

Introduction to Embedded Systems Design and Implementation HW4

學號: 0711282 姓名: 邱頎霖

Discussion 1

- a. Based on the requirements, set correct value in the sample code requirements:

8 samples per measurement

Data Output Rate = 15Hz

Gain=1090(LSb/Gauss)

Convert LSB to Gauss (by using self.scale)

- b. Continuously measurement (infinite loop)

- c. Calibrate your sensor

a.

| CRB7 | CRB6 | CRB5 | CRB4 | CRB3 | CRB2 | CRB1 | CRB0 |
|---------|---------|---------|------|------|------|------|------|
| GN2 (0) | GN1 (0) | GN0 (1) | (0) | (0) | (0) | (0) | (0) |

Table 7: Configuration B Register

Data Output Rate = 15Hz 因此利用 datasheet 中的格式對應是 {CRA4 CRA3 CRA2} = 100

8 samples per measurement 因此設定 {CR6 CR5} = 11

因此整體 configuration register A 8 bit 為 0b01110000

| CRA7 | CRA6 | CRA5 | CRA4 | CRA3 | CRA2 | CRA1 | CRA0 |
|------|--------|--------|---------|---------|---------|---------|---------|
| (0) | MA1(0) | MA0(0) | DO2 (1) | DO1 (0) | DO0 (0) | MS1 (0) | MS0 (0) |

Table 3: Configuration Register A

Gain=1090(LSb/Gauss) 因此按照對應為 {CRA6 CRA5} = 0b001

因此整體 configuration register B 8 bit 為 0b00100000

Operating modes 的部份則設定為 Continuous-Measurement mode

因此整體 mode register 8 bit 為 0b00000000

```
self.scale = 0.92 # convert bit value(LSB) to gauss.
```

```
# Configuration Register A
self.write_byte(HMC5883L_CRA, 0b01110000)
```

```
# Configuration Register B
self.write_byte(HMC5883L_CRB, 0b00100000)
```

```
# Mode Register
```

```
self.write_byte(HMC5883L_MR, 0b00000000)
```

b. 多加 while 迴圈, 程式碼連結 (這裡有 import discussion2 的 file(barQ, 其內容寫在 discussion2), discussion 1 與 2 的主程式碼寫在同一個 file 裡面)

