# APPLICATION NOTE

# Calculating the CRC for Messages using the TCP/IP Communication Protocol

## Purpose

This App Note explains step by step how to calculate the Cyclic Redundancy Check (CRC) that is part of every message sent between the HLC and the Host when using the TCP/IP communication protocol. The Example is intended to help customers who are not familiar with the CRC to understand the calculation and determine how to best implement it in their Host controller code.

## Introduction

A cyclic redundancy check (CRC) is an error-detecting technique that is commonly used to detect accidental changes to data in networks. A check value is attached to the original data prior to transmitting it. When the data is received after being transmitted through a network the check value can be used to determine if the data has been corrupted.

MagneMotion uses the CRC-CCITT standard with an initial value of 0xFFFF and the mask polynomial 0x1021 = $2^{16}+2^{12}+2^{5}+1 = 10001000000100001_2$. Note that the hexadecimal representation (0x1021) omits the leading 1 of the binary representation according to common practice.

## Calculating the CRC - Theory

The CRC value is calculated in a polynomial long division, which takes the message as the dividend and the mask polynomial as the divisor. The quotient is discarded and the remainder becomes the resulting check value.

The CRC-CCITT is a 16 bit CRC, which means the check value will be 16 bits long and the mask polynomial is of the order 16 and therefore has 17 terms as shown in the previous section.

The main advantage of using a CRC for error-detecting is that they are simple to implement in binary hardware. However, this App Note will not discuss the implementation of the CRC

calculation since a large number of efficient commercial or freeware implementations are available. Instead the following section will go through a full example of calculating the check sum for an example message to give the reader a step by step instruction that can be used to select or develop a CRC implementation.

## Calculating the CRC - Example

### Message

In this example we will calculate the CRC for the following example message:

$$AB_{16}\ BA_{16}\ 03_{16}\ 00_{16}\ CRCByte1_{16}\ CRCByte2_{16}$$

Following the MagneMotion TCP/IP communications format:

- The first two Bytes ($AB_{16}$ and $BA_{16}$) represent the two start bytes.
- The third byte represents the message length (number of bytes after the length byte including the CRC bytes) and is correctly indicating that there are 3 more bytes in the message.
- The fourth byte represents the message type. Message type 00 is not used for an actual message within MagneMotion's TCP/IP communications protocol and it is used as an example here to avoid any confusion with a real message. Normally a real message would include further bytes that contain message-specific information after the message type byte. In order to keep this step by step example reasonably short, a 00 message type with no message-specific content will be used.
- The last two bytes will contain the 16 bit CRC value that will be calculated in the example below.

### Initial Value and Appendix

In order to calculate the CRC value, the original message has to be rewritten and modified according to standard CRC theory.

First, the bytes of the message in front of the two CRC bytes are transformed to their binary representation.

$$AB_{16}\ BA_{16}\ 03_{16}\ 00_{16}$$
$$=$$

$$1010\ 1011\ 1011\ 1010\ 0000\ 0011\ 0000\ 0000_2$$

Because an initial value of 0xFFFF is used, the first 16 bits of the message need to be inverted. The message thus becomes:

$$\underline{0101\ 0100\ 0100\ 0101}\ 0000\ 0011\ 0000\ 0000_2$$

Finally, an appendix consisting of zeros is added to the message to allow the whole message to be processed using the sliding XOR method described in the following sections. The length of the appendix is determined by the length of the mask polynomial minus 1. In this example a 17 bit mask is used and thus 16 zeroes are appended to the message:

$$0101\ 0100\ 0100\ 0101\ 0000\ 0011\ 0000\ 0000\ \underline{0000\ 0000\ 0000\ 0000_2}$$

## Mask Polynomial

As mentioned above, the 17 bit mask polynomial 0x1021 is used and its binary representation is:

$$1\ 0001\ 0000\ 0010\ 0001_2$$

## Computation of the polynomial long division

The extended and partially inverted message is now used as the dividend with the polynomial mask as the divisor. In every step, the dividend is divided using the XOR (exclusive OR; 1 XOR 1 = 0, 1 XOR 0 = 1, 0 XOR 1 = 1, 0 XOR 0 = 0) operator starting with the first 1 from the left.

The first step aligns the mask with the first 1 from the left of the message and carries out the XOR operation for the 17 digits where the mask overlaps the message. The tail end of the message remains unchanged.

```
0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
  0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

This operation is repeated until a remainder where the leftmost 1 is located less than 17 bits from the right is found.

```
0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
    0 0 0 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
          0 0 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
            0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
              0 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                  1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                  0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                    1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                    0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                      1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                      0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                        1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                        0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                          1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                          0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                            1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
                            0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                              1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
```

0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0

The remainder is:

$$0001\ 1100\ 1100\ 0100_2$$
$$=$$
$$1C_{16}\ C4_{16}$$

And it follows that:

$$CRCByte1 = 1C_{16}$$
$$CRCByte2 = C4_{16}$$

With that, the original message is completed to:

$$AB_{16}\ BA_{16}\ 03_{16}\ 00_{16}\ 1C_{16}\ C4_{16}$$

## Error Detection

When the message including the CRC value is received, the same CRC calculation can be carried out on the whole message that has been received (including the CRC bytes).

$$AB_{16}\ BA_{16}\ 03_{16}\ 00_{16}\ 1C_{16}\ C4_{16}$$
$$=$$
$$1010\ 1011\ 1011\ 1010\ 0000\ 0011\ 0000\ 0000\ 0001\ 1100\ 1100\ 0100_2$$

As before, the first 16 bits need to be inverted, but no appendix is required:

$$\underline{0101\ 0100\ 0100\ 0101}\ 0000\ 0011\ 0000\ 0000\ 0001\ 1100\ 1100\ 0100_2$$

The CRC calculation then follows the process introduced above:

```
0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
  1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
  0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
      1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
      0 0 0 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
            1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
            0 0 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
              1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
              0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
                1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                0 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
                  1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                  0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
                    1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                    0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
                      1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                      0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0
                        1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                        0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 0 0
                          1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                          0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0
                            1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                            0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0
                              1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                              0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0
                                1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
                                0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

When the remainder is zero, it is very likely that the transmitted message has not been corrupted accidentally. Note that it is comparatively easy to intentionally manipulate the message with errors that aren't detected by CRC, but it is assumed that this case is not relevant to detecting errors in MagneMotion TCP/IP messages.

## Summary

The CRC is used in MagneMotion's TCP/IP Communications Protocol to detect transmission errors. The step by step example in this App Note describes the process to calculate the CRC value and how to use it to detect accidental data corruption.

For any questions related to the content of this document, please contact MagneMotion Customer Support.
Phone: +1 978-757-9102 (9am – 5pm EST)

Email: customersupport@magnemotion.com

## Related Documents:

990000436 – Manual, Host Controller TCP-IP Communication Protocol

## More Information

MagneMotion Website: www.magnemotion.com

Questions & Comments: www.magnemotion.com/about-magnemotion/contact.cfm

## Revision History

| Rev. | Change Description |
|---|---|
| A | Initial release |