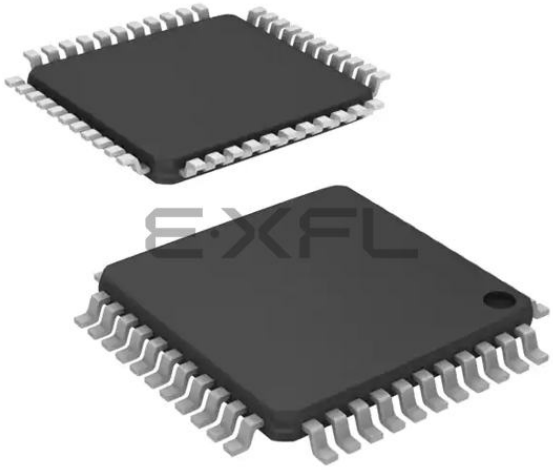


**Welcome to E-XFL.COM**

## 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 | dsPIC |
| Core Size | 16-Bit |
| Speed | 20 MIPS |
| Connectivity | CANbus, I²C, SPI, UART/USART |
| Peripherals | AC'97, Brown-out Detect/Reset, I²S, POR, PWM, WDT |
| Number of I/O | 30 |
| Program Memory Size | 48KB (16K x 24) |
| Program Memory Type | FLASH |
| EEPROM Size | 1K x 8 |
| RAM Size | 2K x 8 |
| Voltage - Supply (Vcc/Vdd) | 2.5V ~ 5.5V |
| Data Converters | A/D 13x12b |
| 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/dspic30f4013-20i-pt |

#### 2.4.2.4 Data Space Write Saturation

In addition to adder/subtracter saturation, writes to data space may also be saturated but without affecting the contents of the source accumulator. The data space write saturation logic block accepts a 16-bit, 1.15 fractional value from the round logic block as its input, together with overflow status from the original source (accumulator) and the 16-bit round adder. These are combined and used to select the appropriate 1.15 fractional value as output to write to data space memory.

If the SATDW bit in the CORCON register is set, data (after rounding or truncation) is tested for overflow and adjusted accordingly. For input data greater than 0x007FFF, data written to memory is forced to the maximum positive 1.15 value, 0x7FFF. For input data less than 0xFF8000, data written to memory is forced to the maximum negative 1.15 value, 0x8000. The Most Significant bit (MSb) of the source (bit 39) is used to determine the sign of the operand being tested.

If the SATDW bit in the CORCON register is not set, the input data is always passed through unmodified under all conditions.

### 2.4.3 BARREL SHIFTER

The barrel shifter is capable of performing up to 16-bit arithmetic or logic right shifts, or up to 16-bit left shifts in a single cycle. The source can be either of the two DSP accumulators, or the X bus (to support multi-bit shifts of register or memory data).

The shifter requires a signed binary value to determine both the magnitude (number of bits) and direction of the shift operation. A positive value shifts the operand right. A negative value shifts the operand left. A value of '0' does not modify the operand.

The barrel shifter is 40 bits wide, thereby obtaining a 40-bit result for DSP shift operations and a 16-bit result for MCU shift operations. Data from the X bus is presented to the barrel shifter between bit positions 16 to 31 for right shifts, and bit positions 0 to 16 for left shifts.

### 3.1.1 DATA ACCESS FROM PROGRAM MEMORY USING TABLE INSTRUCTIONS

This architecture fetches 24-bit wide program memory. Consequently, instructions are always aligned. However, as the architecture is modified Harvard, data can also be present in program space.

There are two methods by which program space can be accessed: via special table instructions, or through the remapping of a 16K word program space page into the upper half of data space (see **Section 3.1.2 "Data Access from Program Memory Using Program Space Visibility"**). The TBLRDL and TBLWTL instructions offer a direct method of reading or writing the lsw of any address within program space, without going through data space. The TBLRDH and TBLWTH instructions are the only method whereby the upper 8 bits of a program space word can be accessed as data.

The PC is incremented by two for each successive 24-bit program word. This allows program memory addresses to directly map to data space addresses. Program memory can thus be regarded as two 16-bit word-wide address spaces, residing side by side, each with the same address range. TBLRDL and TBLWTL access the space which contains the least significant data word, and TBLRDH and TBLWTH access the space which contains the MS Data Byte.

Figure 3-3 shows how the EA is created for table operations and data space accesses (PSV = 1). Here, P<23:0> refers to a program space word, whereas D<15:0> refers to a data space word.

A set of table instructions are provided to move byte or word-sized data to and from program space. (See Figure 3-4 and Figure 3-5.)

1. TBLRDL: Table Read Low
   *Word:* Read the lsw of the program address; P<15:0> maps to D<15:0>.
   *Byte:* Read one of the LSBs of the program address;
   P<7:0> maps to the destination byte when byte select = 0;
   P<15:8> maps to the destination byte when byte select = 1.
2. TBLWTL: Table Write Low (refer to **Section 5.0 "Flash Program Memory"** for details on Flash programming)
3. TBLRDH: Table Read High
   *Word:* Read the most significant word (msw) of the program address; P<23:16> maps to D<7:0>; D<15:8> will always be = 0.
   *Byte:* Read one of the MSBs of the program address;
   P<23:16> maps to the destination byte when byte select = 0;
   The destination byte will always be = 0 when byte select = 1.
4. TBLWTH: Table Write High (refer to **Section 5.0 "Flash Program Memory"** for details on Flash Programming)

**FIGURE 3-4: PROGRAM DATA TABLE ACCESS (LEAST SIGNIFICANT WORD)**

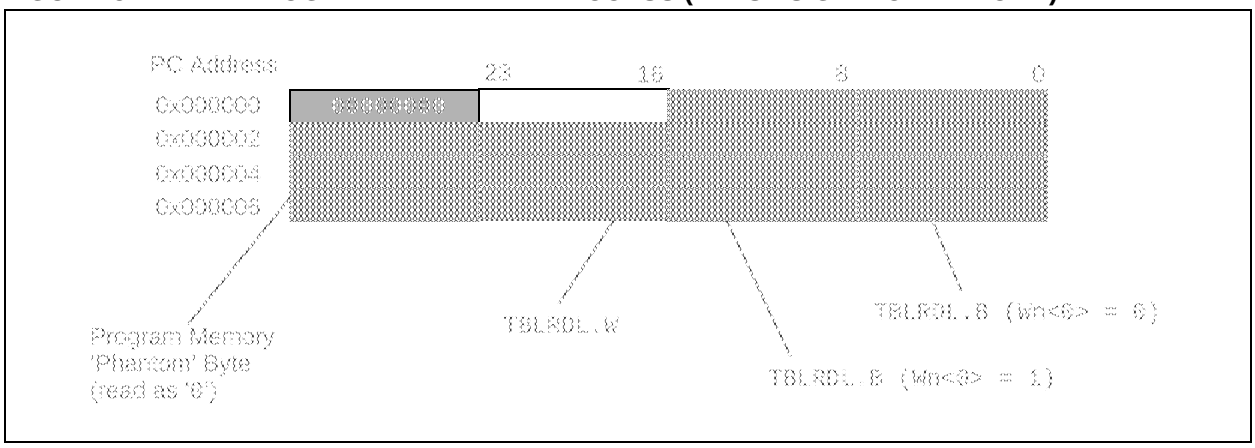
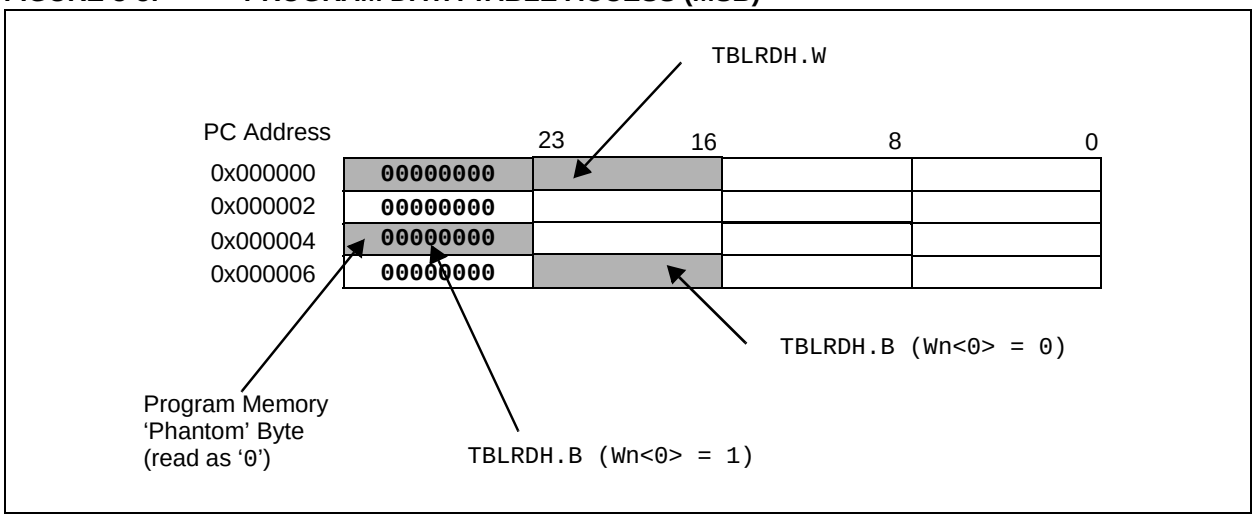


