Section 41. 32-Bit Programmable Cyclic Redundancy Check (CRC)

HIGHLIGHTS

This section of the manual contains the following major topics:

41.1 INTRODUCTION

The 32-bit 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 41-1:

$$(1 * x^4) + (1 * x^3) + (0 * x^2) + (0 * x^1) + (1 * x^0) \text{ or } x^4 + x^3 + x^0$$

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 a 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 section.

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.

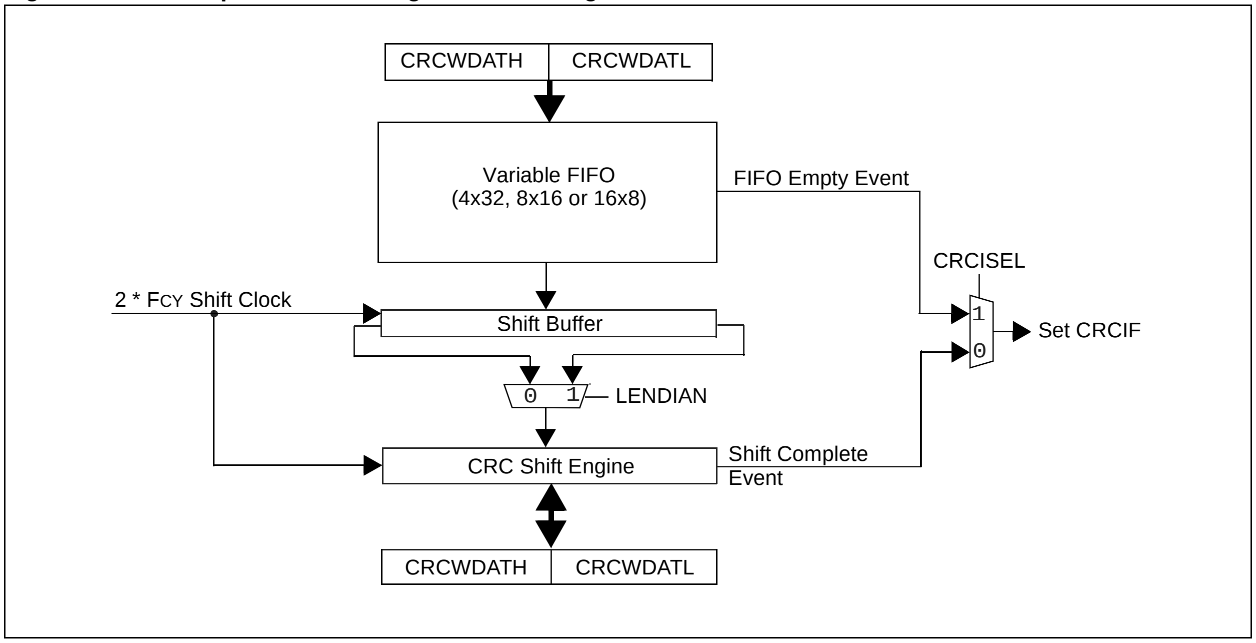
The programmable CRC generator provides a hardware implemented method of quickly generating checksums for various communication and security applications. It provides the following features:

- User-programmable CRC polynomial equation, up to 32 bits
- Programmable shift direction (little or big endian)
- Independent data and polynomial lengths
- Configurable Interrupt output
- Data FIFO

41.2 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 41-1.

Figure 41-1: Simplified Block Diagram of the Programmable CRC Generator



41.3 CRC REGISTERS

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

- CRCCON1 (CRC Control Register 1)
- CRCCON2 (CRC Control Register 2)
- CRCXORL (CRC XOR Low Register)
- CRCXORH (CRC XOR High Register)
- CRCDATL (CRC Data Low Register)
- CRCDATH (CRC Data High Register)
- CRCWDATL (CRC Shift Low Register)
- CRCWDATH (CRC Shift High Register)

The CRCCON1 (Register 41-1) and CRCCON2 Register 41-2) registers control the operation of the module and configure various settings. The CRCXOR registers (Register 41-3 and Register 41-4) select the polynomial terms to be used in the CRC equation. The CRCDAT and CRCWDAT registers are each register pairs that serve as buffers for the double-word input data and CRC processed output, respectively.

