



This repository Search

[Explore](#) [Gist](#) [Blog](#) [Help](#)

gamondue

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



```
1  /*
2  MPL3115A2 Altitude Sensor Example
3  By: A.Weiss, 7/17/2012, changes Nathan Seidle Sept 23rd, 2013
4  License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).
5
6  Hardware Connections (Breakoutboard to Arduino):
7  -VCC = 3.3V
8  -SDA = A4
9  -SCL = A5
10 -INT pins can be left unconnected for this demo
11
12 Usage:
13 -Serial terminal at 9600bps
14 -Prints altitude in meters, temperature in degrees C, with 1/16 resolution.
15 -software enabled interrupt on new data, ~1Hz with full resolution
16
17 During testing, GPS with 9 sattelites reported 5393ft, sensor reported 5360ft (delta of 33ft). Very close!
18
19 */
20
21 #include <Wire.h> // for IIC communication
22
23 #define STATUS      0x00
24 #define OUT_P_MSB   0x01
25 #define OUT_P_CSB   0x02
26 #define OUT_P_LSB   0x03
27 #define OUT_T_MSB   0x04
28 #define OUT_T_LSB   0x05
29 #define DR_STATUS   0x06
30 #define OUT_P_DELTA_MSB 0x07
31 #define OUT_P_DELTA_CSB 0x08
32 #define OUT_P_DELTA_LSB 0x09
33 #define OUT_T_DELTA_MSB 0x0A
34 #define OUT_T_DELTA_LSB 0x0B
35 #define WHO_AM_I    0x0C
36 #define F_STATUS    0x0D
37 #define F_DATA      0x0E
38 #define F_SETUP     0x0F
39 #define TIME_DLY    0x10
40 #define SYSMOD      0x11
41 #define INT_SOURCE  0x12
42 #define PT_DATA_CFG 0x13
43 #define BAR_IN_MSB  0x14
44 #define BAR_IN_LSB  0x15
45 #define P_TGT_MSB   0x16
46 #define P_TGT_LSB   0x17
47 #define T_TGT       0x18
48 #define P_WND_MSB   0x19
49 #define P_WND_LSB   0x1A
50 #define T_WND       0x1B
51 #define P_MIN_MSB   0x1C
52 #define P_MIN_CSB   0x1D
53 #define P_MIN_LSB   0x1E
54 #define T_MIN_MSB   0x1F
```

```
55 #define T_MIN_LSB 0x20
56 #define P_MAX_MSB 0x21
57 #define P_MAX_CSB 0x22
58 #define P_MAX_LSB 0x23
59 #define T_MAX_MSB 0x24
60 #define T_MAX_LSB 0x25
61 #define CTRL_REG1 0x26
62 #define CTRL_REG2 0x27
63 #define CTRL_REG3 0x28
64 #define CTRL_REG4 0x29
65 #define CTRL_REG5 0x2A
66 #define OFF_P 0x2B
67 #define OFF_T 0x2C
68 #define OFF_H 0x2D
69
70 #define MPL3115A2_ADDRESS 0x60 // 7-bit I2C address
71
72 long startTime;
73
74 void setup()
75 {
76   Wire.begin(); // join i2c bus
77   Serial.begin(57600); // start serial for output
78
79   if(IIC_Read(WHO_AM_I) == 196)
80     Serial.println("MPL3115A2 online!");
81   else
82     Serial.println("No response - check connections");
83
84   // Configure the sensor
85   setModeAltimeter(); // Measure altitude above sea level in meters
86   //setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa
87
88   setOversampleRate(7); // Set Oversample to the recommended 128
89   enableEventFlags(); // Enable all three pressure and temp event flags
90 }
91
92 void loop()
93 {
94   startTime = millis();
95
96   float altitude = readAltitude();
97   Serial.print("Altitude(m):");
98   Serial.print(altitude, 2);
99
100   //altitude = readAltitudeFt();
101   //Serial.print(" Altitude(ft):");
102   //Serial.print(altitude, 2);
103
104   /*float pressure = readPressure();
105   Serial.print(" Pressure(Pa):");
106   Serial.println(pressure, 2);*/
107
108   //float temperature = readTemp();
109   //Serial.print(" Temp(c):");
110   //Serial.print(temperature, 2);
111
112   //float temperature = readTempF();
113   //Serial.print(" Temp(f):");
114   //Serial.print(temperature, 2);
115
116   Serial.print(" time diff:");
117   Serial.print(millis() - startTime);
118
119   Serial.println();
120
121   //delay(1);
122 }
123
124 //Returns the number of meters above sea level
125 float readAltitude()
```

```
126 {
127   toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
128
129   //Wait for PDR bit, indicates we have new pressure data
130   int counter = 0;
131   while( (IIC_Read(STATUS) & (1<<1)) == 0)
132   {
133     if(++counter > 100) return(-999); //Error out
134     delay(1);
