Keywords: MAX14915, high-side switch, CRC, error detection, SPI, digital output

APPLICATION NOTE 6633

GUIDELINES TO IMPLEMENT CRC PROGRAMMING FOR THE MAX14915 OCTAL, INDUSTRIAL, HIGH-SIDE SWITCH

Abstract: The MAX14915 octal industrial high-side switch is a high-performance, feature-rich switch capable of delivering 15W nominal to each of eight loads. A microcontroller-compatible SPI provides access to many advanced features. For added safety, a hardware CRC circuit optionally protects this SPI interface against bit errors. This application note provides example C-code implementing CRC generation and detection algorithms in the microcontroller.

Introduction

The MAX14915 is a high-performance, 8-channel, industrial high-side switch with a rich and advanced feature set. An SPI interface allows a microcontroller to monitor and control most aspects of the MAX14915. To enhance robustness, a hardware cyclic redundancy check (CRC) circuit in the MAX14915 can optionally protect all data communication between it and a microcontroller against bit errors. However, enabling the CRC feature in the MAX14915 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 MAX14915, 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 and proven solution, a series of functions are presented in this application note instead. They are written in C and should prove easy to port over to any common microcontroller. For detailed information of the MAX14915 pins, operating modes, and control registers, refer to the MAX14915 data sheet.

CRC Error Detection on the Serial Interface

The MAX14915 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 CRCEN pin = 1, then the MAX14915 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 diagnostics/status 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.

CRC error detection is supported for both SPI-addressed and daisy-chain operation modes, and for standard as well as burst read/write cycles.

Input Data on SDI (Read Cycle)

Setting the CRCEN input high enables CRC error detection. A CRC frame-check sequence (FCS) is sent along with each serial transaction. The 5-bit FCS is based on the generator polynomial $X^5 + X^4 + X^2 + 1$ with CRC starting value = 11111. When CRC is enabled, the MAX14915 expects a check byte appended to the SDI program/configure data that it receives. **Figure 1** shows the check byte format.

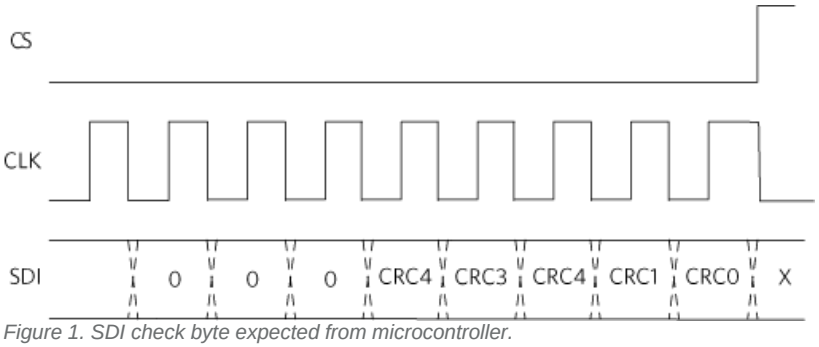

Figure 1. SDI check byte expected from microcontroller.

The 5-bit FCS bits CRC[4:0] are calculated on all the data sent in one SPI command including the three “0” in the MSBs of the check byte. Thus, the CRC is calculated from 8+3 bits up to 24+3 bits in case of a burst command. CRC0 is the LSB of the FCS.

The MAX14915 verifies the received FCS bits and if no error is detected, the MAX14915 sets the OUT_ output switches and/or changes device configuration per the SDI data. If a CRC error is detected, the MAX14915 does not change the OUT_ outputs and/or does not change the device configuration. Instead, the MAX14915 sets the COMERR logic output low, i.e., the open-drain COMERR output transistor is turned on.

If the mask register is set to enable CRCfault bit in the Global Faults register, the MAX14915 also sets the FAULT pin low to provide an interrupt to a microcontroller to further indicate a communication error on the SPI interface.

Output Data on SDO (Write Cycle)

The check byte the MAX14915 appends to the SDO data has the format shown in **Figure 2** when the DAISY pin is low:

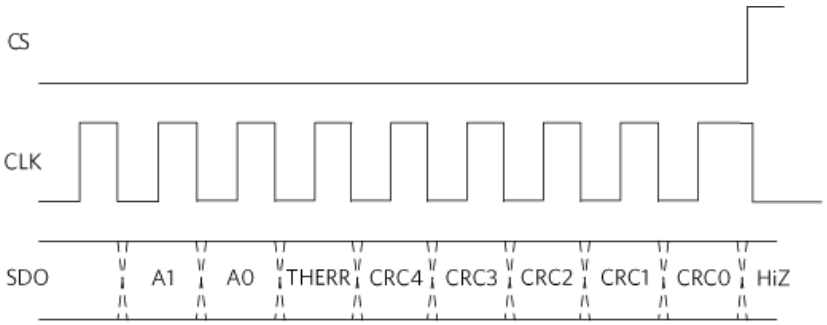

Figure 2. SDO check byte sent by MAX14915 in addressed SPI mode.

