



HY16F3981

红外线测温应用说明书

**Infrared Thermopile
Measurement Application Note**

Table of Contents

1. 内容简介	4
2. 原理说明	5
2.1. 红外线波长介绍	5
2.2. 量测光点与视窗	5
2.3. 传感器简介	6
2.4. 控制芯片	8
3. 系统设计	10
3.1. 硬件说明	10
3.2. 软件说明	14
4. GUI 软件操作	15
4.1. 硬件连接说明	15
4.2. 软件显示界面	16
4.3. 软件操作说明	17
4.4. 总结	19
5. Demo Code 及相关档案	20
6. 参考文献	48
7. 修订记录	49

注意：

- 1、本说明书中的内容，随着产品的改进，有可能不经过预告而更改。请客户及时到本公司网站下载更新 <http://www.hycontek.com>。
- 2、本规格书中的图形、应用电路等，因第三方工业所有权引发的问题，本公司不承担其责任。
- 3、本产品单独应用的情况下，本公司保证它的性能、典型应用和功能符合说明书中的条件。当使用在客户的产品或设备中，以上条件我们不作保证，建议客户做充分的评估和测试。
- 4、请注意输入电压、输出电压、负载电流的使用条件，使 IC 内的功耗不超过封装的容许功耗。对于客户在超出说明书中规定额定值使用产品，即使是瞬间的使用，由此所造成的损失，本公司不承担任何责任。
- 5、本产品虽内置防静电保护电路，但请不要施加超过保护电路性能的过大静电。
- 6、本规格书中的产品，未经书面许可，不可使用在要求高可靠性的电路中。例如健康医疗器械、防灾器械、车辆器械、车载器械及航空器械等对人体产生影响的器械或装置，不得作为其部件使用。
- 7、本公司一直致力于提高产品的质量和可靠度，但所有的半导体产品都有一定的失效概率，这些失效概率可能会导致一些人身事故、火灾事故等。当设计产品时，请充分留意冗余设计并采用安全指标，这样可以避免事故的发生。
- 8、本规格书中内容，未经本公司许可，严禁用于其他目的之转载或复制。

1. 内容简介

常见的红外线传感器应用可分为医疗、工业、消费性用途，如耳温枪、额温枪、工业用温度仪、红外线温度计...等。在耳温枪应用中，需注意红外线传感器进入耳中的升温效应、导波管与传感器的连接方式...等。在红外线温度计应用中，需注意待测物距离、透镜的聚焦距离...等。

HY16F3981 芯片内建仪表放大器(Instrumentation Amplifier)具有高输入阻抗特性与 Sigma-delta 24 Bit ADC 具有高精度讯号量测特性，因此 HY16F3981 非常适用于高输入阻抗与小电压讯号(uV)的量测应用，典型的应用为红外线测温。本文介绍使用 HY16F3981 芯片搭配红外线传感器(IR Sensor)完成红外线测温应用。

2. 原理说明

2.1. 红外线波长介绍

依红外线的波长范围和红外线辐射源可区分：

近红外线(Near Infra-red, NIR)；700~2,000nm

中红外线(Middle Infra-red, MIR)；3,000~5,000nm

远红外线(Far Infra-red, FIR)；8,000~14,000nm

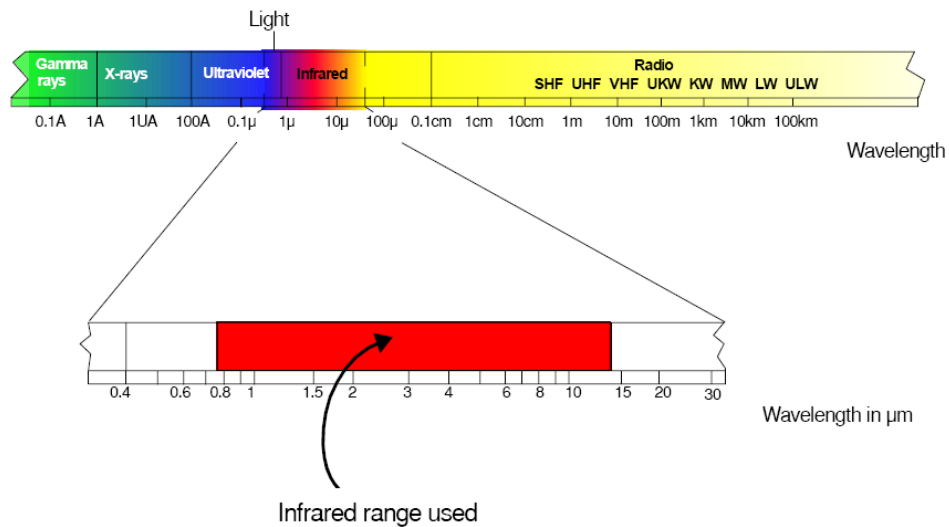


图 1 波长频谱图，红外线常用的量测波长范围：700nm~14000nm

2.2. 量测光点与视窗

在光学系统中，量测目标物的能量唯一关键因素，因此目标物必须完全包含着光点，能量与量测的距离、光点面积大小存在着一定的关系，距离越远侦测器接受到的能量越小，光点面积越大。

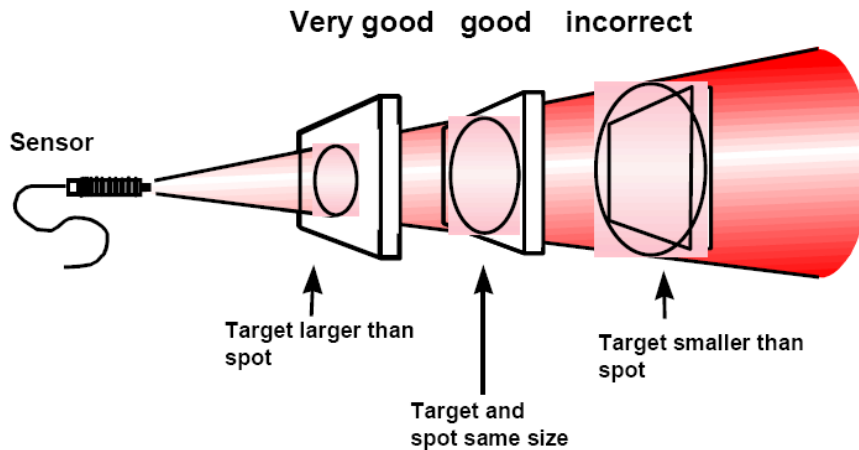


图 2 目标物必须完全包含着光点，否则量测数据会错误

2.3. 传感器简介

红外线传感器(IR sensor)简介:

由两个界面元件组成，分别为 Thermopile 及 Thermistor，两者封装如图。

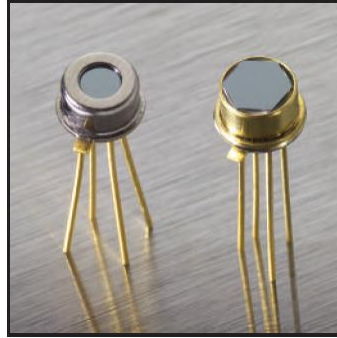


图 3 红外线温度传感器

Thermopile :

输出小电压，其电压值由目标物温度及 Thermopile 所在环境温度决定((图))。建议分辨率 0.01°C的精准度。使用于环温 25°C校正时绝对误差在±0.03°C内。其为半导体材料制作而成的传感器，故容易受温度而影响其测量数值。良好的 IR Sensor 其

Thermopile 的数学模式如下

$$\begin{aligned}
 V_{out} &= K \times [(T_t + 273.13)^4 - (T_a + 273.13)^4] \\
 &= K \times f(T_t, T_a) \quad \dots\dots\dots\text{公式(1)} \\
 &= K \times [f(T_t, T_{ref}) - f(T_a, T_{ref})]
 \end{aligned}$$

Vout : Thermopile 输出电压

K : Sensitivity of Thermopile

Tt : Target Temperature (°C)

Ta : Ambient Temperature (°C)

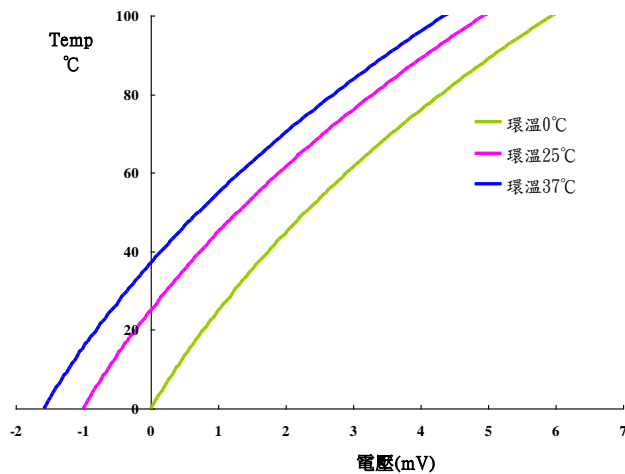


图 4 Thermopile 电压与温度曲线

Good Thermopile 会有以下两个特性

- $T_t = T_a$ 时, $V_{out} = 0$
- K 为常数, 不随环境温度而改变

Thermistor :

随其所在之温度改变而有电阻变化(图), 用来监视 IR sensor 内部温度。在此亦称之为量测时的环境温度。建议测量误差及重复性 $< 0.05^\circ\text{C}$ 。

Thermistor 的数学模式如下

$$R_{th}(T) = R_{25} \times e^{\{B \times [(\frac{1}{T+273.13}) - (\frac{1}{25+273.13})]\}} \dots\dots\dots \text{公式(2)}$$

$R_{th}(T)$: Thermistor 变化电阻值

B : Sensitivity of Thermistor

R_{25} : 25°C 电阻值

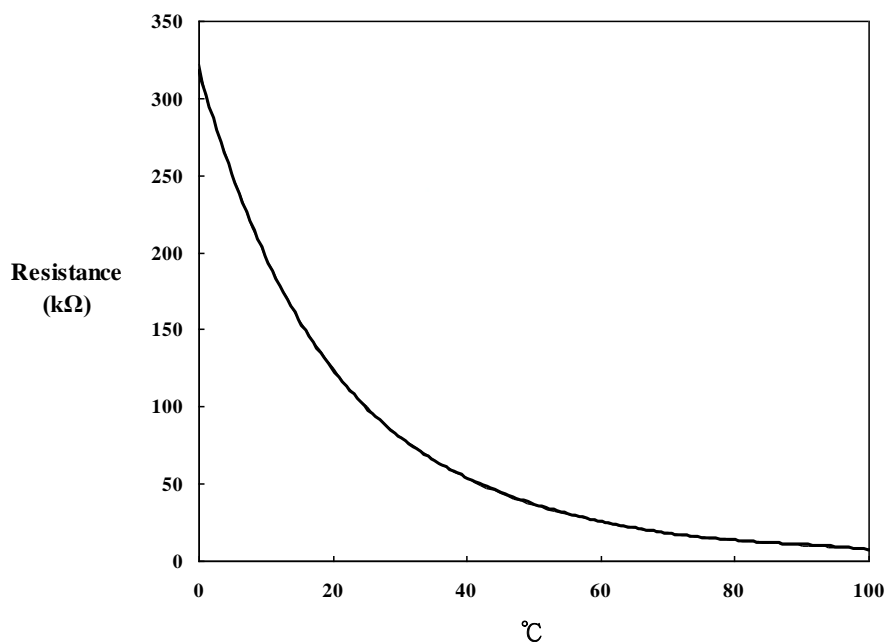


图 5 Thermistor 电阻与温度曲线

2.4. 控制芯片

利用红外线传感器(IR Sensor)做为温度讯号的撷取来源，转换为电阻与小电压讯号(uV)，经由 HYCON 的芯片 HY16F3981 量测讯号，做运算与数位讯号输出，显示目标温度，以最少元件达成红外线量测方案，(如下图 6)。

