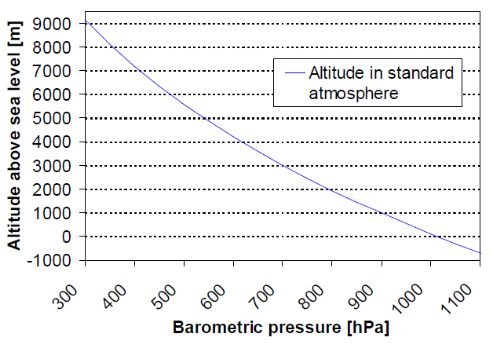
**BMP085 Digital Pressure Sensor**



The BM085 is a digital pressure sensor manufactured by Bosch Sensortec. These pressure sensors are also known as barometers. These modules can read a pressure between 30,000 and 110,000 Pascals (Pa) with a very high calibrated accuracy. The BMP085 also provides the additional function of reading temperatures between ranges of 0 and 65 degrees. This module provides an I2C communication interface through which a microcontroller can connect and collect values. The I2C protocol ensure the data is less susceptible to noise and other causes the may interfere with data being transferred. The BM085 has been designed to be an ultra-low power device with very high accuracy.

The pin provides 6 pins as shown below in the table. The VCC and GND are the live and ground pins. The SDA is the data line and the SCL is the clock pin used the I2C protocol. The XCLR is the reset pin. When pulled to ground the chip will reset back to it default state. The EOC, an abbreviation for End of Conversation, is used to indicate whenever a pressure or temperature translation has completed. A signal is generated to indicate this completion. It was decided that the XCLR and EOC pins will be ignored to conserver pin and power usage of the ATMega328-PU in this project.

These barometric pressure units can be used for many different applications other than just finding pressure and temperature. These units can be used in weather forecast projects or in altitude projects. By knowing the atmospheric pressure, theoretically it is possible to tell the weather conditions. Low pressure usually indicates overcast, rainy or snowy weather. High pressure usually indicates clear skies with a lot of sun. This barometer can also be used in “altimetry”. Pressure has a certain relationship with respect to altitude, thus pressure values can be used to calculate the altitude of the robot. A sea level air pressure is measured at an average of 101325 Pascal (Standard Pressure) while at the height of Mount Everest (8,848 m) the pressure is measured at 33700 Pascal. Below shows a graph that demonstrates the relationship of altitude against pressure. This graph is found in the datasheet. The pressure is in hector Pascal [hPa]. One hector Pascal is equivalent to 100 Pascal.



This pressure module would be perfect to describe what type of weather environment surrounds it and what altitude it stands at.

**Specifications**

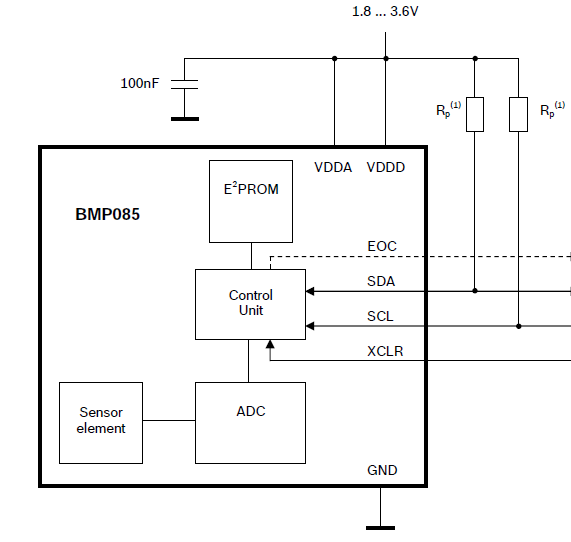
|  |  |
| --- | --- |
| Parameters |  |
| Operating Temperature | -40 to 85 Degrees |
| Supply Voltage | 1.8 to 3.6 Volts |
| Average Supply Current | 5 µA for standard mode |
| Pressure Accuracy | ±100.0 Pascal |
| Temperature Accuracy | ±1.0 Degrees |

