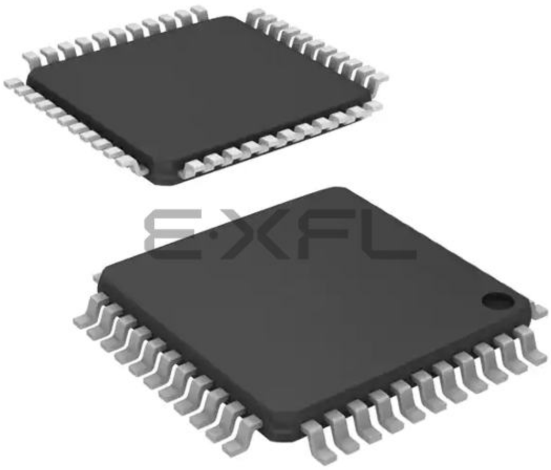


Welcome to E-XFL.COM

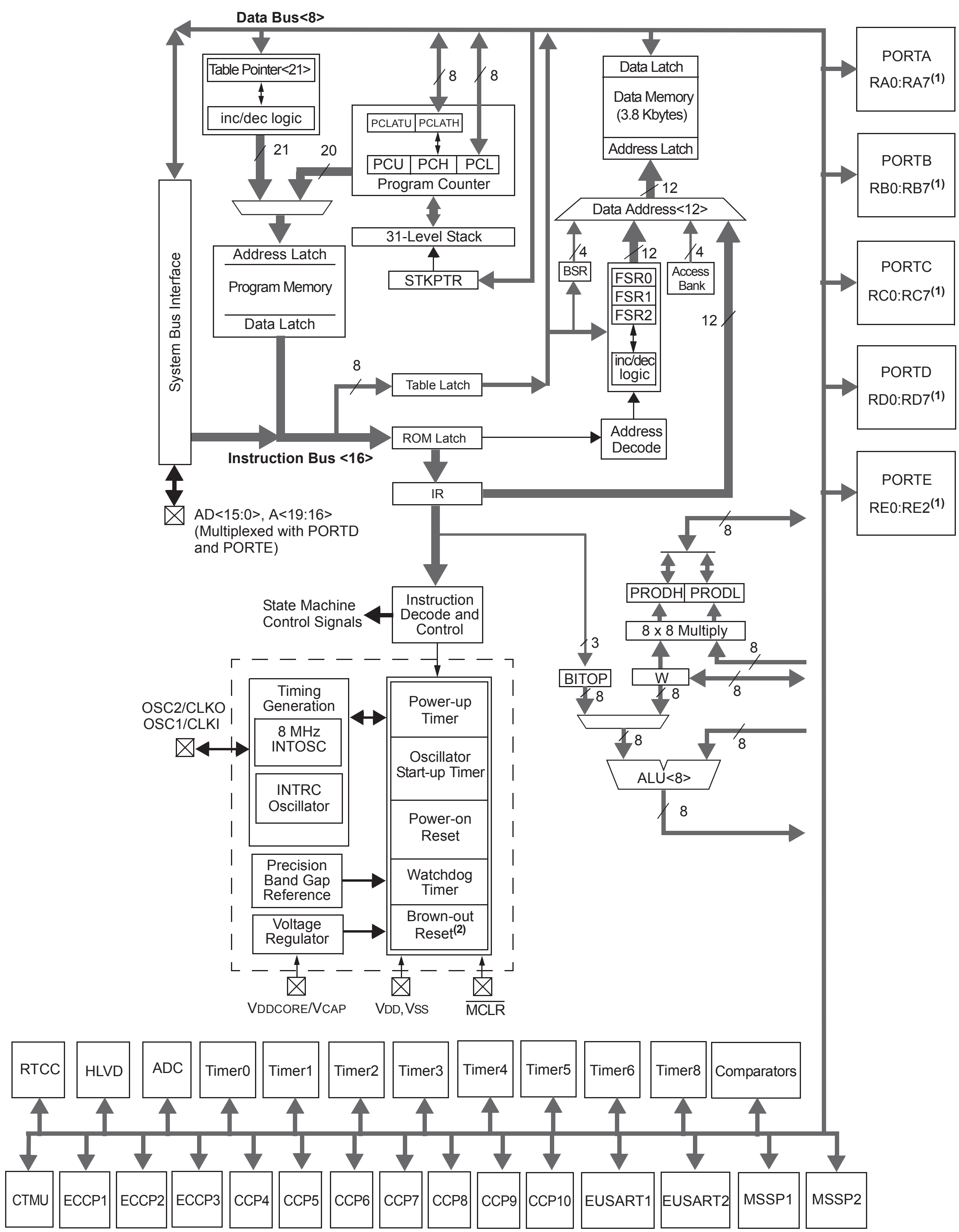
What is "Embedded - Microcontrollers"?

"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	48MHz
Connectivity	I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	34
Program Memory Size	128KB (64K x 16)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	3.8K x 8
Voltage - Supply (Vcc/Vdd)	2.15V ~ 3.6V
Data Converters	A/D 13x10b/12b
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/pic18f47j13-i-pt

FIGURE 1-2: PIC18F4XJ13 (44-PIN) BLOCK DIAGRAM



Note 1: See Table 1-3 for I/O port pin descriptions.

2: The on-chip voltage regulator is always enabled by default.

3.2.5.1 OSCTUNE Register

The internal oscillator's output has been calibrated at the factory but can be adjusted in the user's application. This is done by writing to the OSCTUNE register (Register 3-1).

When the OSCTUNE register is modified, the INTOSC frequency will begin shifting to the new frequency. The INTOSC clock will typically stabilize within 1 μs. Code execution continues during this shift. There is no indication that the shift has occurred.

The OSCTUNE register also contains the INTSRC bit. The INTSRC bit allows users to select which internal oscillator provides the clock source when the 31 kHz frequency option is selected. This is covered in larger detail in **Section 3.3.1 "Oscillator Control Register"**.

The PLLEN bit, contained in the OSCTUNE register, can be used to enable or disable the internal PLL when running in one of the PLL type oscillator modes (e.g., INTOSCPLL). Oscillator modes that do not contain "PLL" in their name cannot be used with the PLL. In these modes, the PLL is always disabled regardless of the setting of the PLLEN bit.

When configured for one of the PLL enabled modes, setting the PLLEN bit does not immediately switch the device clock to the PLL output. The PLL requires up to electrical parameter, t_{rc}, to start-up and lock, during which time, the device continues to be clocked. Once the PLL output is ready, the microcontroller core will automatically switch to the PLL derived frequency.

3.2.5.2 Internal Oscillator Output Frequency and Drift

The internal oscillator block is calibrated at the factory to produce an INTOSC output frequency of 8.0 MHz. However, this frequency may drift as VDD or temperature changes, which can affect the controller operation in a variety of ways.

The low-frequency INTRC oscillator operates independently of the INTOSC source. Any changes in INTOSC across voltage and temperature are not necessarily reflected by changes in INTRC and vice versa.

3.2.5.3 Compensating for INTOSC Drift

It is possible to adjust the INTOSC frequency by modifying the value in the OSCTUNE register. This has no effect on the INTRC clock source frequency.

Tuning the INTOSC source requires knowing when to make the adjustment, in which direction it should be made, and in some cases, how large a change is needed. When using the EUSART, for example, an adjustment may be required when it begins to generate framing errors or receives data with errors while in Asynchronous mode. Framing errors indicate that the device clock frequency is too high; to adjust for this, decrement the value in OSCTUNE to reduce the clock frequency. On the other hand, errors in data may suggest that the clock speed is too low; to compensate, increment OSCTUNE to increase the clock frequency.

