# Section 30. Programmable Cyclic Redundancy Check (CRC)

## HIGHLIGHTS

This section of the manual contains the following major topics:

## 30.1 INTRODUCTION

The programmable Cyclic Redundancy Check (CRC) module in PIC24F devices is a software configurable CRC checksum generator. The checksum is a unique number associated with a message or a particular block of data containing several bytes. Whether it is a data packet for communication, or a block of data stored in memory, a piece of information like checksum helps to validate it before processing. The simplest way to calculate a checksum is to add together all the data bytes present in the message. However, this method of checksum calculation fails badly when the message is modified by inverting or swapping groups of bytes. Also, it fails when null bytes are added anywhere in the message.

The CRC is a more complicated, but robust, error checking algorithm. The main idea behind the CRC algorithm is to treat a message as a binary bit stream and divide it by a fixed binary number. The remainder from this division is considered as the checksum. Like in division, the CRC calculation is also an iterative process. The only difference is that these operations are done on modulo arithmetic based on mod2. For example, division is replaced with the XOR operation (i.e., subtraction without carry). The CRC algorithm uses the term polynomial to perform all of its calculations. The divisor, dividend and remainder that are represented by numbers are termed as polynomials with binary coefficients. For example, the number, 25h (11001), is represented as:

**Equation 30-1:**

$$(1 * x4) + (1 * x3) + (0 * x2) + (0 * x1) + (1* x0), \text{ or } x4 + x3 + x0$$

In order to perform the CRC calculation, a suitable divisor is first selected. This divisor is called the generator polynomial. Since CRC is used to detect errors, a suitable generator polynomial of suitable length needs to be chosen for a given application, as each polynomial has different error detection capabilities. Some polynomials are widely used for many applications, but the error detecting capabilities of any particular polynomial are beyond the scope of this reference chapter.

The CRC calculation is an iterative process and consumes considerable CPU bandwidth when implemented in software. The software configurable CRC hardware module in PIC24F devices facilitates a fast CRC checksum calculation with minimal software overhead.

The primary features of the programmable CRC module are:

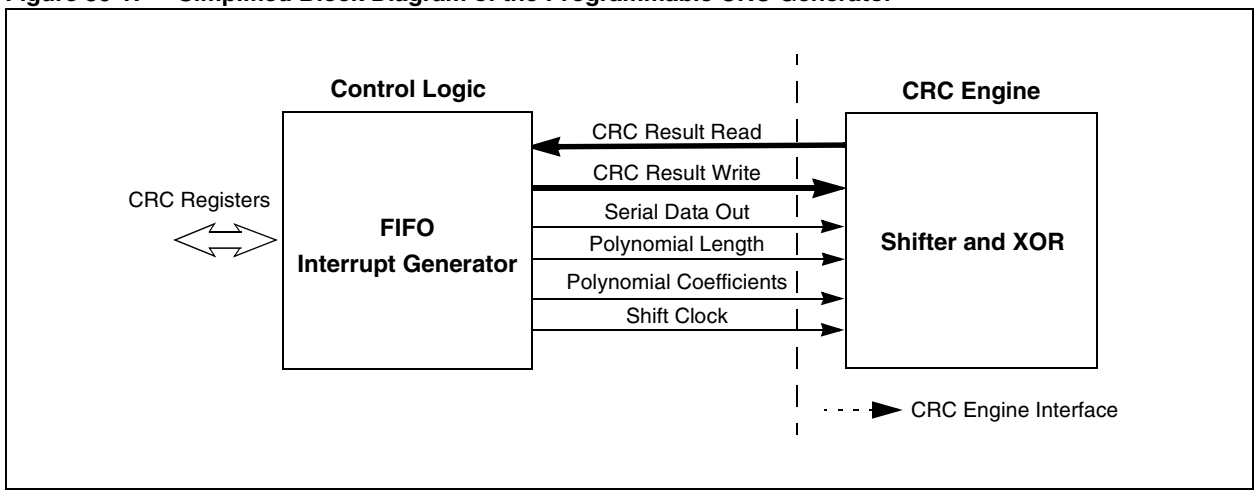
- Programmable bit length for the CRC generator polynomial (up to 16-bit length)
- Programmable CRC generator polynomial
- Interrupt output
- 8-deep, 16-bit or 16-deep, 8-bit FIFO for data input

## 30.2 MODULE OVERVIEW

The programmable CRC generator module in PIC24F devices can be broadly classified into two parts: the control logic and the CRC engine. The control logic incorporates a register interface, FIFO, interrupt generator and CRC engine interface. The CRC engine incorporates a CRC calculator which is implemented using a serial shifter with XOR function. A simplified block diagram is shown in Figure 30-1.

