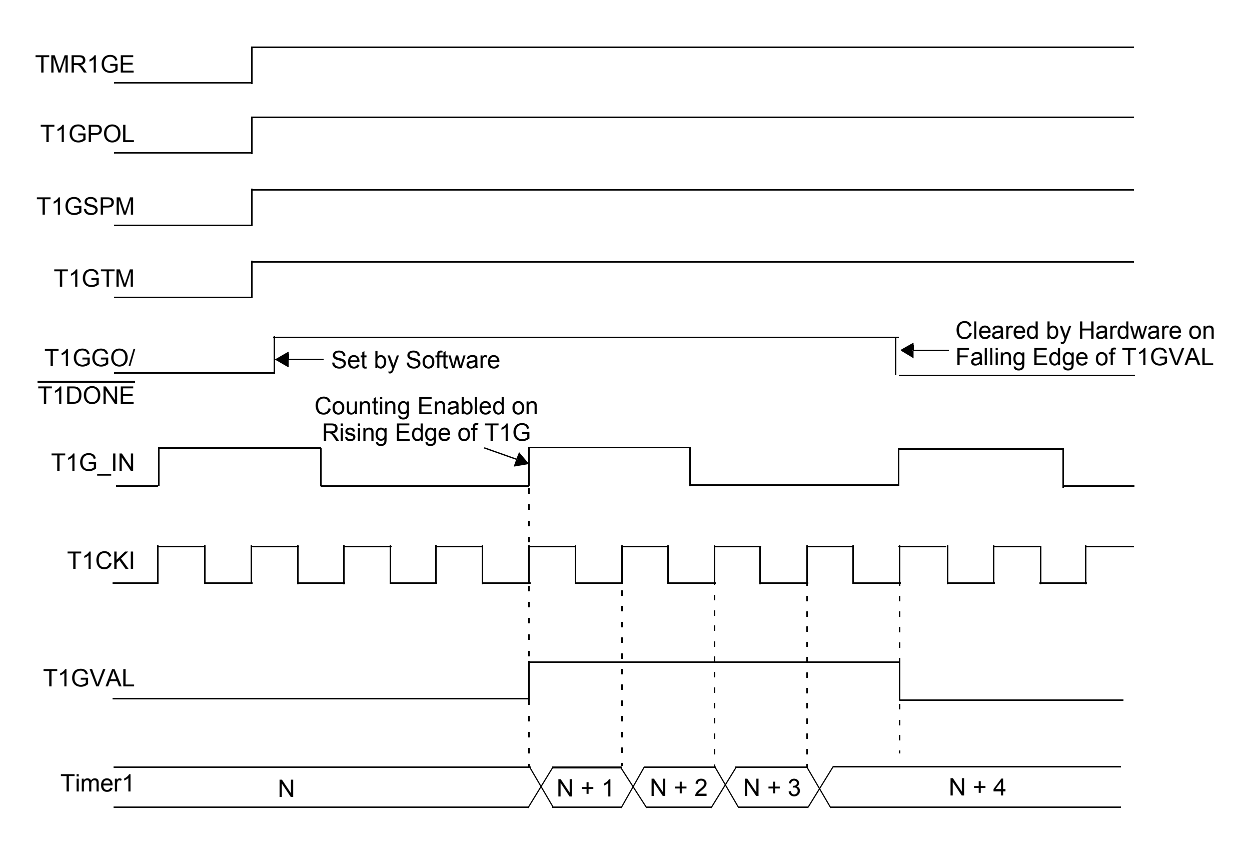


TABLE 14-5: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	PSPIF	ADIF	RC1IF	TX1IF	SSPIF	TMR1GIF	TMR2IF	TMR1IF
PIE1	PSPIE	ADIE	RC1IE	TX1IE	SSPIE	TMR1GIE	TMR2IE	TMR1IE
IPR1	PSPIP	ADIP	RC1IP	TX1IP	SSPIP	TMR1GIP	TMR2IP	TMR1IP
TMR1L	Timer1 Register Low Byte							
TMR1H	Timer1 Register High Byte							
T1CON	TMR1CS1	TMR1CS0	T1CKPS1	T1CKPS0	SOSCEN	$\overline{\text{T1SYNC}}$	RD16	TMR1ON
T1GCON	TMR1GE	T1GPOL	T1GTM	T1GSPM	T1GGO/ $\overline{\text{T1DONE}}$	T1GVAL	T1GSS1	T1GSS0
OSCCON2	—	SOSCRUN	—	SOSCDRV	SOSCGO	—	MFIOFS	MFIOSEL
PMD1	PSPMD	CTMUMD	ADCMD	TMR4MD	TMR3MD	TMR2MD	TMR1MD	TMR0MD

Legend: Shaded cells are not used by the Timer1 module.