截圖:

```
Set as interpreter
1 #!/usr/bin/python3
2
3 import smbus
4 import time
5 from math import *
6 from gy import *
7 from barQ import *
8
9 bus = smbus.SMBus(1);           # 0 for R-Pi Rev. 1, 1 for Rev. 2
10
11 # General constants
12 EARTH_GRAVITY_MS2 = 9.80665 # m/s2
13
14 # the following address is defined by datasheet
15 #HMC5883L (Magnetometer) constants
16 HMC5883L_ADDRESS = 0x30 # I2C address
17
18 HMC5883L_CRA = 0x00 # write CRA(00), Configuration Register A
19 HMC5883L_CRB = 0x01 # write CRB(01), Configuration Register B
20 HMC5883L_MR = 0x02 # write Mode(02)
21 HMC5883L_DO_X_H = 0x03 # Data Output
22 HMC5883L_DO_X_L = 0x04
23 HMC5883L_DO_Z_H = 0x05
24 HMC5883L_DO_Z_L = 0x06
25 HMC5883L_DO_Y_H = 0x07
26 HMC5883L_DO_Y_L = 0x08
27
28
29 # the following address is defined by datasheet
30 ADXL345_ADDRESS = 0x53 # I2C address
31
32 ADXL345_BW_RATE = 0x2C # Data rate and power mode control
33 ADXL345_POWER_CTL = 0x2D # Power-saving features control
34 ADXL345_DATA_FORMAT = 0x31 # Data format control
35 ADXL345_DATAX0 = 0x32
36 ADXL345_DATAX1 = 0x33
37 ADXL345_DATAY0 = 0x34
38 ADXL345_DATAY1 = 0x35
39 ADXL345_DATAZ0 = 0x36
40 ADXL345_DATAZ1 = 0x37
41 #
42
43 # set value
44 ADXL345_SCALE_MULTIPLIER= 0.00390625 # G/LSP. 1/256 = 0.00390625
45 ADXL345_BW_RATE_100HZ = 0x0A          # 0A = 0000 1111
46 ADXL345_MEASURE = 0x08                # 08 = 0000 1000
47
48
```

```
50 class IMU(object):
51
52     def write_byte(self,adr,value):
53         bus.write_byte_data(self.ADDRESS, adr, value)
54
55     def read_byte(self,adr):
56         return bus.read_byte_data(self.ADDRESS, adr)
57
58     def read_word(self,adr,rf=1):
59         # rf=1 Little Endian Format, rf=0 Big Endian Format
60         if (rf == 1):
61             # acc, gyro
62             low = self.read_byte(adr)
63             high = self.read_byte(adr+1)
64         else:
65             # compass
66             high = self.read_byte(adr)
67             low = self.read_byte(adr+1)
68         val = (high << 8) + low
69         return val
70
71     def read_word_2c(self,adr,rf=1):
72         val = self.read_word(adr,rf)
73         if(val & (1 << 16 - 1)):
74             return val - (1<<16)
75         else:
76             return val
77
78     class gy801(object):
79         def __init__(self) :
80             self.compass = HMC5883L()
81             self.accel = ADXL345()
82
```

```
85 # -.
86 class ADXL345(IMU):
87
88     ADDRESS = ADXL345_ADDRESS
89
90     def __init__(self):
91         #Class Properties
92         self.Xoffset = 0 # unit: G, modify by yourself
93         self.Yoffset = 0 # unit: G, modify by yourself
94         self.Zoffset = 0 # unit: G, modify by yourself
95         self.Xraw = 0.0
96         self.Yraw = 0.0
97         self.Zraw = 0.0
98         self.Xg = 0.0
99         self.Yg = 0.0
100        self.Zg = 0.0
101        self.X = 0.0
102        self.Y = 0.0
103        self.Z = 0.0
104        self.t0x = None
105        self.t0y = None
106        self.t0z = None
107
108        self.df_value = 0b00001000
109
110        self.Xcalibr = ADXL345_SCALE_MULTIPLIER
111        self.Ycalibr = ADXL345_SCALE_MULTIPLIER
112        self.Zcalibr = ADXL345_SCALE_MULTIPLIER
113
114        # Register 0x2C: BW_RATE
115        self.write_byte(ADXL345_BW_RATE, ADXL345_BW_RATE_100HZ)
116        # write value= 0xA = 00001111
117        # D3-D0: The default value is 0xA,
118        # which translates to a 100 Hz output data rate.
119
120
121        # Register 0x2D: POWER_CTL
122        self.write_byte(ADXL345_POWER_CTL, ADXL345_MEASURE)
123        # write value: 0x08 = 00001000
124        # D3=1: set 1 for measurement mode.