**Figure 30-1: Simplified Block Diagram of the Programmable CRC Generator**




## 30.3 CRC REGISTERS

Different registers associated with the CRC module are described in detail in this section. These registers are mapped onto the data RAM space as Special Function Registers (SFRs) in PIC24F devices:

- CRCCON (CRC Control Register)
- CRCXOR (CRC XOR Register)
- CRCDAT (CRC Data Register)
- CRCWDAT (Write CRC Shift Register)

The CRCCON register (Register 30-1) is the primary control and status register for the module. The CRCXOR register (Register 30-2) is used to define the generator polynomial by selecting the terms to be used. The CRCDAT and CRCWDAT registers are buffers for data input and result output, respectively.

**Register 30-1: CRCCON: CRC Control Register**

| U-0 | U-0 | R/W-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| — | — | CSIDL | VWORD4 | VWORD3 | VWORD2 | VWORD1 | VWORD0 |
| bit 15 | | | | | | | bit 8 |

| R-0 | R-1 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| CRCFUL | CRCMPT | — | CRCGO | PLEN3 | PLEN2 | PLEN1 | PLEN0 |
| bit 7 | | | | | | | bit 0 |

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

bit 15-14 **Unimplemented:** Read as '0'

bit 13 **CSIDL:** CRC Stop in Idle Mode bit
1 = Discontinue module operation when device enters Idle mode
0 = Continue module operation in Idle mode

bit 12-8 **VWORD4:VWORD0:** FIFO Pointer Value bits
Indicates the number of valid words or bytes in the FIFO. Has a maximum value of 8 when the PLEN3:PLEN0 bits (CRCCON<3:0>) > 7, or a value of 16 when the PLEN3:PLEN0 bits (CRCCON<3:0>) ≤ 7

bit 7 **CRCFUL**: FIFO Full bit
1 = FIFO is full
0 = FIFO is not full

bit 6 **CRCMPT:** FIFO Empty Bit
1 = FIFO is empty
0 = FIFO is not empty

bit 5 **Unimplemented:** Read as '0'

bit 4 **CRCGO:** Start CRC bit
1 = Start CRC serial shifter
0 = CRC serial shifter turned off

bit 3-0 **PLEN3:PLEN0:** Polynomial Length bits
Generator Polynomial Length = Value of PLEN<3:0> plus 1

**Register 30-2: CRCXOR: CRC XOR Register**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| X15 | X14 | X13 | X12 | X11 | X10 | X9 | X8 |
| bit 15 | | | | | | | bit 8 |

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | U-0 |
|---|---|---|---|---|---|---|---|
| X7 | X6 | X5 | X4 | X3 | X2 | X1 | — |
| bit 7 | | | | | | | bit 0 |

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

bit 15-1 **X15:X1:** XOR of Polynomial Term n Enable bits
1 = Include (XOR) the nth term ($x^n$) in the polynomial
0 = Do not include $x^n$ in the polynomial

