



32-Bit Programmable Cyclic Redundancy Check (CRC)

HIGHLIGHTS

This section of the manual contains the following major topics:

1.0	Introduction	2
2.0	Module Overview	3
3.0	CRC Registers	4
4.0	CRC Engine	10
5.0	Control Logic	11
6.0	Advantages of Programmable CRC Module	20
7.0	Application of CRC Module	20
8.0	Operation in Power Save Modes	31
9.0	Register Maps	32
10.0	Related Application Notes	33
11.0	Revision History	34

1.0 INTRODUCTION

The 32-bit programmable Cyclic Redundancy Check (CRC) module in dsPIC33/PIC24 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, such as 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 mod 2. 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 1-1:

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

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 dsPIC33/PIC24 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:

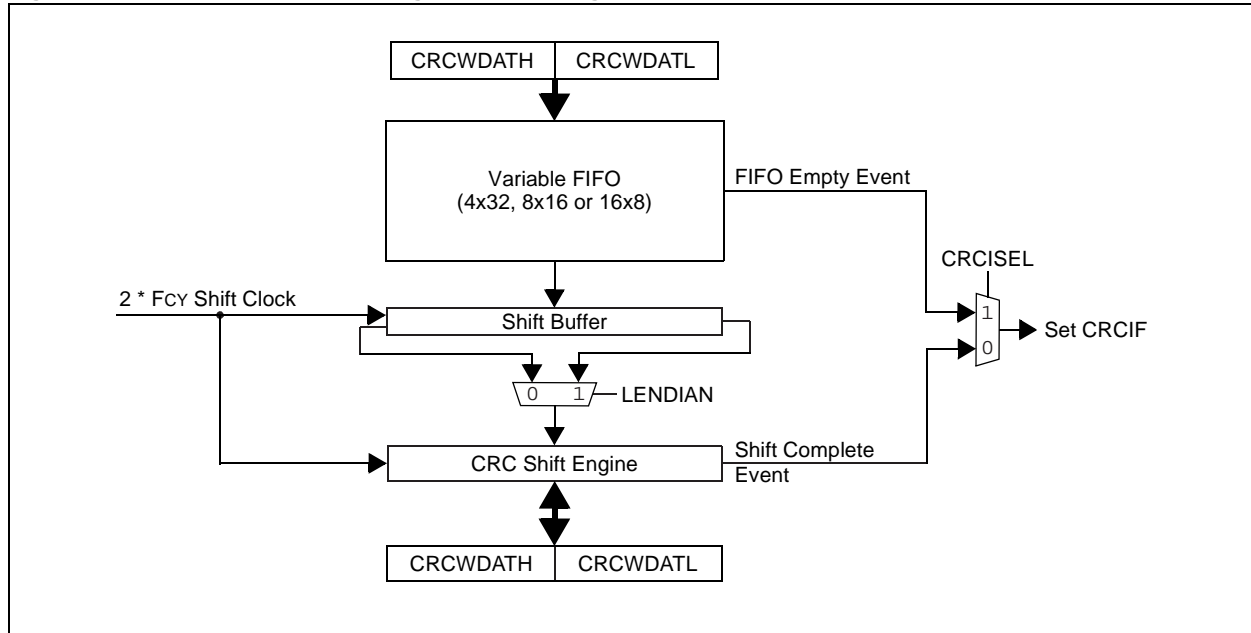
- User-programmable CRC polynomial equation, up to 32 bits
- Programmable shift direction (little or big endian)
- Independent data and polynomial lengths
- Configurable interrupt output
- Data FIFO

32-Bit Programmable Cyclic Redundancy Check (CRC)

2.0 MODULE OVERVIEW

The programmable CRC generator module in dsPIC33/PIC24 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 2-1](#).

Figure 2-1: Simplified Block Diagram of the Programmable CRC Generator



3.0 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 dsPIC33/PIC24 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 3-1](#)) and CRCCON2 ([Register 3-2](#)) registers control the operation of the module and configure various settings. The CRCXORL/H registers ([Register 3-3](#) and [Register 3-4](#)) select the polynomial terms to be used in the CRC equation. The CRCDATL/H and CRCWDATL/H registers are each register pairs that serve as buffers for the double-word input data and CRC processed output, respectively.

32-Bit Programmable Cyclic Redundancy Check (CRC)

Register 3-1: CRCCON1: CRC Control Register 1

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 = Discontinues module operation when device enters Idle mode
 0 = Continues 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:** CRC FIFO Full bit
 1 = FIFO is full
 0 = FIFO is not full
- bit 6 **CRCMPT:** CRC 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 is still shifted through CRC
 0 = Interrupt on shift complete (FIFO is empty and no data is shifted from the shift buffer)
- bit 4 **CRCGO:** Start CRC bit
 1 = Start CRC serial shifter; clearing the bit aborts shifting
 0 = CRC serial shifter is turned off
- bit 3 **LENDIAN:** Data Word Little Endian Configuration bit
 1 = Data word is shifted into 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'

dsPIC33/PIC24 Family Reference Manual

Register 3-2: CRCCON2: CRC Control Register 2

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).

32-Bit Programmable Cyclic Redundancy Check (CRC)

Register 3-3: 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 3-4: 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 15-0 **X<31:16>**: XOR of Polynomial Term x^n Enable bits

dsPIC33/PIC24 Family Reference Manual

Register 3-5: CRCDATL: CRC Data 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
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 3-6: CRCDATAH: CRC Data 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
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'.

32-Bit Programmable Cyclic Redundancy Check (CRC)

Register 3-7: CRCWDATL: CRC Shift 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
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 3-8: CRCWDATH: CRC Shift 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
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 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.

