
HYCON 紘康科技

Force Touch (Single Input)

應用電路說明書

HY16F184

Force Touch (Single Input) Circuit

Table of Contents

1.	內容簡介	4
2.	原理說明	5
2.1.	量測原理	5
2.2.	控制晶片	6
3.	系統設計	9
3.1.	硬體說明	9
3.2.	軟體說明	12
4.	數據規格與總結	13
4.1.	ADC Output Rate測量	13
4.2.	耗電流測量	13
4.3.	最大承受力	13
4.4.	ADC Raw Data與I2C通訊格式說明	13
4.5.	ADC Raw Data資料顯示介面介紹	14
4.6.	ADC Raw Data分辨率測試	14
4.7.	總結	16
5.	DEMO CODE及相關檔案	17
6.	參考文獻	37
7.	修訂記錄	38

注意：

- 1、本說明書中的內容，隨著產品的改進，有可能不經過預告而更改。請客戶及時到本公司網站下載更新 <http://www.hycontek.com>。
- 2、本規格書中的圖形、應用電路等，因第三方工業所有權引發的問題，本公司不承擔其責任。
- 3、本產品在單獨應用的情況下，本公司保證它的性能、典型應用和功能符合說明書中的條件。當使用在客戶的產品或設備中，以上條件我們不作保證，建議客戶做充分的評估和測試。
- 4、請注意輸入電壓、輸出電壓、負載電流的使用條件，使 IC 內的功耗不超過封裝的容許功耗。對於客戶在超出說明書中規定額定值使用產品，即使是瞬間的使用，由此所造成的損失，本公司不承擔任何責任。
- 5、本產品雖內置防靜電保護電路，但請不要施加超過保護電路性能的過大靜電。
- 6、本規格書中的產品，未經書面許可，不可使用在要求高可靠性的電路中。例如健康醫療器械、防災器械、車輛器械、車載器械及航空器械等對人體產生影響的器械或裝置，不得作為其部件使用。
- 7、本公司一直致力於提高產品的品質和可靠度，但所有的半導體產品都有一定的失效概率，這些失效概率可能會導致一些人身事故、火災事故等。當設計產品時，請充分留意冗餘設計並採用安全指標，這樣可以避免事故的發生。
- 8、本規格書中內容，未經本公司許可，嚴禁用於其他目的之轉載或複製。

1. 內容簡介

在 2015 年，蘋果新一代的 MacBook 和 Apple Watch 皆搭載壓力觸控感應技術，它被 Apple 稱為 Force Touch，用戶每次按下觸控板之後除了可以在螢幕看見視覺回饋，它同時能夠分辨出用戶點按的力度強弱來做出一系列的相關操控與應用。而本文將介紹以 HY16F184 內建高精密 Sigma-delta 24 Bit ADC 搭配 Uneo Force Sensor 來實現一個類似 Force Touch 應用電路。在本文中的 Force Touch 應用電路上，主要的元件有：壓力感測器(Force Sensor)、ADC 和 MCU 控制晶片。紘康 HY16F184 控制晶片內建高精密 Sigma-delta 24 Bit ADC、可程式放大 PGA 和多段式穩壓輸出等功能，可以很大幅簡化 PCB 周邊線路，精準完成由類比到數位的訊號轉換。

在一個完整 Force Touch 實際應用上，除了需要考量到 X, Y 軸座標與 Z 軸強度計算，同時也需要考量到電流消耗功率與掃描速度和校正的應用設計。而在本文內，只先探討與介紹如何使用 HY16F184 內建高精密 Sigma-delta 24 Bit ADC 來掃描四個 Force Sensor 並透過 I2C 通訊來輸出基本的 ADC Raw Data。使用 I2C 轉 USB 橋接器與電腦連接，由電腦端 GUI 做即時四個通道的 ADC Raw Data 資料變化量顯示。

2. 原理說明

2.1. 量測原理

Uneo Force Sensor 是以壓阻效應(piezoresistive effects)為基礎的壓力感測器，所謂壓阻效應是指當感測器內部感測元件受到外界壓力作用於其上時，其電阻值會隨之改變的現象。使用者可以依照電阻值的變化量，使其輸出電壓隨之改變，來推算出壓力大小。在本文中，是使用 Uneo Force Sensor 來做單端輸入(Single Input)的訊號量測，其量測方法與原理，可以參考下圖 1。R1 為一個固定的參考電阻，Vr 則為固定的參考電壓，Rs 即為 Uneo Force Sensor 電阻值變化量，當外界壓力作用於其上時，Rs 電阻值則會產生變化，V2 則為 Uneo Force Sensor 相對應的電壓變化量。其公式為： $V2 = +Vr * R1 / (Rs + R1)$ 。Uneo Force Sensor 的特性為當元件受到外界壓力越大時，其電導(Conductance)值會越大，換而言之即是電阻(Resistance)值會越小。所以由以上公式配合 Uneo Force Sensor 特性可以得知，當 V2 電壓越大時候代表所感受到的 Force 壓力也越大。

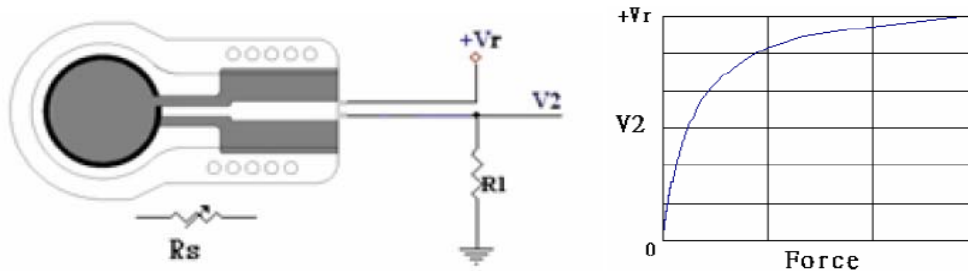


圖 1 Uneo Force Sensor 量測原理

本文 Force Touch 的基本架構如下圖 2 所示，包含四個壓力感測器(Force Sensor)、ADC 和 MCU 單晶片。當有重力施壓在壓力感測器上時，壓力感測器會將所得到的電壓訊號變化量，透過類比數位轉換(ADC)給單晶片(MCU)做後端的訊號處理計算，最後再透過 I2C 通訊輸出資料。在本文中，使用到的壓力感測器為 Uneo Force Sensor，詳細關於 Uneo Force Sensor 規格如下圖 3，MCU 單晶片與 ADC 規格部份可以章節 2.2 介紹。

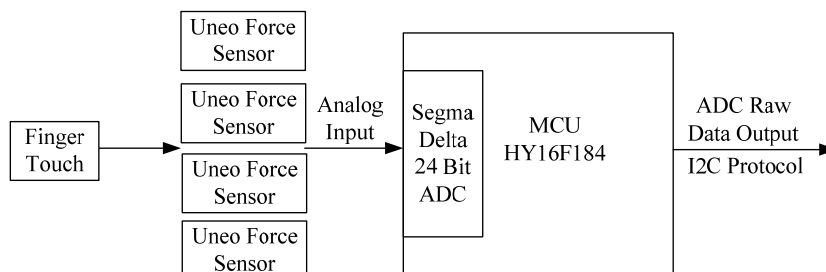
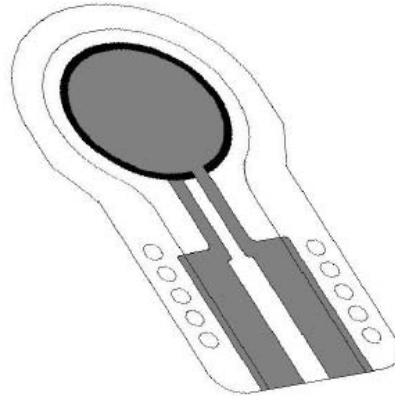


圖 2 HY16F184 Force Touch 基本架構應用圖



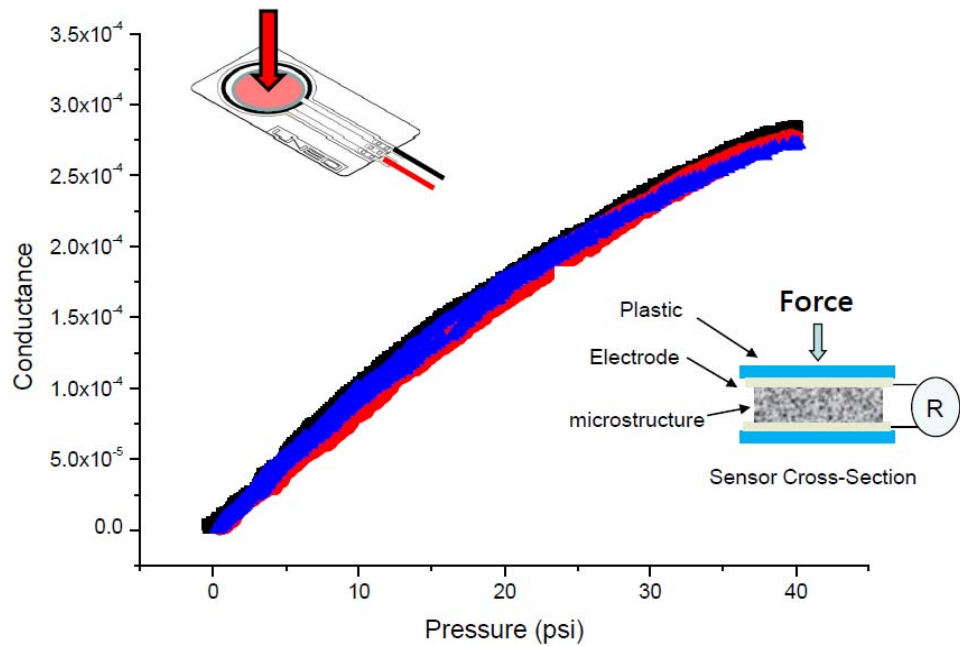
Uneo GD05

Table 1. Operating Characteristics (TA = 25°C unless otherwise noted, F is positive)