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

## 30.4 CRC ENGINE

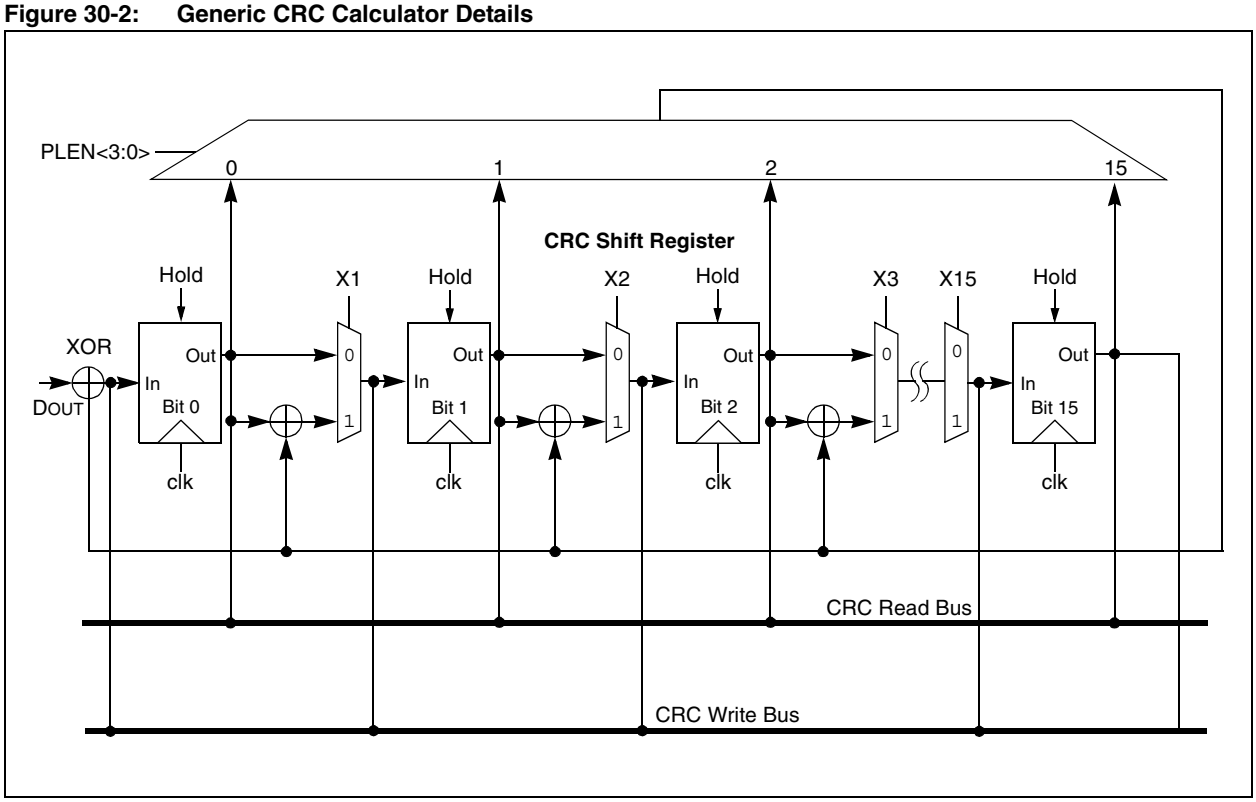
### 30.4.1 Generic CRC Engine

The CRC engine is a serial shifting CRC calculator with feedforward and feedback points, configurable though multiplexor settings. The topology of a generic CRC calculator is shown in Figure 30-2.

The CRC algorithm uses a simplified form of arithmetic process, using the XOR operation instead of binary division. The coefficients of the generator polynomial are programmed with the CRCXOR<15:1> bits. Writing a '1' into a location enables XORing of that element in the polynomial. The length of the polynomial is programmed using the PLEN<3:0> bits in the CRCCON register (CRCCON<3:0>). The PLEN<3:0> value signals the length of the polynomial, and switches a multiplexor to indicate the tap from which the feedback originated.

The result of the CRC calculation is obtained by reading the holding registers through the CRC read bus. A direct write path to the CRC Shift registers is also provided through the CRC write bus. This path is accessed by the CPU through the CRCWDAT register.

**Figure 30-2: Generic CRC Calculator Details**



### 30.4.2 Software Configuration of the CRC Engine

The CRC engine needs to be properly configured in software for a given generator polynomial. The generator polynomial is a hexadecimal number with n bits. The Most Significant bit is represented as $x^n$ and the Least Significant bit is represented as $x^1$. The Most Significant bit is always assumed to be '1'. The $x^0$ coefficient is omitted and understood to be '1'. Hence, only the coefficients of $x^{n-1}$ to $x^1$ need to be programmed in the CRCXOR register.

Consider a specific CRC polynomial as an example:

**Equation 30-2:**

$$x^{16} + x^{12} + x^5 + 1$$

The length of the polynomial is represented by the order of the highest power of the polynomial. Hence, the above polynomial is of a 16-bit length. Some of its coefficients are zeros and some are ones.

