```
/*
Name:       LIS3DH.ino
Created:    3/2/2018 4:50 PM
Author: Michael Langford
*/

#include "LIS3DH.h"

#define STATUS_REG_AUX            0x07

#define WHO_AM_I                  0x0f

#define CTRL_REG1                 0x20
#define CTRL_REG2                 0x21
#define CTRL_REG3                 0x22
#define CTRL_REG4                 0x23
#define CTRL_REG5                 0x24
#define CTRL_REG6                 0x25

#define STATUS_REG_2              0x27

#define OUT_X_L                   0x28
#define OUT_X_H                   0x29
#define OUT_Y_L                   0x2a
#define OUT_Y_H                   0x2b
#define OUT_Z_L                   0x2c
#define OUT_Z_H                   0x2d

#define READ                      0b10111111
#define WRITE                     0b00111111

#define LIS3DH_CS_PIN             9

#define CLOCK_SPEED               4000000

#define LIS3DH_CALIBRATE_TIME 50

float x_acc_offset = 0.0f;
float y_acc_offset = 0.0f;
float z_acc_offset = 0.0f;

float acc_pitch_offset = 0.0f;
float acc_roll_offset = 0.0f;

float X_raw_out;
float Y_raw_out;
float Z_raw_out;

float acc_roll_angle;
float acc_pitch_angle;

#define BUFFERSIZE 20
```

```
53  float xdata[BUFFERSIZE];
54  float ydata[BUFFERSIZE];
55  float zdata[BUFFERSIZE];
56
57  int readAccRegister(byte address);
58  void writeAccRegister(byte address, byte data);
59
60  float Get_raw_X()
61  {
62      return X_raw_out;
63  }
64
65  float Get_raw_Y()
66  {
67      return Y_raw_out;
68  }
69
70  float Get_raw_Z()
71  {
72      return Z_raw_out;
73  }
74
75  float Get_Acc_Pitch()
76  {
77      return acc_pitch_angle;
78  }
79
80  float Get_Acc_Roll()
81  {
82      return acc_roll_angle;
83  }
84
85  bool Calibrating = false;
86
87  void update_LIS3DH()
88  {
89      if (Calibrating)
90      {
91          X_raw_out = ((float)((int16_t)((readAccRegister(OUT_X_H) << 8) |
              (readAccRegister(OUT_X_L)))) / 100.0f) - x_acc_offset;
92          Y_raw_out = ((float)((int16_t)((readAccRegister(OUT_Y_H) << 8) |
              (readAccRegister(OUT_Y_L)))) / 100.0f) - y_acc_offset;
93          Z_raw_out = ((float)((int16_t)((readAccRegister(OUT_Z_H) << 8) |
              (readAccRegister(OUT_Z_L)))) / 100.0f) - z_acc_offset;
94      }
95      else
96      {
97          X_raw_out += ((((float)((int16_t)((readAccRegister(OUT_X_H) << 8) |
              (readAccRegister(OUT_X_L)))) / 100.0f) - x_acc_offset) - X_raw_out)
              *0.02f;
98          Y_raw_out += ((((float)((int16_t)((readAccRegister(OUT_Y_H) << 8) |
              (readAccRegister(OUT_Y_L)))) / 100.0f) - y_acc_offset) - Y_raw_out)
```

```
            *0.02f;
 99         Z_raw_out += ((((float)((int16_t)((readAccRegister(OUT_Z_H) << 8) |
              (readAccRegister(OUT_Z_L)))) / 100.0f) - z_acc_offset) - Z_raw_out)
              *0.02f;
100
101         /*for (int i = 0; i < BUFFERSIZE - 1; i++)
102         {
103             xdata[i] = xdata[i + 1];
104             ydata[i] = ydata[i + 1];
105             zdata[i] = zdata[i + 1];
106         }
107
108         xdata[BUFFERSIZE - 1] = ((float)((int16_t)((readAccRegister(OUT_X_H) <<
              8) | (readAccRegister(OUT_X_L)))) / 100.0f) - x_acc_offset;
109         ydata[BUFFERSIZE - 1] = ((float)((int16_t)((readAccRegister(OUT_Y_H) <<
              8) | (readAccRegister(OUT_Y_L)))) / 100.0f) - x_acc_offset;
110         zdata[BUFFERSIZE - 1] = ((float)((int16_t)((readAccRegister(OUT_Z_H) <<
              8) | (readAccRegister(OUT_Z_L)))) / 100.0f) - x_acc_offset;
111
112         float xa, ya, za;
113         xa = ya = za = 0.0f;
114
115         for (int i = 0; i < BUFFERSIZE; i++)
116         {
117             xa += xdata[i];
118             ya += ydata[i];
119             za += zdata[i];
120         }
121
122         X_raw_out = xa / (float)BUFFERSIZE;
123         Y_raw_out = ya / (float)BUFFERSIZE;
124         Z_raw_out = za / (float)BUFFERSIZE;*/
125     }
126
127     acc_pitch_angle += (degrees(atan2f(Y_raw_out, -Z_raw_out)) - acc_pitch_angle)
          * 1.0f;
128     acc_roll_angle += (degrees(atan2f(X_raw_out, -Z_raw_out)) - acc_roll_angle) *
          1.0f;
129 }
130
131 void Calibrate_Accelerometer()
132 {
133     Calibrating = true;
134
135     float cx, cy, cz;
136     cx = cy = cz = 0.0f;
137
138     //float croll = 0.0f, cpitch = 0.0f;
139
140     x_acc_offset = y_acc_offset = z_acc_offset = 0.0f;
141
142     for (int i = 0; i < LIS3DH_CALIBRATE_TIME; i++)
```

```
143     {
144         delay(1);
145 
146         update_LIS3DH();
147 
148         cx += X_raw_out;
149         cy += Y_raw_out;
150         cz += Z_raw_out + 163.84f;
151     }
152 
153     x_acc_offset = cx / LIS3DH_CALIBRATE_TIME;
154     y_acc_offset = cy / LIS3DH_CALIBRATE_TIME;
155     z_acc_offset = cz / LIS3DH_CALIBRATE_TIME;
156 
157     /*for (int i = 0; i < LIS3DH_CALIBRATE_TIME; i++)
158     {
159         delay(1);
160         update_LIS3DH();
161 
162         cpitch += degrees(atan2f(-X_raw_out, -Z_raw_out));
163         croll += degrees(atan2f(-Y_raw_out, -Z_raw_out));
164     }
165 
166     acc_pitch_offset = cpitch / LIS3DH_CALIBRATE_TIME;
167     acc_roll_offset = croll / LIS3DH_CALIBRATE_TIME;*/
168 
169     Calibrating = false;
170 }
171 
172 void init_LIS3DH() {
173     SPI.begin();
174 
175     pinMode(LIS3DH_CS_PIN, OUTPUT);
176     digitalWrite(LIS3DH_CS_PIN, HIGH);
177 
178     delay(100);
179 
180     //set up initial state, disable pwr down, set data rate
181     writeAccRegister(CTRL_REG1, 0b10010111);
182 
183     x_acc_offset = 0.0f;
184     y_acc_offset = 0.0f;
185     z_acc_offset = 0.0f;
186 
187     acc_pitch_offset = 0.0f;
188     acc_roll_offset = 0.0f;
189 
190     X_raw_out = 0.0f;
191     Y_raw_out = 0.0f;
192     Z_raw_out = 0.0f;
193 
194     acc_roll_angle = 0.0f;
```

```
195     acc_pitch_angle = 0.0f;
196
197     delay(100);
198
199     Calibrate_Accelerometer();
200 }
201
202 int readAccRegister(byte address)
203 {
204     SPI.beginTransaction(SPISettings(CLOCK_SPEED, MSBFIRST, SPI_MODE0));
205
206     int toRead;
207
208     address |= 0x80;
209
210     digitalWrite(LIS3DH_CS_PIN, LOW);
211     SPI.transfer(address);
212     toRead = SPI.transfer(0);
213     digitalWrite(LIS3DH_CS_PIN, HIGH);
214     SPI.endTransaction();
215     return toRead;
216 }
217
218 void writeAccRegister(byte address, byte data)
219 {
220     SPI.beginTransaction(SPISettings(CLOCK_SPEED, MSBFIRST, SPI_MODE0));
221
222     address &= 0x7F;
223
224     digitalWrite(LIS3DH_CS_PIN, LOW);
225     SPI.transfer(address);
226     SPI.transfer(data);
227     digitalWrite(LIS3DH_CS_PIN, HIGH);
228
229     SPI.endTransaction();
230 }
```