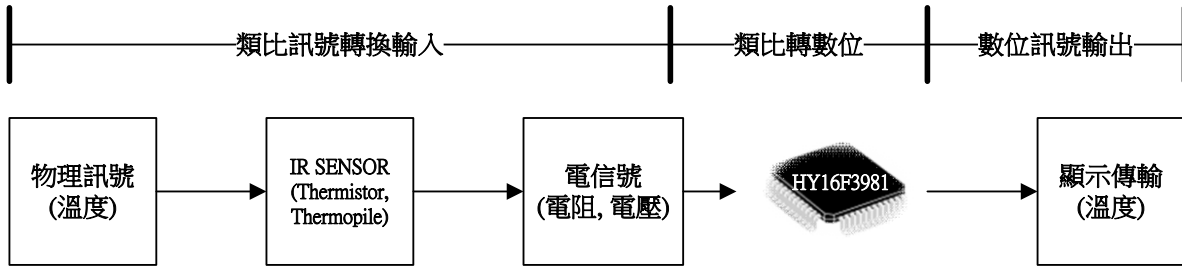


图 6 HY16F3981 红外线传感器量测方案, 类比与数位讯号转换

HY16F3981 芯片简介：

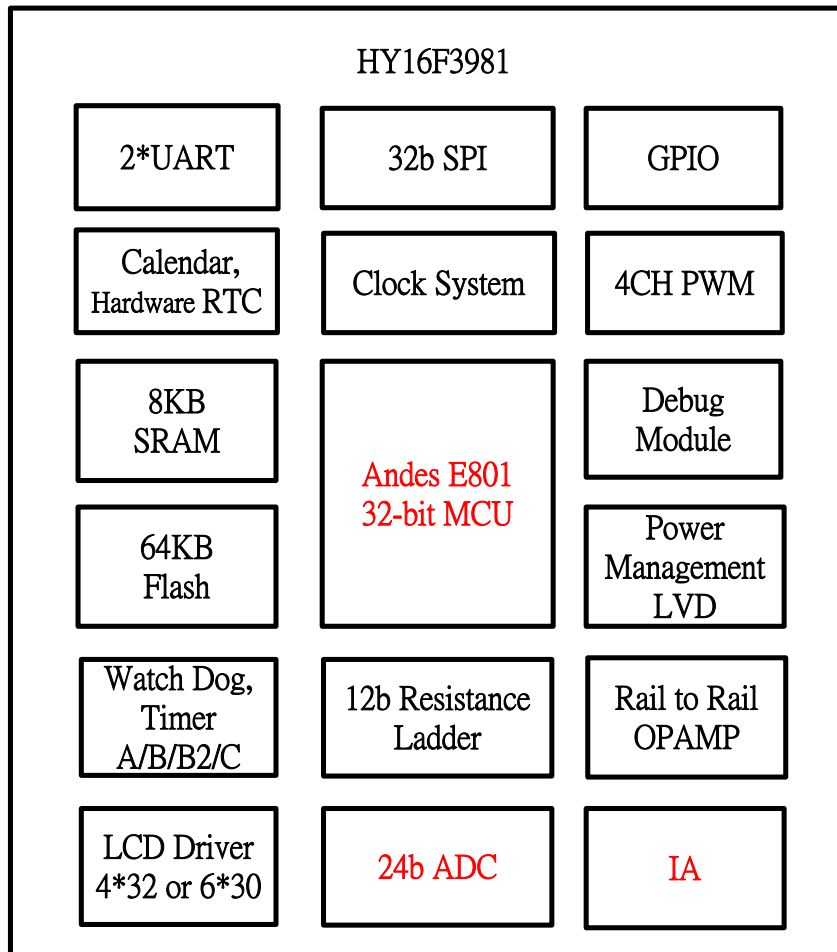


图 7 HY16F3981 应用芯片方块图

- (01)采用晶心科技 Andes 32 位元 CPU 核心 E801 处理器。
- (02)电压操作范围 2.2~3.6V(类比电源没开启情况下), 以及-40°C~85°C工作温度范围。
- (03)支援外部 16MHz 石英震荡器或内部 16MHz 高精度 RC 震荡器,
- (04)程序存储器 64K-Byte Flash ROM
- (05)资料存储器 8K-Byte SRAM。
- (06)拥有 BOR and WDT 功能, 可防止 CPU 死机。
- (07)24-bit 高精度度 $\Sigma\Delta$ ADC 类比数位转换器
(ADC Clock=1MHz@30sps, gain=256 (IA*ADGN), IA HW Chopper On, 0.1uV input RMS noise)
 - (7.1)内置 IA (Instrumentation Amplifier), 最大输入放大倍率高达 $32*8=256$ 倍放大。
 - (7.2)内置温度传感器 TPS。
- (08)内建 1 组 R2ROPA 运算放大器。
- (09)内建硬件 12-bit Resistance Ladder(DAC)。
- (10)16-bit Timer A
- (11)16-bit Timer B/B2 模块具 PWM 波形产生功能
- (12)16-bit Timer C 模块具数位 Capture 功能
- (13)硬件串列通讯 32-bit SPI /UART*2 模块
- (14)硬件 RTC 时钟功能模块
- (15)LCD 驱动模块

HY16F3981

红外线测温应用说明书

3. 系统设计

3.1. 硬件说明

HY16F3981 在红外线测温应用方案提供完整开发套件，系统架构图如下：

1. HW : HY16F3981 IR Demo Board.
2. FW : HY16F3981 IR Demo Code.
3. SW : HY16F3981-AM01-V00(UART) GUI

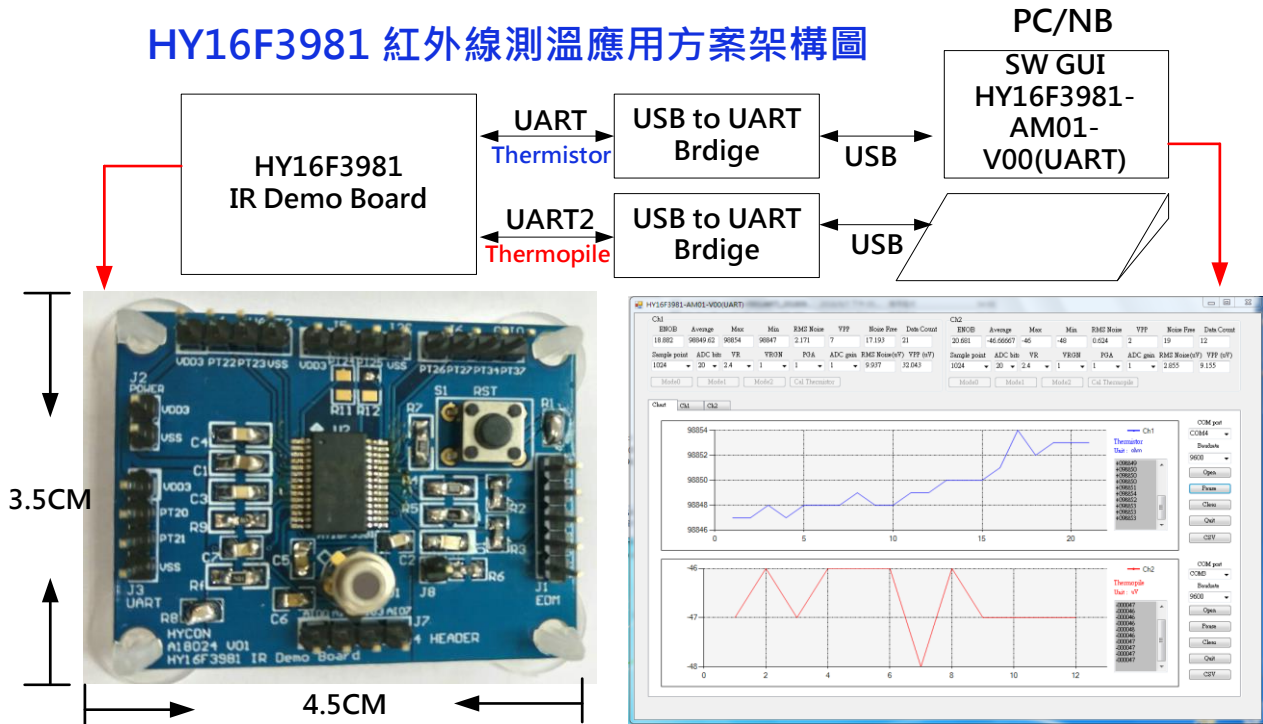


图 8 HY16F3981 红外线测温应用方案架构图

HY16F3981 IR Demo Board 硬件线路图如下：

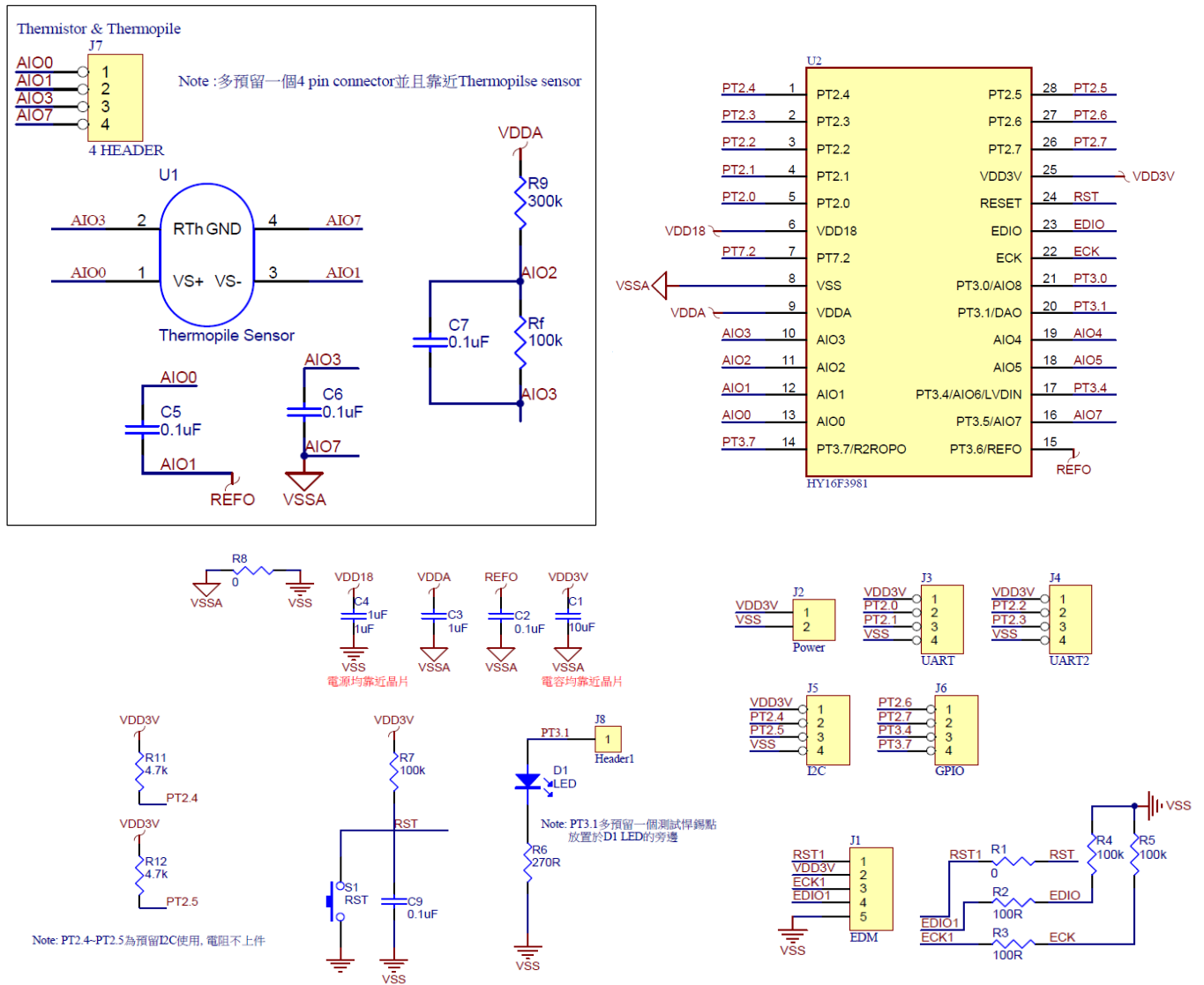
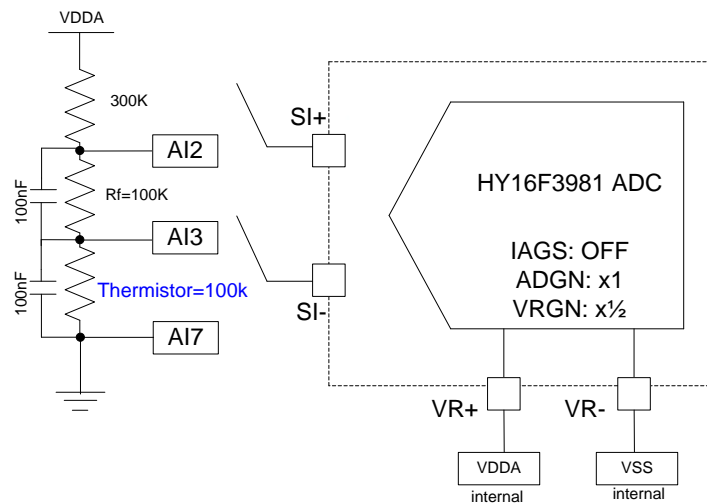