To program this polynomial into a CRC generator, the PLEN bits (CRCCON<3:0>) and CRCXOR<15:1> bits should be programmed as shown in Table 30-1.

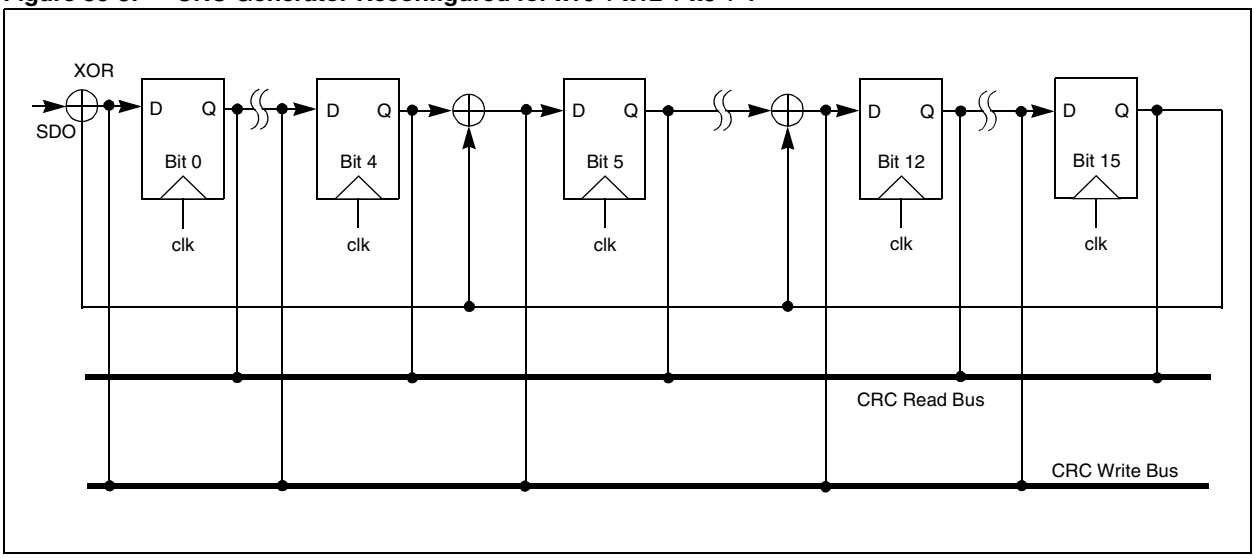
**Table 30-1: Example CRC Setup**

| Register Names | Bit Names | Bit Values |
|---|---|---|
| CRCCON | PLEN3:PLEN0 | 0Fh |
| CRCXOR | X15:X1 | 1020h |

The polynomial length in this case is 16 (PLEN<3:0> + 1). Note that for the value of X15:X1, as programmed in Table 30-1, the 12 and 5 bits are set to '1' as required by the generator polynomial. Bit 0 is always XORed. For a 16-bit polynomial, the 16th bit is always assumed to be XORed; therefore, there is no CRCXOR bit for either bit 0 or bit 16.

The topology of the CRC generator configured for the example polynomial is shown in Figure 30-3.

**Figure 30-3: CRC Generator Reconfigured for x16 + x12 + x5 + 1**




**Note:** The $x^0$ bit in the CRCXOR register is omitted and is always assumed to be '1'. Hence, a polynomial with a Least Significant bit of '0' or '1' (e.g., 1020h or 1021h) has the same effect on the CRC calculation.

## 30.5 CONTROL LOGIC

### 30.5.1 FIFO

The FIFO is physically implemented as an 8-deep, 16-bit wide storage element. The logic associated with the FIFO contains a 5-bit counter, called VWORD (VWORD4:VWORD0, or CRCCON<12:8>). The value in the VWORD4:VWORD0 bits indicates the number of new data elements in the FIFO.

The FIFO behaves as an 8-deep, 16-bit wide array when PLEN3:PLEN0 > 7, and a 16-deep, 8-bit wide array otherwise. The data for which the CRC is to be calculated must first be written into the FIFO by the CPU using the CRCDAT register. Data must always be written into the CRCDAT register. Reading of the CRCDAT register is not allowed and always returns zero.

