```
/**                                                                [xneb2_Tx_33_+altim]

*  Reads Bosch Sensortec's BMP085 athmospheric pressure sensor and generates a variometric
*  audio signal with tone pitch proportional to the current (positive) vertical velocity.
*
*  Sufficient sampling speed is critical for the performance of this application, but
*  absolute athmospheric pressure input values are not necessary, per se. Therefore we should
*  be able to speed things up significantly, while still using reasonably accurate data,
*  by foregoing the normal temperature-compensatated conversion routine, and instead simply
*  multiply each new (internally oversampled) measurement by a constant factor, which was
*  automatically calibrated against the conditions at launch-time. The lesser computational
*  load frees up time in the loop to perform other desireable tasks, like RF-transmission
*  and driving servos.
*
*  Variometer specific code by Martin Bergman <bergman.martin at gmail dot com> 2011.
*  License: CC BY-SA v3.0 - http://creativecommons.org/licenses/by-sa/3.0/legalcode .
*
*  Connections to Arduino I/O board :
*   Vcc -> 3V3
*   GND -> GND
*   SCL -> A5
*   SDA -> A4.
*   RF_Vcc -> 5V
*   RF_GND -> GND
*   RF_Tx  -> D12
*  Cp the schematic "Ardweeny_with_Variometer_and_RF433_Tx.pdf"
*
*  References:
*   http://www.bosch-sensortec.com/content/language1/downloads/BST-BMP085-DS000-06.pdf
*   http://interactive-matter.org/2009/12/arduino-barometric-pressure-sensor-bmp085/
*   http://news.jeelabs.org/2009/02/19/hooking-up-a-bmp085-sensor/
*   http://wmrx00.sourceforge.net/Arduino/BMP085-Calcs.pdf
*/

#include <Wire.h>
#include <EWMAcumStiction.h>
#include <VirtualWire.h>
#define I2C_ADDRESS 0x77
#define DEBUG

const unsigned char oversampling_setting = 3;     //OSS is either 0, 1, 2 or 3
const unsigned char pressure_waittime[4] = { 5, 8, 14, 26 };
const byte loopingtime = 70; //31;
/*
    The following were the fastest "loopingtimes" (update intervals) achieved in testing:
    ------------------------------------------------------------------------------
    Oversampling setting:                  0   1   2   3
    ------------------------------------------------------------------------------
    While doing full conversions:          35  45  55  80
    Without full conversions
      (~ 4k raw units for each of 2^OSS samples):   16  19  26  37
    Pseudo pressures only (minimum printing,
       ~ 10k Pa units for each of 2^OSS samples):   --  --  --  31
*/

/*  We first create a filter object that smoothes and conditions the pressure sensor's output.
    The suggested ranges of the required parameters are shown in [brackets]:
        alpha (float):  A smoothing coefficient, such as 0 < alpha < 1 [0.04 - 0.1]
        laxus (char):   Maximal amount of +/-"spread" in legitimate sensor values [2-5]
        sinkref (char): Nominal still-air sinkrate (see header file for explanation) [2-4]
        limen (char):   Threshold value for signalling climb [4-6]              */

EWMAcumStiction vario(0.1, 4, 0, 2);

// Sensor's calibration values (to be retrieved from its EEPROM):
int ac1, ac2, ac3, b1, b2, mb, mc, md;
unsigned int ac4, ac5, ac6;

void bmp085_read_temperature_and_pressure(int& temperature, long& pressure);
int P2PuncRatio_x10k = 0;
unsigned long previousMillis = 0;
long Pgnd = 0;      // The registered athmospehric pressure at ground level (deciPascal units)
byte msg[2];        // Virtual Wire stuff

void setup()
{
#ifdef DEBUG
  Serial.begin(9600);
  Serial.flush();
  Serial.println();
  Serial.println("    Setting up the BMP085 sensor");
  print_info();
#endif
  Wire.begin();
```

```
  bmp085_get_cal_data();
#ifdef DEBUG
  print_cal_data();
  Serial.println();
  Serial.println("(3) Calibrating for the present conditions... ");