图 9 HY16F3981 IR Demo Board 硬件线路连接图

主要元件介绍：

- (1) MCU: HY16F3981, Andes 32-bit MCU Core + HYCON 24-bit $\Sigma\Delta$ ADC & IA+ MCU 64K Flash。
- (2) J1 Port: 连接 HY16F mini link, HY16F3981 程序开发与除错使用。
- (3) J3&J4 Port: 负责把量测到的 Thermistor 与 Thermopile 数据透过 UART 与 UART2 传输。
- (4) J5&J6 Port: I2C&GPIO 传输预留使用。
- (5) J7 Port: 连接 Thermistor 与 Thermopile 传感器。
- (6) J8 Port: 利用 PT3.1 脚位做控制, 得到一笔完整的 Thermistor & Thermopile 资料, PT3.1 会做 High/Low 变化, 可作为资料传输输出率的辨别。
- (7) R9 分压电阻 & RF 参考电阻: 分压电路设计使用, 主要是用于量测 Thermistor 通道资料。
- (8) 传感器 Thermopile Sensor: AI0-AI1 通道连接 Thermopile, AI3-AI7 通道连接 Thermistor。

Thermistor 电阻量测：



HY16F3981 的 VDDA 类比电源典型输出为 2.4V, 透过 300k 分压电阻, 可让 RF 参考电阻 100k 与 Thermistor 电阻 100k 的电压大小控制在 ADC 可量测的电压讯号范围内。利用 Thermistor 的电阻值查表换算出环境温度, Thermistor 的电阻值可以利用与 RF 参考电阻的比值来比出。以下为 ADC 量测 RF 参考电阻与 Thermistor 电阻的 ADC RAW DATA 计算过程：

ADC_RF_P = AI2-AI3

Comb Filter Reset

ADC_RF_N = AI3-AI2

做 ADC Chopper 扣除 offset 噪声运算

ADC_RF(100k) = (ADC_RF_P - ADC_RF_N) / 2

Comb Filter Reset

ADC_RS_P= AI3-AI7

Comb Filter Reset

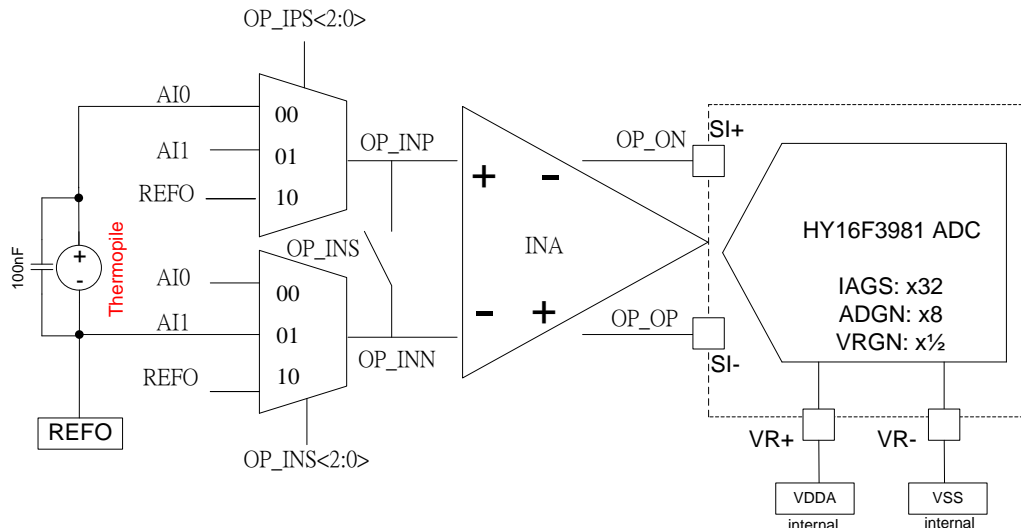
ADC_RS_N= AI7-AI3

做 ADC Chopper 扣除 offset 噪声运算

$ADC_RS(Thermistor)=(ADC_RS_P-ADC_RS_N)/2$

先利用 ADC_RF 与 ADC_RS 的比值换算出电阻值，再利用电阻值查表算出环境温度。

Thermopile 小电压量测：



如果使用 IA chopper on 来做讯号测量，得到的每一笔 ADC 资料都已经是硬件 Chopper 处理过的 ADC 资料，所以不需再透过软件的方式来做 Chopper，因此 Thermopile 的 ADC RAW DATA 资料量测过程如下：

ADC_TP=AI0-AI1 (IA chopper On)

但是在 IA chopper on 开启的情况下，如果量测讯号源的输入阻抗超过 10k 欧姆，会有 ADC 的精度衰减的问题发生，为了追求更高与更精准的 Thermopile 量测，在此提供 IA Chopper Off 的另一种选择方式，需要透过软件的方式处理来做 Chopper，也是利用速度来换取精度的方式。

ADC_TP_P=AI0-AI1

Comb Filter Reset

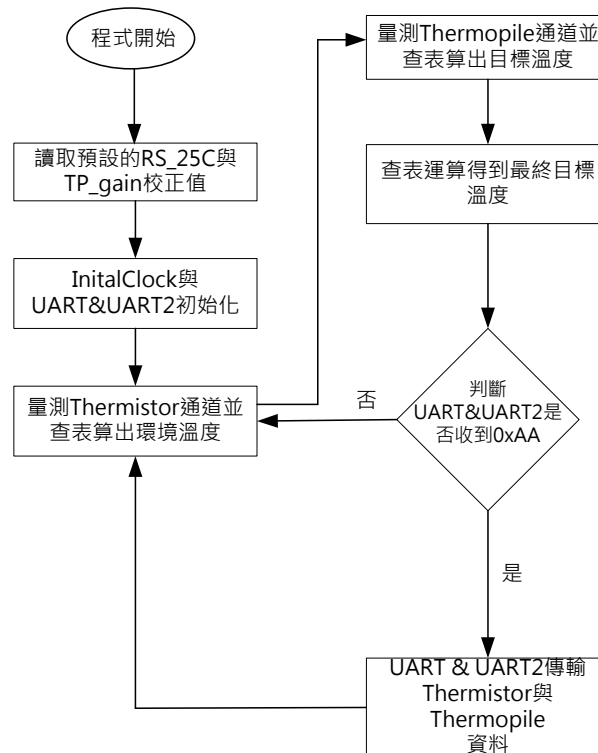
ADC_TP_N=AI1-AI0

$ADC_TP=(ADC_TP_P-ADC_TP_N)/2$, (IA chopper Off)

利用 ADC_TP，先换算出量测到的微小电压值，再把得到的微小电压值(uV)查表做查表换算出目标温度。

3.2. 软件说明

FW : HY16F3981 IR Demo Code 程序流程图如下 :



以下为 UART 传输通讯格式 :

1. 当 UART 与 UART2 分别收到 0xAA 之后, 会开始个别丢出 Thermistor 与 Thermopile 的通道资料到 Host 端

2. UART 与 UART2 传输模式 command 切换如下

UART Mode0 : 0xa9, 0xa2, 0x01, 0x0e, 0x01. RS resistance. unit : ohm

UART Mode1 : 0xa9, 0xa2, 0x01, 0x0e, 0x0B. TH temperatur. Unit : temp

UART Mode2 : 0xa9, 0xa2, 0x01, 0x0e, 0x0D. AIO3-AIO7 ADC Count

UART Cal Thermistor : 0xa9, 0xa2, 0x01, 0x0c, 0x05. Cal Thermistor

UART2 Mode0 : 0xa9, 0xa2, 0x01, 0xe0, 0xA1. TP voltage. unit : uV

UART2 Mode1 : 0xa9, 0xa2, 0x01, 0xe0, 0xAB. TP temperatur. Unit : temp

UART2 Mode2 : 0xa9, 0xa2, 0x01, 0xe0, 0xAD. AIO0-AIO1 ADC Count

UART2 Cal Thermopile : 0xa9, 0xa2, 0x01, 0xc0, 0xA5. Cal Thermopile

4. GUI 软件操作

4.1. 硬件连接说明

SW : HY16F3981-AM01-V00(UART) GUI , 需搭配 FW : HY16F3981_IR_DemoCode_V00 使用.

UART 传输 Thermistor 资料 , UART2 传输 Thermoile 资料 , HY16F3981 IR Demo Board 硬件实体连接如下图。

在本文中使用 FTDI FT232R 这一颗 USB to UART Bridge 与 J3 Port 连接负责接收 Thermistor 资料 , 该芯片可以输出 3.3V 电压 , 直接供给 HY16F3981 芯片 , J4 port 则连接第二组 USB to UART Bridge 负责接收 Thermopile 资料。

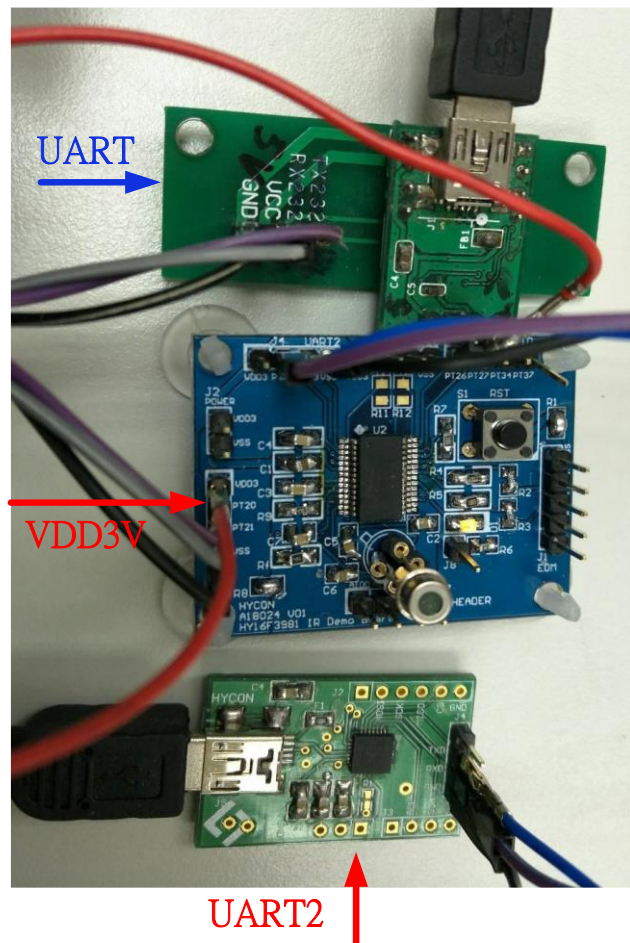


图 10 HY16F3981 IR Demo Board 硬件实体连接图

4.2. 软件显示界面

GUI 操作画面如下图，可同时扫描 Thermistor 与 Thermopile 共两通道资料，图形化界面做资料显示。

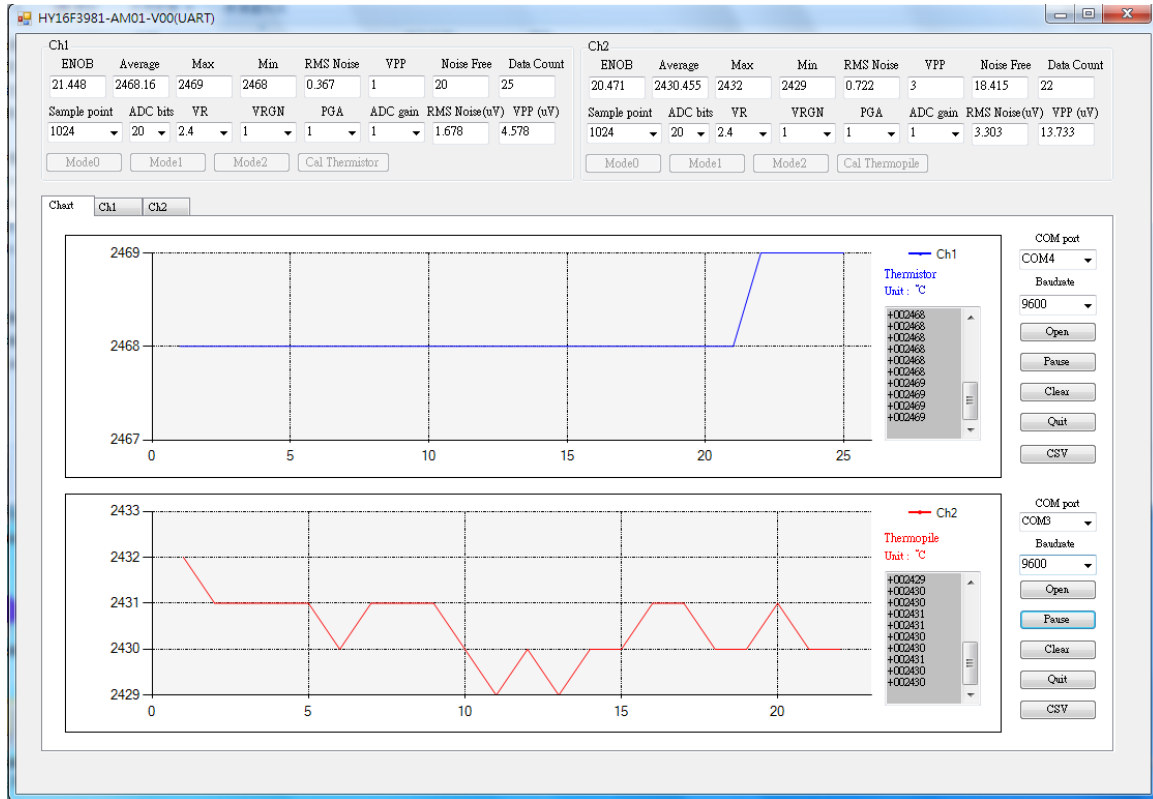


图 11 GUI HY16F3981-AM01-V00(UART)操作画面

注意 1:开启 COM port 所对应的 COM port 号码需要是对应到 Thermistor 与 Thermopile 对应的 Port。以本文为例，从装置管理员中可看到两组 COM port，COM4 是连接到 UART，对应到 Thermistor 通道，COM3 是连接到 UART2，对应到 Thermopile 通道。