**FIGURE 3-5: PROGRAM DATA TABLE ACCESS (MSB)**




### 3.1.2 DATA ACCESS FROM PROGRAM MEMORY USING PROGRAM SPACE VISIBILITY

The upper 32 Kbytes of data space may optionally be mapped into any 16K word program space page. This provides transparent access of stored constant data from X data space without the need to use special instructions (i.e., TBLRDL/H, TBLWTL/H instructions).

Program space access through the data space occurs if the MSb of the data space, EA, is set and program space visibility is enabled by setting the PSV bit in the Core Control register (CORCON). The functions of CORCON are discussed in **Section 2.4 "DSP Engine"**.

Data accesses to this area add an additional cycle to the instruction being executed, since two program memory fetches are required.

Note that the upper half of addressable data space is always part of the X data space. Therefore, when a DSP operation uses program space mapping to access this memory region, Y data space should typically contain state (variable) data for DSP operations, whereas X data space should typically contain coefficient (constant) data.

Although each data space address, 0x8000 and higher, maps directly into a corresponding program memory address (see Figure 3-6), only the lower 16 bits of the 24-bit program word are used to contain the data. The upper 8 bits should be programmed to force an illegal instruction to maintain machine robustness. Refer to the *"16-bit MCU and DSC Programmer's Reference Manual*" (DS70157) for details on instruction encoding.

Note that by incrementing the PC by 2 for each program memory word, the 15 LSbs of data space addresses directly map to the 15 LSbs in the corresponding program space addresses. The remaining bits are provided by the Program Space Visibility Page register, PSVPAG<7:0>, as shown in Figure 3-6.

| **Note:** | PSV access is temporarily disabled during table reads/writes. |
|---|---|

For instructions that use PSV which are executed outside a REPEAT loop:

- The following instructions require one instruction cycle in addition to the specified execution time:
  - MAC class of instructions with data operand prefetch
  - MOV instructions
  - MOV.D instructions
- All other instructions require two instruction cycles in addition to the specified execution time of the instruction.

For instructions that use PSV which are executed inside a REPEAT loop:

- The following instances require two instruction cycles in addition to the specified execution time of the instruction:
  - Execution in the first iteration
  - Execution in the last iteration
  - Execution prior to exiting the loop due to an interrupt
  - Execution upon re-entering the loop after an interrupt is serviced
- Any other iteration of the REPEAT loop allows the instruction accessing data, using PSV, to execute in a single cycle.

**NOTES:**

## 6.0 DATA EEPROM MEMORY

**Note:** This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the "*dsPIC30F Family Reference Manual*" (DS70046). For more information on the device instruction set and programming, refer to the *"16-bit MCU and DSC Programmer's Reference Manual"* (DS70157).

The data EEPROM memory is readable and writable during normal operation over the entire VDD range. The data EEPROM memory is directly mapped in the program memory address space.

The four SFRs used to read and write the program Flash memory are used to access data EEPROM memory as well. As described in **Section 5.5 "Control Registers"**, these registers are:

- NVMCON
- NVMADR
- NVMADRU
- NVMKEY

The EEPROM data memory allows read and write of single words and 16-word blocks. When interfacing to data memory, NVMADR, in conjunction with the NVMADRU register, are used to address the EEPROM location being accessed. TBLRDL and TBLWTL instructions are used to read and write data EEPROM. The dsPIC30F devices have up to 8 Kbytes (4K words) of data EEPROM with an address range from 0x7FF000 to 0x7FFFFE.

A word write operation should be preceded by an erase of the corresponding memory location(s). The write typically requires 2 ms to complete, but the write time varies with voltage and temperature.

A program or erase operation on the data EEPROM does not stop the instruction flow. The user is responsible for waiting for the appropriate duration of time before initiating another data EEPROM write/erase operation. Attempting to read the data EEPROM while a programming or erase operation is in progress results in unspecified data.

Control bit, WR, initiates write operations similar to program Flash writes. This bit cannot be cleared, only set, in software. They are cleared in hardware at the completion of the write operation. The inability to clear the WR bit in software prevents the accidental or premature termination of a write operation.

The WREN bit, when set, allows a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a MCLR Reset or a WDT Time-out Reset during normal operation. In these situations, following Reset, the user can check the WRERR bit and rewrite the location. The address register, NVMADR, remains unchanged.

**Note:** Interrupt flag bit, NVMIF in the IFS0 register, is set when the write is complete. It must be cleared in software.

### 6.1 Reading the Data EEPROM

A TBLRD instruction reads a word at the current program word address. This example uses W0 as a pointer to data EEPROM. The result is placed in register W4 as shown in Example 6-1.

**EXAMPLE 6-1: DATA EEPROM READ**

```
MOV     #LOW_ADDR_WORD,W0  ; Init Pointer
MOV     #HIGH_ADDR_WORD,W1
MOV     W1,TBLPAG
TBLRDL  [ W0 ], W4         ; read data EEPROM
```

**EXAMPLE 6-5: DATA EEPROM BLOCK WRITE**

```
    MOV        #LOW_ADDR_WORD,W0  ; Init pointer
    MOV        #HIGH_ADDR_WORD,W1
    MOV        W1,TBLPAG
    MOV        #data1,W2          ; Get 1st data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data2,W2          ; Get 2nd data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data3,W2          ; Get 3rd data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data4,W2          ; Get 4th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data5,W2          ; Get 5th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data6,W2          ; Get 6th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data7,W2          ; Get 7th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data8,W2          ; Get 8th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data9,W2          ; Get 9th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data10,W2         ; Get 10th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data11,W2         ; Get 11th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data12,W2         ; Get 12th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data13,W2         ; Get 13th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data14,W2         ; Get 14th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data15,W2         ; Get 15th data
    TBLWTL     W2,[ W0]++         ; write data
    MOV        #data16,W2         ; Get 16th data
    TBLWTL     W2,[ W0]++         ; write data. The NVMADR captures last table access address.
    MOV        #0x400A,W0         ; Select data EEPROM for multi word op
    MOV        W0,NVMCON          ; Operate Key to allow program operation
    DISI    #5                    ; Block all interrupts with priority <7 for
                                  ; next 5 instructions
    MOV        #0x55,W0
    MOV        W0,NVMKEY          ; Write the 0x55 key
    MOV        #0xAA,W1
    MOV        W1,NVMKEY          ; Write the 0xAA key
    BSET       NVMCON,#WR         ; Start write cycle
    NOP
    NOP
```

## 6.4 Write Verify

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

## 6.5 Protection Against Spurious Write

There are conditions when the device may not want to write to the data EEPROM memory. To protect against spurious EEPROM writes, various mechanisms have been built-in. On power-up, the WREN bit is cleared; also, the Power-up Timer prevents EEPROM write.

The write initiate sequence, and the WREN bit together, help prevent an accidental write during brown-out, power glitch or software malfunction.

## 8.1 Interrupt Priority

The user-assignable interrupt priority (IP<2:0>) bits for each individual interrupt source are located in the 3 LSbs of each nibble within the IPCx register(s). Bit 3 of each nibble is not used and is read as a '0'. These bits define the priority level assigned to a particular interrupt by the user.

**Note:** The user-assignable priority levels start at 0 as the lowest priority and Level 7 as the highest priority.

Since more than one interrupt request source may be assigned to a specific user-assigned priority level, a means is provided to assign priority within a given level. This method is called "Natural Order Priority" and is final.