Register 41-1: CRCCON1: CRC Control1 Register

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

R-0	R-1	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0
CRCFUL	CRCMPT	CRCISEL	CRCGO	LENDIAN	—	—	—
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	**CRCEN:** CRC Enable bit 1 = Enables module 0 = Disables module
bit 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	**VWORD<4:0>:** Counter Value bits Indicates the number of valid words in the FIFO. Has a maximum value of 16 when DWIDTH<4:0> ≤ 7 (data words, 8-bit wide or less). Has a maximum value of 8 when DWIDTH<4:0> ≤ 15 (data words from 9 to 16-bit wide). Has a maximum value of 4 when DWIDTH<4:0> ≤ 31 (data words from 17 to 32-bit wide).
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	**CRCISEL:** CRC Interrupt Selection bit 1 = Interrupt on FIFO empty; final word of data still shifting through CRC 0 = Interrupt on shift complete and results ready
bit 4	**CRCGO:** Start CRC bit 1 = Start CRC serial shifter; clearing the bit aborts shifting 0 = CRC serial shifter turned off
bit 3	**LENDIAN:** Data Word Little Endian Configuration bit 1 = Data word is shifted ino the CRC, starting with the LSb (little endian); reflected input data 0 = Data word is shifted into the CRC, starting with the MSb (big endian); non-reflected input data
bit 2-0	**Unimplemented:** Read as ‘0’

Register 41-2: CRCCON2: CRC Control2 Register

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DWIDTH4	DWIDTH3	DWIDTH2	DWIDTH1	DWIDTH0
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PLEN4	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 15-13	**Unimplemented:** Read as ‘0’
bit 12-8	**DWIDTH<4:0>:** Data Word Width Configuration bits Configures the width of the data word (data word width – 1).
bit 7-5	**Unimplemented:** Read as ‘0’
bit 4-0	**PLEN<4:0>:** Polynomial Length Configuration bits Configures the length of the polynomial (polynomial length – 1).

Register 41-3: CRCXORL: CRC XOR Low 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	**X<15:1>:** XOR of Polynomial Term x^n Enable bits
bit 0	**Unimplemented:** Read as ‘0’

Register 41-4: CRCXORH: CRC XOR High 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
X31	X30	X29	X28	X27	X26	X25	X24
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	R/W-0
X23	X22	X21	X20	X19	X18	X17	X16
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 **X<31:16>:** XOR of Polynomial Term x^n Enable bits

Register 41-5: CRCDATL: CRC Data Low Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
DATA15	DATA14	DATA13	DATA12	DATA11	DATA10	DATA9	DATA8
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	R/W-0
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
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 15-0 **DATA<15:0>:** CRC Input Data bits
Writing to this register fills the FIFO; reading from this register returns ‘0’.

Register 41-6: CRCDATH: CRC Data High Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
DATA31	DATA30	DATA29	DATA28	DATA27	DATA26	DATA25	DATA24
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	R/W-0
DATA23	DATA22	DATA21	DATA20	DATA19	DATA18	DATA17	DATA16
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 15-0 **DATA<31:16>:** CRC Input Data bits
Writing to this register fills the FIFO; Reading from this register returns ‘0’.

Register 41-7: CRCWDATL: CRC Shift Low Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SDATA15	SDATA14	SDATA13	SDATA12	SDATA11	SDATA10	SDATA9	SDATA8
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	R/W-0
SDATA7	SDATA6	SDATA5	SDATA4	SDATA3	SDATA2	SDATA1	SDATA0
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 15-0 **SDATA<15:0>:** CRC Shift Register bits

Writing to this register writes to the CRC Shift register through the CRC write bus. Reading from this register reads the CRC read bus.

Register 41-8: CRCWDATH: CRC Shift High Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SDATA31	SDATA30	SDATA29	SDATA28	SDATA27	SDATA26	SDATA25	SDATA24
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	R/W-0
SDATA23	SDATA22	SDATA21	SDATA20	SDATA19	SDATA18	SDATA17	SDATA16
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 15-0 **DATA<31:16>:** CRC Input Data bits

Writing to this register writes to the CRC Shift register through the CRC write bus. Reading from this register reads the CRC read bus.