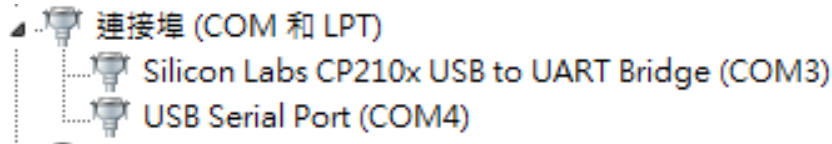


图 12 计算机装置管理员下观察到的 COM Port 状态

4.3. 软件操作说明

HY16F3981-AM01-V00(UART) GUI 分别提供 Thermistor 与 Thermopile 通道在几个不同 Mode 下的资料显示，如下图所示。

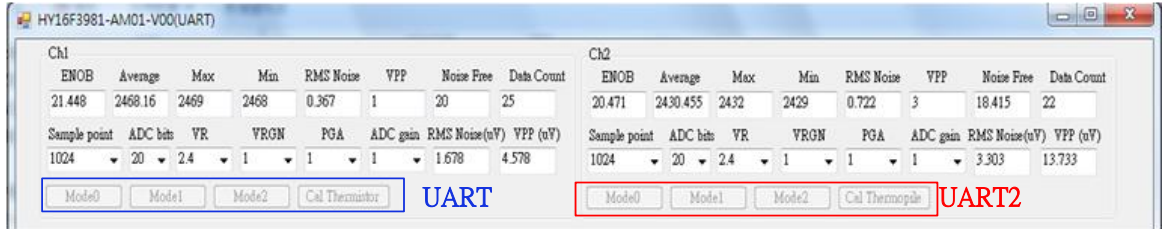


图 13 Thermistor 与 Thermopile 切换不同 Mode 的模式

Thermistor(Ch1) 功能说明：

Mode0：显示测量到的 Thermistor 电阻. unit : ohm

Mode1：显示测量到的 Thermistor 环境温度. unit : temp

Mode2：显示量测到的 AIO3-AIO7 ADC Count. unit : Count

Cal Thermistor：校正 Thermistor 电阻. (校正温度 25C)

Thermopile(Ch2) 功能说明

Mode0:显示测量到的 Thermopile 电压. unit : uV

Mode1:显示测量到的 Thermopile 目标温度. unit : temp

Mode2:显示量测到的 AIO0-AIO1 ADC Count. unit : Count

Cal Thermopile: 校正 Thermopile 电压. (外部输入电压)

参考如下图，使用者于传输资料中，设定 Thermistor 与 Thermopile 都工作于 Mode2，显示温度值。显示数值 2500 代表 25.00 度。

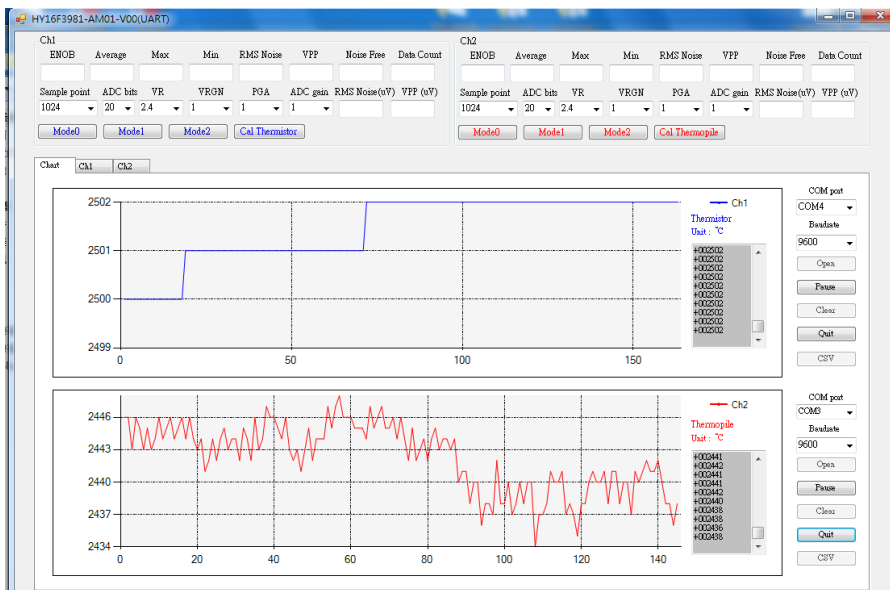


图 14 Thermistor 通道与 Thermopile 通道显示温度(单位:度 C)

参考如下图, 使用者于传输资料中, 手触碰 IR Sensor, 让 IR Sensor 感受到温度变化. 设定 Thermistor 与 Thermopile 都工作于 Mode0. Thermistor 通道显示电阻值(单位: ohm), Thermopile 通道显示电压值(单位: uV).

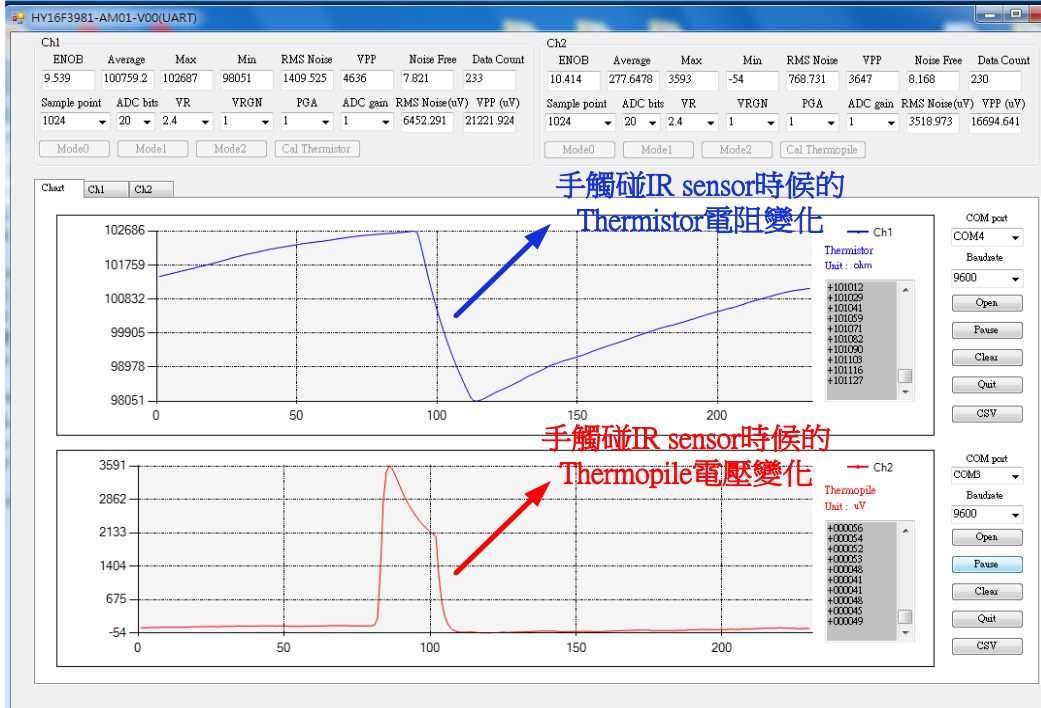


图 15 当 IR Sensor 感受到温度变化时候的 Thermistor 通道与 Thermopile 通道

使用者可以把撷取到的资料, 可以储存为 csv 档案, 由 Excel 开启做资料分析。

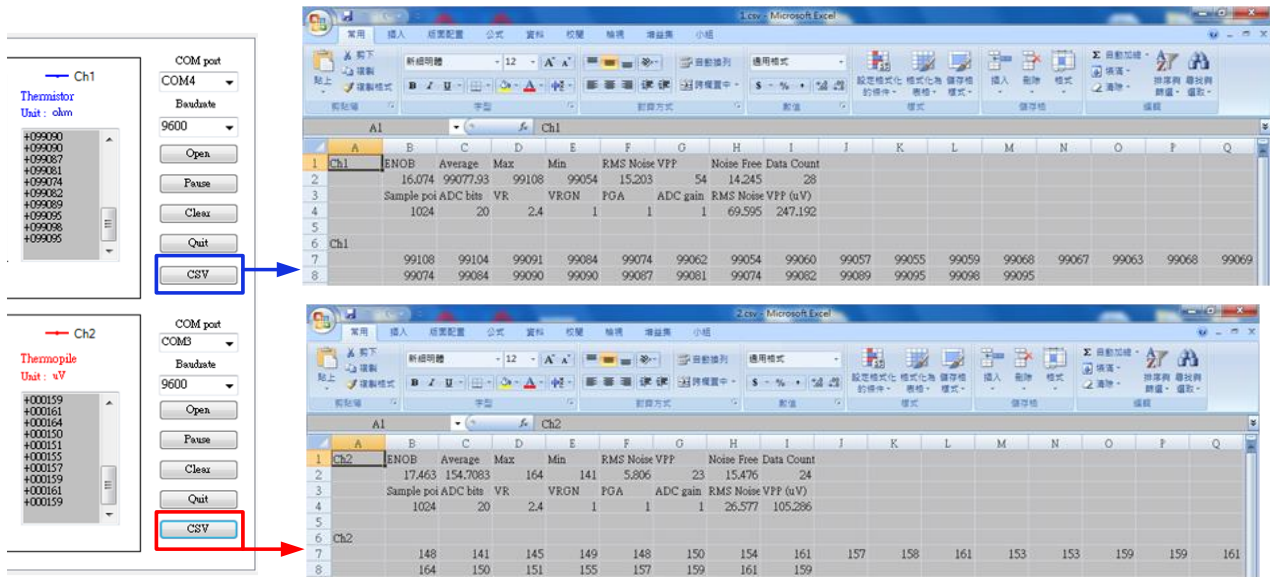


图 16 Thermistor 与 Thermopile 通道储存为 csv 档案

4.4. 总结

在本文中，介绍与提供了完整的红外线测温应用的开发套件供使用者参考，使用者可以依据两个通道 Thermistor 与 Thermopile 的 ADC Raw Data 变化量，透过 UART 传输到 Host PC 端来做实时动态资料显示，以利后续在红外线应用的程序开发与评估。