Natural Order Priority is determined by the position of an interrupt in the vector table, and only affects interrupt operation when multiple interrupts with the same user-assigned priority become pending at the same time.

Table 8-1 and Table 8-2 list the interrupt numbers, corresponding interrupt sources and associated vector numbers for the dsPIC30F3014 and dsPIC30F4013 devices, respectively.

**Note 1:** The natural order priority scheme has 0 as the highest priority and 53 as the lowest priority.

**2:** The natural order priority number is the same as the INT number.

The ability for the user to assign every interrupt to one of seven priority levels means that the user can assign a very high overall priority level to an interrupt with a low natural order priority. For example, the PLVD (Programmable Low-Voltage Detect) can be given a priority of 7. The INT0 (External Interrupt 0) may be assigned to priority Level 1, thus giving it a very low effective priority.

**TABLE 8-1: dsPIC30F3014 INTERRUPT VECTOR TABLE**

| INT Number | Vector Number | Interrupt Source |
|---|---|---|
| Highest Natural Order Priority | | |
| 0 | 8 | INT0 – External Interrupt 0 |
| 1 | 9 | IC1 – Input Capture 1 |
| 2 | 10 | OC1 – Output Compare 1 |
| 3 | 11 | T1 – Timer1 |
| 4 | 12 | IC2 – Input Capture 2 |
| 5 | 13 | OC2 – Output Compare 2 |
| 6 | 14 | T2 – Timer2 |
| 7 | 15 | T3 – Timer3 |
| 8 | 16 | SPI1 |
| 9 | 17 | U1RX – UART1 Receiver |
| 10 | 18 | U1TX – UART1 Transmitter |
| 11 | 19 | ADC – ADC Convert Done |
| 12 | 20 | NVM – NVM Write Complete |
| 13 | 21 | SI2C – $I^2C$™ Slave Interrupt |
| 14 | 22 | MI2C – $I^2C$ Master Interrupt |
| 15 | 23 | Input Change Interrupt |
| 16 | 24 | INT1 – External Interrupt 1 |
| 17-22 | 25-30 | Reserved |
| 23 | 31 | INT2 – External Interrupt 2 |
| 24 | 32 | U2RX – UART2 Receiver |
| 25 | 33 | U2TX – UART2 Transmitter |
| 26 | 34 | Reserved |
| 27 | 35 | C1 – Combined IRQ for CAN1 |
| 28-41 | 36-49 | Reserved |
| 42 | 50 | LVD – Low-Voltage Detect |
| 43-53 | 51-61 | Reserved |
| Lowest Natural Order Priority | | |

## 11.0 TIMER4/5 MODULE

**Note:** This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046).

This section describes the second 32-bit general purpose timer module (Timer4/5) and associated operational modes. Figure 11-1 depicts the simplified block diagram of the 32-bit Timer4/5 module. Figure 11-2 and Figure 11-3 show Timer4/5 configured as two independent 16-bit timers, Timer4 and Timer5, respectively.

The Timer4/5 module is similar in operation to the Timer2/3 module. However, there are some differences:

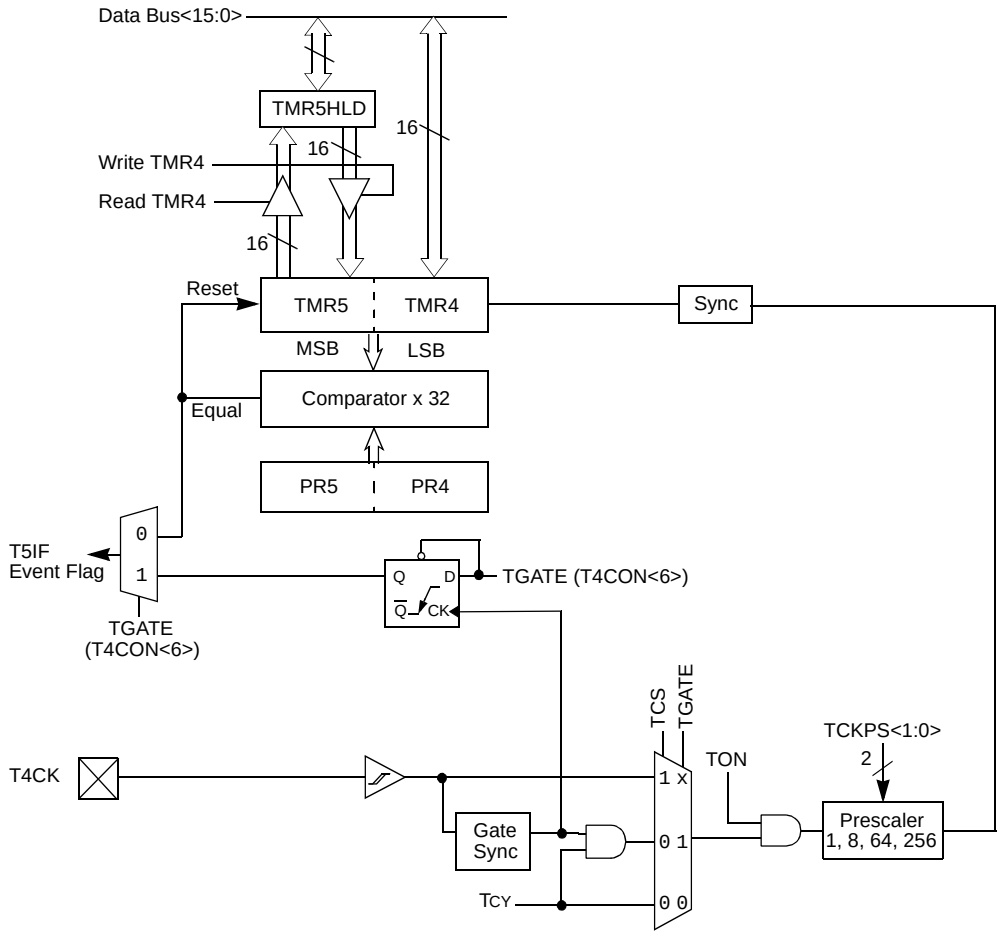
- The Timer4/5 module does not support the ADC event trigger feature
- Timer4/5 can not be utilized by other peripheral modules, such as input capture and output compare

The operating modes of the Timer4/5 module are determined by setting the appropriate bit(s) in the 16-bit T4CON and T5CON SFRs.

For 32-bit timer/counter operation, Timer4 is the lsw and Timer5 is the msw of the 32-bit timer.

**Note:** For 32-bit timer operation, T5CON control bits are ignored. Only T4CON control bits are used for setup and control. Timer4 clock and gate inputs are utilized for the 32-bit timer module but an interrupt is generated with the Timer5 Interrupt Flag (T5IF) and the interrupt is enabled with the Timer5 interrupt enable bit (T5IE).

**FIGURE 11-1: 32-BIT TIMER4/5 BLOCK DIAGRAM**




**Note:** Timer Configuration bit, T32 (T4CON<3>), must be set to '1' for a 32-bit timer/counter operation. All control bits are respective to the T4CON register.

## 14.0 I²C™ MODULE

**Note:** This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the '*dsPIC30F Family Reference Manual*' (DS70046).

The Inter-Integrated Circuit (I²C™) module provides complete hardware support for both Slave and Multi-Master modes of the I²C serial communication standard with a 16-bit interface.

This module offers the following key features:

- I²C interface supporting both master and slave operation.
- I²C Slave mode supports 7-bit and 10-bit addressing.
- I²C Master mode supports 7-bit and 10-bit addressing.
- I²C port allows bidirectional transfers between master and slaves.
- Serial clock synchronization for I²C port can be used as a handshake mechanism to suspend and resume serial transfer (SCLREL control).
- I²C supports multi-master operation; detects bus collision and arbitrates accordingly.

### 14.1 Operating Function Description

The hardware fully implements all the master and slave functions of the I²C Standard and Fast mode specifications, as well as 7 and 10-bit addressing.

Thus, the I²C module can operate either as a slave or a master on an I²C bus.

#### 14.1.1 VARIOUS I²C MODES

The following types of I²C operation are supported:

- I²C slave operation with 7-bit addressing
- I²C slave operation with 10-bit addressing
- I²C master operation with 7-bit or 10-bit addressing

See the I²C programmer's model in Figure 14-1.

#### 14.1.2 PIN CONFIGURATION IN I²C MODE

I²C has a 2-pin interface: the SCL pin is clock and the SDA pin is data.

