**Welcome to E-XFL.COM**

## What is "Embedded - Microcontrollers"?

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

## Applications of "Embedded - Microcontrollers"

| Details | |
|---|---|
| Product Status | Active |
| Core Processor | PIC |
| Core Size | 8-Bit |
| Speed | 64MHz |
| Connectivity | I²C, SPI, UART/USART |
| Peripherals | Brown-out Detect/Reset, POR, PWM, WDT |
| Number of I/O | 25 |
| Program Memory Size | 32KB (32K x 8) |
| Program Memory Type | FLASH |
| EEPROM Size | 256 x 8 |
| RAM Size | 2K x 8 |
| Voltage - Supply (Vcc/Vdd) | 1.8V ~ 5.5V |
| Data Converters | A/D 24x10b; D/A 1x5b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 28-SSOP (0.209", 5.30mm Width) |
| Supplier Device Package | 28-SSOP |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/pic18f25q10t-i-ss |

### 3.7.1 CONFIG1

**Name:** CONFIG1
**Address:** 0x300000

Configuration word 1

Oscillators

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | | | FCMEN | | CSWEN | | | $\overline{\text{CLKOUTEN}}$ |
| Access | | | R/W | | R/W | | | R/W |
| Reset | | | 1 | | 1 | | | 1 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | RSTOSC[2:0] | | | | FEXTOSC[2:0] | | |
| Access | | R/W | R/W | R/W | | R/W | R/W | R/W |
| Reset | | 1 | 1 | 1 | | 1 | 1 | 1 |

**Bit 13 – FCMEN** Fail-Safe Clock Monitor Enable bit

| Value | Description |
|---|---|
| 1 | Fail Safe Clock Monitor enabled |
| 0 | Fail Safe Clock Monitor disabled |

**Bit 11 – CSWEN** Clock Switch Enable bit

| Value | Description |
|---|---|
| 1 | Writing to NOSC and NDIV is allowed |
| 0 | The NOSC and NDIV bits cannot be changed by user software |

**Bit 8 – $\overline{\text{CLKOUTEN}}$** Clock Out Enable bit
If FEXTOSC = HS, XT, LP, then this bit is ignored.

Otherwise:

| Value | Description |
|---|---|
| 1 | CLKOUT function is disabled; I/O function on OSC2 |
| 0 | CLKOUT function is enabled; $F_{OSC}$/4 clock appears at OSC2 |

**Bits 6:4 – RSTOSC[2:0]** Power-up Default Value for COSC bits
This value is the Reset default value for COSC and selects the oscillator first used by user software. Refer to COSC operation.

| Value | Description |
|---|---|
| 111 | EXTOSC operating per FEXTOSC bits (device manufacturing default) |
| 110 | HFINTOSC with HFFRQ = 4 MHz and CDIV = 4:1 |
| 101 | LFINTOSC |
| 100 | SOSC |
| 011 | Reserved |
| 010 | EXTOSC with 4x PLL, with EXTOSC operating per FEXTOSC bits |

then the duty cycle defaults to 50% for all values of DC except 0b00 in which case the duty cycle is 0% (constant low output).

The duty cycle can be changed while the module is enabled. However, in order to prevent glitches on the output, the 5.6.1.2 DC bits should only be changed when the module is disabled (5.6.1.1 EN = 0).

**Important:** The 5.6.1.2 DC value at reset is 10. This makes the default duty cycle 50% and not 0%.

## 5.4 Operation in Sleep Mode

The reference clock module continues to operate and provide a signal output in Sleep for all clock source selections except $F_{OSC}$ (5.6.2.1 CLK=0).

# 7. (PMD) Peripheral Module Disable

This module provides the ability to selectively enable or disable a peripheral. Disabling a peripheral places it in its lowest possible power state. The user can disable unused modules to reduce the overall power consumption.

The PIC18F24/25Q10 devices address this requirement by allowing peripheral modules to be selectively enabled or disabled. Disabling a peripheral places it in the lowest possible power mode.