The smallest data element that can be written into the FIFO is one byte. When PLEN3:PLEN0 ≤ 7, every byte write into the FIFO increments VWORD by one, or by two, for every word write operation.

If PLEN3:PLEN0 > 7, a word write into the FIFO increments the value of VWORD by one. A single byte write to the CRCDAT register does not increment the value of VWORD; instead, VWORD increments by one only after an even number of bytes (integer multiple of words) are written into the CRCDAT register.

The CRCFUL bit is set (indicating the FIFO is full) when the value of VWORD reaches 8 (for the 8-deep, 16-bit FIFO configuration) or 16 (for the 16-deep, 8-bit FIFO configuration). The user needs to ensure that the FIFO is not full while writing a new value to the CRCDAT register.

### 30.5.2 CRC Engine Interface

#### 30.5.2.1 FIFO TO CRC CALCULATOR

To start serial shifting from the FIFO to the CRC calculator, the CRCGO bit (CRCCON<4>) must be set (= 1). The serial shifter starts shifting data, starting from the Most Significant bit first, into the CRC engine only when CRCGO = 1 and the value of VWORD is greater than zero. If the CRCFUL bit was set earlier, then it is cleared when VWORD decrements by one. VWORD decrements by one when a FIFO location gets shifted completely to the CRC calculator. The serial shifter continues shifting until VWORD reaches zero, at which point, the CRCMPT bit becomes set to indicate that the FIFO is empty.

The frequency of the CRC shift clock is twice that of the PIC24F instruction clock cycle, thus making this hardware shifting process faster than a software shifter. The users can write into the FIFO while the shift operation is in progress. For a continuous data feed into the CRC engine, the recommended mode of operation is to initially “prime” the FIFO with a sufficient number of words or bytes. Once this is completely done, the user can start the CRC by setting the CRCGO bit to ‘1’. From this point onwards, either VWORD or the CRCFUL bit should be monitored. If the CRCFUL bit is not set, or the VWORD reads less than 8 or 16, another word can be written onto the FIFO. At least one instruction cycle must pass after a write to the CRCDAT register before a read of the VWORD bits is done.

To empty words already written into a FIFO, the CRCGO bit must be set to ‘1’ and the CRC shifter must be allowed to run until the CRCMPT bit is set.

| | |
|---|---|
| **Note:** | When PLEN3:PLEN0 > 7, an integer multiple of words should be loaded into the FIFO before the application software sets the CRCGO bit. If the CRCGO bit is set after loading an odd number of bytes into the FIFO, the last odd byte is never shifted out, and the CRCMPT bit always remains at ‘0’, indicating that the FIFO is not empty. |

#### 30.5.2.2 NUMBER OF DATA BITS SHIFTED FROM FIFO

The number of data bits to be shifted depends upon the length of the polynomial selected. For example, if PLEN<3:0> = 5, then the length of the generator polynomial and the size of one data is 6 bits (PLEN<3:0> + 1). In this case, a full byte of a FIFO location is not shifted out, even though the CPU can write only a byte. Only 6 bits of a byte are shifted out, starting from the 6th bit (i.e., the MSb in this case). The two Most Significant bits of each byte are don't care bits. Therefore, for a given value of PLEN3:PLEN0, it will take ((PLEN<3:0> + 1) * VWORD) number of shift clock cycles to complete the CRC calculations. Similarly, for a 12-bit polynomial selection, the shifting starts from the 12th bit of the word, which is the Most Significant bit for this selection. The Most Significant 4 bits of each word are ignored.

| | |
|---|---|
| **Note:** | For 'n' bit polynomial selection, the CRC calculation is done with integer multiple of 'n' bit of data. For example, for a 16 bit polynomial, the CRC calculation is done with integer multiple of words. |

#### 30.5.2.3 CRC RESULT

When the CPU reads the CRCWDAT register, the CRC result is read directly out of the shift register through the CRC read bus. To get the correct CRC reading, it is necessary to wait for the CRCMPT bit to go high before reading the CRCWDAT register.