5. Demo Code 及相关档案

```

/*****
*
* Copyright (c) 2016-2026 HYCON Technology, Inc.
* All rights reserved.
* HYCON Technology <www.hycontek.com>
*
* HYCON reserves the right to amend this code without notice at any time.
* HYCON assumes no responsibility for any errors appeared in the code,
* and HYCON disclaims any express or implied warranty, relating to sale
* and/or use of this code including liability or warranties relating
* to fitness for a particular purpose, or infringement of any patent,
* copyright or other intellectual property right.
*
* -----
* Project Name : HY16F3981_IR_DemoCode_V00
* IDE tooling  : AndeSight C/C++ IDE, version: 2.1.1 Build ID : 201608241332
* Device Ver.  : HY16F_RDSP3_DeviceV0.2 crt0.o for HY16F3981 MCU.
* Library Ver. : 0.2
* MCU Device   :
* Description   :
* Created Date : 2018/9/11
* Created by   : Robert.Wang
*
* Program Description:
* -----
* Thermistor(RS) : AIO3-AIO7, min=97k, typ=100k, max=103k
* RF : AIO2-AIO3, 100k
* Thermopile(TP) : AIO0-AIO1
* Environment Range(Thermistor) : 16~35C
* Target Range(Thermopile) : 0~50C
*
*****/
/ * ----- * /
/ *   I n c l u d e s                               * /
/ * ----- * /
#include "DrvCLOCK.h"
#include "my define.h"
#include "DrvREG32.h"
#include "SpecialMacro.h"
#include "HY16F3981.h"
#include "System.h"
#include "DrvADC.h"
#include "DrvPMU.h"
#include "DrvIA.h"
#include "DrvGPIO.h"
#include "DrvUART.h"
#include "DrvFlash.h"
/ * ----- * /
/ *   S T R U C T U R E S                           * /
/ * ----- * /
/ * ----- * /
volatile typedef union _MCUSTATUS
{
    char _byte;
    struct
    {

```

```

    unsigned b_ADCdone:1;
    unsigned b_TMAdone:1;
    unsigned b_TMBdone:1;
    unsigned b_RTCdone:1;
    unsigned b_UART_TxDone:1;
    unsigned b_UART_RxDone:1;
    unsigned b_UART2_TxDone:1;
    unsigned b_UART2_RxDone:1;
};
} MCUSTATUS;

volatile typedef union _PTINTSTATUS
{
    char _byte;
    struct
    {
        unsigned b_PTINT0done:1;
        unsigned b_PTINT1done:1;
        unsigned b_PTINT2done:1;
        unsigned b_PTINT3done:1;
        unsigned b_PTINT4done:1;
        unsigned b_PTINT5done:1;
        unsigned b_PTINT6Done:1;
        unsigned b_PTINT7Done:1;
    };
} PTINTSTATUS;
/* ----- */
/*   D E F I N I T I O N S                               */
/* ----- */
#define HAO_2MHZ
#define HAO_4MHZ
#define HAO_10MHZ
#define HAO_16MHZ

#define IA_chopper_ON
#define IA_chopper_OFF

#define HSRC //Internal HAO

//UART, Thermistor
#define UART_PORT E_PT2
#define UART_TXD BIT0
#define UART_RXD BIT1
#define Uart_RX_BufferSize 16
#define Uart_TX_BufferSize 16

//UART2, Thermopile
#define UART2_PORT E_PT2
#define UART2_TXD BIT2
#define UART2_RXD BIT3
#define Uart2_RX_BufferSize 16
#define Uart2_TX_BufferSize 16

#define UART_AUTOBAUDRATE
/* ----- */
/*   G l o b a l   C O N S T A N T S                               */
/* ----- */
MCUSTATUS MCUSTATUSbits;

float RS_25C;

```

```

float RF_25C;
float RS_measure;
float RS_kth;
float ADC_RF, ADC_RS;
float ADC_RF_P, ADC_RS_P;
float ADC_RF_N, ADC_RS_N;
float Target_voltage;
int TP_gain;
int ADC_TP_P, ADC_TP_N;
int ADC_TP;
int ADCData;
int ADCData1;
int ADCData2;
int TP_voltage;
int TH_temperature,J_voltage;
int TP_temperature;
unsigned char display_count;
unsigned char display_count2;
unsigned char UartRxBuffer[Uart_RX_BufferSize]={0},UartTxBuffer[Uart_TX_BufferSize]={0};
unsigned char UartTxIndex,UartTxLength,UartRxIndex,UartRxLength;
unsigned char Uart2RxBuffer[Uart2_RX_BufferSize]={0},Uart2TxBuffer[Uart2_TX_BufferSize]={0};
unsigned char Uart2TxIndex,Uart2TxLength,Uart2RxIndex,Uart2RxLength;
unsigned char UART_command[3]=
{
0xa9,0xa2,0x01
};
//UART Mode0 : 0xa9, 0xa2, 0x01, 0x0e, 0x01. RS resistance. unit : ohm
//UART Mode1 : 0xa9, 0xa2, 0x01, 0x0e, 0x0B. TH temperatur. Unit : temp
//UART Mode2 : 0xa9, 0xa2, 0x01, 0x0e, 0x0D. AIO3-AIO7 ADC Count
//UART Cal Thermistor : 0xa9, 0xa2, 0x01, 0x0c, 0x05. Cal Thermistor

//UART2 Mode0 : 0xa9, 0xa2, 0x01, 0xe0, 0xA1. TP voltage. unit : uV
//UART2 Mode1 : 0xa9, 0xa2, 0x01, 0xe0, 0xAB. TP temperatur. Unit : temp
//UART2 Mode2 : 0xa9, 0xa2, 0x01, 0xe0, 0xAD. AIO0-AIO1 ADC Count
//UART2 Cal Thermopile : 0xa9, 0xa2, 0x01, 0xc0, 0xA5. Cal Thermopile

//Thermistor
//unit : temperature
int TH[20]={
16,17,18,19,20,
21,22,23,24,25,
26,27,28,29,30,
31,32,33,34,35
};

//Thermistor
//unit : ohm
int TH_ohm[20]={
148766,142183,135936,130012,124400,
119038,113928,109059,104420,100000,
95788,91775,87950,84305,80830,
77517,74357,71342,68466,65720
};

//Thermopile
//unit : temperature
int TP[51]={
0,1,2,3,4,
5,6,7,8,9,
10,11,12,13,14,

```

```
15,16,17,18,19,
20,21,22,23,24,
25,26,27,28,29,
30,31,32,33,34,
35,36,37,38,39,
40,41,42,43,44,
45,46,47,48,49,
50
};
```

```
//Thermopile
//unit : uV
//OTP-638D2, absolute 25C table
int TP_V[51]={
5,78,151,226,302,           //0~4C
380,456,535,614,693,       //5~9C
775,856,938,1022,1105,     //10~14C
1191,1277,1363,1451,1539,  //15~19C
1628,1722,1815,1909,2003,  //20~24C
2097,2191,2285,2379,2473,  //25~29C
2567,2665,2765,2865,2966,  //30~34C
3067,3170,3273,3378,3483,  //35~39C
3590,3696,3804,3912,4022,  //40~44C
4132,4243,4355,4468,4582,  //45~49C
4697                         //50C
};
```

```
/*
//Thermopile
//unit : uV
//OTP-638D2, absolute 0C table
int TP_V[51]={
0,75,150,226,303,          //0~4C
381,460,540,620,702,      //5~9C
784,867,952,1037,1124,    //10~14C
1211,1299,1390,1482,1575, //15~19C
1672,1757,1842,1927,2012, //20~24C
2097,2182,2267,2353,2438, //25~29C
2524,2628,2736,2837,2942, //30~34C
3045,3149,3253,3359,3466, //35~39C
3574,3681,3788,3897,4006, //40~44C
4116,4227,4340,4453,4566, //45~49C
4681}                       //50C
;
*/
unsigned char ACK_KEY,ACK_KEY2 ;
/*----- */
/*  D a t a  F l a s h                               */
/*----- */
//Default calibration data RS_25C and TP_gain
const unsigned char DefaultData[8] __attribute__((section ("DATA_EARE0"))) =
{
0xE8, 0x7A, 0x01, 0x00, //97000=0x17AE8, Default calibration data RS_25C=97000
0x70, 0x00, 0x00, 0x00, //112=0x70, TP_gain, Default calibration data TP_gain=112, calibration at 4mV
(gain=256, VRGN=1/2, OSR=32768)
};
```

```
/*----- */
/*  F u n c t i o n  P R O T O T Y P E S                               */
/*----- */
```

```

/ * ----- */
void Delay(unsigned int num);
void InitalClock(void);
void InitalUart(void);
void InitalUart2(void);
void Inital_GPIO(void);
void Inital_ADC_TH(void);
void Inital_ADC_TP(void);
void Calibrate_ADC_TH(void);
void Calibrate_ADC_TP(void); //input 4mV on AI0-AI1
void Measure_ADC_TH(void);
void Measure_ADC_TP(void);
void Temptable_TH(void);
void Temptable_TP(void);
void Cold_Junction(void);
void SendUART(int data);
void SendUART2(int data);
void InitalUart_AutoBaudRate(void);
unsigned int Handshake(void);
unsigned char ISP_UART_Read(unsigned char* ptr_data, unsigned int count);
void ISP_UART_Write(unsigned char* ptr_data, unsigned int count);
/ * ----- */
/ *   M a i n   F u n c t i o n   */
/ * ----- */
int main(void)
{

    //ACK_KEY & ACK_KEY2
    ACK_KEY=0;
    ACK_KEY2=0;

    //Read Calibration data RS_25C and TP_gain
    unsigned char index=0;
    unsigned int BufferRx[32];
    for(index=0;index<32;index++)
    {
        BufferRx[index]=0xFFFFFFFF; //to set BufferRx[0]=0xFFFFFFFF....BufferRx[31]=0xFFFFFFFF
    }
    ReadPage(0xF000,BufferRx); //Read BufferRx to make sure BurnPage data
    RS_25C=BufferRx[0]; //Read calibration data RS_25C
    TP_gain=BufferRx[1]; //Read calibration data TP_gain

    //Initial variable
    display_count=0;
    display_count2=0;
    RF_25C=100000; //Assume RF=100k

    //Initial IP
    InitalClock();
#ifdef UART_AUTOBAUDRATE
    InitalUart_AutoBaudRate();
    InitalUart2();
#else
    InitalUart();
    InitalUart2();
#endif
    MCUSTATUSbits.b_UART_TxDone=ENABLE;
    MCUSTATUSbits.b_UART2_TxDone=ENABLE;
    DrvGPIO_Open(E_PT3,0x02,E_IO_OUTPUT);
    DrvGPIO_ClrPortBits(E_PT3,0x02);

```



```

SYS_EnableGIE(4,0x3FF);

for(UartRxLength=0;UartRxLength<=Uart_RX_BufferSize;UartRxLength++)
{
    UartRxBuffer[UartRxLength]=0;
    UartRxIndex=0;
}

for(Uart2RxLength=0;Uart2RxLength<=Uart2_RX_BufferSize;Uart2RxLength++)
{
    Uart2RxBuffer[Uart2RxLength]=0;
    Uart2RxIndex=0;
}

while(1)
{

    DrvGPIO_SetPortBits(E_PT3,0x02);
    //Thermistor
    Inital_ADC_TH();
    Measure_ADC_TH(); //ADC raw count to R, get RS_measure and RS_kth
    ( unit :R )
    Temptable_TH(); //R to T, get TH_temperature
    Cold_Junction(); //T to V, get J_voltage

    //Thermopile
    Inital_ADC_TP();
    Measure_ADC_TP(); //ADC raw count to V, get TP_voltage=Thermopile
    measure voltage( unit :uV )
    Target_voltage=TP_voltage+J_voltage; //Target_voltage=TP_voltage+J_voltage
    Temptable_TP(); //V to T, get TP_temperature Temperature
    DrvGPIO_ClrPortBits(E_PT3,0x02);

    //Case if UART receive == 0xAA
    if(UartRxBuffer[0]==0xAA)
    {
        ACK_KEY=1;
        for(UartRxLength=0;UartRxLength<=Uart_RX_BufferSize;UartRxLength++)
        {
            UartRxBuffer[UartRxLength]=0;
            UartRxIndex=0;
        }
    }

    //Case if UART receive == Mode0
    //0xa9, 0xa2, 0x01, 0x0e, 0x01
    if
    (
        (UartRxBuffer[4]==0x01) &&
        (UartRxBuffer[3]==0x0e && UartRxBuffer[2]==UART_command[2]) &&
        (UartRxBuffer[1]==UART_command[1] && UartRxBuffer[0]==UART_command[0])
    )
    {
        display_count=0;
        for(UartRxLength=0;UartRxLength<=Uart_RX_BufferSize;UartRxLength++)
        {
            UartRxBuffer[UartRxLength]=0;
            UartRxIndex=0;
        }
    }
}