**Pin Layout**

|  |  |
| --- | --- |
| Pins |  |
| VCC | 1.8 to 3.6 Volts |
| GND | Ground |
| EOC | End of Conversation |
| XCLR | Reset Pin |
| SCL | I2C Clock |
| SDA | I2C Data |

**Architecture**

The BMP085 is designed using an ultra-sensitive peizo-resistive sensor, an analogue to digital converter, control unit with E2PROM (EEPROM) and Serial I2C interface. The figure below shows the general architecture of the BMP085 as per datasheet.



Notice the additional pull-up resistors. These are not internal pull-resistor but are merely used to indicate the need for them to be connected when using the I2C protocol. The I2C protocol is an active low protocol meaning a digital ‘1’ is represented as 0 Volts. The pull-up resistors are used to keep the connection line high in an idle state. This is an important to know when connecting various microcontrollers. The ATMega328-Pu provides internal pull-up resistors of 20 kΩ within the I2C communication interface. The pull-up resistors can simply be ignored for connecting the BMP085 to the ATmega328-PU microcontroller.

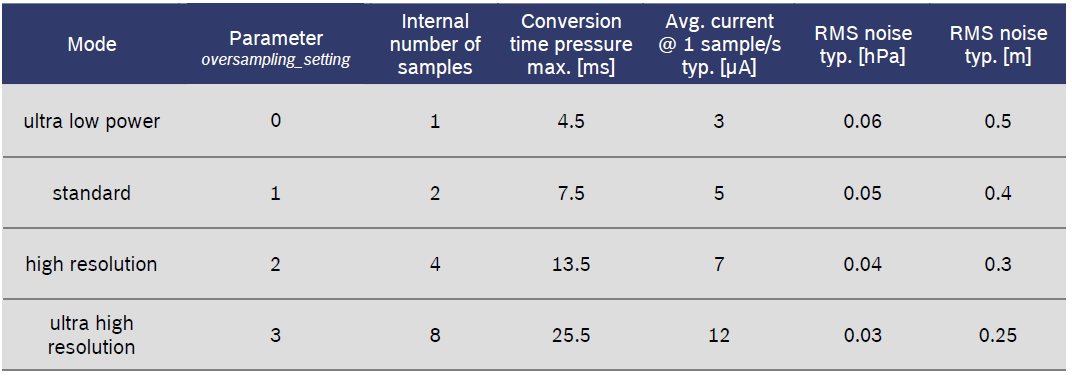
The E2PROM stores 176 bits of calibration data. This used to compensate offset, temperature dependence and other parameter values of the sensor.

* **UP** – pressure data (16-19 bits)
* **UT** – temperature data (16 bits)

**Measuring Pressure and Temperature**

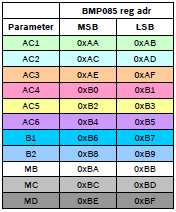
Measuring Modes

Below is a table that shows the different configuration modes the BMP085 sensor can use. Under the parameter heading shows the values needed to configure the module. The Ultra-low power is used for as the name describes. The Standard is normal usage. The High Resolution and Ultra-Resolution are used when high accuracy values of temperatures and pressure are required. Note the oversampling\_setting which is used in the code to set the mode.



Calibration Coefficients

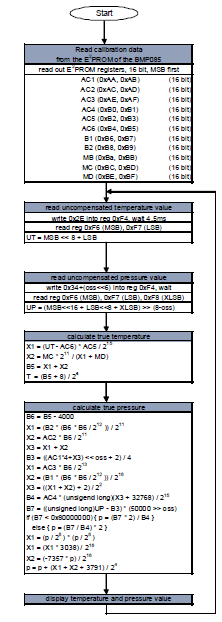
Within the 176 bit E2POM there are 11 calibration coefficients each of 16 bits long. These values are taken via I2C and used to calculate temperature and pressure. The authenticity of these coefficients is correct when there are no values of 0x0000 and 0xFFFF in them.