16.5 Timer3 Gates

Timer3 can be configured to count freely or the count can be enabled and disabled using the Timer3 gate circuitry. This is also referred to as the Timer3 gate count enable.

The Timer3 gate can also be driven by multiple selectable sources.

16.5.1 TIMER3 GATE COUNT ENABLE

The Timer3 Gate Enable mode is enabled by setting the TMR3GE bit (TxGCON<7>). The polarity of the Timer3 Gate Enable mode is configured using the T3GPOL bit (T3GCON<6>).

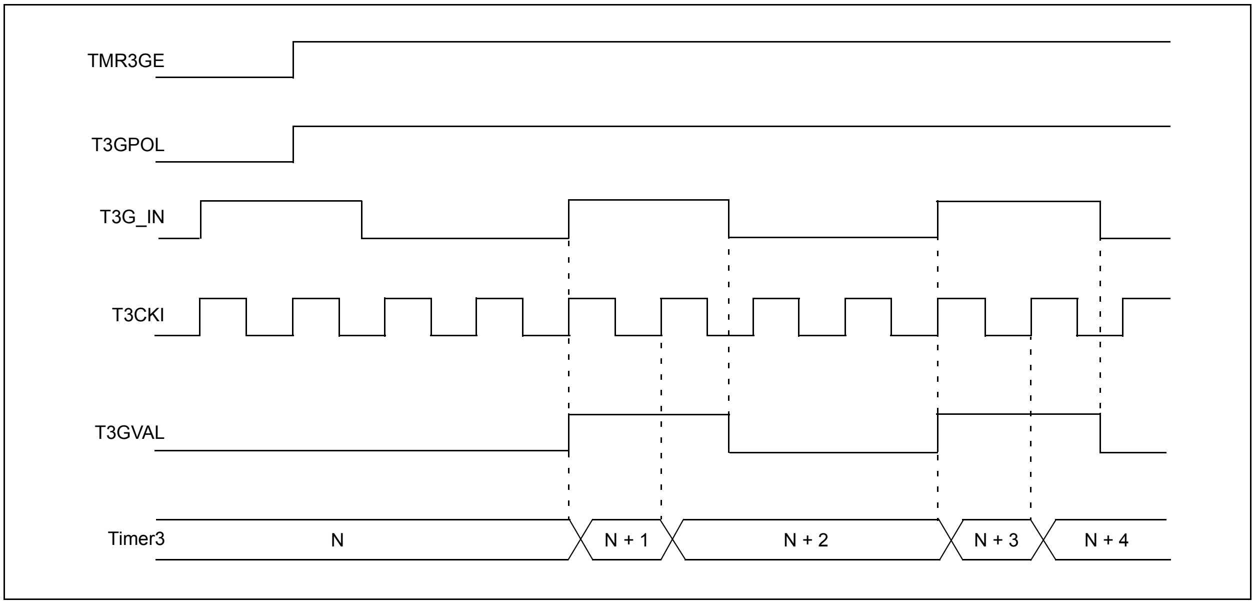
When Timer3 Gate Enable mode is enabled, Timer3 will increment on the rising edge of the Timer3 clock source. When Timer3 Gate Enable mode is disabled, no incrementing will occur and Timer3 will hold the current count. See Figure 16-2 for timing details.

TABLE 16-1: TIMER3 GATE ENABLE SELECTIONS

T3CLK[†]	T3GPOL (T3GCON<6>)	T3G Pin	Timer3 Operation
↑	0	0	Counts
↑	0	1	Holds Count
↑	1	0	Holds Count
↑	1	1	Counts

† The clock on which TMR3 is running. For more information, see T3CLK in Figure 16-1.

FIGURE 16-2: TIMER3 GATE COUNT ENABLE MODE



REGISTER 21-3: SSPSTAT: MSSP STATUS REGISTER (I^{2}C™ MODE)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{\text{A}}$	P[1]	S[1]	R/$\overline{\text{W}}$[2,3]	UA	BF
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 **SMP:** Slew Rate Control bit

In Master or Slave mode:
1 = Slew rate control is disabled for Standard Speed mode (100 kHz and 1 MHz)
0 = Slew rate control is enabled for High-Speed mode (400 kHz)

bit 6 **CKE:** SMBus Select bit

In Master or Slave mode:
1 = Enables SMBus specific inputs
0 = Disables SMBus specific inputs

bit 5 **D/$\overline{\text{A}}$:** Data/$\overline{\text{Address}}$ bit

In Master mode:
Reserved.

