

Cyclic Redundancy Check (CRC)

*Author: Sudhir Bommenna
Microchip Technology Inc.*

INTRODUCTION

CRC is one of the most versatile error checking algorithm used in various digital communication systems. CRC stands for Cyclic Redundancy Code Check or simply Cyclic Redundancy Check.

Most of the popular communication protocols, like CAN, USB, IrDA®, SDLC, HDLC and Ethernet, employ CRC for error detection.

Normally, for the error detection in digital communication systems, a checksum is computed on the message that needs to be transmitted. The computed checksum is then appended at the end of the message stream and is transmitted. At the receiving end, the message stream's checksum is computed and compared with the transmitted checksum. If both are equal, then the message received is treated as error free.

CRC works in a similar way, but it has greater capabilities for error detection than the conventional forms. Different CRC polynomials are employed for error detection. The size of CRC depends upon the polynomial chosen.

This application note describes the CRC operation and its implementation using a dedicated hardware module.

CRC OPERATION

All CRC calculations are carried out in the GF (2) (Galois field for 2 elements); 'field' is something in which we can perform addition, subtraction, multiplication and division and the '2 elements' specifies the field in which we have only two values, either '1' or '0'. This is analogous to the binary field or Modulo 2. Modulo 2 arithmetic uses binary addition or subtraction with no carry, which is equivalent to the XOR operation. Multiplication and division are similar to binary multiplication and division, respectively.

The message to be transmitted is treated as a polynomial and divided by an irreducible (prime) polynomial known as the 'generator polynomial'. The degree of the generator polynomial should be less than that of the message polynomial. For a 'n + 1' bit generator polynomial, the remainder will not be greater than 'n' bits. The CRC checksum of the data is the binary equivalent of the remainder after the division.

Consider a message of 'M' of 'k' bits and generator polynomial 'G' of 'n + 1' bits. Dividing the message by the generator will yield a remainder 'R' of 'n' bits. Therefore, $M = GQ + R$; where Q is the quotient obtained when M is divided by G.

EQUATION 1:

$$M = G \cdot Q + R$$

$$M + R = GQ = M - R$$

Since addition and subtraction are equivalent in Modulo 2 arithmetic.

Now, $M + R = GQ$, where the checksum is embedded into the message ($M + R$). Here, by adding the checksum to the message, we are corrupting the last 'n' bits of the message. Instead of embedding the checksum in the message, we append it to the message, thus avoiding the corruption of message bits.

When the remainder is appended to the message for transmission, it is equivalent to shifting the message bits by the number of remainder bits. Initially, the message stream is appended with zeros. After the checksum calculation, zeros are replaced with the actual checksum computed, which is the binary equivalent of the remainder of the division. The number of appended zeros is dependent on the degree of generator polynomial.

Appending 'n' zero bits to the message polynomial is equivalent to multiplying the polynomial by 2^n . Equation 2 explains this:

EQUATION 2:

$$M = G \cdot Q + R$$

$$M 2^n = Q^1 \cdot G + R^1$$

$$M 2^n + R^1 = Q^1 \cdot G$$

From Equation 2, we notice that G is the exact multiple of ($M 2^n + R^1$). At the receiving end, if G is an exact multiple of the message, then the message is not corrupted.

In the binary field, data is in the form of a polynomial (i.e., if the data field is '11101', then it can be represented in the polynomial form by writing as $X^4 + X^3 + X^2 + X^0$).

Generally, a generator polynomial is irreducible and the Most Significant bits and the Least Significant bits are always '1'. Some generator polynomials and their interpretation are given in Example 1.