125
126
127        # Register 0x31: DATA_FORMAT
128        self.write_byte(ADXL345_DATA_FORMAT, self.df_value)
129        # write value=00001000
130        # D3 = 1: the device is in full resolution mode,
131        # where the output resolution increases with the g range
132        # set by the range bits to maintain a 4 mg/LSB scale factor.
133        # D1 D0 = range. 00 = +-2g
134
135
```

```
135     # RAW readings in LPS
136     # Register 0x32 to Register 0x37:
137     # DATAx0, DATAx1, DATAy0, DATAy1, DATAz0, DATAz1 (Read Only)
138
139     def getRawX(self) :
140         self.Xraw = self.read_word_2c(ADXL345_DATAx0)
141         return self.Xraw
142
143     def getRawY(self) :
144         self.Yraw = self.read_word_2c(ADXL345_DATAy0)
145         return self.Yraw
146
147     def getRawZ(self) :
148         self.Zraw = self.read_word_2c(ADXL345_DATAz0)
149         return self.Zraw
150
151     # G related readings in g
152     # similar to filter. combine current value with previous one.
153     # plf = 1 means it only uses "current reading"
154     def getXg(self,plf = 1.0) :
155         self.Xg = (self.getRawX() * self.Xcalibr - self.Xoffset) * plf + (1.0 - plf) * self.Xg
156         return self.Xg
157
158     def getYg(self,plf = 1.0) :
159         self.Yg = (self.getRawY() * self.Ycalibr - self.Yoffset) * plf + (1.0 - plf) * self.Yg
160         return self.Yg
161
162     def getZg(self,plf = 1.0) :
163         self.Zg = (self.getRawZ() * self.Zcalibr - self.Zoffset) * plf + (1.0 - plf) * self.Zg
164         return self.Zg
165
166     # Absolute reading in m/s2
167     def getX(self,plf = 1.0) :
168         self.X = self.getXg(plf) * EARTH_GRAVITY_MS2
169         return self.X
170
171     def getY(self,plf = 1.0) :
172         self.Y = self.getYg(plf) * EARTH_GRAVITY_MS2
173         return self.Y
174
175     def getZ(self,plf = 1.0) :
176         self.Z = self.getZg(plf) * EARTH_GRAVITY_MS2
177         return self.Z
178
179
```

```
182 class HMC5883L(IMU):
183     ADDRESS = HMC5883L_ADDRESS
184
185     def __init__(self):
186         #Class Properties
187         self.X = None
188         self.Y = None
189         self.Z = None
190         self.angle = None
191         self.Xoffset = 70.0
192         self.Yoffset = -157.5
193         self.Zoffset = 65.0
194
195         # Declination Angle
196         self.angle_offset = (-1 * (4 + (32/60))) / (180 / pi)
197         # Formula: (deg + (min * 60.0)) / (180 / M_PI);
198         # ex: Hsinchu = Magnetic Declination: -4 deg, 32 min
199         # declinationAngle = (-1 * (4 + (32/60))) / (180 / pi)
200         # http://www.magnetic-declination.com/
201
202         self.scale = 0.92 # convert bit value(LSB) to gauss. DigitalResolution
203
204         # Configuration Register A
205         self.write_byte(HMC5883L_CRA, 0b01110000)
206
207         # Configuration Register B
208         self.write_byte(HMC5883L_CRB, 0b00010000)
209
210         # Mode Register
211         self.write_byte(HMC5883L_MR, 0b00000000)
212
213     def getX(self):
214         self.X = (self.read_word_2c(HMC5883L_D0_X_H, rf=0) - self.Xoffset) * self.scale
215         return self.X
216
217     def getY(self):
218         self.Y = (self.read_word_2c(HMC5883L_D0_Y_H, rf=0) - self.Yoffset) * self.scale
219         return self.Y
220
221     def getZ(self):
222         self.Z = (self.read_word_2c(HMC5883L_D0_Z_H, rf=0) - self.Zoffset) * self.scale
223         return self.Z
224
225     def getHeading(self):
226         bearing = degrees(atan2(self.getY(), self.getX()))
227
228         if (bearing < 0):
229             bearing += 360
230         if (bearing > 360):