```

```

    }
}

//Case if UART receive == Mode1
//0xa9, 0xa2, 0x01, 0x0e, 0x0B
if
(
  (UartRxBuffer[4]==0x0B) &&
  (UartRxBuffer[3]==0x0e && UartRxBuffer[2]==UART_command[2]) &&
  (UartRxBuffer[1]==UART_command[1] && UartRxBuffer[0]==UART_command[0])
)
{
  display_count=1;
  for(UartRxLength=0;UartRxLength<=Uart_RX_BufferSize;UartRxLength++)
  {
    UartRxBuffer[UartRxLength]=0;
    UartRxIndex=0;
  }
}

//Case if UART receive == Mode2
//0xa9, 0xa2, 0x01, 0x0e, 0x0D
if
(
  (UartRxBuffer[4]==0x0D) &&
  (UartRxBuffer[3]==0x0e && UartRxBuffer[2]==UART_command[2]) &&
  (UartRxBuffer[1]==UART_command[1] && UartRxBuffer[0]==UART_command[0])
)
{
  display_count=2;
  for(UartRxLength=0;UartRxLength<=Uart_RX_BufferSize;UartRxLength++)
  {
    UartRxBuffer[UartRxLength]=0;
    UartRxIndex=0;
  }
}

//Case if UART receive == Cal Thermistor
//0xa9, 0xa2, 0x01, 0x0c, 0x05
if
(
  (UartRxBuffer[4]==0x05) &&
  (UartRxBuffer[3]==0x0c && UartRxBuffer[2]==UART_command[2]) &&
  (UartRxBuffer[1]==UART_command[1] && UartRxBuffer[0]==UART_command[0])
)
{
  Inital_ADC_TH();
  Calibrate_ADC_TH();
  SYS_DisableGIE();
  DrvFlash_Burn_Word(0xF000,0x2000,RS_25C);
  Delay(100000);
  SYS_EnableGIE(4,0x3FF);
  for(UartRxLength=0;UartRxLength<=Uart_RX_BufferSize;UartRxLength++)
  {
    UartRxBuffer[UartRxLength]=0;
    UartRxIndex=0;
  }
}
}

```

```

//Case if UART2 receive == 0xAA
if(Uart2RxBuffer[0]==0xAA)
{
    ACK_KEY2=1;
    for(Uart2RxLength=0;Uart2RxLength<=Uart2_RX_BufferSize;Uart2RxLength++)
    {
        Uart2RxBuffer[Uart2RxLength]=0;
        Uart2RxIndex=0;
    }
}

//Case if UART2 receive == Mode0
// 0xa9, 0xa2, 0x01, 0xe0, 0xA1
if
(
    (Uart2RxBuffer[4]==0xA1) &&
    (Uart2RxBuffer[3]==0xe0 && Uart2RxBuffer[2]==UART_command[2]) &&
    (Uart2RxBuffer[1]==UART_command[1] && Uart2RxBuffer[0]==UART_command[0])
)
{
    display_count2=0;
    for(Uart2RxLength=0;Uart2RxLength<=Uart2_RX_BufferSize;Uart2RxLength++)
    {
        Uart2RxBuffer[Uart2RxLength]=0;
        Uart2RxIndex=0;
    }
}

//Case if UART2 receive == Mode1
//0xa9, 0xa2, 0x01, 0xe0, 0xAB
if
(
    (Uart2RxBuffer[4]==0xAB) &&
    (Uart2RxBuffer[3]==0xe0 && Uart2RxBuffer[2]==UART_command[2]) &&
    (Uart2RxBuffer[1]==UART_command[1] && Uart2RxBuffer[0]==UART_command[0])
)
{
    display_count2=1;
    for(Uart2RxLength=0;Uart2RxLength<=Uart2_RX_BufferSize;Uart2RxLength++)
    {
        Uart2RxBuffer[Uart2RxLength]=0;
        Uart2RxIndex=0;
    }
}

//Case if UART2 receive == Mode2
//0xa9, 0xa2, 0x01, 0xe0, 0xAD
if
(
    (Uart2RxBuffer[4]==0xAD) &&
    (Uart2RxBuffer[3]==0xe0 && Uart2RxBuffer[2]==UART_command[2]) &&
    (Uart2RxBuffer[1]==UART_command[1] && Uart2RxBuffer[0]==UART_command[0])
)
{
    display_count2=2;
    for(Uart2RxLength=0;Uart2RxLength<=Uart2_RX_BufferSize;Uart2RxLength++)
    {
        Uart2RxBuffer[Uart2RxLength]=0;
        Uart2RxIndex=0;
    }
}

```

```

    }
}

//Case if UART2 receive == Cal Thermopile
//0xa9, 0xa2, 0x01, 0xc0, 0xA5
if
(
  (Uart2RxBuffer[4]==0xA5) &&
  (Uart2RxBuffer[3]==0xc0 && Uart2RxBuffer[2]==UART_command[2]) &&
  (Uart2RxBuffer[1]==UART_command[1] && Uart2RxBuffer[0]==UART_command[0])
)
{
  Inital_ADC_TP();
  Calibrate_ADC_TP();
  SYS_DisableGIE();
  DrvFlash_Burn_Word(0xF004,0x2000,TP_gain);
  Delay(100000);
  SYS_EnableGIE(4,0x3FF);
  for(Uart2RxLength=0;Uart2RxLength<=Uart2_RX_BufferSize;Uart2RxLength++)
  {
    Uart2RxBuffer[Uart2RxLength]=0; //clear RX buffer
    Uart2RxIndex=0;
  }
}

if(ACK_KEY==1)
{
  //UART Mode0
  if(display_count==0)
  {
    DrvADC_DisableInt();
    SendUART(RS_measure); //RS resistance. unit : ohm
    DrvADC_EnableInt();
  }
  //UART Mode1
  if(display_count==1)
  {
    DrvADC_DisableInt();
    SendUART(TH_temperature); //TH temperature. Unit : temp
    DrvADC_EnableInt();
  }
  //UART Mode2
  if(display_count==2)
  {
    DrvADC_DisableInt();
    SendUART(ADC_RF); //AIO3-AIO7 ADC Count
    DrvADC_EnableInt();
  }
}

if(ACK_KEY2==1)
{
  //UART2 Mode0
  if(display_count2==0)
  {
    DrvADC_DisableInt();
    SendUART2(TP_voltage); //TP voltage. unit : uV
    DrvADC_EnableInt();
  }
}

```

```

//UART2 Mode1
if(display_count2==1)
{
    DrvADC_DisableInt();
    SendUART2(TP_temperature);           //TP temperatur. Unit : temp
    DrvADC_EnableInt();
}
//UART2 Mode2
if(display_count2==2)
{
    DrvADC_DisableInt();
    SendUART2(ADC_TP);                   //AIO0-AIO1 ADC Count
    DrvADC_EnableInt();
}
}

} //while(1) end

}

/* ----- */
/* Function Name: HW0_ISR()                */
/* Description  : I2C/UART/SPI interrupt Service Routine (HW0). */
/* Arguments   : None.                    */
/* Return Value: None.                    */
/* Remark     :                            */
/* ----- */
void HW0_ISR(void)
{
    if(DrvUART_GetRxFlag())
    {
        UartRxBuffer[UartRxIndex]=DrvUART_Read();
        UartRxIndex++;
        if(UartRxIndex>=Uart_RX_BufferSize)
        {
            UartRxIndex=0;
            MCUSTATUSbits.b_UART_RxDone=ENABLE;
        }
        DrvUART_ClrRxFlag();
    }

    if(DrvUART_GetTxFlag())
    {
        if(MCUSTATUSbits.b_UART_TxDone==DISABLE)
        {
            DrvUART_Write(UartTxBuffer[UartTxIndex++]);
            DrvUART_ClrTxFlag();
            if(UartTxIndex>=UartTxLength)
            {
                DrvUART_EnableInt(ENABLE,ENABLE); //ENABLE UART Tx Interrupt, ENABLE UART Rx
Interrupt
                MCUSTATUSbits.b_UART_TxDone=ENABLE;
                UartTxIndex=0;
            }
        }
    }
    if(MCUSTATUSbits.b_UART_TxDone==ENABLE)
    {

```

```

        DrvUART_EnableInt(DISABLE,ENABLE); //DISABLE UART Tx Interrupt, ENABLE UART Rx
Interrupt
        DrvUART_ClrTxFlag();
    }
}
}
/*-----*/
/* Function Name: HW1_ISR() */
/* Description : WDT & RTC & Timer A/B/C interrupt Service Routine (HW1). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/*-----*/
void HW1_ISR(void)
{
}
}
/*-----*/
/* Function Name: HW2_ISR() */
/* Description : ADC interrupt Service Routine (HW2). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/*-----*/
void HW2_ISR(void)
{
    if(DrvADC_ReadIntFlag()) //Read ADC Flag
    {
        ADCData=DrvADC_GetConversionData(>>12; //@ ADC get 20bit, +/- 19bit
        MCUSTATUSbits.b_ADCdone=1;
    }
}
}
/*-----*/
/* Function Name: HW4_ISR() */
/* Description : PT3 interrupt Service Routine (HW4). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/*-----*/
void HW4_ISR(void)
{
}
}
}
/*-----*/
/* Function Name: HW5_ISR() */
/* Description : PT2 interrupt Service Routine (HW5). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/*-----*/
void HW5_ISR(void)
{
}
}
}
/*-----*/
/* Function Name: HW7_ISR() */
/* Description : UART2 interrupt Service Routine (HW7). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */

```

```

/*-----*/
void HW7_ISR(void)
{
    if(DrvUART2_GetRxFlag())
    {
        Uart2RxBuffer[Uart2RxIndex]=DrvUART2_Read();
        Uart2RxIndex++;
        if(Uart2RxIndex>=Uart2_RX_BufferSize)
        {
            Uart2RxIndex=0;
            MCUSTATUSbits.b_UART2_RxDone=ENABLE;
        }
        DrvUART2_ClrRxFlag();
    }

    if(DrvUART2_GetTxFlag())
    {
        if(MCUSTATUSbits.b_UART2_TxDone==DISABLE)
        {
            DrvUART2_Write(Uart2TxBuffer[Uart2TxIndex++]);
            DrvUART2_ClrTxFlag();
            if(Uart2TxIndex>=Uart2TxLength)
            {
                DrvUART2_EnableInt(ENABLE,ENABLE); //ENABLE UART2 Tx Interrupt, ENABLE UART2 Rx
Interrupt
                MCUSTATUSbits.b_UART2_TxDone=ENABLE;
                Uart2TxIndex=0;
            }
        }
        if(MCUSTATUSbits.b_UART2_TxDone==ENABLE)
        {
            DrvUART2_EnableInt(DISABLE,ENABLE); //DISABLE UART2 Tx Interrupt, ENABLE UART2 Rx
Interrupt
            DrvUART2_ClrTxFlag();
        }
    }
}
/*-----*/
/* Function Name: HW8_ISR() */
/* Description : TMB2 interrupt Service Routine (HW8). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/*-----*/
void HW8_ISR(void)
{
}
/*-----*/
/* Function Name: HW9_ISR() */
/* Description : OPA interrupt Service Routine (HW9). */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/*-----*/
void HW9_ISR(void)
{

```

```

}
/* ----- */
/* Function Name: tlb_exception_handler() */
/* Description : Exception Service Routines. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void tlb_exception_handler()
{
    long long Exception_link_pointer_temp, Exception_link_pointer;
    Exception_link_pointer_temp=0, Exception_link_pointer=0;
    //可以从变量 Exception_link_pointer 找出进入 exception 的程序段
    asm volatile("add %0, %1, $lp" : "=&r"(Exception_link_pointer) : "r"(Exception_link_pointer_temp));

    int Exception_ITYPE;
    Exception_ITYPE=0;
    //以下语句可以将 ITYPE(ir6)资料从暂存器中搬出到 C 环境中.
    asm volatile("mfsr %0,$ITYPE" : "=&r"(Exception_ITYPE)); //ITYPE 资料存放在变量 Exception_ITYPE.

    while(1);
}
/* ----- */
/* Software Delay Subroutines */
/* ----- */
void Delay(unsigned int num)
{
    for(;num>0;num--)
        asm("NOP");
}

/* ----- */
/* Function Name: Inital_ADC_TH() */
/* Description : ADC thermistor channel Initialization Subroutines. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void Inital_ADC_TH(void)
{
    //Set ADC Clock
    #if defined(HAO_2MHZ)
        DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
        DrvCLOCK_SelectIHOSC(0); //Select internal 2MHZ=HS_CK
        DrvADC_ClkEnable(2); //Setting ADC CLOCK ADCK=HS_CK/4
    #endif
    #if defined(HAO_4MHZ)
        DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
        DrvCLOCK_SelectIHOSC(1); //Select internal 4MHZ=HS_CK
        DrvADC_ClkEnable(2); //Setting ADC CLOCK ADCK=HS_CK/4
    #endif
    #if defined(HAO_10MHZ)
        DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
        DrvCLOCK_SelectIHOSC(2); //Select internal 10MHZ=HS_CK
        DrvADC_ClkEnable(3); //Setting ADC CLOCK ADCK=HS_CK/8
    #endif
    #if defined(HAO_16MHZ)

