ANALOG DEVICES

AN-1251
APPLICATION NOTE

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

ADAS1000 Cyclic Redundancy Code

by Padraic O'Reilly

INTRODUCTION

This application note details the ADAS1000 cyclic redundancy code. The aim of error detection is to enable the receiver of a message to determine whether the message has been corrupted.

To do this, the ADAS1000 constructs a code (called a checksum) that is a function of the message and appends it to the message.

The CRC algorithms simply treat the message as an enormous polynomial, dividing it by another fixed polynomial using Modulo 2 arithmetic for the coefficients, and making the remainder from this division the checksum. Upon receipt of the message, the receiver can perform the same division and compare the remainder with the checksum.

This application note describes how the code (checksum) for the ADAS1000 is calculated.

TABLE OF CONTENTS

REVISION HISTORY

3/14—Revision 0: Initial Version

CYCLIC REDUNDANCY CODE

COMPUTING THE CRC USING THE ADAS1000

128 kHz Data Rate Example

For a 128 kHz data rate, the polynomial used is

$$x^{16} + x^{12} + x^5 + x^0$$

This is also known as the CRC-16-CCITT polynomial.

When translated into hexadecimal, this polynomial reads 0x11021 (polynomial: 10001000000100001).

The ADAS1000 presets the CRC register to all ones (0xFFFF) where the CRC register is 1111111111111111.

For each bit in the message, a series of six steps occurs.

1. For each data bit (below for 16-bits), the remainder shifts left by one.
2. The MSB of the remainder is XOR'd with the data bit.
3. If the resulting MSB is 1, the remainder is then XOR'd with the polynomial.
4. If the resulting MSB is 0, the remainder is not XOR'd with the polynomial.
5. Repeat Step 1 through Step 4 for each data bit.

Note that data bits are shifted MSB first.

The result in the CRC register, after all message bits are shifted through the register as described, is the CRC code.

For example, when transmitting the header only using the ADAS1000, one expects the header to contain the message: 0x8000 (1000000000000000).

```
  1111111111111111  initial CRC is pre-set all ones

< 1111111111111110 shift CRC
^ 1                XOR with data bit 15
= 1111111111111110  CRC after data bit 15

< 1111111111111100 shift CRC
^ 0                XOR with data bit 14
= 1111111111111100
^ 10001000000100001 XOR with poly
= 1110111111011101  CRC after data bit 14

< 1110111111011010 shift CRC
^ 0                XOR with data bit 13
= 1110111111011010
^ 10001000000100001 XOR with poly
=  1100111110011011 CRC after data bit 13

< 1100111110011010 shift CRC
^ 0                XOR with data bit 12
= 1100111110011010
^ 10001000000100001 XOR with poly
=  1000111100010111 CRC after data bit 12

< 1000111100010110 shift CRC
^ 0                XOR with data bit 11
= 1000111100010110
^ 10001000000100001 XOR with poly
=  0000111000001111 CRC after data bit 11

< 0000111000001110 shift CRC
^ 0                XOR with data bit 10
=  0000111000001110 CRC after data bit 10

< 0000111000001100 shift CRC
^ 0                XOR with data bit 9
=  0000111000001100 CRC after data bit 9
```

```
< 00111000001111000 shift CRC
^ 0                 XOR with data bit 8
=  0111000001111000 CRC after data bit 8

< 01110000011110000 shift CRC
^ 0                 XOR with data bit 7
=  1110000011110000 CRC after data bit 7

< 11100000111100000 shift CRC
^ 0                 XOR with data bit 6
= 11100000111100000
^ 10001000000100001 XOR with poly
=  1101000111000001 CRC after data bit 6

< 11010001110000010 shift CRC
^ 0                 XOR with data bit 5
= 11010001110000010
^ 10001000000100001 XOR with poly
=  1011001110100011 CRC after data bit 5

< 10110011101000110 shift CRC
^ 0                 XOR with data bit 4
= 10110011101000110
^ 10001000000100001 XOR with poly
=  0111011101100111 CRC after data bit 4

< 01110111011001110 shift CRC
^ 0                 XOR with data bit 3
=  1110111011001110 CRC after data bit 3

< 11101110110011100 shift CRC
^ 0                 XOR with data bit 2
= 11101110110011100
^ 10001000000100001 XOR with poly
=  1100110110111101 CRC after data bit 2

< 11001101101111010 shift CRC
^ 0                 XOR with data bit 1
= 11001101101111010
^ 10001000000100001 XOR with poly
=  1000101101011011 CRC after data bit 1

< 10001011010110110 shift CRC
^ 0                 XOR with data bit 0
= 10001011010110110
^ 10001000000100001 XOR with poly
=  0000011010010111 CRC after data bit 0
```

In this example, the CRC result is 0x0697.