231             bearing -= 360
232         self.angle = bearing + self.angle_offset
233         return self.angle
234
```

```
237 pitcharr = []
238 rollarr = []
239 pitcharr.append(0)
240 pitcharr.append(0)
241 rollarr.append(0)
242 rollarr.append(0)
243
244 try:
245     sensors = gy801()
246     compass = sensors.compass
247     adxl345 = sensors.accel
248
249     # ##### LAB3 #####
250     sensor2 = gy801_2()
251     gyro = sensor2.gyro
252     # #####
253
254     sensors3 = gy801_3()
255     barometer = sensors3.baro
256
257 while True:
258     magx = compass.getX()
259     magy = compass.getY()
260     magz = compass.getZ()
261
262     # -----
263     # calculate pitch, roll, tilt
264     aX = adxl345.getX()
265     aY = adxl345.getY()
266     aZ = adxl345.getZ()
267
268     roll = atan2(aY,aZ)
269     pitch = -1 * atan2(-aX,sqrt(aY*aY+aZ*aZ))
270     # -----
271
272     # -----
273     # Heading
274     bearingl = degrees(atan2(magy, magx))
275
276     if (bearingl < 0):
277         bearingl += 360
278     if (bearingl > 360):
279         bearingl -= 360
280     bearingl = bearingl + compass.angle_offset
281
282     # Tilt compensate
283     compx = magx * cos(pitch) + magz * sin(pitch)
284     compy = magx * sin(roll) * sin(pitch) \
285             + magy * cos(roll) \
286             - magz * sin(roll) * cos(pitch)
```

```

273     # -----
274     # Heading
275     bearing1 = degrees(atan2(magy, magx))
276
277     if (bearing1 < 0):
278         bearing1 += 360
279     if (bearing1 > 360):
280         bearing1 -= 360
281     bearing1 = bearing1 + compass.angle_offset
282
283     # Tilt compensate
284     compx = magx * cos(pitch) + magz * sin(pitch)
285     compy = magx * sin(roll) * sin(pitch) \
286             + magy * cos(roll) \
287             - magz * sin(roll) * cos(pitch)
288
289     bearing2 = degrees(atan2(compy, compx))
290     if (bearing2 < 0):
291         bearing2 += 360
292     if (bearing2 > 360):
293         bearing2 -= 360
294     bearing2 = bearing2 + compass.angle_offset
295     #
296
297     # ##### LAB3 #####
298     tmpPitch = (pitcharr[0]+gyro.getXangle())*0.98+pitch*0.02
299     tmpRoll = (rollarr[0]+gyro.getYangle())*0.98+roll*0.02
300     pitcharr[0] = pitcharr[1]
301     pitcharr[1] = tmpPitch
302     rollarr[0] = rollarr[1]
303     rollarr[1] = tmpRoll
304     # print("pitch:", pitcharr[1], "roll:", rollarr[1])
305     # ##### LAB3 #####
306
307     # print ("Compass: " )
308     # print ("X = %d , " % ( magx )),
309     # print ("Y = %d , " % ( magy )),
310     # print ("Z = %d (gauss)" % ( magz ))
311     # print ("tiltX = %.3f , " % ( compx )),
312     # print ("tiltY = %.3f , " % ( compy )),
313
314     xh = magx*cos(pitcharr[1])+magz*sin(pitcharr[1])
315     yh = magx*sin(rollarr[1])*sin(pitcharr[1])+magy*cos(rollarr[1])-magz*sin(rollarr[1])*cos(pitcharr[1])
316
317     # print(xh, " ",yh)
318     print("Pitch: ",pitcharr[1], " Roll: ", rollarr[1])
319     print("Heading", atan2(yh,xh)*180/pi)
320
321     # print ("Angle offset = %.3f deg" % ( compass.angle_offset ))
322     print ("Original Heading = %.3f deg, " % ( bearing1 )),
323     print ("Tilt Heading = %.3f deg, " % ( bearing2 ))
324
325     tempC = barometer.getTempC()
326     tempF = barometer.getTempF()
327     press = barometer.getPress()
328     altitude = barometer.getAltitude()
329
330     # print("Barometer:")
331     # print("  Temp: %f C (%f F)" % (tempC, tempF))
332     # print("  Press: %f (hPa)" % (press))
333     print("  Altitude: %f m s.l.m" % (altitude))
334     print("")
335     time.sleep(1)
336
337 except KeyboardInterrupt:
338     print("Cleanup")
339

