Sensirion Interface Board I2C specs.

Table of Contents

Overview: 1

Connection Diagram: 2

Sensor Specs: 3

Interface Specs: 4

Data Transfer Format: 5

Conversion to Physical Values: 6

Hookup for Sensirion Program: 7

Sensirion Software: 7

IO Warrior UST to i2C: Downloads 7

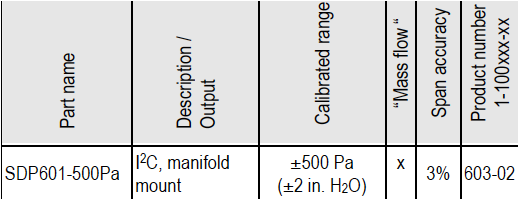
Fit Data to Fifth Order Polynomial: 7

Outputs: 10

# Overview:

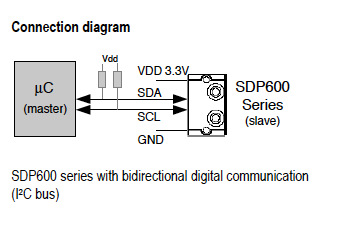
I want to make a board that reads the Sensirion Raw Ticks and then drives a 0-5 and 4-20 mA output to be a fifth order “fit” to the measured value as it relates to the RAW ticks.

The RAW TICKS go from 0-65,535 with 32,767 being 0 and 500Pa being 65,535 and 0 being – 500 Pa.

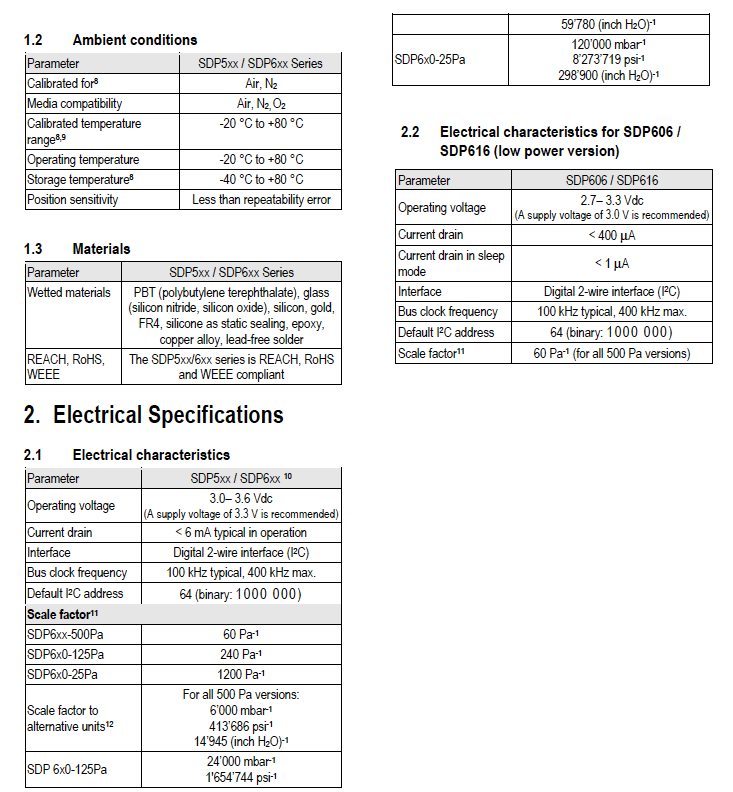


I will use 32,767 to 65,535 in my system…

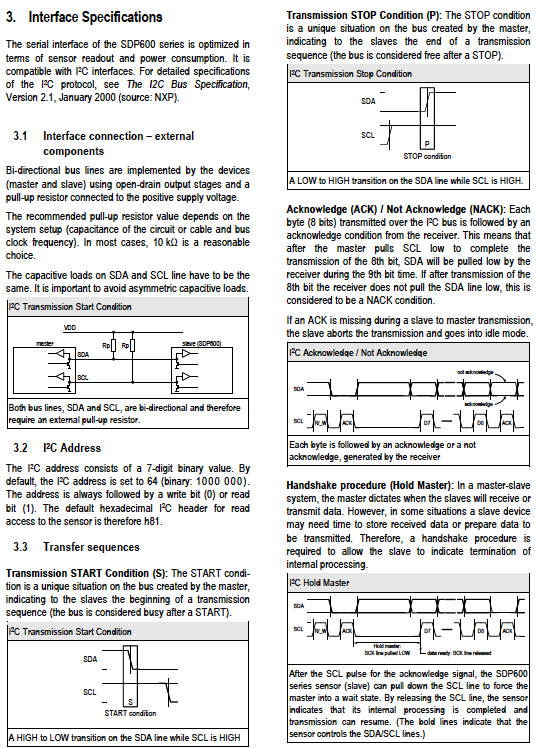
## Connection Diagram:



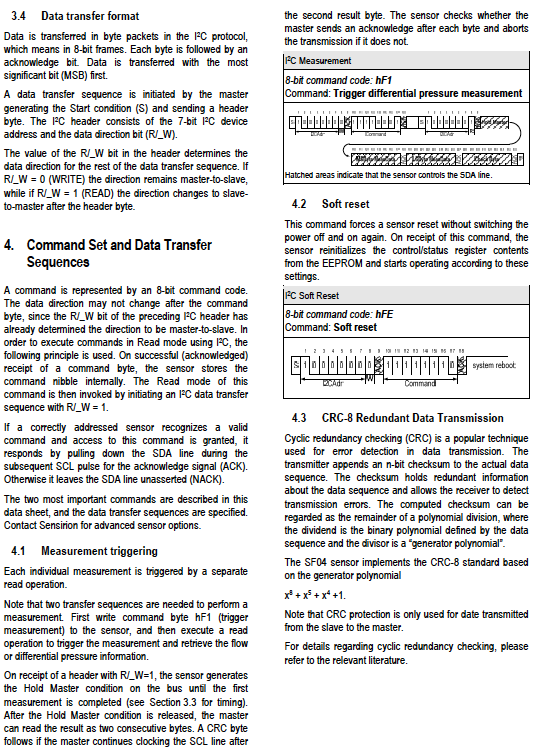
## Sensor Specs:



## Interface Specs:



## Data Transfer Format:

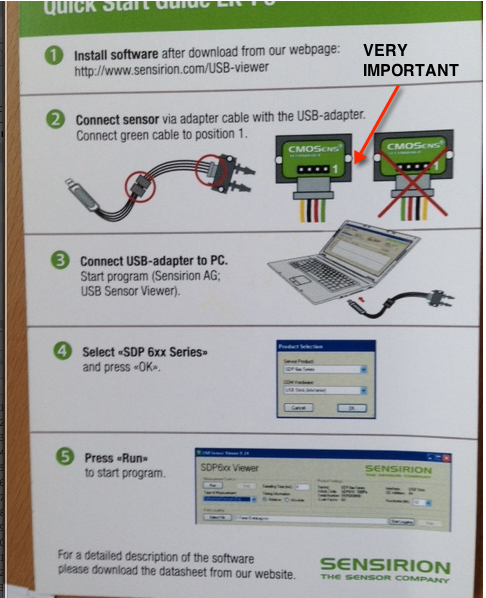


## Conversion to Physical Values:

## 

## Hookup for Sensirion Program:

Note location of GREEN WIRE to Pin 1!



## Sensirion Software:

Link to [Sensirion Software for Windows](http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Sample_Codes_Software/Sensirion_USB_RS485_Sensor_Viewer_V2.12.msi)

A simple executable that works GREAT! But it needs the IOWarrior I2C to USB.. which is pretty nice but expensive….

## IO Warrior UST to i2C: [Downloads](http://www.codemercs.com/165/?L=1)

## Fit Data to Fifth Order Polynomial:

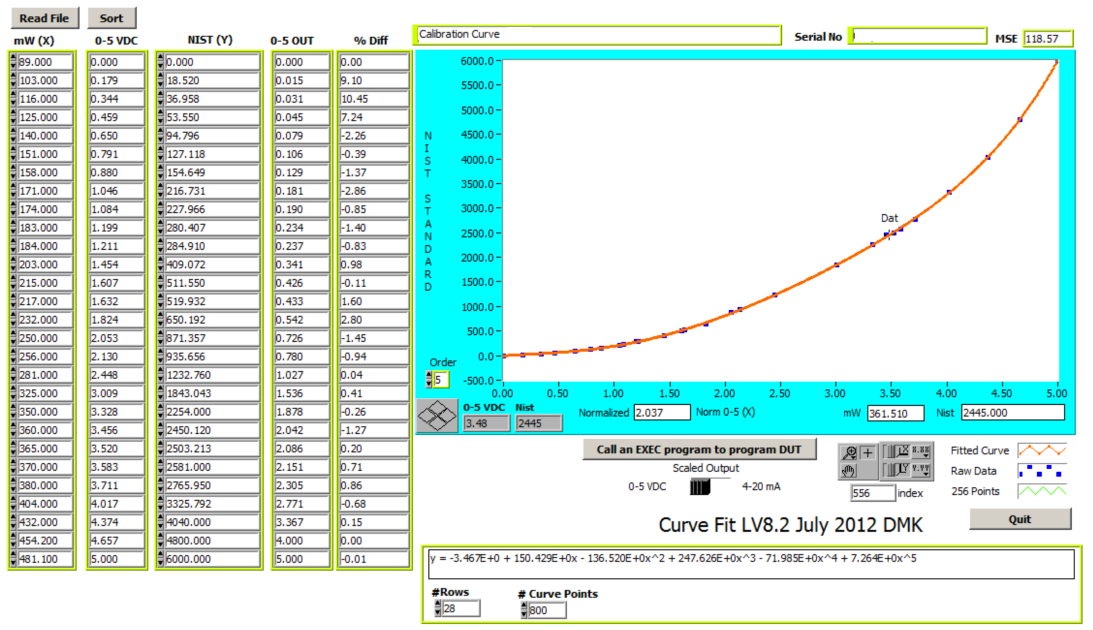
I will take data like this, in the worst case..