41.4 CRC ENGINE

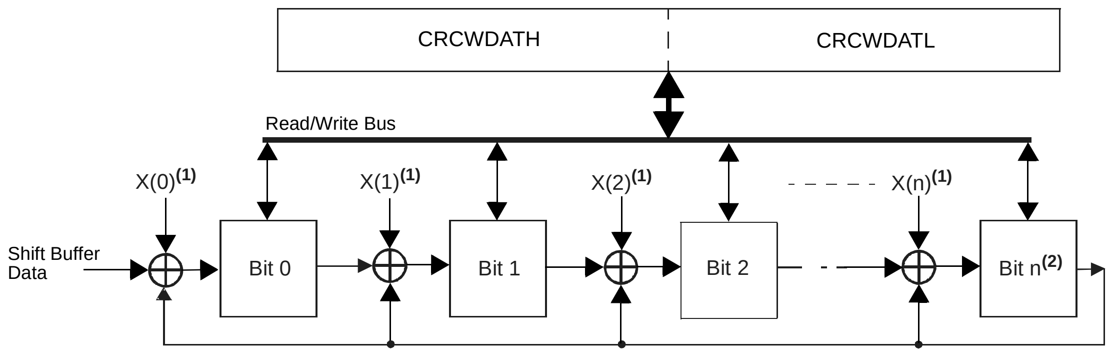
41.4.1 Generic CRC Engine

The CRC engine is a serial shifting CRC calculator with feedforward and feedback points, configurable though multiplexer settings. A simple version of the CRC shift engine is shown in Figure 41-2.

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 CRCXORL<15:0> and CRCXORH<31:16> 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<4:0> bits in the CRCCON2 register (CRCCON2<4:0>). The PLEN<4:0> value signals the length of the polynomial and switches a multiplexer 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 CRCWDATL and CRCWDATH registers.

Figure 41-2: CRC Shift Engine Detail



Note 1: Each XOR stage of the shift engine is programmable. See text for details.

2: Polynomial length n is determined by (<PLEN4:PLEN0> + 1).

41.5 CONTROL LOGIC

41.5.1 Polynomial Interface

The CRC module can be programmed for CRC polynomials of up to the 32nd order, using up to 32 bits. Polynomial length, which reflects the highest exponent in the equation, is selected by the PLEN<4:0> bits (CRCCON2<4:0>). The CRCXORL and CRCXORH registers control which exponent terms are included in the equation. Setting a particular bit includes that exponent term in the equation functionally; this includes an XOR operation on the corresponding bit in the CRC engine. Clearing the bit disables the XOR. For example, consider two CRC polynomials, one a 16-bit equation and the other a 32-bit equation:

Equation 41-2:

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

and

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

To program this polynomial into the CRC generator, set the register bits as shown in Table 41-1.

Table 41-1: CRC Setup Examples for 16 and 32-Bit Polynomials

CRC Control Bits	Bit Values	
	16-Bit Polynomial	32-Bit Polynomial
PLEN<4:0>	`01111`	`11111`
X<31:16>	`0000 0000 0000 0001`	`0000 0100 1100 0001`
X<15:0>	`0001 0000 0010 000x`	`0001 1101 1011 011x`

Note that the appropriate positions are set to ‘1’ to indicate that they are used in the equation (e.g., X26 and X23). The 0 bit required by the equation is always XORed; thus, X0 is a don’t care. For a polynomial of length 32, it is assumed that the 32nd bit will be used. Therefore, the X<31:1> bits do not have the 32nd bit.

41.5.2 Data Interface

The module accommodates user-defined input data width for calculating CRC. Input data width can be configured to any value between 1 and 32 bits using the DWIDTH<4:0> bits (CRCCON2<12:8>).

The input data is fed to the CRCDATL and CRCDATH registers. Depending upon the configuration of the DWIDTH bits, the width of the CRCDATL and CRCDATH registers is configured.

For data width less than or equal to 16 bits, only the CRCDATL register has to be used and any writes to the CRCDATH register will be ignored.

For data width greater than 16 bits, both the CRCDATL and CRCDATH registers should be used. The user must write the lower 16 bits (word) into the CRCDATL register first and then the upper bits into the CRCDATH register.

Note: For data width less than or equal to 8 bits, the user should feed the input data through byte operations into the CRCDATL register.