Calculating Temperature and Pressure

The BMP085 module contains an I2C interface that can be used to communicate with the sensor. The Arduino IDE and the Wire library are used to communicate with this sensor module using the ATMega328-PU chip. The library is used to set up I2C communication between two devices. Data values can be transmitted and received using this communication standard on the ATMega328-PU microcontroller. Below shows a flow chart that describes the procedure that must be done in order to obtain temperature and pressure values.

1

**1. Start**

First the address 0x77 is sent to the I2C interface to establish a connection with the BMP085

2

**2. Read calibration data from the E2PROM of the BMP085**

The coefficients are read from the E2POM. Each of these coefficients is 16 bits long. Within the Arduino IDE each of these values are stored in an unsigned int which is 16 bits long. These coefficients are used later with the uncompensated pressure and temperature value to find the true temperature and pressure.

3

4

**3. Read the uncompensated temperature value**

The value 0x2E is written to the register 0xF4. Once this has been done a 4.5 ms delay is taken for an uncompensated temperature value to be measured. The registers 0xF6 and 0xF7 are read to obtain the uncompensated temperature. The 0xF6 register produces the MSB and 0xF7 produces the LSB. Thus the uncompensated temperature is 2 bytes long. The MSB and LSB values are stored in a 16 bit unsigned integer UT:

6

5

UT = MSB << 8 + LSB

The MSB (16 bits) is shifted 8 bits to the left and the LSB (8 or 16 bits) is added on. This final value is stored in UT (16 bits).

**4. Read the uncompensated pressure value**

The value 0x2E+(oss<<6) is written into register 0xF4. A 4.5 ms delay is taken again. The registers 0xF6 and 0xF7 and 0xF8 are read. Each produces a byte of data. MSB from 0xF6, LSB from 0xF7 and XLSB from 0xF8. The uncompensated pressure is stored in a 32 bit unsigned integer UP

UP = (MSB << 16 + LSB<<8 + XLSB) >> (8-oss)

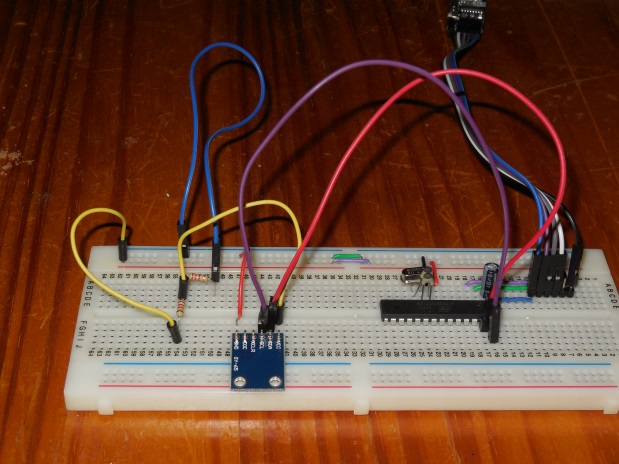
The MSB (16 bits) is shifted 16 bits to the left, the LSB (16 bits) is shifted left 8 bits to the left and XLSB (8 bits) is added. MSB and LSB are added and shifted 8-oss bits to the right. The oss is the over sampling setting which can be found in the Mode table above #########. There are 4 modes and can be selected by setting oss between the values 0-3. Shifting right by 8-oss bits sets the mode.

**5. Calculate true pressure**

These are all calculations that need to be done in order to obtain the true temperature and pressure. All the equations are given as per data sheet. All the variables are 32 bits long and they are all signed variables except for B4 and B7.