4.0 CRC ENGINE

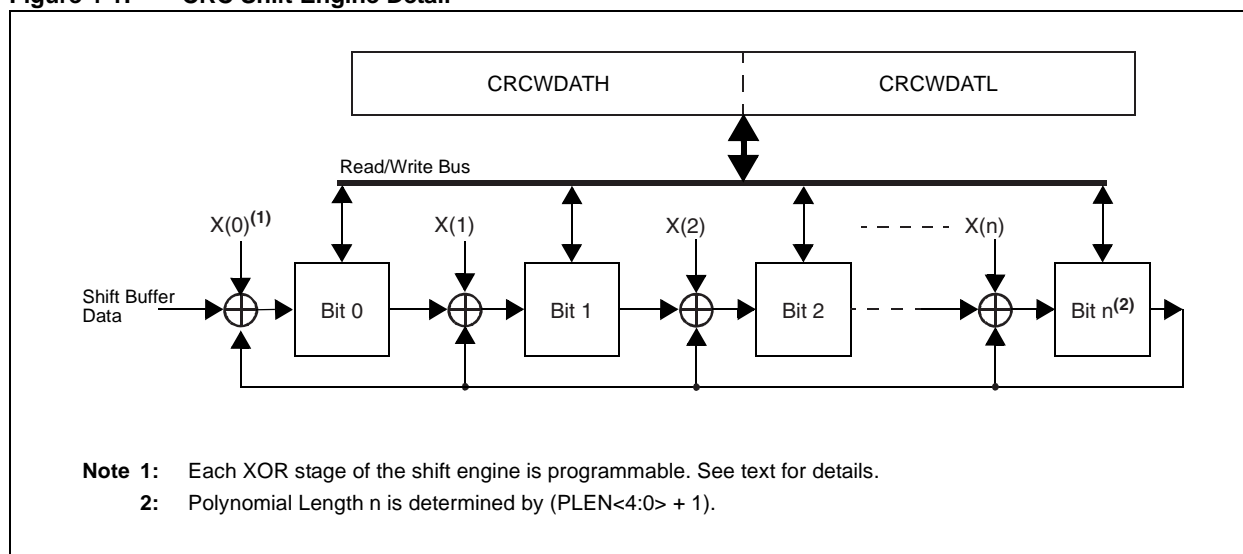
4.1 Generic CRC Engine

The CRC engine is a serial shifting CRC calculator with feedforward and feedback points, configurable through multiplexer settings. A simple version of the CRC shift engine is shown in Figure 4-1.

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<15:1> 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 4-1: CRC Shift Engine Detail



32-Bit Programmable Cyclic Redundancy Check (CRC)

5.0 CONTROL LOGIC

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 5-1:

$$\begin{aligned} & x^{16} + x^{12} + x^5 + 1 \\ & \text{and} \\ & x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \end{aligned}$$

To program this polynomial into the CRC generator, set the register bits as shown in [Table 5-1](#).

Table 5-1: 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 0000	0000 0100 1100 0001
X<15:1>	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. The Most Significant bit (MSb) of the polynomial does not affect the calculation and can be set to zero.

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<4:0> 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.

5.3 Data Shift Direction

The LENDIAN bit (CRCCON1<3>) is used to control the shift direction. By default, the CRC module will shift data through the engine, MSb first (LENDIAN = 0). Setting LENDIAN to '1' causes the CRC module 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.

The PIC24 and dsPIC33 are little endian devices. When the CRC module is configured for the big endian (LENDIAN = 0), the input data bytes and words must be swapped in the application code before loading them into the CRCDAT registers.

5.4 FIFO

The module incorporates a FIFO that works with a variable data width. The data width is defined by the DWIDTH<4:0> bits (CRCCON2<12:8>). It can be configured to any value, between 1 and 32 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 CRCDAT registers. Reading the CRCDAT registers always returns zero.

Filling the FIFO with Less Than or Equal to 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. When a single 16-bit word is loaded into the CRCDATL register, the lower byte is written into the FIFO first and the higher byte is written next.

For example, if DWIDTH<4:0> is five, then the size of the data is DWIDTH<4:0> + 1 or six. The data is written as a whole byte; the two unused upper bits are ignored by the module. Once the data byte is written into the CRCDATL register, the value of the VWORD<4:0> bits (CRCCON1<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 is loaded on a write to the CRCDATL register. Any write to the CRCDATH register will be ignored. The value of the VWORD<4:0> bits is incremented 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<4:0> bits by one. Writing the lower word into the CRCDATL register must be done before writing the upper word into the CRCDATH register.

To accommodate the, MSb first shift method (LENDIAN = 0), byte and word swapping must be done in software when filling the FIFO. Pictorial descriptions of FIFO for different data widths are shown in the [Figure 5-1](#), [Figure 5-2](#) and [Figure 5-3](#).

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.

When all shifts are done (the FIFO is empty and the CRC shift engine is Idle), it is possible to change the FIFO width (DWIDTH<4:0> bits) without any information loss or CRC result damage.