  Serial.println("    (Please wait 5 secs)");
  Serial.println();
#endif
  establish_conversion_ratio();  // sets the 'P2PuncRatio' and 'Pgnd' values for the session
  vario.startup(Pgnd, 9);  // attaches an init value and an output pin to the 'vario' object
  pinMode(13, OUTPUT);        // Tx LED flash
  vw_setup(1000);   // We keep a sedate pace (max 20 x 1 byte = 160 bit/s to be expected)
  vw_set_tx_pin(12); // We will be Tx-ing on VW default pin 12
}

void loop()
{
  unsigned long currentMillis = millis();
  if(currentMillis - previousMillis >= loopingtime) {
      previousMillis = currentMillis;
      long Punc = bmp085_read_up();                  // uncompensated pressure reading (raw units)
      long Ppseudo1 = (P2PuncRatio_x10k * Punc) / 1000; // first approxim. of true pressure (dPa)
      long Ppseudo2 = Pgnd - ((5 * (Pgnd - Ppseudo1)) / 4); // close-enough correction (dPa)

      vario.read(Ppseudo2);
      digitalWrite(13, false);
      msg[0] = vario.signal();
      msg[1] = altimeter(Ppseudo2);
#ifdef DEBUG
      Serial.print("  p2: "); Serial.print(Ppseudo2,DEC);
      Serial.print("....Alt :"); Serial.println(altimeter(Ppseudo2));
#endif
      vw_send( (uint8_t *)msg, sizeof(msg));
      vw_wait_tx();
      delay(10);
      digitalWrite(13, true);
      }
}

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

void print_info()
{
  Serial.println("*****************************************");
  Serial.println();
  Serial.print("(1) Oversampling setting: ");
  Serial.print(oversampling_setting, DEC);
  Serial.print("  (");
  byte overSampling = pow(2,(oversampling_setting));
  Serial.print(overSampling, DEC);
  Serial.println(" samples)");
  Serial.print("         Looping time is: ");
  Serial.print(loopingtime,DEC);
  Serial.print(" ms  (");
  Serial.print(1000/loopingtime,DEC);
  Serial.println(" Hz)");
  Serial.println();
}

void establish_conversion_ratio()
{
  int temperature = 0;
  long pressure = 0;     // the temperature-compensated, "true" pressure of a sample (Pa units)
  long totalTempoPunc = 0;         // sum of temporary uncompensated pressure values (raw units)
  long totalTempoPcomp = 0;        // sum of temporary compensated pressure values (Pa units)
  unsigned long averagePunc = 0;     // average of 50 uncompensated pressure samples (raw units)
  unsigned long averagePcomp_x10k = 0; // average of 50 compensated samples multiplied by 10 000

  for (int index = 0; index < 50; index++) {
    totalTempoPunc += bmp085_read_up();        // add up 50 uncompensated sample values
    bmp085_read_temperature_and_pressure(&temperature,&pressure);
    totalTempoPcomp += pressure;               // add up 50 compensated sample values
    delay(100);
  }
  averagePunc = totalTempoPunc / 50;
  averagePcomp_x10k = totalTempoPcomp * 200;
  Pgnd = totalTempoPcomp / 5;  // we keep the sampled average expressed as deciPascals (dPa)
  P2PuncRatio_x10k = averagePcomp_x10k / averagePunc;
  tone(9, 150, 100);delay(100);tone(9, 400, 250);
#ifdef DEBUG
  Serial.println("*****************************************");
  Serial.print("    Pcomp/Punc ratio this time is: 0.");
  Serial.println(P2PuncRatio_x10k, DEC);
  Serial.println("*****************************************");
```

```
  Serial.println();
  Serial.print("(4)");delay(1000);Serial.println(" Let's go!");
  Serial.println();
//  Serial.println(Pgnd,DEC);             // Debuggging: (average true P @ launch elevation, dPa)
//  Serial.println(averagePcomp_x10k,DEC);// Debuggging: (ditto scaled up....)
//  Serial.println(averagePunc,DEC);      // Debuggging
  Serial.println("_____________________________________________");
#endif
  delay(1000);
}

int altimeter(long Pps) {
    int multip_5m = (Pgnd - Pps) / 600;     // 600 dPa equals approx 5 m!