41.5.3 Data Shift Direction

The LENDIAN bit (CRCCON1<3>) is used to control the shift direction. By default, the CRC will shift data through the engine, MSb first (LENDIAN = `0`). Setting LENDIAN (= `1`) causes the CRC to shift data, LSb first. This setting allows better integration with various communication schemes and removes the overhead of reversing the bit order in software. Note that this only changes the direction the data is shifted into the engine. The result of the CRC calculation will still be a normal CRC result, not a reverse CRC result.

41.5.4 FIFO

The module incorporates a FIFO that works with a variable data width. The FIFO is physically implemented as an 8-deep, 16-bit wide storage element. FIFO is defined by the DWIDTH<4:0> bits (CRCCON2<12:8>). Input data width can be configured to any value between 1 and 32 bits using the DWIDTH bits. The logic associated with the FIFO contains a 5-bit counter, called VWORD (VWORD<4:0> or CRCCON1<12:8>). The value in the VWORD<4:0> bits indicates the number of new data elements in the FIFO.

The FIFO is:

- 16-word deep when DWIDTH<4:0> ≤ 7 (data words, 8-bit wide or less)
- 8-word deep when DWIDTH<4:0> ≤ 15 (data words from 9 to 16-bit wide)
- 4-word deep when DWIDTH<4:0> ≤ 31 (data words from 17 to 32-bit wide)

The data for which the CRC is to be calculated must first be written into the FIFO by the CPU using the CRCDATL register. Reading of the CRCDATL register is not allowed and always returns zero.

Filling the FIFO with 8-Bit Data:

With an 8-bit or less data word width setting, the FIFO increments on a write to either the lower or the upper byte of the CRCDATL register. The smallest data element that can be written into the FIFO is 1 byte. This allows for single, 8-bit data bytes to be written to the lower byte of the CRCDATL<7:0> register, followed by another single 8-bit data byte write into the CRCDATL<15:8> register.

For example, if DWIDTH is five, then the size of the data is DWIDTH + 1 or six. The data is written as a whole byte; the two unused upper bits are ignored by the module. Once data is written into the MSb of the CRCDAT registers (that is, MSb as defined by the data width), the value of the VWORD<4:0> bits (CRCCON< 12:8>) increments by one.

Filling the FIFO with Greater Than 8-Bit and Less Than/Equal to 16-Bit Data:

With greater than 8-bit, and less than or equal to a 16-bit data word width setting, the FIFO increments on a write to the CRCDATL register. Any write to the CRCDATH register will be ignored. The value of the VWORD<4:0> bits increment for every write to the CRCDATL register.

Filling the FIFO with Greater Than 16 and Less Than 32-Bit Data:

When the data width is greater than 16 bits, any write to the CRCDATH register increments the VWORD by one and writes the lower word into the CRCDATL register before writing the upper word into the CRCDATH register. For example, if DWIDTH is 24, VWORD will increment when bit 7 of CRCDATH is written. Therefore, CRCDATL must be written before CRCDATH.

To accommodate left or right shift methods, byte swapping can take place when filling the FIFO with different data widths. Pictorial descriptions of FIFO for different data widths and LENDIAN bit settings are shown in the Figure 41-3, Figure 41-4 and Figure 41-5.

Note: Ensure that the new data is not written into the CRCDATL and CRCDATH registers when the CRCFUL bit is set; if the new data is written, it will be ignored.

Figure 41-3: Filling the FIFO Word Width ≤ 8 Bits

Initial Conditions:

```
W1 = 0x0A01
W2 = 0x0B02
W3 = 0x0C03
W4 = 0x0D04
W5 = 0x0E05
W6 = 0x0F06
```

Code Executed to Fill the FIFO:

```
MOVW.b W1L, CRCDAT.L  'low byte to low byte write
MOVW.b W1H, CRCDAT.H  'high byte to high byte write
MOVW.b W2H, CRCDAT.L  'high byte to low byte write
MOVW.b W2L, CRCDAT.H  'low byte to high byte write
MOVW.w W3,  CRCDAT    'double byte to double byte write
MOVW.b W4L, CRCDAT.L  'set up single byte write prior to
MOVW.w W5,  CRCDAT    'double byte write
```

Resulting FIFO for Little Endian Mode:

Write	Location	Value
W5H	8	0x0E
W5L	7	0x05
W4L	6	0x04
W3H	5	0x0C
W3L	4	0x03
W2L	3	0x02
W2H	2	0x0B
W1H	1	0x0A
W1L	0	0x01

Feed LSB First

Resulting FIFO for Big Endian Mode:

Write	Location	Value
W5L	8	0x05
W5H	7	0x0E
W4L	6	0x04
W3L	5	0x03
W3H	4	0x0C
W2L	3	0x02
W2H	2	0x0B
W1H	1	0x0A
W1L	0	0x01

Feed MSB First

Note: Shaded cells highlight the difference in resulting FIFO contents between the Little Endian and Big Endian modes.

