Gas Sensor 應用電路說明書

HY16F184

Gas Sensor Circuit

Table of Contents

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

近年來由於日趨嚴重的環境污染及工業上之需求，使得感測器的發展倍受重視。在空氣污染防治日益複雜及工業上迫切需要的今日，高效能的氣體感測器愈來愈受重視。以金屬氧化物半導體（MOS）為材料之氣體感測器，由於其耐熱性及耐蝕性佳、應答速率快、元件製作容易，以及易與微處理器組合成氣體感測系統或攜帶式監測器，因此被廣泛的使用在家庭、工廠環境中以偵測毒性氣體及燃燒爆炸性氣體。而本文將介紹以 HY16F184 內建高精密 Sigma-Delta 24 Bit ADC 搭配 CCS801 CMOS Sensor 來實現一個 Gas Sensor 應用電路。 在本文中的 Gas Sensor 應用電路上，主要的元件有：氣體感測器(CCS801 Gas Sensor)、ADC 和 MCU 控制晶片。 而紘康 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。

本文 Gas Sensor 應用電路是由紘康 HY16F184 晶片之內建 ADC 精確的量測到 CCS801 CMOS Sensor 內的 RS 電阻變化量，並且透搭配 CCS801 CMOS Sensor 所提供的 C Library 演算法，可以換算出相對應的 PPM 濃度數值。而在加熱驅動器(Heater)迴路上的微小電流變化量(RH_Current)，同樣也可使用 HY16F184 內建 ADC 精準的量測到。本文內也提供了 GUI 軟體介面，透過 I2C 通訊來輸出即時的 PPM 與 RS 和 RH_Current 資料變化量。使用 I2C 轉 USB 橋接器與電腦連接，由電腦端 GUI 做即時三個通道的資料變化量顯示。

2. 原理說明

2.1. 量測原理

CCS801 Gas sensor 半導體氣體感測材料在偵測氣體時，RS 電阻會產生變化，如下圖 1。此情況主要導因於偵測可燃性氣體如一氧化碳(CO)及多種揮發性有機化合物（VOC）與吸附在半導體氧化物且帶負電荷的氧離子產生反應。當空氣偵測到可燃性氣體時候，RS 電阻會產生變化，此時可測量到 RS 兩端的電壓會有所改變。典型的 RS 電阻值範圍在 100k~2M 歐姆之間. RH 電阻則是可當溫度反應電阻，當 Gas Sensor 有一電流迴路流經 Heater+與 Heater GND，則可視為加熱現象，隨著 Heater 溫度的變化，RH 端的電阻也會有所改變，典型的 RH 電阻值範圍在 20~100 歐姆之間。

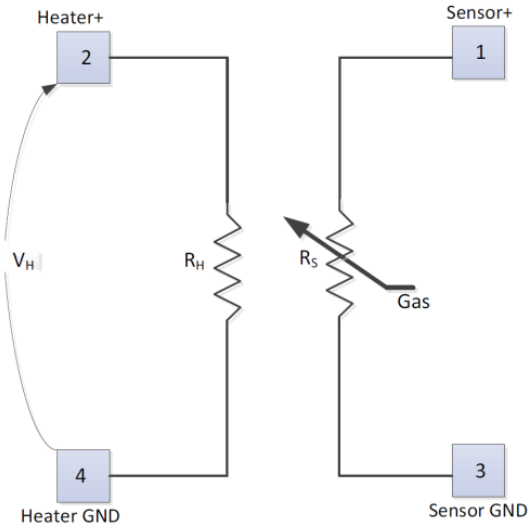

圖 1 CCS801 CMOS Sensor

2.2. Gas Sesnor 應用電路基本架構

本文 Gas Sensor 的基本架構如下圖 2 所示，包含一個氣體感測器(CCS801 Gas Sensor)、PMOS NX2301、ADC 和 MCU 單晶片。HY16F184 可輸出 PWM 來控制 PMOS NX2301 做為電流開關控制。當 PWM 輸出為 High 時候，則是關閉 PMOS，此時較為省電，不會有電流流經過 Heater 端。而當 PWM 輸出為 Low 時候，則是導通 PMOS，會有電流流經過 Heater 端，此時則開始做 Gas Sensor 加熱動作，當 Gas Sensor 再加熱的時相對來說也會比較耗電。本文的電路應用架構即是利用 PWM 來做整體消耗電流功耗控制，設定 PWM 輸出週期為 97us，PWM 輸出 Low 的時間為比 57us 而 PWM 輸出 High 的時間為 40us。PWM On 的輸出持續時間是 100ms，此時為 CCS801 的加熱時間，之後 PWM Off 的時間為持續 400ms，當 PWM Off 時候，此時會輸出保持 High，以 500ms 為一個控制週期不斷的循環控制 PMOS NX2301 開關，做為加驅動器(Heater)的控制。詳細的 PWM 控制時間圖，可以參考以下圖 3。 HY16F184 除了使用 PWM 做 PMOS 開關控制來達到功耗控制與省電的設計效果，還使用了高精度 ADC 來做 RS 與 RH_Current 變化量測量，而擷取到的資料可以由 I2C 來做資料的輸出與讀取，詳細 HY16F184 ADC 規格可以參考下圖 4。

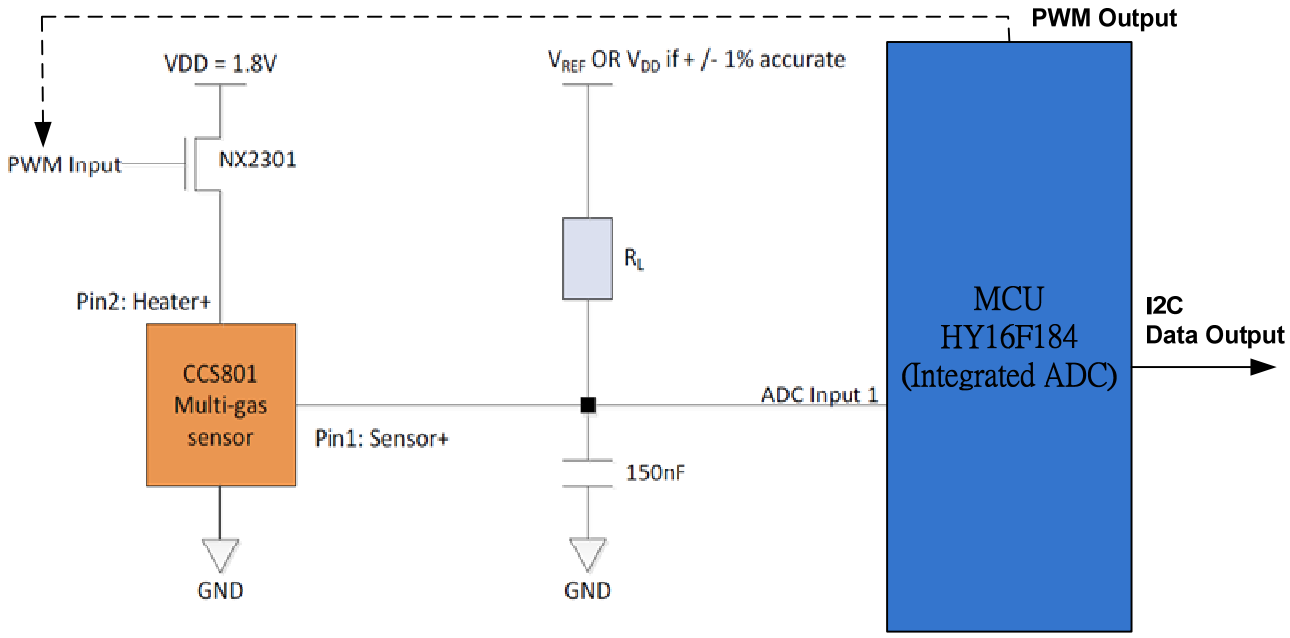

圖 2 HY16F184 Gas Sensor 基本架構圖

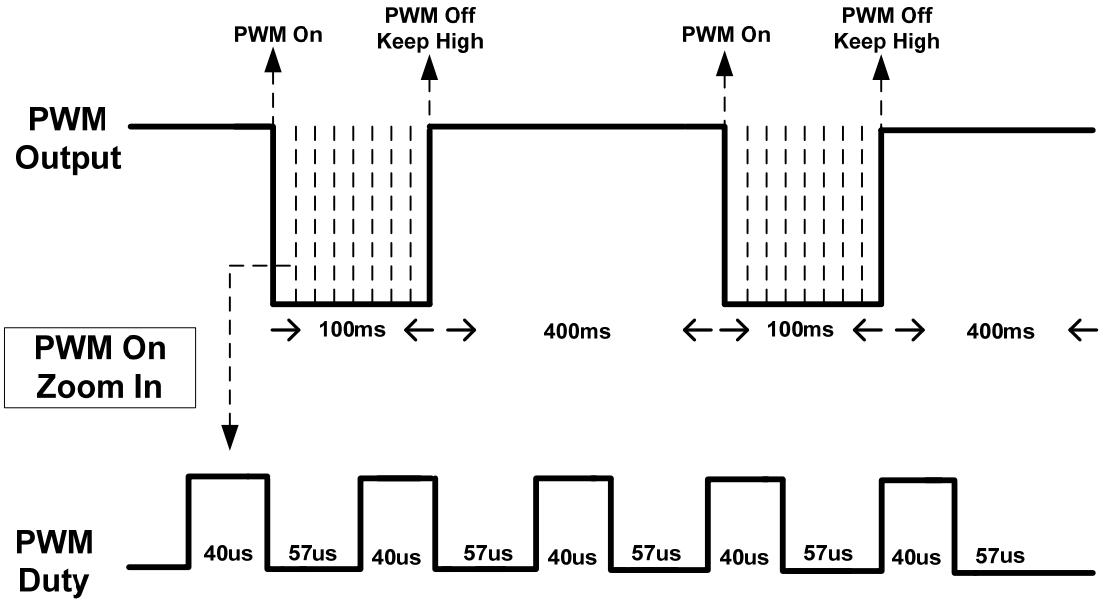

圖 3 HY16F184 PWM 輸出控制時序圖

2.3. 控制晶片

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

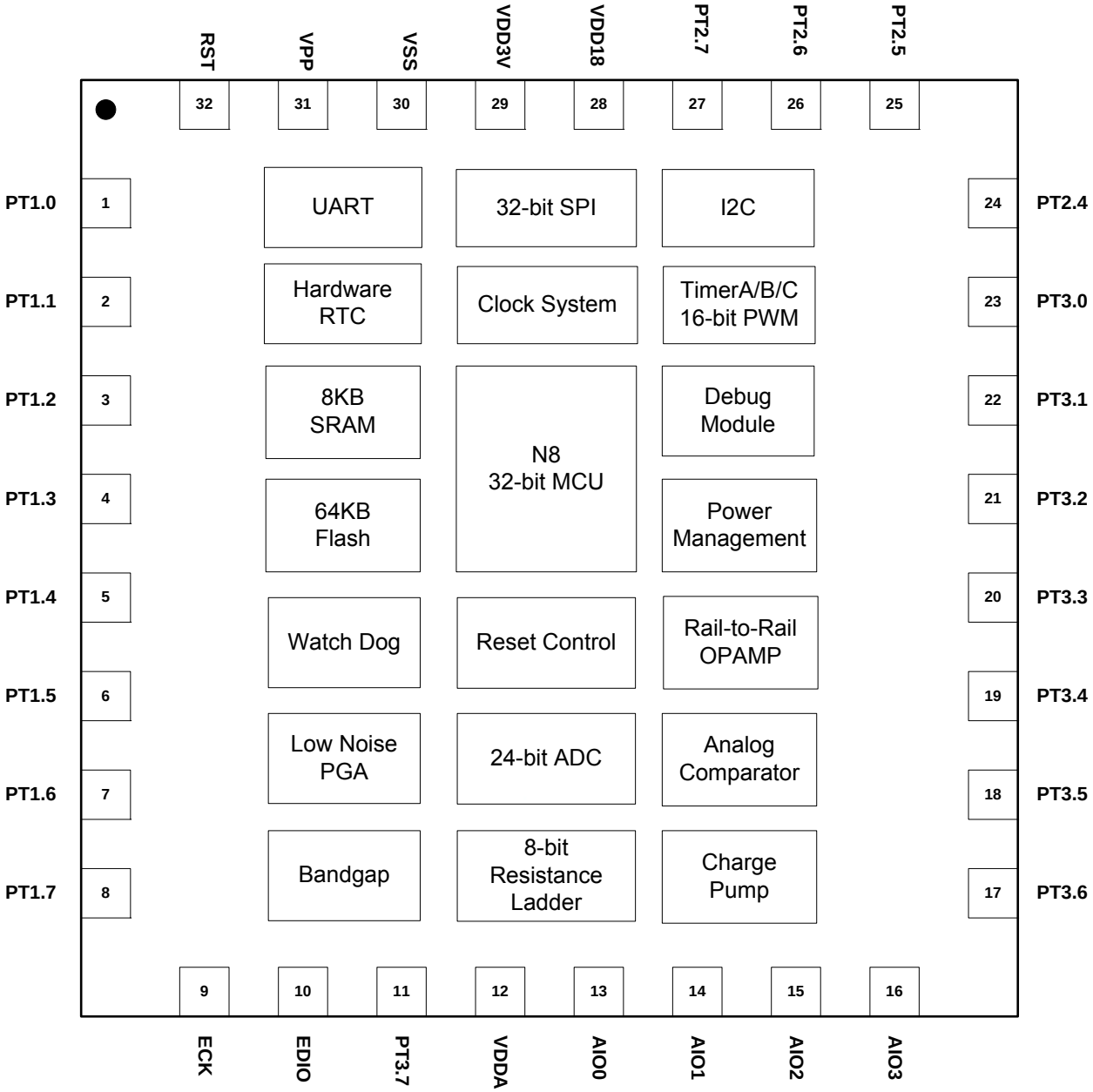

圖 4 紘康 HY16F 系列 32 位元高性能 Flash 單片機(HY16F184)

(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℃~85℃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 ΣΔADC 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	×	ADGN											
±1080	1	=	1	×	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	×	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	×	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	×	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	×	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	×	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	×	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	×	ADGN											
±1080	1	=	1	×	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	×	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	×	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	×	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	×	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	×	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	×	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 CCS801 CMOS Sensor 做 Gas Sensor 應用電路。HY16F184 的 ADC 通道類比腳位會使用到 AIO0/AIO1/AIO2/AIO3。AIO0 與 AIO3 主要負責做流經過 RH 電阻的 RH_Current 電流變化量測量，AIO1 與 AIO2 負責做 RS 電阻的電壓變化量測量，因為 RS 電阻本身為高阻抗(100k~2M 歐姆)，而 ADC 的輸入阻抗大約只有 200k 歐姆，所以在 AIO2 的輸入腳位部份會先經過 HY16F184 內建 R2ROP 做一個 Unit Gain buffer 輸出，再由 OPOI 輸出到 ADC 當作輸入參考引腳，這樣可以避免量測訊號時負載效應的產生，詳細關於 AIO1 與 AIO2 的通道設置，可以參考下圖 5。完整硬體線路圖可以參考下圖 6。

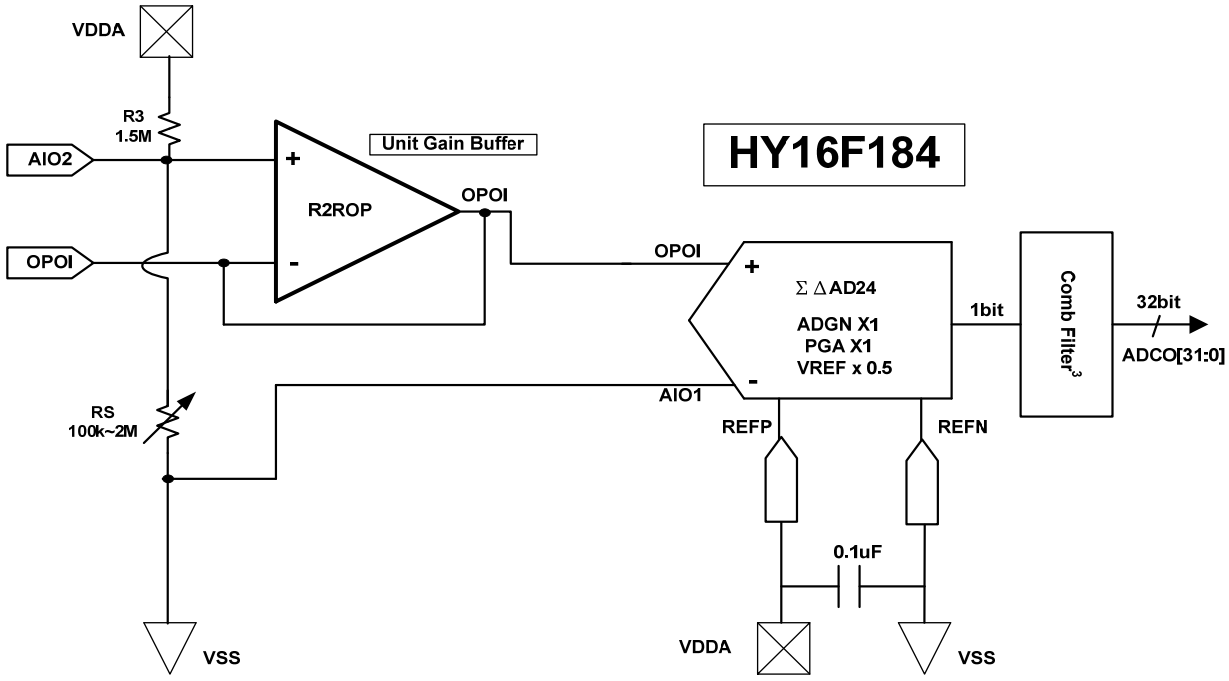

圖 5 HY16F184 ADC 通道設置 AIO2 與 AIO1

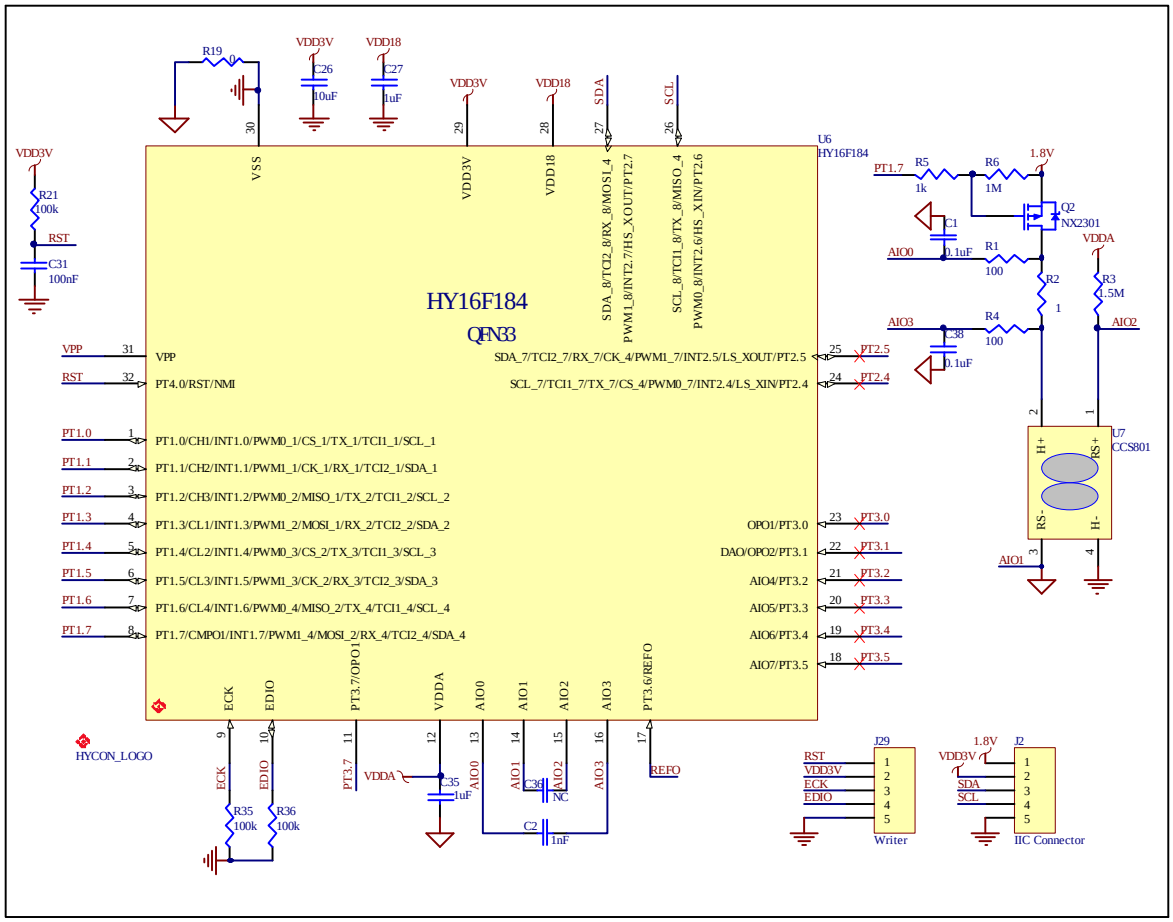

圖 6 HY16F184 Gas Sensor 硬體線路連接圖(TOP)

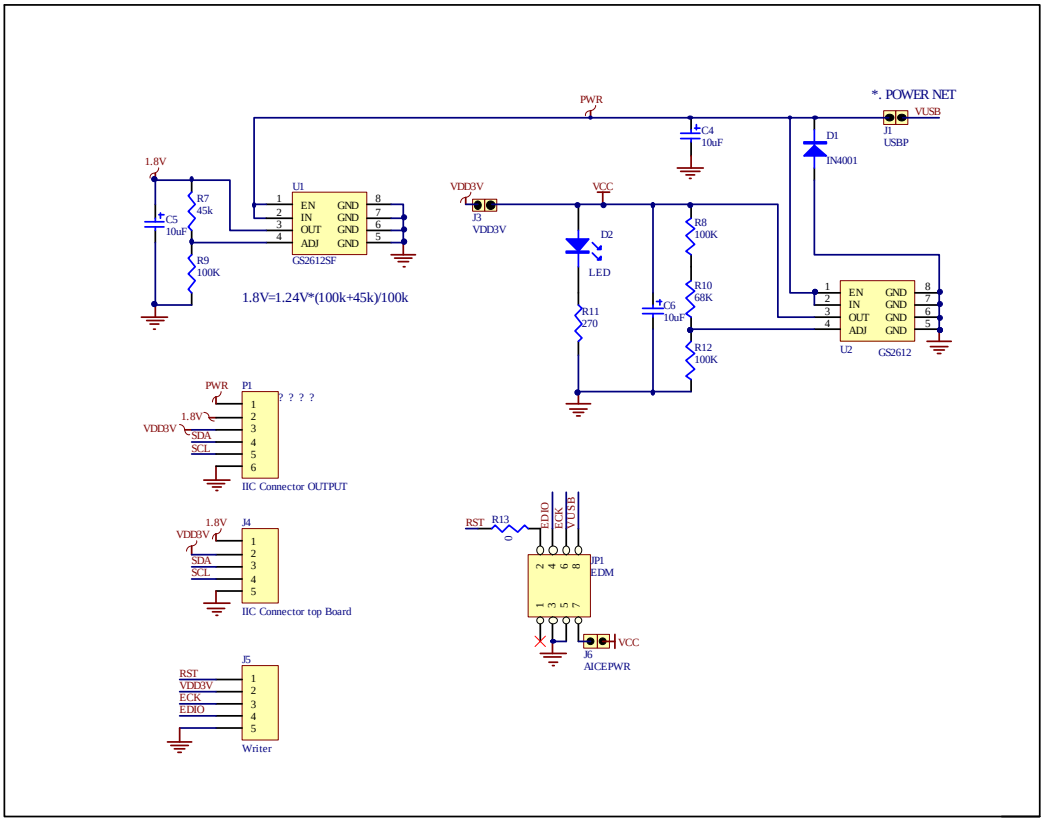

圖 7 HY16F184 Gas Sensor 硬體線路連接圖(Bottom)

主要元件介紹

(1)HY16F184： 資料處理與運算核心，主要負責執行 PWM 輸出與運算 CCS801 Gas Sensor 的 PPM, RS, RH_Current 資料，並且透過 I2C 通訊做資料輸出。

(2)ADC：HY16F184 內建之類比數位轉換器，能夠精確的將 Gas sensor 上的 RS 與 RH_Current 訊號，做類比數位電壓訊號轉換。

(3)CCS801 Gas Sensor :氣體感測器，負責偵測環境中氣體變化量，內部的 RS 電阻值會隨著氣體中可燃性氣體濃度的不同而產生變化量。

(4)NX2301 PMOS： 在此主要當作開關使用，由 HY16F184 的 PWM 來做開關控制使用。開啓時候可對 Heater 做加熱動作，關閉時候沒有電流經過不做加熱動作。

3.2. 軟體說明

程式流程圖 ：

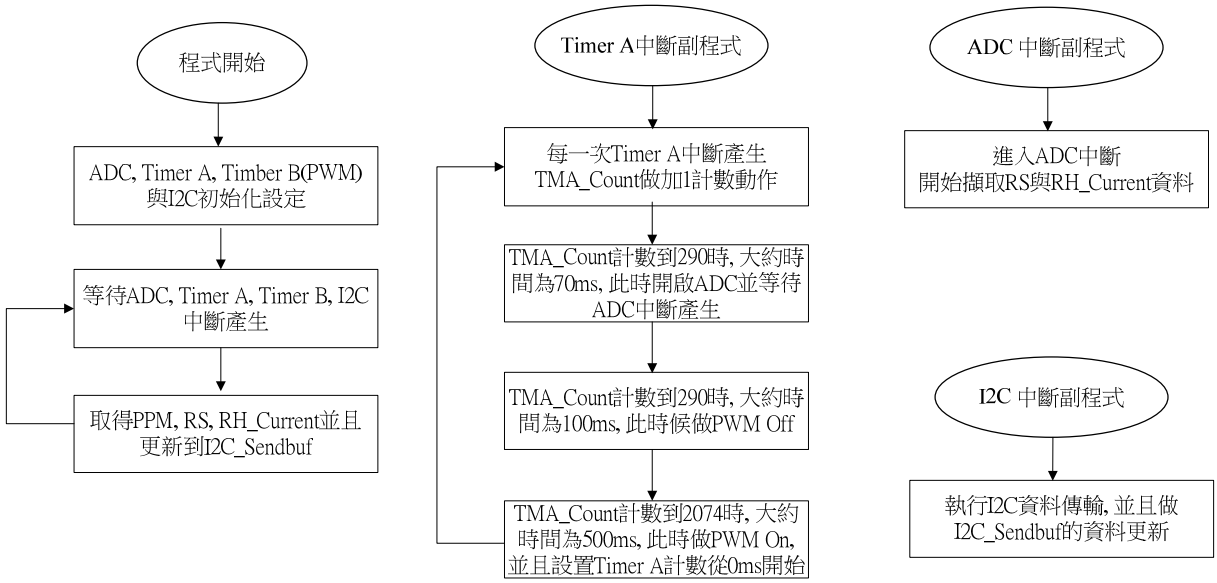

圖 8 Gas Sensor 程式流程圖

4. 數據規格與總結

4.1. 耗電流測量

在 CPU 頻率設定為 2MHz 與工作電壓 VDD=3V, VDDA=2.4V, 使用 PWM 做 PMOS 開關控制. PWM On 的輸出持續時間是 100ms，此時為 CCS801 的加熱時間，之後 PWM Off 的時間為持續 400ms，以 500ms 為一個控制週期不斷的循環控制 PMOS，在此情況下所測得到的耗電流約 0.89mA。

4.2. ADC Raw Data 與 I2C 通訊格式說明

I2C Slave Address:0x20

I2C Command:0x80

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: RS 的 ADC RawData 經過 ccsmox_set_data_iaq 計算後所得 PPM

CH2: RS 端的 ADC RawData

CH3: RH_Current 端的 ADC RawData

CH4: NC

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

每個通道數據(Chx)共 8*3=24bit

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

4.3. ADC Raw Data 資料顯示介面介紹

掃描 Gas 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 Gas sensor 掃描資料。

4.Scan: 開始讀取四個通道的 Gas sensor 掃描資料。

5.Save: 存取四個通道的 Gas sensor 掃描資料。

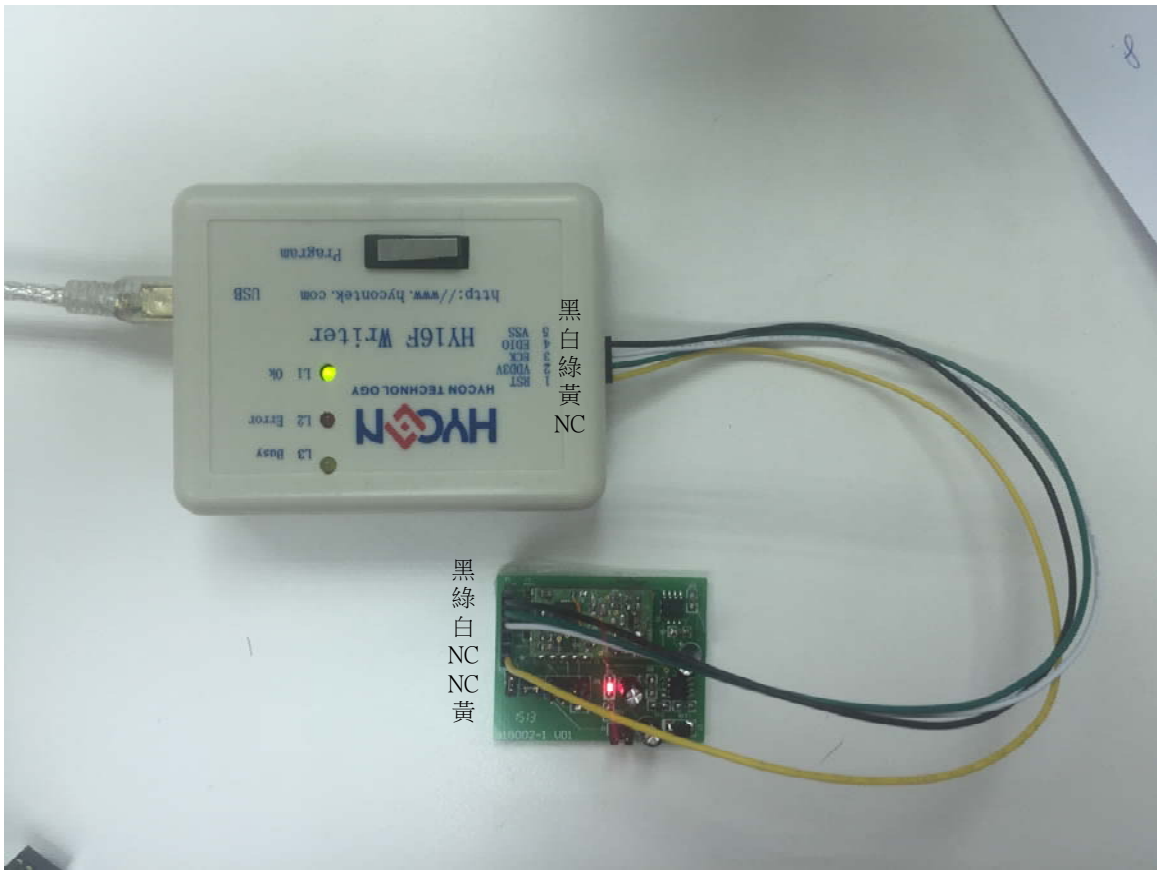

圖 9 HY16F184 Gas Sensor 與 USB 轉 I2C Board 硬體接線圖

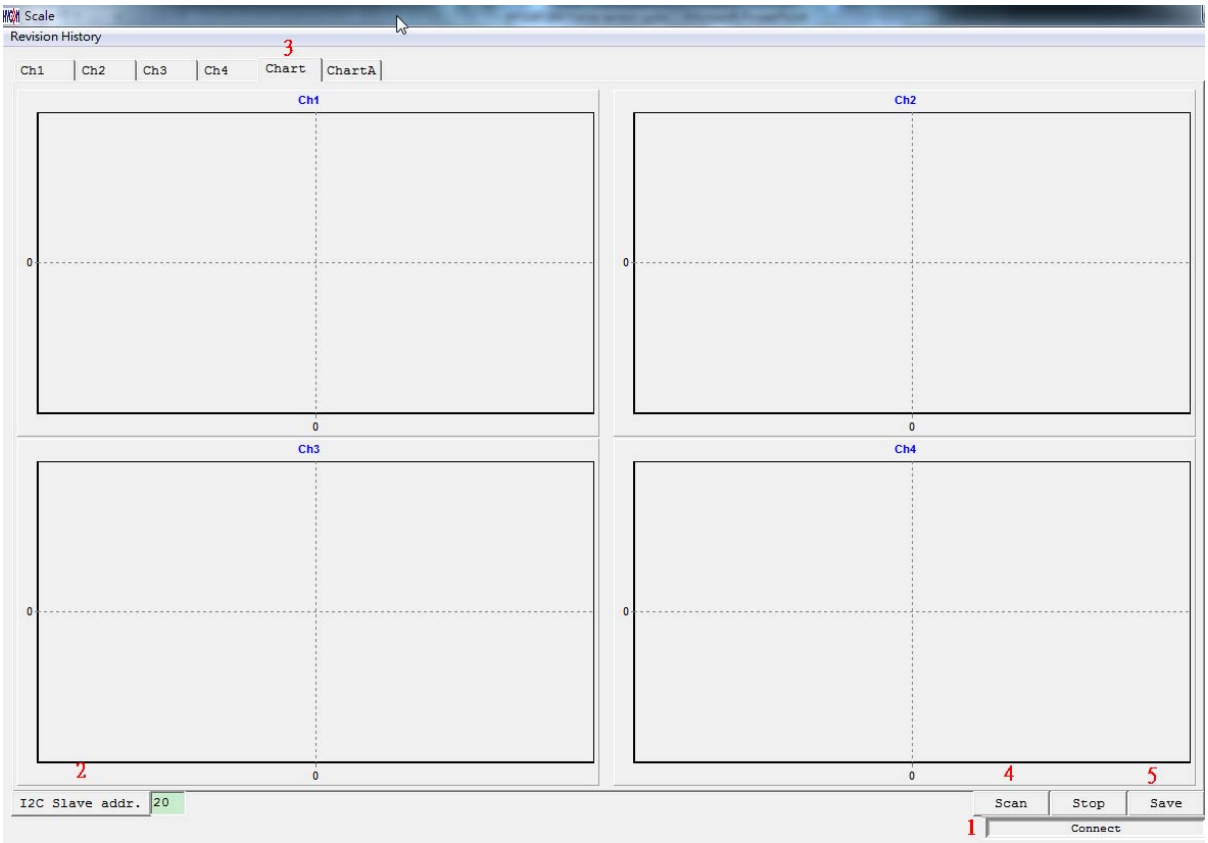

圖 10 ADC Raw Data 資料顯示介面

連上 GUI 觀察 CH1~CH3 的資料顯示畫面如下圖，因為 PWM On 與 PWM Off 的周期為 500ms， 所以換算頻率約每 2 秒在 GUI 畫面做一次資料的更新。CH1 的資料為氣體濃度 PPM，當 Gas Sensor 沒有偵測到任何的可燃性氣體時候，可以看到都為平穩數值不會跳動，CH2 的資料為 RS，一開始的 ADC Raw Data 會呈現平穩的上升，CH3 則為 RH_Current 資料，一開始的 ADC Raw data 會呈現平穩的下降。

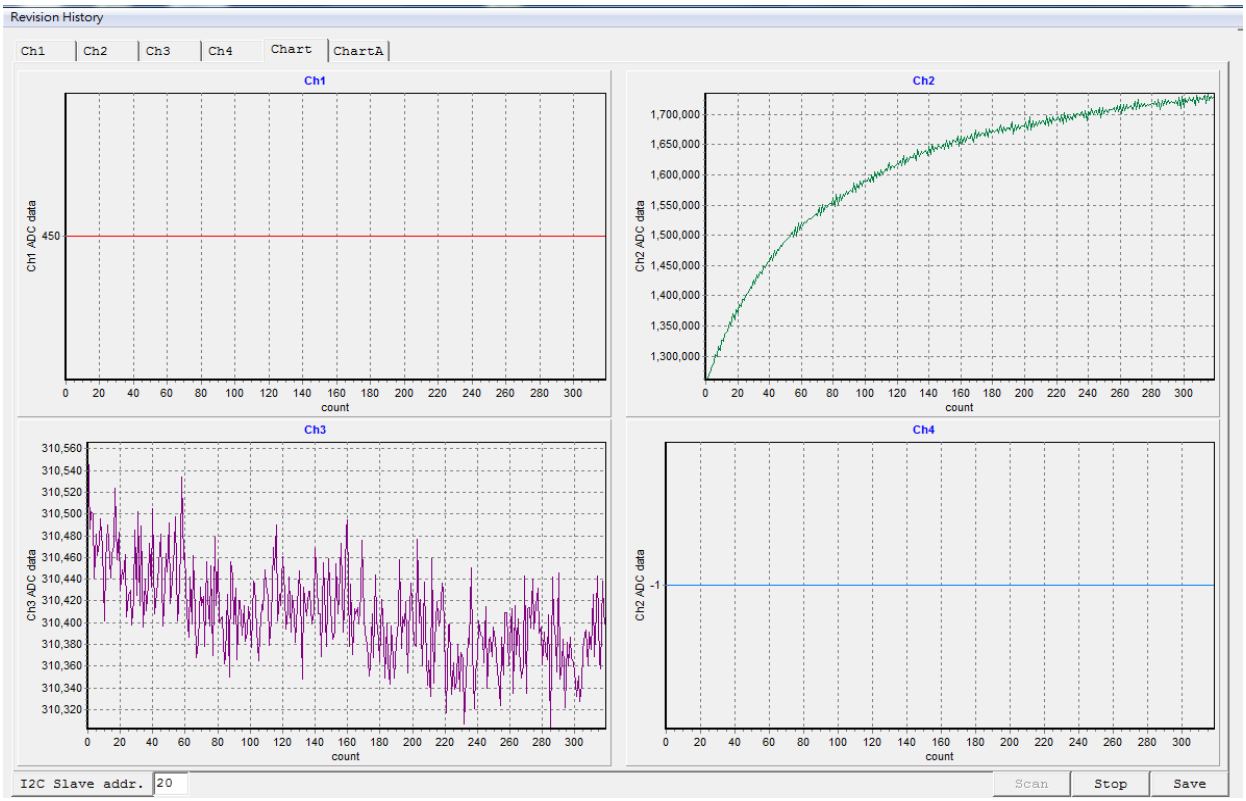

圖 11 Gas Sensor 無偵測到任何可燃性氣體時候的 ADC Raw Data 變化量

當有揮發性有機化合物接近 Gas Sensor 時候，在此是拿奇異筆接近 Gas Sesnor，可以看到 CH1 的 PPM 數據會透過演算法算出變化量，同時 CH2 的 RS ADC Raw data 會呈現明顯下降狀態，CH3 的 RH_Current ADC Raw data 也會呈現明顯下降狀態，在奇異筆離開 Gas Sensor 之後才會慢慢回復平穩狀態。

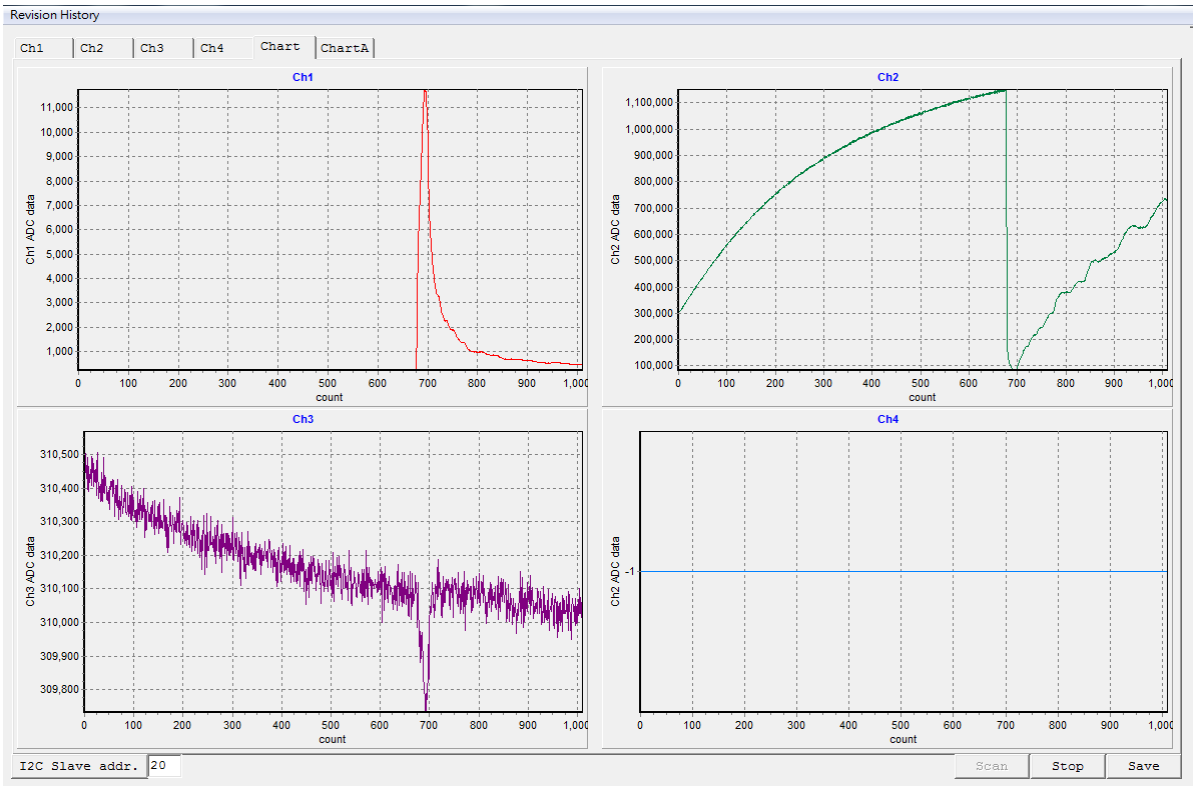

圖 12 Gas Sensor 偵測到可燃性氣體時候的 ADC Raw Data 變化量

4.4. 總結

在本文中，提供了完整的 Gas Sensor 相關應用與開發工具供使用者參考，使用者可以依據三個通道 PPM, RS, RH_Current 的 ADC Raw Data 變化量，來做後續功能設計與開發。

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 "ccs_types.h"
#include "libCcsMox_v6.h"
#include "DrvADC.h"
#include "DrvTimer.h"
#include "stdlib.h"
#include "DrvPMU.h"
#include "DrvOP.h"
#include "DrvREG32.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 PWM_ADC_StartTime 207 //50ms Start measuring ADC
#define PWM_ADC_StartTime 290 //70ms Start measuring ADC
//#define PWM_ADC_StartTime 373 //90ms Start measuring ADC
#define PWM_Heater_FinishTime 414 //100ms
//#define PWM_Heater_FinishTime 621 //150ms
//#define PWM_Heater_FinishTime 828 //200ms
#define PWM_PeriodTime 2074 //500ms
#define I2CBufferSize 256

/*---------------------------------------------------------------------------*/
/* Global CONSTANTS                                                          */
/*---------------------------------------------------------------------------*/
MCUSTATUS  MCUSTATUSbits;

unsigned int TMA_Count;
unsigned char ADfinish;
unsigned char ADC_Count;
unsigned char I2C_Done;
char RS_LowByte, RH_LowByte, PPM_LowByte;
char RS_MiddleByte, RH_MiddleByte, PPM_MiddleByte;
char RS_HighByte, RH_HighByte, PPM_HighByte;
int RS,RH_Current;
int PPM;
int ADCData;
int i;

// Constants for I2C
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 ADchange(void);
void InitalADC(void);
void HWI2C_INI(void);
```

```
/*---------------------------------------------------------------------------*/
/* Main Function                                                             */
/*---------------------------------------------------------------------------*/
int main(void)
{

  struct CcsValue value = {{0,0},{0,0,0,0},0,0,0};  //Initial all data =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; //Initial I2C data buffer=0
  }

  ADC_Count=0;
  TMA_Count=0;
  I2C_Done=0;
  ADfinish=0;
  RS_LowByte=0, RH_LowByte=0, PPM_LowByte=0;
  RS_MiddleByte=0, RH_MiddleByte=0, PPM_MiddleByte=0;
  RS_HighByte=0, RH_HighByte=0, PPM_HighByte=0;

  HWI2C_INI();
  InitalADC();
  SYS_EnableGIE(7,0x3F);  // Enable GIE(Global Interrupt)

  DrvGPIO_Open(E_PT1,0x80,E_IO_OUTPUT);  //PT1.7 set Output, PWM1 Output pin
  //DrvGPIO_Open(E_PT1,0x40,E_IO_OUTPUT);  //PT1.6 set Output, check ADC measure time
  DrvTMBC_Clk_Source(0,0);  //TimerB Clock enable HS_CK, PerScale=1
  DrvTMB_Open(E_TMB_MODE0,E_TMB_NORMAL,0x00c8);  //TimerB overflow 0Xc8->PWM period 0xc8
  DrvPWM1_Open(0,1,3);  //PWM1 Enable, Port 1.7=PWMO1(Output PWM signal, PWM frequency=97us, Low=57us,
High=40us
  DrvPWM_CountCondition(0xffff,0x0078);                  //To control PMOS, PWM1 Duty 0x78, PWM0 Duty
0xffff(PWM0 No use)
  DrvTMA_Open(3,0);                   //TimerA Overflow, TMA Speed, 250us
                                     //3:taclk/16, if ENTAD=1, taclk=32
                                     //00:HS_CK
  DrvTIMER_ClearIntFlag(E_TMA);        //Clear Timer A interrupt flag
  DrvTIMER_EnableInt(E_TMA);           //Timer A interrupt enable

  //Following to do Volatile Organic Compounds VOC calculation
```

```
struct CcsConf conf = { TYPE_VOC, sizeof(CCS_PROFILE_VOC), (uintptr_t)&CCS_PROFILE_VOC};
ccsmox_set_conf(CONF_SET_PROFILE, (uintptr_t)&conf, sizeof(conf));

while(1)
{
  if(ADfinish==1)
   {
     ADfinish=0;
     RH_Current=RH_Current>>(3);//RH做4筆chopper, 所以要有/8動作.
     value.ref.value = RS;
     ccsmox_set_data_iaq( (uintptr_t)&value, sizeof(value) );
     PPM=value.concentration & 0xffff;

     PPM_LowByte=(PPM << 1) & 0xFF;
     PPM_MiddleByte=(PPM >> 7) & 0xFF;
     PPM_HighByte=(PPM >> 15) & 0xFF;
     RS_LowByte=(RS >> 0) & 0xFF;
     RS_MiddleByte=(RS >> 8) & 0xFF;
     RS_HighByte=(RS >> 16) & 0xFF;
     RH_LowByte=(RH_Current >> 0) & 0xFF;
     RH_MiddleByte=(RH_Current >> 8) & 0xFF;
     RH_HighByte=(RH_Current >> 16) & 0xFF;

     I2C_Sendbuf[13]=PPM_LowByte;
     I2C_Sendbuf[14]=PPM_MiddleByte;
     I2C_Sendbuf[15]=PPM_HighByte;
     I2C_Sendbuf[16]=RS_LowByte;
     I2C_Sendbuf[17]=RS_MiddleByte;
     I2C_Sendbuf[18]=RS_HighByte;
     I2C_Sendbuf[19]=RH_LowByte;
     I2C_Sendbuf[20]=RH_MiddleByte;
     I2C_Sendbuf[21]=RH_HighByte;
     I2C_Sendbuf[22]=0xff;
     I2C_Sendbuf[23]=0xff;
     I2C_Sendbuf[24]=0xff;
     I2C_Done=1;  //Enable I2C Slave data Buffer Update
   }

}
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(I2C_Done==1)
                {
                   I2C_Sendbuf[1]=I2C_Sendbuf[13]|0x1; //bit0=1b, new data
                I2C_Sendbuf[2]=I2C_Sendbuf[14];
                I2C_Sendbuf[3]=I2C_Sendbuf[15];
                I2C_Sendbuf[4]=I2C_Sendbuf[16]|0x1;  //bit0=1b, new data
                I2C_Sendbuf[5]=I2C_Sendbuf[17];
                I2C_Sendbuf[6]=I2C_Sendbuf[18];
                I2C_Sendbuf[7]=I2C_Sendbuf[19]|0x1;  //bit0=1b, new data
                I2C_Sendbuf[8]=I2C_Sendbuf[20];
                I2C_Sendbuf[9]=I2C_Sendbuf[21];
                I2C_Sendbuf[10]=I2C_Sendbuf[22]|0x1;  //bit0=1b, new data
                I2C_Sendbuf[11]=I2C_Sendbuf[23];
```

```
            I2C_Sendbuf[12]=I2C_Sendbuf[24];
            I2C_Done=0;
            }
            else
            {
                 I2C_Sendbuf[1]=(I2C_Sendbuf[1]&0xFE);  //bit0=0b, old data
            I2C_Sendbuf[2]=I2C_Sendbuf[2];
            I2C_Sendbuf[3]=I2C_Sendbuf[3];
            I2C_Sendbuf[4]=(I2C_Sendbuf[4]&0xFE);  //bit0=0b, old data
            I2C_Sendbuf[5]=I2C_Sendbuf[5];
            I2C_Sendbuf[6]=I2C_Sendbuf[6];
            I2C_Sendbuf[7]=(I2C_Sendbuf[7]&0xFE);  //bit0=0b, old data
            I2C_Sendbuf[8]=I2C_Sendbuf[8];
            I2C_Sendbuf[9]=I2C_Sendbuf[9];
            I2C_Sendbuf[10]=(I2C_Sendbuf[10]&0xFE);  //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)
{
  DrvTIMER_ClearIntFlag(E_TMA);        //Clear TMA interrupt flag
  TMA_Count++;

  if(TMA_Count==PWM_ADC_StartTime)  //If TMA count to PWM_ADC_StartTime, Start Measuring ADC
  {
     //DrvGPIO_SetPortBits(E_PT1,0x40);  //check ADC start Time, PT1.6=high
     //DrvGPIO_ClrPortBits(E_PT1,0x40);  //Check ADC start Time, PT1.6=low
    DrvADC_Enable();
```

```
    DrvADC_CombFilter(ENABLE);
  }


  if(TMA_Count==PWM_Heater_FinishTime) //If TMA count to PWM_Heater_FinishTime, PT1.7 set High, PMOS
off
  {
    DrvPWM1_Close();
    DrvGPIO_SetPortBits(E_PT1,0x80);  //PT1.7 set High, PMOS Off
  }

  if(TMA_Count>=PWM_PeriodTime)  //If TMA count to PWM_PeriodTime, repeat to do PWM output
  {
    TMA_Count=0;    // Let TMA count=0=0ms,
    DrvPWM1_Open(0,1,3);  //(PWM output signal, PWM frequency=97us, Low=57us, High=40us
                          //PWM Output signal from the time ( 0ms ~ PWM_Heater_FinishTime )
  }

}

/*---------------------------------------------------------------------------*/
/* HW2 ADC Interrupt Subroutines                                             */
/*---------------------------------------------------------------------------*/
void HW2_ISR(void)
{

  if(DrvADC_ReadIntFlag()) //讀取ADC插斷要求標誌位元
  {
    DrvADC_ClearIntFlag();
    ADCData=DrvADC_GetConversionData()>>8;  //get +/-23bit ADO
     ADchange();  //get ADC data, RS and RH data
     ADC_Count++;
     if(ADC_Count>=9)
     {
       //If DrvADC_OSR(5), spend 44ms, (6):24ms, (7):12ms, (8):7.5ms
       //DrvGPIO_SetPortBits(E_PT1,0x40);  //check ADC Finish Time, PT1.6=high
       //DrvGPIO_ClrPortBits(E_PT1,0x40);  //check ADC Finish Time, PT1.6=low
       ADC_Count=0;
      }
  }
```

```
}

/*---------------------------------------------------------------------------*/
/* 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)
{

}

/*---------------------------------------------------------------------------*/
/* Get ADchange Data                                                         */
/*---------------------------------------------------------------------------*/
void ADchange(void)
{
     switch(ADC_Count)
     {
       case 0:
       {
          RS=ADCData;
          DrvADC_CombFilter(DISABLE);
          DrvADC_SetADCInputChannel(ADC_Input_AIO0,ADC_Input_AIO3);
          DrvADC_Gain(ADC_PGA_Disable,ADC_Gain_32);
```

```
        DrvADC_CombFilter(ENABLE);
        RH_Current=0;
        break;
    }

    case 1:
    {
        RH_Current=ADCData;
        break;
    }



    case 2:
    {
        RH_Current+=ADCData;
        break;
    }

    case 3:
    {
        RH_Current+=ADCData;
        break;
    }

    case 4:
    {
        RH_Current+=ADCData;
        DrvADC_CombFilter(DISABLE);
        DrvADC_SetADCInputChannel(ADC_Input_AIO3,ADC_Input_AIO0);
        DrvADC_CombFilter(ENABLE);
        break;
    }

    case 5:
    {
        RH_Current-=ADCData;
        break;
    }

    case 6:
```

```
        {
            RH_Current-=ADCData;
            break;
        }

        case 7:
        {
            RH_Current-=ADCData;
            break;
        }

        case 8:
        {
            RH_Current-=ADCData;
            DrvADC_Disable();
            DrvADC_CombFilter(DISABLE);
            DrvADC_SetADCInputChannel(OPO,ADC_Input_AIO1);
            DrvADC_Gain(ADC_PGA_Disable,ADC_PGA_Disable);
            ADfinish=1;
            break;
       }

        default:break;

      }
}

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

  DrvI2C_SetIOPin(7);  //Set I2C port, PT2.6=SCL, PT2.7=SDA
  clk_00=0xff01;  //ENHAO=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=0xffff80;//0x41004//i2c slaver en
  i2c_0c=0x00000021;//slaver id_0
  asm("NOP");
  int_00=0xff30ff00;  //I2CIE=1, I2CEIE

}

/*---------------------------------------------------------------------*/
/* Hardware ADC Initial                                                */
/*---------------------------------------------------------------------*/
void InitalADC(void)
{

  //Set ADC input pin
  DrvADC_SetADCInputChannel(OPO,ADC_Input_AIO1); //Set the ADC positive=PT3.7/negative=AIO1 input
voltage source.
  DrvADC_InputSwitch(OPEN);              //ADC signal input (positive and negative) short(VISHR)
control.
  DrvADC_RefInputShort(OPEN);             //Set the ADC reference input (positive and negative)
short(VRSHR) control.

  DrvADC_Gain(ADC_PGA_Disable,ADC_PGA_Disable); //Input signal gain for modulator, Set the ADC Gain
1, PGA=1
  DrvADC_DCoffset(0);                    //DC offset input voltage selection (VREF=REFP-REFN)
  DrvADC_RefVoltage(VDDA,0);                  //Set the ADC reference voltage. (VDDA-VSSA)
  DrvADC_FullRefRange(1);                //Set the ADC full reference range select.
                                        //0: Full reference range input
                                        //1: 1/2 reference range input
  //For CCS801 CMOS Sensor, ADC acquisition rate at least 1kHz
  DrvADC_OSR(7);               //7 : ÷256==1280sps
  //DrvADC_OSR(8);            //8:OSR=÷128, Speed=2.56k sps
  DrvADC_ClkEnable(0,1);       //Setting ADC CLOCK ADCK=HS_CK/6 & Rising edge is high
  //DrvADC_FastChopper(1);     //fast chopper enable

  //Set VDDA voltage
  DrvPMU_VDDA_LDO_Ctrl(E_LDO);
```

```
  DrvPMU_VDDA_Voltage(E_VDDA2_4);
  DrvPMU_BandgapEnable();
  //DrvPMU_REFO_Enable();
  DrvPMU_REFO_Disable();
  DrvPMU_AnalogGround(ENABLE);  //ADC analog ground source selection.
                                //1 : Enable buffer and use internal source(need to work with ADC)
  Delay(1000);

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

  //Set OPA input pin
  DrvOP_Open();
  DrvOP_PInput(0x01);  //Selection the Rail-to-rail OPAMP positive input of AIO2
  DrvOP_NInput(0x08);  //Selection the Rail-to-rail OPAMP negative input of OPOI
  //DrvOP_OPOoutEnable();

}

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

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


HY16F184_ADC_Gas_Sensor.zip


HY16F184_I2CTool V0.1.zip

6. 參考文獻

[1] http://www.hycontek.com/attachments/MSP/DS-HY16F188_TC.pdf, 紘康科技 HY16F188 Datasheet.
[2] http://www.hycontek.com/attachments/MSP/UG-HY16F188_TC.pdf, 紘康科技 HY16F188 User Guide.
[3] http://www.ccmoss.com/products/ccs801, CCS801 CMOS 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
2015/06/26	V01	All	1.初版發行