**Important:** All modules are ON by default following any system Reset.

## 7.1 Disabling a Module

A peripheral can be disabled by setting the corresponding peripheral disable bit in the PMDx register. Disabling a module has the following effects:

- The module is held in Reset and does not function.
- All the SFRs pertaining to that peripheral become “unimplemented”
  - Writing is disabled
  - Reading returns 0x00
- Module outputs are disabled

**Related Links**

15.1 I/O Priorities

## 7.2 Enabling a Module

Clearing the corresponding module disable bit in the PMDx register, re-enables the module and the SFRs will reflect the Power-on Reset values.

**Important:** There should be no reads/writes to the module SFRs for at least two instruction cycles after it has been re-enabled.

# 9. (WWDT) Windowed Watchdog Timer

The Watchdog Timer (WDT) is a system timer that generates a Reset if the firmware does not issue a CLRWDT instruction within the time-out period. The Watchdog Timer is typically used to recover the system from unexpected events. The Windowed Watchdog Timer (WWDT) differs in that CLRWDT instructions are only accepted when they are performed within a specific window during the time-out period.

The WWDT has the following features:

- Selectable clock source
- Multiple operating modes
  - WWDT is always on
  - WWDT is off when in Sleep
  - WWDT is controlled by software
  - WWDT is always off
- Configurable time-out period is from 1 ms to 256s (nominal)
- Configurable window size from 12.5% to 100% of the time-out period
- Multiple Reset conditions

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.

### 10.4.3 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 which are to be read or written. Since the FSRs are themselves located in RAM as Special File Registers, 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 following example of clearing an entire RAM bank.

**Example 10-3. How to Clear RAM (Bank 1) Using Indirect Addressing**

```
     LFSR    FSR0,100h   ; Set FSR0 to beginning of Bank1
NEXT:
     CLRF    POSTINC0    ; Clear location in Bank1 then increment FSR0

     BTFSS   FSR0H,1     ; Has high FSR0 byte incremented to next bank?
     BRA     NEXT        ; NO, clear next byte in Bank1

CONTINUE:                ; YES, continue
```

#### 10.4.3.1 FSR Registers and the INDF Operand

At the core of indirect addressing are three sets of registers: FSR0, FSR1 and FSR2. Each represents a pair of 8-bit registers, FSRnH and FSRnL. Each FSR pair holds a 12-bit value, therefore, the four upper bits of the FSRnH register are not used. The 12-bit FSR value can address the entire range of the data memory in a linear fashion. The FSR register pairs, then, serve as pointers to data memory locations.

Indirect addressing is accomplished with a set of Indirect File Operands, INDF0 through INDF2. These can be thought of as “virtual” registers; they are mapped in the SFR space but are not physically implemented. Reading or writing to a particular INDF register actually accesses its corresponding FSR register pair. A read from INDF1, for example, reads the data at the address indicated by FSR1H:FSR1L. Instructions that use the INDF registers as operands actually use the contents of their corresponding FSR as a pointer to the instruction’s target. The INDF operand is just a convenient way of using the pointer.

Because indirect addressing uses a full 12-bit address, the FSR value can target any location in any bank regardless of the BSR value. However, the Access RAM bit must be cleared to 0 to ensure that the INDF register in Access space is the object of the operation instead of a register in one of the other banks. The assembler default value for the Access RAM bit is zero when targeting any of the indirect operands.