#### 14.1.3 I²C REGISTERS

I2CCON and I2CSTAT are control and status registers, respectively. The I2CCON register is readable and writable. The lower 6 bits of I2CSTAT are read-only. The remaining bits of the I2CSTAT are read/write.

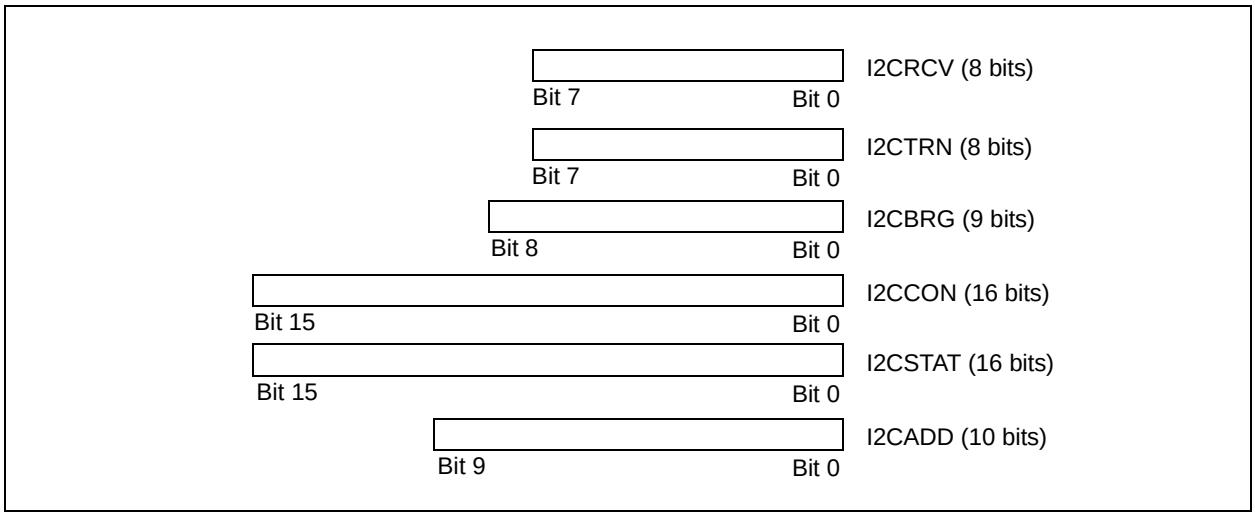
I2CRSR is the shift register used for shifting data, whereas I2CRCV is the buffer register to which data bytes are written, or from which data bytes are read. I2CRCV is the receive buffer as shown in Figure 14-1. I2CTRN is the transmit register to which bytes are written during a transmit operation, as shown in Figure 14-2.

The I2CADD register holds the slave address. A status bit, ADD10, indicates 10-Bit Addressing mode. The I2CBRG acts as the Baud Rate Generator reload value.

In receive operations, I2CRSR and I2CRCV together form a double-buffered receiver. When I2CRSR receives a complete byte, it is transferred to I2CRCV and an interrupt pulse is generated. During transmission, the I2CTRN is not double-buffered.

**Note:** Following a Restart condition in 10-bit mode, the user only needs to match the first 7-bit address.

**FIGURE 14-1: PROGRAMMER'S MODEL**



## 14.2 I²C Module Addresses

The I2CADD register contains the Slave mode addresses. The register is a 10-bit register.

If the A10M bit (I2CCON<10>) is '0', the address is interpreted by the module as a 7-bit address. When an address is received, it is compared to the 7 LSbs of the I2CADD register.

If the A10M bit is '1', the address is assumed to be a 10-bit address. When an address is received, it is compared with the binary value, '11110 A9 A8' (where A9 and A8 are two Most Significant bits of I2CADD). If that value matches, the next address is compared with the Least Significant 8 bits of I2CADD, as specified in the 10-bit addressing protocol.

**TABLE 14-1: 7-BIT I²C™ SLAVE ADDRESSES SUPPORTED BY dsPIC30F**

| | |
|---|---|
| 0x00 | General Call Address or Start Byte |
| 0x01-0x03 | Reserved |
| 0x04-0x07 | HS mode Master Codes |
| 0x08-0x77 | Valid 7-Bit Addresses |
| 0x78-0x7b | Valid 10-Bit Addresses (lower 7 bits) |
| 0x7c-0x7f | Reserved |

## 14.3 I²C 7-Bit Slave Mode Operation

Once enabled (I2CEN = 1), the slave module waits for a Start bit to occur (i.e., the I²C module is 'Idle'). Following the detection of a Start bit, 8 bits are shifted into I2CRSR, and the address is compared against I2CADD. In 7-bit mode (A10M = 0), bits I2CADD<6:0> are compared against I2CRSR<7:1> and I2CRSR<0> is the R_W bit. All incoming bits are sampled on the rising edge of SCL.

If an address match occurs, an Acknowledgement is sent and the Slave Event Interrupt Flag (SI2CIF) is set on the falling edge of the ninth (ACK) bit. The address match does not affect the contents of the I2CRCV buffer or the RBF bit.

### 14.3.1 SLAVE TRANSMISSION

If the R_W bit received is a '1', the serial port goes into Transmit mode. It sends an ACK on the ninth bit and then holds SCL to '0' until the CPU responds by writing to I2CTRN. SCL is released by setting the SCLREL bit, and 8 bits of data are shifted out. Data bits are shifted out on the falling edge of SCL, such that SDA is valid during SCL high. The interrupt pulse is sent on the falling edge of the ninth clock pulse, regardless of the status of the ACK received from the master.

### 14.3.2 SLAVE RECEPTION

If the R_W bit received is a '0' during an address match, then Receive mode is initiated. Incoming bits are sampled on the rising edge of SCL. After 8 bits are received, if I2CRCV is not full or I2COV is not set, I2CRSR is transferred to I2CRCV. ACK is sent on the ninth clock.

If the RBF flag is set, indicating that I2CRCV is still holding data from a previous operation (RBF = 1), then ACK is not sent; however, the interrupt pulse is generated. In the case of an overflow, the contents of the I2CRSR are not loaded into the I2CRCV.

| | |
|---|---|
| **Note:** | The I2CRCV is loaded if the I2COV bit = 1 and the RBF flag = 0. In this case, a read of the I2CRCV was performed but the user did not clear the state of the I2COV bit before the next receive occurred. The acknowledgement is not sent (ACK = 1) and the I2CRCV is updated. |

## 14.4 I²C 10-Bit Slave Mode Operation

In 10-bit mode, the basic receive and transmit operations are the same as in the 7-bit mode. However, the criteria for address match is more complex.

The I²C specification dictates that a slave must be addressed for a write operation with two address bytes following a Start bit.

The A10M bit is a control bit that signifies that the address in I2CADD is a 10-bit address rather than a 7-bit address. The address detection protocol for the first byte of a message address is identical for 7-bit and 10-bit messages, but the bits being compared are different.

I2CADD holds the entire 10-bit address. Upon receiving an address following a Start bit, I2CRSR <7:3> is compared against a literal '11110' (the default 10-bit address) and I2CRSR<2:1> are compared against I2CADD<9:8>. If a match occurs and if R_W = 0, the interrupt pulse is sent. The ADD10 bit is cleared to indicate a partial address match. If a match fails or R_W = 1, the ADD10 bit is cleared and the module returns to the Idle state.

The low byte of the address is then received and compared with I2CADD<7:0>. If an address match occurs, the interrupt pulse is generated and the ADD10 bit is set, indicating a complete 10-bit address match. If an address match did not occur, the ADD10 bit is cleared and the module returns to the Idle state.

### 14.4.1 10-BIT MODE SLAVE TRANSMISSION

Once a slave is addressed in this fashion with the full 10-bit address (we refer to this state as "PRIOR_ADDR_MATCH"), the master can begin sending data bytes for a slave reception operation.

## 15.0 SPI MODULE

| | |
|---|---|
| **Note:** | This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the "*dsPIC30F Family Reference Manual*" (DS70046). |

The Serial Peripheral Interface (SPI) module is a synchronous serial interface. It is useful for communicating with other peripheral devices, such as EEPROMs, shift registers, display drivers and A/D converters, or other microcontrollers. It is compatible with Motorola's SPI and SIOP interfaces. The dsPIC30F3014 and dsPIC30F4013 devices feature one SPI module, SPI1.

## 15.1 Operating Function Description

Each SPI module consists of a 16-bit shift register, SPIxSR (where x = 1 or 2), used for shifting data in and out, and a buffer register, SPIxBUF. A control register, SPIxCON, configures the module. Additionally, a status register, SPIxSTAT, indicates various status conditions.

The serial interface consists of 4 pins: SDIx (Serial Data Input), SDOx (Serial Data Output), SCKx (Shift Clock Input or Output), and $\overline{SSx}$ (Active-Low Slave Select).

In Master mode operation, SCKx is a clock output but in Slave mode, it is a clock input.

A series of eight (8) or sixteen (16) clock pulses shift out bits from the SPIxSR to SDOx pin and simultaneously shift in data from SDIx pin. An interrupt is generated when the transfer is complete and the corresponding interrupt flag bit (SPI1IF or SPI2IF) is set. This interrupt can be disabled through an interrupt enable bit (SPI1IE or SPI2IE).

The receive operation is double-buffered. When a complete byte is received, it is transferred from SPIxSR to SPIxBUF.

If the receive buffer is full when new data is being transferred from SPIxSR to SPIxBUF, the module sets the SPIROV bit, indicating an overflow condition. The transfer of the data from SPIxSR to SPIxBUF is not completed and new data is lost. The module does not respond to SCL transitions while SPIROV is '1', effectively disabling the module until SPIxBUF is read by user software.

Transmit writes are also double-buffered. The user writes to SPIxBUF. When the master or slave transfer is completed, the contents of the shift register (SPIxSR) are moved to the receive buffer. If any transmit data has been written to the buffer register, the contents of the transmit buffer are moved to SPIxSR. The received data is thus placed in SPIxBUF and the transmit data in SPIxSR is ready for the next transfer.

| | |
|---|---|
| **Note:** | Both the transmit buffer (SPIxTXB) and the receive buffer (SPIxRXB) are mapped to the same register address, SPIxBUF. |

In Master mode, the clock is generated by prescaling the system clock. Data is transmitted as soon as a value is written to SPIxBUF. The interrupt is generated at the middle of the transfer of the last bit.

In Slave mode, data is transmitted and received as external clock pulses appear on SCKx. Again, the interrupt is generated when the last bit is latched. If $\overline{SSx}$ control is enabled, then transmission and reception are enabled only when $\overline{SSx}$ = low. The SDOx output is disabled in $\overline{SSx}$ mode with $\overline{SSx}$ high.

The clock provided to the module is (FOSC/4). This clock is then prescaled by the primary (PPRE<1:0>) and the secondary (SPRE<2:0>) prescale factors. The CKE bit determines whether transmit occurs on transition from active clock state to Idle clock state, or vice versa. The CKP bit selects the Idle state (high or low) for the clock.

### 15.1.1 WORD AND BYTE COMMUNICATION

A control bit, MODE16 (SPIxCON<10>), allows the module to communicate in either 16-bit or 8-bit mode. 16-bit operation is identical to 8-bit operation except that the number of bits transmitted is 16 instead of 8.

The user software must disable the module prior to changing the MODE16 bit. The SPI module is reset when the MODE16 bit is changed by the user.

A basic difference between 8-bit and 16-bit operation is that the data is transmitted out of bit 7 of the SPIxSR for 8-bit operation, and data is transmitted out of bit 15 of the SPIxSR for 16-bit operation. In both modes, data is shifted into bit 0 of the SPIxSR.

### 15.1.2 SDOx DISABLE

A control bit, DISSDO, is provided to the SPIxCON register to allow the SDOx output to be disabled. This allows the SPI module to be connected in an input-only configuration. SDOx can also be used for general purpose I/O.

### 16.5.2 FRAMING ERROR (FERR)

The FERR bit (UxSTA<2>) is set if a '0' is detected instead of a Stop bit. If two Stop bits are selected, both Stop bits must be '1'; otherwise, FERR is set. The read-only FERR bit is buffered along with the received data; it is cleared on any Reset.

### 16.5.3 PARITY ERROR (PERR)

The PERR bit (UxSTA<3>) is set if the parity of the received word is incorrect. This error bit is applicable only if a Parity mode (odd or even) is selected. The read-only PERR bit is buffered along with the received data bytes; it is cleared on any Reset.

### 16.5.4 IDLE STATUS

When the receiver is active (i.e., between the initial detection of the Start bit and the completion of the Stop bit), the RIDLE bit (UxSTA<4>) is '0'. Between the completion of the Stop bit and detection of the next Start bit, the RIDLE bit is '1', indicating that the UART is Idle.

### 16.5.5 RECEIVE BREAK

The receiver counts and expects a certain number of bit times based on the values programmed in the PDSEL (UxMODE<2:1>) and STSEL (UxMODE<0>) bits.

If the break is longer than 13 bit times, the reception is considered complete after the number of bit times specified by PDSEL and STSEL. The URXDA bit is set, FERR is set, zeros are loaded into the receive FIFO, interrupts are generated if appropriate, and the RIDLE bit is set.

When the module receives a long Break signal and the receiver has detected the Start bit, the data bits and the invalid Stop bit (which sets the FERR), the receiver must wait for a valid Stop bit before looking for the next Start bit. It cannot assume that the Break condition on the line is the next Start bit.

Break is regarded as a character containing all '0's with the FERR bit set. The Break character is loaded into the buffer. No further reception can occur until a Stop bit is received. Note that RIDLE goes high when the Stop bit has not yet been received.

## 16.6 Address Detect Mode

Setting the ADDEN bit (UxSTA<5>) enables this special mode in which a 9th bit (URX8) value of '1' identifies the received word as an address, rather than data. This mode is only applicable for 9-bit data communication. The URXISEL control bit does not have any impact on interrupt generation in this mode since an interrupt (if enabled) is generated every time the received word has the 9th bit set.

## 16.7 Loopback Mode

Setting the LPBACK bit enables this special mode in which the UxTX pin is internally connected to the UxRX pin. When configured for the Loopback mode, the UxRX pin is disconnected from the internal UART receive logic. However, the UxTX pin still functions as in a normal operation.

To select this mode:

a) Configure UART for desired mode of operation.
b) Set LPBACK = 1 to enable Loopback mode.
c) Enable transmission as defined in **Section 16.3 "Transmitting Data"**.

## 16.8 Baud Rate Generator

The UART has a 16-bit Baud Rate Generator to allow maximum flexibility in baud rate generation. The Baud Rate Generator register (UxBRG) is readable and writable. The baud rate is computed as follows:

BRG = 16-bit value held in UxBRG register (0 through 65535)

$F_{CY}$ = Instruction Clock Rate ($1/T_{CY}$)

The Baud Rate is given by Equation 16-1.

**EQUATION 16-1: BAUD RATE**

$$\text{Baud Rate} = F_{CY}/(16*(BRG+1))$$

Therefore, the maximum baud rate possible is:

$F_{CY}/16$ (if BRG = 0),

and the minimum baud rate possible is:

$F_{CY}/(16 * 65536)$.

With a full 16-bit Baud Rate Generator at 30 MIPS operation, the minimum baud rate achievable is 28.5 bps.

## 17.0 CAN MODULE

**Note:** This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the "*dsPIC30F Family Reference Manual*" (DS70046).

### 17.1 Overview

The Controller Area Network (CAN) module is a serial interface, useful for communicating with other CAN modules or microcontroller devices. This interface/ protocol was designed to allow communications within noisy environments.

The CAN module is a communication controller implementing the CAN 2.0 A/B protocol, as defined in the BOSCH specification. The module supports CAN 1.2, CAN 2.0A, CAN 2.0B Passive, and CAN 2.0B Active versions of the protocol. The module implementation is a full CAN system. The CAN specification is not covered within this data sheet. The reader may refer to the BOSCH CAN specification for further details.

The module features are as follows:

- Implementation of the CAN protocol CAN 1.2, CAN 2.0A and CAN 2.0B
- Standard and extended data frames
- 0-8 bytes data length
- Programmable bit rate up to 1 Mbit/sec
- Support for remote frames
- Double-buffered receiver with two prioritized received message storage buffers (each buffer may contain up to 8 bytes of data)
- 6 full (standard/extended identifier), acceptance filters, 2 associated with the high-priority receive buffer and 4 associated with the low-priority receive buffer
- 2 full, acceptance filter masks, one each associated with the high and low-priority receive buffers
- Three transmit buffers with application specified prioritization and abort capability (each buffer may contain up to 8 bytes of data)
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loopback mode supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to input capture module (IC2, for both CAN1 and CAN2) for time-stamping and network synchronization
- Low-power Sleep and Idle mode

The CAN bus module consists of a protocol engine and message buffering/control. The CAN protocol engine handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate data registers. Status and errors can be checked by reading the appropriate registers. Any message detected on the CAN bus is checked for errors and then matched against filters to see if it should be received and stored in one of the receive registers.

### 17.2 Frame Types

The CAN module transmits various types of frames which include data messages or remote transmission requests, initiated by the user, as other frames that are automatically generated for control purposes. The following frame types are supported:

- Standard Data Frame:

  A standard data frame is generated by a node when the node wishes to transmit data. It includes an 11-bit Standard Identifier (SID) but not an 18-bit Extended Identifier (EID).

- Extended Data Frame:

  An extended data frame is similar to a standard data frame but includes an extended identifier as well.

- Remote Frame:

  It is possible for a destination node to request the data from the source. For this purpose, the destination node sends a remote frame with an identifier that matches the identifier of the required data frame. The appropriate data source node then sends a data frame as a response to this remote request.

- Error Frame:

  An error frame is generated by any node that detects a bus error. An error frame consists of 2 fields: an error flag field and an error delimiter field.

- Overload Frame:

  An overload frame can be generated by a node as a result of 2 conditions. First, the node detects a dominant bit during interframe space which is an illegal condition. Second, due to internal conditions, the node is not yet able to start reception of the next message. A node may generate a maximum of 2 sequential overload frames to delay the start of the next message.

- Interframe Space:

  Interframe space separates a proceeding frame (of whatever type) from a following data or remote frame.

## 17.4 Message Reception

### 17.4.1 RECEIVE BUFFERS

The CAN bus module has 3 receive buffers. However, one of the receive buffers is always committed to monitoring the bus for incoming messages. This buffer is called the Message Assembly Buffer (MAB). So there are 2 receive buffers visible, denoted as RXB0 and RXB1, that can essentially instantaneously receive a complete message from the protocol engine.

All messages are assembled by the MAB and are transferred to the RXBn buffers only if the acceptance filter criterion are met. When a message is received, the RXnIF flag (CiINRF<0> or CiINRF<1>) is set. This bit can only be set by the module when a message is received. The bit is cleared by the CPU when it has completed processing the message in the buffer. If the RXnIE bit (CiINTE<0> or CiINTE<1>) is set, an interrupt is generated when a message is received.

RXF0 and RXF1 filters with RXM0 mask are associated with RXB0. The filters RXF2, RXF3, RXF4 and RXF5, and the mask RXM1 are associated with RXB1.

### 17.4.2 MESSAGE ACCEPTANCE FILTERS

The message acceptance filters and masks are used to determine if a message in the message assembly buffer should be loaded into either of the receive buffers. Once a valid message has been received into the Message Assembly Buffer (MAB), the identifier fields of the message are compared to the filter values. If there is a match, that message is loaded into the appropriate receive buffer.

The acceptance filter looks at incoming messages for the RXIDE bit (CiRXnSID<0>) to determine how to compare the identifiers. If the RXIDE bit is clear, the message is a standard frame and only filters with the EXIDE bit (CiRXFnSID<0>) clear are compared. If the RXIDE bit is set, the message is an extended frame and only filters with the EXIDE bit set are compared.

### 17.4.3 MESSAGE ACCEPTANCE FILTER MASKS

The mask bits essentially determine which bits to apply the filter to. If any mask bit is set to a zero, that bit is automatically accepted regardless of the filter bit. There are two programmable acceptance filter masks associated with the receive buffers, one for each buffer.

### 17.4.4 RECEIVE OVERRUN

An overrun condition occurs when the Message Assembly Buffer (MAB) has assembled a valid received message, the message is accepted through the acceptance filters, and when the receive buffer associated with the filter has not been designated as clear of the previous message.

The overrun error flag, RXnOVR (CiINTF<15> or CiINTF<14>), and the ERRIF bit (CiINTF<5>) are set and the message in the MAB is discarded.

If the DBEN bit is clear, RXB1 and RXB0 operate independently. When this is the case, a message intended for RXB0 is not diverted into RXB1 if RXB0 contains an unread message, and the RX0OVR bit is set.

If the DBEN bit is set, the overrun for RXB0 is handled differently. If a valid message is received for RXB0 and RXFUL = 1 it indicates that RXB0 is full and RXFUL = 0 indicates that RXB1 is empty, the message for RXB0 is loaded into RXB1. An overrun error is not generated for RXB0. If a valid message is received for RXB0 and RXFUL = 1, indicates that both RXB0 and RXB1 are full, the message is lost and an overrun is indicated for RXB1.

### 17.4.5 RECEIVE ERRORS

The CAN module detects the following receive errors:

- Cyclic Redundancy Check (CRC) error
- Bit Stuffing error
- Invalid Message Receive Error

The receive error counter is incremented by one in case one of these errors occur. The RXWAR bit (CiINTF<9>) indicates that the receive error counter has reached the CPU warning limit of 96 and an interrupt is generated.

### 17.4.6 RECEIVE INTERRUPTS

Receive interrupts can be divided into 3 major groups, each including various conditions that generate interrupts:

- Receive Interrupt:

  A message has been successfully received and loaded into one of the receive buffers. This interrupt is activated immediately after receiving the End-of-Frame (EOF) field. Reading the RXnIF flag indicates which receive buffer caused the interrupt.

- Wake-up Interrupt:

  The CAN module has woken up from Disable mode or the device has woken up from Sleep mode.

## 20.0 SYSTEM INTEGRATION

**Note:** This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the "*dsPIC30F Family Reference Manual*" (DS70046). For more information on the device instruction set and programming, refer to the *"16-bit MCU and DSC Programmer's Reference Manual"* (DS70157).

There are several features intended to maximize system reliability, minimize cost through elimination of external components, provide power-saving operating modes and offer code protection:

- Oscillator Selection
- Reset
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Programmable Brown-out Reset (BOR)
- Watchdog Timer (WDT)
- Low-Voltage Detect
- Power-Saving modes (Sleep and Idle)
- Code Protection
- Unit ID Locations
- In-Circuit Serial Programming (ICSP)

dsPIC30F devices have a Watchdog Timer which is permanently enabled via the Configuration bits or can be software controlled. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in Reset until the crystal oscillator is stable. The other is the Power-up Timer (PWRT) which provides a delay on power-up only, designed to keep the part in Reset while the power supply stabilizes. With these two timers on-chip, most applications need no external Reset circuitry.

Sleep mode is designed to offer a very low-current Power-Down mode. The user can wake-up from Sleep through external Reset, Watchdog Timer wake-up, or through an interrupt. Several oscillator options are also made available to allow the part to fit a wide variety of applications. In the Idle mode, the clock sources are still active but the CPU is shut off. The RC oscillator option saves system cost while the LP crystal option saves power.

### 20.1 Oscillator System Overview

The dsPIC30F oscillator system has the following modules and features:

- Various external and internal oscillator options as clock sources
- An on-chip PLL to boost internal operating frequency
- A clock switching mechanism between various clock sources
- Programmable clock postscaler for system power savings
- A Fail-Safe Clock Monitor (FSCM) that detects clock failure and takes fail-safe measures
- Clock Control register (OSCCON)
- Configuration bits for main oscillator selection

Configuration bits determine the clock source upon Power-on Reset (POR) and Brown-out Reset (BOR). Thereafter, the clock source can be changed between permissible clock sources. The OSCCON register controls the clock switching and reflects system clock related status bits.

Table 20-1 provides a summary of the dsPIC30F oscillator operating modes. A simplified diagram of the oscillator system is shown in Figure 20-1.

## 20.5 Watchdog Timer (WDT)

### 20.5.1 WATCHDOG TIMER OPERATION

The primary function of the Watchdog Timer (WDT) is to reset the processor in the event of a software malfunction. The WDT is a free-running timer that runs off an on-chip RC oscillator, requiring no external component. Therefore, the WDT timer continues to operate even if the main processor clock (e.g., the crystal oscillator) fails.

### 20.5.2 ENABLING AND DISABLING THE WDT

The Watchdog Timer can be “Enabled” or “Disabled” only through a Configuration bit (FWDTEN) in the Configuration register, FWDT.

Setting FWDTEN = 1 enables the Watchdog Timer. The enabling is done when programming the device. By default, after chip erase, FWDTEN bit = 1. Any device programmer capable of programming dsPIC30F devices allows programming of this and other Configuration bits.

If enabled, the WDT increments until it overflows or “times out”. A WDT time-out forces a device Reset (except during Sleep). To prevent a WDT time-out, the user must clear the Watchdog Timer using a CLRWDT instruction.

If a WDT times out during Sleep, the device wakes up. The WDTO bit in the RCON register is cleared to indicate a wake-up resulting from a WDT time-out.

Setting FWDTEN = 0 allows user software to enable/disable the Watchdog Timer via the SWDTEN (RCON<5>) control bit.

## 20.6 Low-Voltage Detect

The Low-Voltage Detect (LVD) module is used to detect when the $V_{DD}$ of the device drops below a threshold value, $V_{LVD}$, which is determined by the LVDL<3:0> bits (RCON<11:8>) and is thus user programmable. The internal voltage reference circuitry requires a nominal amount of time to stabilize, and the BGST bit (RCON<13>) indicates when the voltage reference has stabilized.

In some devices, the LVD threshold voltage may be applied externally on the LVDIN pin.

The LVD module is enabled by setting the LVDEN bit (RCON<12>).

## 20.7 Power-Saving Modes

There are two power-saving states that can be entered through the execution of a special instruction, PWRSAV; these are Sleep and Idle.

The format of the PWRSAV instruction is as follows:

PWRSAV <parameter>, where ‘parameter’ defines Idle or Sleep mode.

### 20.7.1 SLEEP MODE

In Sleep mode, the clock to the CPU and peripherals is shut down. If an on-chip oscillator is being used, it is shut down.

The Fail-Safe Clock Monitor is not functional during Sleep since there is no clock to monitor. However, the LPRC clock remains active if WDT is operational during Sleep.

The brown-out protection circuit and the Low-Voltage Detect (LVD) circuit, if enabled, remains functional during Sleep.

The processor wakes up from Sleep if at least one of the following conditions has occurred:

- any interrupt that is individually enabled and meets the required priority level
- any Reset (POR, BOR and $\overline{\text{MCLR}}$)
- WDT time-out

On waking up from Sleep mode, the processor restarts the same clock that was active prior to entry into Sleep mode. When clock switching is enabled, bits, COSC<2:0>, determine the oscillator source to be used on wake-up. If clock switch is disabled, then there is only one system clock.

| **Note:** | If a POR or BOR occurred, the selection of the oscillator is based on the FOS<2:0> and FPR<4:0> Configuration bits. |
|---|---|

If the clock source is an oscillator, the clock to the device is held off until OST times out (indicating a stable oscillator). If PLL is used, the system clock is held off until LOCK = 1 (indicating that the PLL is stable). In either case, $T_{POR}$, $T_{LOCK}$ and $T_{PWRT}$ delays are applied.

If EC, FRC, LPRC or ERC oscillators are used, then a delay of $T_{POR}$ (~ 10 μs) is applied. This is the smallest delay possible on wake-up from Sleep.

Moreover, if the LP oscillator was active during Sleep and LP is the oscillator used on wake-up, then the start-up delay is equal to $T_{POR}$. PWRT delay and OST timer delay are not applied. In order to have the smallest possible start-up delay when waking up from Sleep, one of these faster wake-up options should be selected before entering Sleep.

**TABLE 21-2: INSTRUCTION SET OVERVIEW (CONTINUED)**

| Base Instr # | Assembly Mnemonic | Assembly Syntax | Description | # of Words | # of Cycles | Status Flags Affected |
|---|---|---|---|---|---|---|
| 48 | MPY | MPY Wm*Wn,Acc,Wx,Wxd,Wy,Wyd | Multiply Wm by Wn to Accumulator | 1 | 1 | OA,OB,OAB, SA,SB,SAB |
| | | MPY Wm*Wm,Acc,Wx,Wxd,Wy,Wyd | Square Wm to Accumulator | 1 | 1 | OA,OB,OAB, SA,SB,SAB |
| 49 | MPY.N | MPY.N Wm*Wn,Acc,Wx,Wxd,Wy,Wyd | -(Multiply Wm by Wn) to Accumulator | 1 | 1 | None |
| 50 | MSC | MSC Wm*Wm,Acc,Wx,Wxd,Wy,Wyd, AWB | Multiply and Subtract from Accumulator | 1 | 1 | OA,OB,OAB, SA,SB,SAB |
| 51 | MUL | MUL.SS Wb,Ws,Wnd | {Wnd+1, Wnd} = Signed(Wb) * Signed(Ws) | 1 | 1 | None |
| | | MUL.SU Wb,Ws,Wnd | {Wnd+1, Wnd} = Signed(Wb) * Unsigned(Ws) | 1 | 1 | None |
| | | MUL.US Wb,Ws,Wnd | {Wnd+1, Wnd} = Unsigned(Wb) * Signed(Ws) | 1 | 1 | None |
| | | MUL.UU Wb,Ws,Wnd | {Wnd+1, Wnd} = Unsigned(Wb) * Unsigned(Ws) | 1 | 1 | None |
| | | MUL.SU Wb,#lit5,Wnd | {Wnd+1, Wnd} = Signed(Wb) * Unsigned(lit5) | 1 | 1 | None |
| | | MUL.UU Wb,#lit5,Wnd | {Wnd+1, Wnd} = Unsigned(Wb) * Unsigned(lit5) | 1 | 1 | None |
| | | MUL f | W3:W2 = f * WREG | 1 | 1 | None |
| 52 | NEG | NEG Acc | Negate Accumulator | 1 | 1 | OA,OB,OAB, SA,SB,SAB |
| | | NEG f | f = $\bar{f}$ + 1 | 1 | 1 | C,DC,N,OV,Z |
| | | NEG f,WREG | WREG = $\bar{f}$ + 1 | 1 | 1 | C,DC,N,OV,Z |
| | | NEG Ws,Wd | Wd = $\overline{Ws}$ + 1 | 1 | 1 | C,DC,N,OV,Z |
| 53 | NOP | NOP | No Operation | 1 | 1 | None |
| | | NOPR | No Operation | 1 | 1 | None |
| 54 | POP | POP f | Pop f from Top-of-Stack (TOS) | 1 | 1 | None |
| | | POP Wdo | Pop from Top-of-Stack (TOS) to Wdo | 1 | 1 | None |
| | | POP.D Wnd | Pop from Top-of-Stack (TOS) to W(nd):W(nd+1) | 1 | 2 | None |
| | | POP.S | Pop Shadow Registers | 1 | 1 | All |
| 55 | PUSH | PUSH f | Push f to Top-of-Stack (TOS) | 1 | 1 | None |
| | | PUSH Wso | Push Wso to Top-of-Stack (TOS) | 1 | 1 | None |
| | | PUSH.D Wns | Push W(ns):W(ns+1) to Top-of-Stack (TOS) | 1 | 2 | None |
| | | PUSH.S | Push Shadow Registers | 1 | 1 | None |
| 56 | PWRSAV | PWRSAV #lit1 | Go into Sleep or Idle mode | 1 | 1 | WDTO, Sleep |
| 57 | RCALL | RCALL Expr | Relative Call | 1 | 2 | None |
| | | RCALL Wn | Computed Call | 1 | 2 | None |
| 58 | REPEAT | REPEAT #lit14 | Repeat Next Instruction lit14+1 Times | 1 | 1 | None |
| | | REPEAT Wn | Repeat Next Instruction (Wn)+1 Times | 1 | 1 | None |
| 59 | RESET | RESET | Software Device Reset | 1 | 1 | None |
| 60 | RETFIE | RETFIE | Return from Interrupt | 1 | 3 (2) | None |
| 61 | RETLW | RETLW #lit10,Wn | Return with Literal in Wn | 1 | 3 (2) | None |
| 62 | RETURN | RETURN | Return from Subroutine | 1 | 3 (2) | None |
| 63 | RLC | RLC f | f = Rotate Left through Carry f | 1 | 1 | C,N,Z |
| | | RLC f,WREG | WREG = Rotate Left through Carry f | 1 | 1 | C,N,Z |
| | | RLC Ws,Wd | Wd = Rotate Left through Carry Ws | 1 | 1 | C,N,Z |
| 64 | RLNC | RLNC f | f = Rotate Left (No Carry) f | 1 | 1 | N,Z |
| | | RLNC f,WREG | WREG = Rotate Left (No Carry) f | 1 | 1 | N,Z |
| | | RLNC Ws,Wd | Wd = Rotate Left (No Carry) Ws | 1 | 1 | N,Z |
| 65 | RRC | RRC f | f = Rotate Right through Carry f | 1 | 1 | C,N,Z |
| | | RRC f,WREG | WREG = Rotate Right through Carry f | 1 | 1 | C,N,Z |
| | | RRC Ws,Wd | Wd = Rotate Right through Carry Ws | 1 | 1 | C,N,Z |

## 22.2 MPLAB C Compilers for Various Device Families

The MPLAB C Compiler code development systems are complete ANSI C compilers for Microchip's PIC18, PIC24 and PIC32 families of microcontrollers and the dsPIC30 and dsPIC33 families of digital signal controllers. These compilers provide powerful integration capabilities, superior code optimization and ease of use.

