Application Report
SLAA221A–November 2004–Revised June 2018

CRC Implementation With MSP430™ MCUs

MSP430 Applications

ABSTRACT

Cyclic Redundancy Code (CRC) is commonly used to determine the correctness of a data transmission or storage. This application report presents a solution to compute 16-bit and 32-bit CRCs on the ultra-low-power TI MSP430™ microcontrollers for the bitwise algorithm (low memory, low cost) and the table-based algorithm (low MIPS, low power). Both algorithms are presented in C and MSP430 assembly and can be downloaded from www.ti.com/lit/zip/slaa221. Test code to verify the implementations is also included.

Contents

Trademarks

1 Introduction

The fundamental mathematics behind the CRC is polynomial division. An arbitrary message (a fixed block of k information bits) is treated as if each bit were the binary coefficient of a polynomial of degree k–1. The transmitter adds an arbitrary number of bits (called the parity bits) to the end of the message. The parity bits are chosen so that the original message plus the parity bits (called the code word) is evenly divisible by a known polynomial (called the generator polynomial). If the receiver can evenly divide the received code word by the generator polynomial, then the receiver can assume that there were no transmission errors. However, in practice, it is possible to introduce errors into the received message that make detection of these errors impossible for a given generator polynomial. The generator polynomial order and coefficients are related to how many errors it can detect, but that discussion is beyond the scope of this application report.

Many communications protocols, such as HDLC and Ethernet, use 16-bit or 32-bit CRCs. The native implementation for computing and checking a CRC is bit based, which typically makes hardware a more natural fit. However, adding hardware to an application can be complicated by increased cost, increased power, and increased board size. The MSP430 MCUs can calculate the bit-by-bit division very well, but these MCUs are much more efficient when handling data in bytes or words. Fortunately, from the CRC mathematics, a table-based solution has been developed that trades processor cycles for memory and allows a processor to operate on bytes rather than bits. The designer can decide which resource is more important: MIPS (power) or memory (cost), and choose the appropriate solution.

This application report includes the source code to compute 16-bit and 32-bit CRCs on the low-power TI MSP430 MCUs for both the bit-by-bit algorithm and the table-based algorithm. Both algorithms are supplied in C and MSP430 assembly. Projects and test code to verify the CRC implementation are also included and can be run on an MSP430 MCU (C and assembly code) or a PC using Microsoft Visual C++ (C code only).

2 CRC Theory

Assume that the message polynomial `m(x)` of degree `k` by is augmented by multiplying it by an arbitrary polynomial `g(x)`. `g(x)` is the generator polynomial.

$$c(x) = m(x)g(x)$$

The result of this calculation increases the size of the message by the degree of the generator polynomial. This augmented message `c(x)` is referred to as the code word and is of degree `n`. The original message `m(x)` can be recovered by dividing `c(x)` by `g(x)`.

The code word can also be written as the sum of two polynomials, the original message `m(x)`, where each component is increased in degree by `(n-k)`, and an arbitrary polynomial `r(x)` of degree `(n-k)`. This form has the advantage of not disturbing the original message and is the basis for the CRC algorithm.

$$c(x) = m(x)x^{n-k} + r(x)$$

`r(x)` is the remainder polynomial, which is the remainder of $m(x)x^{n-k}$ divided by `g(x)`.

$$r(x) = R_{g(x)}m(x)\ x^{n-k}$$

The binary coefficients of the remainder polynomial are the parity bits that are appended to the end of the original message. Therefore, the code word is simply the original message followed by the parity bits. The following example shows the resulting code word derived from the input ASCII hex values for the test sequence `123456789` and the computed CRC given the CRC-16 generator polynomial.

```
m(x) = 31 32 33 34 35 36 37 38 39
g(x) = 80 05
r(x) = FE E8
c(x) = 31 32 33 34 35 36 37 38 39 FE E8
```

The CRC process adds redundancy to the original message by increasing the number of bits required to transmit the same amount of information, to n bits. These additional bits add the ability to detect errors at the receiver, which is of tremendous value.

Checking for errors at the receiver uses the same functions that were used to compute the CRC. There are two ways to check for errors at the receiver. First, the receiver can divide the code word by the generator polynomial, which should give a remainder of zero. Second, the receiver can extract the original message (because it is only a translated copy in the code word), divide the original message by the generator polynomial, and then compare the result to the parity bits in the code word. If the remainder and the parity bits match, no errors were detected.

Determining a suitable generator polynomial is outside the scope of this application report. Many existing protocols use a few well known and well understood generator polynomials (see Table 1).

Table 1. Common CRC Polynomials

Name (Protocols)	Polynomial
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT (SDLC, HDLC/X.25)	$x^{16} + x^{12} + x^5 + 1$
CRC-32 (Ethernet)	$x^{32} + x^{25} + x^{23} + X^{22} + x^{16} + x^{12} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

3 CRC Algorithms

3.1 Bitwise Method

The bit method is a brute force binary polynomial division where we keep the remainder and throw away the quotient. The remainder is then appended to the message to form the code word. Figure 1 is a Linear Feedback Shift Register (LFSR) implementation of the bitwise CRC algorithm which is the basis for the software bitwise implementation.

