AN1817

Using a Hardware or Software CRC with Enhanced Core PIC16F1XXX in Class B Applications

Author: *Corey Simoncic*
Microchip Technology Inc.

1.0 INTRODUCTION

Class B safety routines are increasingly used in microcontrollers to detect faults in safety-critical applications. The primary method for detecting faults in microcontroller program memory is by using a Cyclic Redundancy Check (CRC) as defined by the IEC 60730 standard.

A CRC can be used to prevent application faults due to corrupted program memory by performing a periodic check to determine if the check value has changed.

This application note will describe how to implement the Software CRC available as part of the Class B Safety Software Library and the hardware CRC used in selected microcontrollers (this document will focus on the PIC16F161X family).

Both methods discussed in this application note satisfy IEC 60730 spec H.2.19.3.2 to test Invariable Memory for all single-bit faults with 99.6% coverage.

For additional information on Class B and full example code for this application note please visit:

www.microchip.com/classb

For additional information on CRC algorithms, please refer to "*A Painless Guide on CRC Algorithms*" by Ross N. Williams (August 19, 1993).

1.1 Cyclic Redundancy Check

CRC uses a method very similar to polynomial long division to identify a unique check value similar to the remainder in polynomial long division. This is done by choosing a very specific divisor known as the CRC polynomial. CRC polynomials are unique polynomials selected to identify the maximum amount of errors in any given data stream. The CRC polynomial used in this application note is CRC-16-ANSI, as shown in Figure 2. Another popular CRC algorithm is CRC-16-CCITT. This algorithm is primarily used in communication CRCs.

The check value can be used in Class B applications by running an initial CRC then periodically running a CRC to confirm that the check value has not changed.

1.2 CRC Implementation

There are a few common ways to implement a CRC. The most common hardware implementation of a CRC is the Linear Feedback Shift Register (LFSR). The LFSR for CRC-16-ANSI is shown in Figure 2. This implementation will feed the data stream into the CRC by placing the XOR gates at the appropriate locations, according to the CRC algorithm chosen.

A common software implementation of CRC is by using a table. However, this method uses a large amount of memory and is not efficient to use on low-memory PIC16s. A parallel computation method was chosen to do a Software CRC in the Class B Safety Software Library. This works by selectively using XOR's on bits determined by different forms of parity.

1.3 CRC Error Detection

The polynomial in a CRC calculation is selectively chosen to find as many bit errors as possible. The common CRC polynomials, including the CRC-16-CCITT and CRC-16-ANSI, are designed to provide the maximum coverage for error detection. These polynomials are designed to identify all single-bit errors, two-bit errors, odd number-bit errors, and burst errors.

The effectiveness of each polynomial for errors other than these is under much debate and is not within the scope of this application note.

1.4 CRC Terms

- Polynomial – This is the divisor for the CRC algorithm. This is the primary distinction between various CRC methods.
- Initial Value – This is the starting value of the CRC calculation. Most CRC algorithms have predefined initial values in order to ensure maximum error detection.
- Augmented Zeros – Zeros are appended to the end of the data sequence for CRC calculation.
- Endianness – Determines the bit order for the data into the CRC calculation; can be either MSb first or LSb first.
- Check Value – This is the final product of the CRC calculation; can also be referred to as checksum or remainder.

1.5 Class B CRC Scope

In this application note, the terms used for CRC are defined as:

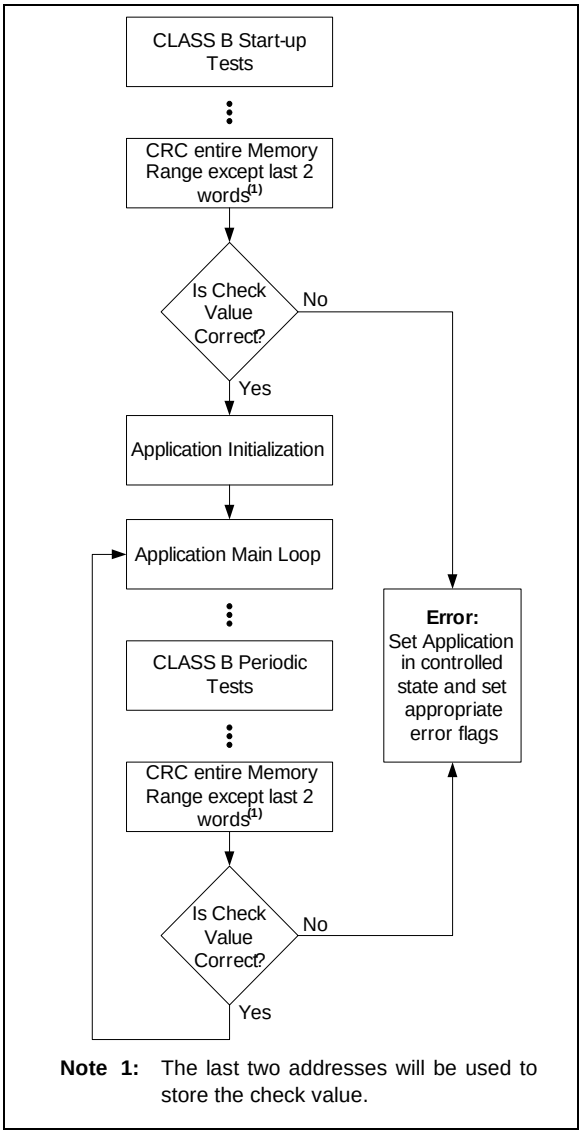
- Polynomial = CRC-16-ANSI
- Initial Value = 0xFFFF
- Augmented zeros will be used
- Endianness = MSb first

1.6 Using CRC for Class B in Embedded Applications

A CRC can be an accurate and reliable way of testing the program memory of embedded applications such as those using PIC16F1613.