32-Bit Programmable Cyclic Redundancy Check (CRC)

Figure 5-1: Filling the FIFO Word Width ≤ 8 Bits

Initial Conditions:

W1 = 0x0201

W2 = 0x0403

W3 = 0x0005

W4 = 0x0006

W5 = 0x0007

W6 = 0x0008

Code Executed to Fill the FIFO:

MOV.W W1, CRCDATL ; low byte is written first, then high byte

MOV.W W2, CRCDATL ; low byte is written first, then high byte

MOV.B W3, CRCDATL ; low byte only

MOV.B W4, CRCDATL ; low byte only

MOV.B W5, CRCDATL ; low byte only

MOV.B W6, CRCDATL ; low byte only

Resulting FIFO:

Write	Location	Value
W6L	7	0x08
W5L	6	0x07
W4L	5	0x06
W3L	4	0x05
W2H	3	0x04
W2L	2	0x03
W1H	1	0x02
W1L	0	0x01

Figure 5-2: Filling the FIFO Word Width > 8 Bits and ≤ 16 Bits

Initial Conditions:	Code Executed to Fill the FIFO:
W1 = 0x0A01	MOV.W W1, CRCDATL ; word
W2 = 0x0B02	MOV.B W2, CRCDATL ; low byte
W3 = 0x0C03	MOV.W W3, CRCDATL ; word
W4 = 0x0D04	MOV.W W4, CRCDATL ; word

Resulting FIFO:

Write	Location	Value
W4	3	0x0D04
W3	2	0x0C03
W2L	1	0x-02 ⁽¹⁾
W1	0	0x0A01

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

Figure 5-3: Filling the FIFO Word Width > 16 Bits and ≤ 32 Bits

Initial Conditions:

W1 = 0x0A01

W2 = 0x0B02

W3 = 0x0C03

W4 = 0x0D04

Code Executed to Fill the FIFO:

MOV.W W1, CRCDATL ; low word

MOV.W W2, CRCDATH ; high word, write to FIFO occurs here

MOV.W W3, CRCDATL ; low word

MOV.W W4, CRCDATH ; high word, write to FIFO occurs here

Word Width > 16 Bits and ≤ 32 Bits:

Write	Location	Value
W4, W3	1	0x0D040C03
W2, W1	0	0x0B020A01

5.5 CRC Engine Interface

5.5.1 FIFO TO CRC SHIFT ENGINE

To start moving the data from the FIFO to the CRC shift buffer, the CRCGO bit (CRCCON1<4>) must be set. The serial shifter starts shifting data from the shift buffer to the CRC shift engine, starting from the MSb first for LENDIAN = 0, and LSb first for LENDIAN = 1, when CRCGO = 1 and the value of VWORD<4:0> is greater than zero. If the CRCFUL bit was set earlier, then it is cleared when the VWORDx bits decrement by one. The VWORD<4:0> bits decrement by one when a FIFO location is moved to the shift buffer. The serial shifter continues shifting until the VWORD<4:0> 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 shift engine will stop calculating until the CRCGO bit is set.

The application can write into the FIFO while the shift operation is in progress. The CRCFUL bit should be monitored. If the CRCFUL bit is not set, another word can be written into the FIFO. At least one instruction cycle must pass after a write to the CRCDAT registers, before a read of the valid value of the VWORD<4:0> bits.

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

The frequency of the CRC shift clock is twice that of the CPU 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.

5.5.2 NUMBER OF INSTRUCTION CYCLES TO SHIFT DATA

The data from FIFO goes to the shift buffer. It takes 2 instruction cycles to start moving the data words from FIFO to the shift buffer. The data from the shift buffer is then shifted to the CRC shift engine. For a given value of the DWIDTH<4:0> bits, it will take $(\text{DWIDTH}<4:0> + 1)/2$ instruction cycles to completely move the data from the shift buffer to the CRC shift engine. For example, if $\text{DWIDTH}<4:0> = 5$, then the data length is 6 bits ($\text{DWIDTH}<4:0> + 1$) and 3 cycles are required to shift the data. In this case, only 6 bits of a byte are shifted out. The two MSBs of each byte are don't care bits. Similarly, for a 12-bit polynomial selection, the Most Significant 4 bits of each word are ignored.

5.5.3 CRC INITIAL VALUE

The direct write path to the CRC Shift registers is provided through the CRC write bus. This path is accessed by the CPU through the CRCWDATL and CRCWDATH registers. These registers can be loaded with a desired CRC initial value prior to the start of the calculations. The CRC initial value must be in non-direct form. The non-direct form is a value for which the CRC is equal to the desired CRC initial value (direct initial value). For example, if the application uses CRC-32 polynomial, 0x04C11DB7, and must start the calculations from the CRC direct initial value, 0xFFFFFFFF, then the non-direct value, 0x46AF6449, must be loaded in the CRCWDATL and CRCWDATH registers (the CRC of this non-direct value, 0x46AF6449, is 0xFFFFFFFF). When the non-direct initial value is written into the shift engine using the CRCWDAT registers, it will be converted by the CRC module to the direct initial value after $(\text{PLEN}<4:0> + 1)/2$ instruction cycles.

Note: The write to CRCWDAT registers clears/resets the Shift Buffer.

Usually the CRC calculation starts from the same initial value every time. In this case, the non-direct initial value can be found just once and then can be defined as a constant in the application code.

Note: The CRC non-direct initial value of zero is zero.

32-Bit Programmable Cyclic Redundancy Check (CRC)

[Example 5-1](#) shows a possible software routine to get the non-direct initial value from the direct initial value.