**Testing the BMP085 Sensor**

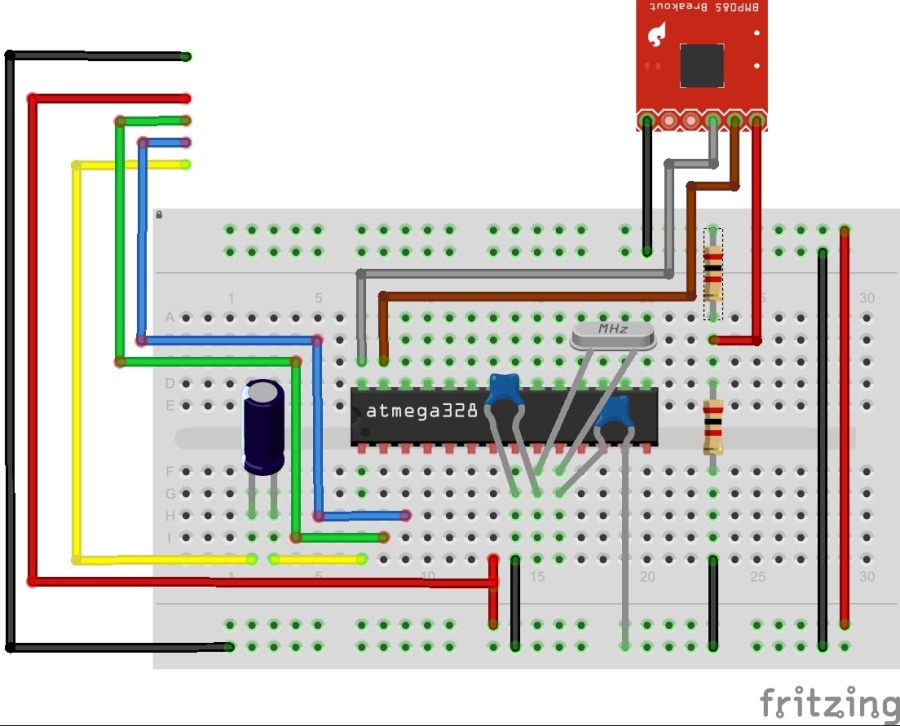
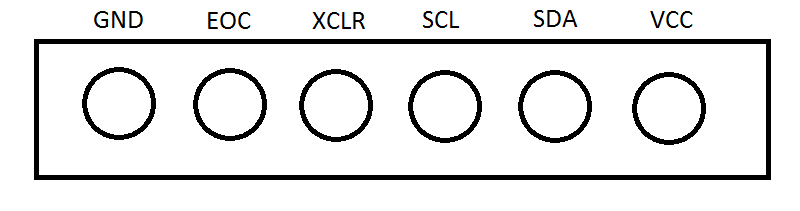
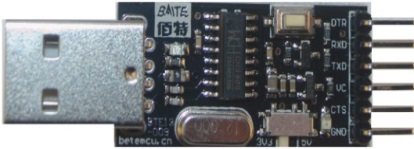
For this part of the project, testing was done on the BMP085 senor module using the ATMega328-PU microcontroller and the Arduino IDE. The BMP085 pressure and temperature sensor is very easy to setup. There are only four pins that need to be connected of the sensor module. These pins include the VCC, GND, SDA and the SCL pin. The VCC should connect to a supply voltage within the range of 1.8-3.6 Volts. The microcontroller produces a supply voltage of 5 Volts. It is necessary that a voltage divider circuit be implemented to reduce the voltage input into the BMP085 sensor. GND connect to ground, SDA (Data pin) connects to the corresponding SDA on the microcontroller and the SCL (clock pin) connects to the corresponding SCL of the microcontroller. Shown below is an image caption of the test circuit.



**Components**

* Two 2 kΩ Resistors
* Jumper Wires
* Breadboard
* BMP085 Temperature and Pressure Sensor
* ATMega328-PU microcontroller with serial communication to the computer discussed earlier in this report.

