Introduction to Embedded Systems Design and Implementation HW3

學號: **0711282** 姓名: 邱頎霖

Discussion 1
a. Based on the requirements, set correct value in the sample code

requirements:
1. Output data rate: 100Hz
2. Range: +-2g
3. Convert LSB to G (by using SCALE_MULTIPLIER)

b. Continuously measurement (infinite loop)
c. Calibrate your sensor (see next page)(做校正)

根據 sample code 可以知道已經事先寫好 write byte 等動作, 我們只需要將值正確填入變數就可以寫到對應的 address 之中, requirements 所要求的內容與相應的變數為:

Output data rate : ADXL345_BW_RATE_100HZ
Range : self.df_value
Convert LSB to G :

```
# Register 0x2C: BW_RATE
self.write_byte(ADXL345_BW_RATE, ADXL345_BW_RATE_100HZ)
# write value= 0x0A = 0000 1010
# D3-D0: The default value is 0x0A,
# which translates to a 100 Hz output data rate.


# Register 0x2D: POWER_CTL
self.write_byte(ADXL345_POWER_CTL, ADXL345_MEASURE)
# write value: 0x08 = 0000 1000
# D3=1: set 1 for measurement mode.


# Register 0x31: DATA_FORMAT
self.write_byte(ADXL345_DATA_FORMAT, self.df_value)
# write value=0000 1000
# D3 = 1: the device is in full resolution mode,
# where the output resolution increases with the g range
# set by the range bits to maintain a 4 mg/LSB scale factor.
# D1 D0 = range. 00 = +-2g
```

基本上就是根據 data sheet 查如何設定, 根據講義中所附贈的 datasheet (下圖一二三), 可以知道:

output data rate : 100hz 的 rate code 為 **1010**,

range +- 2g : 對應值為 b00 因為沒有要求 FULL_RES Bit 設定, 在此假設為 1, 因此需要將 **self.df_value** 設定為 **0b00001000**

convert LSB to G: 1/256 = 0.00390625

Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

D3D2D1D0 = 1010

LOW_POWER Bit

A setting of 0 in the LOW_POWER bit selects normal operation, and a setting of 1 selects reduced power operation, which has somewhat higher noise (see the Power Modes section for details).

Rate Bits

These bits select the device bandwidth and output data rate (see Table 7 and Table 8 for details). The default value is 0x0A, which translates to a 100 Hz output data rate. An output data rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an output data rate with a low communication speed results in samples being discarded.

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
3200	1600	1111	140
1600	800	1110	90
800	400	1101	140
400	200	1100	140
200	100	1011	140
100	50	1010	140
50	25	1001	90
25	12.5	1000	60

(圖一)

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

D3 = 1

D1D0 = 00

FULL_RES Bit

When this bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the *g* range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum *g* range and scale factor.

Table 21. *g* Range Setting

Setting		
D1	D0	*g* Range
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

27

(圖二)

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*

```
ADXL345_SCALE_MULTIPLIER= 0.00390625     # G/LSB. 1/256 = 0.00390625
self.Xcalibr = ADXL345_SCALE_MULTIPLIER
```

SCALE_MULTIPLIER = 1/SENSITIVITY

(圖三)

助教在 spec 之中挖了以下三個空要填入:

```
ADXL345_MEASURE = 0x08
ADXL345_SCALE_MULTIPLIER = 0.00390625     # G/LSP
ADXL345_BW_RATE_100HZ = 0x0A
```

(ADXL345_MEASURE 在 requirements 中並無特別規定要怎麼設定, 因此我自行假設為 measure mode 非 standby mode 了)

以下為程式碼截圖, 基本是從 quiz1,2 改過來的, 而我 quiz1,2 都寫在一起因此有點長, 為了助教的眼睛著想可以考慮到 Github 上看 😀 兩邊內容應該是一樣的

LAB3 > src > discussion1.py > ...