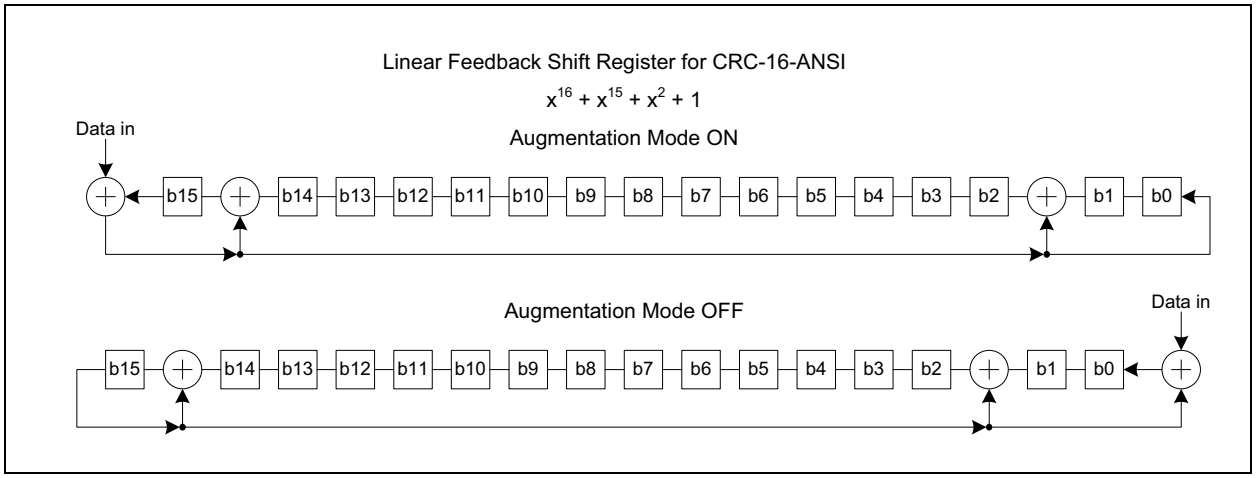
The basic flow of using CRC (along with the rest of the Class B tests) in a typical application is shown in Figure 1. The CRC calculation code provided along with this application note will be used to calculate the checksum prior to programming the part. The check value will then be programmed into the final two addresses in the program memory. This is how the check value will be verified each time the CRC is calculated.

FIGURE 1: CLASS B FLOWCHART



Note 1: The last two addresses will be used to store the check value.

FIGURE 2: CRC LFSR



2.0 USING THE CRC PERIPHERAL FOR CLASS B

The PIC16F161X family of products has a hardware implementation of a CRC. This section will provide step-by-step instructions on how to use the CRC peripheral in a Class B application, including code samples.

A memory scanner is included in the CRC peripheral in the PIC16F161X. This memory scanner can be used to load the CRC peripheral with data directly from the Flash program memory of the device.

The first step in using the hardware CRC in the PIC16F1613 will be to configure the registers for the CRC and the memory scanner. Further information is located in Section 11.0 of the PIC16F1613 data sheet (DS40001737).

2.1 Configuring the CRC Module

Polynomial of CRC-16:

```
CRCXORH = 0b10000000;
CRCXORL = 0b00000101;
```

The most significant ‘1’ is assumed and automatically implemented by the module.

Initial Value:

```
CRCACCH = 0b11111111;
CRCACCL = 0b11111111;
```

Augmented Zeros enabled:

```
CRCCON0bits.ACCM = 1;
```

Endianness (shifting MSb first):

```
CRCCON0bits.SHIFTM = 0;
```

In order to be kept adaptable, the CRC module was designed to take in a range of different polynomials and data string lengths. Because of this, the module must also be configured for the polynomial length and the data length.

Polynomial Length of 16 bits:

```
CRCCON1bits.PLEN = 0b1111;
```

Data Length of 16 bits:

```
CRCCON1bits.DLEN = 0b1111;
```

Note:	The data length of the program memory in the PIC16F1613 device is actually 14 bits. A length of 16 bits was selected here to allow for verifying the check value against the CRC calculator and the Software CRC. This change appends each program memory word with two zeros to create a 16-bit word. See Example 1 for more information.

.

.

2.2 Configuring the Memory Scanner

Because the CRC is a safety-critical task, the memory scanner will be used in Burst mode. This means that all CPU functions will be halted while the CRC is running. At 8 MHz FOSC, this means that the CPU is halted for 4.086 ms to run the CRC on the entire memory panel (2000 program words).