AN1148

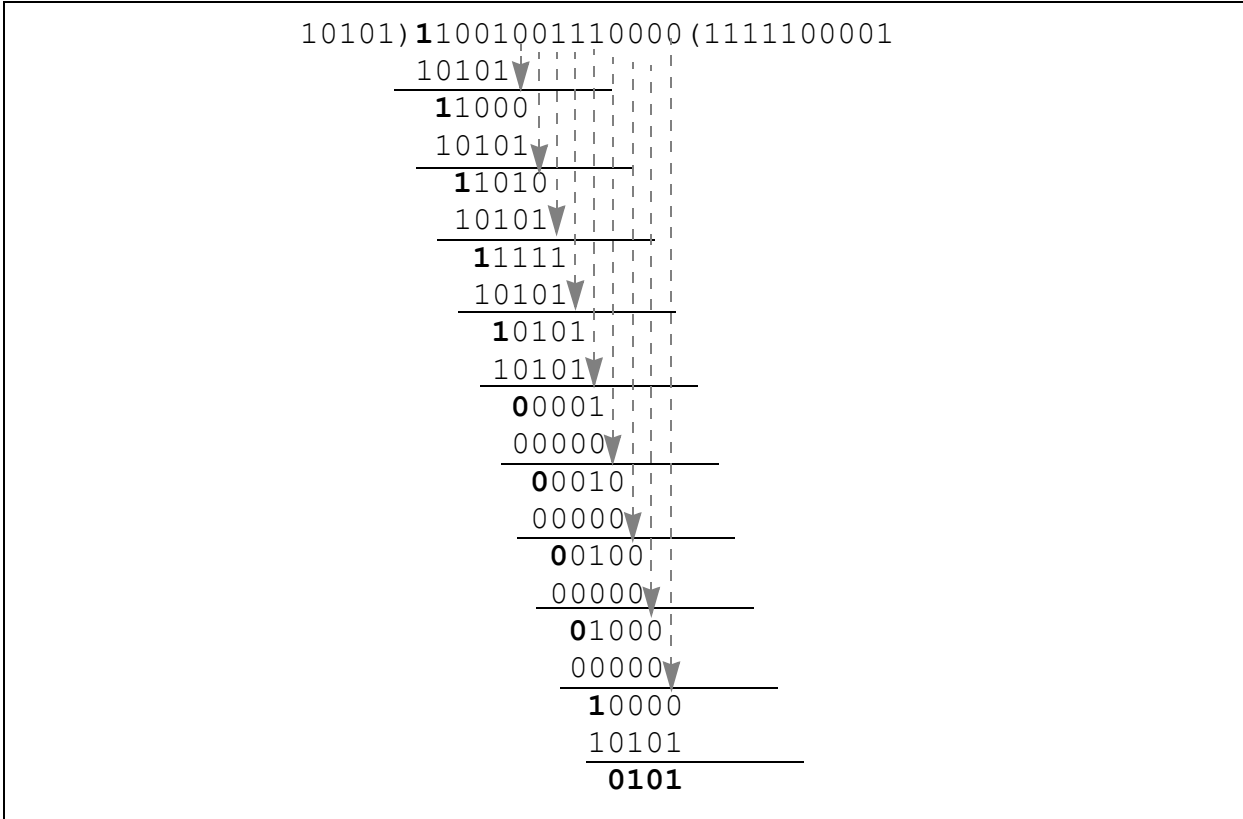
EXAMPLE 1: INTERPRETATION OF POLYNOMIALS

- | | |
|--------------|--|
| Ex.1) | CRC – 12: $X^{12} + X^{11} + X^3 + X^2 + X + 1$
1100000001111:180Fh |
| Ex.2) | CRC – 16: $X^{16} + X^{15} + X^2 + 1$
11000000000000101:11021h |
| Ex.3) | CRC – CCITT: $X^{16} + X^{12} + X^5 + 1$
10001000000100001:18005h |

Consider a message polynomial – '1100100111' and a generator polynomial – '10101' ($n + 1 = 5$).

For the CRC computation purpose, append four (n) zeros to the message polynomial and divide the message polynomial by the generator polynomial.

FIGURE 1: POLYNOMIAL DIVISION



From Figure 1, we can infer that in every step the upper most bit of the result is shifted out. These shifted out bits form the quotient of the polynomial division. The remainder of the polynomial division is the CRC of the message.

With the given division example, we can deduce that:

- If the upper most bit of a message polynomial is '1', then a XOR with the generator polynomial is performed. Then, this message polynomial is shifted by 1 bit.
- If the upper most bit is a '0', perform a shift operation by 1 bit on the message polynomial (because XOR with zeros results in the same polynomial).