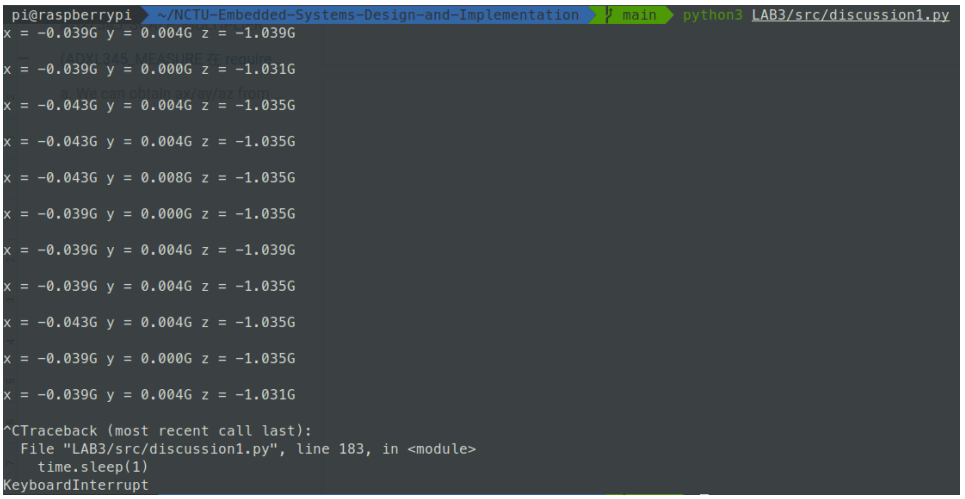
```python
import smbus
import time
from math import *
from gy import *

bus = smbus.SMBus(1)            # 0 for R-Pi Rev. 1, 1 for Rev. 2

EARTH_GRAVITY_MS2 = 9.80665  # m/s2

ADXL345_ADDRESS = 0x53

ADXL345_BW_RATE = 0x2C
ADXL345_POWER_CTL = 0x2D
ADXL345_DATA_FORMAT = 0x31
ADXL345_DATAX0 = 0x32
ADXL345_DATAY0 = 0x34
ADXL345_DATAZ0 = 0x36
ADXL345_SCALE_MULTIPLIER = 0.00390625    # G/LSP
ADXL345_BW_RATE_100HZ = 0x0A
ADXL345_MEASURE = 0x08


class IMU(object):

    def write_byte(self, adr, value):
        bus.write_byte_data(self.ADDRESS, adr, value)

    def read_byte(self, adr):
        return bus.read_byte_data(self.ADDRESS, adr)

    def read_word(self, adr, rf=1):
        # rf=1 Little Endian Format, rf=0 Big Endian Format
        if (rf == 1):
            low = self.read_byte(adr)
            high = self.read_byte(adr+1)
        else:
            high = self.read_byte(adr)
            low = self.read_byte(adr+1)
        val = (high << 8) + low
        return val

    def read_word_2c(self, adr, rf=1):
        val = self.read_word(adr, rf)
        if(val & (1 << 16 - 1)):
            return val - (1 << 16)
        else:
            return val