Burst mode:

`SCANCON0bits.MODE = 0b01;`

If the 4.086 ms is too long, an additional bit can be set for the scanner, which will halt the scanner during an interrupt. This means that Safety Critical interrupts can still be served.

Optional interrupt service:

`SCANCON0bits.INTM = 1;`

The final step to configuring the memory scanner is setting the start and final addresses. The SCANLADRH:L register pair holds the starting address for the memory scanner. The SCANHADRH:L register pair holds the final address to be scanned. During the scan operation, the SCANLADR register pair is incremented to show the current address being retrieved. In Example 1, the last two memory addresses will be used to store the resulting check value.

First Address:

SCANLADRH = 0x00;

SCANLADRL = 0xFD;

Last Address:

SCANHADRH = 0x07;

SCANHADRL = 0xFB;

EXAMPLE 1: CRC PERIPHERAL CODE

```
uint16_t HWCRC (uint16_t lastAddress)
{
    uint16_t HWCRCresult;

    CRCACCL = 0xFF; //Seed with 0xFFFF
    CRCACCH = 0xFF;
    CRCXORH = 0x80; //using CRC-16-ANSI 0x8005
    CRCXORL = 0x05;
    CRCCON1bits.DLEN = 15; //using 16 bit data length to match the Software CRC
                           //the most-significant 2 bits will be treated as 0.
    CRCCON1bits.PLEN = 15; //using the maximum 17-bit polynomial (-2)
    CRCCON0bits.ACCM = 1; //turn on augmented zeros
    CRCCON0bits.SHIFTM = 0; //MSb-first (normal)
    SCANCON0bits.MODE = 0b01; //turn on "Burst mode" to stop all
                              //other execution until CRC complete
    SCANLADRH = 0x00; //set the first address for memory scan
    SCANLADRL = 0x00;
    SCANHADRH = lastAddress >> 8; //set the last address for memory scan
    SCANHADRL = lastAddress;

    SCANCON0bits.EN = 1;
    CRCCON0bits.EN = 1;
    CRCCON0bits.CRCGO = 1; //Turn on the CRC
    SCANCON0bits.SCANGO = 1; //Turn on the scan to begin the CRC
    //This should halt CPU Execution until the Scanner is complete and the final
    //memory location is in the CRC

    while(CRCCON0bits.BUSY);

    HWCRCresult = ((CRCACCH<<8) | CRCACCL);
    return HWCRCresult;
}
```

2.3 Running the CRC

Both the CRC and scanner have now been configured to run the CRC-16-ANSI algorithm in the desired memory region. The CRC and scanner modules can now be enabled.

First set the CRCGO bit to begin the CRC, then set the SCANGO bit. The CPU will halt normal code execution here because the scanner has been set to Burst mode (with the possible exception to interrupts as stated earlier).

The CPU will be shut down for an approximate 4.086 ms. The BUSY bit of CRCCON0 will be cleared by hardware when the CRC operation has finished. When this bit is cleared, the final check value will be located in the CRCACCH:L register pair.

2.4 CRC Peripheral Timing

The timing for the CRC peripheral changes depending on the data width, FOSC, and the number of addresses being scanned.

The CRC takes four instruction cycles per word of program memory tested if using 16-bit data width. For 2046 addresses of Flash memory being tested, the CRC will take 2046 * 4 instruction cycles to finish, plus a few additional instructions for calls and returns.

For this application note, the CRC takes 8211 instruction cycles at 8 MHz making it take just over 4 ms. This timing was determined using the Signal Measurement Timer (SMT) on the device.

For more information about how to obtain the timing for the CRC or any other peripheral using the SMT, please refer to **Appendix B: "SMT Timing"**.

3.0 CRC CALCULATOR

This application note includes an easy-to-use CRC calculator that supports multiple polynomials and all features of the CRC peripheral. This tool will be used to verify all CRC calculations done by the CRC peripheral and the Class B Library.