**Schematic**



The SCL pin of the sensor is connected to analog input 5 (pin 26), the SDA sensor pin is connected to analog input 4 and ground is connect to ground. Note the VCC of the sensor module is not connected directly to the supply voltage but rather to a voltage divider circuit. The sensor module has a maximum voltage input of 3.6 Volts and so hooking up the VCC directly to the supply voltage might permentally damage the sensor. The voltage divider consists of two 2 kΩ resistors which halves the voltage input into the sensor. The sensor will then get a supply voltage of 2.5 Volts, which is within operation voltage (1.8-3.6 Volts).

**Code**

The code shown below is found at Sparkfun.com and is used to operate this sensor module using the Arduino IDE. This code was referenced and modified from the original C code files provided by the BMP085 manufacturer. The code will be heavily commented to explain each step that takes place

Below are two functions that are used to fetch data. The first function is used to fetch one byte of data while the second is used to fetch two bytes of data.

// Put this line at the top of your program

#define BMP085\_ADDRESS 0x77 // I2C address of BMP085

// Read 1 byte from the BMP085 at 'address'

char bmp085Read**(**unsigned char address**)**

**{**

unsigned char data**;**

//Begin transmission to slave device with BMP085\_ADDRESS. The slave device is the sensor

//and has an address of 0x77.

Wire**.**beginTransmission**(**BMP085\_ADDRESS**);**

//Queue bytes for transmission. The byte in the queue is the address

//passed.

Wire**.**write**(**address**);**

//transmit the bytes in the queue. In this case "address" is sent.

Wire**.**endTransmission**();**

Wire**.**requestFrom**(**BMP085\_ADDRESS**,** 1**);** //Request one byte

**while(!**Wire**.**available**());** //Wait until all the data is received.

**return** Wire**.**read**();** //Read a byte of data return from slave and return this value.

**}**

// Read 2 bytes from the BMP085

// First byte will be from 'address'

// Second byte will be from 'address'+1

int Read2Bytes**(**unsigned char address**)**

**{**

unsigned char msb**,** lsb**;**

Wire**.**beginTransmission**(**BMP085\_ADDRESS**);**

Wire**.**write**(**address**);**

Wire**.**endTransmission**();**

Wire**.**requestFrom**(**BMP085\_ADDRESS**,** 2**);** //Request two bytes.

**while(**Wire**.**available**()<**2**)** //Wait until all two bytes are received.

**;**

msb **=** Wire**.**read**();** //First byte that is received.

lsb **=** Wire**.**read**();** //Second byte that is received

//Shift the msb 8 bits to the left and OR lsb to make them both a 16 bit word.

**return** **(**int**)** msb**<<**8 **|** lsb**;**

**}**

The function below is used to fetch all the calibration values within the E2PROM. Some variable have been defined which will be used to store the true and final values of pressure, temperature and altitude. This code corresponds to part 2 of the flow diagram:

#####################################################################

#include <Wire.h>

const unsigned char OSS **=** 0**;** // Oversampling Setting

// Calibration values that need to be fetched.

int ac1**;**

int ac2**;**

int ac3**;**

unsigned int ac4**;**

unsigned int ac5**;**

unsigned int ac6**;**

int b1**;**

int b2**;**

int mb**;**

int mc**;**

int md**;**

//////////////////////////////////////////////////

long b5**;**

short temperature**;** // Store the value of temperature.

long pressure**;** // Store the value of pressure.

// Use these for altitude conversions.

const float p0 **=** 101325**;** // Pressure at sea level (Pa)

float altitude; // Altitude variable.