Figure 41-4: Filling the FIFO Word Width > 8 Bits and ≤ 16 Bits

Initial Conditions:

```
W1 = 0x0A01
W2 = 0x0B02
W3 = 0x0C03
W4 = 0x0D04
W5 = 0x0E05
W6 = 0x0F06
```

Code Executed to Fill the FIFO:

```
MOVW.w W1,  CRCDAT     'word to word write
MOVW.b W2L, CRCDAT.L   'write low byte first
MOVW.b W2H, CRCDAT.H   'write high byte second
MOVW.b W3H, CRCDAT.H   'write high byte first FIFO advances
MOVW.b W3L, CRCDAT.L   'low byte goes to next word
MOVW.b W4H, CRCDAT.H   'completes above word
MOVW.b W5L, CRCDAT.L   'write low byte first
MOVW.w W6,  CRCDAT     'overrides low byte from WSL
```

Resulting FIFO:

Write	Location	Value
W6	4	0x0F06
W4H, W3L	3	0x0D03
W3H	2	0x0C-[(1)]
W2L, W2H	1	0x0B02
W1	0	0x0A01

Feed MSB First

Note 1: Values of "-" indicate that the FIFO contains stale data from previous operations due to the way in which it was filled.

Figure 41-5: Filling the FIFO Word Width > 16 Bits and ≤ 32 Bits

Initial Conditions:

```
W1 = 0x0A01
W2 = 0x0B02
W3 = 0x0C03
W4 = 0x0D04
W5 = 0x0E05
W6 = 0x0F06
```

Code Executed to Fill the FIFO:

```
MOVW.w W1,  CRCDAT        'word to word write
MOVW.b W2L, CRCDATH.L     'write low byte first
MOVW.b W2H, CRCDATH.H     'write high byte second, FIFO advances
MOVW.b W3H, CRCDAT.H      'write high byte first
MOVW.b W3L, CRCDAT.L      'write low byte next
MOVW.w W4H, CRCDATH.H     'write high byte, FIFO advances
MOVW.w W5,  CRCDAT        'write low word first
MOVW.w W6,  CRCDATH       'write upper word second, FIFO advances
```

Word Width > 16 Bits and ≤ 32 Bits:

Write	Location	Value
W6, W5	2	0x0F060E05
W4H, W3L, W3H	1	0x0D-0C03(1)
W2L, W3H, W1	0	0x0B020A01

Note 1: Values of "-" indicate that the FIFO contains stale date from previous operations due to the way in which it was filled.

41.5.5 CRC Engine Interface

41.5.5.1 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 then starts shifting data, starting from the MSb 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 the VWORD bits decrement by one. Each word is copied out of the FIFO into a buffer register; the data is then shifted out of the buffer. The VWORD bits decrement by one when a FIFO location gets shifted completely to the CRC calculator. The serial shifter continues shifting until the VWORD bits reach zero, at which point, the CRCMPT bit becomes set to indicate that the FIFO is empty. If the CRCGO bit is reset manually during CRC calculation, then the CRC calculator will stop calculating until the CRCGO bit is set.

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 CRCDATL register before a read of the VWORD bits is done.

When the VWORD bits reach the maximum value for the configured value of the DWIDTH bits (4, 8, 16 or 32), the CRCFUL bit becomes set. When the VWORD bits reach zero, the CRCMPT bit becomes set. The FIFO is emptied and VWORD<4:0> are set to '00000' whenever CRCEN is '0'.

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. This means that for a given data width, it takes half that number of instructions for each word to complete the calculation. For example, it takes 16 cycles to calculate the CRC for a single word of 32-bit data.