Example 5-1: Software Routine to Calculate the Non-Direct Initial Value

```
unsigned long CalculateNonDirectSeed(
unsigned long seed,           // direct CRC initial value
unsigned long polynomial,    // polynomial
unsigned char polynomialOrder) // polynomial order
{
    unsigned char lsb;
    unsigned char i;
    unsigned long msbmask;

    msbmask = ((unsigned long)1)<<(polynomialOrder-1);

    for (i=0; i<polynomialOrder; i++) {
        lsb = seed & 1;
        if (lsb) seed ^= polynomial;
        seed >>= 1;
        if (lsb) seed |= msbmask;
    }

    return seed;               // return the non-direct CRC initial value
}
```

The CRC module can be used to get the non-direct initial value. To do this, the following steps should be completed:

1. Enable the CRC module (CRCEN = 1) and shifts (CRCGO = 1).
2. Shift the polynomial value right by one.
3. Reverse the bit order of the shifted polynomial value.
4. Write this result in the CRCXOR registers.
5. Set data width and polynomial length (DWIDTH<4:0> and PLEN<4:0> bits) to the polynomial order (length).
6. Reverse the bit order of the desired direct initial value.
7. Write the reversed initial value in CRCWDAT registers.
8. Write a dummy data to the CRCDAT registers and wait 2 instruction cycles to move the data from the FIFO to the shift buffer, and (PLEN<4:0> + 1)/2 instruction cycles to shift out the result;

OR

Clear the CRC Interrupt Selection bit (CRCISEL = 0) to get the interrupt when shifts from the shift buffer are done, clear CRC interrupt flag, write a dummy data in the CRCDAT registers and wait for the CRC interrupt flag to set.

9. Read the value from CRCWDAT registers.
10. Reverse the bit order of the read result; it will give the final non-direct initial value.

dsPIC33/PIC24 Family Reference Manual

Example 5-2 explains the steps described above.

Example 5-2: Routine to Calculate the Non-Direct Initial Value Using the CRC Module

```
unsigned long CalculateNonDirectSeed(unsigned long seed, // direct CRC initial value
unsigned long polynomial, // polynomial
unsigned char polynomialOrder) // polynomial order (valid values are
// 8, 16, 32 bits)
{
    CRCCON1 = 0;
    CRCCON2 = 0;

    CRCCON1bits.CRCEN = 1; // enable CRC
    CRCCON1bits.CRCISEL = 0; // interrupt when all shifts are done
    CRCCON2bits.DWIDTH = polynomialOrder-1; // data width
    CRCCON2bits.PLEN = polynomialOrder-1; // polynomial length
    CRCCON1bits.CRCSGO = 1; // start CRC calculation

    polynomial >>= 1; // shift the polynomial right

    polynomial = ReverseBitOrder(polynomial, polynomialOrder); // reverse bits order of the
// polynomial
    CRCXORL = (unsigned short)(polynomial&0x0000FFFF); // set the reversed polynomial
    CRCXORH = (unsigned short)(polynomial>>16);
    seed = ReverseBitOrder(seed, polynomialOrder); // reverse bits order of the seed value
    CRCWDATL = (unsigned short)(seed&0x0000FFFF); // set seed value
    CRCWDATH = (unsigned short)(seed>>16);

    _CRCIF = 0; // clear interrupt flag
    switch(polynomialOrder) // load dummy data to shift out the
// seed result
    {
        case 8:
            *((unsigned char*)&CRCDATL) = 0; // load byte
            while(!_CRCIF); // wait until shifts are done
            seed = CRCWDATL&0x00ff; // read reversed seed
        case 16:
            CRCDATL = 0; // load short
            while(!_CRCIF); // wait until shifts are done
            seed = CRCWDATL; // read reversed seed
            break;
        case 32:
            // load long
            CRCDATL = 0;
            CRCWDATH = 0;
            while(!_CRCIF); // wait for shifts are done
            seed = ((unsigned long)CRCWDATH<<16)|CRCWDATL; // read reversed seed
            break;
        default:
            ;
    }

    seed = ReverseBitOrder(seed, polynomialOrder); // reverse the bit order to get the
// non-direct seed
    return seed; // return the non-direct CRC initial value
}
```


32-Bit Programmable Cyclic Redundancy Check (CRC)

Example 5-2: Routine to Calculate the Non-Direct Initial Value Using the CRC Module (Continued)

```
// WHERE THE FUNCTION TO REVERSE THE BIT ORDER CAN BE

unsigned long ReverseBitOrder(unsigned long data,      // input data
                             unsigned char numberOfBits) // width of the input data,
                                                         // valid values are 8,16,32 bits
{
    unsigned long maskin = 0;
    unsigned long maskout = 0;
    unsigned long result = 0;
    unsigned char i;

    switch(numberOfBits)
    {
        case 8:
            maskin = 0x80;
            maskout = 0x01;
            break;
        case 16:
            maskin = 0x8000;
            maskout = 0x0001;
            break;
        case 32:
            maskin = 0x80000000;
            maskout = 0x00000001;
            break;
        default:
            ;
    }

    for(i=0; i<numberOfBits; i++)
    {
        if(data&maskin){
            result |= maskout;
        }
        maskin >>= 1;
        maskout <<= 1;
    }

    return result;
}
```

To continue calculations of the full data message in the applications where the intermediate CRC sums must be read in the middle of the calculations, the non-direct value must be calculated and set to the CRCWDAT registers again. In this case, the CRC direct initial value will be an intermediate CRC result read.

5.5.4 CRC RESULT

The CRC module requires extra $(\text{PLEN}_{<4:0>} + 1)/2$ instruction cycles to finish the calculations. To generate these additional cycles, the dummy data, with the width equal to the polynomial order (length), must be loaded into the CRCDAT registers. After the shifts are finished, the final CRC result can be read from the CRCWDAT registers (the CRCWDAT registers provide the direct access to the CRC Shift register).