void setup**()**

**{**

Serial**.**begin**(**9600**);**

Wire**.**begin**();**

bmp085Calibration**();**

**}**

// Stores all of the bmp085's calibration values into global variables

// Calibration values are required to calculate temp and pressure

// This function should be called at the beginning of the program

void bmp085Calibration**()**

**{**

ac1 **=** bmp085ReadInt**(**0xAA**);**

ac2 **=** bmp085ReadInt**(**0xAC**);**

ac3 **=** bmp085ReadInt**(**0xAE**);**

ac4 **=** bmp085ReadInt**(**0xB0**);**

ac5 **=** bmp085ReadInt**(**0xB2**);**

ac6 **=** bmp085ReadInt**(**0xB4**);**

b1 **=** bmp085ReadInt**(**0xB6**);**

b2 **=** bmp085ReadInt**(**0xB8**);**

mb **=** bmp085ReadInt**(**0xBA**);**

mc **=** bmp085ReadInt**(**0xBC**);**

md **=** bmp085ReadInt**(**0xBE**);**

**}**

The function bmp085ReadUT is used to fetch the 16 bit uncompensated temperature value, ut. This value is used later on in calculations to find the true value of temperature. The code corresponds to part 3 of the flow diagram:

##################################################################

// Read the uncompensated temperature value

unsigned int bmp085ReadUT**()**

**{**

unsigned int ut**;**

// Write 0x2E into Register 0xF4

// This requests a temperature reading

Wire**.**beginTransmission**(**BMP085\_ADDRESS**);**

Wire**.**write**(**0xF4**);** // Select register 0xF4

Wire**.**write**(**0x2E**);** // Write value 0x2E into register 0xF4

Wire**.**endTransmission**();**

// Wait at least 4.5ms

delay**(**5**);**

// Read two bytes from registers 0xF6 and 0xF7

ut **=** bmp085ReadInt**(**0xF6**);**

**return** ut**;**

**}**

This function finds the uncompensated pressure value, up. This is also later on used together with up in calculations to find the true value of temperature and pressure. The function falls under pared 4 of the flow diagram:

###################################################################

// Read the uncompensated pressure value

unsigned long bmp085ReadUP**()**

**{**

unsigned char msb**,** lsb**,** xlsb**;**

unsigned long up **=** 0**;**

// Write 0x34+(OSS<<6) into register 0xF4

// Request a pressure reading w/ oversampling setting

Wire**.**beginTransmission**(**BMP085\_ADDRESS**);**

Wire**.**write**(**0xF4**);**

Wire**.**write**(**0x34 **+** **(**OSS**<<**6**));**

Wire**.**endTransmission**();**

// Wait for conversion, delay time dependent on OSS

delay**(**2 **+** **(**3**<<**OSS**));**

// Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)

Wire**.**beginTransmission**(**BMP085\_ADDRESS**);**

Wire**.**write**(**0xF6**);**

Wire**.**endTransmission**();**

Wire**.**requestFrom**(**BMP085\_ADDRESS**,** 3**);**

// Wait for data to become available

**while(**Wire**.**available**()** **<** 3**)**

**;**

msb **=** Wire**.**read**();** // Read first byte and store in msb.

lsb **=** Wire**.**read**();** // Read second byte and store in lsb.

xlsb **=** Wire**.**read**();** // Read third byte and store in xlsb.

// This formula UP = (MSB<<16+LSB<<8+XLSB)>>(8-oss) is described in part

// 4 of the flow diagram. Note that in the addition operator is not used.

//The ‘|’ operator is used instead which OR operator. Either can be used

//as the same result will be produced.

up **=** **(((**unsigned long**)** msb **<<** 16**)** **|** **((**unsigned long**)** lsb **<<** 8**)** **|** **(**unsigned long**)** xlsb**)** **>>** **(**8**-**OSS**);**

**return** up**;**

**}**

This piece of code contains the two functions that calculate the true value of temperature and pressure. The formulas used in the functions are given by the datasheet for this sensor module. There is no real need to understand the technicalities of the formulas, but there is the need to understand the implementation is C. This part of the code corresponds to part 5 and 6 of the flow diagram:

############################################

// Calculate temperature given ut.

