Application Note for I²C Flow and Differential Pressure Sensors

CRC Checksum

Summary

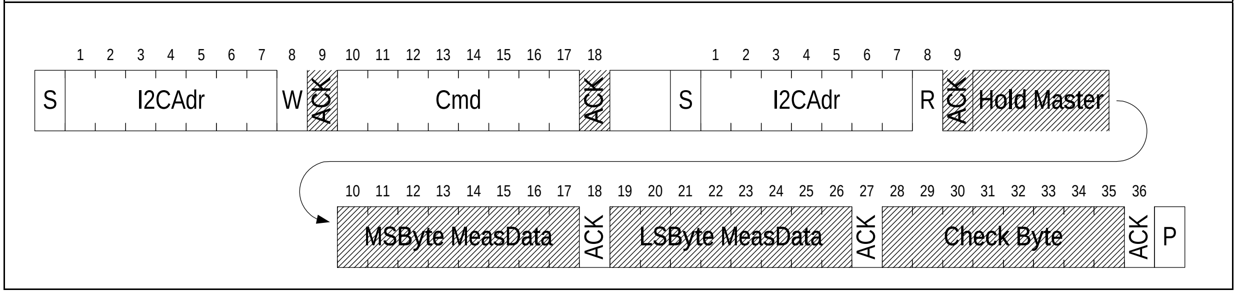
Inside Sensirion flow and differential pressure sensors with I²C interface, the SF04 CMOSens® sensor is used. This sensor features a switcheable CRC calculation which is enabled by default. This document explains how to do the CRC calculation.

After every 16bit data transmission, the CRC checksum is sent if the master sends an acknowledge (ACK) after each data byte and continues to clock the SCL line. The CRC checksum is calculated over both data bytes. If a CRC mismatch is detected, the sensor should be reset (command 0xFE) and the measurement should be repeated.

1. Trigger Measurement

The measurement cycle of I²C flow and differential pressure sensors is illustrated by the following figure. Note that there is no CRC check byte needed for the communication from the master to the sensor. A CRC check byte is only sent by the sensor if both data bytes are acknowledged by the master.

I²C Measurement



2. Theory

CRC stands for Cyclic Redundancy Check. It is one of the most effective error detection schemes and requires a minimal amount of hardware.

For in-depth information on CRC we recommend the comprehensive: "A painless guide to CRC error detection algorithms" available at: http://www.repairfaq.org/filipg/LINK/F_crc_v3.html. An online CRC calculator can be found under http://www.zorc.breitbandkatze.de/crc.html (Hint: For hexadecimal data sequences, a % has to be written in front of every byte.)

The polynomial used in the SF04 is: $x^8 + x^5 + x^4 + 1$. The types of errors that are detectable with this polynomial are:

1. Any odd number of errors anywhere within the transmission.

2. All double-bit errors anywhere within the transmission.

3. Any cluster of errors that can be contained within an 8-bit "window" (1-8 bits incorrect).

4. Most larger clusters of errors.

The CRC register initializes with the value “00000000”. It covers a full data transmission (2 bytes) without the acknowledge bits. In other words, the CRC byte is calculated from the “MSByte MeasData” and the “LSByte MeasData” as marked in the above figure.

The receiver can perform the CRC calculation upon the data transmission and then compare the result with the received CRC-8 byte. If a CRC mismatch is detected, the sensor should be reset (command hFE) and the measurement should be repeated.

The sensor system implements the CRC-8 standard with the following parameters:

Name	*Width*	*Polynom*	*Init*	*RefIn*	*RefOut*	*XorOut*	*Check*[1)]
CRC-8	8	0x31 ($x^8+x^5+x^4+1$)	0x00	false	false	0x00	0xA2

1) CRC for an ASCII input string "123456789" (i.e. 0x3132, 0x3334, ...)

In the following chapters, the application note explains how to calculate the CRC sum bitwise.

3. Bitwise CRC-8 calculation

3.1 XOR division

The CRC computation resembles a long division operation in which the quotient is discarded and the remainder becomes the result, with the important distinction that the arithmetic used is the carry-less arithmetic of a GF(2) finite field (binary). The following example shows how to calculate the CRC byte based on XOR division.