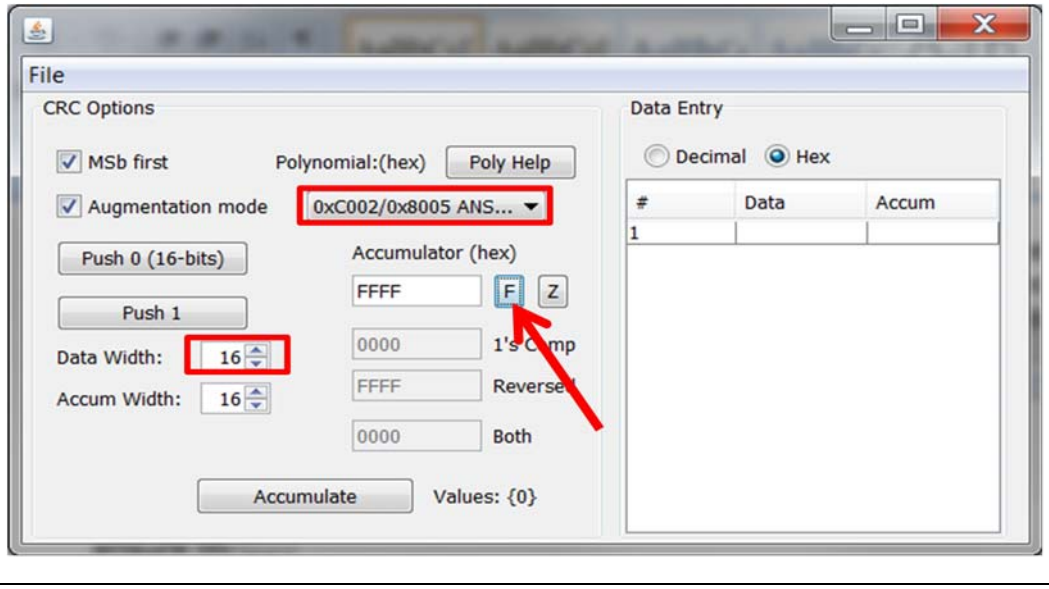
3.1 Features

The CRC calculator has all the features of the CRC peripheral. This includes multiple polynomial selections to choose from, option to turn off Augmentation mode, toggle between MSb first and LSb first, and optional data and accumulator widths. The CRC calculator has two data entry methods. The first is to manually enter data into the Data column on the right side. The second is to import a file of line-delimited hex values, like those that can be taken from MPLAB X IDE (explained in **Section 3.3 "Using MPLAB® X IDE to fill the table data"**).

3.2 Setup for CRC-16-ANSI

To setup the CRC calculator for CRC-16-ANSI, a few options must be changed from the defaults in the calculator. First, select the 0x8005 polynomial using the drop-down polynomial selection box. Then press the F under the accumulator; this will set the initial value to 0xFFFF. Finally, the Data Width must be increased to 16. The rest of the default options can remain unchanged (see Figure 3).

FIGURE 3: CRC CALCULATOR SETUP FOR CRC-16-ANSI



3.3 Using MPLAB® X IDE to fill the table data

The program memory view in MPLAB X IDE can be used to quickly fill the Data table for the CRC calculator. To do this, first open the program memory view in MPLAB X IDE, as shown in Figure 4. Copy the opcodes needed using the copy hot key (windows is CTRL+C), as shown in Figure 5. The opcodes will be copied into the clipboard in a standard-line delimited format that can then be pasted into a standard text file (see Figure 6). Next, save the file and open the CRC calculator and go to *file -> import file* (see Figure 7). Choose the saved file, and the Data table will be automatically filled with the opcodes from the MPLAB X IDE project.