After all data is loaded into the CRC module, the following steps should be done to get the final CRC result. If the data width ($\text{DWIDTH}_{<4:0>}$ bits) is more than the polynomial length ($\text{PLEN}_{<4:0>}$ bits):

1. Wait for the data FIFO to empty (CRCMPT bit is set).
2. Wait $(\text{DWIDTH}_{<4:0>} + 1)/2$ instruction cycles to make sure that shifts from the shift buffer are finished.
3. Change the data width to the polynomial length ($\text{DWIDTH}_{<4:0>} = \text{PLEN}_{<4:0>}$).
4. Write one dummy data word to the CRCDAT registers.
5. Wait 2 instruction cycles to move the data from the FIFO to the shift buffer and $(\text{PLEN}_{<4:0>} + 1)/2$ instruction cycles to shift out the result;
OR
Clear the CRC Interrupt Selection bit ($\text{CRCISEL} = 0$) to get the interrupt when all shifts are done. Clear the CRC interrupt flag. Write dummy data in the CRCDAT registers and wait until the CRC interrupt flag is set.
6. Read the final CRC result from the CRCWDAT registers.
7. Restore the data width ($\text{DWIDTH}_{<4:0>}$ bits) for further calculations (OPTIONAL).

If the data width ($\text{DWIDTH}_{<4:0>}$ bits) is equal to, or less than, the polynomial length ($\text{PLEN}_{<4:0>}$ bits), the procedure to get the result can be different:

1. Clear the CRC Interrupt Selection bit ($\text{CRCISEL} = 0$) to get the interrupt when all shifts are done.
2. Suspend the calculation by setting $\text{CRCGO} = 0$.
3. Clear the CRC interrupt flag.
4. Write the dummy data with the total data length equal to the polynomial length in the CRCDAT registers.
5. Resume the calculation by setting $\text{CRCGO} = 1$.
6. Wait until the CRC interrupt flag is set.
7. Read the final CRC result from the CRCWDAT registers.

When the CRC result is achieved, the CRC non-direct initial value should be written again into the CRCWDAT registers to clear/reset the shift buffer from the previously loaded dummy data to start a new calculation.

32-Bit Programmable Cyclic Redundancy Check (CRC)

Example 5-3 shows the steps described above for the polynomial orders of 8, 16 and 32 bits.

Example 5-3: Routine to Get the Final CRC Result

```
unsigned long GetCRC(unsigned char polynomialOrder,    // valid values are 8,16,32
unsigned char  currentDataWidth)                    // valid values are 8,16,32
{
    unsigned long crc = 0;

    while(!CRCCON1bits.CRCMPT);                      // wait until data FIFO is empty

    asm volatile ("repeat %0\n nop" : : "r"(currentDataWidth>>1)); // wait until previous data
                                                                // shifts are done
    CRCCON2bits.DWIDTH = polynomialOrder-1;          // set data width to polynomial
                                                                // length
    CRCCON1bits.CRCISEL = 0;                          // interrupt when all shifts are done

    _CRCIF = 0;                                       // clear interrupt flag

    switch(polynomialOrder)
    {
        case 8:                                       // polynomial length is 8 bits
            *((unsigned char*)&CRCDATL) = 0;          // load byte
            while(!_CRCIF);                          // wait until shifts are done
            crc = CRCWDATL&0x00ff;                   // get crc
            break;
        case 16:                                     // polynomial length is 16 bits
            CRCDATL = 0;                             // load short
            while(!_CRCIF);                          // wait until shifts are done
            crc = CRCWDATL;                           // get crc
            break;
        case 32:                                     // polynomial length is 32 bits
            CRCDATL = 0;                             // load long
            CRCDATH = 0;
            while(!_CRCIF);                          // wait until shifts are done
            crc = ((unsigned long)CRCWDATH<<16)|CRCWDATL; // get crc
            break;
        default:
            ;
    }
    CRCCON2bits.DWIDTH = currentDataWidth-1;         // restore data width for further
                                                                // calculations

    return crc;                                       // return the final CRC value
}
```

5.6 Interrupt Operation

The module generates an interrupt that is configurable by the user for either of the two conditions. If CRCISEL is '1', an interrupt is generated when the VWORD<4:0> bits make a transition from a value of '1' to '0'. If CRCISEL is '0', an interrupt will be generated when the FIFO is empty and shifts from the shift buffer are finished.

The table in **Section 9.0 "Register Maps"** details the Interrupt register associated with the CRC module. For more details on interrupts and interrupt priority settings, refer to the **"Interrupts"** section in the *"dsPIC33/PIC24 Family Reference Manual"*.

6.0 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 the dsPIC33/PIC24 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 dsPIC33/PIC24 instruction clock cycle. For example, the CRC hardware engine takes only about 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.

7.0 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.

7.1 Variations

The 32-bit programmable CRC module of the dsPIC33/PIC24 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 dsPIC33/PIC24 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 following sections explain the recommended step-by-step procedure for CRC calculation. 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.

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 CRCXOR registers and PLEN<4:0> bits.
 - b) Configure the data width and shift direction using the DWIDTH<4:0> and LENDIAN bits.
3. Set the CRCGO bit to start the calculations.
4. Set the desired CRC non-direct initial value by writing to the CRCWDAT registers.
5. Load all data into the FIFO by writing to the CRCDAT registers as space becomes available (the CRCFUL bit must be zero before the next data loading).
6. Wait until the data FIFO is empty (CRCMPT bit is set).
7. Read the CRC result as described in [Section 5.5.4 “CRC Result”](#).

32-Bit Programmable Cyclic Redundancy Check (CRC)

[Example 7-1](#), [Example 7-2](#), [Example 7-3](#), [Example 7-4](#) and [Example 7-5](#) provide examples for different combinations between polynomial length, data width and shift direction.