135   }
136
137   // Read pressure registers
138   Wire.beginTransmission(MPL3115A2_ADDRESS);
139   Wire.write(OUT_P_MSB); // Address of data to get
140   Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
141   Wire.requestFrom(MPL3115A2_ADDRESS, 3); // Request three bytes
142
143   //Wait for data to become available
144   counter = 0;
145   while(Wire.available() < 3)
146   {
147     if(counter++ > 100) return(-999); //Error out
148     delay(1);
149   }
150
151   byte msb, csb, lsb;
152   msb = Wire.read();
153   csb = Wire.read();
154   lsb = Wire.read();
155
156   toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
157
158   // The least significant bytes l_altitude and l_temp are 4-bit,
159   // fractional values, so you must cast the calculation in (float),
160   // shift the value over 4 spots to the right and divide by 16 (since
161   // there are 16 values in 4-bits).
162   float tempcsb = (lsb>>4)/16.0;
163
164   float altitude = (float)( (msb << 8) | csb) + tempcsb;
165
166   return(altitude);
167 }
168
169 //Returns the number of feet above sea level
170 float readAltitudeFt()
171 {
172   return(readAltitude() * 3.28084);
173 }
174
175 //Reads the current pressure in Pa
176 //Unit must be set in barometric pressure mode
177 float readPressure()
178 {
179   toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
180
181   //Wait for PDR bit, indicates we have new pressure data
182   int counter = 0;
183   while( (IIC_Read(STATUS) & (1<<2)) == 0)
184   {
185     if(++counter > 100) return(-999); //Error out
186     delay(1);
187   }
188
189   // Read pressure registers
190   Wire.beginTransmission(MPL3115A2_ADDRESS);
191   Wire.write(OUT_P_MSB); // Address of data to get
192   Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
193   Wire.requestFrom(MPL3115A2_ADDRESS, 3); // Request three bytes
194
195   //Wait for data to become available
196   counter = 0;
```

```
197 while(Wire.available() < 3)
198 {
199     if(counter++ > 100) return(-999); //Error out
200     delay(1);
201 }
202
203 byte msb, csb, lsb;
204 msb = Wire.read();
205 csb = Wire.read();
206 lsb = Wire.read();
207
208 toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
209
210 // Pressure comes back as a left shifted 20 bit number
211 long pressure_whole = (long)msb<<16 | (long)csb<<8 | (long)lsb;
212 pressure_whole >>= 6; //Pressure is an 18 bit number with 2 bits of decimal. Get rid of decimal portion.
213
214 lsb &= 0b00110000; //Bits 5/4 represent the fractional component
215 lsb >>= 4; //Get it right aligned
216 float pressure_decimal = (float)lsb/4.0; //Turn it into fraction
217
218 float pressure = (float)pressure_whole + pressure_decimal;
219
220 return(pressure);
221 }
222
223 float readTemp()
224 {
225     toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
226
227     //Wait for TDR bit, indicates we have new temp data
228     int counter = 0;
229     while( (IIC_Read(STATUS) & (1<<1)) == 0)
230     {
231         if(++counter > 100) return(-999); //Error out
232         delay(1);
233     }
234
235     // Read temperature registers
236     Wire.beginTransmission(MPL3115A2_ADDRESS);
237     Wire.write(OUT_T_MSB); // Address of data to get
238     Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
239     Wire.requestFrom(MPL3115A2_ADDRESS, 2); // Request two bytes
240
241     //Wait for data to become available
242     counter = 0;
243     while(Wire.available() < 2)
244     {
245         if(++counter > 100) return(-999); //Error out
246         delay(1);
247     }
248
249     byte msb, lsb;
250     msb = Wire.read();
251     lsb = Wire.read();
252
253     // The least significant bytes l_altitude and l_temp are 4-bit,
254     // fractional values, so you must cast the calculation in (float),
255     // shift the value over 4 spots to the right and divide by 16 (since
256     // there are 16 values in 4-bits).