41.5.5.2 NUMBER OF DATA BITS SHIFTED FROM FIFO

For a given value of DWIDTH<4:0>, it will take (DWIDTH<4:0> + 1) * VWORD<4:0> number of shift clock cycles to complete the CRC calculations.The number of data bits to be shifted depends upon the length of the DWIDTH selected. For example, if DWIDTH<4:0> = 5, then the length of the DWIDTH is 6 bits (DWIDTH<4:0> + 1) and 3 cycles are required to shift the data. 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 MSbs of each byte are don’t care bits. Similarly, for a 12-bit polynomial selection, the shifting starts from the 12th bit of the word, which is the MSb for this selection. The Most Significant 4 bits of each word are ignored.

41.5.5.3 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 CRC calculation to complete before reading the CRCWDAT registers. The CRC calculation is not complete when CRCMPT is set. This only indicates that the FIFO is empty. When CRCMPT is set, there is still one word of data to be shifted that remains in the shift buffer and (DWIDTH + 1)/2 cycles are left until the calculation finishes. When the calculation is complete, the module will clear the GO bit and set the CRC Interrupt Flag (if CRCISEL = `1`).

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 CRCWDATL and CRCWDATH registers. The CRCWDATL and CRCWDATH registers 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 CRCWDATL register, the CRCGO bit must be ‘`0`’.

41.5.6 Interrupt Operation

The module generates an interrupt that is configurable by the user for either of the two conditions. If CRCISEL is ‘`0`’, an interrupt is generated when the VWORD<4:0> bits make a transition from a value of ‘`1`’ to ‘`0`’. If CRCISEL is ‘`1`’, an interrupt will be generated after the CRC operation completes and the module sets the CRCGO bit to ‘`0`’. Manually setting CRCGO to ‘`0`’ will not generate an interrupt.

Table 41-2 in **Section 41.9 “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”**.

41.6 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. If the same calculation is implemented in software, it will consume more than a thousand instruction cycles, even for an optimized piece of code.

41.7 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.

41.7.1 Variations

The 32-bit programmable CRC module of PIC24F devices can be programmed to shift out either the MSb or LSb first. MSb first is a popular implementation as employed in XMODEM protocol. In one of the variations (CCITT protocol) for CRC calculation, the LSb is shifted out first. Discussions on all the variations are beyond the scope of this document, but several variations of CRC can be implemented using the 32-bit programmable CRC module in PIC24F devices.

The choice of the polynomial length, and the polynomial itself, are application dependent. Polynomial lengths of 5, 7, 8, 10, 12, 16 and 32 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.

41.7.2 Typical Operation

To use the module for a typical CRC calculation:

1. Set the CRCEN bit to enable the module.
2. Configure the module for desired operation:
   a) Program the desired polynomial using the CRCXORL and CRCXORH registers, and the PLEN<4:0> bits.
   b) Configure the data width and shift direction using the DWIDTH and LENDIAN bits.
   c) Select the desired interrupt mode using the CRCISEL bit.
3. Preload the FIFO by writing to the CRCDATL and CRCDATH registers until the CRCFUL bit is set or no data is left.
4. Clear old results by writing 00h to CRCWDATL and CRCWDATH. CRCWDATL can also be left unchanged to resume a previously halted calculation.
5. Set the CRCGO bit to start calculation.
6. Write remaining data into the FIFO as space becomes available.
7. When the calculation completes, CRCGO is automatically cleared. An interrupt will be generated if CRCISEL = 1.
8. Read CRCWDATL and CRCWDATH for the result of the calculation.

Example 41-1, Example 41-2 and Example 41-3 provide polynomial and data width examples.

Example 41-1: 16-Bit Polynomial with 8-Bit Data Width

```
main()
{
unsigned char *ptr;

ptr=(unsigned char *)&CRCDATL;        //For data width less than or equal to 8 bits,
//the user has to feed the input data through
//byte operations into the CRCDATL register.
    CRCCON1bits.CRCEN=1;              // enable CRC
CRCCON2bits.PLEN=0x0F;                // configure the polynomial width
CRCXORL=0x1021;                       // configure polynomial data
    CRCXORH=0x0000;//
CRCCON2bits.DWIDTH=0x08;              //configure the data width
    CRCCON1bits.LENDIAN=0;            // set the data shift direction
    CRC1CONbits.CRCISEL=1;            //select the interrupt source
    *ptr=0x01;                        // write data into the FIFO
    *ptr=0x02;
    *ptr=0x03;
    *ptr=0x04;
    *ptr=0x05;
    *ptr=0x06;
    *ptr=0x07;
*ptr=0x07;
Nop ();
    CRCCON1bits.CRCGO=1;              // start the CRC calculation
    while(IFS4bits.CRCIF!=1);         // check for end of calculation
    Nop ();
}
```