With the above inferences, an algorithm for CRC computation can be defined as:

1. If the Most Significant bit is '1', shift the message bits by 1 position and perform a XOR operation.
2. If the Most Significant bit is '0', shift the message bits by 1 position.
3. If there are still more bits, then repeat from step 1.

It can be noted that the shift operation is performed first and then the XOR operation. When the generator polynomial is 'n + 1' bits, the Most Significant bit is always '1'. Since it is always '1', it can be made redundant or need not be indicated. So, effectively, we can use 'n' bits to represent the generator polynomial instead of 'n + 1' bits. Consider '10101' as the generator polynomial, since the MSb is the redundant bit, the actual bit length is 4 bits instead of 5 bits. Actual XOR operation should be performed when the shift register MSb is '1'; since we are not considering the 5th bit, we will observe the 4th bit, and when it is logic '1', we shift it by 1 more bit and perform the XOR operation. Therefore, shift operation is performed first before the XOR operation.

The above algorithm can be used for the CRC computation both in hardware and software. In hardware, CRC calculation is done using the Linear Feedback Shift Register (LFSR). The LFSR constitutes D-flip-flops and XOR gates.

As shown in Figure 2, the number of shift registers is equal to the degree of the selected generator polynomial. The XOR gates form a feedback from the LFSR register to act as a tap controller for the polynomial. After the entire message bits have shifted out, the bits which have been shifted out form the quotient and the remaining bits in the shift register form the remainder.

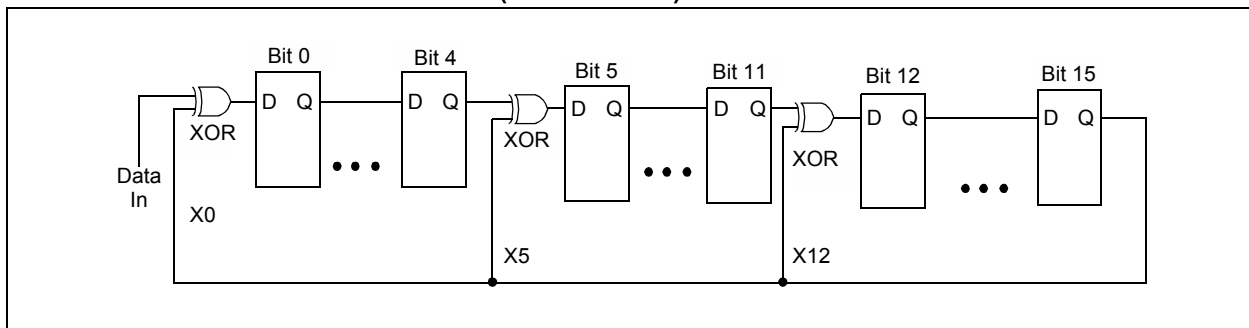
CRC is a simple and robust operation which is accomplished using a shift register in Modulo 2 arithmetic. This is because Modulo 2 calculation is simply realized by XORing numbers; hence, it is very popular. In this type of CRC operation, we can have both the remainder and the quotient. Since, the quotient in CRC computation is not needed, it's neglected, and only the remainder is taken from the LFSR register.

At the receiving end, the message stream plus the checksum are treated as the message for CRC computation. If the result is zero, then the received message, along with the CRC, is error free, else the message stream is corrupted.

The only problem with the CRC computation is that it cannot have a foolproof mechanism for leading zeros. To avoid this scenario, the CRC is first initialized to a particular value and then the computed checksum is complemented. The result is then appended to the message stream and transmitted. In this way, errors due to leading zeros can be avoided.

The most important parameter in error detection of a message stream is the selection of the generator polynomial. The polynomial selection is beyond the scope of this application note; however, the standard polynomials for different data streams are listed in Example 2 which the user can use.