```

```
49
50  class gy801(object):
51      def __init__(self):
52          self.accel = ADXL345()
53
54
55  class ADXL345(IMU):
56
57      ADDRESS = ADXL345_ADDRESS
58
59      def __init__(self):
60          #Class Properties
61          self.Xoffset = 0.0
62          self.Yoffset = 0.0
63          self.Zoffset = 0.0
64          self.Xraw = 0.0
65          self.Yraw = 0.0
66          self.Zraw = 0.0
67          self.Xg = 0.0
68          self.Yg = 0.0
69          self.Zg = 0.0
70          self.X = 0.0
71          self.Y = 0.0
72          self.Z = 0.0
73          self.df_value = 0b00001000    # Self test disabled, 4-wire interface
74          # Full resolution, Range = +/-2g
75          self.Xcalibr = ADXL345_SCALE_MULTIPLIER
76          self.Ycalibr = ADXL345_SCALE_MULTIPLIER
77          self.Zcalibr = ADXL345_SCALE_MULTIPLIER
78
79          # Normal mode, Output data rate = 100 Hz
80          self.write_byte(ADXL345_BW_RATE, ADXL345_BW_RATE_100HZ)
81          # Auto Sleep disable
82          self.write_byte(ADXL345_POWER_CTL, ADXL345_MEASURE)
83          self.write_byte(ADXL345_DATA_FORMAT, self.df_value)
84
85      # RAW readings in LPS
86      def getRawX(self):
87          self.Xraw = self.read_word_2c(ADXL345_DATAX0)
88          return self.Xraw
89
90      def getRawY(self):
91          self.Yraw = self.read_word_2c(ADXL345_DATAY0)
92          return self.Yraw
93
94      def getRawZ(self):
95          self.Zraw = self.read_word_2c(ADXL345_DATAZ0)
```

```
    def getRawZ(self):
        self.Zraw = self.read_word_2c(ADXL345_DATAZ0)
        return self.Zraw

    # G related readings in g
    def getXg(self, plf=1.0):
        self.Xg = (self.getRawX() * self.Xcalibr + self.Xoffset) * \
            plf + (1.0 - plf) * self.Xg
        return self.Xg

    def getYg(self, plf=1.0):
        self.Yg = (self.getRawY() * self.Ycalibr + self.Yoffset) * \
            plf + (1.0 - plf) * self.Yg
        return self.Yg

    def getZg(self, plf=1.0):
        self.Zg = (self.getRawZ() * self.Zcalibr + self.Zoffset) * \
            plf + (1.0 - plf) * self.Zg
        return self.Zg

    # Absolute reading in m/s2
    def getX(self, plf=1.0):
        self.X = self.getXg(plf) * EARTH_GRAVITY_MS2
        return self.X

    def getY(self, plf=1.0):
        self.Y = self.getYg(plf) * EARTH_GRAVITY_MS2
        return self.Y

    def getZ(self, plf=1.0):
        self.Z = self.getZg(plf) * EARTH_GRAVITY_MS2
        return self.Z

    # write your code
    def getPitch(self):
        aX = self.getXg()
        aY = self.getYg()
        aZ = self.getZg()
        self.pitch = atan2(-aX, sqrt(aY*aY+aZ*aZ))
        return self.pitch

    def getRoll(self):
        aX = self.getXg()
        aY = self.getYg()
        aZ = self.getZg()
        self.roll = atan2(aY, aZ)
        return self.roll
```

```
141
142
143 if __name__ == "__main__":
144     pitch = []
145     roll = []
146     pitch.append(0)
147     pitch.append(0)
148     roll.append(0)
149     roll.append(0)
150     while(1):
151         try:
152             sensors = gy801()
153             adxl345 = sensors.accel
154             sensor2 = gy801_2()
155             gyro = sensor2.gyro
156
157             adxl345.getX()
158             adxl345.getY()
159             adxl345.getZ()
160
161             print("x = %.3fG y = %.3fG z = %.3fG" %
162                   (adxl345.Xg, adxl345.Yg, adxl345.Zg))
163
164             # print("x = %.3fG" % ( adxl345.Xg ))
165             # print("y = %.3fG" % ( adxl345.Yg ))
166             # print("z = %.3fG" % ( adxl345.Zg ))
167             # print("x = %.3f" % (adxl345.Xraw))
168             # print("y = %.3f" % (adxl345.Yraw))
169             # print("z = %.3f" % (adxl345.Zraw))
170
171             tmpPitch = (pitch[0]+gyro.getXangle())*0.98+adxl345.getPitch()*0.02
172             tmpRoll = (roll[0]+gyro.getYangle())*0.98+adxl345.getRoll()*0.02
173             pitch[0] = pitch[1]
174             pitch[1] = tmpPitch
175             roll[0] = roll[1]
176             roll[1] = tmpRoll
177
178
179             print("")
180
181         except KeyboardInterrupt:
182             print("Cleanup")
183         time.sleep(1)
184
```