It is also possible to verify device clock speed against a reference clock. Two timers may be used: one timer is clocked by the peripheral clock, while the other is clocked by a fixed reference source, such as the Timer1 oscillator. Both timers are cleared, but the timer clocked by the reference generates interrupts. When an interrupt occurs, the internally clocked timer is read and both timers are cleared. If the internally clocked timer value is greater than expected, then the internal oscillator block is running too fast. To adjust for this, decrement the OSCTUNE register.

Finally, an ECCP module can use free-running Timer1 (or Timer3), clocked by the internal oscillator block and an external event with a known period (i.e., AC power frequency). The time of the first event is captured in the CCPRxH:CCPRxL registers and is recorded for use later. When the second event causes a capture, the time of the first event is subtracted from the time of the second event. Since the period of the external event is known, the time difference between events can be calculated.

If the measured time is greater than the calculated time, the internal oscillator block is running too fast; to compensate, decrement the OSCTUNE register. If the measured time is less than the calculated time, the internal oscillator block is running too slow; to compensate, increment the OSCTUNE register.

6.4 Data Addressing Modes

> **Note:** The execution of some instructions in the core PIC18 instruction set are changed when the PIC18 extended instruction set is enabled. See **Section 6.6 "Data Memory and the Extended Instruction Set"** for more information.

While the program memory can be addressed in only one way through the PC, information in the data memory space can be addressed in several ways. For most instructions, the addressing mode is fixed. Other instructions may use up to three modes, depending on which operands are used and whether or not the extended instruction set is enabled.

The addressing modes are:

- Inherent
- Literal
- Direct
- Indirect

An additional addressing mode, Indexed Literal Offset, is available when the extended instruction set is enabled (XINST Configuration bit = 1). Its operation is discussed in more detail in **Section 6.6.1 "Indexed Addressing with Literal Offset"**.

6.4.1 INHERENT AND LITERAL ADDRESSING

Many PIC18 control instructions do not need any argument at all; they either perform an operation that globally affects the device, or they operate implicitly on one register. This addressing mode is known as Inherent Addressing. Examples include SLEEP, RESET and DAW.

Other instructions work in a similar way, but require an additional explicit argument in the opcode. This is known as Literal Addressing mode, because they require some literal value as an argument. Examples include ADDLW and MOVLW, which respectively, add or move a literal value to the W register. Other examples include CALL and GOTO, which include a 20-bit program memory address.

6.4.2 DIRECT ADDRESSING

Direct Addressing specifies all or part of the source and/or destination address of the operation within the opcode itself. The options are specified by the arguments accompanying the instruction.

In the core PIC18 instruction set, bit-oriented and byte-oriented instructions use some version of Direct Addressing by default. All of these instructions include some 8-bit literal address as their LSB. This address specifies either a register address in one of the banks of data RAM (**Section 6.3.3 "General Purpose Register File"**) or a location in the Access Bank (**Section 6.3.2 "Access Bank"**) as the data source for the instruction.

The Access RAM bit, 'a', determines how the address is interpreted. When 'a' is '1', the contents of the BSR (**Section 6.3.1 "Bank Select Register"**) are used with the address to determine the complete 12-bit address of the register. When 'a' is '0', the address is interpreted as being a register in the Access Bank. Addressing that uses the Access RAM is sometimes also known as Direct Forced Addressing mode.

A few instructions, such as MOVFF, include the entire 12-bit address (either source or destination) in their opcodes. In these cases, the BSR is ignored entirely.

The destination of the operation's results is determined by the destination bit, 'd'. When 'd' is '1', the results are stored back in the source register, overwriting its original contents. When 'd' is '0', the results are stored in the W register. Instructions without the 'd' argument have a destination that is implicit in the instruction; their destination is either the target register being operated on or the W register.

6.4.3 INDIRECT ADDRESSING

Indirect Addressing allows the user to access a location in data memory without giving a fixed address in the instruction. This is done by using File Select Registers (FSRs) as pointers to the locations to be read or written to. Since the FSRs are themselves located in RAM as SFRs, they can also be directly manipulated under program control. This makes FSRs very useful in implementing data structures such as tables and arrays in data memory.

The registers for Indirect Addressing are also implemented with Indirect File Operands (INDFs) that permit automatic manipulation of the pointer value with auto-incrementing, auto-decrementing or offsetting with another value. This allows for efficient code using loops, such as the example of clearing an entire RAM bank in Example 6-5. It also enables users to perform Indexed Addressing and other Stack Pointer operations for program memory in data memory.

EXAMPLE 6-5: HOW TO CLEAR RAM (BANK 1) USING INDIRECT ADDRESSING

```
        LFSR    FSR0, 0x100 ;
NEXT    CLRF    POSTINC0    ; Clear INDF
                            ; register then
                            ; inc pointer
        BTFSS   FSR0H, 1    ; All done with
                            ; Bank1?
        BRA     NEXT        ; NO, clear next
CONTINUE                    ; YES, continue
```

7.5 Writing to Flash Program Memory

The programming block is 32 words or 64 bytes. Programming one word or 2 bytes at a time is also supported.

Table writes are used internally to load the holding registers needed to program the Flash memory. There are 64 holding registers used by the table writes for programming.

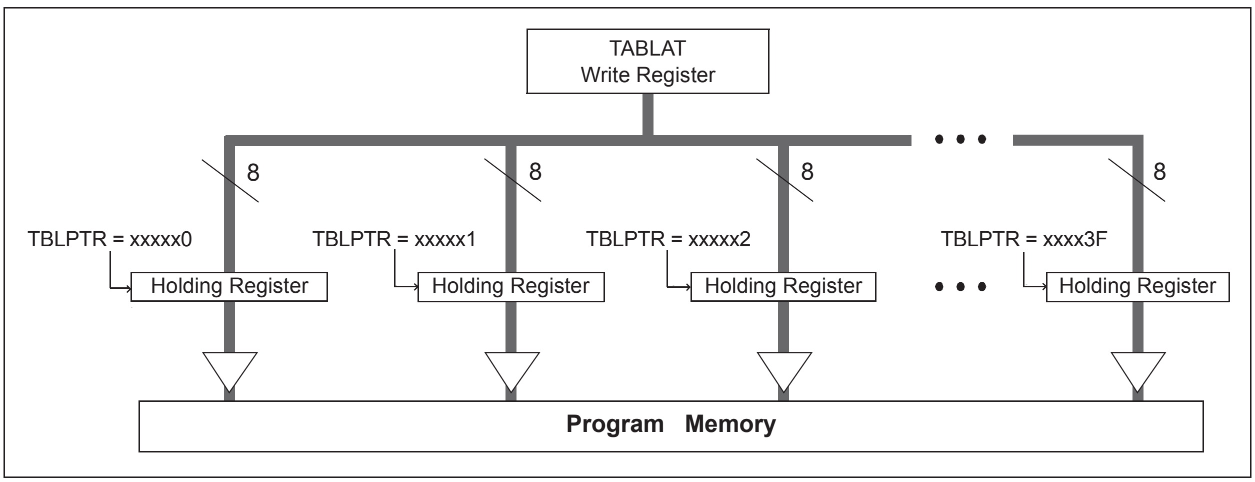
Since the Table Latch (TABLAT) is only a single byte, the `TBLWT` instruction may need to be executed 64 times for each programming operation (if WPROG = `0`). All of the table write operations will essentially be short writes because only the holding registers are written. At the end of updating the 64 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 will be terminated by the internal programming timer.