FIGURE 2: CRC-16 GENERATOR (CRC – CCITT)



AN1148

DIFFERENT STANDARD POLYNOMIALS

There are various standards for the CRC computation depending on its data width. The most popular standard polynomials are listed in Example 2:

EXAMPLE 2: STANDARD POLYNOMIALS

1. CRC – 12: $X^{12} + X^{11} + X^3 + X^2 + X + 1$
2. CRC – 16: $X^{16} + X^{15} + X^2 + 1$
3. CRC – CCITT: $X^{16} + X^{12} + X^5 + 1$
4. SDLC: $X^{16} + X^{12} + X^5 + 1$

Implementation in PIC24F

The CRC hardware module in the PIC24F devices is a programmable CRC generator.

The CRC engine has the following registers:

1. CRCDAT
2. CRCWDAT
3. CRCCON
4. CRCXOR

Note: For more information on the working of the CRC engine, refer to the "PIC24F Family Reference Manual".

Users can program any user-defined generator polynomial into this module for CRC computation. The CRC result is obtained from the CRCWDAT register. For correct calculation of the CRC, PLEN + 1 number of zeros are to be appended to the data stream. Since CRCWDAT is a 16-bit register, the users need to mask the Most Significant bits while reading the final result. Masking is dependent on the width of the polynomial used. For example, if the user is using an 8-bit polynomial, then the 8 Most Significant bits should be masked while reading the result.

The CRC computation also depends upon the type of transmission. For example, let's consider a message stream of 4 bytes (0x83, 0x27, 0x49, 0x15) followed by 2 bytes of the CRC. In CRC computation, the first bit transmitted is considered to be the Most Significant bit and the last bit as the Least Significant bit.

1. If the transmission media transmits the Most Significant bit (MSb) first, then the data format for computation of CRC for the above 4 bytes of message is shown in Table 1.

TABLE 1: MSb BIT FIRST

1st	2nd	3rd	4th	CRC (2 bytes)
0x83	0x27	0x49	0x15	0xB072

2. If the transmission media transmits the Least Significant bit first, then the data format for computation of CRC is shown in Table 2.

TABLE 2: LSb BIT FIRST

1st	2nd	3rd	4th	CRC (2 bytes)
0xC1	0xE4	0x92	0xA8	0x51CF

We observe that when the LSb is transmitted first, then all the bytes are reversed and the CRC needs to be calculated accordingly.

A manual CRC calculation procedure is shown in Example 3.

In a software implementation, there would be numerous iterations involved; for instance, for every byte, the software has to perform eight iterations and each iteration has to undergo a conditional check for an XOR operation and the actual XOR operation itself. This process usually consumes a lot of MIPS.

A flowchart in Figure A-1 details the calculations performed by the CPU for each bit. Even the most optimized code for the above algorithm takes at least 4 instruction cycles, which implies that for every bit operation, it takes around 8 clock cycles. However, the CRC hardware module performs the same bit operation in a single clock cycle.

Example 3 illustrates the amount of time reduction in the CRC hardware module, when compared to its software implementation, for a given data of 100 bytes.

EXAMPLE 3: MANUAL CRC CALCULATION

Case 1: Software Implementation Process

100 bytes = 800 bits
 1 bit = 8 clock cycles
 800 bits = 6400 clock cycles

Case 2: Hardware Module

100 bytes = 800 bits
 1 bit = 1 clock cycle
 800 bits = 800 clock cycles

Number of clock cycles saved is 5600, which is a 700% time reduction in comparison to a software implementation process.

Note: The above calculations are approximations for a highly optimized code in software. The actuals would be greater than the stated numbers.

A brief comparison of hardware and software memory requirements is given in Table 3.

TABLE 3: COMPARISON OF HARDWARE AND SOFTWARE MEMORY REQUIREMENTS

	RAM (bytes)	ROM (instructions)
Software	6	45
Hardware	—	20

An experimental CRC calculation has been tested for 7 words of data, both in hardware and software, to measure the speed of calculations. In this calculation, the software took 550 microseconds, and the hardware took 17 microseconds, using an 8 MHz clock source; also, the CPU is free to do other tasks when the hardware CRC is being calculated.

Figure A-2 presents a CRC computation in PIC24F devices using the CRC module. This details the CRC computation flowchart in PIC24F devices using the CRC module.

A generalized software code is shown in Example B-1 for the CRC computation using the CRC engine in PIC24F devices.