FIGURE 4: PROGRAM MEMORY VIEW

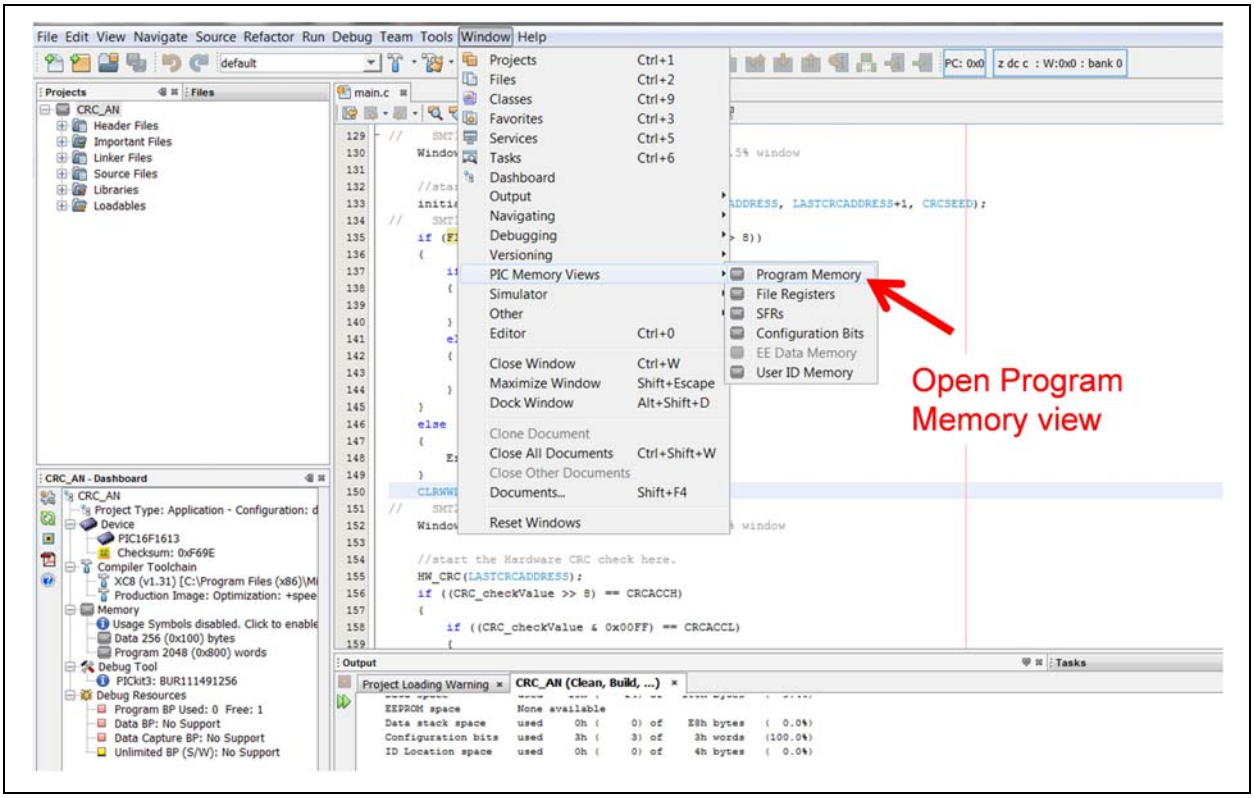


FIGURE 5: COPY OPCODES FROM PROGRAM MEMORY