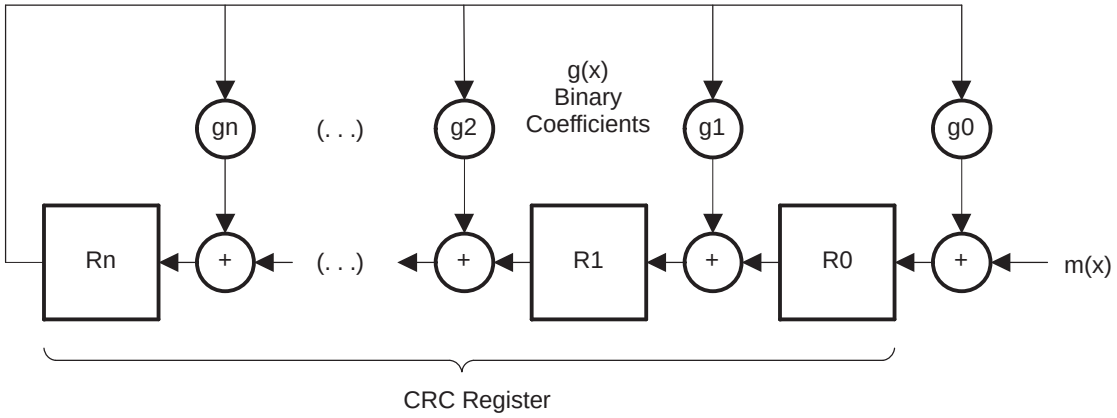

Figure 1. CRC-n Bitwise LFSR Implementation

Algorithm:

1. Initialize the CRC register
2. Shift the CRC left by one bit while shifting in the next message bit
3. If the bit just shifted out is set, XOR the CRC with the generator polynomial
4. Continue with step 2 until there are no message bits left
5. XOR the CRC register with the final XOR value

3.2 Table Method

Processors tend to be more efficient when operating on bytes and words rather than bits. The advantage of the table method is that CRC values can be precalculated and stored in a table, so the fundamental unit of data shifts from bits to bytes. On the MSP430 MCU, these tables can be stored in flash at either run time or link time. The table algorithm included with this document uses table sizes of 256 words to balance between MIPS and memory.

Other table algorithms are available that can use larger tables to further reduce run time, or reduced tables that are a further compromise between table size and run time. Neither of these options is discussed in this application report, because they either require too much memory or the speed advantage is marginal on the MSP430 MCUs.

Algorithm:

1. Initialize the CRC register
2. XOR the CRC most significant byte with the incoming message byte
3. Use this byte to index into the 256 entry table
4. Shift the CRC register to the left by one byte
5. XOR the CRC register with the value indexed into the table
6. Continue with step 2 until no more message bytes are left
7. XOR the CRC register with the final XOR value

4 Implementation

The source files associated with this application report include projects that were developed in IAR Embedded Workbench V3.20A and Microsoft Visual C++ 6.0. The source files should function as expected in later versions. Table 2 is an overview of the project directory structure of the associated archive.

Table 2. Project Directory Structure

Description	Directory
Microsoft Visual C++ project files	\CRC\MSVCpp\
IAR Embedded Workbench project files	\CRC\MSP430\
Source (.c, .a43)	\CRC\src\
Header files (.h)	\CRC\inc\
Data files (.txt, .dat)	\CRC\dat\

4.1 Coding Conventions

- Both the C and assembly versions of the CRC functions have the same name except that the assembly version is preceded by two underscores, `__<name> (arg1, arg2...)`. Two underscores are used because the C compiler generates references to the C functions with a single underscore. This avoids multiply defined symbols and less confusion on the name space.
- Each function starts with `crc` followed by either `16` or `32` to indicate the size of the CRC calculated.
- If the next character is an `r`, then the algorithm is reflected, otherwise it is normal.
- The last part of the name indicates whether it is bitwise, `MakeBitwise`, or table method, `MakeTableMethod`.
- All of the assembly functions are C callable.

4.2 CRC-16 and CRC-32

All of the code is either 16 bit or 32 bit. For most, a C function and C-callable assembly function are provided. Other widths can be supported by extending the source code.

4.3 Normal or Reflected

Some standards such as SDLC expect the data to come into the processor least significant bit first (for example, from a UART). Therefore, the standard specifies that the incoming data and the resulting CRC must be bit reflected. This means that the bits are swapped around their center position (for example, in a 16-bit word, b0↔b15, b1↔b14, b2↔b13, and so on). Rather than waste processor time reflecting the input bits and then the CRC, the algorithm is reflected. So for each normal algorithm, an associated reflected algorithm exists. This saves both CPU time and power.