The 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 1:	Unlike previous PIC® devices, devices of the PIC18F47J13 Family do not reset the holding registers after a write occurs. The holding registers must be cleared or overwritten before a programming sequence.
2:	To maintain the endurance of the program memory cells, each Flash byte should not be programmed more than once between erase operations. Before attempting to modify the contents of the target cell a second time, an erase of the target page, or a bulk erase of the entire memory, must be performed.

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 1024 bytes into RAM.
2. Update data values in RAM as necessary.
3. Load the Table Pointer register with address being erased.
4. Execute the erase procedure.
5. Load the Table Pointer register with the address of the first byte being written, minus 1.
6. Write the 64 bytes into the holding registers with auto-increment.
7. Set the WREN bit (EECON1<2>) to enable byte writes.
8. Disable interrupts.
9. Write 55h to EECON2.
10. Write 0AAh to EECON2.
11. Set the WR bit; this will begin the write cycle.
12. The CPU will stall for the duration of the write for T_{IW} (see parameter D133A).
13. Re-enable interrupts.
14. Repeat Steps 6 through 13 until all 1024 bytes are written to program memory.
15. Verify the memory (table read).

An example of the required code is provided 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 11-24: WRITE TIMING, 16-BIT MULTIPLEXED DATA, PARTIALLY MULTIPLEXED ADDRESS

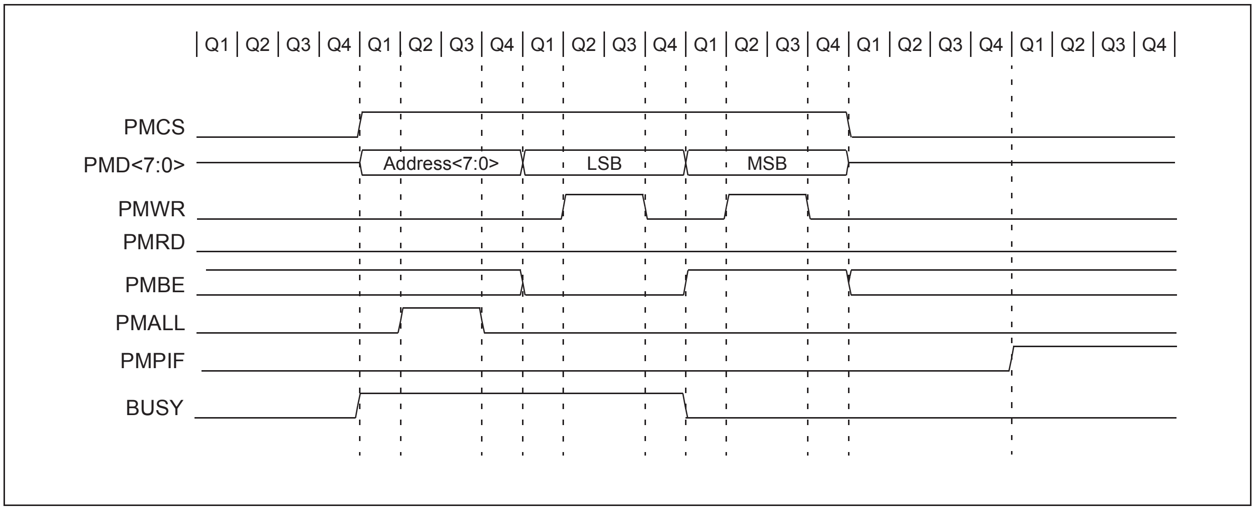


FIGURE 11-25: READ TIMING, 16-BIT MULTIPLEXED DATA, FULLY MULTIPLEXED 16-BIT ADDRESS

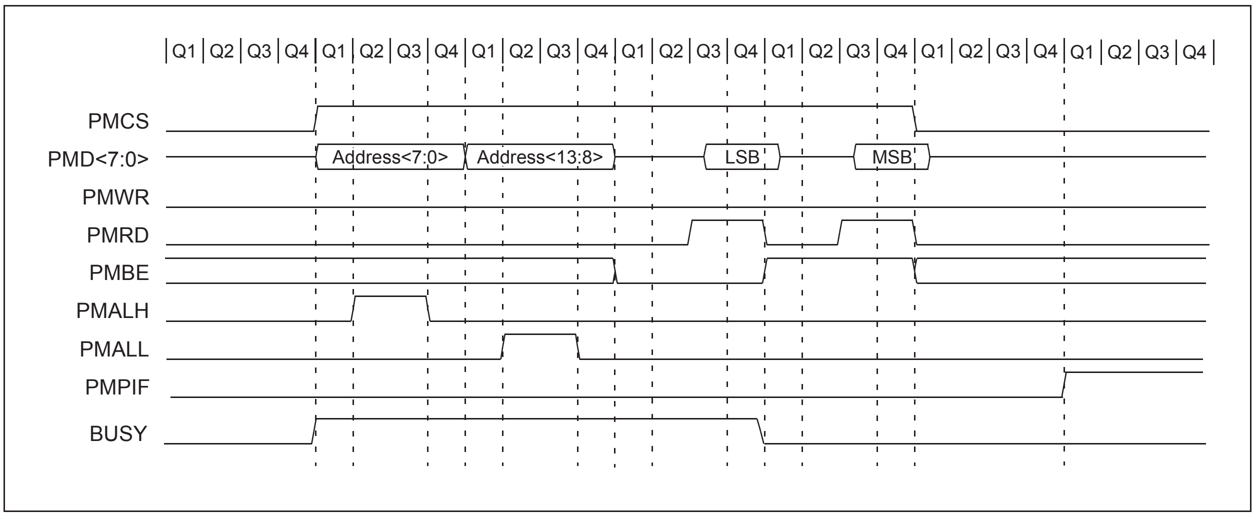
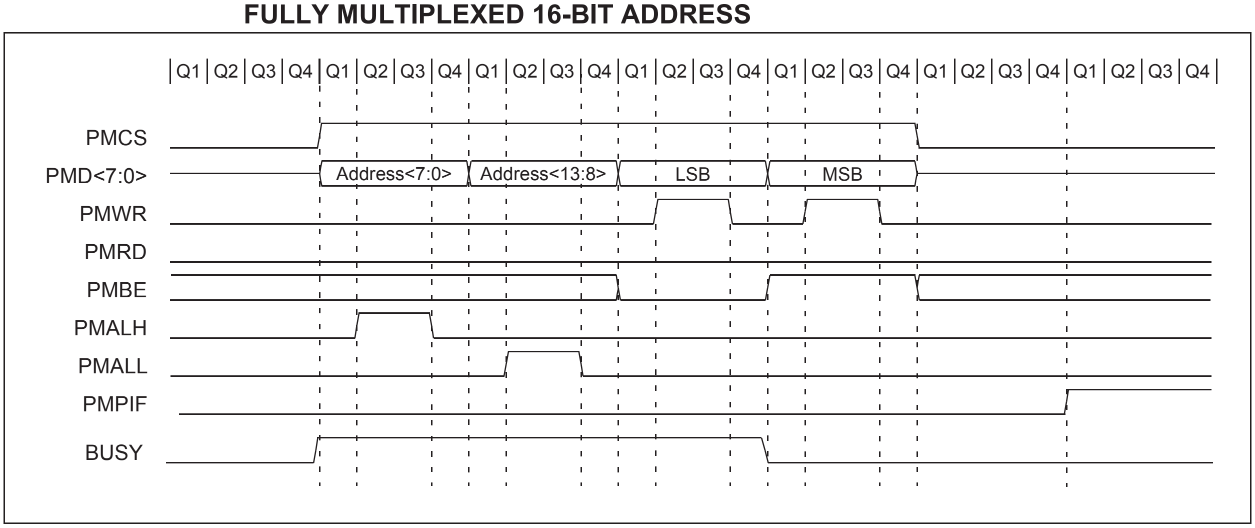


FIGURE 11-26: WRITE TIMING, 16-BIT MULTIPLEXED DATA, FULLY MULTIPLEXED 16-BIT ADDRESS



13.8.4 TIMER1 GATE SINGLE PULSE MODE

When Timer1 Gate Single Pulse mode is enabled, it is possible to capture a single pulse gate event. Timer1 Gate Single Pulse mode is first enabled by setting the T1GSPM bit in the T1GCON register. Next, the T1GGO/T1DONE bit in the T1GCON register must be set. The Timer1 will be fully enabled on the next incrementing edge. On the next trailing edge of the pulse, the T1GGO/T1DONE bit will automatically be cleared. No other gate events will be allowed to increment Timer1 until the T1GGO/T1DONE bit is once again set in software.