A direct write path to the CRC Shift registers is also provided through the CRC write bus. This path is accessed by the CPU through the CRCWDAT register. The CRCWDAT register can be loaded with a desired value prior to the start of the shift process.

| | |
|---|---|
| **Note:** | When the CPU writes the shift registers directly though the CRCWDAT register, the CRCGO bit must be '0'. |

### 30.5.3 Interrupt Operation

Serial shifting of the FIFO data to the CRC engine begins when the CRCGO bit is set and the VWORD4:VWORD0 bits (VWORD) are greater than zero. During this process, if the CRCMPT bit makes a transition from '0' (not empty) to '1' (empty), or when the VWORD4:VWORD0 bits make a transition from any value greater than zero to zero, the CRCIF interrupt flag becomes set. If the CRC interrupt is enabled by setting the CRCIE bit, and the CRCIF bit becomes set, then an interrupt is generated.

Table 30-2 in **Section 30.9 "Register Maps"** details the interrupt register associated with the CRC module. For more details on interrupts and interrupt priority settings, refer to **Section 8. "Interrupts"**.

| | |
|---|---|
| **Note:** | If new data is written into the CRCDAT register when the CRCFUL bit is set, the VWORD Pointer rolls over through '0'. However, the CRC Interrupt Flag, CRCIF, is not set in this condition. In this condition, the CRCFUL bit gets reset, all previous data written into the FIFO is lost and the new data is written into the first location of the FIFO. Remaining locations of the FIFO are empty and new data can be written into the empty locations. |

## 30.6 ADVANTAGES OF PROGRAMMABLE CRC MODULE

The CRC algorithm is straightforward to implement in software. However, it requires considerable CPU bandwidth to implement the basic requirements, such as shift, bit test and XOR. Moreover, CRC calculation is an iterative process and additional software overhead for data transfer instructions puts enormous burden on the MIPS requirement of a microcontroller.

The CRC engine in PIC24F devices calculates the CRC checksum without CPU intervention. Moreover, it is much faster than the software implementation; the CRC engine consumes only half of an instruction cycle per bit for its calculation as the frequency of the CRC shift clock is twice that of the PIC24F instruction clock cycle. For example, the CRC hardware engine takes only 64 instruction cycles to calculate a CRC checksum on a message that is 128 bits (16x8) long. The same calculation, if implemented in software, will consume more than a thousand instruction cycles even for a well optimized piece of code.

## 30.7 APPLICATION OF CRC MODULE

Calculating a CRC is a robust error checking algorithm in digital communication for messages containing several bytes or words. After calculation, the checksum is appended to the message and transmitted to the receiving station. The receiver calculates the checksum with the received message to verify the data integrity.

### 30.7.1 Variations

The CRC module of PIC24F devices shifts out the Most Significant bit first. This is a popular implementation as employed in XMODEM protocol. In one of the variations (CCITT protocol) for CRC calculation, the Least Significant bit is shifted out first. This requires bit reversal of the message polynomial in the software before feeding the message to the PIC24F CRC hardware module, and this also adds considerable software overhead. Discussions on all the variations are beyond the scope of this document, but several variations of CRC can be implemented using the programmable CRC module in PIC24F devices with minimal software overhead.

The choice of the polynomial length, and the polynomial itself, are application dependent. Polynomial lengths of 5, 7, 8, 10, 12 and 16 are normally used in various standard implementations. The CRC module in PIC24F devices can be configured for different polynomial lengths and for different equations. If a polynomial of n bits is selected for calculation, normally n zeros are appended to the message stream, though there are variations in this process as well. The following sections explain the recommended step-by-step procedure for CRC calculation, where n zeros are appended to the message stream for an n bit polynomial. Users can decide whether zeros, or any other values, need to be appended to the message stream. Depending on the application, the user may decide whether any value needs to be appended at all.