理論尚在靜止時 x,y,z 分別應該為 0,0,1, 但實際上可以發現有誤差存在, 根據取樣大概可以抓誤差值放入 offset 中修正誤差:



修正誤差, 注意在修正過程中是 "加" 修正值(如下), 檔案於 Github 連結 裡, 基本上與上面不同的只有下圖修正設定中的三行

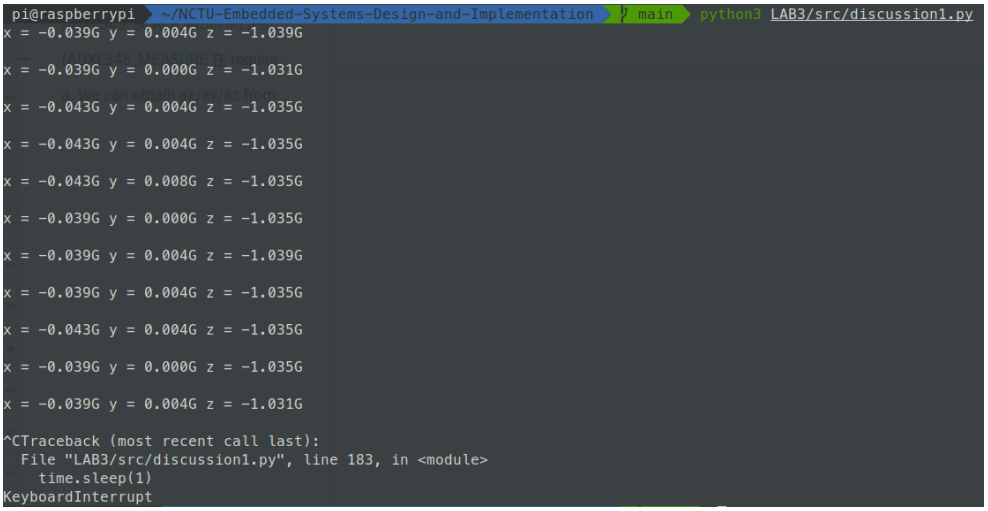
```
self.Xg = (self.getRawX() * self.Xcalibr + self.Xoffset) * \
          plf + (1.0 - plf) * self.Xg
```

(修正過程)

```
        self.Xoffset = 0.039
        self.Yoffset = -0.004
        self.Zoffset = 1.035
```

(修正設定)

修正結果已比當初發生較少錯誤:



Discussion 2

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

requirement:
1. Data rate: 100Hz, cut-off = 12.5
2. Full Scale selection = 250 dps
3. Set Sensitivity for 250 dps (by using SCALE_MULTIPLIER)

要填入的內容:

```
self.write_byte(L3G4200D_CTRL_REG1, 0x0F)
self.write_byte(L3G4200D_CTRL_REG4, 0x80)
self.gain_std = 0.00875     # dps/digit
```

其中 REG1 跟 REG4 就根據 data sheet(如下) 與要求填入即可:
b00001111 -> 0x0f
b10000000 -> 0x80

CTRL_REG1 (20h)

Table 20. CTRL_REG1 register

DR1	DR0	BW1	BW0	PD	Zen	Yen	Xen

Table 21. CTRL_REG1 description

DR1-DR0	Output Data Rate selection. Refer to *Table 22*
BW1-BW0	Bandwidth selection. Refer to *Table 22*
PD	Power down mode enable. Default value: 0 (0: power down mode, 1: normal mode or sleep mode)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled; 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled; 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled; 1: X axis enabled)

CTRL_REG4 (23h)