    return multip_5m;
}



/** ********** The Bosch API main computation, using integer math only: *************/

void bmp085_read_temperature_and_pressure(int* temperature, long* pressure) {
  int ut= bmp085_read_ut();
  long up = bmp085_read_up();
  long x1, x2, x3, b3, b5, b6, p;
  unsigned long b4, b7;

  // Calculate the temperature
  x1 = ((long)ut - ac6) * ac5 >> 15;
  x2 = ((long) mc << 11) / (x1 + md);
  b5 = x1 + x2;
  *temperature = (b5 + 8) >> 4;

  // Calculate the pressure
  b6 = b5 - 4000;
  x1 = (b2 * (b6 * b6 >> 12)) >> 11;
  x2 = ac2 * b6 >> 11;
  x3 = x1 + x2;
  b3 = (((((long) ac1) * 4 + x3)<<oversampling_setting) + 2) >> 2;
  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 >> oversampling_setting);
  p = b7 < 0x80000000 ? (b7 * 2) / b4 : (b7 / b4) * 2;
  x1 = (p >> 8) * (p >> 8);
  x1 = (x1 * 3038) >> 16;
  x2 = (-7357 * p) >> 16;
  *pressure = p + ((x1 + x2 + 3791) >> 4);
}


void bmp085_get_cal_data() {
  ac1 = read_int_register(0xAA);
  ac2 = read_int_register(0xAC);
  ac3 = read_int_register(0xAE);
  ac4 = read_int_register(0xB0);
  ac5 = read_int_register(0xB2);
  ac6 = read_int_register(0xB4);
  b1 = read_int_register(0xB6);
  b2 = read_int_register(0xB8);
  mb = read_int_register(0xBA);
  mc = read_int_register(0xBC);
  md = read_int_register(0xBE);
}


void print_cal_data() {                  // These printlines are for debugging mainly:
  delay(2000);Serial.println("(2) Now retrieving calibration data...");
  delay(500);Serial.println();
//  Serial.print("AC1: ");Serial.println(ac1,DEC);
//  Serial.print("AC2: ");Serial.println(ac2,DEC);
//  Serial.print("AC3: ");Serial.println(ac3,DEC);
//  Serial.print("AC4: ");Serial.println(ac4,DEC);
//  Serial.print("AC5: ");Serial.println(ac5,DEC);
//  Serial.print("AC6: ");Serial.println(ac6,DEC);
//  Serial.print("B1: ");Serial.println(b1,DEC);
//  Serial.print("B2: ");Serial.println(b2,DEC);
//  Serial.print("MB: ");Serial.println(mb,DEC);
//  Serial.print("MC: ");Serial.println(mc,DEC);
//  Serial.print("MD: ");Serial.println(md,DEC);
}

/** ****************** Various I2C (TWI) helper functions: **********************/

```

```
unsigned int bmp085_read_ut() {
  write_register(0xf4,0x2e);
  delay(5); // longer than 4.5 ms
  return read_int_register(0xf6);
}


long bmp085_read_up() {
  write_register(0xf4,0x34+(oversampling_setting<<6));
  delay(pressure_waittime[oversampling_setting]);
  unsigned char msb, lsb, xlsb;
  Wire.beginTransmission(I2C_ADDRESS);
  Wire.send(0xf6);                  // register to read
  Wire.endTransmission();

  Wire.requestFrom(I2C_ADDRESS, 3); // read a byte
  while(!Wire.available()) {}       // waiting
  msb = Wire.receive();
  while(!Wire.available()) {}       // waiting
  lsb |= Wire.receive();
  while(!Wire.available()) {}       // waiting
  xlsb |= Wire.receive();
  return (((long)msb<<16) | ((long)lsb<<8) | ((long)xlsb)) >>(8-oversampling_setting);
}


void write_register(unsigned char r, unsigned char v) {
  Wire.beginTransmission(I2C_ADDRESS);
  Wire.send(r);
  Wire.send(v);
  Wire.endTransmission();
}


char read_register(unsigned char r) {
  unsigned char v;
  Wire.beginTransmission(I2C_ADDRESS);
  Wire.send(r);             // register to read
  Wire.endTransmission();

  Wire.requestFrom(I2C_ADDRESS, 1); // read a byte
  while(!Wire.available()) {}       // waiting
  v = Wire.receive();
  return v;
}


int read_int_register(unsigned char r) {
  unsigned char msb, lsb;
  Wire.beginTransmission(I2C_ADDRESS);
  Wire.send(r);             // register to read
  Wire.endTransmission();

  Wire.requestFrom(I2C_ADDRESS, 2); // read a byte
  while(!Wire.available()) {}       // waiting
  msb = Wire.receive();
  while(!Wire.available()) {}       // waiting
  lsb = Wire.receive();
  return (((int)msb<<8) | ((int)lsb));
}
```