#### 10.4.3.2 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 which cannot be directly read or written. Accessing these registers

### 10.8.4 STKPTR

**Name:** STKPTR
**Address:** 0xFFC

Stack Pointer Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | STKPTR[4:0] | | | | |
| Access | | | | R/W | R/W | R/W | R/W | R/W |
| Reset | | | | 0 | 0 | 0 | 0 | 0 |

**Bits 4:0 – STKPTR[4:0]** Stack Pointer Location bits

### 11.5.2 NVMCON1

**Name:** NVMCON1
**Address:** 0xF80

Nonvolatile Memory Control 1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | SECER | SECWR | WR | | | SECRD | RD |
| Access | | R/S/HC | R/S/HC | R/S/HC | | | R/S/HC | R/S/HC |
| Reset | | 0 | 0 | 0 | | | 0 | 0 |

**Bit 6 – SECER**
NVM Sector Erase Enable Control bit

| Value | Condition | Description |
|---|---|---|
| 1 | NVMADR points to PFM | Immediately following the sector erase unlock sequence, perform a sector erase operation. Stays set until operation is complete. Cannot be cleared by software. |
| 0 | NVMADR points to PFM | NVM sector erase operation is complete and inactive. |

**Bit 5 – SECWR**
NVM Sector Write Enable Control bit

| Value | Condition | Description |
|---|---|---|
| 1 | NVMADR points to PFM or CONFIG | Immediately following the sector write unlock sequence, initiates the PFM sector write operation. Stays set until the write is complete. Cannot be cleared by software. |
| 0 | NVMADR points to PFM or CONFIG | NVM sector write operation is complete and inactive. |

**Bit 4 – WR**
NVM Write Control bit

| Value | Condition | Description |
|---|---|---|
| 1 | NVMADR points to DFM | Immediately following the DFM write unlock sequence, initiates a DFM byte erase/write sequence using data in the NVMDATL register. Stays set until the operation is complete. Cannot be cleared by software. |
| 1 | NVMADR points to PFM | Immediately following the PFM write unlock sequence, initiates an NVM word write sequence using data in the NVMDATH:L register pair. Stays set until the operation is complete. Cannot be cleared by software. |
| 0 | NVMADR points to PFM or DFM | NVM byte/word write operation is complete and inactive. |

**Bit 1 – SECRD**
PFM Sector Read Enable Control bit

## 17.8 Register Summary - PPS

| Address | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x0E9B | PPSLOCK | 7:0 | | | | | | | | PPSLOCKED |
| 0x0E9C | INT0PPS | 7:0 | | | | | PORT | PIN[2:0] | | |
| 0x0E9D | INT1PPS | 7:0 | | | | | PORT | PIN[2:0] | | |
| 0x0E9E | INT2PPS | 7:0 | | | | | PORT | PIN[2:0] | | |
| 0x0E9F | T0CKIPPS | 7:0 | | | | | PORT | PIN[2:0] | | |
| 0x0EA0 | T1CKIPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA1 | T1GPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA2 | T3CKIPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA3 | T3GPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA4 | T5CKIPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA5 | T5GPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA6 | T2INPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA7 | T4INPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA8 | T6INPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EA9 | ADACTPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EAA | CCP1PPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EAB | CCP2PPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EAC | CWG1PPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EAD | MDCARLPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EAE | MDCARHPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EAF | MDSRCPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EB0 | RX1PPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EB1 | CK1PPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EB2 | SSP1CLKPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EB3 | SSP1DATPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EB4 | SSP1SSPPS | 7:0 | | | | PORT[1:0] | | PIN[2:0] | | |
| 0x0EB5 ... 0x0EE1 | Reserved | | | | | | | | | |
| 0x0EE2 | RA0PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE3 | RA1PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE4 | RA2PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE5 | RA3PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE6 | RA4PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE7 | RA5PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE8 | RA6PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EE9 | RA7PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EEA | RB0PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EEB | RB1PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EEC | RB2PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EED | RB3PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EEE | RB4PPS | 7:0 | | | | PPS[4:0] | | | | |
| 0x0EEF | RB5PPS | 7:0 | | | | PPS[4:0] | | | | |

## 18.1 Timer0 Operation

Timer0 can operate as either an 8-bit or 16-bit timer. The mode is selected with the T016BIT bit.