Table 30. CTRL_REG4 register

BDU	BLE	FS1	FS0	-	ST1	ST0	SIM

Table 31. CTRL_REG4 description

BDU	Block Data Update. Default value: 0 (0: continous update; 1: output registers not updated until MSB and LSB reading)
BLE	Big/Little Endian Data Selection. Default value 0. (0: Data LSB @ lower address; 1: Data MSB @ lower address)
FS1-FS0	Full Scale selection. Default value: 00 (00: 250 dps; 01: 500 dps; 10: 2000 dps; 11: 2000 dps)
ST1-ST0	Self Test Enable. Default value: 00 (00: Self Test Disabled; Other: See *Table*)
SIM	SPI Serial Interface Mode selection. Default value: 0 (0: 4-wire interface; 1: 3-wire interface).

gain_std 的部份是到 data sheet 裡頭挖, 換算 8.75 mdps/digit = 0.00875 dps/digit

				±2000		
So	Sensitivity	FS = 250 dps		8.75		mdps/digit
		FS = 500 dps		17.50		
		FS = 2000 dps		70		
	Sensitivity change vs					

b. Continuously measurement (infinite loop)

下頁開始為程式碼截圖, [Github 連結]

```python
import smbus
import time
from math import *

bus = smbus.SMBus(1)            # 0 for R-Pi Rev. 1, 1 for Rev. 2

L3G4200D_ADDRESS = 0x69
L3G4200D_CTRL_REG1 = 0x20
L3G4200D_CTRL_REG4 = 0x23
L3G4200D_OUT_X_L = 0x28
L3G4200D_OUT_X_H = 0x29
L3G4200D_OUT_Y_L = 0x2A
L3G4200D_OUT_Y_H = 0x2B
L3G4200D_OUT_Z_L = 0x2C
L3G4200D_OUT_Z_H = 0x2D


class IMU(object):

    def write_byte(self, adr, value):
        bus.write_byte_data(self.ADDRESS, adr, value)

    def read_byte(self, adr):
        return bus.read_byte_data(self.ADDRESS, adr)

    def read_word(self, adr, rf=1):
        # rf=1 Little Endian Format, rf=0 Big Endian Format
        if (rf == 1):
            low = self.read_byte(adr)
            high = self.read_byte(adr+1)
        else:
            high = self.read_byte(adr)
            low = self.read_byte(adr+1)
        val = (high << 8) + low
        return val

    def read_word_2c(self, adr, rf=1):
        val = self.read_word(adr, rf)
        if(val & (1 << 16 - 1)):
            return val - (1 << 16)
        else:
            return val


class gy801_2(object):
    def __init__(self):
        self.gyro = L3G4200D()