257     float templs = (lsb>>4)/16.0; //temp, fraction of a degree
258
259     float temperature = (float)(msb + templs);
260
261     return(temperature);
262 }
263
264 //Give me temperature in fahrenheit!
265 float readTempF()
266 {
267     return((readTemp() * 9.0)/ 5.0 + 32.0); // Convert celsius to fahrenheit
```

```
268 }
269
270 //Sets the mode to Barometer
271 //CTRL_REG1, ALT bit
272 void setModeBarometer()
273 {
274     byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
275     tempSetting &= ~(1<<7); //Clear ALT bit
276     IIC_Write(CTRL_REG1, tempSetting);
277 }
278
279 //Sets the mode to Altimeter
280 //CTRL_REG1, ALT bit
281 void setModeAltimeter()
282 {
283     byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
284     tempSetting |= (1<<7); //Set ALT bit
285     IIC_Write(CTRL_REG1, tempSetting);
286 }
287
288 //Puts the sensor in standby mode
289 //This is needed so that we can modify the major control registers
290 void setModeStandby()
291 {
292     byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
293     tempSetting &= ~(1<<0); //Clear SBYB bit for Standby mode
294     IIC_Write(CTRL_REG1, tempSetting);
295 }
296
297 //Puts the sensor in active mode
298 //This is needed so that we can modify the major control registers
299 void setModeActive()
300 {
301     byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
302     tempSetting |= (1<<0); //Set SBYB bit for Active mode
303     IIC_Write(CTRL_REG1, tempSetting);
304 }
305
306 //Setup FIFO mode to one of three modes. See page 26, table 31
307 //From user j4284
308 void setFIFOmode(byte f_Mode)
309 {
310     if (f_Mode > 3) f_Mode = 3; // FIFO value cannot exceed 3.
311     f_Mode <<= 6; // Shift FIFO byte left 6 to put it in bits 6, 7.
312
313     byte tempSetting = IIC_Read(F_SETUP); //Read current settings
314     tempSetting &= ~(3<<6); // clear bits 6, 7
315     tempSetting |= f_Mode; //Mask in new FIFO bits
316     IIC_Write(F_SETUP, tempSetting);
317 }
318
319 //Call with a rate from 0 to 7. See page 33 for table of ratios.
320 //Sets the over sample rate. Datasheet calls for 128 but you can set it
321 //from 1 to 128 samples. The higher the oversample rate the greater
322 //the time between data samples.
323 void setOversampleRate(byte sampleRate)
324 {
325     if(sampleRate > 7) sampleRate = 7; //OS cannot be larger than 0b.0111
326     sampleRate <<= 3; //Align it for the CTRL_REG1 register
327
328     byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
329     tempSetting &= 0b11000111; //Clear out old OS bits
330     tempSetting |= sampleRate; //Mask in new OS bits
331     IIC_Write(CTRL_REG1, tempSetting);
332 }
333
334 //Clears then sets the OST bit which causes the sensor to immediately take another reading
335 //Needed to sample faster than 1Hz
336 void toggleOneShot(void)
337 {
338     byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
```

```
339     tempSetting &= ~(1<<1); //Clear OST bit
340     IIC_Write(CTRL_REG1, tempSetting);
341
342     tempSetting = IIC_Read(CTRL_REG1); //Read current settings to be safe
343     tempSetting |= (1<<1); //Set OST bit
344     IIC_Write(CTRL_REG1, tempSetting);
345 }
346
347 //Enables the pressure and temp measurement event flags so that we can
348 //test against them. This is recommended in datasheet during setup.
349 void enableEventFlags()
350 {
351     IIC_Write(PT_DATA_CFG, 0x07); // Enable all three pressure and temp event flags
352 }
353
354 // These are the two I2C functions in this sketch.
355 byte IIC_Read(byte regAddr)
356 {
357     // This function reads one byte over IIC
358     Wire.beginTransmission(MPL3115A2_ADDRESS);
359     Wire.write(regAddr); // Address of CTRL_REG1
360     Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not suppo
361     Wire.requestFrom(MPL3115A2_ADDRESS, 1); // Request the data...
362     return Wire.read();
363 }
364
365 void IIC_Write(byte regAddr, byte value)
366 {
367     // This function writes one byto over IIC
368     Wire.beginTransmission(MPL3115A2_ADDRESS);
369     Wire.write(regAddr);
370     Wire.write(value);
371     Wire.endTransmission(true);
372 }
```