### 18.1.1 8-bit Mode

In this mode Timer0 increments on the rising edge of the selected clock source. A prescaler on the clock input gives several prescale options (see prescaler control bits, T0CKPS).

In this mode as shown in Figure 18-1, a buffered version of TMR0H is maintained. This is compared with the value of TMR0L on each cycle of the selected clock source. When the two values match, the following events occur:

- TMR0L is reset
- The contents of TMR0H are copied to the TMR0H buffer for next comparison

### 18.1.2 16-Bit Mode

In this mode Timer0 increments on the rising edge of the selected clock source. A prescaler on the clock input gives several prescale options (see prescaler control bits, T0CKPS).

In this mode TMR0H:TMR0L form the 16-bit timer value. As shown in Figure 18-1, read and write of the TMR0H register are buffered. TMR0H register is updated with the contents of the high byte of Timer0 during a read of TMR0L register. Similarly, a write to the high byte of Timer0 takes place through the TMR0H buffer register. The high byte is updated with the contents of TMR0H register when a write occurs to TMR0L register. This allows all 16 bits of Timer0 to be read and written at the same time.

Timer0 rolls over to 0x0000 on incrementing past 0xFFFF. This makes the timer free running. TMR0L/H registers cannot be reloaded in this mode once started.

## 18.2 Clock Selection

Timer0 has several options for clock source selections, option to operate synchronously/asynchronously and a programmable prescaler.

### 18.2.1 Clock Source Selection

The T0CS bits are used to select the clock source for Timer0. The possible clock sources are listed in the table below.

**Table 18-1. Timer 0 Clock Source Selections**

| T0CS | Clock Source |
|---|---|
| 111 | Reserved |
| 110 | Reserved |
| 101 | SOSC |
| 100 | LFINTOSC |
| 011 | HFINTOSC |
| 010 | Fosc/4 |
| 001 | Pin selected by T0CKIPPS (Inverted) |
| 000 | Pin selected by T0CKIPPS (Non-inverted) |

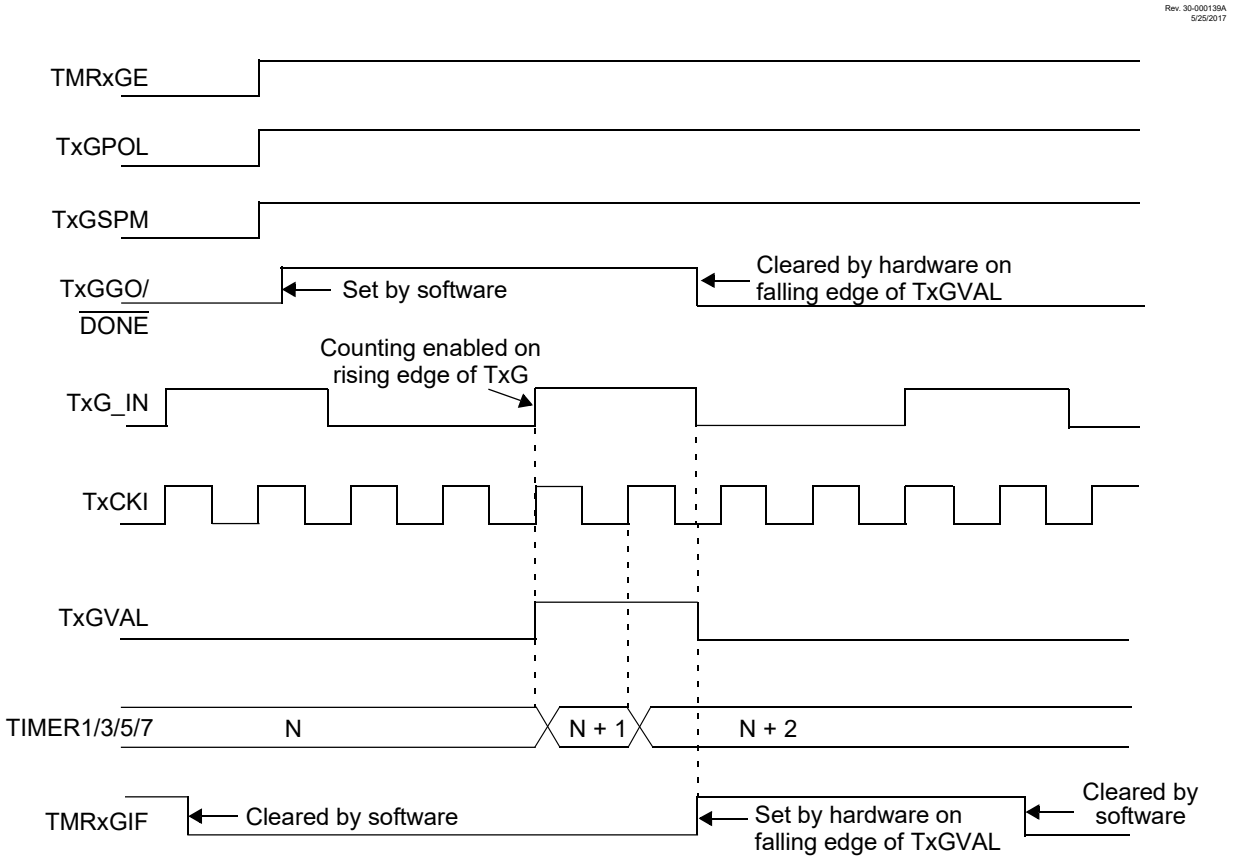
### 19.7.4 Timer1 Gate Single-Pulse Mode