In Slave mode:
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit[1]

1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last

bit 3 **S:** Start bit[1]

1 = Indicates that a Start bit has been detected last
0 = Start bit was not detected last

bit 2 **R/$\overline{\text{W}}$:** Read/$\overline{\text{Write}}$ Information bit[2,3]

In Slave mode:
1 = Read
0 = Write

In Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress

bit 1 **UA:** Update Address bit (10-Bit Slave mode only)

1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit

In Transmit mode:
1 = SSPBUF is full
0 = SSPBUF is empty

In Receive mode:
1 = SSPBUF is full (does not include the $\overline{\text{ACK}}$ and Stop bits)
0 = SSPBUF is empty (does not include the $\overline{\text{ACK}}$ and Stop bits)

Note 1: This bit is cleared on Reset and when SSPEN is cleared.

2: This bit holds the R/$\overline{\text{W}}$ bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not $\overline{\text{ACK}}$ bit.

3: ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Active mode.

REGISTER 21-4: SSPCON1: MSSP CONTROL REGISTER 1 (I^{2}C™ MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN[(1)]	CKP	SSPM3[(2)]	SSPM2[(2)]	SSPM1[(2)]	SSPM0[(2)]
bit 7							bit 0

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

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:
1 = A write to the SSPBUF register was attempted while the I^{2}C conditions were not valid for a transmission to be started (must be cleared in software)
0 = No collision

In Slave Transmit mode:
1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

In Receive mode (Master or Slave modes):
This is a “don’t care” bit.

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:
1 = A byte is received while the SSPBUF register is still holding the previous byte (must be cleared in software)
0 = No overflow

In Transmit mode:
This is a “don’t care” bit in Transmit mode.

bit 5 **SSPEN:** Master Synchronous Serial Port Enable bit[(1)]

1 = Enables the serial port and configures the SDA and SCL pins as the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP:** SCK Release Control bit

In Slave mode:
1 = Releases clock
0 = Holds clock low (clock stretch), used to ensure data setup time

In Master mode:
Unused in this mode.

bit 3-0 **SSPM<3:0>:** Master Synchronous Serial Port Mode Select bits[(2)]

1111 = I^{2}C Slave mode, 10-bit address with Start and Stop bit interrupts enabled
1110 = I^{2}C Slave mode, 7-bit address with Start and Stop bit interrupts enabled
1011 = I^{2}C Firmware Controlled Master mode (slave Idle)
1001 = Load SSPMSK register at SSPADD SFR address[(3,4)]
1000 = I^{2}C Master mode, clock = FOSC/(4 * (SSPADD + 1))
0111 = I^{2}C Slave mode, 10-bit address
0110 = I^{2}C Slave mode, 7-bit address

Note 1: When enabled, the SDA and SCL pins must be configured as inputs.
2: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.
3: When SSPM<3:0> = 1001, any reads or writes to the SSPADD SFR address actually access the SSPMSK register.
4: This mode is only available when 7-Bit Address Masking mode is selected (MSSPMSK Configuration bit is ‘1’).

REGISTER 21-5: SSPCON2: MSSP CONTROL REGISTER 2 (I²C™ MASTER MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT(1)	ACKEN(2)	RCEN(2)	PEN(2)	RSEN(2)	SEN(2)
bit 7							bit 0

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

bit 7 **GCEN:** General Call Enable bit

Unused in Master mode.

bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)

1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)[1]

1 = Not Acknowledged
0 = Acknowledged

bit 4 **ACKEN:** Acknowledge Sequence Enable bit[2]

1 = Initiates Acknowledge sequence on SDA and SCL pins and transmits ACKDT data bit; automatically cleared by hardware
0 = Acknowledge sequence is Idle

bit 3 **RCEN:** Receive Enable bit (Master Receive mode only)[2]

1 = Enables Receive mode for I²C™
0 = Receive is Idle

bit 2 **PEN:** Stop Condition Enable bit[2]

1 = Initiates Stop condition on SDA and SCL pins; automatically cleared by hardware
0 = Stop condition is Idle

bit 1 **RSEN:** Repeated Start Condition Enable bit[2]

1 = Initiates Repeated Start condition on SDA and SCL pins; automatically cleared by hardware
0 = Repeated Start condition Idle

bit 0 **SEN:** Start Condition Enable bit[2]

1 = Initiates Start condition on SDA and SCL pins; automatically cleared by hardware
0 = Start condition Idle

Note 1: The value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

2: If the I²C module is active, these bits may not be set (no spooling) and the SSPBUF may not be written to (or writes to the SSPBUF are disabled).