Example

1st Byte for calculation	→ 1000 0111 (0x87)
2nd Byte for calculation	→ 0000 0001 (0x01)
Polynom ($x^8+ x^5+ x^4+ 1$)	→ 1 0011 0001 (0x131)

```
1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 : 1 0 0 1 1 0 0 0 1
1 0 0 1 1 0 0 0 1
0 0 0 1 1 1 1 1 1 0 0 0
      1 0 0 1 1 0 0 0 1
      0 1 1 0 0 1 0 0 1 0
        1 0 0 1 1 0 0 0 1
        0 1 0 1 0 0 0 1 1 0
          1 0 0 1 1 0 0 0 1
          0 0 1 1 1 0 1 1 1 0 1
              1 0 0 1 1 0 0 0 1
              0 1 1 1 0 1 1 0 0 0
                1 0 0 1 1 0 0 0 1
                0 1 1 1 0 1 0 0 1 0
                  1 0 0 1 1 0 0 0 1
                  0 1 1 1 0 0 0 1 1 0
                    1 0 0 1 1 0 0 0 1
                    0 1 1 1 1 0 1 1 1 0
                      1 0 0 1 1 0 0 0 1
                      0 1 1 0 1 1 1 1 1 0
                        1 0 0 1 1 0 0 0 1
                        0 1 0 0 0 1 1 1 1 0
                          1 0 0 1 1 0 0 0 1
                          0 0 0 1 0 1 1 1 1 0 0
```

3.2 Generator model

With the generator model, the receiver copies the structure of the CRC generator in hard- or software.

An algorithm to calculate could look like this:

1) Initialise CRC Register to 0x00 (initial value)
2) Compare each (transmitted and received) bit with bit 7
3) If the same: shift CRC register, bit0='0'
   else: shift CRC register and then invert bit4 and bit5, bit0='1' (see Figure 1)
4) receive new bit and go to 2)

Example:

1st Byte for calculation	→ 1000 0111 (0x87)
2nd Byte for calculation	→ 0000 0001 (0x01)

Input bit's	bit7 … bit0	0x	Comment
	0000 0000	00	Start value
1	0011 0001	31	1st byte (MSB) of measurement
0	0110 0010	62	
0	1100 0100	C4	…
0	1011 1001	B9	
0	0100 0011	43	
1	1011 0111	B7	
1	0110 1110	6E	
1	1110 1101	ED	
0	1110 1011	EB	2nd data byte (MSB)
0	1110 0111	E7	
0	1111 1111	FF	
0	1100 1111	CF	
0	1010 1111	AF	
0	0110 1111	6F	
0	1101 1110	DE	
1	1011 1100	BC	Final CRC value

4. C++ code

For easy implementation to a microcontroller, the following simple C++ routine can be used. Note that this code is not optimized for speed.

```
//CRC
#define POLYNOMIAL 0x131     //P(x)=x^8+x^5+x^4+1 = 100110001

//============================================================
u8t SF04_CheckCrc (u8t data[], u8t nbrOfBytes, u8t checksum)
//============================================================
//calculates checksum for n bytes of data
//and compares it with expected checksum
//input:    data[]        checksum is built based on this data
//          nbrOfBytes    checksum is built for n bytes of data
//          checksum      expected checksum
//return:   error:        CHECKSUM_ERROR = checksum does not match
//                        0 = checksum matches
//============================================================

{
u8t crc = 0;
u8t byteCtr;
//calculates 8-Bit checksum with given polynomial
for (byteCtr = 0; byteCtr < nbrOfBytes; ++byteCtr)
{ crc ^= (data[byteCtr]);
  for (u8t bit = 8; bit > 0; --bit)
  { if (crc & 0x80) crc = (crc << 1) ^ POLYNOMIAL;
    else crc = (crc << 1);
  }
}
if (crc != checksum) return CHECKSUM_ERROR;
else return 0;
}
```

With the type definitions:

```
typedef enum{
CHECKSUM_ERROR = 0X04
}etError;

typedef unsigned char   u8t;
```

5. Revision history

Date	Version	Changes
Juni 2009	v1.0	Initial release

Headquarter and Sales Offices

SENSIRION AG Laubisruetistr. 50 CH-8712 Staefa ZH Switzerland	Phone: + 41 (0)44 306 40 00 Fax: + 41 (0)44 306 40 30 info@sensirion.com www.sensirion.com	SENSIRION Korea Co. Ltd. #1414, Anyang Construction Tower B/D, 1112-1, Bisan-dong, Anyang-city, Gyeonggi-Province, South Korea	Phone: +82-31-440-9925~27 Fax: +82-31-440-9927 info@sensirion.co.kr www.sensirion.co.kr
SENSIRION Inc Westlake Pl. Ctr. I, suite 204 2801 Townsgate Road Westlake Village, CA 91361 USA	Phone: +1 805-409 4900 Fax: +1 805-435 0467 michael.karst@sensirion.com www.sensirion.com	SENSIRION China Co. Ltd. Room 2411, Main Tower Jin Zhong Huan Business Building, Postal Code 518048 Futian District, Shenzhen, PR China	Phone: +86 755 8252 1501 Fax: +86 755 8252 1580 info@sensirion.com.cn/ www.sensirion.com.cn
SENSIRION Japan Sensirion Japan Co. Ltd. Shinagawa Station Bldg. 7F 4-23-5 Takanawa Minato-ku, Tokyo, Japan	phone: +81 3-3444-4940 fax: +81 3-3444-4939 info@sensirion.co.jp www.sensirion.co.jp		

Find your local representative at: http://www.sensirion.com/reps