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 19.14.2.4 GSPM bit. Next, the 19.14.2.5 GGO/DONE bit must be set. The Timer1 will be fully enabled on the next incrementing edge. On the next trailing edge of the pulse, the GGO/$\overline{\text{DONE}}$ bit will automatically be cleared. No other gate events will be allowed to increment Timer1 until the GGO/$\overline{\text{DONE}}$ bit is once again set in software.

Clearing the GSPM bit will also clear the GGO/$\overline{\text{DONE}}$ bit. See figure below 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 below for timing details.

**Figure 19-6. TIMER1 GATE SINGLE-PULSE MODE**



### 21.6.4 CCPTMRS

**Name:** CCPTMRS
**Address:** 0xFAD

CCP Timers Control Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | P4TSEL[1:0] | | P3TSEL[1:0] | | C2TSEL[1:0] | | C1TSEL[1:0] | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**Bits 4:5, 6:7 – PnTSEL** PWMn Timer Selection bits

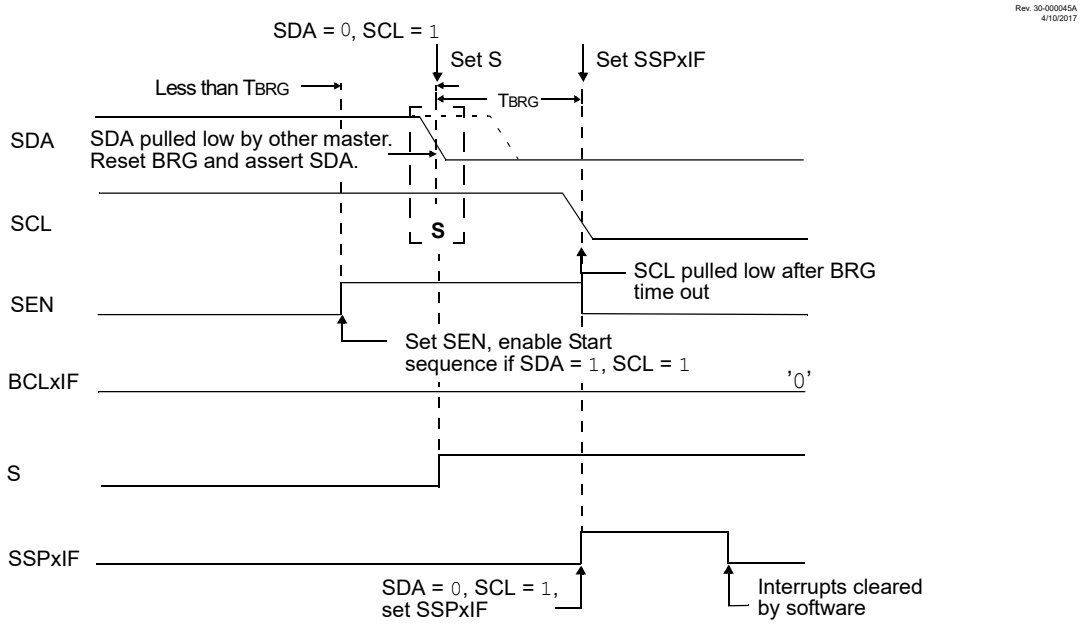
| Value | Description |
|---|---|
| 11 | PWMn based on Timer6 |
| 10 | PWMn based on Timer4 |
| 01 | PWMn based on Timer2 |
| 00 | Reserved |