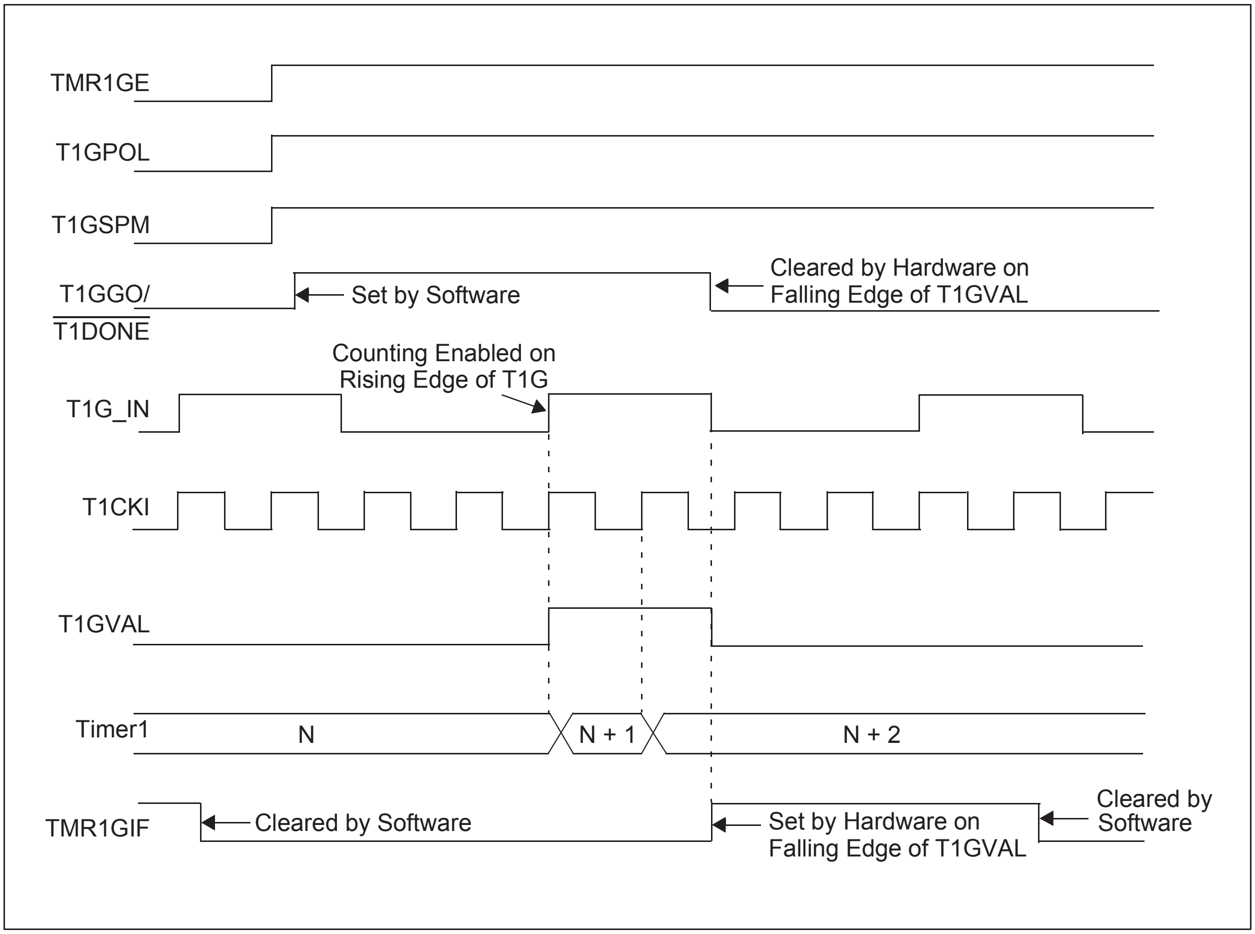
Clearing the T1GSPM bit of the T1GCON register will also clear the T1GGO/T1DONE bit. See Figure 13-6 for timing details.

Enabling the Toggle mode and the Single Pulse mode, simultaneously, will permit both sections to work together. This allows the cycle times on the Timer1 gate source to be measured. See Figure 13-7 for timing details.

13.8.5 TIMER1 GATE VALUE STATUS

When the Timer1 gate value status is utilized, it is possible to read the most current level of the gate control value. The value is stored in the T1GVAL bit in the T1GCON register. The T1GVAL bit is valid even when the Timer1 gate is not enabled (TMR1GE bit is cleared).

FIGURE 13-6: TIMER1 GATE SINGLE PULSE MODE



REGISTER 15-3: OSCCON2: OSCILLATOR CONTROL REGISTER 2 (ACCESS F87h)

U-0	R-0(2)	U-0	R/W-1	R/W-0(2)	R/W-1	U-0	U-0
—	SOSCRUN	—	SOSCDRV	SOSCGO(3)	PRISD	—	—
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 **Unimplemented:** Read as ‘0’

bit 6 **SOSCRUN:** SOSC Run Status bit
1 = System clock comes from secondary SOSC
0 = System clock comes from an oscillator other than SOSC

bit 5 **Unimplemented:** Read as ‘0’

bit 4 **SOSCDRV:** SOSC Drive Control bit
1 = T1OSC/SOSC oscillator drive circuit is selected by Configuration bits, CONFIG2L<4:3>
0 = Low-power T1OSC/SOSC circuit is selected

bit 3 **SOSCGO:** Oscillator Start Control bit
1 = Turns on the oscillator, even if no peripherals are requesting it
0 = Oscillator is shut off unless peripherals are requesting it

bit 2 **PRISD:** Primary Oscillator Drive Circuit Shutdown bit
1 = Oscillator drive circuit is on
0 = Oscillator drive circuit is off (zero power)

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

Note 1: Reset value is ‘0’ when Two-Speed Start-up is enabled and ‘1’ if disabled.
2: Default output frequency of INTOSC on Reset (4 MHz).
3: When the SOSC is selected to run from a digital clock input, rather than an external crystal, this bit has no effect.

20.3.1 REGISTERS

Each MSSP module has four registers for SPI mode operation. These are:

- MSSPx Control Register 1 (SSPxCON1)
- MSSPx Status Register (SSPxSTAT)
- Serial Receive/Transmit Buffer Register (SSPxBUF)
- MSSPx Shift Register (SSPxSR) – Not directly accessible

SSPxCON1 and SSPxSTAT are the control and status registers in SPI mode operation. The SSPxCON1 register is readable and writable. The lower six bits of the SSPxSTAT are read-only. The upper two bits of the SSPxSTAT are read/write.

SSPxSR is the shift register used for shifting data in or out. SSPxBUF is the buffer register to which data bytes are written to or read from.