REGISTER 22-2: RCSTAx: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
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 **SPEN:** Serial Port Enable bit

1 = Serial port is enabled (configures RXx/DTx and TXx/CKx pins as serial port pins)
0 = Serial port is disabled (held in Reset)

bit 6 **RX9:** 9-Bit Receive Enable bit

1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit

Asynchronous mode:
Don’t care.

Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode – Slave:
Don’t care.

bit 4 **CREN:** Continuous Receive Enable bit

Asynchronous mode:
1 = Enables receiver
0 = Disables receiver

Synchronous mode:
1 = Enables continuous receive until enable bit, CREN, is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit

Asynchronous mode 9-Bit (RX9 = 1):
1 = Enables address detection; enables interrupt and loads the receive buffer when RSR<8> is set
0 = Disables address detection; all bytes are received and the ninth bit can be used as a parity bit

Asynchronous mode 9-Bit (RX9 = 0):
Don’t care.

bit 2 **FERR:** Framing Error bit

1 = Framing error (can be cleared by reading the RCREGx register and receiving next valid byte)
0 = No framing error

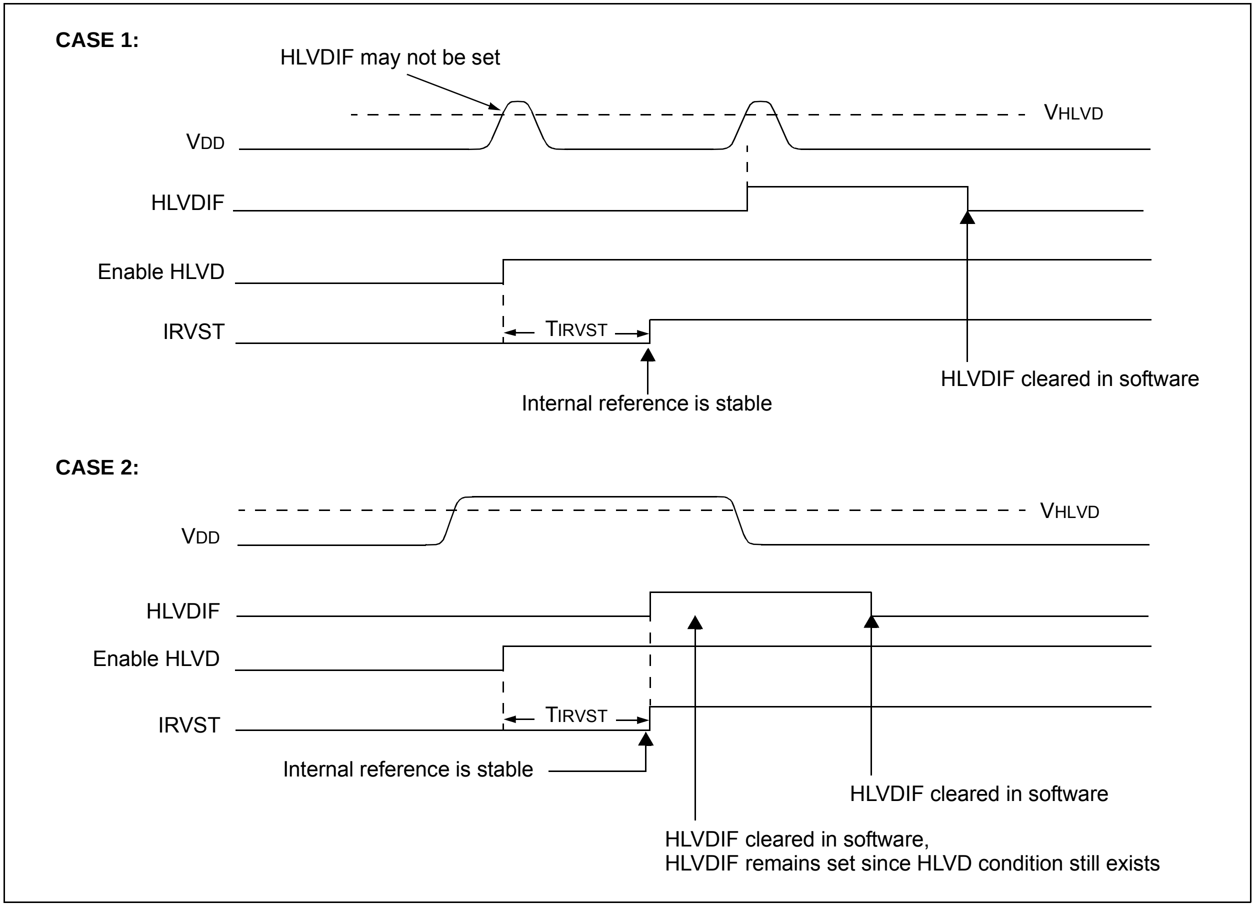
bit 1 **OERR:** Overrun Error bit

1 = Overrun error (can be cleared by clearing bit, CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data

This can be address/data bit or a parity bit and must be calculated by user firmware.

FIGURE 26-3: HIGH-VOLTAGE DETECT OPERATION (VDIRMAG = 1)

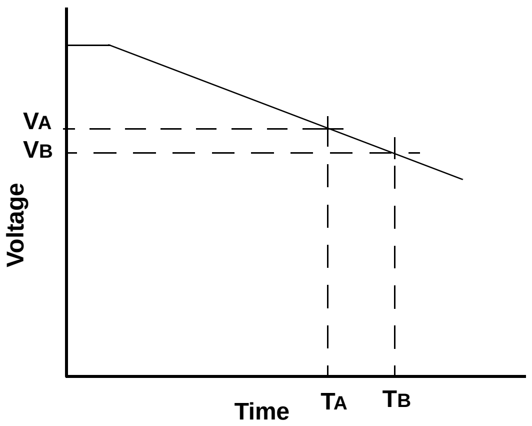

CASE 1:
HLVDIF may not be set
VHLVD
VDD
HLVDIF
Enable HLVD
IRVST
TIRVST
Internal reference is stable
HLVDIF cleared in software
CASE 2:
VHLVD
VDD
HLVDIF
Enable HLVD
IRVST
TIRVST
Internal reference is stable
HLVDIF cleared in software
HLVDIF cleared in software,
HLVDIF remains set since HLVD condition still exists

26.5 Applications

In many applications, it is desirable to detect a drop below, or rise above, a particular voltage threshold. For example, the HLVD module could be periodically enabled to detect Universal Serial Bus (USB) attach or detach. This assumes the device is powered by a lower voltage source than the USB when detached. An attach would indicate a high-voltage detect from, for example, 3.3V to 5V (the voltage on USB) and vice versa for a detach. This feature could save a design a few extra components and an attach signal (input pin).