Example 7-1: 8-Bit Polynomial with 32-Bit Data Width When MSb is Shifted First (CRC SMBus)

```
// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(2))) message[] = {'1','2','3','4','5','6','7','8'};

volatile unsigned char crcResultCRCSMBUS = 0;

int main (void)
{
    unsigned short* pointer;
    unsigned short length;
    unsigned short data_high;
    unsigned short data_low;

    //////////////////////////////////////
    // standard CRC-SMBUS
    //////////////////////////////////////

#define CRCSMBUS_POLYNOMIAL ((unsigned short)0x0007)
#define CRCSMBUS_SEED_VALUE ((unsigned short)0x0000) // non-direct of 0x00

    CRCCON1 = 0;
    CRCCON2 = 0;

    CRCCON1bits.CRCEN = 1; // enable CRC
    CRCCON1bits.LENDIAN = 0; // big endian
    CRCCON1bits.CRCISEL = 0; // interrupt when all shifts are done
    CRCCON2bits.DWIDTH = 32-1; // 32-bit data width
    CRCCON2bits.PLEN = 8-1; // 8-bit polynomial order
    CRCCON1bits.CRCGO = 1; // start CRC calculation

    CRCXORL = CRCSMBUS_POLYNOMIAL; // set polynomial
    CRCXORH = 0;

    CRCWDATL = CRCSMBUS_SEED_VALUE; // set initial value
    CRCWDATH = 0;

    pointer = (unsigned short*)message; // calculate CRC
    length = sizeof(message)/sizeof(unsigned long);
    while(length--)
    {
```

dsPIC33/PIC24 Family Reference Manual

Example 7-1: 8-Bit Polynomial with 32-Bit Data Width When MSb is Shifted First (CRC SMBus) (Continued)

```
while(CRCCON1bits.CRCFUL);           // wait if FIFO is full

    data_low  = *pointer++;           // load from little endian
    data_high = *pointer++;

    asm      volatile ("swap %0" : "+r"(data_low)); // swap bytes for big endian
    asm      volatile ("swap %0" : "+r"(data_high));

    CRCDATL = data_high;              // 32-bit word access to FIFO
    CRCDATH = data_low;              // swap 16-bit words for big endian
}

while(!CRCCON1bits.CRCMPT);          // wait until FIFO is empty

asm      volatile ("repeat #16-#2\n nop"); // wait until previous data shifts are done
                                           // 16 cycles maximum for 32-bit data width

CRCCON2bits.DWIDTH = 8-1;            // 8-bit
                                           // switch data width to polynomial length

_CRCIF = 0;                          // clear the interrupt flag
                                           // dummy data to shift out the CRC result

*((unsigned char*)&CRCDATL) = 0;     // byte access to FIFO

while(!_CRCIF);                      // wait until shifts are done
crcResultCRCMBUS = CRCWDATL&0x00ff;  // get CRC result (must be 0xC7)

while(1);

return 1;
}
```

32-Bit Programmable Cyclic Redundancy Check (CRC)

Example 7-2: 16-Bit Polynomial with 16-Bit Data Width When LSb is Shifted First (CRC 16)

```
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};

volatile unsigned short crcResultCRC16 = 0;

int main (void)
{
    unsigned short* pointer;
    unsigned short length;
    unsigned short data;

    //////////////////////////////////////
    // standard CRC-16
    //////////////////////////////////////
    #define CRC16_POLYNOMIAL ((unsigned short)0x8005)
    #define CRC16_SEED_VALUE ((unsigned short)0x0000) // non-direct of 0x0000

    CRCCON1 = 0;
    CRCCON2 = 0;

    CRCCON1bits.CRCEN = 1; // enable CRC
    CRCCON1bits.CRCISEL = 0; // interrupt when all shifts are done
    CRCCON1bits.LENDIAN = 1; // little endian
    CRCCON2bits.DWIDTH = 16-1; // 16-bit data width
    CRCCON2bits.PLEN = 16-1; // 16-bit polynomial order
    CRCCON1bits.CRCGO = 1; // start CRC calculation

    CRCXORL = CRC16_POLYNOMIAL; // set polynomial
    CRCXORH = 0;

    CRCWDATL = CRC16_SEED_VALUE; // set initial value
    CRCWDATH = 0;

    pointer = (unsigned short*)message; // calculate CRC
    length = sizeof(message)/sizeof(unsigned short);

    while(length--)
    {
        while(CRCCON1bits.CRCFUL); // wait if FIFO is full

        data = *pointer++; // load data

        CRCDATL = data; // 16-bit word access to FIFO
    }

    while(CRCCON1bits.CRCFUL); // wait if FIFO is full
```

dsPIC33/PIC24 Family Reference Manual

Example 7-2: 16-Bit Polynomial with 16-Bit Data Width When LSb is Shifted First (CRC 16) (Continued)

```
CRCCON1bits.CRCGO = 0;           // suspend CRC calculation to clear interrupt flag

_CRCIF = 0;                       // clear interrupt flag

CRCDATL = 0;                      // load dummy data to shift out the CRC result
                                // data width must be equal to polynomial length

CRCCON1bits.CRCGO = 1;           // resume CRC calculation

while(!_CRCIF);                  // wait until shifts are done

crcResultCRC16 = CRCWDATL;        // get CRC result (must be 0xE716)

while(1);

return 1;

}
```


32-Bit Programmable Cyclic Redundancy Check (CRC)