**Bits 0:1, 2:3 – CnTSEL** CCPn Timer Selection bits

| Value | Description |
|---|---|
| 11 | CCPn is based off Timer5 in Capture/Compare mode and Timer6 in PWM mode |
| 10 | CCPn is based off Timer3 in Capture/Compare mode and Timer4 in PWM mode |
| 01 | CCPn is based off Timer1 in Capture/Compare mode and Timer2 in PWM mode |
| 00 | Reserved |

If the SDA pin is sampled low during this count, the BRG is reset and the SDA line is asserted early (Figure 26-35). If, however, a '1' is sampled on the SDA pin, the SDA pin is asserted low at the end of the BRG count. The Baud Rate Generator is then reloaded and counts down to zero; if the SCL pin is sampled as '0' during this time, a bus collision does not occur. At the end of the BRG count, the SCL pin is asserted low.

**Figure 26-35. BRG Reset Due to SDA Arbitration During Start Condition**




**Important:** 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 SDA 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.

### 26.6.13.2 Bus Collision During a Repeated Start Condition

During a Repeated Start condition, a bus collision occurs if:

1. A low level is sampled on SDA when SCL goes from low level to high level (Case 1).
2. SCL goes low before SDA is asserted low, indicating that another master is attempting to transmit a data '1' (Case 2).

When the user releases SDA and the pin is allowed to float high, the BRG is loaded with SSPxADD and counts down to zero. The SCL pin is then deasserted and when sampled high, the SDA pin is sampled.

If SDA is low, a bus collision has occurred (i.e., another master is attempting to transmit a data '0', Figure 26-36). If SDA is sampled high, the BRG is reloaded and begins counting. If SDA goes from high-to-low before the BRG times out, no bus collision occurs because no two masters can assert SDA at exactly the same time.

### 26.9.5 SSPxBUF

**Name:** SSPxBUF
**Address:** 0x0F91

MSSP Data Buffer Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | BUF[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – BUF[7:0]** MSSP Input and Output Data Buffer bits

**Note:** If the RX/DT function is on an analog pin, the corresponding ANSEL bit must be cleared for the receiver to function.

#### 27.3.1.6 Receive Overrun Error

The receive FIFO buffer can hold two characters. An overrun error will be generated if a third character, in its entirety, is received before RCxREG is read to access the FIFO. When this happens the OERR bit of the RCxSTA register is set. Previous data in the FIFO will not be overwritten. The two characters in the FIFO buffer can be read, however, no additional characters will be received until the error is cleared. The OERR bit can only be cleared by clearing the overrun condition. If the overrun error occurred when the SREN bit is set and CREN is clear then the error is cleared by reading RCxREG. If the overrun occurred when the CREN bit is set then the error condition is cleared by either clearing the CREN bit of the RCxSTA register or by clearing the SPEN bit which resets the EUSART.