Characteristic	Symbol	Min	Typ	Max	Unit
Force Range ⁽¹⁾	F _{OP}	0	—	10	N
Linearity ⁽²⁾	—	—	0.99	—	—
Repeatability ⁽³⁾	—	—	—	±2	%
Hysteresis ⁽⁴⁾	—	—	—	±2	%
Drift ⁽⁵⁾	—	—	—	6	%/log(s)
Response Time ⁽⁶⁾	t _R	—	—	0.1	ms
Temperature Sensitivity ⁽⁷⁾	Y/T	—	—	0.35	%/°C

Sensor Characteristic

High linearity output vs. Pressure (or Force)



■ Sensors A, B, C

圖 3 Uneo Force Sensor 基本規格與特性

2.2. 控制晶片

單片機簡介：HY16F 系列 32 位元高性能 Flash 單片機(HY16F184)

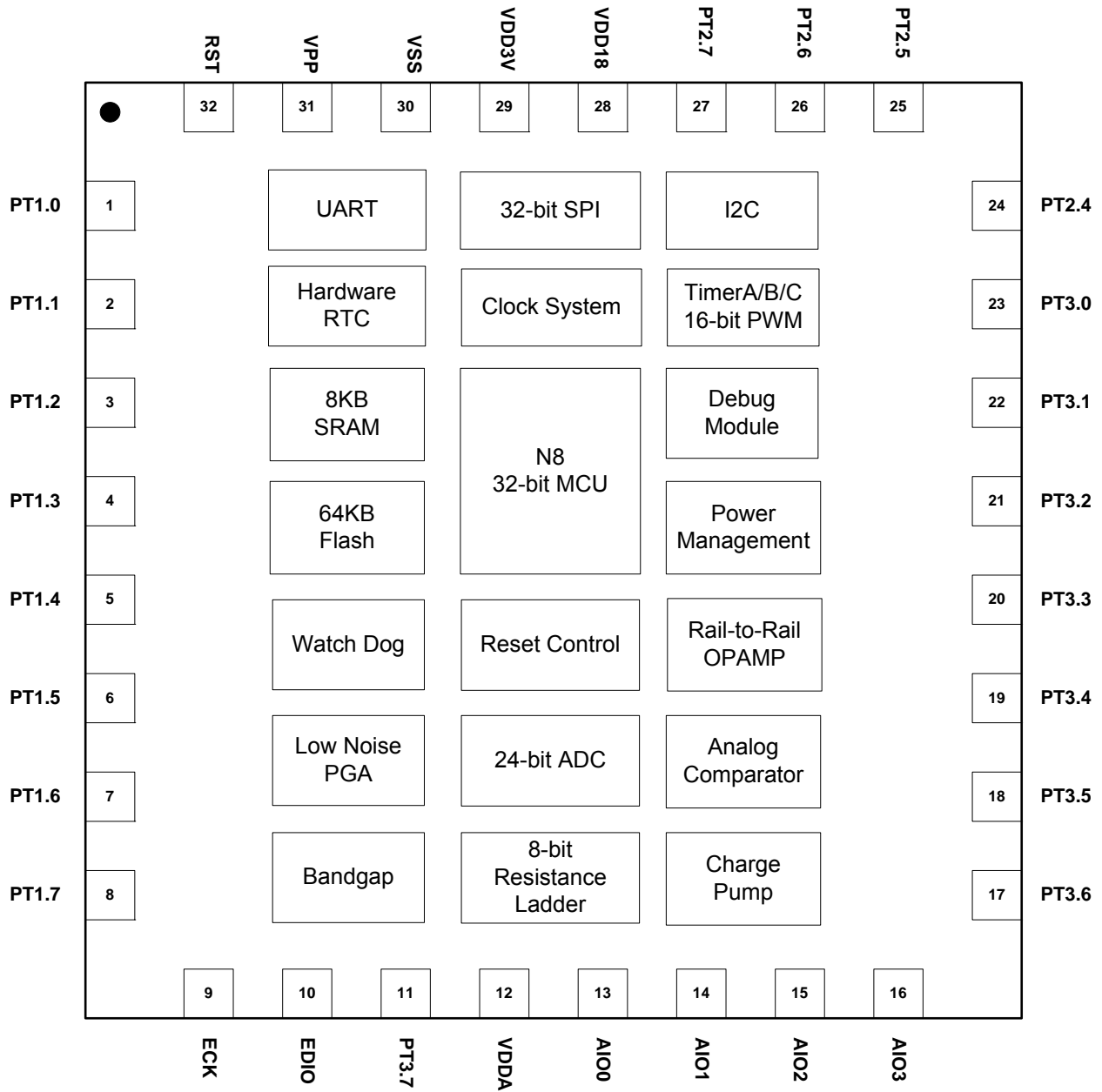


圖 4 紘康 HY16F 系列 32 位元高性能 Flash 單片機(HY16F184)

- (1)採用最新 Andes 32 位元 CPU 核心 N801 處理器。
- (2)電壓操作範圍 2.4~3.6V，以及-40°C~85°C工作溫度範圍。
- (3)支援外部 16MHz 石英震盪器或內部 20MHz 高精度 RC 震盪器，
擁有多種 CPU 工作時脈切換選擇，可讓使用者達到最佳省電規劃。
- (3.1)運行模式 350uA@2MHz/2(3.2)待機模式 10uA@32KHz/2(3.3)休眠模式 2.5uA
- (4)程式記憶體 64KBytes Flash ROM
- (5)資料記憶體 8KBytes SRAM。
- (6)擁有 BOR and WDT 功能，可防止 CPU 死機。
- (7)24-bit 高精準度 $\Sigma\Delta$ ADC 類比數位轉換器
- (7.1)內置 PGA (Programmable Gain Amplifier)最高可達 128 倍放大。
- (7.2)內置溫度感測器 TPS。
- (8)超低輸入雜訊運算放大器 OPAMP。
- (9)16-bit Timer A
- (10)16-bit Timer B 模組具 PWM 波形產生功能
- (11)16-bit Timer C 模組具數位 Capture/Compare 功能
- (12)硬體串列通訊 SPI 模組
- (13)硬體串列通訊 I2C 模組
- (14)硬體串列通訊 UART 模組
- (15)硬體 RTC 時鐘功能模組
- (16)硬體 Touch KEY 功能模組
- (17)Sigma-delta 24 Bit ADC ENOB & RMS Noise

ENOB(RMS) with OSR/GAIN at A/D Clock=333KHz, VDDA=2.4V, VREF=1.2V																
Max. Vin(mV) =0.9*VREF ⁽¹⁾	OSR				32	64	128	256	512	1024	2048	4096	8192	16384	32768	
	Output rate(HZ)				10417	5208	2604	1302	651	326	163	81	41	20	10	
	Gain	=	PGA	x												ADGN
±1080	1	=	1	x	1	12.5	15.0	16.6	17.3	17.7	18.1	18.7	19.2	19.6	20.3	20.7
±540	2	=	1	x	2	12.4	14.4	16.3	16.9	17.0	17.4	17.9	19.1	19.3	20.0	20.4
±135	4	=	1	x	4	12.2	14.6	16.1	16.6	16.9	17.2	17.9	18.8	19.4	19.8	20.3
±33.75	32	=	8	x	4	12.2	13.7	15.1	15.6	16.1	16.5	17.0	17.7	18.1	18.6	19.1
±16.875	64	=	16	x	4	12.1	13.8	14.6	15.2	15.6	16.2	16.7	17.2	17.6	18.1	18.5
±11.25	96	=	24	x	4	12.1	13.4	14.4	14.8	15.4	15.9	16.4	16.9	17.4	17.9	18.3
±8.435	128	=	32	x	4	12.0	13.3	14.1	14.7	15.1	15.6	16.1	16.6	17.1	17.6	18.1

(1) Max.Vin (mV) is the max. input voltage of single end to ground (VSS).

RMS Noise(uV) with OSR/GAIN at A/D Clock=333KHz, VDDA=2.4V, VREF=1.2V																
Max. Vin(mV) =0.9*VREF	OSR				32	64	128	256	512	1024	2048	4096	8192	16384	32768	
	Output rate(HZ)				10417	5208	2604	1302	651	326	163	81	41	20	10	
	Gain	=	PGA	x												ADGN
±1080	1	=	1	x	1	426.3	71.0	23.3	14.56	10.92	8.29	5.72	3.98	2.95	1.89	1.410
±540	2	=	1	x	2	216.6	54.1	14.5	9.93	9.37	7.17	4.77	2.19	1.89	1.12	0.838
±135	4	=	1	x	4	129.5	23.5	8.5	6.04	4.85	4.02	2.46	1.29	0.89	0.65	0.455
±33.75	32	=	8	x	4	15.8	5.5	2.1	1.53	1.07	0.78	0.56	0.36	0.27	0.18	0.135
±16.875	64	=	16	x	4	8.5	2.6	1.5	0.99	0.75	0.51	0.36	0.26	0.19	0.13	0.098
±11.25	96	=	24	x	4	5.9	2.3	1.2	0.85	0.59	0.41	0.30	0.21	0.14	0.10	0.078
±8.435	128	=	32	x	4	4.6	1.9	1.1	0.71	0.52	0.37	0.27	0.19	0.14	0.10	0.068

3. 系統設計

3.1. 硬體說明

使用 HY16F184 內建 ADC 搭配 Uneo Force Sensor 做 Force Touch 應用電路。
HY16F184 的 ADC 通道類比腳位分別與四個 Force Sensor 的 S+與 S-做連接，採取單端輸入式的連接方式。詳細 ADC 設置圖可以參考下圖 5~8 設定，詳細完整線路設計圖可以參考以下圖 9。