For general battery applications, Figure 26-4 shows a possible voltage curve. Over time, the device voltage decreases. When the device voltage reaches voltage, VA, the HLVD logic generates an interrupt at time, TA. The interrupt could cause the execution of an ISR, which would allow the application to perform "house-keeping tasks" and a controlled shutdown before the device voltage exits the valid operating range at TB. This would give the application a time window, represented by the difference between TA and TB, to safely exit.

FIGURE 26-4: TYPICAL LOW-VOLTAGE DETECT APPLICATION



Legend: VA = HLVD trip point
VB = Minimum valid device operating voltage

EXAMPLE 27-2: WIN AND ICODE BITS USAGE IN INTERRUPT SERVICE ROUTINE TO ACCESS TX/RX BUFFERS (CONTINUED)

```
ErrorInterrupt
    BCF    PIR3, ERRIF                    ; Clear the interrupt flag
    …                                     ; Handle error.
    RETFIE
TXB2Interrupt
    BCF    PIR3, TXB2IF                   ; Clear the interrupt flag
    GOTO   AccessBuffer
TXB1Interrupt
    BCF    PIR3, TXB1IF                   ; Clear the interrupt flag
    GOTO   AccessBuffer
TXB0Interrupt
    BCF    PIR3, TXB0IF                   ; Clear the interrupt flag
    GOTO   AccessBuffer
RXB1Interrupt
    BCF    PIR3, RXB1IF                   ; Clear the interrupt flag
    GOTO   Accessbuffer
RXB0Interrupt
    BCF    PIR3, RXB0IF                   ; Clear the interrupt flag
    GOTO   AccessBuffer
AccessBuffer                              ; This is either TX or RX interrupt
    ; Copy CANSTAT.ICODE bits to CANCON.WIN bits
    MOVF   TempCANCON, W                  ; Clear CANCON.WIN bits before copying
                                          ; new ones.
    ANDLW  B’11110001’                    ; Use previously saved CANCON value to
                                          ; make sure same value.
    MOVWF  TempCANCON                     ; Copy masked value back to TempCANCON
    MOVF   TempCANSTAT, W                 ; Retrieve ICODE bits
    ANDLW  B’00001110’                    ; Use previously saved CANSTAT value
                                          ; to make sure same value.
    IORWF  TempCANCON                     ; Copy ICODE bits to WIN bits.
    MOVFF  TempCANCON, CANCON             ; Copy the result to actual CANCON
    ; Access current buffer…
    ; User code
    ; Restore CANCON.WIN bits
    MOVF   CANCON, W                      ; Preserve current non WIN bits
    ANDLW  B’11110001’
    IORWF  TempCANCON                     ; Restore original WIN bits
    ; Do not need to restore CANSTAT - it is read-only register.
    ; Return from interrupt or check for another module interrupt source
```