```

```
49
50  class L3G4200D(IMU):
51
52      ADDRESS = L3G4200D_ADDRESS
53
54      def __init__(self):
55          #Class Properties
56          self.Xraw = 0.0
57          self.Yraw = 0.0
58          self.Zraw = 0.0
59          self.X = 0.0
60          self.Y = 0.0
61          self.Z = 0.0
62          self.Xangle = 0.0
63          self.Yangle = 0.0
64          self.Zangle = 0.0
65          self.t0x = None
66          self.t0y = None
67          self.t0z = None
68
69          # set value
70          self.gain_std = 0.00875     # dps/digit
71
72          self.write_byte(L3G4200D_CTRL_REG1, 0x0F)
73          self.write_byte(L3G4200D_CTRL_REG4, 0x80)
74
75          self.setCalibration()
76
77      def setCalibration(self):
78          gyr_r = self.read_byte(L3G4200D_CTRL_REG4)
79
80          self.gain = 2 ** (gyr_r & 48 >> 4) * self.gain_std
81
82      def getRawX(self):
83          self.Xraw = self.read_word_2c(L3G4200D_OUT_X_L)
84          return self.Xraw
85
86      def getRawY(self):
87          self.Yraw = self.read_word_2c(L3G4200D_OUT_Y_L)
88          return self.Yraw
89
90      def getRawZ(self):
91          self.Zraw = self.read_word_2c(L3G4200D_OUT_Z_L)
92          return self.Zraw
93
94      def getX(self, plf=1.0):
95          self.X = (self.getRawX() * self.gain) * plf + (1.0 - plf) * self.X
```

```
    def getX(self, plf=1.0):
        self.X = (self.getRawX() * self.gain) * plf + (1.0 - plf) * self.X
        return self.X

    def getY(self, plf=1.0):
        self.Y = (self.getRawY() * self.gain) * plf + (1.0 - plf) * self.Y
        return self.Y

    def getZ(self, plf=1.0):
        self.Z = (self.getRawZ() * self.gain) * plf + (1.0 - plf) * self.Z
        return self.Z

    def getXangle(self, plf=1.0):
        if self.t0x is None:
            self.t0x = time.time()
        t1x = time.time()
        LP = t1x - self.t0x
        self.t0x = t1x
        self.Xangle = self.getX(plf) * LP
        return self.Xangle

    def getYangle(self, plf=1.0):
        if self.t0y is None:
            self.t0y = time.time()
        t1y = time.time()
        LP = t1y - self.t0y
        self.t0y = t1y
        self.Yangle = self.getY(plf) * LP
        return self.Yangle

    def getZangle(self, plf=1.0):
        if self.t0z is None:
            self.t0z = time.time()
        t1z = time.time()
        LP = t1z - self.t0z
        self.t0z = t1z
        self.Zangle = self.getZ(plf) * LP
        return self.Zangle
```

```
132
133
134 if __name__ == "__main__":
135
136     try:
137         # if run directly we'll just create an instance of the class and output
138         # the current readings
139
140         sensors = gy801_2()
141
142         gyro = sensors.gyro
143
144         gyro.getXangle()
145         gyro.getYangle()
146         gyro.getZangle()
147
148         print("Gyro: ")
149         print("Xangle = %.3f deg" % (gyro.getXangle()))
150         print("Yangle = %.3f deg" % (gyro.getYangle()))
151         print("Zangle = %.3f deg" % (gyro.getZangle()))
152
153     except KeyboardInterrupt:
154         print("Cleanup")
155
```

Discussion 3

a. We can obtain ax/ay/az from accelerometer. How to calculate the distance?

經過詢問助教可以假設為等加速度運動後, 可以使用等加速度運動公式推導:

$v(t) = v_0 + at$ (若初始速度為 0, 可以簡化為 $v(t) = at$) ...(1)

$v(t)^2 = v_0^2 + 2aS$ S 為所求的 distnace ...(2)

將 (1) 代入 (2) 可得 $(v_0 + at)^2 = v_0^2 + 2aS$ 左右代換後得 $S = ((v_0 + at)^2 - v_0^2) / (2a) = (2atv_0+a^2t^2)/(2*a)$

此處的 a 利用 quiz 1 中的 norm of acceleration = sqrt(ax*ax+ay*ay+az*az) 代入

且在程式中固定 sleep(1), 這裡將 t 直接代 1 方便計算: $(2av_0+a^2)/(2*a)$

並且持續將 v(t) 更新到 v(0) 中:

[程式碼完整連結]

```
142
143 if __name__ == "__main__":
144     v0 = 0.0
145     vt = 0.0
146     while(1):
147         try:
148             sensors = gy801()
149             adxl345 = sensors.accel
150             sensor2 = gy801_2()
151             gyro = sensor2.gyro
152
153             adxl345.getX()
154             adxl345.getY()
155             adxl345.getZ()
156
157             normacc = sqrt(adxl345.X*adxl345.X + adxl345.Y*adxl345.Y + adxl345.Z*adxl345.Z)
158
159             print("distance is :", (normacc*2*v0+normacc*normacc)/(2*normacc), " norm: ",normacc)
160
161             vt = v0 + normacc
162             v0 = vt
163
164
165         except KeyboardInterrupt:
166             print("Cleanup")
167         time.sleep(1)
168
```