### 30.7.2 8-Bit Polynomial

The recommended procedure to calculate a CRC with an 8-bit polynomial is as follows:

1. Program PLEN3:PLEN0 bits (CRCCON<3:0>) = 07h.
2. Program a value to CRCXOR (e.g., CRCXOR = 31h).
3. Program a value in CRCWDAT:
   - 0000h (for the start of a new calculation), or
   - the previously calculated partial result, (for part of the whole message stream).
4. If the CRCFUL bit is not set and if all the data bytes of the message stream are not written into the FIFO, then write a data byte to the CRCDAT register.
5. If the CRCFUL bit is not set, and if all the data bytes of the message stream have already been written into the FIFO, then write a byte of 00h in the CRCDAT register and set a software flag bit in the application using the CRC (i.e., `FINAL_CALCULATION`).
6. If the CRCFUL bit or the software `FINAL_CALCULATION` flag is set, then start CRC by setting the CRCGO bit.
7. When CRCMPT is set, clear the CRCGO bit and read the result byte from the CRCWDAT register.
8. For a partial result (CRC calculation is done but the `FINAL_CALCULATION` flag is not set), pass the partial result to the next calculation process.

### 30.7.3 5-Bit or 7-Bit Polynomials

For 5-bit or 7-bit polynomials, the CRC module will calculate the checksum taking into account the 5 or 7 Least Significant bits of a byte, respectively. In the case of 5 bits of data, a byte should contain the 5 bits of data in its 5 Least Significant bits; the Most Significant 3 bits of the byte may be programmed as zeros. In case of a 7-bit calculation, a byte should contain the 7 bits of data in its 7 Least Significant bits; the Most Significant bit of a byte may be programmed as zero. Refer to **Section 30.5.2.1 "FIFO to CRC Calculator"** for more details.

After forming the bytes from a message stream, the same steps as explained in **Section 30.7.2 "8-Bit Polynomial"** can be applied. For the polynomial length (PLEN3:PLEN0), use values of 04h or 06h for 5-bit and 7-bit polynomials, respectively. A suitable 5-bit or 7-bit generator polynomial may be programmed in the CRCXOR register.

### 30.7.4 16-Bit Polynomials

The recommended procedure to calculate a CRC with a 16-bit polynomial is as follows:

1. Program PLEN3:PLEN0 bits (CRCCON<3:0>) = 0Fh.
2. Program a value to CRCXOR (e.g., CRCXOR = 8005h).
3. Program a value in CRCWDAT:
   - 0000h (for the start of a new calculation), or
   - the previously calculated partial result (for part of the whole message stream).
4. If the CRCFUL bit is not set and if all the data words of the message stream are not written into the FIFO, then write a data word to the CRCDAT register.
5. If the CRCFUL bit is not set and if all the data words of the message stream are already written into the FIFO, then write a word of 0000h in CRCDAT and set a software flag in the application using the CRC (i.e., `FINAL_CALCULATION`).
6. If the CRCFUL bit or the software `FINAL_CALCULATION` flag is set, then start CRC by setting the CRCGO bit.
7. When CRCMPT is set, then clear the CRCGO bit and read the result byte from the CRCWDAT register.
8. For a partial result (CRC calculation is done but the `FINAL_CALCULATION` flag is not set), pass the partial result to the next calculation process.

| **Note:** | If the length of the polynomial is 16 bits, the CRC module expects an integer multiple of 16 bits in the FIFO |
|---|---|

A word write is a simple process for a 16-bit polynomial. However, in some applications, byte write operations may be used with 16-bit polynomials (e.g. in UART transmission/reception). In these applications, an odd number of bytes may need to be padded up with an extra dummy byte. A dummy byte should not be added if the message stream contains an even number of bytes. In this case, the procedure explained above for 16-bit polynomials may need to be modified as follows:

1. Program PLEN3:PLEN0 bits (CRCCON<3:0>) = 0Fh.
2. Program a value to CRCXOR (e.g., CRCXOR = 8005h).
3. Program a value in CRCWDAT:
   - 0000h (for the start of a new calculation), or
   - The previously calculated partial result (for part of the whole message stream).
4. If the CRCFUL bit is not set, and if all the data bytes of the message stream are not written into the FIFO, then write a data byte to the CRCDAT register and increment a counter to keep track of the number of bytes written to the FIFO.
5. If the CRCFUL bit is not set, and if all the data bytes of the message stream are already written into the FIFO which are odd, then write a byte of 00h (dummy byte) to CRCDAT and set a software flag in the software application (i.e., `MESSAGE_OVER`).
6. If the CRCFUL bit is not set and if all the data bytes of the message stream are already written into the FIFO which are even, then set a software flag (`MESSAGE_OVER`).
7. If the CRCFUL bit is not set and if the `MESSAGE_OVER` flag is set, write a word of 0000h to CRCDAT and set a software flag (i.e., `FINAL_CALCULATION`).
8. If the CRCFUL bit or the `FINAL_CALCULATION` flag is set, start CRC by setting the CRCGO bit.
9. When CRCMPT is set, clear the CRCGO bit and read the result byte from the CRCWDAT register.
10. For a partial result (CRC calculation is done but the `FINAL_CALCULATION` flag is not set), pass the partial result to the next calculation process.