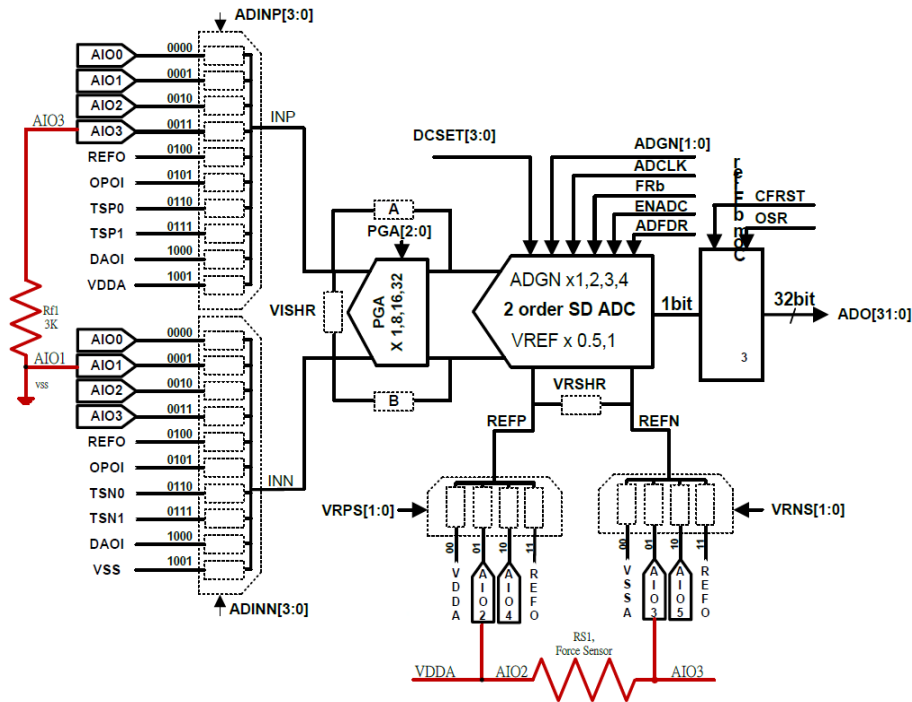


圖 5 通道 1 設置圖

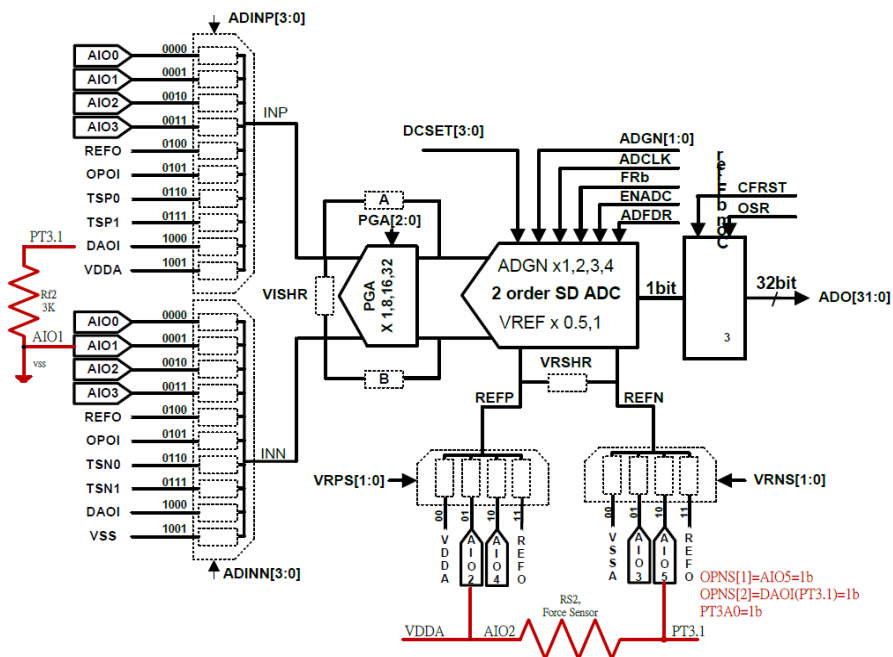


圖 6 通道 2 設置圖

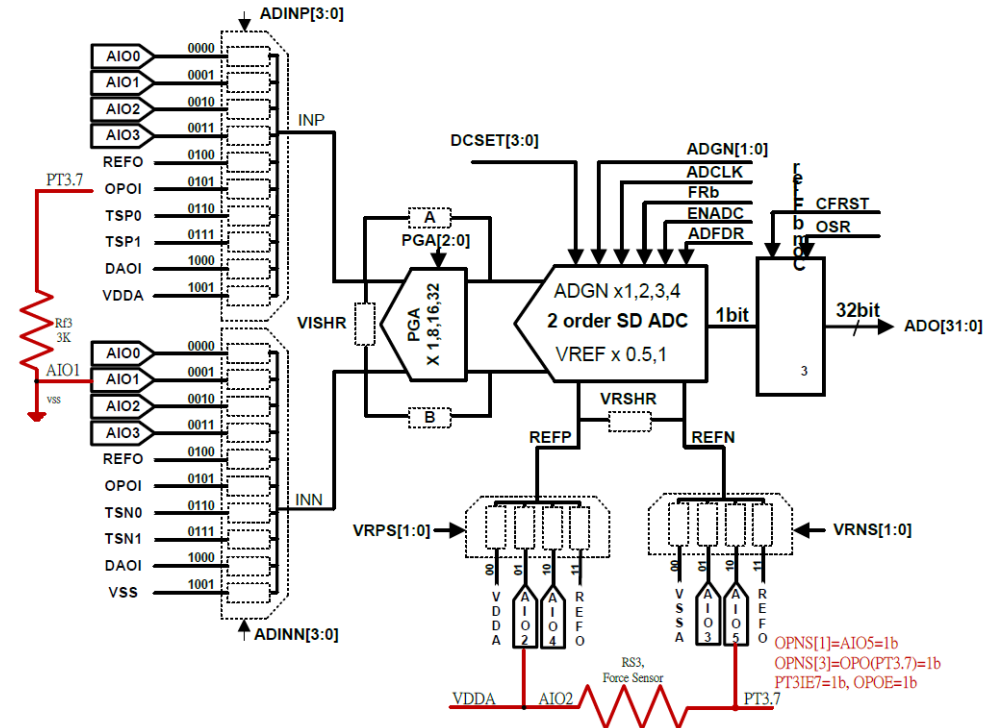


圖 7 通道 3 設置圖

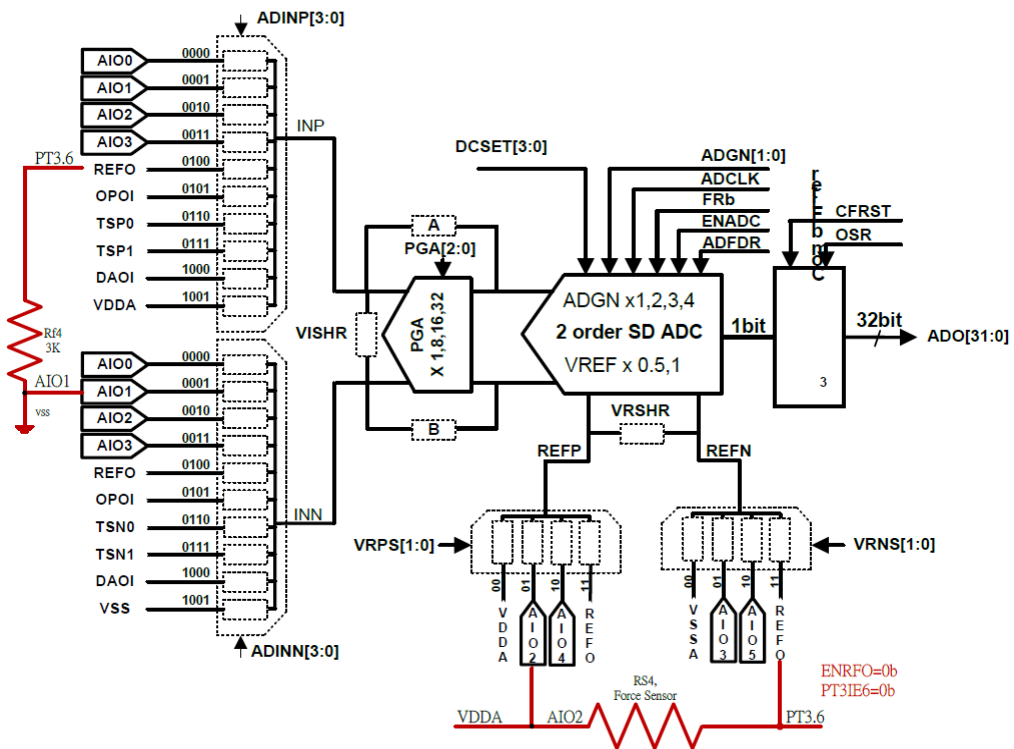


圖 8 通道 4 設置圖

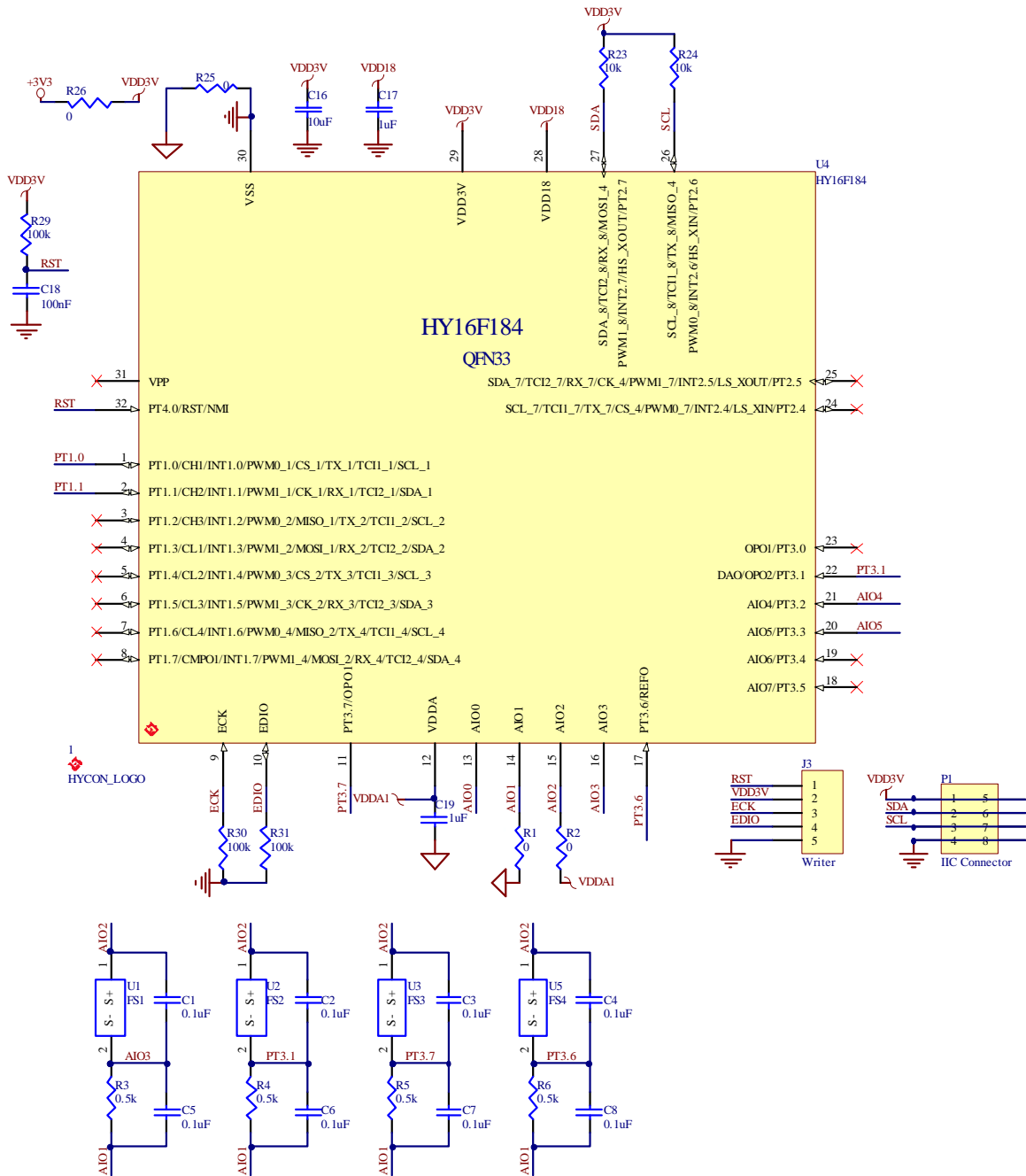


圖 9 HY16F184 Uno Force Sensor 硬體線路連接圖

主要元件介紹

- (1)HY16F184：資料處理與運算核心，主要負責執行運算四個 Force sensor 的掃描資料，並且透過 I2C 通訊做 ADC Raw Data 資料輸出。
- (2)ADC：HY16F184 內建之類比數位轉換器，將 Force sensor 所測得的電壓訊號，做類比數位電壓訊號轉換。
- (3)Uno Force Sensor：壓力感測器，負責偵測壓力大小，並且將壓力轉換為電壓訊號。
- (4)R3~R6 電阻：為分壓用之固定參考電阻，使用者可適應用自行調整固定參考電阻值大小。

3.2. 軟體說明

程式流程圖：

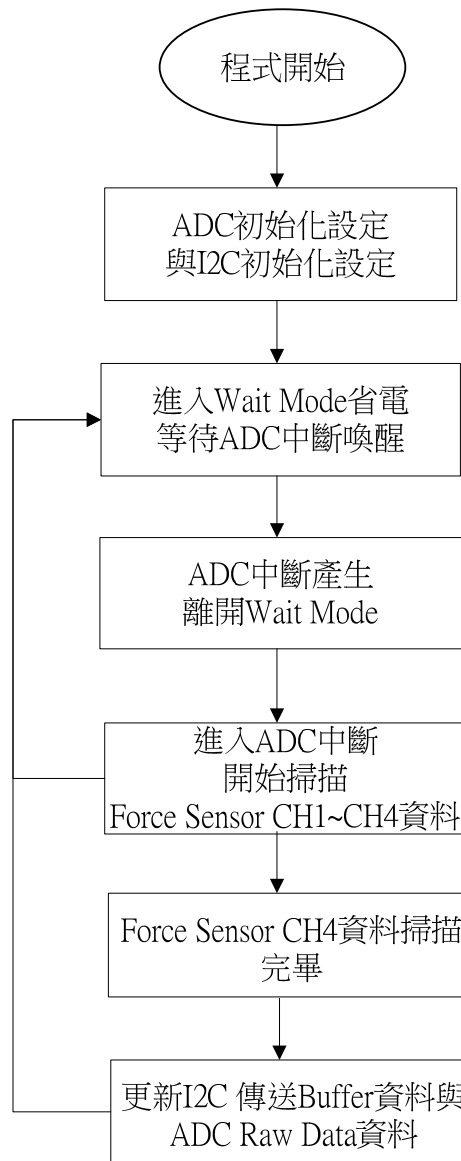


圖 10 Force sensor 掃描流程圖

4. 數據規格與總結

4.1. ADC Output Rate 測量

在本文中，通道掃描的方式為每一個通道都掃描四次再取平均值，所以四個通道總共會需要掃描 16 次，在 CPU 頻率設定為 2MHz 與 ADC OSR 設定為 128 的情況下(即 ADC 的資料輸出率為 2560sps)，每次從 CH1 到 CH4 的掃描時間總共需要花費 10ms，換算頻率約為 100Hz。在此條件下，如果要提升 ADC 的掃描速度，可選擇提升 ADC OSR 設置，但是這可能會損失了解析度，如果選擇提升了 CPU 的工作頻率，也可能會造成整體消耗功率過大，在此情況下，本文建議可使用移動平均法來做資料的平均與計算，使用此方式做掃描，可以在不提升 CPU 功耗與 ADC 頻率的情況下，滿足每個通道也為取四筆值取平均的條件，把 ADC Output Rate 速度從 100Hz 提升到 192Hz。移動平均法的方法為，只有第一次掃描四通道的 ADC Raw Data 需要完整的掃描 16 次，之後各個通道的掃描只需要做一次掃描，再與前面三筆舊的資料做平均值計算，不斷的遞迴更新資料。

4.2. 耗電流測量

在 CPU 頻率設定為 2MHz 與 ADC OSR 設定為 128 的情況下，使用移動平均法可以得到的 ADC Output Rate 為 192Hz，當 CPU 工作電壓 VDD=3V, VDDA=2.4V, R3~R6 固定參考電阻=0.5k 歐姆的時候，在此情況下所測得到的耗電流約 525uA，此為 Uneo Force Sensor 在沒有承受任何重力情況下的耗電流，但是消耗電流，是會隨著當下 Uneo Force Sensor 感應到的重力越大而相對的增加耗電流，主要原因在於 Force Sensor 本身阻抗變化的關係。

4.3. 最大承受力

在 ADC Gain=1，PGA=1，R3~R6 固定參考電阻=0.5k 歐姆的情況下，可以滿足最大約 1.2kg 秤重，使用者可以自行修改 ADC 的 Gain 值，以滿足不同的應用。

4.4. ADC Raw Data 與 I2C 通訊格式說明

I2C Slave Address:0x20

I2C Command:0x80

ADC Raw Data Format:

S+Addr+0x80+rS+(Addr+1)+CH1Data_L+CH1Data_M+CH1Data_H+CH2Data_L+CH2Data_M+CH2Data_H+CH3Data_L+CH3Data_M+CH3Data_H+CH4Data_L+CH4Data_M+CH4Data_H+P

S: Star; Addr: Slave address; rS: repeat start; P: stop.

CH1,CH2,CH3,CH4: Force Sensor ADC Raw Data;

L: ADC Low byte; M: ADC Middle byte; H: ADC High byte;

每個通道數據(Chx)共 $8*3=24$ bit

Bit0，統一為旗標，Bit0=0b，代表為舊資料; Bit0=1b，代表為新資料;

使用者應該在 Bit0=1b 時，取得資料才有效.

Bit23，統一為 Sign bit，

Bit23=0b，代表正數; Bit23=1b，代表負數

4.5. ADC Raw Data 資料顯示介面介紹

掃描 Force Sensor 所輸出的 ADC Raw Data 可透過 I2C 介面來做資料的傳輸與讀取，搭配紘康設計的 I2C 轉 USB 的橋接器配合 PC 端的 GUI，可以做為即時的 ADC Raw Data 資料顯示。詳細資料畫面顯示 GUI 操作說明，可以參考如下：

- 1.Connect : USB 連接狀態，如果有正常連接會顯示 Connect，如果連接不正常，會顯示 control board connect fail
- 2.I2C Slave addr: 預設為 0x20.
- 3.Chart: 顯示四個通道的 Force sensor 掃描資料
- 4.Scan: 開始讀取四個通道的 Force sensor 掃描資料
- 5.Save: 存取四個通道的 Forec sensor 掃描資料

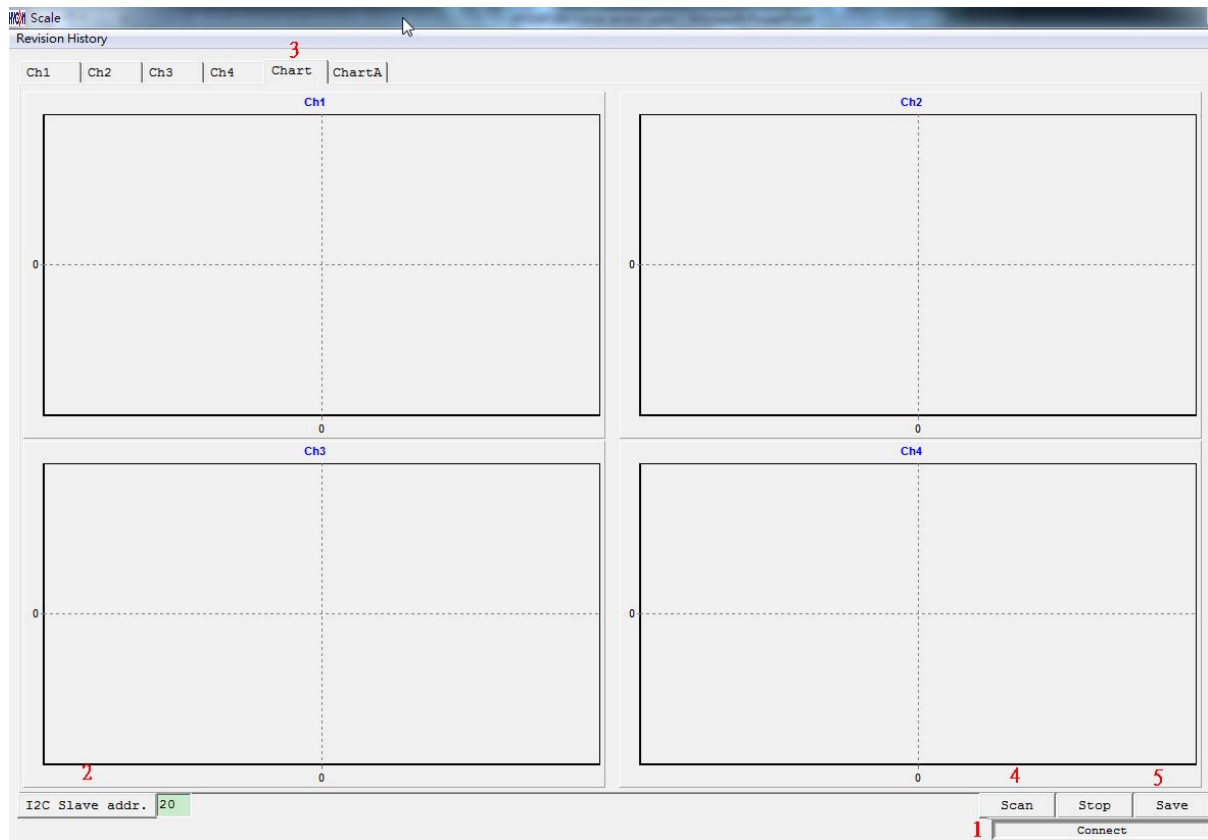


圖 11 ADC Raw Data 資料顯示介面

4.6. ADC Raw Data 分辨率測試

在本文測試中，如果在 Uneo Force Sensor 沒有放置任何重力物的情況下，可以得到穩定度約 +/-2~ +/-3 counts 誤差內(ADC output code: +/-15bits)，詳細可以由下圖觀察之。

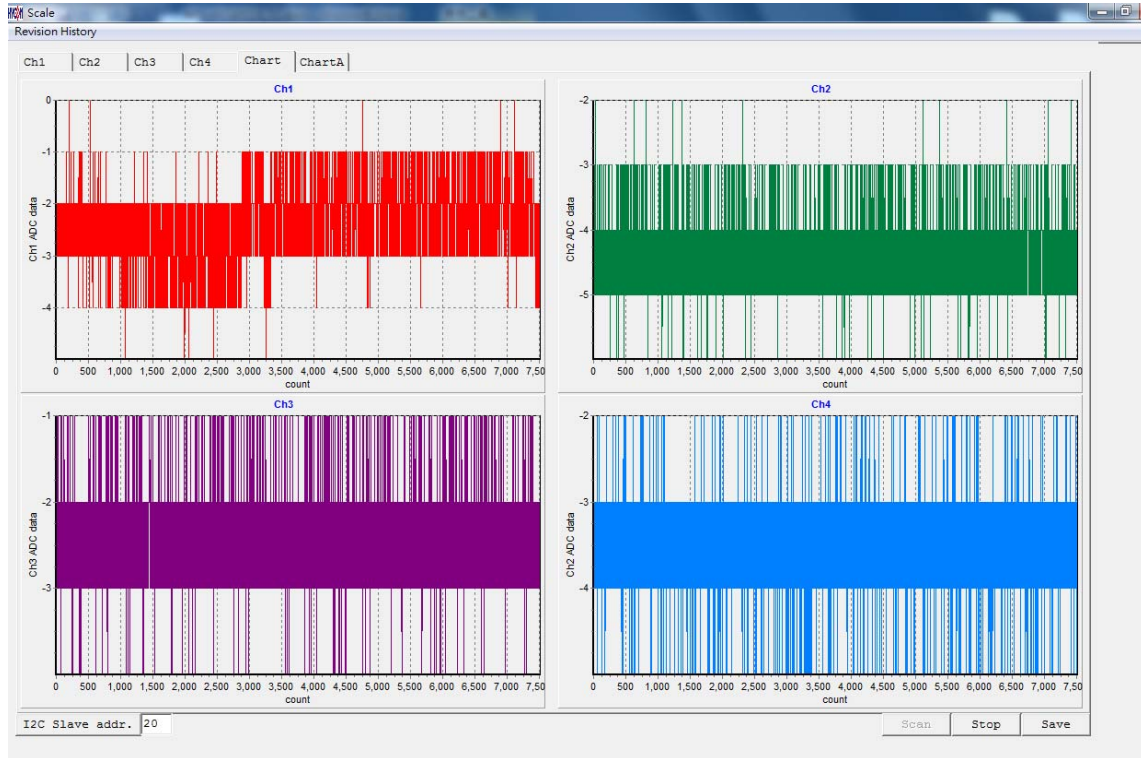


圖 12 Uneo Force Sensor 無放置任何重力物情況下的 ADC Raw Data 變化量

嘗試在 CH1 的位置上放置 100g 的砝碼，觀察 CH1 的 ADC Raw Data 之變化量。CH3 在尚未放置 100g 砝碼前的 ADC Raw Data 約-1~-2，放置之後 ADC Raw Data 約 2150，所以 100g 的砝碼大約會有 2150 的 ADC Count 分辨率，此為 ADC Gain=1, PGA=1 時候的測試情況。測試結果可以由下圖畫面顯示清楚看到 ADC Raw Data 的變化量。



圖 13 在 CH1 放置 100g 砝碼時候的 ADC Raw Data 變化量

4.7. 總結

在本文中，提供了完整的 Uneo Force Sensor 相關應用與開發工具供使用者參考，使用者可以依據四個通道的 ADC Raw Data 變化量，來做後續的 X, Y 軸座標與 Z 軸強度計算，等功能的設計與開發。

5. Demo Code 及相關檔案

```
/*-----*/  
/* Includes */  
/*-----*/  
  
#include "HY16F188.h"  
#include "System.h"  
#include "DrvGPIO.h"  
#include "DrvI2C.h"  
#include "DrvCLOCK.h"  
#include "my_define.h"  
#include "DrvADC.h"  
#include "DrvTimer.h"  
#include "DrvADC.h"  
#include "DrvPMU.h"  
  
/*-----*/  
/* STRUCTURES */  
/*-----*/  
  
typedef union _MCUSTATUS  
{  
    char _byte;  
    struct  
    {  
        unsigned b_ADCdone:1;  
        unsigned b_TMAdone:1;  
        unsigned b_TMBdone:1;  
        unsigned b_TMC0done:1;  
        unsigned b_TMC1done:1;  
        unsigned b_RTCdone:1;  
        unsigned b_I2C_TxDone:1;  
        unsigned b_I2C_RxDone:1;  
    };  
} MCUSTATUS;  
  
/*-----*/  
/* DEFINITIONS */  
/*-----*/  
  
#define Disable 0  
#define Enable 1  
#define I2CBufferSize 256
```

```
/*-----*/
/* Global CONSTANTS                                     */
/*-----*/

MCUSTATUS MCUSTATUSbits;

// Data range 32767~-32768, +- 15bit format(bit30~bit16), Bit31==Sign, Bit0=Flag; 0:old, 1:new
int ADC_CH1, ADC_CH2, ADC_CH3, ADC_CH4;
int ADC_CH1_Array[4], ADC_CH2_Array[4], ADC_CH3_Array[4], ADC_CH4_Array[4];
int ADC_CH1_Count, ADC_CH2_Count, ADC_CH3_Count, ADC_CH4_Count;
char CH1_LowByte, CH2_LowByte, CH3_LowByte, CH4_LowByte;
char CH1_MiddleByte, CH2_MiddleByte, CH3_MiddleByte, CH4_MiddleByte;
char CH1_HighByte, CH2_HighByte, CH3_HighByte, CH4_HighByte;
unsigned char ADC_CH4_Finish;

int i;
unsigned char ADfinish;
int ADCData;

unsigned char Buffer[I2CBufferSize];
unsigned char I2C_RW;
unsigned char I2C_TARGET; // Target I2C slave address
unsigned char I2C_Sendbuf[I2CBufferSize];
unsigned char I2C_Recbuf[I2CBufferSize];
unsigned int DataTxLen,DataTxIndex,DataRxLen,DataRxIndex,EndFlag;
unsigned int GeneralFlag;

/*-----*/
/* Function PROTOTYPES                                     */
/*-----*/

void Delay(unsigned int num);
void InitalADC0(void);
void HWI2C_INI(void);

/*-----*/
/* Main Function                                           */
/*-----*/

int main(void)
{

// GPIO Trigger/Output Rate setting
pio_00=0x00000808; //PT10E3=1, //PT1.3 Output Mode, check output rate=Interrupt pin
```

```
pio_04=0x00000808; //PT1D03=1 PT1.3 pull High, check output rate=Interrupt pin

//ADC initialization
op_00=0x00000202; // OPOE=1, let OPOI=OPO, setting pin15
pio_24=0x80800000; //PT3IE7=1b, let OPOI=OPO, setting pin15
InitalADC0(); //ADC CH0, 16F188, pin20/pin19

//Data initialization=0
ADC_CH1=0, ADC_CH2=0, ADC_CH3=0, ADC_CH4=0, ADCData=0, i=0;
CH1_LowByte=0, CH2_LowByte=0, CH3_LowByte=0, CH4_LowByte=0;
CH1_MiddleByte=0, CH2_MiddleByte=0, CH3_MiddleByte=0, CH4_MiddleByte=0;
CH1_HighByte=0, CH2_HighByte=0, CH3_HighByte=0, CH4_HighByte=0;
ADC_CH1_Count=0, ADC_CH2_Count=0, ADC_CH3_Count=0, ADC_CH4_Count=0;
ADC_CH4_Finish=0; //Disable I2C Slave data Buffer Update
ADfinish=0;

for(i=0;i<=256;i++) //if 6=HY14E10 Test tool AP
{
    I2C_Sendbuf[i]=0; //Initial I2C data buffer=0
    I2C_Recbuf[i]=0;
}

for(i=0;i<=3;i++)
{
    ADC_CH1_Array[i]=0;
    ADC_CH2_Array[i]=0;
    ADC_CH3_Array[i]=0;
    ADC_CH4_Array[i]=0;
}

DataRxIndex=0;
DataTxIndex=0;
EndFlag=0;
DataTxLen=256;

// I2C initialization
HWI2C_INI();

// SPI Interrupt Enable, Dummy setting, in order to enter wait mode
```

```
int_00=0x03030303; //STXIE=1, SRXIE=1, STXIF=1, SRXIF=1
```

```
DrvADC_CombFilter(Disable); //Disable CombFilter  
DrvADC_CombFilter(Enable); //Enable CombFilte  
SYS_EnableGIE(7,0x3F); // Enable GIE(Global Interrupt)
```

```
for(i=0;i<=15;i++)  
{  
    switch (i)  
    {  
        case 0: //讀左上-1  
            while(!ADfinish);  
            ADfinish=0;  
            int_00=0x30003000; //I2CIE=0, I2CEIE=0  
            ADC_CH1_Array[0]=ADCData;  
            DrvADC_ClearIntFlag();  
            DrvADC_EnableInt();  
            SYS_LowPower(2); //wait mode  
            break;  
  
        case 1: //讀左上-2  
            ADfinish=0;  
            int_00=0x30003000; //I2CIE=0, I2CEIE=0  
            ADC_CH1_Array[1]=ADCData;//>>16;  
            DrvADC_ClearIntFlag();  
            DrvADC_EnableInt();  
            SYS_LowPower(2); //wait mode  
            break;  
  
        case 2: //讀左上-3  
            ADfinish=0;  
            int_00=0x30003000; //I2CIE=0, I2CEIE=0  
            ADC_CH1_Array[2]=ADCData;  
            DrvADC_ClearIntFlag();  
            DrvADC_EnableInt();  
            SYS_LowPower(2); //wait mode  
            break;  
  
        case 3: //讀左上-4
```

```
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH1_Array[3]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
op_04=0x0000606; // OPNS[1]=AI05=1b, OPNS[2]=DAOI(PT3.1)=1b, ADC_CH2 Setting
pio_28=0x0202000; // PT3A0=1b, ADC_CH2 Setting
DrvADC_SetADCInputChannel(DAO,ADC_Input_AI01); //Switch to ADC_CH2 ADC input
DrvADC_RefVoltage(AI02,AI05); //Set the ADC reference voltage.//Switch to ADC_CH2 ADC
RefVoltage
DrvADC_CombFilter(Enable); //Enable CombFilter
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH1=(ADC_CH1_Array[0]+ ADC_CH1_Array[1]+ ADC_CH1_Array[2]+ ADC_CH1_Array[3])>>2;
//divider by 4
CH1_LowByte=(ADC_CH1 >> 7) & 0xFF;
CH1_MiddleByte=(ADC_CH1 >> 15) & 0xFF;
CH1_HighByte=(ADC_CH1 >> 23) & 0xFF;
SYS_LowPower(2); //wait mode
break;

case 4: //讀右上-1
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH2_Array[0]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 5: //讀右上-2
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH2_Array[1]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 6: //讀右上-3
ADfinish=0;
```

```
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH2_Array[2]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 7: //讀右上-4
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH2_Array[3]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
op_04=0x00000600; // OPNS[1]=AI05=0b, OPNS[2]=DAOI(PT3.1)=0b, cancel ADC_CH2 Setting
pio_28=0x02000000; // PT3A0=0b, cancel ADC_CH2 Setting
op_04=0x00000A0A; // OPNS[1]=AI05=1b, OPNS[3]=OPO(PT3.7)=1b, cancel ADC_CH2 Setting
pio_24=0x80800000; //PT3IE7=1b, ADC_CH3 Setting
op_00=0x00000202; //OPOE=1b, ADC_CH3 Setting
DrvADC_SetADCInputChannel(OPO,ADC_Input_AI01); //Switch to ADC_CH3 ADC input
DrvADC_RefVoltage(AIO2,AIO5); //Set the ADC reference voltage.//Switch to ADC_CH3 ADC
RefVoltage
DrvADC_CombFilter(Enable); //Enable CombFilte
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH2=(ADC_CH2_Array[0]+ ADC_CH2_Array[1]+ ADC_CH2_Array[2]+ ADC_CH2_Array[3])>>2;
//divider by 4
CH2_LowByte=(ADC_CH2 >> 7) & 0xFF;
CH2_MiddleByte=(ADC_CH2 >> 15) & 0xFF;
CH2_HighByte=(ADC_CH2 >> 23) & 0xFF;
SYS_LowPower(2); //wait mode
break;

case 8: //讀右下-1
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH3_Array[0]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;
```

```
case 9: //讀右下-2
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH3_Array[1]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 10: //讀右下-3
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH3_Array[2]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 11: //讀右下-4
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH3_Array[3]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
op_04=0x00000A00; // OPNS[1]=AI05=0b, OPNS[3]=OPO(PT3.7)=0b, cancel ADC_CH3 Setting
pio_24=0x80000000; //PT3IE7=0b, cancel ADC_CH3 Setting
op_00=0x00000200; //OPOE=0b, cancel ADC_CH3 Setting
pmu_00=0x00000200; //ENRFO=0b, ADC_CH4 Setting
pio_24=0x40000000; // PT3IE6=0b, ADC_CH4 Setting
DrvADC_SetADCInputChannel(ACM,ADC_Input_AI01); //Switch to ADC_CH4 ADC input
DrvADC_RefVoltage(AIO2,3); //Set the ADC reference voltage. //Switch to ADC_CH4 ADC
RefVoltage
DrvADC_CombFilter(Enable); //Enable CombFilte
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH3=(ADC_CH3_Array[0]+ADC_CH3_Array[1]+ADC_CH3_Array[2]+ADC_CH3_Array[3])>>2;
//divider by 4
CH3_LowByte=(ADC_CH3 >> 7) & 0xFF;
CH3_MiddleByte=(ADC_CH3 >> 15) & 0xFF;
CH3_HighByte=(ADC_CH3 >> 23) & 0xFF;
SYS_LowPower(2); //wait mode
```

```
break;

case 12: //讀左下-1
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH4_Array[0]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 13: //讀左下-2
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH4_Array[1]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 14: //讀左下-3
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH4_Array[2]=ADCData;
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
SYS_LowPower(2); //wait mode
break;

case 15: //讀左下-4
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH4_Array[3]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
pmu_00=0x0000202; //ENRF0=1b, Cancel ADC_CH4 Setting
pio_24=0x4040000; // PT3IE6=1b, Cancel ADC_CH4 Setting
DrvADC_SetADCInputChannel(ADC_Input_AI03,ADC_Input_AI01); //Switch to ADC_CH1 ADC input
DrvADC_RefVoltage(AI02,AI03); //Set the ADC reference voltage. //Switch to ADC_CH1 ADC
RefVoltage
//DrvADC_RefVoltage(AI02,VSSA); //test only
```



```
    DrvADC_CombFilter(Enable); //Enable CombFilte
    DrvADC_ClearIntFlag();
    DrvADC_EnableInt();
    ADC_CH4=(ADC_CH4_Array[0]+ADC_CH4_Array[1]+ADC_CH4_Array[2]+ADC_CH4_Array[3])>>2;
//divider by 4
    CH4_LowByte=(ADC_CH4 >> 7) & 0xFF;
    CH4_MiddleByte=(ADC_CH4 >> 15) & 0xFF;
    CH4_HighByte=(ADC_CH4 >> 23) & 0xFF;
    I2C_Sendbuf[13]=CH1_LowByte;
    I2C_Sendbuf[14]=CH1_MiddleByte;
    I2C_Sendbuf[15]=CH1_HighByte;
    I2C_Sendbuf[16]=CH2_LowByte;
    I2C_Sendbuf[17]=CH2_MiddleByte;
    I2C_Sendbuf[18]=CH2_HighByte;
    I2C_Sendbuf[19]=CH3_LowByte;
    I2C_Sendbuf[20]=CH3_MiddleByte;
    I2C_Sendbuf[21]=CH3_HighByte;
    I2C_Sendbuf[22]=CH4_LowByte;
    I2C_Sendbuf[23]=CH4_MiddleByte;
    I2C_Sendbuf[24]=CH4_HighByte;
    ADC_CH4_Finish=1; //Enable I2C Slave data Buffer Update
    pio_04=0x00000800; //PT1D03=0 PT1.3 pull low, check output rate=Interrupt pin
    pio_04=0x00000808; //PT1D03=1 PT1.3 pull High, check output rate=Interrupt pin
    SYS_LowPower(2); //idle mode
    break;
}
}
```

```
while(1) //to do Moving_Average
{
    for(i=0;i<=3;i++)
    {
        switch (i)
        {

            case 0: //讀左上-1
                ADfinish=0;
                int_00=0x30003000; //I2CIE=0, I2CEIE=0
                ADC_CH1_Array[ADC_CH1_Count]=ADCData;
                DrvADC_CombFilter(Disable); //Disable CombFilter
```

```
op_04=0x00000606; // OPNS[1]=AI05=1b, OPNS[2]=DAOI(PT3.1)=1b, ADC_CH2 Setting
pio_28=0x02020000; // PT3A0=1b, ADC_CH2 Setting
DrvADC_SetADCInputChannel(DAO,ADC_Input_AI01); //Switch to ADC_CH2 ADC input
DrvADC_RefVoltage(AI02,AI05); //Set the ADC reference voltage. //Switch to ADC_CH2
ADC RefVoltage
DrvADC_CombFilter(Enable); //Enable CombFilter
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH1_Count++;
if(ADC_CH1_Count==4)
{
    ADC_CH1_Count=0;
}
ADC_CH1=(ADC_CH1_Array[0]+ ADC_CH1_Array[1]+ ADC_CH1_Array[2]+ ADC_CH1_Array[3])>>2;
//divider by 4
CH1_LowByte=(ADC_CH1 >> 7) & 0xFF;
CH1_MiddleByte=(ADC_CH1 >> 15) & 0xFF;
CH1_HighByte=(ADC_CH1 >> 23) & 0xFF;
SYS_LowPower(2); //wait mode
break;

case 1: //讀右上-1
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH2_Array[ADC_CH2_Count]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
op_04=0x00000600; // OPNS[1]=AI05=0b, OPNS[2]=DAOI(PT3.1)=0b, cancel ADC_CH2 Setting
pio_28=0x02000000; // PT3A0=0b, cancel ADC_CH2 Setting
op_04=0x00000A0A; // OPNS[1]=AI05=1b, OPNS[3]=OPO(PT3.7)=1b, cancel ADC_CH2 Setting
pio_24=0x80800000; //PT3IE7=1b, ADC_CH3 Setting
op_00=0x00000202; //OPOE=1b, ADC_CH3 Setting
DrvADC_SetADCInputChannel(OPO,ADC_Input_AI01); //Switch to ADC_CH3 ADC input
DrvADC_RefVoltage(AI02,AI05); //Set the ADC reference voltage. //Switch to ADC_CH3
ADC RefVoltage
DrvADC_CombFilter(Enable); //Enable CombFilte
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH2_Count++;
if(ADC_CH2_Count==4)
```

```
{
    ADC_CH2_Count=0;
}
ADC_CH2=(ADC_CH2_Array[0]+ ADC_CH2_Array[1]+ ADC_CH2_Array[2]+ ADC_CH2_Array[3])>>2;
//divider by 4
CH2_LowByte=(ADC_CH2 >> 7) & 0xFF;
CH2_MiddleByte=(ADC_CH2 >> 15) & 0xFF;
CH2_HighByte=(ADC_CH2 >> 23) & 0xFF;
SYS_LowPower(2); //wait mode
break;

case 2: //讀右下-1
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH3_Array[ADC_CH3_Count]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
op_04=0x00000A00; // OPNS[1]=AI05=0b, OPNS[3]=OPO(PT3.7)=0b, cancel ADC_CH3 Setting
pio_24=0x80000000; //PT3IE7=0b, cancel ADC_CH3 Setting
op_00=0x00000200; //OPOE=0b, cancel ADC_CH3 Setting
pmu_00=0x00000200; //ENRF0=0b, ADC_CH4 Setting
pio_24=0x40000000; // PT3IE6=0b, ADC_CH4 Setting
DrvADC_SetADCInputChannel(ACM,ADC_Input_AI01); //Switch to ADC_CH4 ADC input
DrvADC_RefVoltage(AI02,3); //Set the ADC reference voltage. //Switch to ADC_CH4
ADC RefVoltage
DrvADC_CombFilter(Enable); //Enable CombFilte
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH3_Count++;
if(ADC_CH3_Count==4)
{
    ADC_CH3_Count=0;
}
ADC_CH3=(ADC_CH3_Array[0]+ADC_CH3_Array[1]+ADC_CH3_Array[2]+ADC_CH3_Array[3])>>2;
//divider by 4
CH3_LowByte=(ADC_CH3 >> 7) & 0xFF;
CH3_MiddleByte=(ADC_CH3 >> 15) & 0xFF;
CH3_HighByte=(ADC_CH3 >> 23) & 0xFF;
SYS_LowPower(2); //wait mode
break;
```

```
case 3: //讀左下-1
ADfinish=0;
int_00=0x30003000; //I2CIE=0, I2CEIE=0
ADC_CH4_Array[ADC_CH4_Count]=ADCData;
DrvADC_CombFilter(Disable); //Disable CombFilter
pmu_00=0x00000202; //ENRF0=1b, Cancel ADC_CH4 Setting
pio_24=0x40400000; // PT3IE6=1b, Cancel ADC_CH4 Setting
DrvADC_SetADCInputChannel(ADC_Input_AI03,ADC_Input_AI01); //Switch to ADC_CH1 ADC
input
DrvADC_RefVoltage(AI02,AI03); //Set the ADC reference voltage. //Switch to ADC_CH1
ADC RefVoltage
//DrvADC_RefVoltage(AI02,VSSA); //test only
DrvADC_CombFilter(Enable); //Enable CombFilter
DrvADC_ClearIntFlag();
DrvADC_EnableInt();
ADC_CH4_Count++;
if(ADC_CH4_Count==4)
{
ADC_CH4_Count=0;
}
ADC_CH4=(ADC_CH4_Array[0]+ADC_CH4_Array[1]+ADC_CH4_Array[2]+ADC_CH4_Array[3])>>2;
//divider by 4
CH4_LowByte=(ADC_CH4 >> 7) & 0xFF;
CH4_MiddleByte=(ADC_CH4 >> 15) & 0xFF;
CH4_HighByte=(ADC_CH4 >> 23) & 0xFF;
I2C_Sendbuf[13]=CH1_LowByte;
I2C_Sendbuf[14]=CH1_MiddleByte;
I2C_Sendbuf[15]=CH1_HighByte;
I2C_Sendbuf[16]=CH2_LowByte;
I2C_Sendbuf[17]=CH2_MiddleByte;
I2C_Sendbuf[18]=CH2_HighByte;
I2C_Sendbuf[19]=CH3_LowByte;
I2C_Sendbuf[20]=CH3_MiddleByte;
I2C_Sendbuf[21]=CH3_HighByte;
I2C_Sendbuf[22]=CH4_LowByte;
I2C_Sendbuf[23]=CH4_MiddleByte;
I2C_Sendbuf[24]=CH4_HighByte;
ADC_CH4_Finish=1; //Enable I2C Slave data Buffer Update
pio_04=0x00000800; //PT1D03=0 PT1.3 pull low, check output rate=Interrupt pin
```

```
        pio_04=0x00000808; //PT1D03=1 PT1.3 pull High, check output rate=Interrupt pin
        SYS_LowPower(2); //idle mode
        break;
    }
}

asm("NOP");
return 0;

}

/*-----*/
/* Hardware Communication Interrupt */
/* UART/SPI/I2C Interrupt Service Routines */
/*-----*/
void HW0_ISR(void)
{
    unsigned char I2C_Status;

    if(DrvI2C_ReadIntFlag()==E_DRVI2C_INT) // Get I2C Interrupt Flag
    {
        I2C_Status=DrvI2C_GetStatusFlag(); // Get I2C Status Flag
        switch(I2C_Status)
        {
            case 0x44: //SACTFlag+ACKFlag
                { //Own slave A + W has been received. ACK has been transmitted.
                    DataRxIndex=0;
                    MCUSTATUSbits.b_I2C_RxDone=1;
                    DrvI2C_Ctrl(0,0,0,1); // Set ACK bit
                    break;
                }
            case 0x4C: //SACTFlag+DFFlag+ACKFlag
                { //Data byte has been received. ACK has been transmitted.
                    I2C_Recbuf[DataRxIndex]=DrvI2C_ReadData();
                    if(I2C_Recbuf[DataRxIndex]==0x80)
                    {
                        if(ADC_CH4_Finish==1)
                        {
                            I2C_Sendbuf[1]=I2C_Sendbuf[13]|0x1; //Ch1 bit0=1b, new data
                        }
                    }
                }
        }
    }
}
```

```
I2C_Sendbuf[2]=I2C_Sendbuf[14];
I2C_Sendbuf[3]=I2C_Sendbuf[15];
I2C_Sendbuf[4]=I2C_Sendbuf[16]|0x1; //Ch2 bit0=1b, new data
I2C_Sendbuf[5]=I2C_Sendbuf[17];
I2C_Sendbuf[6]=I2C_Sendbuf[18];
I2C_Sendbuf[7]=I2C_Sendbuf[19]|0x1; //Ch3 bit0=1b, new data
I2C_Sendbuf[8]=I2C_Sendbuf[20];
I2C_Sendbuf[9]=I2C_Sendbuf[21];
I2C_Sendbuf[10]=I2C_Sendbuf[22]|0x1; //Ch4 bit0=1b, new data
I2C_Sendbuf[11]=I2C_Sendbuf[23];
I2C_Sendbuf[12]=I2C_Sendbuf[24];
    ADC_CH4_Finish=0;
}
else
{
    I2C_Sendbuf[1]=I2C_Sendbuf[1]|0x0; //Ch1 bit0=0b, old data
I2C_Sendbuf[2]=I2C_Sendbuf[2];
I2C_Sendbuf[3]=I2C_Sendbuf[3];
I2C_Sendbuf[4]=I2C_Sendbuf[4]|0x0; //Ch2 bit0=0b, old data
I2C_Sendbuf[5]=I2C_Sendbuf[5];
I2C_Sendbuf[6]=I2C_Sendbuf[6];
I2C_Sendbuf[7]=I2C_Sendbuf[7]|0x0; //Ch3 bit0=0b, old data
I2C_Sendbuf[8]=I2C_Sendbuf[8];
I2C_Sendbuf[9]=I2C_Sendbuf[9];
I2C_Sendbuf[10]=I2C_Sendbuf[10]|0x0; //Ch4 bit0=0b, old data
I2C_Sendbuf[11]=I2C_Sendbuf[11];
I2C_Sendbuf[12]=I2C_Sendbuf[12];

}
}
DataRxIndex++;
DrvI2C_Ctrl(0,0,0,1); // Set ACK bit
break;
}
case 0x48: //SACTFlag+DFFlag
{ //Data byte has been received. NACK has been transmitted.
I2C_Recbuf[DataRxIndex++]=DrvI2C_ReadData();
DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag
break;
}
}
```

```
case 0x4A: // I2C General Call
{ //One data byte has been received. NACK has been transmitted
  GeneralFlag=1;
  DataRxIndex=0;
  MCUSTATUSbits.b_I2C_RxDone=1;
  I2C_Recbuf[DataRxIndex++]=DrvI2C_ReadData();
  DrvI2C_Ctrl(0,0,0,1); // Set ACK bit
  break;
}

case 0x4E: // I2C General Call
{ //One data byte has been received. ACK has been transmitted
  GeneralFlag=1;
  I2C_Recbuf[DataRxIndex++]=DrvI2C_ReadData();
  DrvI2C_Ctrl(0,0,0,1); // Set ACK bit
  break;
}

case 0x54: //SACTFlag+RWFlag+ACKFlag
{ //Own slave A + R has been received. ACK has been transmitted.
  MCUSTATUSbits.b_I2C_TxDone=1;
  DataTxIndex=0;
  DataTxIndex++; //Robert Add, Read data index
  DrvI2C_WriteData(I2C_Sendbuf[DataTxIndex++]); // Send Data to Master
  DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag
  break;
};

case 0x55: //SACTFlag+RWFlag
{ //Own slave A + R has been received. NACK has been transmitted.
  DataTxIndex=0;
  MCUSTATUSbits.b_I2C_TxDone=1;
  DrvI2C_WriteData(I2C_Sendbuf[DataTxIndex++]); // Send Data to Master
  DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag
  break;
};

case 0x5C:
{ //Data byte has been transmitted. ACK has been transmitted.
  DrvI2C_WriteData(I2C_Sendbuf[DataTxIndex++]); // Send Data to Master
  DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag
  break;
};
```

```
};  
case 0x58:  
{ //Data byte has been transmitted. NACK has been received.  
  DataTxLen=DataTxIndex;  
  EndFlag=1;  
  DrvI2C_WriteData(0x80); // set MSB==High, NACK then STOP  
  DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag  
  break;  
};  
case 0x30:  
{ //A STOP has been received.  
  DataRxLen=DataRxIndex;  
  DataTxLen=DataTxIndex;  
  EndFlag=1;  
  DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag  
  break;  
};  
default:  
{  
  DataTxIndex=0;  
  DataRxIndex=0;  
  EndFlag=1;  
  DrvI2C_Ctrl(0,0,0,0); // Clear all I2C flag  
  break;  
};  
}  
DrvI2C_ClearEIRQ();  
DrvI2C_ClearIntFlag(2); // Clear I2C Interrupt Flag(I2CIF)  
}  
if(DrvI2C_ReadIntFlag()==E_DRVI2C_ERROR_INT) //Get I2C Error Interrupt Flag  
{  
  EndFlag=1;  
  DrvI2C_ClearIRQ();  
  DrvI2C_ClearIntFlag(2); // Clear I2C Interrupt Flag(I2CEIF)  
  DrvI2C_Ctrl(0,1,0,0); // I2C as master sends STOP signal  
}  
}  
  
/*-----*/  
/* WDT & RTC & Timer A/B/C Interrupt Service Routines */
```



```
/*-----*/  
void HW1_ISR(void)  
{  
  
}  
  
/*-----*/  
/* HW2 ADC Interrupt Subroutines */  
/*-----*/  
void HW2_ISR(void)  
{  
    if(DrvADC_ReadIntFlag()==1) //讀取ADC插斷要求標誌位元  
    {  
        DrvADC_ClearIntFlag();  
        ADCData=DrvADC_GetConversionData()>>8;  
        ADfinish=1;  
        DrvADC_DisableInt();  
        int_00=0x30300000; //I2CIE=1, I2CEIE=1  
    }  
}  
  
/*-----*/  
/* CMP/OPA Interrupt Service Routines */  
/*-----*/  
void HW3_ISR(void)  
{  
  
}  
  
/*-----*/  
/* PT1 Interrupt Service Routines */  
/*-----*/  
void HW4_ISR(void)  
{  
  
}  
  
/*-----*/  
/* PT2 Interrupt Service Routines */
```

```

/*-----*/
void HW5_ISR(void)
{

}

/*-----*/
/* Software Delay Subroutines */
/*-----*/
void Delay(unsigned int num)
{
    for(;num>0;num--)
        asm("NOP");
}

/*-----*/
/* ADC Function */
/*-----*/
void InitalADC0(void)
{
    //Set ADC input pin
    DrvADC_SetADCInputChannel(ADC_Input_AI03,ADC_Input_AI01); //Switch to ADC_CH1 ADC input
    DrvADC_InputSwitch(OPEN);
    DrvADC_RefInputShort(OPEN);
    DrvADC_Gain(0,0); //Set the ADC Gain 1, PGA=1
    //DrvADC_Gain(1,0); //Set the ADC Gain 1, PGA=8
    DrvADC_DCOffset(0); //DC offset input voltage selection
    DrvADC_RefVoltage(AI02,AI03); //Set the ADC reference voltage. //Switch to ADC_CH1 ADC
    RefVoltage
    //DrvADC_RefVoltage(AI02,VSSA); //test only
    //DrvADC_FullRefRange(0); //Set the ADC reference range select.
    DrvADC_FullRefRange(1); //Set the ADC reference range select.
    //0:Full reference range input
    //1:1/2 reference range input
    DrvADC_OSR(8); //8:OSR=÷128, Speed=2.56k sps
    DrvADC_ClkEnable(0,1); //Setting ADC CLOCK ADCK=HS_CK/6
    DrvADC_FastChopper(1); //fast chopper enable

```

```
//Set VDDA voltage
DrvPMU_VDDA_LDO_Ctrl(E_LDO);
DrvPMU_VDDA_Voltage(E_VDDA2_4);
//DrvPMU_VDDA_Voltage(E_VDDA2_7);
//DrvPMU_VDDA_Voltage(E_VDDA3_0);
DrvPMU_BandgapEnable();

//DrvPMU_REF0_Enable();
DrvPMU_AnalogGround(Enable); //ADC analog ground source selection.
//1:Enable buffer and use internal source
//DrvPMU_LDO_LowPower(1); //VDD LDO with low power control.
//0 : Normal;1 : Low power

Delay(0x1000);

//Set ADC interrupt
DrvADC_EnableInt();
DrvADC_ClearIntFlag();
DrvADC_Enable();
DrvADC_CombFilter(Enable); //CombFilter open
}

/*-----*/
/* Hardware I2C Initial */
/*-----*/
void HWI2C_INI(void)
{

//DrvI2C_SetIOPin(0); //Set I2C port, PT1.0=SCL, PT1.1=SDA
DrvI2C_SetIOPin(7); //Set I2C port, PT2.6=SCL, PT2.7=SDA
clk_00=0xff01; //ENHA0=1

i2c_00=0xff00ff00; //0x41000//i2c off
i2c_14=0x00ffffff; //0x44014//Transmitter, Data Buffer=1

i2c_08=0xff200000; //0x41008, Set I2C CRG
i2c_00=0xff00ff01; //0x41000//ok i2c enable
i2c_08=0xff04ff7f; //0x41008//I2C Clock Control//Time-out Control Register

i2c_0c=0x00ff0020; //I2C slaver id_0=0x20
```

```
i2c_04=0xfffff80;//0x41004//i2c slaver en
i2c_0c=0x0000021;//slaver id_0
asm("NOP");

int_00=0xff30ff00; //I2CIE=1, I2CEIE

}

/*-----*/
/* End Of File */
/*-----*/
```



HY16F184_ADC_Fo
rceSensor_GD05.zip



HY16F184_I2CTool
V0.1.zip

6. 參考文獻

[1] http://www.hycontek.com/attachments/MSP/DS-HY16F198_TC.pdf, 紘康科技
HY16F198 Datasheet.

[2] http://www.hycontek.com/attachments/MSP/UG-HY16F198_TC.pdf, 紘康科技
HY16F198 User Guide.

[3] <http://www.uneotech.com/uneo/online-store/105/gs0001-4-96-uneo.html>, Uneo Force
Sensor Website.

7. 修訂記錄

以下描述本文件差異較大的地方，而標點符號與字形的改變不在此描述範圍。

日期	文件版次	頁次	摘要
2015/06/03	V01	All	1.初版發行