sparkfun / **MPL3115A2_Breakout**

Watch 42 Star

branch: **master** MPL3115A2_Breakout / firmware / MPL3115A2 / **MPL3115A2.ino**

nseidle on 23 Oct 2013 Changed sensor checking to one shot polling.

1 contributor

373 lines (309 sloc) 11.017 kb

Raw Blame History

```
/*
 MPL3115A2 Altitude Sensor Example
 By: A.Weiss, 7/17/2012, changes Nathan Seidle Sept 23rd, 2013
 License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).

 Hardware Connections (Breakoutboard to Arduino):
 -VCC = 3.3V
 -SDA = A4
 -SCL = A5
 -INT pins can be left unconnected for this demo

 Usage:
 -Serial terminal at 9600bps
 -Prints altitude in meters, temperature in degrees C, with 1/16 resolution.
 -software enabled interrupt on new data, ~1Hz with full resolution

 During testing, GPS with 9 sattelites reported 5393ft, sensor reported 5360ft (delta of 33ft). Very close!

 */

#include <Wire.h> // for IIC communication

#define STATUS     0x00
#define OUT_P_MSB  0x01
#define OUT_P_CSB  0x02
#define OUT_P_LSB  0x03
#define OUT_T_MSB  0x04
#define OUT_T_LSB  0x05
#define DR_STATUS  0x06
#define OUT_P_DELTA_MSB  0x07
#define OUT_P_DELTA_CSB  0x08
#define OUT_P_DELTA_LSB  0x09
#define OUT_T_DELTA_MSB  0x0A
#define OUT_T_DELTA_LSB  0x0B
#define WHO_AM_I   0x0C
#define F_STATUS   0x0D
#define F_DATA     0x0E
#define F_SETUP    0x0F
#define TIME_DLY   0x10
#define SYSMOD     0x11
#define INT_SOURCE 0x12
#define PT_DATA_CFG 0x13
#define BAR_IN_MSB 0x14
#define BAR_IN_LSB 0x15
#define P_TGT_MSB  0x16
#define P_TGT_LSB  0x17
#define T_TGT      0x18
#define P_WND_MSB  0x19
#define P_WND_LSB  0x1A
#define T_WND      0x1B
#define P_MIN_MSB  0x1C
#define P_MIN_CSB  0x1D
#define P_MIN_LSB  0x1E
#define T_MIN_MSB  0x1F
```

```
#define T_MIN_LSB  0x20
#define P_MAX_MSB  0x21
#define P_MAX_CSB  0x22
#define P_MAX_LSB  0x23
#define T_MAX_MSB  0x24
#define T_MAX_LSB  0x25
#define CTRL_REG1  0x26
#define CTRL_REG2  0x27
#define CTRL_REG3  0x28
#define CTRL_REG4  0x29
#define CTRL_REG5  0x2A
#define OFF_P      0x2B
#define OFF_T      0x2C
#define OFF_H      0x2D

#define MPL3115A2_ADDRESS 0x60 // 7-bit I2C address

long startTime;

void setup()
{
  Wire.begin();        // join i2c bus
  Serial.begin(57600);  // start serial for output

  if(IIC_Read(WHO_AM_I) == 196)
    Serial.println("MPL3115A2 online!");
  else
    Serial.println("No response - check connections");

  // Configure the sensor
  setModeAltimeter(); // Measure altitude above sea level in meters
  //setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa

  setOversampleRate(7); // Set Oversample to the recommended 128
  enableEventFlags(); // Enable all three pressure and temp event flags
}

void loop()
{
  startTime = millis();

  float altitude = readAltitude();
  Serial.print("Altitude(m):");
  Serial.print(altitude, 2);

  //altitude = readAltitudeFt();
  //Serial.print(" Altitude(ft):");
  //Serial.print(altitude, 2);

  /*float pressure = readPressure();
   Serial.print(" Pressure(Pa):");
   Serial.println(pressure, 2);*/

  //float temperature = readTemp();
  //Serial.print(" Temp(c):");
  //Serial.print(temperature, 2);

  //float temperature = readTempF();
  //Serial.print(" Temp(f):");
  //Serial.print(temperature, 2);

  Serial.print(" time diff:");
  Serial.print(millis() - startTime);

  Serial.println();

  //delay(1);
}

//Returns the number of meters above sea level
float readAltitude()
```

```
{
  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  //Wait for PDR bit, indicates we have new pressure data
  int counter = 0;
  while( (IIC_Read(STATUS) & (1<<1)) == 0)
  {
      if(++counter > 100) return(-999); //Error out
      delay(1);
  }

  // Read pressure registers
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(OUT_P_MSB);  // Address of data to get
  Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
  Wire.requestFrom(MPL3115A2_ADDRESS, 3); // Request three bytes

  //Wait for data to become available
  counter = 0;
  while(Wire.available() < 3)
  {
    if(counter++ > 100) return(-999); //Error out
    delay(1);
  }

  byte msb, csb, lsb;
  msb = Wire.read();
  csb = Wire.read();
  lsb = Wire.read();

  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  // 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 tempcsb = (lsb>>4)/16.0;

  float altitude = (float)( (msb << 8) | csb) + tempcsb;

  return(altitude);
}

//Returns the number of feet above sea level
float readAltitudeFt()
{
  return(readAltitude() * 3.28084);
}

//Reads the current pressure in Pa
//Unit must be set in barometric pressure mode
float readPressure()
{
  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  //Wait for PDR bit, indicates we have new pressure data
  int counter = 0;
  while( (IIC_Read(STATUS) & (1<<2)) == 0)
  {
      if(++counter > 100) return(-999); //Error out
      delay(1);
  }

  // Read pressure registers
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(OUT_P_MSB);  // Address of data to get
  Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
