APPLICATION NOTE 7072

GUIDELINES TO IMPLEMENT CRC PROGRAMMING FOR THE MAX22000 CONFIGURABLE ANALOG IO

Abstract:
The MAX22000 configurable I/O is a high-performance, feature-rich analog I/O IC designed for industrial process control systems. A microcontroller-compatible SPI provides access to many advanced features. For added safety, a hardware CRC circuit optionally protects this SPI against bit errors. This application note provides example C code implementing CRC generation and error-detection algorithms in the microcontroller.

Introduction

The MAX22000 is a high-performance, configurable analog I/O IC designed for industrial process-control systems. A serial peripheral interface (SPI) allows a microcontroller to monitor and control most aspects of the MAX22000. To enhance robustness, a hardware cyclic redundancy check (CRC) circuit in the MAX22000 can optionally protect all data communication between it and a microcontroller against bit errors. However, enabling the CRC feature in the MAX22000 is not enough. The microcontroller must also implement the same CRC algorithm in software, both to append check bits to data being sent to the MAX22000, as well as to verify those being received from it. One way to accomplish this is to inspect the data sheet and use it to create custom firmware to implement the necessary CRC functionality. To provide a faster, proven solution, a series of functions are presented in this application note instead. They are written in C and should prove easy to port to any common microcontroller. For detailed information of the MAX22000 pins, operating modes, and control registers, refer to the MAX22000 data sheet.

CRC Error Detection on the Serial Interface

The MAX22000 CRC error detection of the serial data can be enabled to minimize incorrect operation or misinformation due to data corruption of the SDI and SDO signals. If error detection is enabled by setting the CRC_EN bit in register GEN_CNFG (0x02), then the MAX22000 does the following:

- Performs error detection on the SDI data that it receives from the microcontroller, and
- Calculates a CRC on the SDO data that it sends to the microcontroller and appends a check byte to the SDO data that it sends to the microcontroller

This ensures that both the data that it receives from the microcontroller (setting/configuration) and the data that it sends to the microcontroller (diagnostics/status) have a low likelihood of undetected errors.

Input Data on SDI (Microcontroller Write Cycle to the MAX22000)

Setting the CRC_EN bit to 1 enables CRC error detection. A CRC frame-check sequence (FCS) is sent along with each serial transaction. The 8-bit FCS is based on the generator polynomial $X^8 + X^5 + X^4 + X^0$ with CRC starting value = 0. When CRC is enabled, the MAX22000 expects a check byte appended to the SDI data that it receives.

The 8-bit CRC bits are calculated on all the data sent in one SPI command including the Register byte, which contains the R/W bit. Thus, the CRC is calculated from 32 bits.

The MAX22000 verifies the received FCS bits and, if no error is detected, the MAX22000 sets the register according to the SDI data. If a CRC error is detected, the MAX22000 ignores the register write request. Instead, the MAX22000 sets the CRC_INT bit in register GEN_INT (0x07).

If the mask register (GEN_INTEN) is set to enable the CRC_INTEN bit, the MAX22000 also sets the INT pin low to provide an interrupt to the microcontroller to further indicate a communication error on the SPI.

Output Data on SDO (Microcontroller Read Cycle from the MAX22000)

The MAX22000 appends a check byte to the SDO data, which the microcontroller must read immediately after any read operation.

Source Code

This application note provides C source code to allow CRC generation and CRC checking. The MAX22000 communicates with a microcontroller using 4-byte (32-bit) packets (the Address byte plus 3 data bytes). The source code provides a CRC encoder, which can be used for both read and write operation.

To use the following function, fill BYTE1 with the Address byte as sent to the MAX22000, including the Read/Write bit.

In the case of a write, fill BYTE2, BYTE3, BYTE4 with the data to be sent to the MAX22000. The function will return the CRC byte, which must be appended to that write transaction.

In the case of a read, fill BYTE2, BYTE3, BYTE4 with the data received from the MAX22000. The function will return the CRC code, which should be compared with the byte received from the MAX22000.

Please note that, in these code examples, "byte" is an alias for an 8-bit unsigned value, which is sometimes labeled differently, such as UINT8.

```
public byte crc8(byte BYTE1, byte BYTE2, byte BYTE3, byte BYTE4)
        {
            byte crc8_start = 0x00;
            byte crc8_poly  = 0x8c; // rotated 0x31, which is our polynomial
```

```
            byte crc_result = crc8_start;

            // BYTE1
            for (int i=0; i<8; i++)
            {
                if( ( (( BYTE1>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
                { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
                else
                { crc_result = (byte) (crc_result>>1);              }
            }

            // BYTE2
            for (int i=0; i<8; i++)
            {
                if( ( (( BYTE2>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
                { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
                else
                { crc_result = (byte) (crc_result>>1);              }
            }

            // BYTE3
            for (int i=0; i<8; i++)
            {
                if( ( (( BYTE3>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
                { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
                else
                { crc_result = (byte) (crc_result>>1);              }
            }

            // BYTE4
            for (int i=0; i<8; i++)
            {
                if( ( (( BYTE4>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
```

```
            { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
            else
            { crc_result = (byte) (crc_result>>1);              }
        }

        return crc_result;
    }
```

Conclusion

This application note has shown how to calculate the CRC for the MAX22000 read and write operations algorithms on a microcontroller. This code was tested using the MAX22000EVKIT and the corresponding GUI. By taking advantage of the C code examples in this application note, the engineer has a proven solution to implement this extra data communication protection. In some cases, some benchmarking should be performed on the target microcontroller, especially if fast execution speed is a priority.