#### 27.3.1.7 Receiving 9-Bit Characters

The EUSART supports 9-bit character reception. When the RX9 bit of the RCxSTA register is set the EUSART will shift nine bits into the RSR for each character received. The RX9D bit of the RCxSTA register is the ninth, and Most Significant, data bit of the top unread character in the receive FIFO. When reading 9-bit data from the receive FIFO buffer, the RX9D data bit must be read before reading the eight Least Significant bits from the RCxREG.

#### 27.3.1.8 Synchronous Master Reception Setup

1. Initialize the SPxBRGH:SPxBRGL register pair and set or clear the BRG16 bit, as required, to achieve the desired baud rate.
2. Select the receive input pin by writing the appropriate values to the RxyPPS register and RXxPPS register. Both selections should enable the same pin.
3. Select the clock output pin by writing the appropriate values to the RxyPPS register and CKxPPS register. Both selections should enable the same pin.
4. Clear the ANSEL bit for the RXx pin (if applicable).
5. Enable the synchronous master serial port by setting bits SYNC, SPEN and CSRC.
6. Ensure bits CREN and SREN are clear.
7. If interrupts are desired, set the RCxIE bit of the PIEx register and the GIE and PEIE bits of the INTCON register.
8. If 9-bit reception is desired, set bit RX9.
9. Start reception by setting the SREN bit or for continuous reception, set the CREN bit.
10. Interrupt flag bit RCxIF will be set when reception of a character is complete. An interrupt will be generated if the enable bit RCxIE was set.
11. Read the RCxSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
12. Read the 8-bit received data by reading the RCxREG register.
13. If an overrun error occurs, clear the error by either clearing the CREN bit of the RCxSTA register or by clearing the SPEN bit which resets the EUSART.

### 27.6.4 SPxBRG

**Name:** SPxBRG
**Address:** 0x0F9A

Baud Rate Determination Register

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | SPBRGH[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPBRGL[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 15:8 – SPBRGH[7:0]** Baud Rate High Byte Register

**Bits 7:0 – SPBRGL[7:0]** Baud Rate Low Byte Register

# 36. Instruction Set Summary

PIC18F24/25Q10 devices incorporate the standard set of 75 PIC18 core instructions, as well as an extended set of eight new instructions, for the optimization of code that is recursive or that utilizes a software stack. The extended set is discussed later in this section.

## 36.1 Standard Instruction Set

The standard PIC18 instruction set adds many enhancements to the previous PIC® MCU instruction sets, while maintaining an easy migration from these PIC® MCU instruction sets. Most instructions are a single program memory word (16 bits), but there are four instructions that require two program memory locations.

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

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

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

The PIC18 instruction set summary in Table 36-2 lists byte-oriented, bit-oriented, literal and control operations. Table 36-1 shows the opcode field descriptions.

Most byte-oriented instructions have three operands:

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

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

All bit-oriented instructions have three operands:

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

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

The literal instructions may use some of the following operands:

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

The control instructions may use some of the following operands:

- A program memory address (specified by ‘n’)

| NEGF | Negate f | | | |
|---|---|---|---|---|
| Syntax: | NEGF f {,a} | | | |
| Operands: | 0 ≤ f ≤ 255<br>a ∈ [0,1] | | | |
| Operation: | $(\overline{f}) + 1 \rightarrow f$ | | | |
| Status Affected: | N, OV, C, DC, Z | | | |
| Encoding: | 0110 | 110a | ffff | ffff |
| Description: | Location ‘f’ is negated using two’s complement. The result is placed in the data memory location ‘f’.<br>If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the GPR bank.<br>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 36.2.3 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 register ‘f’ |

| Example: | `NEGF REG, 1` |
|---|---|
| Before Instruction<br>REG = 0011 1010 [3Ah]<br>After Instruction<br>REG = 1100 0110 [C6h] | |

| NOP | No Operation | | | |
|---|---|---|---|---|
| Syntax: | NOP | | | |
| Operands: | None | | | |
| Operation: | No operation | | | |
| Status Affected: | None | | | |
| Encoding: | 0000<br>1111 | 0000<br>xxxx | 0000<br>xxxx | 0000<br>xxxx |