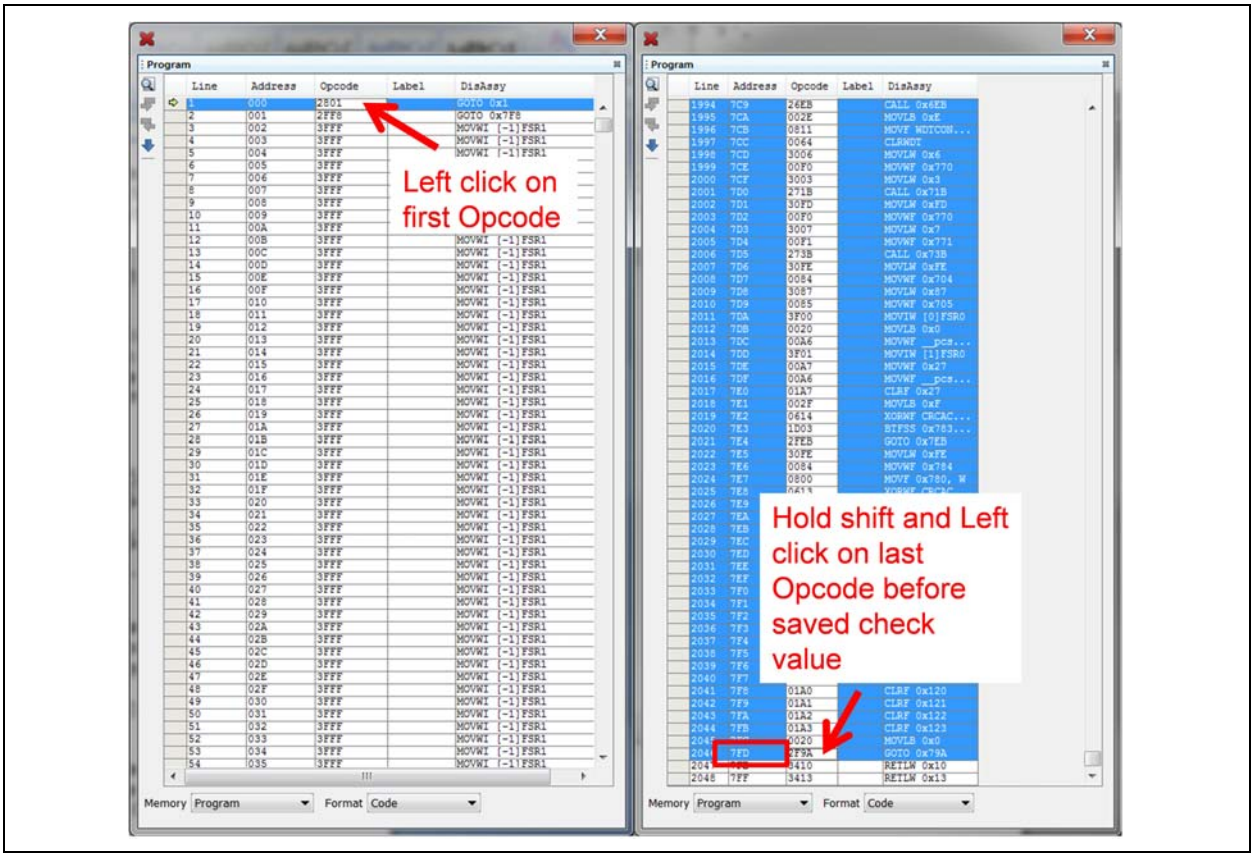


FIGURE 6: OPCODE LIST

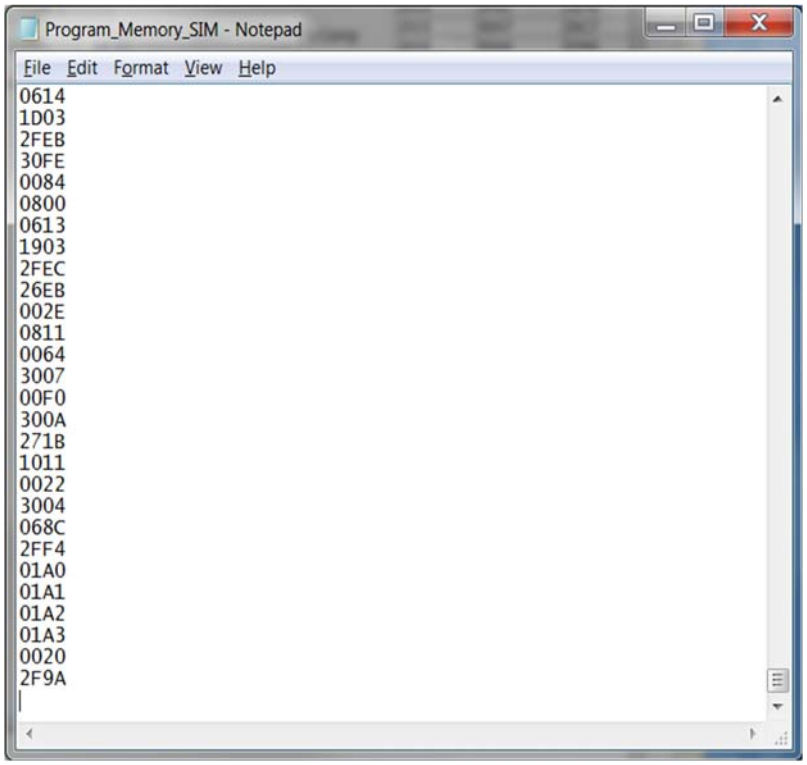
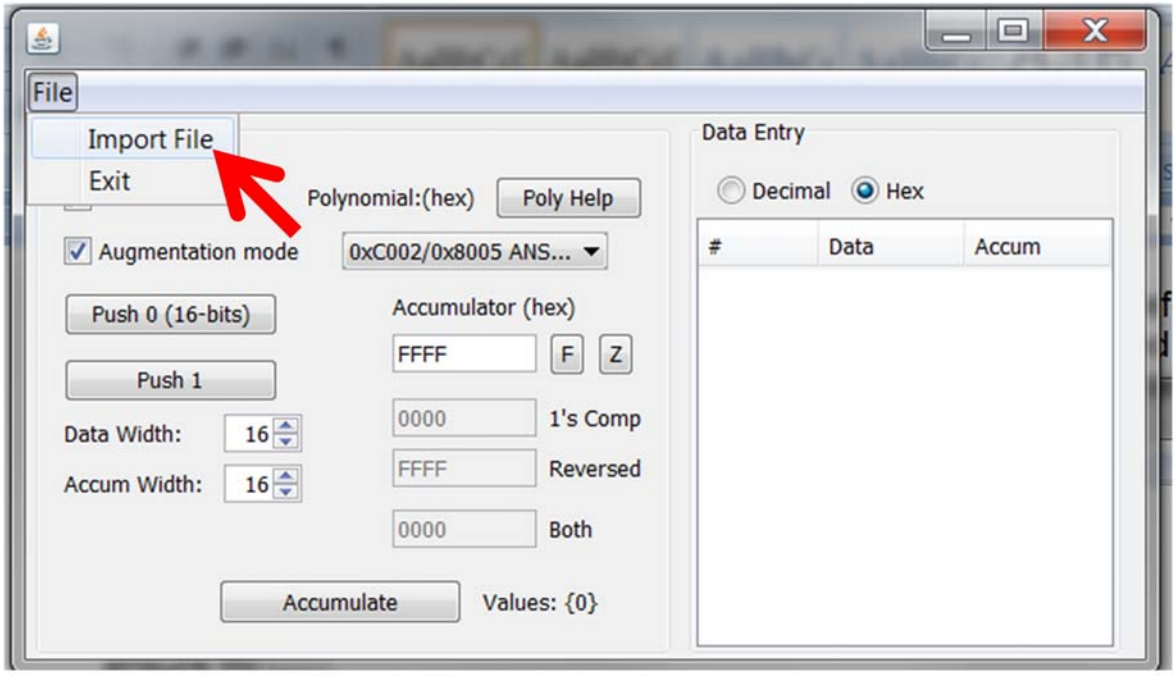