REGISTER 27-49: MSEL1: MASK SELECT REGISTER 1[1]

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1
FIL7_1	FIL7_0	FIL6_1	FIL6_0	FIL5_1	FIL5_0	FIL4_1	FIL4_0
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 **FIL7_<1:0>:** Filter 7 Select bits 1 and 0

11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 5-4 **FIL6_<1:0>:** Filter 6 Select bits 1 and 0

11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 3-2 **FIL5_<1:0>:** Filter 5 Select bits 1 and 0

11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 1-0 **FIL4_<1:0>:** Filter 4 Select bits 1 and 0

11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

Note 1: This register is available in Mode 1 and 2 only.

REGISTER 27-57: PIE5: PERIPHERAL INTERRUPT ENABLE REGISTER 5

Mode 0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IRXIE	WAKIE	ERRIE	TXB2IE	TXB1IE[(1)]	TXB0IE[(1)]	RXB1IE	RXB0IE

Mode 1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IRXIE	WAKIE	ERRIE	TXBnIE	TXB1IE[(1)]	TXB0IE[(1)]	RXBnIE	FIFOWMIE
	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 **IRXIE:** CAN Bus Error Message Received Interrupt Enable bit
1 = Enable invalid message received interrupt
0 = Disable invalid message received interrupt

bit 6 **WAKIE:** CAN bus Activity Wake-up Interrupt Enable bit
1 = Enable bus activity wake-up interrupt
0 = Disable bus activity wake-up interrupt

bit 5 **ERRIE:** CAN bus Error Interrupt Enable bit
1 = Enable CAN module error interrupt
0 = Disable CAN module error interrupt

bit 4 When CAN is in Mode 0:
TXB2IE: CAN Transmit Buffer 2 Interrupt Enable bit
1 = Enable Transmit Buffer 2 interrupt
0 = Disable Transmit Buffer 2 interrupt

When CAN is in Mode 1 or 2:
TXBnIE: CAN Transmit Buffer Interrupts Enable bit
1 = Enable transmit buffer interrupt; individual interrupt is enabled by TXBIE and BIE0
0 = Disable all transmit buffer interrupts

bit 3 **TXB1IE:** CAN Transmit Buffer 1 Interrupt Enable bit[(1)]
1 = Enable Transmit Buffer 1 interrupt
0 = Disable Transmit Buffer 1 interrupt

bit 2 **TXB0IE:** CAN Transmit Buffer 0 Interrupt Enable bit[(1)]
1 = Enable Transmit Buffer 0 interrupt
0 = Disable Transmit Buffer 0 interrupt

bit 1 When CAN is in Mode 0:
RXB1IE: CAN Receive Buffer 1 Interrupt Enable bit
1 = Enable Receive Buffer 1 interrupt
0 = Disable Receive Buffer 1 interrupt

When CAN is in Mode 1 or 2:
RXBnIE: CAN Receive Buffer Interrupts Enable bit
1 = Enable receive buffer interrupt; individual interrupt is enabled by BIE0
0 = Disable all receive buffer interrupts

bit 0 When CAN is in Mode 0:
RXB0IE: CAN Receive Buffer 0 Interrupt Enable bit
1 = Enable Receive Buffer 0 interrupt
0 = Disable Receive Buffer 0 interrupt

When CAN is in Mode 1:
Unimplemented: Read as ‘0’

When CAN is in Mode 2:
FIFOWMIE: FIFO Watermark Interrupt Enable bit
1 = Enable FIFO watermark interrupt
0 = Disable FIFO watermark interrupt