Example 41-2: 32-Bit Polynomial with 16-Bit Data Width

```
main()
{

    CRCCON1bits.CRCEN=1;              // enable CRC
CRCCON2bits.PLEN=0x1F;                // configure the polynomial width
CRCXORL= 0x1DB7;                      // configure polynomial data
    CRCXORH=0x04C1;
CRCCON2bits.DWIDTH=0x0F;              //configure the data width
    CRCCON1bits.LENDIAN=0;            // set the data shift direction
    CRC1CONbits.CRCISEL=1;            //select the interrupt source
    CRCDATL=0x0201;
    CRCDATL=0x0403;
    CRCDATL=0x0605;
    CRCDATL=0x0807;
    Nop();
    CRCCON1bits.CRCGO=1;              // start the CRC calculation
    while(IFS4bits.CRCIF!=1);         // check for end of calculation
    Nop ();
}
```

Example 41-3: 32-Bit Polynomial with 32-Bit Data Width

```
main()
{
    CRCCON1bits.CRCEN=1;                // enable CRC
CRCCON2bits.PLEN=0x1F;                  // configure the polynomial width
CRCXORL= 0x1DB7;                        // configure polynomial data
    CRCXORH=0x04C1;
CRCCON2bits.DWIDTH=0x1F;                //configure the data width
    CRCCON1bits.LENDIAN=0;              // set the data shift direction
    CRC1CONbits.CRCISEL=1;              //select the interrupt source
CRCDATL=0x0201;
    CRCDATH=0x0403;
    CRCDATL=0x0605;
    CRCDATH=0x0807;
    Nop();
    CRCCON1bits.CRCGO=1;                // start the CRC calculation
    while(IFS4bits.CRCIF!=1);           // check for end of calculation
    Nop ();
}
```

41.8 OPERATION IN POWER SAVE MODES

41.8.1 Sleep Mode

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

41.8.2 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.

41.9 REGISTER MAPS

A summary of the Special Function Registers associated with the PIC24F programmable CRC module is provided in Table 41-2.

Table 41-2: 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
CRCCON1	CRCEN	—	CSIDL	VWORD4	VWORD3	VWORD2	VWORD1	VWORD0	CRCFUL	CRCMPT	CRCISEL	CRCGO	LENDIAN	—	—	—	`0040`
CRCCON2	—	—	—	DWIDTH4	DWIDTH3	DWIDTH2	DWIDTH1	DWIDTH0	—	—	—	PLEN4	PLEN3	PLEN2	PLEN1	PLEN0	
CRCXORL	X15	X14	X13	X12	X11	X10	X9	X8	X7	X6	X5	X4	X3	X2	X1	—	`0000`
CRCXORH	X31	X30	X29	X28	X27	X26	X25	X24	X23	X22	X21	X20	X19	X18	X17	X16	
CRCDATL	DATA15	DATA14	DATA13	DATA12	DATA11	DATA10	DATA9	DATA8	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0	`0000`
CRCDATH	DATA31	DATA30	DATA29	DATA28	DATA27	DATA26	DATA25	DATA24	DATA23	DATA22	DATA21	DATA20	DATA19	DATA18	DATA17	DATA16	
CRCWDATL	SDATA15	SDATA14	SDATA13	SDATA12	SDATA11	SDATA10	SDATA9	SDATA8	SDATA7	SDATA6	SDATA5	SDATA4	SDATA3	SDATA2	SDATA1	SDATA0	`0000`
CRCWDATH	SDATA31	SDATA30	SDATA29	SDATA28	SDATA27	SDATA26	SDATA25	SDATA24	SDATA23	SDATA22	SDATA21	SDATA20	SDATA19	SDATA18	SDATA17	SDATA16	
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: Refer to the device data sheet for specific memory map details.

41.10 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 Reset with Programmable Brown-out Reset 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.

41.11 REVISION HISTORY

Revision A (April 2009)

This is the initial released revision of this document.

NOTES: