/\*

Mini\_Altimeter

D.R.Patterson

7/12/2014

MPL3115A2 Barometric Pressure Sensor

Display Altitude and Temperature range

With Atmospheric Pressure Correction

Data acquired in sensor immediate capture mode

Data ready flag polled to test for new data

Acknowlegements:

http://www.instructables.com/id/The-Ultimate-Altimeter-A-compact-Arduino-altimeter/?ALLSTEPS

Nathan Seidle, SparkFun Electronics, September 24th, 2013

Originally used the MPL3115A2 and debounce libraries:

Data acquisition rewritten after some MPL3115A2 library errors

Debounce library removed and replaced

Sea level pressure calibration added using Base altitude

Temperature meaurement added- with correction factor

Base altitude configuration in SET mode

Hardware Connections (Breakoutboard to Arduino):

VCC = 3.3V

SDA = A4 (use inline 10k resistor if your board is 5V)

SCL = A5 (use inline 10k resistor if your board is 5V)

Gnd = Gnd

Pullups required on 5V mega

See defines for modes

After selecting the SET mode release the button then press and hold when "Pres" appears.

In the absence of the second button press Altitude sensing continues after a ~= 3 second delay.

The base altitude increments and is displayed as long as the button is held down.

Releasing the button sets the base altitude at the displayed height.

The sensor then re-calibrates sea level pressure, base altitude and base temperature.

The base altitiude range varies between -10 to 400m

\*/

#include <Wire.h>

#include "SevSeg.h"

#define ALT 0 // altitude (m)

#define AVER 1 // average altitude over 5S

#define TOP 2 // max altitude

#define BOT 3 // min altitude

#define DIFF 4 // Difference in altitude

#define TEMP 5 // Temp in centigrade

#define TDIF 6 // Temp range in centigrade

#define STBY 7 // stand by - sensor and bubble off

#define SET 8 // reset Base altitude

//Create an instance of the bubble object

SevSeg myDisplay;

// configure \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

byte brightness = 50; // Data display brighness as a percent. In calibration modes display uses 100%

boolean eco = true; // if true only displays when data is ready otherwise displays continuously

boolean debug = false; // Used for activating Serial Debugging

const int buttonPin = 3; // the number of the pushbutton pin 10 on mega

// kitchen table 51.25 m

// red room pc 55 m

// cadmac 47 m

float ALTBASIS = 51.25; // base altitude in m

// temperature measurement is affected by enclosure

float tempcorrection = -1.8; // mega sensor -1.0 - NB Do not expect fast response!

boolean maxoversample = true; // greater accuracy, less speed

// bit 2 in CTRL\_REG1 (26) allows over-sampling when set

// bit 4,5,6 contain the over sampling rate. 56 is max rate(512ms min), 48 next (258mS)

const byte averaged = 10; // set the number of readings to be averaged

// end configure \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

float average[averaged];

int waitTime;

byte avpos = 0;

char tempString[10]; //Used for sprintf

float maxaltitude = -99999;

float minaltitude = 99999;

float mintemp, maxtemp, basetemp, seapress;

byte mode;

char \*s;

const int SENSORADDRESS = 0x60; // address specific to the MPL3115A2, value found in datasheet

const int STATUS = 0x00; // status register

// by polling the status register for the new pressure data flag we avoid using timers

const int PT\_DATA\_CFG = 0x13; // set flag register

const byte tready = 2;

const byte pready = 4;

unsigned long timer, timedrate;

byte IICdata[5] = {0,0,0,0,0}; //buffer for sensor data

float altsmooth = 0; //for exponential smoothing

void setup(){

pinMode(buttonPin, INPUT\_PULLUP);

Wire.begin(); // Join i2c bus

Serial.begin(115200); // Start serial for output

if(debug) Serial.flush();

if(IIC\_Read(0x0C) == 196) { //checks whether sensor is readable (who\_am\_i bit)

if(debug) Serial.println("\nSensor OK.");

} else {

if(debug) Serial.println("\nSensor un-responsive.");

while(1); // stall forever

}

// proto pic bubble displays have common cathode

int displayType = COMMON\_CATHODE; //Your display is either common cathode or common anode

/\*

SevSeg library displays each digit once for 2mS

So 8ms for 4

Appears to be an error in the library when dp = 3 so use 4

This pinout is for a bubble display

Declare what pins are connected to the GND pins (cathodes)

\*/

int digit1 = A1; //Pin 12

int digit2 = 12; //Pin 9

int digit3 = 11; //Pin 8

int digit4 = 4; //Pin 6

//Declare what pins are connected to the segments (anodes)

int segA = A0; //Pin 11

int segB = 10; //Pin 7

int segC = 6; //Pin 4

int segD = 8; //Pin 2

int segE = 9; //Pin 1

int segF = 13; //Pin 10

int segG = 5; //Pin 5

int segDP= 7; //Pin 3

int numberOfDigits = 4; //Do you have a 1, 2 or 4 digit display?