When a 16-bit polynomial is selected, the PIC24F CRC engine expects the data width to be 16 bits.

Example B-1 provides software for computing the CRC. In some cases, it may be required to calculate the CRC with a different width other than the specified (i.e., data width may not be equal to the generator polynomial).

Example B-2 illustrates a software code when the generator polynomial is 16 bits and the data width is 8 bits.

SUMMARY

This application note gives an overview of the CRC algorithm, manual CRC calculation and CRC calculation using the PIC24F hardware module. This application note also compares the software-based approach with the hardware approach. It can be, therefore, concluded that a dedicated and programmable hardware peripheral for computing CRC is much better than a software approach.

For a detailed description of the software implementation, refer to AN730, “CRC Generating and Checking” (DS00730) on Microchip Technology’s web site: www.microchip.com.

REFERENCES

- TV Ramabadran and Sunil S. Gaitonde, “A Tutorial on CRC Computations”, IEEE MICRO
- A. Perez, “Byte Wise CRC Calculations”, IEEE MICRO
- A.S. Tanenbaum, “Computer Networks”, Prentice Hall
- Ross N. Williams, “A Painless Guide to CRC Error Detection Algorithm”

APPENDIX A: FLOWCHARTS

FIGURE A-1: SOFTWARE FLOWCHART

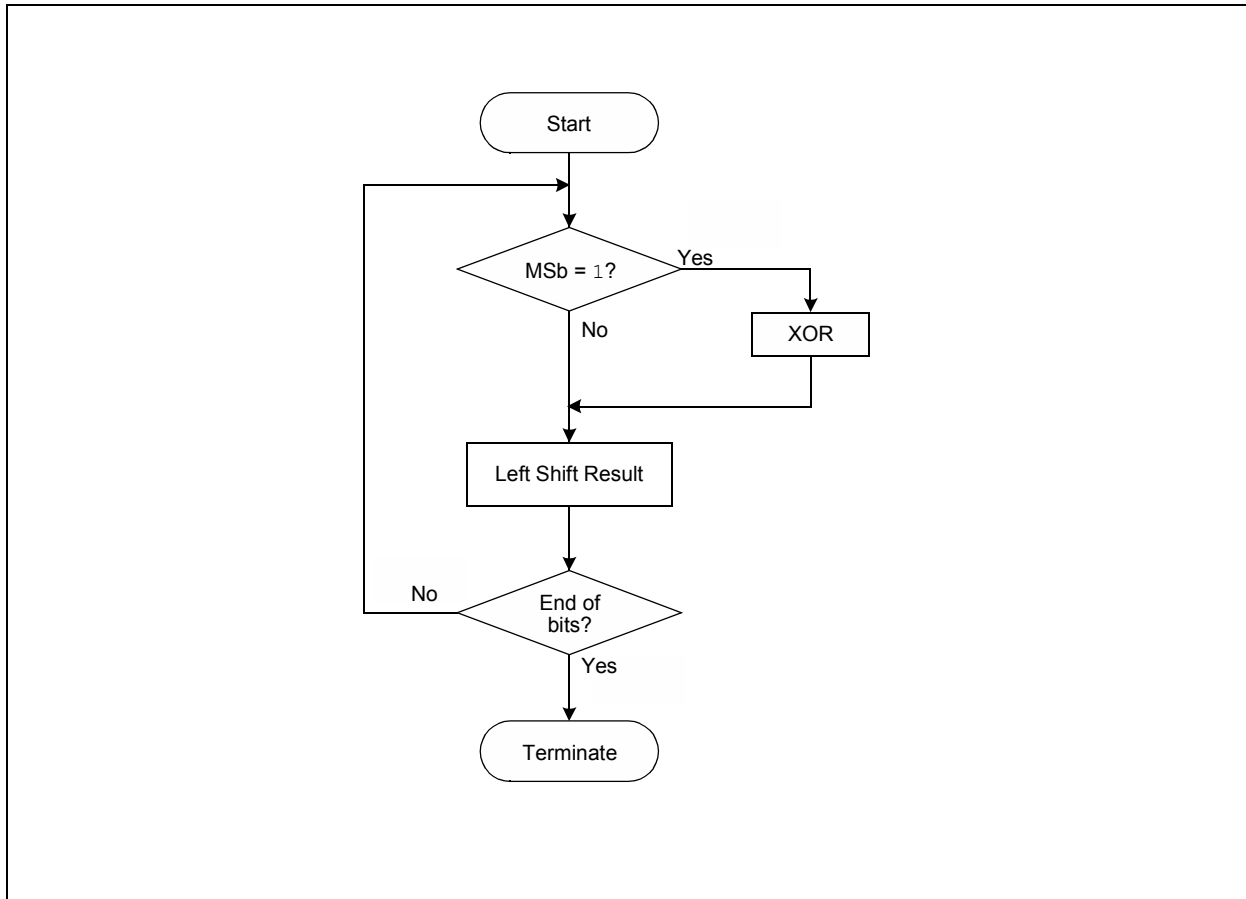
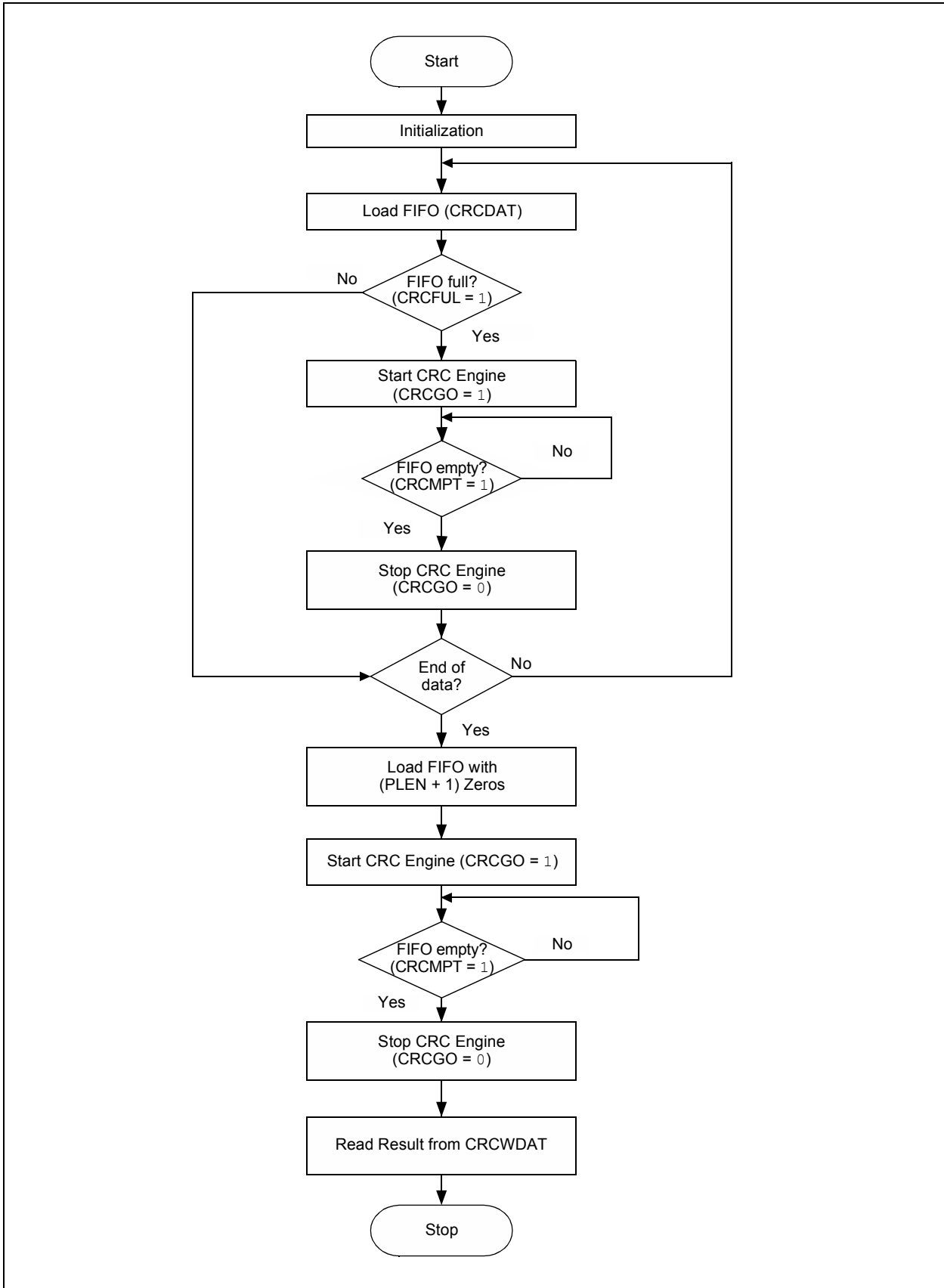