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

BNC Branch if Not Carry

Syntax:	BNC n
Operands:	-128 ≤ n ≤ 127
Operation:	if Carry bit is ‘0’, (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	`1110` `0011` `nnnn` `nnnn`
Description:	If the Carry bit is ‘0’, 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      BNC   Jump

   Before Instruction
      PC          =    address (HERE)
   After Instruction
      If Carry    =    0;
             PC   =    address (Jump)
      If Carry    =    1;
             PC   =    address (HERE + 2)
```

BNN Branch if Not Negative

Syntax:	BNN n
Operands:	-128 ≤ n ≤ 127
Operation:	if Negative bit is ‘0’, (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	`1110` `0111` `nnnn` `nnnn`
Description:	If the Negative bit is ‘0’, 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      BNN   Jump

   Before Instruction
      PC            =    address (HERE)
   After Instruction
      If Negative   =    0;
             PC     =    address (Jump)
      If Negative   =    1;
             PC     =    address (HERE + 2)
```

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`

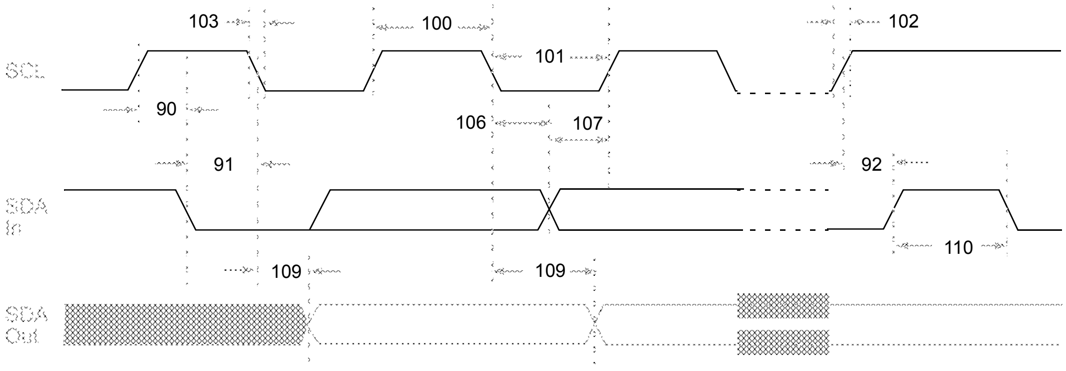
Before Instruction

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

After Instruction

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

FIGURE 31-16: I²C™ BUS DATA TIMING



Note: Refer to Figure 31-3 for load conditions.

TABLE 31-20: I²C™ BUS DATA REQUIREMENTS (SLAVE MODE)

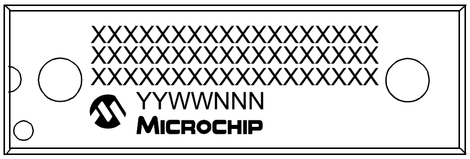
Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock High Time	100 kHz mode	4.0	—	μs	
			400 kHz mode	0.6	—	μs	
			MSSP module	1.5 TCY	—		
101	TLOW	Clock Low Time	100 kHz mode	4.7	—	μs	
			400 kHz mode	1.3	—	μs	
			MSSP module	1.5 TCY	—		
102	TR	SDA and SCL Rise Time	100 kHz mode	—	1000	ns	
			400 kHz mode	20 + 0.1 CB	300	ns	CB is specified to be from 10 to 400 pF
103	TF	SDA and SCL Fall Time	100 kHz mode	—	300	ns	
			400 kHz mode	20 + 0.1 CB	300	ns	CB is specified to be from 10 to 400 pF
90	TSU:STA	Start Condition Setup Time	100 kHz mode	4.7	—	μs	Only relevant for Repeated Start condition
			400 kHz mode	0.6	—	μs	
91	THD:STA	Start Condition Hold Time	100 kHz mode	4.0	—	μs	After this period, the first clock pulse is generated
			400 kHz mode	0.6	—	μs	
106	THD:DAT	Data Input Hold Time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	μs	
107	TSU:DAT	Data Input Setup Time	100 kHz mode	250	—	ns	**(Note 2)**
			400 kHz mode	100	—	ns	
92	TSU:STO	Stop Condition Setup Time	100 kHz mode	4.7	—	μs	
			400 kHz mode	0.6	—	μs	
109	TAA	Output Valid from Clock	100 kHz mode	—	3500	ns	**(Note 1)**
			400 kHz mode	—	—	ns	
110	TBUF	Bus Free Time	100 kHz mode	4.7	—	μs	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3	—	μs	
D102	CB	Bus Capacitive Loading		—	400	pF	

Note 1: As a transmitter, the device must provide this internal minimum delay time to bridge the undefined region (min. 300 ns) of the falling edge of SCL to avoid unintended generation of Start or Stop conditions.