myDisplay.Begin(displayType, numberOfDigits, digit1, digit2, digit3, digit4, segA, segB, segC, segD, segE, segF, segG, segDP);

calibrate();

oneshot();

}

void loop() {

float myaltitude, mytemperature, pressure;

byte dp;

checkbutton();

if(mode != STBY){

if((IIC\_Read(STATUS) & pready) == pready){ // new altitude data ready

myaltitude = Alt\_Read();

//exponential smoothing to get a smooth time series

//altsmooth=(altsmooth \* 3 + myaltitude)/4;

average[avpos] = myaltitude;

avpos++;

if(avpos == averaged){

altsmooth = 0;

for (byte i=0; i<averaged; i++) altsmooth += average[i];

altsmooth = altsmooth / averaged;

avpos = 0;

}

mytemperature = Temp\_Read(); // temp in centigrade

// reset for new altitude sample

oneshot();

if (mytemperature > maxtemp) maxtemp = mytemperature;

if (mytemperature < mintemp) mintemp = mytemperature;

if (myaltitude > maxaltitude) maxaltitude = myaltitude;

if (myaltitude < minaltitude) minaltitude = myaltitude;

dp =0;

switch (mode) {

case ALT:

sprintf(tempString, "%4d", int(myaltitude + 0.5)); //Convert altitude into a right adjusted string

break;

case AVER:

sprintf(tempString, "%4d", int(altsmooth + 0.5)); //Convert smoothed altitude

break;

case TOP:

sprintf(tempString, "%4d", int(maxaltitude + 0.5));

break;

case BOT:

sprintf(tempString, "%4d", int(minaltitude + 0.5));

break;

case DIFF:

sprintf(tempString, "%4d", int(maxaltitude - minaltitude + 0.5));

break;

case TEMP:

//dtostrf(mytemperature,4,1,tempString);

sprintf(tempString, "%4.2d", int(mytemperature\*10 + 0.5));

dp = 4;

break;

case TDIF:

//dtostrf(maxtemp - mintemp,4,1,tempString);

// %[flags][width][.precision][length]specifier

// the following %4.2d translates to width 4 minimum 2 Signed decimal integer

sprintf(tempString, "%4.2d", int((maxtemp - mintemp)\*10 + 0.5));

dp = 4;

break;

case STBY:

sprintf(tempString, " ");

break;

}

s = tempString;

if (eco) longDisplay(s,48,dp);

if (debug){

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

Serial.print(myaltitude, 1);Serial.print(" (");Serial.print(minaltitude,1);

Serial.print("-");Serial.print(maxaltitude,1);Serial.print(") Temperature ");

Serial.print(mytemperature, 1);Serial.print(" (");Serial.print(mintemp,1);

Serial.print("-");Serial.print(maxtemp,1);Serial.print(") Display (");

Serial.print(mode);Serial.print("):");Serial.print(s);

Serial.println();Serial.flush();

}

}

}

if (!eco) myDisplay.DisplayString(s, dp);

}

void oneshot(){

if(maxoversample){

IIC\_Write(0x26, 0b10111001);

IIC\_Write(0x26, 0b10111011);

}else{

IIC\_Write(0x26, 0b10110001);

IIC\_Write(0x26, 0b10110011);

}

}

float Temp\_Read(){

// return temperature in C

// correction applied by writing to reg 0x2C in calibrate

// NB a call to this function assumes function Baro/Alt\_Read has occured

// otherwise IIC\_ReadData() will not have been called for current value

bool negSign = false;

byte msb = IICdata[3];

byte lsb = IICdata[4];

word foo = 0;

//Check for 2s compliment (negative temperature representation!)

if(msb > 0x7F){

foo = ~((msb << 8) + lsb) + 1; //2’s complement

msb = foo >> 8;

lsb = foo & 0x00F0;

negSign = true;

}

// The least significant bytes l\_altitude and l\_temp are 4-bit,

// fractional values, so you must cast the calulation in (float),

// shift the value over 4 spots to the right and divide by 16 (since

// there are 16 values in 4-bits).

float templsb = (lsb>>4)/16.0; //temp, fraction of a degree

float temperature = (float)(msb + templsb);

if (negSign) temperature = 0 - temperature;

return temperature;

//return IICdata[3]+(float)(IICdata[4]>>4)/16 + tempcorrection;

}

float Baro\_Read(){

//this function takes values from the read buffer and converts them to pressure units

