



DKAN0013A

CRC Basics

24 June 2010

Features

- Basic explanation of CRC (appending zeros method)
- Short discussion on brute force calculation of CRC
- Examples of calculating a CRC sum
- C code for simple implementation of CRC

Introduction

A Cyclic Redundancy Check (CRC) is a verification method used to ensure that data being sent is not corrupted during transfer. The use of CRCs is common in communication mediums that transmit digital data, such as WiFi and Ethernet. There is a need to check for communication errors in embedded systems, as technology drives them to be capable of creating and sending larger data packets in a faster and more complex manner. This application note discusses a method for computing and verifying a CRC.

Background

The CRC algorithm computes a checksum for each specific data set that is sent through it. This algorithm utilizes a polynomial key and the transferred data.

The sending system calculates the check or verification code and appends it to the outgoing message. On the receiving end, the data is put through the same process. If the CRC produced at the receiving end does not match the sent CRC, then the data is possibly corrupt. The receiver can request that the data be retransmitted or simply ignore the data. When the CRC matches the received version, the user can be fairly confident that the transmission is not corrupted.

Implementing CRCs is accomplished using two different methods. The brute force method requires more computational resources and can be more time consuming. The lookup table method requires more memory resources and is useful for defined and/or repetitive data sets.

Application

This application discusses the brute force method of computing CRCs. CRC math can be accomplished in software by shifting the data or shifting the polynomial key, then performing the computations. There are several implementations that compute the CRC checksum.

CRC Polynomial

The polynomial key is an important part of the CRC. Keys are not just random polynomials; they are generated using a set of mathematical formulas and are intended to increase the number of mistakes that the CRC process identifies. The polynomial is typically defined by the network protocol or external device. Since there is a well-established set of keys available, the processes for defining keys is not discussed here.

The polynomial key is translated into a binary expression. This is done by expanding the polynomial, and then taking the coefficients. An example polynomial to binary translation is shown below.

Polynomial key:

$$x^8 + x^5 + x^4 + 1$$

Expanded polynomial key:

$$1 \cdot x^8 + 0 \cdot x^7 + 0 \cdot x^6 + 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

Polynomial coefficients:

100110001

The highest power of the polynomial shown is eight, which makes this polynomial an 8-bit CRC, commonly referred to as CRC-8. Four, eight, sixteen, and thirty-two bit are very common polynomial sizes used in CRCs.

Computing a CRC Checksum

The sending device calculates the CRC by appending the data with the correct number of zeros. The data is then processed using binary math before sending it over the communication channel. This method is the “appended zeros” implementation show in Figure 1. Example 1 in the Appendix demonstrates this technique.

First, zeros are appended to the full packet of data. Align the polynomial with the leftmost 1 in the data. The data is XORed with the polynomial. The result is the new data. Again align the polynomial with the leftmost 1 in the data. The process continues until the data is smaller than the polynomial. This CRC is added to the data stream before transmission.

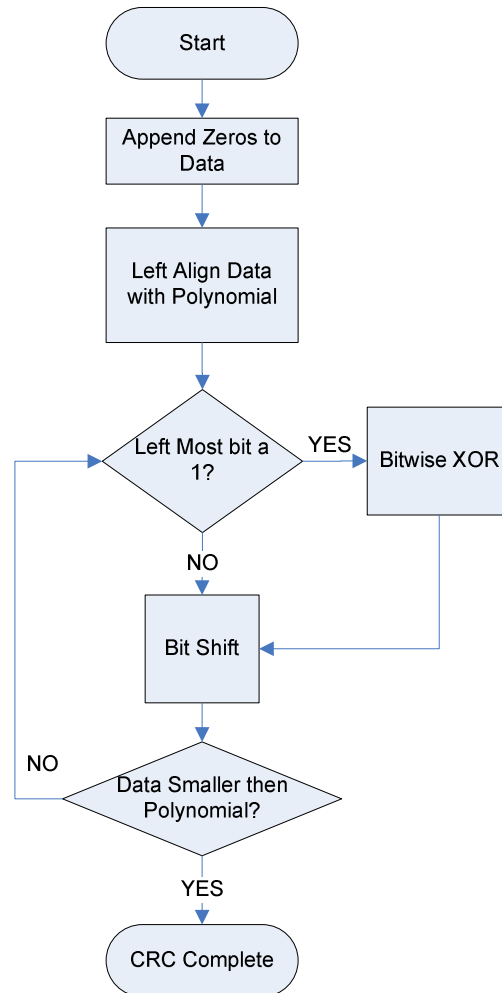


Figure 1. Appending Zeros Algorithm

CRC Verification

On the receiving side of the system, there are two methods explained here for verifying the CRC: the “appended zeros” implementation or the “zero result” implementation.