A1 and A0 are the levels for the A1 and A0 pins, while the THERR bit is set when a chip thermal shutdown event has occurred. CRC[4:0] are the five CRC bits the MAX14915 calculates on the SDO data, including the A1, A0, and THERR values. This allows the microcontroller to check for errors on the SDO data received from the MAX14915.

Source Code

This application note provides C source code for implementing a CRC generator and a CRC checker. The MAX14915 communicates with a microcontroller using either single-byte packets, or double-byte packets. The source code provides an encoder and decoder for each case:

- CRC5encode_2byte (for transmitting 2 bytes)
- CRC5encode_1byte (for transmitting 1 byte)
- CRC5check_2byte (for checking 2 bytes response from the MAX14915)
- CRC5check_1byte (for checking 1 byte response from the MAX14915)

Besides the choice of single-byte or double-byte packets for communication with the MAX14915, note that in these code examples "byte" is an alias for an 8-bit unsigned value, sometimes labeled differently, for example, UINT8.

Send1 is the first byte, and *send2* is the second byte to send to the MAX14915. The code should transmit *send1* followed by *send2*, then crc_code to the MAX14915.

```
crc_code = crc5encode (send1, send2);
```

The user should send 3 bytes, *send1*, followed by *send2*, and then crc_code, over the SPI interface. While the microcontroller sends configuration settings to the MAX14915 over the SPI interface, the MAX14915 simultaneously returns status information back to the microcontroller. This is the way to check if the crc_code received from MAX14915 is correct:

```
crc_ret = crc5decode (ret1, ret2);
```

The byte result "crc_ret" should be identical to the third byte received from the MAX14915.

Figure 3 and **Figure 4** show data locations mentioned in the comments of the C functions. Download the spreadsheet

			POLY	21	15 10101	
	DATA on SDI		CRC5			
			dec	hex	bin	
init			31	1F	11111	
1	A1	0	11	0B	01011	
2	A0	0	22	16	10110	
3	BRST	0	25	19	11001	
4	R3	0	7	07	00111	
5	R2	0	14	0E	01110	
6	R1	0	28	1C	11100	
7	R0	1	24	18	11000	
8	R/W	1	16	10	10000	
9	D7	1	0	00	00000	
10	D6	1	21	15	10101	
11	D5	1	10	0A	01010	
12	D4	1	1	01	00001	
13	D3	1	23	17	10111	
14	D2	1	14	0E	01110	
15	D1	1	9	09	01001	
16	D0	1	7	07	00111	
17	0	0	14	0E	01110	
18	0	1	9	09	01001	
19	0	0	18	12	10010	This is CRC5 to send to MAX14915
20	CRC4	1	4	04	00100	
21	CRC3	0	8	08	01000	
22	CRC2	0	16	10	10000	
23	CRC1	1	0	00	00000	
24	CRC0	0	0	00	00000	

Figure 3. CRC5 data to send to MAX14915 on SDI.

			POLY	21	15 10101	
	DATA on SDO		CRC5			
			dec	hex	bin	
init			-	-	-	
1	-	0	-	-	-	
2	-	0	31	1F	11111	
3	SHTVDD	0	11	0B	01011	
4	OWOnF	0	22	16	10110	
5	OWOffF	0	25	19	11001	
6	OvrCurr	0	7	07	00111	
7	OvldF	0	14	0E	01110	
8	GLOBLF	1	9	09	01001	
9	F8	0	18	12	10010	
10	F7	0	17	11	10001	
11	F6	0	23	17	10111	
12	F5	0	27	1B	11011	
13	F4	0	3	03	00011	
14	F3	0	6	06	00110	
15	F2	0	12	0C	01100	
16	F1	1	13	0D	01101	
17	A1	0	26	1A	11010	
18	A0	0	1	01	00001	
19	THERR	0	2	02	00010	This is CRC5 sent by MAX14915
20	CRC4	0	4	04	00100	
21	CRC3	0	8	08	01000	
22	CRC2	0	16	10	10000	
23	CRC1	1	0	00	00000	
24	CRC0	0	0	00	00000	