IIC\_ReadData(); //reads registers from the sensor

unsigned long m\_altitude = IICdata[0];

unsigned long c\_altitude = IICdata[1];

float l\_altitude = (float)(IICdata[2]>>4)/4; //dividing by 4, since two lowest bits are fractional value

return((float)(m\_altitude<<10 | c\_altitude<<2)+l\_altitude); //shifting 2 to the left to make room for LSB

}

float Alt\_Read(){

//Reads altitude in m (if CTRL\_REG1 is set to altitude mode)

IIC\_ReadData(); //reads registers from the sensor

int m\_altitude = IICdata[0];

int c\_altitude = IICdata[1];

float l\_altitude = (float)(IICdata[2]>>4)/16;

return((float)((m\_altitude << 8)|c\_altitude) + l\_altitude);

}

byte IIC\_Read(byte regAddr){

// This function reads one byte over I2C

Wire.beginTransmission(SENSORADDRESS);

Wire.write(regAddr); // Address of CTRL\_REG1

Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. Works in Arduino V1.0.1

Wire.requestFrom(SENSORADDRESS, 1);

return Wire.read();

}

void IIC\_Write(byte regAddr, byte value){

// This function writes one byto over I2C

Wire.beginTransmission(SENSORADDRESS);

Wire.write(regAddr);

Wire.write(value);

Wire.endTransmission(true);

}

void IIC\_ReadData(){ //Read Altitude/Barometer and Temperature data (5 bytes)

//This is faster than reading individual register, as the sensor automatically

//increments the register address, so we just keep reading...

byte i=0;

Wire.beginTransmission(SENSORADDRESS);

Wire.write(0x01); // Address of CTRL\_REG1

Wire.endTransmission(false);

Wire.requestFrom(SENSORADDRESS,5); //read 5 bytes: 3 for altitude or pressure, 2 for temperature

while(Wire.available()) IICdata[i++] = Wire.read();

}

void checkbutton(){

// look for a negative transition

if (digitalRead(buttonPin) == HIGH) return;

// increment mode variable

mode++;

switch (mode) {

case ALT:

s = "Alt ";

break;

case AVER:

s = "AUER";

break;

case TOP:

s = "HiGH";

break;

case BOT:

s = "Lo ";

break;

case DIFF:

s = "diFF";

break;

case TEMP:

s = "CEnt";

break;

case TDIF:

s = "tdiF";

break;

case SET:

s = "SEt ";

break;

case STBY:

s = "StbY";

}

if (debug) Serial.println(s);

longDisplay(s,504,0);

unsigned long dbTime;

byte state;

dbTime= millis();

again:

do{// wait for button release

state = digitalRead(buttonPin);

if (state == LOW) dbTime= millis();

} while(state == LOW);

if(millis()-dbTime < 100) goto again; // require a debounce period

if(mode == SET)setheight();

if(mode == STBY){

IIC\_Write(0x26, IIC\_Read(0x26) & 254); // standby on MPL3115A2- clear lowest bit

if (debug) Serial.println("Altimeter in Standby");

} else {

oneshot(); // prepare for next read

}

}

void setheight(){ // limits -10 to 400m

myDisplay.SetBrightness(100);

boolean change = false;

sprintf(tempString, "%4d", int(ALTBASIS));

longDisplay(tempString,1000,0);

boolean exit = false;

unsigned long timeNow = millis();

s="Pres";

do{ // wait 2 secs for a new button press

myDisplay.DisplayString(s,0);

if(!digitalRead(buttonPin) ){

exit = true;

break;

}

} while (millis()-timeNow < 2001 );

if(exit){

while(!digitalRead(buttonPin)){

timeNow = millis();

do{// wait for 300 mS for button release

if(digitalRead(buttonPin)){

exit = false; // signal end of select

break;

}

} while(millis()-timeNow < 301);

if (exit){

ALTBASIS ++;

change = true;

if (ALTBASIS > 400) ALTBASIS = -10;

sprintf(tempString, "%4d", int(ALTBASIS));

longDisplay(tempString,208,0);

}

}

}

sprintf(tempString, "%4d", int(ALTBASIS));

longDisplay(tempString,1000,0);

mode=0;

if(change){

calibrate();

}else{

longDisplay("Alt ",504,0);

/\*

wake up the sensor because previous mode was standby

Manual and testing indicate that altitude offset, temperature offset and

sea pressure correction remain intact

\*/

oneshot();

while ((IIC\_Read(STATUS) & pready) == 0);

IIC\_ReadData(); //reads registers from the sensor and reject

}

//altsmooth = 0; // reset averaged altitude- only required for expo average

avpos = 0; // reset average array pointer

myDisplay.SetBrightness(100);

}

void tempcalib(){

// temp correction = 0.0625 per bit

// -8 to +7.9375 128 = -8 127 = 7.9375

byte twoc;

if (tempcorrection <0) twoc = ~byte(abs(tempcorrection) \*16 - 1); else twoc = tempcorrection \* 16;

IIC\_Write(0x2C,twoc); //write temperature offset

}

void calibrate(){

// Reference sea pressure is 101326, atmospheric conditions alters this!