4.4 Code Size and Performance

For obtaining the following test results, the IAR Embedded Workbench V3.20A was used. Results may differ for other versions of this product. Performance is measured in cycles for the input ASCII sequence 123456789. The C code was measured using the Profiler in the IAR Simulator. The C-callable assembly was measured using the cycle counter in the Simulator by setting the counter to 0 at the functional call then stepping over the function to the next instruction. The C compiler was configured for speed at the highest optimization level. Note that there are two versions of the crc16MakeBitwise and the crc32MakeBitwise functions. The one without the numeric appendix is an implementation equivalent to the assembly version. The one with the *2* is a more C-efficient implementation. This can be seen in Table 3 and Table 4 which give the code size and CPU time for both.

Table 3. CRC-16 Profiled Functions

Function	Code Size (Bytes)	Total CPU Time (Cycles/72 Bytes)	CPU Time (Cycles/Bit)
`Crc16MakeBitwise`	72	1290	17.9
`Crc16MakeBitwise2`	60	1063	14.8
`__crc16MakeBitwise`	64	665	9.2
`Crc16rMakeBitwise`	Not Implemented	Not Implemented	NA
`__crc16rMakeBitwise`	62	645	9.0
`Crc16MakeTableMethod`	52	252	3.5
`__crc16MakeTableMethod`	48	153	2.1
`Crc16rMakeTableMethod`	50	243	3.4
`__crc16rMakeTableMethod`	Not Implemented	Not Implemented	NA

Table 4. CRC-32 Profiled Functions

Function	Code Size (Bytes)	Total CPU Time (Cycles/72 Bytes)	CPU Time (Cycles/Bit)
`crc32MakeBitwise`	98	1490	20.7
`crc32MakeBitwise2`	74	1265	17.6
`__crc32MakeBitwise`	82	781	10.8
`crc32rMakeBitwise`	Not Implemented	Not Implemented	NA
`__crc32rMakeBitwise`	80	766	10.6
`crc32MakeTableMethod`	68	348	4.8
`__crc32MakeTableMethod`	68	224	3.1
`crc32rMakeTableMethod`	68	341	4.7
`__crc32rMakeTableMethod`	Not Implemented	Not Implemented	NA

4.5 Static and Dynamic Table Generation

For the Table Method, the CRC tables can be generated statically at link-time or dynamically at run-time. Static generation saves initialization time unless the device needs to support multiple protocols and there is not enough flash memory to store all possible tables. Although the tables could be generated at run-time and stored in RAM, flash is optimal because the MSP430 MCUs typically have limited RAM. The source to generate the tables is provided in C. An additional step is required to generate the reflected tables from the normal tables.

To generate the tables statically, run the Visual C++ code with the appropriate generator polynomials set in the `crc.h` file. This code generates both 16-bit and 32-bit tables and writes them to corresponding files. Separate table files are generated and can be included in a C style array or an MSP430 assembly style array.

4.6 Test Environment and CRC Verification

Projects are included for both Visual C++ and Embedded Workbench. These projects both include a single main() function for test and verification. The C-callable assembly functions are included in the build only if the symbol `__ICC430__` is defined. This symbol is automatically defined in the Embedded Workbench C environment. These functions are not included when building with Visual C++, so there will not be any unresolved symbols. The test message is the character sequence `123456789`. Note that this is not a string, as there is no `\0` terminating character. After each function call, the resulting CRC is sent to the consol using `printf()`. The user can then verify that the computed result matches the expected result.

Table 5. CRC Parameters and Verification

FORM	CRC-16	CRC-16	CRC-32	CRC-32
Polynomial	0x8005	0x8005	0x04C11DB7	0x04C11DB7
Init CRC register	0x0000	0x0000	0xFFFFFFFF	0xFFFFFFFF
Final XOR value	0x0000	0x0000	0xFFFFFFFF	0xFFFFFFFF
Reflect data (byte)	NO	YES	NO	YES
Reflect CRC (word)	NO	YES	NO	YES
Input sequence (ASCII)	123456789	123456789	123456789	123456789
Expected CRC	0xFEE8	0xBB3D	0xFC891918	0xCBF43926

4.7 Other Software Examples

Texas Instruments also provides CRC software examples for the MSP430F5xx, MSP430F6xx, and MSP430 FRAM families. See the MSP430 Driver Library for these examples.

5 References

1. MSP430x1xx Family User's Guide
2. Mixing C and Assembler with the MSP430
3. Cyclic Redundancy Check Computation: An Implementation Using the TMS320C54x
4. *A Tutorial on CRC Computations*, Gaitonde, Sunil G., Ramabadran, Tenkasi V., IEEE Micro, August 1988, pp. 62–74
5. *A Painless Guide to CRC Error Detection Algorithms*, Williams, Ross N., ftp://ftp.rocksoft.com/papers/crc_v3.txt, August 1993
6. MSP430 Driver Library

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from November 4, 2004 to June 21, 2018	Page
• Editorial changes throughout document	1
• Added Section 4.7, *Other Software Examples*	6

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265