REG = F5h (1111 0101)

; [2's comp]

W = 0Eh (0000 1110)

C = 0

Z = 0

N = 1 ; result is negative

| SWAPF | Swap f | | | |
|---|---|---|---|---|
| Syntax: | SWAPF f {,d {,a}} | | | |
| Operands: | 0 ≤ f ≤ 255<br>d ∈ [0,1]<br>a ∈ [0,1] | | | |
| Operation: | (f<3:0>) → dest<7:4>,<br>(f<7:4>) → dest<3:0> | | | |
| Status Affected: | None | | | |
| Encoding: | 0011 | 10da | ffff | ffff |
| Description: | The upper and lower nibbles of register 'f' are exchanged. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed in register 'f' (default).<br>If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.<br>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 36.2.3 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: | SWAPF | REG, 1, 0 |
|---|---|---|
| Before Instruction<br>REG = 53h<br>After Instruction | | |

| PUSHL | Store Literal at FSR2, Decrement FSR2 | | | |
|---|---|---|---|---|
| Description: | The 8-bit literal ‘k’ is written to the data memory address specified by FSR2. FSR2 is decremented by 1 after the operation.<br>This instruction allows users to push values onto a software stack. | | | |
| Words: | 1 | | | |
| Cycles: | 1 | | | |

| Q Cycle Activity: | | | |
|---|---|---|---|
| Q1 | Q2 | Q3 | Q4 |
| Decode | Read ‘k’ | Process data | Write to destination |

| Example: | `PUSHL 08h` |
|---|---|
| Before Instruction<br>FSR2H:FSR2L = 01ECh<br>Memory (01ECh) = 00h<br>After Instruction<br>FSR2H:FSR2L = 01EBh<br>Memory (01ECh) = 08h | |

| SUBFSR | Subtract Literal from FSR | | | |
|---|---|---|---|---|
| Syntax: | SUBFSR f, k | | | |
| Operands: | 0 ≤ k ≤ 63 | | | |
| | f ∈ [ 0, 1, 2 ] | | | |
| Operation: | FSR(f) – k → FSRf | | | |
| Status Affected: | None | | | |
| Encoding: | 1110 | 1001 | ffkk | kkkk |
| Description: | The 6-bit literal ‘k’ is subtracted from the contents of the FSR specified by ‘f’. | | | |
| Words: | 1 | | | |
| Cycles: | 1 | | | |

| Q Cycle Activity: | | | |
|---|---|---|---|
| Q1 | Q2 | Q3 | Q4 |
| Decode | Read register ‘f’ | Process Data | Write to destination |

- Conditional assembly for multipurpose source files
- Directives that allow complete control over the assembly process

## 37.4 MPLINK Object Linker/MPLIB Object Librarian

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

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

The object linker/library features include:

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

## 37.5 MPLAB Assembler, Linker and Librarian for Various Device Families

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

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

## 37.6 MPLAB X SIM Software Simulator

The MPLAB X SIM Software Simulator allows code development in a PC-hosted environment by simulating the PIC MCUs and dsPIC DSCs on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a comprehensive stimulus controller. Registers can be logged to files for further run-time analysis. The trace buffer and logic analyzer display extend the power of the simulator to record and track program execution, actions on I/O, most peripherals and internal registers.

The MPLAB X SIM Software Simulator fully supports symbolic debugging using the MPLAB XC Compilers, and the MPASM and MPLAB Assemblers. The software simulator offers the flexibility to develop and debug code outside of the hardware laboratory environment, making it an excellent, economical software development tool.