// Calibration requires estimating sea pressure using the measured pressure at a known altitude

myDisplay.SetBrightness(100);

longDisplay("Cali",696,0);

delay(2304); // let device settle mechanically after switch movement

float myaltitude;

byte samples = 10;

tempcalib(); // set temperaure offset

if (debug){

Serial.println("\nPressure calibration...\n");

Serial.print("OverSample ");Serial.println(maxoversample);

Serial.print("Wait time ");Serial.println(waitTime);

Serial.print("Base altitude ");Serial.print(ALTBASIS);Serial.println(" m");

Serial.print("Pressure offset ");Serial.print(IIC\_Read(0x2B),BIN);Serial.println(" (at 4 Pa per bit)");

Serial.print("Temperature offset ");Serial.print(IIC\_Read(0x2C),BIN);Serial.println(" (at 0.0625 C per bit)");

Serial.print("Altitude offset ");Serial.print(IIC\_Read(0x2D),BIN);Serial.println(" (at 1 m per bit)");

}

IIC\_Write(0x2D,0); //write altitude offset=0 (because calculation below is based on offset = 0)

IIC\_Write(0x26, 0b10111000);

IIC\_Write(PT\_DATA\_CFG, 0x07); // Enable all three pressure and temp event flags

// read and discard 1st altitude value

IIC\_Write(0x26, 0b10111001);

IIC\_Write(0x26, 0b10111011);

while ((IIC\_Read(STATUS) & pready) == 0);

Alt\_Read();

// calibration uses the maximum precision readings

IIC\_Write(0x26, 0b10111001);

IIC\_Write(0x26, 0b10111011);

while ((IIC\_Read(STATUS) & pready) == 0);

myaltitude = Alt\_Read();

if (debug){

Serial.print("Uncalibrated ");

Serial.print(myaltitude,1);

Serial.println(" m");

}

float currpress = 0;

basetemp = 0;

for (byte i=0; i < samples + 2; i++){

myDisplay.DisplayString("Cali", 0);

IIC\_Write(0x26, 0b00111001);

IIC\_Write(0x26, 0b00111011);

while ((IIC\_Read(STATUS) & (pready + tready)) < (pready +tready) ); // look for new pressure and temperature flag

if(i>1){ // discard first 2 readings

currpress += Baro\_Read();

basetemp += Temp\_Read();

}

}

currpress = currpress /samples;

basetemp = basetemp / samples;

mintemp = basetemp;

maxtemp = basetemp;

seapress = currpress/pow(1-ALTBASIS\*0.0000225577,5.255877);

if (debug){

Serial.print("Average Pressure ");

Serial.print(currpress,1);

Serial.println(" Pa");

Serial.print("Sea Pressure ");

Serial.print(seapress,1);

Serial.println(" Pa");

}

// This configuration option calibrates the sensor according to

// the sea level pressure for the measurement location (2 Pa per LSB)

IIC\_Write(0x14, (unsigned int)(seapress / 2)>>8);

IIC\_Write(0x15, (unsigned int)(seapress / 2)&0xFF);

IIC\_Write(0x26, 0b10111001);

IIC\_Write(0x26, 0b10111011);

while ((IIC\_Read(STATUS) & pready) == 0);

Alt\_Read();// discard

unsigned long startTime, endTime;

timedrate = 0;

myaltitude = 0;

for(byte i=0; i < 4; i++) { // settle altitude reading

myDisplay.DisplayString("Cali", 0);

startTime=millis();

oneshot(); // use selected over sample rate

while ((IIC\_Read(STATUS) & pready) == 0); // look for pressure/altitude flag

myaltitude += Alt\_Read();

endTime=millis();

timedrate += endTime - startTime;

}

myaltitude = myaltitude / 4;

timedrate = timedrate / 4;

sprintf(tempString, "%4d.2", int(basetemp\*10 + 0.5));

//dtostrf(basetemp,4,1,tempString); // this version leaves a blank digit at dp

s = tempString;

longDisplay(s,696,4); // display temperature

if (debug){

Serial.print("Calibrated ");Serial.print(myaltitude,1); Serial.println(" m");

Serial.print("Base temp ");Serial.print(basetemp,1); Serial.println(" C");

Serial.print("Average Oversample period ");Serial.print(timedrate); Serial.println(" mS\n");

Serial.flush();

}

myDisplay.SetBrightness(100);

}

void longDisplay(char \*temp,unsigned int period, byte dp){

unsigned long timeNow = millis();

do{ // allow for operator to read change in display

myDisplay.DisplayString(temp, dp); // output new mode

} while (millis()- timeNow < period);

}