```

c. 校正

照著投影片的校正動作並同時跑 "3calibrate-hmc5883l.py" 之後會輸出 x, y, z offset 的數值, 我的結果為:

```
self.Xoffset = 70.0  
self.Yoffset = -157.5  
self.Zoffset = 65.0
```

Discussion 2

- 1. Based on the datasheet, set correct value in the sample code
 - Standard mode

| |
|--|
| read uncompensated pressure value |
| write 0x34+(oss<<6) into reg 0xF4, wait |
| read reg 0xF6 (MSB), 0xF7 (LSB), 0xF8 (XLSB) |
| UP = (MSB<<16 + LSB<<8 + XLSB) >> (8-oss) |

| |
|--------------------------------------|
| read uncompensated temperature value |
| write 0xE into reg 0xF4, wait 4.5ms |
| read reg 0xF6 (MSB), 0xF7 (LSB) |
| UT = MSB << 8 + LSB |

```
def getPress(self) :  
    # print ("Calculating temperature...")  
    self.write_byte(0xF4, 0x??)  
    time.sleep(0.005)
```

```
# read uncompensated temperature value  
def getTempC(self) :  
    # print ("Calculating temperature...")  
    self.write_byte(0xF4, 0x??)  
    time.sleep(0.005)
```

- 2. Continuously measurement(infinite loop)

1.

```
def getTempC(self) :  
    # print ("Calculating temperature...")  
    self.write_byte(0xF4, 0x2E)  
    time.sleep(0.005)
```

```
def getPress(self) :  
    # print ("Calculating temperature...")  
    self.write_byte(0xF4, 0x2E)  
    time.sleep(0.005)
```

2.

discussion 1 與 2 的主程式寫在同一個 file(在 discussion 1 的 b), 因此這裡僅附上 import 的 file(barQ) 內容:

```
1 import smbus
2 import time
3 from math import *
4
5 bus = smbus.SMBus(1);           # 0 for R-Pi Rev. 1, 1 for Rev. 2
6
7 STANDARD_PRESSURE    = 1013.25 # hPa
8
9 #BMP180 (Barometer) constants
10 BMP180_ADDRESS       = 0x77
11
12 # Calibration coefficients
13 BMP180_AC1            = 0xAA
14 BMP180_AC2            = 0xAC
15 BMP180_AC3            = 0xAE
16 BMP180_AC4            = 0xB0
17 BMP180_AC5            = 0xB2
18 BMP180_AC6            = 0xB4
19 BMP180_B1             = 0xB6
20 BMP180_B2             = 0xB8
21 BMP180_MB             = 0xBA
22 BMP180_MC             = 0xBC
23 BMP180_MD             = 0xBE
24
25 class IMU(object):
26
27     def write_byte(self,adr,value):
28         bus.write_byte_data(self.ADDRESS, adr, value)
29
30     def read_byte(self,adr):
31         return bus.read_byte_data(self.ADDRESS, adr)
32
33     def read_word(self,adr,rf=1):
34         # rf=1 Little Endian Format, rf=0 Big Endian Format
35         if (rf == 1):
36             low = self.read_byte(adr)
37             high = self.read_byte(adr+1)
38         else:
39             high = self.read_byte(adr)
40             low = self.read_byte(adr+1)
41         val = (high << 8) + low
42         return val
43
44     def read_word_2c(self,adr,rf=1):
45         val = self.read_word(adr,rf)
46         if(val & (1 << 16 - 1)):
47             return val - (1<<16)
48         else:
49             return val
50
51 class gy801_3(object):
52     def __init__(self) :
53         self.baro = BMP180()
54
55
56 class BMP180(IMU):
57     ADDRESS = BMP180_ADDRESS
```

```
51 class gy801_3(object):
52     def __init__(self) :
53         self.baro = BMP180()
54
55
56 class BMP180(IMU):
57
58     ADDRESS = BMP180_ADDRESS
59
60     def __init__(self) :
61         #Class Properties
62         self.tempC = None
63         self.tempF = None
64         self.press = None
65         self.altitude = None
66
67         self.oversampling = 0
68
69         self._read_calibratio_params()
70
71     # read calibration data
72     def _read_calibratio_params(self) :
73         self.ac1_val = self.read_word_2c(BMP180_AC1,0)
74         self.ac2_val = self.read_word_2c(BMP180_AC2,0)
75         self.ac3_val = self.read_word_2c(BMP180_AC3,0)
76         self.ac4_val = self.read_word(BMP180_AC4,0)
77         self.ac5_val = self.read_word(BMP180_AC5,0)
78         self.ac6_val = self.read_word(BMP180_AC6,0)
79         self.b1_val = self.read_word_2c(BMP180_B1,0)
80         self.b2_val = self.read_word_2c(BMP180_B2,0)
81         self.mc_val = self.read_word_2c(BMP180_MC,0)
82         self.md_val = self.read_word_2c(BMP180_MD,0)
83
84     # read uncompensated temperature value
85     def getTempC(self) :
86         # print ("Calculating temperature...")
87         self.write_byte(0xF4, 0x2E)
88         time.sleep(0.005)
89
90         ut = self.read_word(0xF6,0)
91
92         # calculate true temperature
93         x1 = ((ut - self.ac6_val) * self.ac5_val) >> 15
94         x2 = (self.mc_val << 11) // (x1 + self.md_val)
95         b5 = x1 + x2
96         self.tempC = ((b5 + 8) >> 4) / 10.0
97
98         return self.tempC
99
100    def getTempF(self) :
101        #print ("Calculating temperature (Fahrenheit)...")
102        self.tempF = self.getTempC() * 1.8 + 32
103
104        return self.tempF
105
```

```
100     def getTempF(self) :
101         #print ("Calculating temperature (Fahrenheit)...") 
102         self.tempF = self.getTempC() * 1.8 + 32
103
104         return self.tempF
105
106     # read uncompensated pressure value
107     def getPress(self) :
108         # print ("Calculating pressure...")
109         self.write_byte(0xF4, 0x2E)
110         time.sleep(0.005)
111
112         ut = self.read_word(0xF6,0)
113
114         x1 = ((ut - self.ac6_val) * self.ac5_val) >> 15
115         x2 = (self.mc_val << 11) // (x1 + self.md_val)
116         b5 = x1 + x2
117
118         #print ("Calculating pressure...")
119         self.write_byte(0xF4, 0x34 + (self.oversampling << 6))
120         time.sleep(0.04)
121
122         msb = self.read_byte(0xF6)
123         lsb = self.read_byte(0xF7)
124         xsb = self.read_byte(0xF8)
125
126         up = ((msb << 16) + (lsb << 8) + xsb) >> (8 - self.oversampling)
127
128         # calculate true pressure
129         b6 = b5 - 4000
130         b62 = b6 * b6 >> 12
131         x1 = (self.b2_val * b62) >> 11
132         x2 = self.ac2_val * b6 >> 11
133         x3 = x1 + x2
134         b3 = (((self.ac1_val * 4 + x3) << self.oversampling) + 2) >> 2
135
136         x1 = self.ac3_val * b6 >> 13
137         x2 = (self.b1_val * b62) >> 16
138         x3 = ((x1 + x2) + 2) >> 2
139         b4 = (self.ac4_val * (x3 + 32768)) >> 15
140         b7 = (up - b3) * (50000 >> self.oversampling)
141
142         press = (b7 * 2) // b4
143         #press = (b7 / b4) * 2
144
145         x1 = (press >> 8) * (press >> 8)
146         x1 = (x1 * 3038) >> 16
147         x2 = (-7357 * press) >> 16
148         self.press = ( press + ((x1 + x2 + 3791) >> 4) ) / 100.0
149
150         return self.press
151
```

```
153     def getAltitude(self) :
154         #    print ("Calculating altitude...")
155         self.altitude = 44330 * (1 - ((self.getPress() / STANDARD_PRESSURE) ** 0.1903))
156
157         return self.altitude
```