The ADAS1000 inverts message 0x0697 (0000011010010111) before transmission, as stated in the data sheet, and the message becomes 0xF968 (1111100101101000).

The full transmission should contain 0x8000F968. The message is appended by the CRC.

VERIFYING THE CRC ON THE RECEIVER SIDE

128 kHz Data Rate Example

To verify that data was correctly received, the software should compute a CRC on both the data and the received checksum.

The same steps (Step 1 through Step 5 in the Computing the CRC Using the ADAS1000 section) are applied to the each data bit and received checksum 0x8000F968 (10000000000000001111100101101000).

```
1111111111111111    initial CRC is pre-set all ones

< 1111111111111110 shift CRC
^ 1                XOR with data bit 31
= 1111111111111110  CRC after data bit 31

< 1111111111111100 shift CRC
^ 0                XOR with data bit 30
= 1111111111111100
^ 1000100000100001 XOR with poly
= 1110111111011101  CRC after data bit 30

< 1110111111011010 shift CRC
^ 0                XOR with data bit 29
= 1110111111011010
^ 1000100000100001 XOR with poly
=  1100111110011011 CRC after data bit 29

< 1100111110011010 shift CRC
^ 0                XOR with data bit 28
= 1100111110011010
^ 1000100000100001 XOR with poly
=  1000111100010111 CRC after data bit 28

< 1000111100010110 shift CRC
^ 0                XOR with data bit 27
= 1000111100010110
^ 1000100000100001 XOR with poly
=  0000111000001111 CRC after data bit 27

< 0000111000011110 shift CRC
^ 0                XOR with data bit 26
=  0001110000011110 CRC after data bit 26

< 0001110000111100 shift CRC
^ 0                XOR with data bit 25
=  0011100000111100 CRC after data bit 25

< 0011100001111000 shift CRC
^ 0                XOR with data bit 24
=  0111000001111000 CRC after data bit 24

< 0111000011110000 shift CRC
^ 0                XOR with data bit 23
=  1110000011110000 CRC after data bit 23

< 1110000111100000 shift CRC
^ 0                XOR with data bit 22
= 1110000111100000
^ 1000100000100001 XOR with poly
=  1101000111000001 CRC after data bit 22

< 1101000111000010 shift CRC
```

```
^ 0                 XOR with data bit 21
= 11010001110000010
^ 10001000000100001 XOR with poly
=  1011001110100011 CRC after data bit 21

< 10110011101000110 shift CRC
^ 0                 XOR with data bit 20
= 10110011101000110
^ 10001000000100001 XOR with poly
=  0111011101100111 CRC after data bit 20

< 01110111011001110 shift CRC
^ 0                 XOR with data bit 19
=  1110111011001110 CRC after data bit 19

< 11101110110011100 shift CRC
^ 0                 XOR with data bit 18
= 11101110110011100
^ 10001000000100001 XOR with poly
=  1100110110111101 CRC after data bit 18

< 11001101101111010 shift CRC
^ 0                 XOR with data bit 17
= 11001101101111010
^ 10001000000100001 XOR with poly
=  1000101101011011 CRC after data bit 17

< 10001011010110110 shift CRC
^ 0                 XOR with data bit 16
= 10001011010110110
^ 10001000000100001 XOR with poly
=  0000011010010111 CRC after data bit 16

< 00000110100101110 shift CRC
^ 1                 XOR  with RX CRC bit 15
= 10000110100101110
^ 10001000000100001 XOR with poly
=  0001110100001111 CRC after RX CRC bit 15

< 00011101000011110 shift CRC
^ 1                 XOR  with RX CRC bit 14
= 10011101000011110
^ 10001000000100001 XOR with poly
=  0010101000111111 CRC after RX CRC bit 14

< 00101010001111110 shift CRC
^ 1                 XOR  with RX CRC bit 13
= 10101010001111110
^ 10001000000100001 XOR with poly
=  0100010001011111 CRC after RX CRC bit 13

< 01000100010111110 shift CRC
^ 1                 XOR  with RX CRC bit 12
= 11000100010111110
^ 10001000000100001 XOR with poly
=  1001100010011111 CRC after RX CRC bit 12

< 10011000100111110 shift CRC
^ 1                 XOR  with RX CRC bit 11
=  0011000100111110 CRC after RX CRC bit 11
```

```
< 00110001001111100 shift CRC
^ 0                 XOR  with RX CRC bit 10
=  0110001001111100 CRC after RX CRC bit 10

< 01100010011111000 shift CRC
^ 0                 XOR  with RX CRC bit 9
=  1100010011111000 CRC after RX CRC bit 9

< 11000100111110000 shift CRC
^ 1                 XOR  with RX CRC bit 8
=  1000100111110000 CRC after RX CRC bit 8

< 10001001111100000 shift CRC
^ 0                 XOR  with RX CRC bit 7
= 10001001111100000
^ 10001000000100001 XOR with poly
=  0000001111000001 CRC after RX CRC bit 7

< 00000011110000010 shift CRC
^ 1                 XOR  with RX CRC bit 6
= 10000011110000010
^ 10001000000100001 XOR with poly
=  0001011110100011 CRC after RX CRC bit 6


< 00010111101000110 shift CRC
^ 1                 XOR  with RX CRC bit 5
= 10010111101000110
^ 10001000000100001 XOR with poly
=  0011111101100111 CRC after RX CRC bit 5

< 00111111011001110 shift CRC
^ 0                 XOR  with RX CRC bit 4
=  0111111011001110 CRC after RX CRC bit 4

< 01111110110011100 shift CRC
^ 1                 XOR  with RX CRC bit 3
= 11111110110011100
^ 10001000000100001 XOR with poly
=  1110110110111101 CRC after RX CRC bit 3

< 11101101101111010 shift CRC
^ 0                 XOR  with RX CRC bit 2
= 11101101101111010
^ 10001000000100001 XOR with poly
=  1100101101011011 CRC after RX CRC bit 2

< 11001011010110110 shift CRC
^ 0                 XOR  with RX CRC bit 1
= 11001011010110110
^ 10001000000100001 XOR with poly
=  1000011010010111 CRC after RX CRC bit 1

< 10000110100101110 shift CRC
^ 0                 XOR  with RX CRC bit 0
= 10000110100101110
^ 10001000000100001 XOR with poly
=  0001110100001111 CRC after RX CRC bit 0
```