// Value returned will be in units of 0.1 deg C

short bmp085GetTemperature**(**unsigned int ut**)**

**{**

long x1**,** x2**;**

x1 **=** **(((**long**)**ut **-** **(**long**)**ac6**)\*(**long**)**ac5**)** **>>** 15**;**

x2 **=** **((**long**)**mc **<<** 11**)/(**x1 **+** md**);**

b5 **=** x1 **+** x2**;**

**return** **((**b5 **+** 8**)>>**4**);**

**}**

// Calculate pressure given up

// calibration values must be known

// b5 is also required so bmp085GetTemperature(...) must be called first.

// Value returned will be pressure in units of Pa.

long bmp085GetPressure**(**unsigned long up**)**

**{**

long x1**,** x2**,** x3**,** b3**,** b6**,** p**;**

unsigned long b4**,** b7**;**

b6 **=** b5 **-** 4000**;**

// Calculate B3

x1 **=** **(**b2 **\*** **(**b6 **\*** b6**)>>**12**)>>**11**;**

x2 **=** **(**ac2 **\*** b6**)>>**11**;**

x3 **=** x1 **+** x2**;**

b3 **=** **(((((**long**)**ac1**)\***4 **+** x3**)<<**OSS**)** **+** 2**)>>**2**;**

// Calculate B4

x1 **=** **(**ac3 **\*** b6**)>>**13**;**

x2 **=** **(**b1 **\*** **((**b6 **\*** b6**)>>**12**))>>**16**;**

x3 **=** **((**x1 **+** x2**)** **+** 2**)>>**2**;**

b4 **=** **(**ac4 **\*** **(**unsigned long**)(**x3 **+** 32768**))>>**15**;**

b7 **=** **((**unsigned long**)(**up **-** b3**)** **\*** **(**50000**>>**OSS**));**

**if** **(**b7 **<** 0x80000000**)**

p **=** **(**b7**<<**1**)/**b4**;**

**else**

p **=** **(**b7**/**b4**)<<**1**;**

x1 **=** **(**p**>>**8**)** **\*** **(**p**>>**8**);**

x1 **=** **(**x1 **\*** 3038**)>>**16**;**

x2 **=** **(-**7357 **\*** p**)>>**16**;**

p **+=** **(**x1 **+** x2 **+** 3791**)>>**4**;**

**return** p**;**

**}**

The function getAltitude is used to calculate the altitude of the robot. The formula is provided by the datasheet.

################################# Equation ##################################

float getAltitude**(**long pressure**)**

**{**

const float p0 **=** 101325**;** // Pressure at sea level (Pa)

float altitude**;**

// Add this into loop(), after you've calculated the pressure

altitude **=** **(**float**)**44330 **\*** **(**1 **-** pow**(((**float**)** pressure**/**p0**),** 0.190295**));**

**return** altitude**;**

**}**

Below is the main loop that will continuously print out the values of temperature, pressure and altitude.

void loop**()** // A loop that will run forever.

**{**

temperature **=** bmp085GetTemperature**(**bmp085ReadUT**());**

pressure **=** bmp085GetPressure**(**bmp085ReadUP**());**

altitude **=** getAltitude**(**pressure**);**

Serial**.**print**(**"Temperature: "**);**

Serial**.**print**(**temperature**,** DEC**);**

Serial**.**println**(**" \*0.1 deg C"**);**

Serial**.**print**(**"Pressure: "**);**

Serial**.**print**(**pressure**,** DEC**);**

Serial**.**println**(**" Pa"**);**

Serial**.**println**();**

Serial**.**print**(**"Altitude: "**);**

Serial**.**print**(**altitude**,** DEC**);**

Serial**.**println**(**" Meters"**);**

Serial**.**println**();**

delay**(**1000**);**

**}**

**Result**

#######################################################

**Reference**

Datasheet:

<http://www.adafruit.com/datasheets/BMP085_DataSheet_Rev.1.0_01July2008.pdf>

Sparkfun:

<https://www.sparkfun.com/tutorials/253>

Sensor Picture at the top:

<http://www.ladyada.net/images/sensors/BMP085_t.jpg>