Example 7-3: 16-Bit Polynomial with 16-Bit Data Width When MSb is Shifted First (CRC CCITT)

```
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};

volatile unsigned short crcResultCRCCCITT = 0;

int main (void)
{
    unsigned short* pointer;
    unsigned short length;
    unsigned short data;

    ////////////////////////////////////////
    // standard CRC-CCITT
    ////////////////////////////////////////
    #define CRCCCITT_POLYNOMIAL ((unsigned short)0x1021)
    #define CRCCCITT_SEED_VALUE ((unsigned short)0x84CF) // non-direct of 0xffff

    CRCCON1 = 0;
    CRCCON2 = 0;

    CRCCON1bits.CRCEN = 1; // enable CRC
    CRCCON1bits.CRCISEL = 0; // interrupt when all shifts are done
    CRCCON1bits.LENDIAN = 0; // big endian
    CRCCON2bits.DWIDTH = 16-1; // 16-bit data width
    CRCCON2bits.PLEN = 16-1; // 16-bit polynomial order
    CRCCON1bits.CRCGO = 1; // start CRC calculation

    CRCXORL = CRCCCITT_POLYNOMIAL; // set polynomial
    CRCXORH = 0;

    CRCWDATL = CRCCCITT_SEED_VALUE; // set initial value
    CRCWDATH = 0;

    pointer = (unsigned short*)message; // calculate CRC
    length = sizeof(message)/sizeof(unsigned short);

    while(length--)
    {
        while(CRCCON1bits.CRCFUL); // wait if FIFO is full

        data = *pointer++; // load data

        asm volatile ("swap %0" : "+r"(data)); // swap bytes for big endian
    }
}
```

dsPIC33/PIC24 Family Reference Manual

Example 7-3: 16-Bit Polynomial with 16-Bit Data Width When MSb is Shifted First (CRC CCITT) (Continued)

```
CRCDATL = data;                // 16 bit word access to FIFO
}

while(CRCCON1bits.CRCFUL);      // wait if FIFO is full

CRCCON1bits.CRCGO = 0;          // suspend CRC calculation to clear interrupt flag

_CRCIF = 0;                     // clear interrupt flag

CRCDATL = 0;                    // load dummy data to shift out the CRC result
                                // data width must be equal to polynomial length

CRCCON1bits.CRCGO = 1;          // resume CRC calculation

while(!_CRCIF);                // wait until shifts are done

crcResultCRCCCITT = CRCWDATL;   // get CRC result (must be 0x9B4D)

while(1);

return 1;
}
```

32-Bit Programmable Cyclic Redundancy Check (CRC)

Example 7-4: 32-Bit Polynomial with 32-Bit Data Width When LSb is Shifted First (CRC 32)

```
// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};

// function to reverse the bit order (OPTIONAL)
unsigned long ReverseBitOrder(unsigned long data);

volatile unsigned long crcResultCRC32 = 0;

int main(void)
{
    unsigned short* pointer;
    unsigned short length;
    ////////////////////////////////////////////////////
    // standard CRC-32
    ////////////////////////////////////////////////////
#define CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define CRC32_SEED_VALUE ((unsigned long)0x46AF6449) // non-direct of 0xffffffff

    CRCCON1 = 0;
    CRCCON2 = 0;

    CRCCON1bits.CRCEEN = 1; // enable CRC
    CRCCON1bits.CRCISEL = 0; // interrupt when all shifts are done
    CRCCON1bits.LENDIAN = 1; // little endian
    CRCCON2bits.DWIDTH = 32-1; // 32-bit data width
    CRCCON2bits.PLEN = 32-1; // 32-bit polynomial order
    CRCCON1bits.CRCGO = 1; // start CRC calculation

    CRCXORL = CRC32_POLYNOMIAL&0x0000ffff; // set polynomial
    CRCXORH = CRC32_POLYNOMIAL>>16;

    CRCWDATL = CRC32_SEED_VALUE&0x0000ffff; // set initial value
    CRCWDATH = CRC32_SEED_VALUE>>16;

    pointer = (unsigned short*)message; // calculate CRC
    length = sizeof(message)/sizeof(unsigned short);
    while(length--)
    {
        while(CRCCON1bits.CRCFUL); // wait if FIFO is full

        CRCDATL = *pointer++; // 32-bit word access to FIFO
                               // must be written first
        CRCDATH = *pointer++; // must be written last
    }

    while(CRCCON1bits.CRCFUL); // wait if FIFO is full
```

dsPIC33/PIC24 Family Reference Manual

Example 7-4: 32-Bit Polynomial with 32-Bit Data Width When LSb is Shifted First (CRC 32) (Continued)

```
CRCCON1bits.CRCGO = 0;           // suspend CRC calculation to clear interrupt flag

_CRCIF = 0;                       // clear interrupt flag

CRCDATL = 0;                      // dummy data to shift out the CRC result
CRCDATH = 0;

CRCCON1bits.CRCGO = 1;           // resume CRC calculation

while(!_CRCIF);                  // wait until shifts are done

crcResultCRC32 = ((unsigned long)CRCWDATH<<16)|CRCWDATL;    // get the final CRC result

crcResultCRC32 = ~ReverseBitOrder(crcResultCRC32);          // OPTIONAL
                                                            // reverse CRC value bit order and
                                                            // invert (must be 0x9AE0DAAF)

while(1);

return 1;
}

unsigned long ReverseBitOrder(unsigned long data)
{
    unsigned long maskin;
    unsigned long maskout;
    unsigned long result = 0;
    unsigned char i;

    maskin = 0x80000000;
    maskout = 0x00000001;

    for(i=0; i<32; i++)
    {
        if(data&maskin){
            result |= maskout;
        }
        maskin >>= 1;
        maskout <<= 1;
    }

    return result;
}
```