### 30.7.5 10-Bit or 12-Bit Polynomial

For 10-bit or 12-bit polynomials, the CRC module calculates the checksums by taking into account the 10 or 12 Least Significant bits of a word, respectively. For 10 bits of data, the Most Significant 6 bits of a word may be programmed as zero. For 12-bit calculation, the Most Significant 4 bits of a word may be programmed as zero. Refer to **Section 30.5.2.1 "FIFO to CRC Calculator"** for more details.

After forming the words with 10 or 12 bits of actual data and the rest as don't care bits, the same steps as explained in **Section 30.7.4 "16-Bit Polynomials"** can be applied. For the PLEN3:PLEN0 bits, use a value of 09h or 0Bh for 10-bit or 12-bit polynomials, respectively. A suitable generator polynomial of the same length may be programmed in the CRCXOR register.

## 30.8 OPERATION IN POWER SAVE MODES

### 30.8.1 Sleep Mode

If Sleep mode is entered while the module is operating, the module is suspended in its current state until clock execution resumes.

### 30.8.2 Idle Mode

To continue full module operation in Idle mode, the CSIDL bit must be cleared prior to entry into the mode.

If CSIDL = `1`, the module behaves the same way as it does in Sleep mode; pending interrupt events will be passed on, even though the module clocks are not available.

## 30.9 REGISTER MAPS

A summary of the Special Function Registers associated with the PIC24F programmable CRC module is provided in Table 30-2.

**Table 30-2: Special Function Registers Associated with the Programmable CRC Module(1)**

| File Name | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | All Resets |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CRCCON | — | — | CSIDL | VWORD4 | VWORD3 | VWORD2 | VWORD1 | VWORD0 | CRCFUL | CRCMPT | — | CRCGO | PLEN3 | PLEN2 | PLEN1 | PLEN0 | 0040 |
| CRCXOR | X15 | X14 | X13 | X12 | X11 | X10 | X9 | X8 | X7 | X6 | X5 | X4 | X3 | X2 | X1 | — | 0000 |
| CRCDAT | DATA15 | DATA14 | DATA13 | DATA12 | DATA11 | DATA10 | DATA9 | DATA8 | DATA7 | DATA6 | DATA5 | DATA4 | DATA3 | DATA2 | DATA1 | DATA0 | 0000 |
| CRCWDAT | SDATA15 | SDATA14 | SDATA13 | SDATA12 | SDATA11 | SDATA10 | SDATA9 | SDATA8 | SDATA7 | SDATA6 | SDATA5 | SDATA4 | SDATA3 | SDATA2 | SDATA1 | SDATA0 | 0000 |
| IFS4 | — | — | — | — | — | — | — | — | — | — | — | — | CRCIF | U2ERIF | U1ERIF | — | 0000 |
| IEC4 | — | — | — | — | — | — | — | — | — | — | — | — | CRCIE | U2ERIE | U1ERIE | — | 0000 |
| IPC16 | — | CRCIP2 | CRCIP1 | CRCIP0 | — | U2ERIP2 | U2ERIP1 | U2ERIP0 | — | U1ERIP2 | U1ERIP1 | U1ERIP0 | — | — | — | — | 4440 |

**Legend:** — = unimplemented, read as '0'. Shaded bits are not used in the operation of the programmable CRC module.
**Note 1:** Please refer to the device data sheet for specific memory map details.

## 30.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC24F device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the programmable CRC are:

| Title | Application Note # |
|---|---|
| No related application notes at this time. | |

**Note:** Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC24F family of devices.

## 30.11 REVISION HISTORY

### Revision A (November 2006)

This is the initial released revision of this document.

**NOTES:**