If the CRC is correct, the remainder on the receive side equals 0x1D0F, the check constant.

2 kHz and 16 kHz Data Rate Examples

For 2 kHz and 16 kHz data rates, the polynomial used is

$$x^{24} + x^{22} + x^{20} + x^{19} + x^{18} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^{8} + x^{7} + x^{6} + x^{3} + x^{1} + x^{0}$$

This is also known as the 24-bit CRC polynomial.

When translated into hexadecimal, this polynomial reads 0x15D6DCB (polynomial: 10101110101101101110010 11).

The ADAS1000 presets the CRC register to all ones (0xFFFFFF) where the CRC register is 111111111111111111111111.

For each bit in the message, a series of steps occurs.

1. For each data bit, the remainder shifts left by one.
2. The MSB of the remainder is XOR'd with the data bit.
3. If the resulting MSB is 1, the remainder is then XOR'd with the polynomial.
4. If the resulting MSB is 0, the remainder is not XOR'd with the polynomial.
5. Repeat Step 1 through Step 4 for each data bit.

Note that data bits are shifted MSB first.

The result in the CRC register, after all message bits are shifted through the register as described, is the CRC code.

The ADAS1000 inverts message CRC before transmission, as stated in the data sheet, and the message becomes the data and the inverted CRC.

To verify that data was correctly received, the software should compute a CRC on both the data and the received checksum.

The same steps are applied to each data bit and received checksum and data.

If the CRC is correct, the remainder on the receive side equals 0x15A0BA, the check constant.