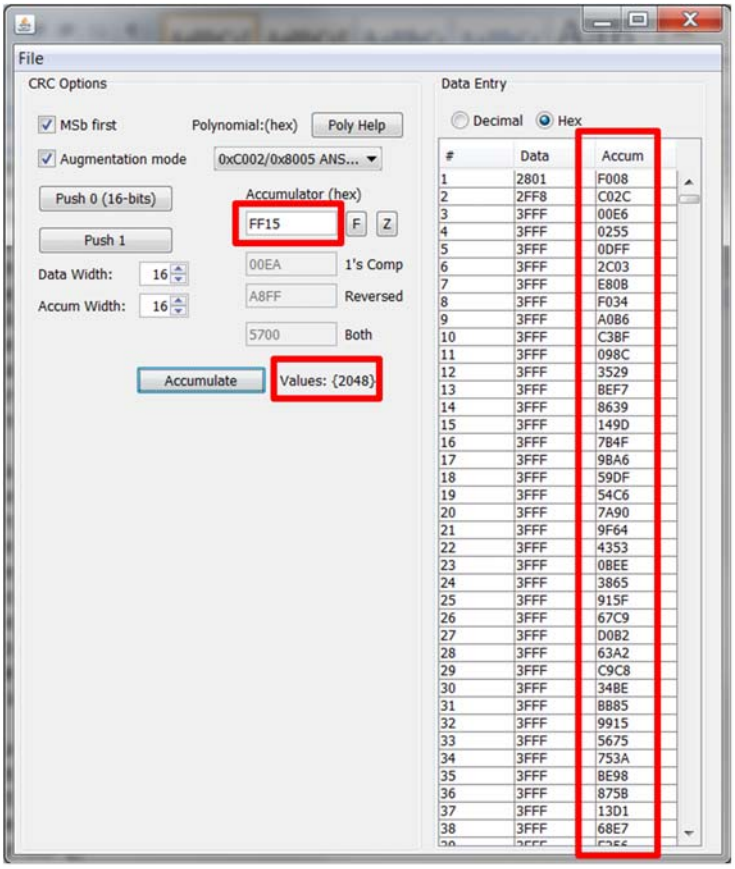
FIGURE 7: IMPORT FILE



3.4 Calculating the Check value

After the data table is filled out with the desired information, the *Accumulate* button will generate the final check value in the accumulator text box. The values dialogue indicates how many total words were calculated using the CRC. The Accum column in the data entry section shows the accumulator value after each word of data is entered into the CRC. This can be quite useful for debugging (see Figure 8).

FIGURE 8: ACCUMULATE USING CRC CALCULATOR



4.0 USING THE CLASS B LIBRARY CRC FUNCTION

This section will describe how to implement the Flash program memory CRC function from the library.

The Class B Safety Software Library comes with a software implemented CRC function to test Flash program memory and EEPROM.

The Software CRC is limited to the CRC-16-ANSI algorithm, unlike the CRC peripheral.

For more information on using the Class B library, see DS00001799.

4.1 API

To run the CRC test across Flash memory, the following function must be called:

`CLASSB_CRCFlashTest(…);`

The function has three arguments: the first address to be tested, the length of the test, and the seed for the CRC algorithm.

To match the CRC peripheral, these arguments would have the following values:

- `uint16_t FlashAddress = 0x00`
- `uint16_t Flashlength = 0x07FC`
- `uint16_t crcSeed = 0xFFFF`

Note:	The length here will include the first address. The test will return the 16-bit check value. See Example 2 for more information.

EXAMPLE 2: CRC LIBRARY CODE

```
uint16_t flashAddress = 0x00;
uint16_t flashLength = 0x07FC;
uint16_t crcSeed = 0xFFFF;
uint16_t CRC-libraryResult;

CRC_libraryResult = CLASSB_CRCFlashTest(myAddress,length,crcSeed);
```

4.2 Software CRC Timing

The Software CRC takes 216,956 instruction cycles compared to the 8172 instruction cycles of the CRC peripherals for the same amount of memory. This equates to 108.478 ms.

5.0 USING THE CHECK VALUE FOR ERROR CHECKING

Using the CRC calculation application, a reference check value can be programmed into the device. This reference can then be checked periodically against the resulting check value from the CRC peripheral or the Class B Library CRC function. If the two values match, then there has not been a program memory failure with a large degree of certainty. If the two values do not match, it means that there has been a memory corruption and steps should be taken to ensure the safety of the device.

If a reference check value cannot be used because of calibration or other run-time values stored in program memory, the CRC can be run twice (after the memory values have changed) and the two check values can be compared. For this, the previous check value can be stored in Flash program memory using the self-write functionality of the PIC® MCU or in a static RAM location.

Due to the nature of the CRC algorithm, it is not possible to find out where the actual error came into the program memory. Depending on the application, a number of steps can be taken.

- An error flag can be set, halting operation
- The device can be held in Reset
- The device can enter an infinite loop halting all other operation
- An Error signal can be sent to inform the consumer of an error

5.1 Reference Check Value

The reference check value determined using the CRC calculation application (See **Section 3.0 "CRC Calculator"**) can be stored in program memory directly, at the time of programming using `const` variables.

For example, if the reference check value was determined to be 0x1234, the following could be used to store this value:

```
const uint16_t CRC_checkValue @ 0x7FE = 0x1234
```

This will put 0x3434 at location 0x7FE and 0x3412 at location 0x7FF. The MSB contains 0x34 because this is the opcode for a `RETLW` instruction. This is shown in Table 1.

TABLE 1: CHECK VALUE IN PROGRAM MEMORY

Line	Address	Opcode	DisAssy
2047	7FE	3434	`RETLW` 0x34
2048	7FF	3412	`RETLW` 0x12

The reference check value can then be used to compare to a measured check value from the Library or the Peripheral CRC (refer to Example 3 and Example 4).