32-Bit Programmable Cyclic Redundancy Check (CRC)

Example 7-5: 32-Bit Polynomial with Switched Data Width When MSb is Shifted First

```
// ASCII bytes "12345678"
volatile unsigned long    message1[] = {0x34333231,0x38373635};

// ASCII bytes "123"
volatile unsigned char    message2[] = {'1','2','3'};

volatile unsigned long    crcResultCRC32 = 0;

int main(void)
{
    unsigned char*    pointer8;
    unsigned short*   pointer16;
    unsigned short    length;

#define CRC32_POLYNOMIAL    ((unsigned long)0x04C11DB7)
#define CRC32_SEED_VALUE    ((unsigned long)0x46AF6449)    // non-direct of 0xffffffff

    CRCCON1 = 0;
    CRCCON2 = 0;

    CRCCON1bits.CRCEN    = 1;                // enable CRC
    CRCCON1bits.LENDIAN  = 1;                // little endian
    CRCCON2bits.DWIDTH   = 32-1;             // 32-bit data width
    CRCCON2bits.PLEN     = 32-1;             // 32-bit polynomial order
    CRCCON1bits.CRCGO    = 1;                // start CRC calculation

    CRCXORL = CRC32_POLYNOMIAL&0x0000ffff;    // set polynomial
    CRCXORH = CRC32_POLYNOMIAL>>16;

    CRCWDATL = CRC32_SEED_VALUE&0x0000ffff;    // set initial value
    CRCWDATH = CRC32_SEED_VALUE>>16;

    pointer16 = (unsigned short*)message1;    // calculate CRC
    length    = sizeof(message1)/sizeof(unsigned long);
    while(length--)
    {
        while(CRCCON1bits.CRCFUL);            // wait if FIFO is full
```

dsPIC33/PIC24 Family Reference Manual

Example 7-5: 32-Bit Polynomial with Switched Data Width When MSb is Shifted First (Continued)

```

    CRCDATL = *pointer16++;           // 32-bit word access to FIFO
    CRCDATH = *pointer16++;           // must be written first
}                                     // must be written last

                                     // wait until previous
                                     // data shifts are done
while(!CRCCON1bits.CRCMPT);          // wait until FIFO is empty

asm volatile ("repeat #16-#2\n nop"); // 16 cycles maximum for 32-bit data

CRCCON2bits.DWIDTH = 8-1;            // switch the data width to 8-bit

pointer8 = (unsigned char*)message2; // calculate CRC
length = sizeof(message2)/sizeof(unsigned char);
while(length--)
{
    while(CRCCON1bits.CRCFUL);        // wait if FIFO is full

    *((unsigned char*)&CRCDATL) = *pointer8++; // byte access to FIFO
}

while(!CRCCON1bits.CRCMPT);          // wait until FIFO is empty

                                     // wait until previous data shifts are done
asm volatile ("repeat #4-#2\n nop"); // 4 cycles maximum for 8-bit data

                                     // switch the data width to polynomial length
CRCCON2bits.DWIDTH = 32-1;           // 32-bit

CRCDATL = 0;                         // dummy data to shift out the CRC result
CRCDATH = 0;

asm volatile ("repeat #2+#16-#2\n nop"); // delay 2 cycles to move data from FIFO
                                     // to shift buffer
                                     // and 16 cycles for 32-bit word to shift out
                                     // the final result

crcResultCRC32 = ((unsigned long)CRCWDATH<<16)|CRCWDATL; // get the final CRC result
                                                         // (must be 0xE092727E)

while(1);
return 1;
}
```

32-Bit Programmable Cyclic Redundancy Check (CRC)

8.0 OPERATION IN POWER SAVE MODES

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.

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.

9.0 REGISTER MAPS

A summary of the Special Function Registers associated with the dsPIC33/PIC24 32-Bit Programmable Cyclic Redundancy Check (CRC) module is provided in [Table 9-1](#).

Table 9-1: 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	0000
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	0000
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	0000
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	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: Refer to the specific device data sheet for memory map details.

32-Bit Programmable Cyclic Redundancy Check (CRC)

10.0 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 dsPIC33/PIC24 device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the 32-Bit Programmable Cyclic Redundancy Check (CRC) are:

Title	Application Note #
-------	--------------------

No related application notes at this time.

<p>Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the dsPIC33/PIC24 family of devices.</p>
--

11.0 REVISION HISTORY

Revision A (April 2009)

This is the initial released revision of this document.

Revision B (August 2013)

This revision includes the following changes:

- Changed the document name from PIC24F Family Reference Manual to dsPIC33/PIC24 Family Reference Manual.
- Revised description of CRCISEL in Register 3-1.
- Added additional information to [Section 5.3 “Data Shift Direction”](#).
- Added additional information to [Section 5.4 “FIFO”](#).
- Made corrections to [Figure 5-1](#), [Figure 5-2](#) and [Figure 5-3](#).
- Revised [Section 5.5 “CRC Engine Interface”](#).
- Revised [Section 5.6 “Interrupt Operation”](#) and added code examples.
- Revised [Section 7.2 “Typical Operation”](#) and added code examples.
- Minor grammatical corrections throughout the document.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² logo, rPIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MTP, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.


Analog-for-the-Digital Age, Application Maestro, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniclient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rLAB, Select Mode, SQI, Serial Quad I/O, Total Endurance, TSHARC, UniWinDriver, WiperLock, ZENA and Z-Scale are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

GestIC and ULPP are registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2009-2013, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-62077-400-7

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
= ISO/TS 16949 =

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://www.microchip.com/support>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880-3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820