In receive operations, SSPxSR and SSPxBUF together, create a double-buffered receiver. When SSPxSR receives a complete byte, it is transferred to SSPxBUF and the SSPxIF interrupt is set.

During transmission, the SSPxBUF is not double-buffered. A write to SSPxBUF will write to both SSPxBUF and SSPxSR.

Note: Because the SSPxBUF register is double-buffered, using read-modify-write instructions such as `BCF`, `COMF`, etc., will not work.

Similarly, when debugging under an in-circuit debugger, performing actions that cause reads of SSPxBUF (mouse hovering, watch, etc.) can consume data that the application code was expecting to receive.

REGISTER 20-1: SSPxSTAT: MSSPx STATUS REGISTER (SPI MODE) (ACCESS 1, FC7h; 2, F73h)

R/W-1	R/W-1	R-1	R-1	R-1	R-1	R-1	R-1
SMP	CKE(1)	D/$\overline{A}$	P	S	R/$\overline{W}$	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:** Sample bit
SPI Master mode:
1 = Input data sampled at the end of data output time
0 = Input data sampled at the middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in Slave mode.

bit 6 **CKE:** SPI Clock Select bit(1)
1 = Transmit occurs on transition from active to Idle clock state
0 = Transmit occurs on transition from Idle to active clock state

bit 5 **D/$\overline{A}$:** Data/$\overline{\text{Address}}$ bit
Used in I^2C mode only.

bit 4 **P:** Stop bit
Used in I^2C mode only; this bit is cleared when the MSSP module is disabled, SSPEN is cleared.

bit 3 **S:** Start bit
Used in I^2C mode only.

bit 2 **R/$\overline{W}$:** Read/$\overline{\text{Write}}$ Information bit
Used in I^2C mode only.

bit 1 **UA:** Update Address bit
Used in I^2C mode only.

bit 0 **BF:** Buffer Full Status bit
1 = Receive complete, SSPxBUF is full
0 = Receive not complete, SSPxBUF is empty

Note 1: Polarity of the clock state is set by the CKP bit (SSPxCON1<4>).

20.3.2 OPERATION

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits (SSPxCON1<5:0> and SSPxSTAT<7:6>). These control bits allow the following to be specified:

- Master mode (SCKx is the clock output)
- Slave mode (SCKx is the clock input)
- Clock Polarity (Idle state of SCKx)
- Data Input Sample Phase (middle or end of data output time)
- Clock Edge (output data on rising/falling edge of SCKx)
- Clock Rate (Master mode only)
- Slave Select mode (Slave mode only)

Each MSSP module consists of a Transmit/Receive Shift register (SSPxSR) and a Buffer register (SSPxBUF). The SSPxSR shifts the data in and out of the device, MSb first. The SSPxBUF holds the data that was written to the SSPxSR until the received data is ready. Once the 8 bits of data have been received, that byte is moved to the SSPxBUF register. Then, the Buffer Full (BF) detect bit (SSPxSTAT<0>) and the interrupt flag bit, SSPxIF, are set. This double-buffering of the received data (SSPxBUF) allows the next byte to start reception before reading the data that was just received.

Any write to the SSPxBUF register during transmission or reception of data will be ignored and the Write Collision Detect bit, WCOL (SSPxCON1<7>), will be set. User software must clear the WCOL bit so that it can be determined if the following write(s) to the SSPxBUF register completed successfully.

Note:	When the application software is expecting to receive valid data, the SSPxBUF should be read before the next byte of transfer data is written to the SSPxBUF. Application software should follow this process even when the current contents of SSPxBUF are not important.

The Buffer Full bit, BF (SSPxSTAT<0>), indicates when SSPxBUF has been loaded with the received data (transmission is complete). When the SSPxBUF is read, the BF bit is cleared. This data may be irrelevant if the SPI is only a transmitter. Generally, the MSSP interrupt is used to determine when the transmission/reception has completed. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur.

Example 20-1 provides the loading of the SSPxBUF (SSPxSR) for data transmission.

The SSPxSR is not directly readable or writable and can only be accessed by addressing the SSPxBUF register. Additionally, the SSPxSTAT register indicates the various status conditions.

20.3.3 OPEN-DRAIN OUTPUT OPTION

The drivers for the SDOx output and SCKx clock pins can be optionally configured as open-drain outputs. This feature allows the voltage level on the pin to be pulled to a higher level through an external pull-up resistor, provided the SDOx or SCKx pin is not multiplexed with an ANx analog function. This allows the output to communicate with external circuits without the need for additional level shifters. For more information, see **Section 10.1.4 "Open-Drain Outputs"**.

The open-drain output option is controlled by the SPI2OD and SPI1OD bits (ODCON3<1:0>). Setting an SPIxOD bit configures both the SDOx and SCKx pins for the corresponding open-drain operation.

EXAMPLE 20-1: LOADING THE SSP1BUF (SSP1SR) REGISTER

```
LOOP    BTFSS   SSP1STAT, BF    ;Has data been received (transmit complete)?
        BRA     LOOP            ;No
        MOVF    SSP1BUF, W      ;WREG reg = contents of SSP1BUF

        MOVWF   RXDATA          ;Save in user RAM, if data is meaningful

        MOVF    TXDATA, W       ;W reg = contents of TXDATA
        MOVWF   SSP1BUF         ;New data to xmit
```

TABLE 21-8: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER RECEPTION

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	PMPIF(1)	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
PIE1	PMPIE(1)	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
IPR1	PMPIP(1)	ADIP	RC1IP	TX1IP	SSP1IP	CCP1IP	TMR2IP	TMR1IP
PIR3	SSP2IF	BCL2IF	RC2IF	TX2IF	TMR4IF	CTMUIF	TMR3GIF	RTCCIF
PIE3	SSP2IE	BCL2IE	RC2IE	TX2IE	TMR4IE	CTMUIE	TMR3GIE	RTCCIE
IPR3	SSP2IP	BCL2IP	RC2IP	TX2IP	TMR4IP	CTMUIP	TMR3GIP	RTCCIP
RCSTAx	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
RCREGx	EUSARTx Receive Register							
TXSTAx	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
BAUDCONx	ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN
SPBRGHx	EUSARTx Baud Rate Generator High Byte							
SPBRGx	EUSARTx Baud Rate Generator Low Byte							
ODCON2	—	—	—	—	CCP10OD	CCP9OD	U2OD	U1OD

Legend: — = unimplemented, read as '0'. Shaded cells are not used for synchronous master reception.

Note 1: These pins are only available on 44-pin devices.

25.0 HIGH/LOW VOLTAGE DETECT (HLVD)

The High/Low-Voltage Detect (HLVD) module can be used to monitor the absolute voltage on VDD or the HLVDIN pin. This is a programmable circuit that allows the user to specify both a device voltage trip point and the direction of change from that point.

If the module detects an excursion past the trip point in that direction, an interrupt flag is set. If the interrupt is enabled, the program execution will branch to the interrupt vector address and the software can then respond to the interrupt.

The High/Low-Voltage Detect Control register (Register 25-1) completely controls the operation of the HLVD module. This allows the circuitry to be "turned off" by the user under software control, which minimizes the current consumption for the device.

Figure 25-1 provides a block diagram for the HLVD module.

REGISTER 25-1: HLVDCON: HIGH/LOW-VOLTAGE DETECT CONTROL REGISTER (ACCESS F85h)

R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
VDIRMAG	BGVST	IRVST	HLVDEN	HLVDL3(1)	HLVDL2(1)	HLVDL1(1)	HLVDL0(1)
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 **VDIRMAG:** Voltage Direction Magnitude Select bit
1 = Event occurs when the voltage equals or exceeds the trip point (HLVDL<3:0>)
0 = Event occurs when the voltage equals or falls below the trip point (HLVDL<3:0>)

bit 6 **BGVST:** Band Gap Reference Voltages Stable Status Flag bit
1 = Indicates internal band gap voltage references is stable
0 = Indicates internal band gap voltage reference is not stable

bit 5 **IRVST:** Internal Reference Voltage Stable Flag bit
1 = Indicates that the voltage detect logic will generate the interrupt flag at the specified voltage range
0 = Indicates that the voltage detect logic will not generate the interrupt flag at the specified voltage range and the HLVD interrupt should not be enabled

bit 4 **HLVDEN:** High/Low-Voltage Detect Power Enable bit
1 = HLVD is enabled
0 = HLVD is disabled

bit 3-0 **HLVDL<3:0>:** Voltage Detection Limit bits(1)
1111 = External analog input is used (input comes from the HLVDIN pin)
1110 = Maximum setting
.
.
.
0000 = Minimum setting

Note 1: See Table 30-8 in **Section 30.0 "Electrical Characteristics"** for specifications.

The module is enabled by setting the HLVDEN bit. Each time the module is enabled, the circuitry requires some time to stabilize. The IRVST bit is a read-only bit that indicates when the circuit is stable. The module can generate an interrupt only after the circuit is stable and IRVST is set.

The VDIRMAG bit determines the overall operation of the module. When VDIRMAG is cleared, the module monitors for drops in V_{DD} below a predetermined set point. When the bit is set, the module monitors for rises in V_{DD} above the set point.

25.1 Operation

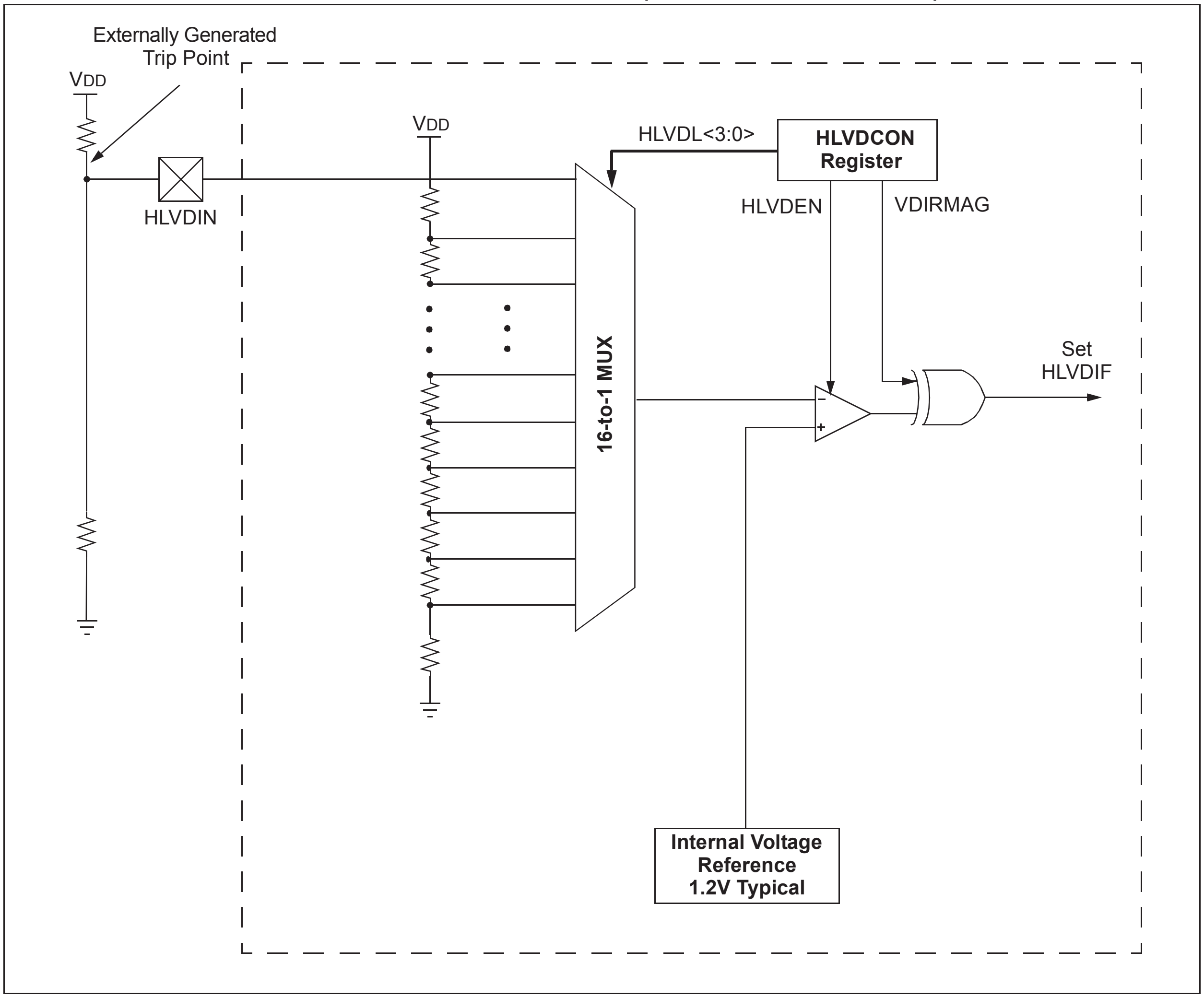
When the HLVD module is enabled, a comparator uses an internally generated reference voltage as the set point. The set point is compared with the trip point, where each node in the resistor divider represents a trip point voltage. The "trip point" voltage is the voltage level at which the device detects a high or low-voltage event, depending on the configuration of the module.

When the supply voltage is equal to the trip point, the voltage tapped off of the resistor array is equal to the internal reference voltage generated by the voltage reference module. The comparator then generates an interrupt signal by setting the HLVDIF bit.

The trip point voltage is software programmable to any one of 16 values. The trip point is selected by programming the HLVDL<3:0> bits (HLVDCON<3:0>).

Additionally, the HLVD module allows the user to supply the trip voltage to the module from an external source. This mode is enabled when bits, HLVDL<3:0>, are set to '1111'. In this state, the comparator input is multiplexed from the external input pin, HLVDIN. This gives users flexibility because it allows them to configure the HLVD interrupt to occur at any voltage in the valid operating range.

FIGURE 25-1: HLVD MODULE BLOCK DIAGRAM (WITH EXTERNAL INPUT)



27.0 SPECIAL FEATURES OF THE CPU

PIC18F47J13 Family devices include several features intended to maximize reliability and minimize cost through elimination of external components. These are:

- Oscillator Selection
- Resets:
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Brown-out Reset (BOR)
- Interrupts
- Watchdog Timer (WDT)
- Fail-Safe Clock Monitor (FSCM)
- Two-Speed Start-up
- Code Protection
- In-Circuit Serial Programming (ICSP)

The oscillator can be configured for the application depending on frequency, power, accuracy and cost. All of the options are discussed in detail in **Section 3.0 "Oscillator Configurations"**.

A complete discussion of device Resets and interrupts is available in previous sections of this data sheet. In addition to their Power-up and Oscillator Start-up Timers provided for Resets, the PIC18F47J13 Family of devices has a configurable Watchdog Timer (WDT), which is controlled in software.

The inclusion of an internal RC oscillator also provides the additional benefits of a Fail-Safe Clock Monitor (FSCM) and Two-Speed Start-up. FSCM provides for background monitoring of the peripheral clock and automatic switchover in the event of its failure. Two-Speed Start-up enables code to be executed almost immediately on start-up, while the primary clock source completes its start-up delays.

All of these features are enabled and configured by setting the appropriate Configuration register bits.

27.1 Configuration Bits

The Configuration bits can be programmed to select various device configurations. The configuration data is stored in the last four words of Flash program memory; Figure 6-1 depicts this. The configuration data gets loaded into the volatile Configuration registers, CONFIG1L through CONFIG4H, which are readable and mapped to program memory, starting at location 300000h.

Table 27-2 provides a complete list of the Configuration bits and Device IDs. A detailed explanation of the various bit functions is provided in Register 27-1 through Register 27-6.

27.1.1 CONSIDERATIONS FOR CONFIGURING THE PIC18F47J13 FAMILY DEVICES

Unlike some previous PIC18 microcontrollers, PIC18F47J13 Family devices do not use persistent memory registers to store configuration information. The Configuration registers, CONFIG1L through CONFIG4H, are implemented as volatile memory.

Immediately after power-up, or after a device Reset, the microcontroller hardware automatically loads the CONFIG1L through CONFIG4L registers with configuration data stored in nonvolatile Flash program memory. The last four words of Flash program memory, known as the Flash Configuration Words (FCW), are used to store the configuration data.

Table 27-1 provides the Flash program memory, which will be loaded into the corresponding Configuration register.

When creating applications for these devices, users should always specifically allocate the location of the FCW for configuration data. This is to make certain that program code is not stored in this address when the code is compiled.

The four Most Significant bits (MSb) of the FCW, corresponding to CONFIG1H, CONFIG2H, CONFIG3H and CONFIG4H, should always be programmed to '1111'. This makes these FCWs appear to be NOP instructions in the remote event that their locations are ever executed by accident.

The four MSbs of the CONFIG1H, CONFIG2H, CONFIG3H and CONFIG4H registers are not implemented, so writing '1's to their corresponding FCW has no effect on device operation.

To prevent inadvertent configuration changes during code execution, the Configuration registers, CONFIG1L through CONFIG4L, are loaded only once per power-up or Reset cycle. User's firmware can still change the configuration by using self-reprogramming to modify the contents of the FCW.

Modifying the FCW will not change the active contents being used in the CONFIG1L through CONFIG4H registers until after the device is reset.

REGISTER 27-8: CONFIG4H: CONFIGURATION REGISTER 4 HIGH (BYTE ADDRESS 300007h)

U-1	U-1	U-1	U-1	U-0	U-0	R/WO-1	R/WO-1
—	—	—	—	—	—	WPEND	WPDIS
bit 7							bit 0

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

bit 7-4 **Unimplemented:** Program the corresponding Flash Configuration bit to ‘1’

bit 3-2 **Unimplemented:** Read as ‘0’

bit 1 **WPEND:** Write-Protect Disable bit
1 = Flash pages, WPFP<6:0> to (Configuration Words page), are erase/write-protected
0 = Flash pages 0 to WPFP<6:0> are erase/write-protected

bit 0 **WPDIS:** Write-Protect Disable bit
1 = WPFP<5:0>, WPEND and WPCFG bits are ignored; all Flash memory may be erased or written
0 = WPFP<5:0>, WPEND and WPCFG bits are enabled; erase/write-protect is active for the selected region(s)

REGISTER 27-9: DEVID1: DEVICE ID REGISTER 1 FOR PIC18F47J13 FAMILY DEVICES (BYTE ADDRESS 3FFFFEh)

R	R	R	R	R	R	R	R
DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0
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 7-5 **DEV<2:0>:** Device ID bits
These bits are used with DEV<10:3> bits in Device ID Register 2 to identify the part number. See Register 27-10.

bit 4-0 **REV<4:0>:** Revision ID bits
These bits are used to indicate the device revision.

TABLE 28-2: PIC18F47J13 FAMILY INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BIT-ORIENTED 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, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call Subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to Address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop Top of Return Stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push Top of Return Stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software Device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from Interrupt Enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

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 an 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 the 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 two-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.

RCALL	**Relative Call**
Syntax:	RCALL n
Operands:	$-1024 \leq n \leq 1023$
Operation:	(PC) + 2 → TOS, (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	`1101` `1nnn` `nnnn` `nnnn`
Description:	Subroutine call with a jump up to 1K from the current location. First, return address (PC + 2) is pushed onto the stack. Then, add the 2's complement number '2n' 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 a 2-cycle instruction.
Words:	1
Cycles:	2

Q Cycle Activity:

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

Example: `HERE    RCALL Jump`

Before Instruction
PC = Address `(HERE)`
After Instruction
PC = Address `(Jump)`
TOS = Address `(HERE + 2)`

RESET	**Reset**
Syntax:	RESET
Operands:	None
Operation:	Reset all registers and flags that are affected by a $\overline{\text{MCLR}}$ Reset.
Status Affected:	All
Encoding:	`0000` `0000` `1111` `1111`
Description:	This instruction provides a way to execute a $\overline{\text{MCLR}}$ Reset in software.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Start reset	No operation	No operation

Example: `RESET`

After Instruction
Registers = Reset Value
Flags* = Reset Value

XORWF	Exclusive OR W with f
Syntax:	XORWF f {,d {,a}}
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) .XOR. (f) $\rightarrow$ dest
Status Affected:	N, Z
Encoding:	`0001` `10da` `ffff` `ffff`
Description:	Exclusive OR the contents of W with register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in the register 'f' (default). If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default). If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \leq 95$ (5Fh). See **Section 28.2.3 "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: `XORWF REG, 1, 0`

Before Instruction
REG = AFh
W = B5h
After Instruction
REG = 1Ah
W = B5h

28.2.3 BYTE-ORIENTED AND BIT-ORIENTED INSTRUCTIONS IN INDEXED LITERAL OFFSET MODE

Note:	Enabling the PIC18 instruction set extension may cause legacy applications to behave erratically or fail entirely.

In addition to eight new commands in the extended set, enabling the extended instruction set also enables Indexed Literal Offset Addressing (**Section 6.6.1 "Indexed Addressing with Literal Offset"**). This has a significant impact on the way that many commands of the standard PIC18 instruction set are interpreted.

When the extended set is disabled, addresses embedded in opcodes are treated as literal memory locations: either as a location in the Access Bank (a = `0`) or in a GPR bank designated by the BSR (a = `1`). When the extended instruction set is enabled and a = `0`, however, a file register argument of 5Fh or less is interpreted as an offset from the pointer value in FSR2 and not as a literal address. For practical purposes, this means that all instructions that use the Access RAM bit as an argument – that is, all byte-oriented and bit-oriented instructions, or almost half of the core PIC18 instructions – may behave differently when the extended instruction set is enabled.

When the content of FSR2 is 00h, the boundaries of the Access RAM are essentially remapped to their original values. This may be useful in creating backward-compatible code. If this technique is used, it may be necessary to save the value of FSR2 and restore it when moving back and forth between C and assembly routines in order to preserve the Stack Pointer. Users must also keep in mind the syntax requirements of the extended instruction set (see **Section 28.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**).

Although the Indexed Literal Offset mode can be very useful for dynamic stack and pointer manipulation, it can also be very annoying if a simple arithmetic operation is carried out on the wrong register. Users who are accustomed to the PIC18 programming must keep in mind that, when the extended instruction set is enabled, register addresses of 5Fh or less are used for Indexed Literal Offset Addressing.

Representative examples of typical byte-oriented and bit-oriented instructions in the Indexed Literal Offset mode are provided on the following page to show how execution is affected. The operand conditions shown in the examples are applicable to all instructions of these types.

28.2.3.1 Extended Instruction Syntax with Standard PIC18 Commands

When the extended instruction set is enabled, the file register argument, 'f', in the standard byte-oriented and bit-oriented commands is replaced with the literal offset value, 'k'. As already noted, this occurs only when 'f' is less than or equal to 5Fh. When an offset value is used, it must be indicated by square brackets ("[ ]"). As with the extended instructions, the use of brackets indicates to the compiler that the value is to be interpreted as an index or an offset. Omitting the brackets, or using a value greater than 5Fh within the brackets, will generate an error in the MPASM Assembler.

If the index argument is properly bracketed for Indexed Literal Offset Addressing, the Access RAM argument is never specified; it will automatically be assumed to be '`0`'. This is in contrast to standard operation (extended instruction set disabled), when 'a' is set on the basis of the target address. Declaring the Access RAM bit in this mode will also generate an error in the MPASM Assembler.

The destination argument 'd' functions as before.

In the latest versions of the MPASM Assembler, language support for the extended instruction set must be explicitly invoked. This is done with either the command line option, `/y`, or the PE directive in the source listing.

28.2.4 CONSIDERATIONS WHEN ENABLING THE EXTENDED INSTRUCTION SET

It is important to note that the extensions to the instruction set may not be beneficial to all users. In particular, users who are not writing code that uses a software stack may not benefit from using the extensions to the instruction set.

Additionally, the Indexed Literal Offset Addressing mode may create issues with legacy applications written to the PIC18 assembler. This is because instructions in the legacy code may attempt to address registers in the Access Bank below 5Fh. Since these addresses are interpreted as literal offsets to FSR2 when the instruction set extension is enabled, the application may read or write to the wrong data addresses.

When porting an application to the PIC18F47J13 Family, it is very important to consider the type of code. A large, re-entrant application that is written in C and would benefit from efficient compilation will do well when using the instruction set extensions. Legacy applications that heavily use the Access Bank will most likely not benefit from using the extended instruction set.

TABLE 30-10: 96 MHz PLL CLOCK TIMING SPECIFICATIONS (VDDCORE = 2.35V TO 2.75V)

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
F10	FOSC	Oscillator Frequency Range	4	—	48	MHz	
F11	FSYS	On-Chip VCO System Frequency	—	96	—	MHz	
F12	t_{rc}	PLL Start-up Time (lock time)	—	—	2	ms	

TABLE 30-11: 4x PLL CLOCK TIMING SPECIFICATIONS

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
F10	FPLLIN	PLL Input Frequency Range	4	—	12	MHz	
F11	FPLLO	PLL Output Frequency (4x FPLLIN)	16	—	48	MHz	
F12	t_{rc}	PLL Start-up Time (lock time)	—	—	2	ms	

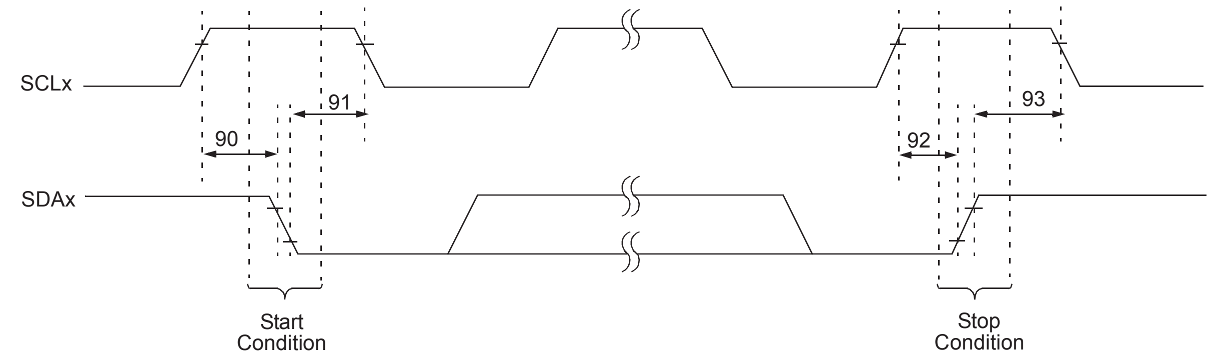
† Data in “Typ” column is at 3.3V, 25°C, unless otherwise stated. These parameters are for design guidance only and are not tested.

TABLE 30-12: INTERNAL RC ACCURACY (INTOSC AND INTRC SOURCES)

Param No.	Device	Min	Typ	Max	Units	Conditions	
	INTOSC Accuracy @ Freq = 8 MHz, 4 MHz, 2 MHz, 1 MHz, 500 kHz, 250 kHz, 125 kHz, 31 kHz(1)						
	All Devices	-1	±0.15	+1	%	0°C to +85°C	VDD = 2.4V-3.6V, VDDCORE = 2.3V-2.7V
		-1	±0.25	+1	%	-40°C to +85°C	VDD = 2.0V-3.6V, VDDCORE = 2.0V-2.7V
	INTRC Accuracy @ Freq = 31 kHz(1)						
	All Devices	20.3	—	42.2	kHz	-40°C to +85°C	VDD = 2.0V-3.6V, VDDCORE = 2.0V-2.7V

Note 1: The accuracy specification of the 31 kHz clock is determined by which source is providing it at a given time. When INTSRC (OSCTUNE<7>) is ‘1’, use the INTOSC accuracy specification. When INTSRC is ‘0’, use the INTRC accuracy specification.

FIGURE 30-19: MSSPx I²C BUS START/STOP BITS TIMING WAVEFORMS

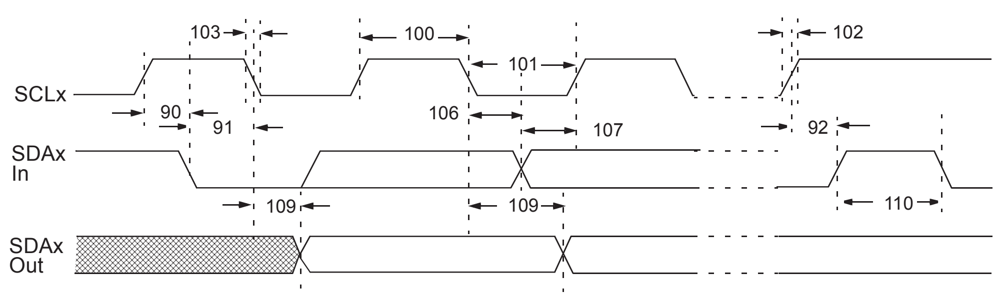


Note: Refer to Figure 30-4 for load conditions.

TABLE 30-27: MSSPx I²C BUS START/STOP BITS REQUIREMENTS

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
90	TSU:STA	Start Condition Setup Time	100 kHz mode	2(TOSC)(BRG + 1)	—	ns	Only relevant for Repeated Start condition
			400 kHz mode	2(TOSC)(BRG + 1)	—		
91	THD:STA	Start Condition Hold Time	100 kHz mode	2(TOSC)(BRG + 1)	—	ns	After this period, the first clock pulse is generated
			400 kHz mode	2(TOSC)(BRG + 1)	—		
92	TSU:STO	Stop Condition Setup Time	100 kHz mode	2(TOSC)(BRG + 1)	—	ns	
			400 kHz mode	2(TOSC)(BRG + 1)	—		
93	THD:STO	Stop Condition Hold Time	100 kHz mode	2(TOSC)(BRG + 1)	—	ns	
			400 kHz mode	2(TOSC)(BRG + 1)	—		

FIGURE 30-20: MSSPx I²C BUS DATA TIMING



Note: Refer to Figure 30-4 for load conditions.