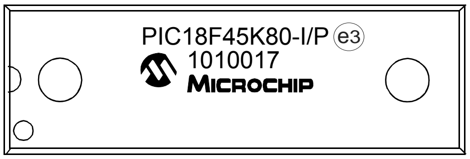
2: A Fast mode I²C™ bus device can be used in a Standard mode I²C bus system, but the requirement, TSU:DAT ≥ 250 ns, must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line, TR max. + TSU:DAT = 1000 + 250 = 1250 ns (according to the Standard mode I²C bus specification), before the SCL line is released.

32.1 Package Marking Information (Continued)

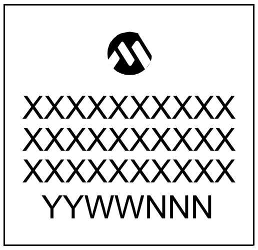
40-Lead PDIP



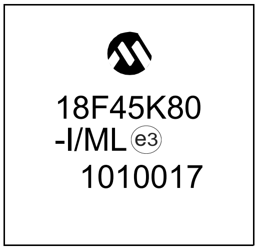
Example



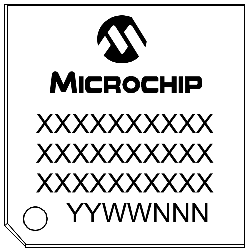
44-Lead QFN



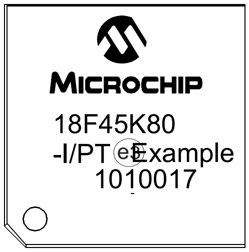
Example



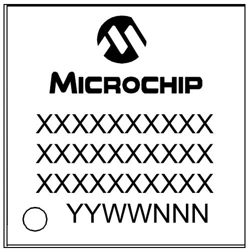
44-Lead TQFP

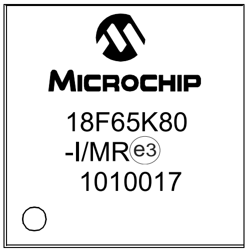


Example

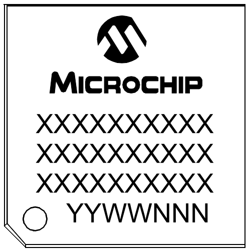


64-Lead QFN

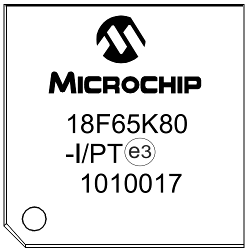




64-Lead TQFP

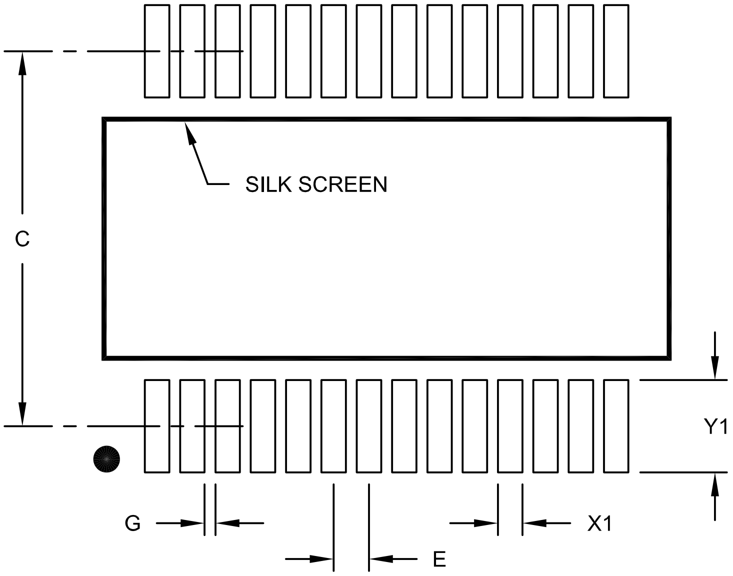


Example



28-Lead Plastic Shrink Small Outline (SS) - 5.30 mm Body [SSOP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



RECOMMENDED LAND PATTERN

Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Contact Pitch	E	0.65 BSC		
Contact Pad Spacing	C		7.20	
Contact Pad Width (X28)	X1			0.45
Contact Pad Length (X28)	Y1			1.75
Distance Between Pads	G	0.20		

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing No. C04-2073A