```



```

    DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
    DrvCLOCK_SelectIHOSC(3); //Select internal 16MHZ=HS_CK
    DrvADC_ClkEnable(4); //Setting ADC CLOCK ADCK=HS_CK/16
#endif

//Set VDDA voltage
DrvPMU_VDDA_LDO_Ctrl(E_LDO);
DrvPMU_VDDA_Voltage(E_VDDA2_4);
DrvPMU_BandgapEnable();
DrvPMU_REFO_Enable();
Delay(0x1000);
DrvPMU_AnalogGround(ENABLE); //ADC analog ground source selection.
//1 : Enable buffer and use internal source(need to
work with ADC)

//Set ADC input pin
DrvADC_SetADCInputChannel(ADC_Input_AIO3,ADC_Input_AIO7); //Set the ADC positive/negative 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_ADGain(0);
DrvADC_DCoffset(0); //DC offset input voltage selection (VREF=REFP-REFN)
DrvADC_RefVoltage(0,0); //Set the ADC reference voltage. VDDA-VSSA
DrvADC_FullRefRange(1); //Set the ADC full reference range select.
DrvADC_OSR(2); //2 : 120sps

//Set ADC interrupt
DrvADC_EnableInt();
DrvADC_Enable();
DrvADC_CombFilter(ENABLE); //Enable comb filter
}

/* ----- */
/* Function Name: Calibrate_ADC_TH() */
/* Description : ADC thermistor channel calibration. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void Calibrate_ADC_TH(void)
{
    DrvADC_SetADCInputChannel(ADC_Input_AIO3,ADC_Input_AIO7);
    DrvADC_CombFilter(DISABLE);
    DrvADC_CombFilter(ENABLE);
    MCUSTATUSbits.b_ADCdone=0;
    while(MCUSTATUSbits.b_ADCdone==0);
    ADC_RS_P=ADCData;
    MCUSTATUSbits.b_ADCdone=0;

    DrvADC_SetADCInputChannel(ADC_Input_AIO7,ADC_Input_AIO3);
    DrvADC_CombFilter(DISABLE);
    DrvADC_CombFilter(ENABLE);
    MCUSTATUSbits.b_ADCdone=0;
    while(MCUSTATUSbits.b_ADCdone==0);
    ADC_RS_N=ADCData;
    MCUSTATUSbits.b_ADCdone=0;
}

```

```

ADC_RS=(ADC_RS_P-ADC_RS_N)/2; //get ADC Raw count on Thermistor(RS)

DrvADC_SetADCInputChannel(ADC_Input_AIO2,ADC_Input_AIO3);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_RF_P=ADCData;
MCUSTATUSbits.b_ADCdone=0;

DrvADC_SetADCInputChannel(ADC_Input_AIO3,ADC_Input_AIO2);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_RF_N=ADCData;
MCUSTATUSbits.b_ADCdone=0;

ADC_RF=(ADC_RF_P-ADC_RS_N)/2; //get ADC Raw count on RF

RS_25C = (ADC_RS*RF_25C)/ADC_RF;

}
/* ----- */
/* Function Name: Measure_ADC_TH() */
/* Description : ADC thermistor channel measurement. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void Measure_ADC_TH(void)
{

DrvADC_SetADCInputChannel(ADC_Input_AIO3,ADC_Input_AIO7);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_RS_P=ADCData;
MCUSTATUSbits.b_ADCdone=0;

DrvADC_SetADCInputChannel(ADC_Input_AIO7,ADC_Input_AIO3);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_RS_N=ADCData;
MCUSTATUSbits.b_ADCdone=0;

ADC_RS=(ADC_RS_P-ADC_RS_N)/2; //get ADC Raw count on Thermistor(RS)

DrvADC_SetADCInputChannel(ADC_Input_AIO2,ADC_Input_AIO3);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_RF_P=ADCData;
MCUSTATUSbits.b_ADCdone=0;

DrvADC_SetADCInputChannel(ADC_Input_AIO3,ADC_Input_AIO2);

```

```

DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_RF_N=ADCData;
MCUSTATUSbits.b_ADCdone=0;

ADC_RF=(ADC_RF_P-ADC_RS_N)/2; //get ADC Raw count on RF

RS_measure=(ADC_RS*RF_25C)/ADC_RF;
RS_kth = RS_measure*RF_25C/RS_25C;

}

/* ----- */
/* Function Name: Inital_ADC_TP() */
/* Description : ADC Thermopile Initialization Subroutines. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void Inital_ADC_TP(void)
{
//Set ADC Clock
#if defined(HAO_2MHZ)
DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
DrvCLOCK_SelectHOSC(0); //Select internal 2MHZ=HS_CK
DrvADC_ClkEnable(2); //Setting ADC CLOCK ADCK=HS_CK/4
#endif
#if defined(HAO_4MHZ)
DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
DrvCLOCK_SelectHOSC(1); //Select internal 4MHZ=HS_CK
DrvADC_ClkEnable(2); //Setting ADC CLOCK ADCK=HS_CK/4
#endif
#if defined(HAO_10MHZ)
DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
DrvCLOCK_SelectHOSC(2); //Select internal 10MHZ=HS_CK
DrvADC_ClkEnable(3); //Setting ADC CLOCK ADCK=HS_CK/8
#endif
#if defined(HAO_16MHZ)
DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
DrvCLOCK_SelectHOSC(3); //Select internal 16MHZ=HS_CK
DrvADC_ClkEnable(4); //Setting ADC CLOCK ADCK=HS_CK/16
#endif

//Set VDDA voltage
DrvPMU_VDDA_LDO_Ctrl(E_LDO);
DrvPMU_VDDA_Voltage(E_VDDA2_4);
DrvPMU_BandgapEnable();
DrvPMU_REFO_Enable();
Delay(0x1000);
DrvPMU_AnalogGround(ENABLE); //ADC analog ground source selection.
//1 : Enable buffer and use internal source(need to
work with ADC)
#if defined(IA_chopper_OFF)
//Set ADC input pin
DrvADC_SetADCInputChannel(OP_OP,OP_ON); //Set the ADC positive/negative 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_ADGain(7); //ADC gain=8
    DrvADC_DCOffset(0); //DC offset input voltage selection (VREF=REFP-REFN)
    DrvADC_RefVoltage(0,0); //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

    DrvADC_OSR(2); //2 : 120sps
//Set IA
    DrvIA_SetIAInputChannel(IA_Input_AIO0,IA_Input_AIO1); //ADC positive=AIO0, negative=AIO1
    DrvIA_IAGain(IA_IAGain_32); //IA gain=32
    DrvIA_IACHM(IA_IACHM_NoChopper); //IA Chopper OFF
    DrvIA_IAIS(0); //Input control=open
    DrvIA_ENIA(0); //Disable IA
    DrvIA_ENIA(1);

//Set ADC interrupt
    DrvADC_EnableInt();
    DrvADC_Enable();
    DrvADC_CombFilter(ENABLE); //Enable comb filter
#endif

#if defined(IA_chopper_ON)
//Set ADC input pin
    DrvADC_SetADCInputChannel(OP_OP,OP_ON); //Set the ADC positive/negative 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_ADGain(7); //ADC gain=8
    DrvADC_DCOffset(0); //DC offset input voltage selection
(VREF=REFP-REFN)
    DrvADC_RefVoltage(0,0); //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

    DrvADC_OSR(0); //0 : OSR=32768
//Set IA
    DrvIA_SetIAInputChannel(IA_Input_AIO0,IA_Input_AIO1); //ADC positive=AIO0, negative=AIO1
    DrvIA_IAGain(IA_IAGain_32); //IA Gain=32
    DrvIA_IACHM(IA_IACHM_Both); //IA Chopper On
    DrvIA_IAIS(0); //Input control=open
    DrvIA_ENIA(0); //Disable IA
    DrvIA_ENIA(1); //Enable IA

//Set ADC interrupt
    DrvADC_EnableInt();
    DrvADC_Enable();
    DrvADC_CombFilter(ENABLE); //Enable comb filter
#endif
}

/* ----- */
/* Function Name: Measure_ADC_TP() */
/* Description : ADC Thermopile measurement. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */

```

```

/* ----- */
void Measure_ADC_TP(void)
{
#if defined(IA_chopper_OFF)
//positive measure
DrvIA_SetIInputChannel(IA_Input_AIO0,IA_Input_AIO1);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_TP_P=ADCData;
MCUSTATUSbits.b_ADCdone=0;
//negative measure
DrvIA_SetIInputChannel(IA_Input_AIO1,IA_Input_AIO0);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_TP_N=ADCData;
MCUSTATUSbits.b_ADCdone=0;
ADC_TP=(ADC_TP_P-ADC_TP_N)/2;
TP_voltage=(ADC_TP_P-ADC_TP_N)/2/TP_gain;
#endif

#if defined(IA_chopper_ON)
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_TP=ADCData;
TP_voltage=ADCData/TP_gain;
MCUSTATUSbits.b_ADCdone=0;
#endif
}

/* ----- */
/* Function Name: Calibrate_ADC_TP() */
/* Description : */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void Calibrate_ADC_TP(void)
{
#if defined(IA_chopper_OFF)
DrvIA_SetIInputChannel(IA_Input_AIO0,IA_Input_AIO1);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_TP_P=ADCData;
MCUSTATUSbits.b_ADCdone=0;
DrvIA_SetIInputChannel(IA_Input_AIO1,IA_Input_AIO0);
DrvADC_CombFilter(DISABLE);
DrvADC_CombFilter(ENABLE);
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_TP_N=ADCData;
MCUSTATUSbits.b_ADCdone=0;
#endif
}

```

```

ADC_TP=(ADC_TP_P-ADC_TP_N)/2;
TP_gain=(ADC_TP_P-ADC_TP_N)/2/4000; //4mV=4000, assume ADC count=0 at 0mV, calibration at 4mV
#endif

#if defined(IA_chopper_ON)
MCUSTATUSbits.b_ADCdone=0;
while(MCUSTATUSbits.b_ADCdone==0);
ADC_TP=ADCData;
TP_gain=ADCData/4000;
MCUSTATUSbits.b_ADCdone=0;
#endif
}

/* ----- */
/* Function Name: Temptable_TH() */
/* Description : Thermistor temperature table searching */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void Temptable_TH(void)
{
float count1;
float tempH,tempL;
char count2;
unsigned char a=0;
unsigned char b=0;
unsigned char max=19,min=0;
a=(max+min)/2;
b=a+1;
while(1)
{
tempH=TH_ohm[a+1];
tempL=TH_ohm[a];
if(RS_kth>tempH&&RS_kth<=tempL)
break;
else
{
b=b/2;
if(b==0)
b=1;
if(RS_kth>tempL)
a=a-b;
else if(RS_kth<=tempH)
a=a+b;
if(a<min||a>max)
break;
}
}

count1=(tempL-tempH)/100; //Note : /100 代表两个点之间分成 100 阶数, count1 代表每
一阶的数值大小
count2=(RS_kth-tempL)/count1; //count2 代表总共的阶数为多少
TH_temperature=TH[a]*100-count2; //Note : *100 代表两个点之间分成 100 阶数, 十位数与个位
数代表小数点后两位
if(a==255) //a=a-b, but a=0 and b=1, a will be 255
(TH_temperature=TH[min]*100); //to show TH[min]
if(a>max && a!=255)

```

```

        (TH_temperature=TH[max]*100-count2);    //to show TH[max]
    }

/* ----- */
/* Function Name: Temptable_TP()                */
/* Description   : Thermopile temperature table searching */
/* Arguments     : None.                          */
/* Return Value  : None.                          */
/* Remark       :                                  */
/* ----- */
void Temptable_TP(void)
{
    float count1;
    float tempH,tempL;
    char count2;
    unsigned char a=0;
    unsigned char b=0;
    unsigned char max=50,min=0;
    a=(max+min)/2;
    b=a+1;
    while(1)
    {
        tempH=TP_V[a+1];
        tempL=TP_V[a];
        if(Target_voltage<tempH&&Target_voltage>=tempL)
            break;
        else
        {
            b=b/2;
            if(b==0)
                b=1;
            if(Target_voltage<tempL)
                a=a-b;
            else if(Target_voltage>=tempH)
                a=a+b;
            if(a<min||a>max)
                break;
        }
    }

    count1=(tempH-tempL)/100;
    count2=(Target_voltage-tempL)/count1;
    TP_temperature=TP[a]*100+count2+6;    //+6=rounding
    if(a==255)                            //a=a-b, but a=0 and b=1, a will be 255
        (TP_temperature=TP[min]*100);    //to show TP[min]
    if(a>max && a!=255)
        (TP_temperature=TP[max]*100);    //to show TP[max]
}

/* ----- */
/* Function Name: Cold_Junction()                */
/* Description   :                                  */
/* Arguments     : None.                          */
/* Return Value  : None.                          */
/* Remark       :                                  */
/* ----- */
void Cold_Junction(void)
{