FIGURE A-2: SOFTWARE FLOWCHART FOR IMPLEMENTATION IN PIC24F



APPENDIX B: COMPUTATION CODE

EXAMPLE B-1: CRC COMPUTATION WITH 16-BIT DATA

```
#include<p24fxxxx.h>
unsigned int Result;

unsigned int dataw[]={
0x0001,0x0002,0x0003,0x0004,
0x0005,0x0006,0x0007,0x0008,
0x0009,0x000A,0x000B,0x000C,
0x000D,0x000E,0x000F,0x0010,
0x0011,0x0012,0x0013,0x0014,
0x0015
};

unsigned int CRC_HW_calculateW(unsigned int *,unsigned int);

main()
{
    Nop();
    Result=CRC_HW_calculateW(dataw,sizeof(dataw));
    Nop();
    Nop();
    while(1);
}

unsigned int CRC_HW_calculateW(unsigned int *temp,unsigned int Len)
{
    unsigned int j;
    CRCCON = 0x0000F; // ="Length of polynomial-1"
    CRCXOR = 0x1021; // generator Polynomial
    CRCWDAT= 0x0000; // Initialize CRCWDAT with 0
    Len=Len/2;
    for(j=0;j<Len;j++)
    {
        CRCDAT =*temp++; //write data into FIFO
        if(CRCCONbits.CRCFUL==1)//check if FIFO is full
        {
            CRCCONbits.CRCGO=1; //start CRC engine
            while(CRCCONbits.CRCMPT!=1);//check if FIFO is empty
            Nop();
            Nop();
            Nop();
            Nop();
            Nop();
            CRCCONbits.CRCGO=0; //stop CRC engine
        }
    }

    if(CRCCONbits.CRCGO!=1)
        CRCCONbits.CRCGO=1;

    CRCDAT = 0x0000; //appending PLEN+1 zeros (multiply by 2^16)

    while(CRCCONbits.CRCMPT!=1);//check if FIFO is empty
        Nop();
        Nop();
        Nop();
        Nop();
        Nop();
    CRCCONbits.CRCGO=0; //stop CRC engine
    Nop();

    return CRCWDAT;
}
```


EXAMPLE B-2: CRC COMPUTATION WITH 8-BIT DATA

```

#include<p24fxxxx.h>
unsigned int Result;
unsigned char datab[]={
0x01,0x02,0x03,0x04,
0x05,0x06,0x07,0x08,
0x09,0x0A,0x0B,0x0C,
0x0D,0x0E,0x0F,0x10,
0x11,0x12,0x13,0x14,
0x15
};
unsigned int CRC_HW_calculateB(unsigned char *,unsigned int);
main()
{
    Nop();
    Result2=CRC_HW_calculateB(datab,sizeof(datab));
    Nop();
    Nop();
    while(1);
}
unsigned int CRC_HW_calculateB(unsigned char *temp,unsigned int Len)
{
    unsigned int Carry,j;
    unsigned char *ptr,Flag;
    ptr=(unsigned char *)&CRCDAT;
    CRCCON = 0x0000F; // ="Length of polynomial-1"
    CRCXOR = 0x1021; // generator Polynomial
    CRCWDAT= 0x0000; // Initialize CRCWDAT with 0
    Flag=0x00;
    for(j=0;j<Len;j++)
    {
        *ptr =*temp++; //write data into FIFO
        Flag=Flag^0x01;//Flag for odd or even bytes
        if(CRCCONbits.CRCFUL==1)//check if FIFO is full
        {
            CRCCONbits.CRCGO=1; //start CRC engine
            while(CRCCONbits.CRCMPT!=1);//check if FIFO is empty
            CRCCONbits.CRCGO=0; //stop CRC engine
        }
    }
    if(CRCCONbits.CRCGO!=1)
        CRCCONbits.CRCGO=1;
    if(Flag==0)
        CRCDAT = 0x0000; //appending PLEN+1 zeros (multiply by 2^16)
    else
        *ptr=0x00;//appending (PLEN+1)/2 zeros (multiply by 2^8)
    while(CRCCONbits.CRCMPT!=1);//check if FIFO is empty
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    CRCCONbits.CRCGO=0; //stop CRC engine
    Nop();
    if(Flag==1) // if odd number of bytes,
    { //append (PLEN+1)/2 zeros (multiply by 2^8)
        for(j = 0; j < 8; j ++ )
        {
            Carry =( CRCWDAT & 0x8000);
            CRCWDAT <<= 1;
            if(Carry)
                CRCWDAT ^= 0x1021;
        }
    }
    return CRCWDAT;
}