For easy source level debugging, the compilers provide symbol information that is optimized to the MPLAB IDE debugger.

## 22.3 HI-TECH C for Various Device Families

The HI-TECH C Compiler code development systems are complete ANSI C compilers for Microchip's PIC family of microcontrollers and the dsPIC family of digital signal controllers. These compilers provide powerful integration capabilities, omniscient code generation and ease of use.

For easy source level debugging, the compilers provide symbol information that is optimized to the MPLAB IDE debugger.

The compilers include a macro assembler, linker, preprocessor, and one-step driver, and can run on multiple platforms.

## 22.4 MPASM Assembler

The MPASM Assembler is a full-featured, universal macro assembler for PIC10/12/16/18 MCUs.

The MPASM Assembler generates relocatable object files for the MPLINK Object Linker, Intel® standard HEX files, MAP files to detail memory usage and symbol reference, absolute LST files that contain source lines and generated machine code and COFF files for debugging.

The MPASM Assembler features include:

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

## 22.5 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK Object Linker combines relocatable objects created by the MPASM Assembler and the MPLAB C18 C Compiler. 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

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

MPLAB Assembler produces relocatable machine code from symbolic assembly language for PIC24, PIC32 and dsPIC devices. MPLAB C 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 IDE compatibility

## 22.11 PICkit 2 Development Programmer/Debugger and PICkit 2 Debug Express

The PICkit™ 2 Development Programmer/Debugger is a low-cost development tool with an easy to use interface for programming and debugging Microchip's Flash families of microcontrollers. The full featured Windows® programming interface supports baseline (PIC10F, PIC12F5xx, PIC16F5xx), midrange (PIC12F6xx, PIC16F), PIC18F, PIC24, dsPIC30, dsPIC33, and PIC32 families of 8-bit, 16-bit, and 32-bit microcontrollers, and many Microchip Serial EEPROM products. With Microchip's powerful MPLAB Integrated Development Environment (IDE) the PICkit™ 2 enables in-circuit debugging on most PIC® microcontrollers. In-Circuit-Debugging runs, halts and single steps the program while the PIC microcontroller is embedded in the application. When halted at a breakpoint, the file registers can be examined and modified.

The PICkit 2 Debug Express include the PICkit 2, demo board and microcontroller, hookup cables and CDROM with user's guide, lessons, tutorial, compiler and MPLAB IDE software.

## 22.12 MPLAB PM3 Device Programmer

The MPLAB PM3 Device Programmer is a universal, CE compliant device programmer with programmable voltage verification at $V_{DDMIN}$ and $V_{DDMAX}$ for maximum reliability. It features a large LCD display (128 x 64) for menus and error messages and a modular, detachable socket assembly to support various package types. The ICSP™ cable assembly is included as a standard item. In Stand-Alone mode, the MPLAB PM3 Device Programmer can read, verify and program PIC devices without a PC connection. It can also set code protection in this mode. The MPLAB PM3 connects to the host PC via an RS-232 or USB cable. The MPLAB PM3 has high-speed communications and optimized algorithms for quick programming of large memory devices and incorporates an MMC card for file storage and data applications.

## 22.13 Demonstration/Development Boards, Evaluation Kits, and Starter Kits

A wide variety of demonstration, development and evaluation boards for various PIC MCUs and dsPIC DSCs allows quick application development on fully functional systems. Most boards include prototyping areas for adding custom circuitry and provide application firmware and source code for examination and modification.

The boards support a variety of features, including LEDs, temperature sensors, switches, speakers, RS-232 interfaces, LCD displays, potentiometers and additional EEPROM memory.

The demonstration and development boards can be used in teaching environments, for prototyping custom circuits and for learning about various microcontroller applications.

In addition to the PICDEM™ and dsPICDEM™ demonstration/development board series of circuits, Microchip has a line of evaluation kits and demonstration software for analog filter design, KEELOQ® security ICs, CAN, IrDA®, PowerSmart battery management, SEEVAL® evaluation system, Sigma-Delta ADC, flow rate sensing, plus many more.

Also available are starter kits that contain everything needed to experience the specified device. This usually includes a single application and debug capability, all on one board.

Check the Microchip web page (www.microchip.com) for the complete list of demonstration, development and evaluation kits.

**TABLE 23-11: ELECTRICAL CHARACTERISTICS: BOR**

| DC CHARACTERISTICS | | | Standard Operating Conditions: 2.5V to 5.5V (unless otherwise stated)<br>Operating temperature -40°C ≤ TA ≤ +85°C for Industrial<br>-40°C ≤ TA ≤ +125°C for Extended | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Param No.** | **Symbol** | **Characteristic** | | **Min** | **Typ(1)** | **Max** | **Units** | **Conditions** |
| BO10 | VBOR | BOR Voltage on VDD Transition High-to-Low(2) | BORV = 11(3) | — | — | — | V | Not in operating range |
| | | | BORV = 10 | 2.6 | — | 2.71 | V | |
| | | | BORV = 01 | 4.1 | — | 4.4 | V | |
| | | | BORV = 00 | 4.58 | — | 4.73 | V | |
| BO15 | VBHYS | | | — | 5 | — | mV | |

**Note 1:** Data in "Typ" column is at 5V, 25°C unless otherwise stated. Parameters are for design guidance only and are not tested.
**2:** These parameters are characterized but not tested in manufacturing.
**3:** '11' values not in usable operating range.

**TABLE 23-12: DC CHARACTERISTICS: PROGRAM AND EEPROM**

| DC CHARACTERISTICS | | | Standard Operating Conditions: 2.5V to 5.5V (unless otherwise stated)<br>Operating temperature -40°C ≤ TA ≤ +85°C for Industrial<br>-40°C ≤ TA ≤ +125°C for Extended | | | | |
|---|---|---|---|---|---|---|---|
| **Param No.** | **Symbol** | **Characteristic** | **Min** | **Typ(1)** | **Max** | **Units** | **Conditions** |
| | | **Data EEPROM Memory(2)** | | | | | |
| D120 | ED | Byte Endurance | 100K | 1M | — | E/W | -40°C ≤ TA ≤ +85°C |
| D121 | VDRW | VDD for Read/Write | VMIN | — | 5.5 | V | Using EECON to read/write<br>VMIN = Minimum operating voltage |
| D122 | TDEW | Erase/Write Cycle Time | 0.8 | 2 | 2.6 | ms | RTSP |
| D123 | TRETD | Characteristic Retention | 40 | 100 | — | Year | Provided no other specifications are violated |
| D124 | IDEW | IDD During Programming | — | 10 | 30 | mA | Row Erase |
| | | **Program Flash Memory(2)** | | | | | |
| D130 | EP | Cell Endurance | 10K | 100K | — | E/W | -40°C ≤ TA ≤ +85°C |
| D131 | VPR | VDD for Read | VMIN | — | 5.5 | V | VMIN = Minimum operating voltage |
| D132 | VEB | VDD for Bulk Erase | 4.5 | — | 5.5 | V | |
| D133 | VPEW | VDD for Erase/Write | 3.0 | — | 5.5 | V | |
| D134 | TPEW | Erase/Write Cycle Time | 0.8 | 2 | 2.6 | ms | RTSP |
| D135 | TRETD | Characteristic Retention | 40 | 100 | — | Year | Provided no other specifications are violated |
| D137 | IPEW | IDD During Programming | — | 10 | 30 | mA | Row Erase |
| D138 | IEB | IDD During Programming | — | 10 | 30 | mA | Bulk Erase |

**Note 1:** Data in "Typ" column is at 5V, 25°C unless otherwise stated.
**2:** These parameters are characterized but not tested in manufacturing.

## 23.2 AC Characteristics and Timing Parameters

The information contained in this section defines dsPIC30F AC characteristics and timing parameters.