EXAMPLE 3: COMPARING LIBRARY MEASURED CHECK VALUE

```
if (CRC_checkValue == CRC_libraryResult)
{
    // do nothing, check value matches
}
else
{
    ErrorMode();
}
```

EXAMPLE 4: COMPARING PERIPHERAL MEASURED CHECK VALUE

```
if ((CRC_checkValue >> 8) == CRCACCH)
{
    if ((CRC_checkValue & 0x00FF) == CRCACCL)
    {
       // do nothing, check value matches
    }
    else
    {
       ErrorMode();
    }
}
else
{
     ErrorMode();
}
```

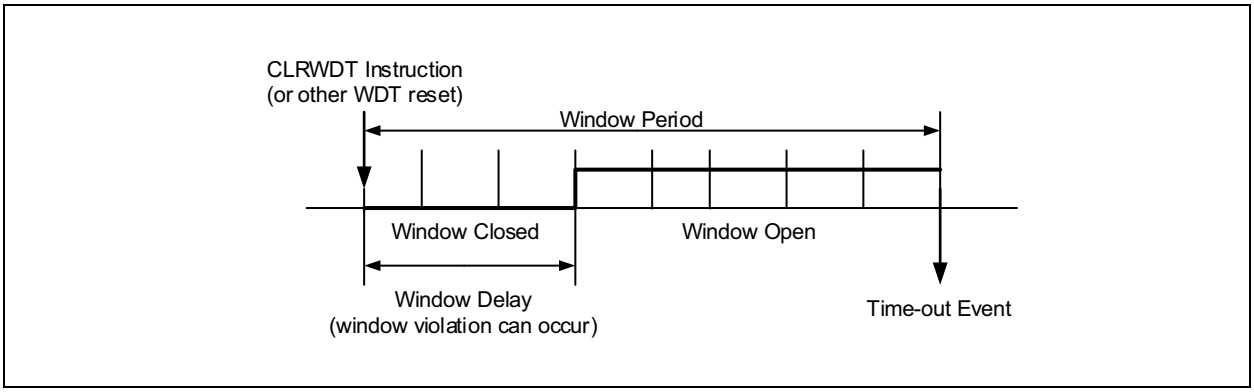
APPENDIX A: WINDOWED WATCHDOG TIMER

The windowed Watchdog Timer is a module added in the PIC16F161X family of products that is better suited for time slot monitoring than a typical watchdog. A windowed Watchdog Timer can detect both too slow and too fast clock speeds. A Reset will occur if the program attempts to clear the timer before the window or without arming the `CLRWDT` instruction. In order to use this feature, precise timing is needed for the processes running. The CRC peripheral is one of the modules that does provide precise timing (see section **Section 2.4 "CRC Peripheral Timing"**).

The window timing must be adjusted to the tolerance of the reference clock, in this case, the LFINTOSC. The LFINTOSC is specified to have +-15% accuracy over temperature and voltage. This means that the windows must be guardbanded to cover this range of oscillator frequencies.

For example, to test the full memory range of the PIC16F1613 (2 Kwords), the CRC peripheral will take 4 ms in Burst mode at 8 MHz. This means that a window period should be added to the Watchdog Timer that would provide an error if the CRC module ended early, was called incorrectly, or a problem with the program counter occurred. This is done by setting the WDT period to 8 ms with a window of 87.5%. The extended window will provide enough guardband for the tolerance of the LFINTOSC, while still providing additional error coverage.

FIGURE A-1: WATCHDOG TIMER WINDOW



APPENDIX B: SMT TIMING

The Signal Measurement Timer (SMT) can be used to obtain exact timings for peripherals or other software events on the PIC16F161X family of devices. The SMT peripheral is particularly useful for slower applications because it is a 24-bit timer instead of the 16-bit timer of Timer 1.

For information on how to set up the SMT for timing operation using the FOSC/4 clock, see Example B-1.

EXAMPLE B-1: SMT INITIALIZATION

```
SMT2CON0 = 0b10100000;    //SMT enabled, rising edges, prescaler 1:1,
                          //counter will halt at PR
SMT2CLKbits.CSEL = 0b001; // Fosc/4
SMTxPR = 0xFFFFFF;
```

Setting the PR register at maximum ensures that the entire 24-bit range of the SMT can be used for timing purposes. If the STP bit (bit 5) is set, the counter will halt at the SMTxPR value. This limits the timing of events to the 24-bit range of the SMT without using overflows.

Once the SMT has been initialized as shown above (see Example B-1), the SMT can be enabled and disabled using the SMTXGO bit of the SMTXCON1 register. To obtain timing information for any event, simply set the SMTXGO bit before the event, then clear the SMTXGO bit immediately after the event. The time will be stored in the SMTXTMRU:H:L registers. The value will be in units of instruction cycles (FOSC/4). This is shown in the CRC sample code in Example B-2.

EXAMPLE B-2: SMT TIMING

```
SMT2CON1bits.SMT2GO = 1; // start SMT timing
//start the Hardware CRC check here.
HW_CRC(LASTCRCADDRESS);
SMT2CON1bits.SMT2GO = 0; // finish SMT timing
```

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.

Trademarks

ISBN: 978-1-63276-582-6

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

Austin, TX
Tel: 512-257-3370

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
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

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

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

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
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: 852-2943-5100
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: 86-571-8792-8115
Fax: 86-571-8792-8116

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: 91-20-3019-1500

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: 886-7-213-7830

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 - Dusseldorf
Tel: 49-2129-3766400

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

Germany - Pforzheim
Tel: 49-7231-424750

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

Italy - Venice
Tel: 39-049-7625286

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

Poland - Warsaw
Tel: 48-22-3325737

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

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

03/25/14