In the “appended zeros” implementation, the CRC is removed, and the data is appended with n zeros, where n is the highest power of the polynomial. With CRC-8, eight zeros are appended to the end of the data. The CRC is then calculated as show in Figure 1 and compared with the CRC value received.

In the “zero result” implementation, a similar process to that shown in Figure 1 is followed, except zeros are not appended, and data includes the received CRC. Once the process is complete, the result equals zero if the data matches the CRC. Example 2 in the Appendix demonstrates this technique.

Many applications use a lookup table, since they require fewer computational resources. These lookup tables contain CRC values that correspond to specific data sets. This method is effective for applications with defined and/or repetitive data transmissions.

Considerations

Endianness is the term used to describe the byte order of the data. Big-endian means that data is being sent with the most significant bit (MSB) first. Conversely, little-endian is sent with the least significant bit (LSB) first. Endianness matters, because the CRC changes when the byte is flipped from MSB first to LSB first. Typically, the endianness is determined by the type of data being transferred.

Conclusion

A CRC is a simple but effective way to check for errors in data being sent over a noisy medium. This application note discussed how CRCs work and presented methods for calculating and verifying them. A CRC can be easily incorporated into most embedded systems.

Appendix: Examples

Example 1: Appended Zeros Method

The incoming data MSB first is 01101010 10101111 (two bytes). This example uses the polynomial key: $x^8 + x^2 + x^1 + 1$ or 100000111. For the CRC-8, which is 8-bit, the key is actually 9 bits.

```
011010101010111100000000  Data appended with 8 zeros
100000111                 Right shift polynomial key
01101010101010111100000000
100000111                 XOR data with key and right shift
010101101101111000000000
100000111                 XOR and right shift
0010111000111100000000
100000111                 Right shift
0101110001111000000000
100000111                 XOR and right shift
00111011011100000000
100000111                 Right shift
01110110111000000000
100000111                 XOR and right shift
011011100100000000
100000111                 XOR and right shift
0101111000000000
100000111                 XOR and right shift
0011110110000000
100000111                 Right shift
0111101100000000
100000111                 XOR and right shift
01110101100000
100000111                 XOR and right shift
011010001000
100000111                 XOR and right shift
010100101000
100000111                 XOR and right shift
00100110100
100000111                 Right shift
0100110100
100000111                 XOR
000110011                 Result < Polynomial Key
```

The resulting check for the data 0x6AAF is 0x33. In this example, the data stream out of the transmitter (in HEX) is 34 (xx) 6A AF 33. The first byte(s) are protocols for transmission, the next two bytes are the data, and the last byte is the CRC.

Example 2: Zero Result Method

The incoming data MSB first is 01101010 10101111 (two bytes). This example uses the polynomial key: $x^8 + x^2 + x^1 + 1$ or 100000111. For the CRC-8, which is 8-bit, the key is actually 9 bits.

```

011010101010111100110011  Data and CRC received
100000111                 Right shift polynomial key
011010101010111100110011
100000111                 XOR data with key and right shift
01010110110111100110011
100000111                 XOR and right shift
0010111000111100110011
100000111                 Right shift
010111000111100110011
100000111                 XOR and right shift
00111011011100110011
100000111                 Right shift
0111011011100110011
100000111                 XOR and right shift
011011100100110011
100000111                 XOR and right shift
01011111000110011
100000111                 XOR and right shift
0011110110110011
100000111                 Right shift
011110110110011
100000111                 XOR and right shift
01110101010011
100000111                 XOR and right shift
0110100100011
100000111                 XOR and right shift
010100011011
100000111                 XOR and right shift
00100000111
100000111                 Right shift
0100000111
100000111                 XOR
000000000                 Result < polynomial Key
    
```

The check for 0x6AAF is 0x00 using the “zero result method,” which confirms that the proper data was received.

Disclaimer

This document is for informational use only and is subject to change without prior notice. Digi-Key makes no commitment to update or keep current the information contained herein. Digi-Key does not guarantee or warrant that any information provided is accurate, complete, or correct and disclaims any and all liability associated with the use of the information contained herein. The use of this information and Digi-Key's liability is subject to Digi-Key's standard Terms & Conditions which can be found at www.digi-key.com by clicking on the Terms & Conditions link at the bottom of the web page.

No license, whether express, implied, arising by estoppel or otherwise is granted under any intellectual property or other rights of Digi-Key or others.

Trademarks

DIGI-KEY® is a registered trademark of Digi-Key Corporation. All other trademarks, service marks and product names contained herein are the sole property of their respective owner and their use is for informational purposes only and does not imply any endorsement, recommendation, sponsorship or approval by the trademark owner of the contents.

Copyright

Use of this document is limited to customer's internal business use for the evaluation and purchase of products. No permission is granted to the user to copy, print, store, distribute, transmit, display in public or modify the content of this document in any way for any other purpose.

© Copyright 2010 Digi-Key Corporation. All rights reserved.