Most likely it will fit to a cubic, third order fit, but I made this fifth order example..

y = -3.467E+0 + 150.429E+0x - 136.520E+0x^2 + 247.626E+0x^3 - 71.985E+0x^4 + 7.264E+0x^5

X in this case is some low mW number but WE will be using nominal values from 37,767 to 65,535 for the “RAW” values because I do not want to invoke any “correction factors” from the device.

The RAW values are GIVEN to the equation and the OUTPUT is NORMALIZED for the easiest to scale Full Scale. For example, if 0-32767 raw clicks represent 0 – 65.23 “SLPM” and we want to have – 0 5 VDC and 4 – 20 mA VDC be equal to 0 - 50 SLPM then we would SPAN the device to have the Full Scale at 50 SLPM but our input calibration happens to be for 0 – 65.23 SLPM using the full 32,767 available.



**Drive OutputDAC:**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void D2AOutputController()

{

if (gnSerialPinged != 0) return;

// 4 - 20ma - Current

{ // Calculate output

// 12 bit A/D -> 10 bit PWM

// Ten bit pulse width modulator {0-0x3FF)

// @10Mhz period of main loop is about 5ms. Set PWM to

// match that as closely as possible at the highest

// resolution possible (10 bit resolution max).

//REIT1 Doug: 0 - 1000 = 0 - 20 so 200 = 4 and 1000 = 20 Cool??

//REIT: this leaves the overrange and the under range for 4 - 20 but

//nuttin for 0 - 5!!

// 0-1023 = 0-20 ma

// 205 = 4 ma

//

// gnSpanDelta = (gnConfigD2ASpanCurrent + gnConfigD2AZeroCurrent);

//REIT1: Daves 4-20 idea for PWM...

//dabuffer=gnD2APWMFlowRate/gnConfigFlowUserFullScale \* gnConfigD2ASpanCurrent + gnConfigD2AZeroCurrent; Holey shit!!

//gnD2APWM4\_20maOutputValue = dabuffer;

dabuffer = (((unsigned long)gnD2APWMFlowRate \* gnConfigD2ASpanCurrent)/1023L);

gnD2APWM4\_20maOutputValue = dabuffer + gnConfigD2AZeroCurrent;

// 4ma offset (5.5v supply)

gnD2APWM4\_20maOutputValue += 120; //test

//

if (gnD2APWM4\_20maOutputValue > 0x3FF) {gnD2APWM4\_20maOutputValue = 0x3FF;}

// 10bit output

CCP2X = (gnD2APWM4\_20maOutputValue &0x02) ?1 :0;

CCP2Y = (gnD2APWM4\_20maOutputValue &0x01) ?1 :0;

CCPR2L = (UINT8)(gnD2APWM4\_20maOutputValue >>2);

}

// 0 - 5 volt - Voltage

{ //

// Ten bit pulse width modulator {0-0x3FF)

//

//

// Calculate output

//REIT1: Daves 0-5 idea for PWM...

//dabuffer=gnD2APWMFlowRate/gnConfigFlowUserFullScale \* gnConfigD2ASpanVoltage + gnConfigD2AZeroVoltage; Holey shit!!

//gnD2APWM5VoltOutputValue = dabuffer;

// gotta handle exceptions...

dabuffer = (((unsigned long)gnD2APWMFlowRate \* gnConfigD2ASpanVoltage)/1023L);

gnD2APWM5VoltOutputValue = dabuffer + gnConfigD2AZeroVoltage;

// Handle underflow (0<=n<=1023)

// issue: avoid overflow displayed as large positive number

if (gnD2APWM5VoltOutputValue > 0x3FF) {gnD2APWM5VoltOutputValue = 0x3FF;} //

// if (gnD2APWM5VoltOutputValue < 0x00F) { gnD2APWM5VoltOutputValue = 0x00F; } //was 0x3ff

// 12 bit A2D Result -> 10bit output

CCP1X = (gnD2APWM5VoltOutputValue &0x02) ?1 :0; // set up the two lsb's

CCP1Y = (gnD2APWM5VoltOutputValue &0x01) ?1 :0;

CCPR1L = (UINT8)(gnD2APWM5VoltOutputValue >>2);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Outputs:

0-5 VDC for 0 – SCALED Full Scale flow adjusted with a curve as above

4-20 mA DC for 0 – SCALED Full Scale flow adjusted with a curve as above

Digital: Modbus® compliant RS485 RTU or RS-232 to begin with