  Wire.requestFrom(MPL3115A2_ADDRESS, 3); // Request three bytes

  //Wait for data to become available
  counter = 0;
  while(Wire.available() < 3)
  {
    if(counter++ > 100) return(-999); //Error out
    delay(1);
  }

  byte msb, csb, lsb;
  msb = Wire.read();
  csb = Wire.read();
  lsb = Wire.read();

  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  // Pressure comes back as a left shifted 20 bit number
  long pressure_whole = (long)msb<<16 | (long)csb<<8 | (long)lsb;
  pressure_whole >>= 6; //Pressure is an 18 bit number with 2 bits of decimal. Get rid of decimal portion.

  lsb &= 0b00110000; //Bits 5/4 represent the fractional component
  lsb >>= 4; //Get it right aligned
  float pressure_decimal = (float)lsb/4.0; //Turn it into fraction

  float pressure = (float)pressure_whole + pressure_decimal;

  return(pressure);
}

float readTemp()
{
  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  //Wait for TDR bit, indicates we have new temp data
  int counter = 0;
  while( (IIC_Read(STATUS) & (1<<1)) == 0)
  {
      if(++counter > 100) return(-999); //Error out
      delay(1);
  }

  // Read temperature registers
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(OUT_T_MSB);  // Address of data to get
  Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
  Wire.requestFrom(MPL3115A2_ADDRESS, 2); // Request two bytes

  //Wait for data to become available
  counter = 0;
  while(Wire.available() < 2)
  {
    if(++counter > 100) return(-999); //Error out
    delay(1);
  }

  byte msb, lsb;
  msb = Wire.read();
  lsb = Wire.read();

  // 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);

  return(temperature);
}

//Give me temperature in fahrenheit!
float readTempF()
{
  return((readTemp() * 9.0)/ 5.0 + 32.0); // Convert celsius to fahrenheit
```

```
}

//Sets the mode to Barometer
//CTRL_REG1, ALT bit
void setModeBarometer()
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= ~(1<<7); //Clear ALT bit
  IIC_Write(CTRL_REG1, tempSetting);
}

//Sets the mode to Altimeter
//CTRL_REG1, ALT bit
void setModeAltimeter()
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting |= (1<<7); //Set ALT bit
  IIC_Write(CTRL_REG1, tempSetting);
}

//Puts the sensor in standby mode
//This is needed so that we can modify the major control registers
void setModeStandby()
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= ~(1<<0); //Clear SBYB bit for Standby mode
  IIC_Write(CTRL_REG1, tempSetting);
}

//Puts the sensor in active mode
//This is needed so that we can modify the major control registers
void setModeActive()
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting |= (1<<0); //Set SBYB bit for Active mode
  IIC_Write(CTRL_REG1, tempSetting);
}

//Setup FIFO mode to one of three modes. See page 26, table 31
//From user jr4284
void setFIFOMode(byte f_Mode)
{
  if (f_Mode > 3) f_Mode = 3; // FIFO value cannot exceed 3.
  f_Mode <<= 6; // Shift FIFO byte left 6 to put it in bits 6, 7.

  byte tempSetting = IIC_Read(F_SETUP); //Read current settings
  tempSetting &= ~(3<<6); // clear bits 6, 7
  tempSetting |= f_Mode; //Mask in new FIFO bits
  IIC_Write(F_SETUP, tempSetting);
}

//Call with a rate from 0 to 7. See page 33 for table of ratios.
//Sets the over sample rate. Datasheet calls for 128 but you can set it
//from 1 to 128 samples. The higher the oversample rate the greater
//the time between data samples.
void setOversampleRate(byte sampleRate)
{
  if(sampleRate > 7) sampleRate = 7; //OS cannot be larger than 0b.0111
  sampleRate <<= 3; //Align it for the CTRL_REG1 register

  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= 0b11000111; //Clear out old OS bits
  tempSetting |= sampleRate; //Mask in new OS bits
  IIC_Write(CTRL_REG1, tempSetting);
}

//Clears then sets the OST bit which causes the sensor to immediately take another reading
//Needed to sample faster than 1Hz
void toggleOneShot(void)
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= ~(1<<1); //Clear OST bit
  IIC_Write(CTRL_REG1, tempSetting);

  tempSetting = IIC_Read(CTRL_REG1); //Read current settings to be safe
  tempSetting |= (1<<1); //Set OST bit
  IIC_Write(CTRL_REG1, tempSetting);
}

//Enables the pressure and temp measurement event flags so that we can
//test against them. This is recommended in datasheet during setup.
void enableEventFlags()
{
  IIC_Write(PT_DATA_CFG, 0x07); // Enable all three pressure and temp event flags
}

// These are the two I2C functions in this sketch.
byte IIC_Read(byte regAddr)
{
  // This function reads one byte over IIC
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(regAddr);  // Address of CTRL_REG1
  Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
  Wire.requestFrom(MPL3115A2_ADDRESS, 1); // Request the data...
  return Wire.read();
}

void IIC_Write(byte regAddr, byte value)
{
  // This function writes one byto over IIC
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(regAddr);
  Wire.write(value);
  Wire.endTransmission(true);
}
```