Figure 4. CRC5 data sent by MAX14915 on SDO.

```
public byte crc5encode(byte BYTE1, byte BYTE2)
    {
        byte crc5_start = 0x1f;
        byte crc5_poly = 0x15;
        byte crc_result = crc5_start;

        // BYTE1
        for (int i=0; i<8; i++)
        {
            if( ((( BYTE1>>(7-i) ) &0x01) ^ ((crc_result & 0x10)>>4)) > 0 ) // IF(XOR(C6;BITAND(D5;2^4)/2^4)
            { crc_result = (byte) (crc5_poly ^ ((crc_result<<1) & 0x1f)); } // BITXOR($D$1;BITAND((D5*2);31))
            else
            { crc_result = (byte)((crc_result<<1) & 0x1f); } // shift left, keep only lower 6 bits
        }

        // BYTE2
        for (int i=0; i<8; i++)
        {
```

```
        if( ((( BYTE2>>(7-i) ) &0x01) ^ ((crc_result & 0x10)>>4)) > 0 ) // IF(XOR(C6;BITAND(D5;2^4)/2^4)
        { crc_result = (byte) (crc5_poly ^ ((crc_result<<1) & 0x1f)); } // BITXOR($D$1;BITAND((D5*2);31))
        else
        { crc_result = (byte)((crc_result<<1) & 0x1f); } // shift left, keep only lower 6 bits
    }

    // 3 extra bits set to zero
    byte BYTE3=0x00;
    for (int i=0; i<3; i++)
    {
        if( ((( BYTE3>>(7-i) ) &0x01) ^ ((crc_result & 0x10)>>4)) > 0 ) // IF(XOR(C6;BITAND(D5;2^4)/2^4)
        { crc_result = (byte) (crc5_poly ^ ((crc_result<<1) & 0x1f)); } // BITXOR($D$1;BITAND((D5*2);31))
        else
        { crc_result = (byte)((crc_result<<1) & 0x1f)); } // shift left, keep only lower 6 bits
    }
    return crc_result;

}
byte crc5_decode(byte BYTE1, byte BYTE2)
{
    byte crc5_start = 0x1f;
    byte crc5_poly = 0x15;
    byte crc_result = crc5_start;

    // BYTE1
    for (int i=2; i<8; i++)
    {
       if( ((( BYTE1>>(7-i) ) &0x01) ^ ((crc_result & 0x10)>>4)) > 0 ) // IF(XOR(C6;BITAND(D5;2^4)/2^4)
       { crc_result = (byte) (crc5_poly ^ ((crc_result<<1) & 0x1f)); } // BITXOR($D$1;BITAND((D5*2);31))
       else
       { crc_result = (byte)((crc_result<<1) & 0x1f); } // shift left, keep only lower 6 bits
    }

    // BYTE2
    for (int i=0; i<8; i++)
    {
       if( ((( BYTE2>>(7-i) ) &0x01) ^ ((crc_result & 0x10)>>4)) > 0 ) // IF(XOR(C6;BITAND(D5;2^4)/2^4)
       { crc_result = (byte) (crc5_poly ^ ((crc_result<<1) & 0x1f)); } // BITXOR($D$1;BITAND((D5*2);31))
       else
       { crc_result = (byte)((crc_result<<1) & 0x1f); } // shift left, keep only lower 6 bits
    }

    // 3 extra bits set to zero
    byte BYTE3=0x00;
    for (int i=0; i<3; i++)
    {
        if( ((( BYTE3>>(7-i) ) &0x01) ^ ((crc_result & 0x10)>>4)) > 0 ) // IF(XOR(C6;BITAND(D5;2^4)/2^4)
        { crc_result = (byte) (crc5_poly ^ ((crc_result<<1) & 0x1f)); } // BITXOR($D$1;BITAND((D5*2);31))
        else
        { crc_result = (byte)((crc_result<<1) & 0x1f); } // shift left, keep only lower 6 bits
    }
    return crc_result;

}
```

Conclusion

This application note has shown how to code CRC algorithms on a microcontroller communicating with a MAX14915 octal industrial high-side switch. This code was tested using the MAX14915EVKIT 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.

More Information
For Technical Support: https://www.maximintegrated.com/en/support
For Samples: https://www.maximintegrated.com/en/samples
Other Questions and Comments: https://www.maximintegrated.com/en/contact

---

Application Note 6633: https://www.maximintegrated.com/en/an6633
APPLICATION NOTE 6633, AN6633, AN 6633, APP6633, Appnote6633, Appnote 6633