```

AN1148

APPENDIX C: CRC CALCULATIONS

TABLE C-1: MANUAL CRC CALCULATION

Data = 0x51; Poly = 0x1021; Result = 0x0000			
	CRC Shift Register	Message Bits with Appended '0's	MSb Shift
	0000 0000 0000 0000	0101 0001 0000 0000 0000 0000	
	0000 0000 0000 0000	1010 0010 0000 0000 0000 000	1st bit shift
	0000 0000 0000 0001	0100 0100 0000 0000 0000 00	2nd bit shift
	0000 0000 0000 0010	1000 1000 0000 0000 0000 0	3rd bit shift
	0000 0000 0000 0101	0001 0000 0000 0000 0000	4th bit shift
	0000 0000 0000 1010	0010 0000 0000 0000 000	5th bit shift
	0000 0000 0001 0100	0100 0000 0000 0000 00	6th bit shift
	0000 0000 0010 1000	1000 0000 0000 0000 0	7th bit shift
	0000 0000 0101 0001	0000 0000 0000 0000	8th bit shift
	0000 0000 1010 0010	0000 0000 0000 000	9th bit shift
	0000 0001 0100 0100	0000 0000 0000 00	10th bit shift
	0000 0010 1000 1000	0000 0000 0000 0	11th bit shift
	0000 0101 0001 0000	0000 0000 0000	12th bit shift
	0000 1010 0010 0000	0000 0000 000	13th bit shift
	0001 0100 0100 0000	0000 0000 00	14th bit shift
	0010 1000 1000 0000	0000 0000 0	15th bit shift
	0101 0001 0000 0000	0000 0000	16th bit shift
MSb = 1	1010 0010 0000 0000	0000 000	17th bit shift
Shift	0100 0100 0000 0000	0000 00	18th bit shift
XOR	0001 0000 0010 0001		
Result	0101 0100 0010 0001	0000 00	
MSb = 1	1010 1000 0100 0010	0000 0	19th bit shift
Shift	0101 0000 1000 0100	0000	20th bit shift
XOR	0001 0000 0010 0001		
Result	0100 0000 1010 0101	0000	
MSb = 1	1000 0001 0100 1010	000	21st bit shift
Shift	0000 0010 1001 0100	00	22nd bit shift
XOR	0001 0000 0010 0001		
Result =	0001 0010 1011 0101	00	
Shift	0010 0101 0110 1010	0	23rd bit shift
Final Result =	0100 1010 1101 0100		24th bit shift

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, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


AmpLab, FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rLAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA 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.

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

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==**

Microchip received ISO/TS-16949:2002 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://support.microchip.com>
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

Dallas

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

Detroit

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

Kokomo

Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

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: 86-10-8528-2100
Fax: 86-10-8528-2104

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

China - Hong Kong SAR
Tel: 852-2401-1200
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: 86-755-8203-2660
Fax: 86-755-8203-1760

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

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

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

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

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

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 - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

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: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
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