```

```

int tempH,tempL;
float Temp_remain;
unsigned char a=0;
unsigned char b=0;
unsigned char max=50,min=0;
a=(max+min)/2;
b=a+1;
while(1)
{
    tempH=TP[a+1];
    tempL=TP[a];
    if((TH_temperature/100)<tempH&&(TH_temperature/100)>=tempL)
        break;
    else
    {
        b=b/2;
        if(b==0)
            b=1;
        if((TH_temperature/100)<tempL)
            a=a-b;
        else if((TH_temperature/100)>=tempH)
            a=a+b;
        if(a<min||a>max)
            break;
    }
}
Temp_remain=(TH_temperature%100)*(TP_V[a+1]-TP_V[a])/100;
J_voltage=TP_V[a]+Temp_remain;
if(a==255) //a=a-b, but a=0 and b=1, a will be 255
(J_voltage=TP_V[min]); //to show TP_V[min]
if(a>max && a!=255)
(J_voltage=TP_V[max]+Temp_remain); //to show TP_V[max]
}

/* ----- */
/* Function Name: InitalClock() */
/* Description : CLOCK Initial Subroutines. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void InitalClock(void)
{
    #if defined(HSRC)
    #if defined(HAO_2MHZ)
    //Clock INIT 2MHZ
    DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
    DrvCLOCK_SelectIHOSC(0); //Select internal 2MHZ
    DrvCLOCK_SelectMCUClock(0,0); //CPU CLOCK IS 'hs_ck/1'
    DrvCLOCK_CalibrateHAO(0); //Calibration 1.843MHz
    #endif
    #if defined(HAO_4MHZ)
    //Clock INIT 4MHZ
    DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
    DrvCLOCK_SelectIHOSC(1); //Select internal 4MHZ
    DrvCLOCK_SelectMCUClock(0,0); //CPU CLOCK IS 'hs_ck/1'
    DrvCLOCK_CalibrateHAO(1); //Calibration 4.147MHz
    #endif
}

```



```

#if defined(HAO_10MHZ)
//Clock INIT 10MHZ
DrvCLOCK_EnableHighOSC(E_INTERNAL,10);           //Select HSRC
DrvCLOCK_SelectIHOSC(2);                         //Select internal 10MHZ
DrvCLOCK_SelectMCUClock(0,0);                   //CPU CLOCK IS 'hs_ck/1'
DrvCLOCK_CalibrateHAO(2);                       //Calibration 9.216MHz
#endif
#if defined(HAO_16MHZ)
//Clock INIT 16MHZ
DrvCLOCK_EnableHighOSC(E_INTERNAL,10);           //Select HSRC
DrvCLOCK_SelectIHOSC(3);                         //Select internal 16MHZ
DrvCLOCK_SelectMCUClock(0,0);                   //CPU CLOCK IS 'hs_ck/1'
DrvCLOCK_CalibrateHAO(3);                       //Calibration 15.667MHz
#endif
#endif

}

/* ----- */
/* Function Name: InitalUart()                    */
/* Description   : UART Initial Subroutines.      */
/* Arguments    : None.                          */
/* Return Value : None.                          */
/* Remark      :                                  */
/* ----- */
void InitalUart(void)
{
#if defined(HSRC)
    DrvUART_ClkEnable(1,0);                       //Choose the internal HAO as clock source
    #if defined(HAO_2MHZ)

DrvUART_Open(1843,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBITS
_1,2);
    #endif
    #if defined(HAO_4MHZ)

DrvUART_Open(4147,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBITS
_1,2);
    #endif
    #if defined(HAO_10MHZ)

DrvUART_Open(9216,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBITS
_1,2);
    #endif
    #if defined(HAO_16MHZ)

DrvUART_Open(15667,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBIT
S_1,2);
    #endif
#endif

//1843 : oscillator frequency 2MHz Unit After Calibration HAO = 1843kHz
//4147 : oscillator frequency 4MHz Unit After Calibration HAO = 4147kHz
//9216 : oscillator frequency 10MHz Unit After Calibration HAO = 9216kHz
//15667 : oscillator frequency 16MHz Unit After Calibration HAO = 15667kHz
//None parity
//8 data bits.
//2 : Port 2.0 =TX, Port 2.1 =RX

```

```

DrvGPIO_Open(UART_PORT,UART_TXD,E_IO_OUTPUT);
DrvGPIO_Open(UART_PORT,UART_RXD,E_IO_INPUT);
DrvGPIO_Open(UART_PORT,UART_RXD|UART_TXD,E_IO_PullHigh);
DrvGPIO_ClkGenerator(E_HS_CK,1);
DrvUART_EnableInt(ENABLE,ENABLE); //Enable UART Tx Interrupt, Enable UART Rx Interrupt
DrvUART_Disable_AutoBaudrate();
DrvUART_Close();
DrvUART_Enable();
}

/* ----- */
/* Function Name: InitalUart2() */
/* Description : UART2 Initial Subroutines. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void InitalUart2(void)
{
#if defined(HSRC)
    DrvUART2_ClkEnable(1,0); //Choose the internal HAO as clock source
    #if defined(HAO_2MHZ)

DrvUART2_Open(1843,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBIT
S_1,2);
    #endif
    #if defined(HAO_4MHZ)

DrvUART2_Open(4147,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBIT
S_1,2);
    #endif
    #if defined(HAO_10MHZ)

DrvUART2_Open(9216,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBIT
S_1,2);
    #endif
    #if defined(HAO_16MHZ)

DrvUART2_Open(15667,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBI
TS_1,2);
    #endif
#endif
//1843 : oscillator frequency 4MHz Unit After Calibration HAO = 1843kHz
//4147 : oscillator frequency 4MHz Unit After Calibration HAO = 4147kHz
//9216 : oscillator frequency 4MHz Unit After Calibration HAO = 9216kHz
//15667 : oscillator frequency 4MHz Unit After Calibration HAO = 15667kHz
//None parity
//8 data bits.
//2 : Port 2.2 =TX, Port 2.3 =RX
DrvGPIO_Open(UART2_PORT,UART2_TXD,E_IO_OUTPUT);
DrvGPIO_Open(UART2_PORT,UART2_RXD,E_IO_INPUT);
DrvGPIO_Open(UART2_PORT,UART2_RXD|UART2_TXD,E_IO_PullHigh);
DrvUART2_EnableInt(ENABLE,ENABLE); //ENABLE UART2 Tx Interrupt, ENABLE UART2 Rx Interrupt
DrvUART2_Disable_AutoBaudrate();
DrvUART2_Close();
DrvUART2_Enable();
}

/* ----- */

```

```

/* Function Name: void SendUART(int data) */
/* Description : UART send data. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void SendUART(int data)
{
    ADCData1=data;
    if((ADCData1<0)||((ADCData1>0x80000000)) // plus-minus sign judgment
    {
        ADCData1=~ADCData1;
        ADCData1++;
        UartTxBuffer[0]='-';
    }
    else
    {
        UartTxBuffer[0]='+';
    }
    UartTxBuffer[6]=ADCData1%10 | '0'; //unit
    ADCData1=ADCData1/10;
    UartTxBuffer[5]=ADCData1%10 | '0'; //ten
    ADCData1=ADCData1/10;
    UartTxBuffer[4]=ADCData1%10 | '0'; //hundred
    ADCData1=ADCData1/10;
    UartTxBuffer[3]=ADCData1%10 | '0'; //thousand
    ADCData1=ADCData1/10;
    UartTxBuffer[2]=ADCData1%10 | '0'; //ten thousand
    ADCData1=ADCData1/10;
    UartTxBuffer[1]=ADCData1%10 | '0'; //hundred thousand
    ADCData1=ADCData1/10;
    UartTxBuffer[7]='\r';
    UartTxBuffer[8]='\n';
    MCUSTATUSbits.b_UART_TxDone=DISABLE;
    UartTxLength=9;
    UartTxIndex=0;
    DrvUART_EnableInt(ENABLE,ENABLE); //Enable UART Tx Interrupt;Enable UART Rx Interrupt
    while(!MCUSTATUSbits.b_UART_TxDone); //If MCUSTATUSbits.b_UART_TxDone=DISABLE, stop at
here
}

/* ----- */
/* Function Name: void SendUART2(int data) */
/* Description : UART send data. */
/* Arguments : None. */
/* Return Value : None. */
/* Remark : */
/* ----- */
void SendUART2(int data)
{
    ADCData2=data;
    if((ADCData2<0)||((ADCData2>0x80000000)) // plus-minus sign judgment
    {
        ADCData2=~ADCData2;
        ADCData2++;
        Uart2TxBuffer[0]='-';
    }
    else
    {
        Uart2TxBuffer[0]='+';
    }
}

```

```

}

Uart2TxBuffer[6]=ADCDData2%10 | '0'; //unit
ADCDData2=ADCDData2/10;
Uart2TxBuffer[5]=ADCDData2%10 | '0'; //ten
ADCDData2=ADCDData2/10;
Uart2TxBuffer[4]=ADCDData2%10 | '0'; //hundred
ADCDData2=ADCDData2/10;
Uart2TxBuffer[3]=ADCDData2%10 | '0'; //thousand
ADCDData2=ADCDData2/10;
Uart2TxBuffer[2]=ADCDData2%10 | '0'; //ten thousand
ADCDData2=ADCDData2/10;
Uart2TxBuffer[1]=ADCDData2%10 | '0'; //hundred thousand
ADCDData2=ADCDData2/10;
Uart2TxBuffer[7]='\r';
Uart2TxBuffer[8]='\n';
MCUSTATUSbits.b_UART2_TxDone=DISABLE;
Uart2TxLength=9;
Uart2TxIndex=0;
DrvUART2_EnableInt(ENABLE,ENABLE); //Enable UART Tx Interrupt;Enable UART Rx Interrupt
while(!MCUSTATUSbits.b_UART2_TxDone); //If MCUSTATUSbits.b_UART_TxDone=DISABLE, stop
at here
}

/* ----- */
/* UART AutoBaud Rate Initial Subroutines */
/* ----- */
void InitalUart_AutoBaudRate(void)
{
    //Step0. UART clock and UART open
    #if defined(HSRC)
    #if defined(HAO_2MHZ)
    //Clock INIT 2MHZ
    DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
    DrvCLOCK_SelectIHOSC(0); //Select internal 2MHZ
    DrvCLOCK_SelectMCUClock(0,0); //CPU CLOCK IS 'hs_ck/1'
    DrvCLOCK_CalibrateHAO(0); //Calibration 1.843MHz

    DrvUART_Open(1843,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBITS
    _1,2);
    #endif
    #if defined(HAO_4MHZ)
    //Clock INIT 4MHZ
    DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
    DrvCLOCK_SelectIHOSC(1); //Select internal 4MHZ
    DrvCLOCK_SelectMCUClock(0,0); //CPU CLOCK IS 'hs_ck/1'
    DrvCLOCK_CalibrateHAO(1); //Calibration 4.147MHz

    DrvUART_Open(4147,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBITS
    _1,2);
    #endif
    #if defined(HAO_10MHZ)
    //Clock INIT 10MHZ
    DrvCLOCK_EnableHighOSC(E_INTERNAL,10); //Select HSRC
    DrvCLOCK_SelectIHOSC(2); //Select internal 10MHZ
    DrvCLOCK_SelectMCUClock(0,0); //CPU CLOCK IS 'hs_ck/1'
    DrvCLOCK_CalibrateHAO(2); //Calibration 9.216MHz

    DrvUART_Open(9216,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBITS

```

```

_1,2);
#endif
#if defined(HAO_16MHZ)
//Clock INIT 16MHZ
DrvCLOCK_EnableHighOSC(E_INTERNAL,10);           //Select HSRC
DrvCLOCK_SelectHOSC(3);                          //Select internal 16MHZ
DrvCLOCK_SelectMCUClock(0,0);                    //CPU CLOCK IS 'hs_ck/1'
DrvCLOCK_CalibrateHAO(3);                        //Calibration 15.667MHz

DrvUART_Open(15667,B9600,DRVUART_PARITY_NONE,DRVUART_DATABITS_8,DRVUART_STOPBIT
S_1,2);
#endif
#endif

DrvGPIO_Open(UART_PORT,UART_TXD,E_IO_OUTPUT);
DrvGPIO_Open(UART_PORT,UART_RXD,E_IO_INPUT);
DrvGPIO_Open(UART_PORT,UART_RXD|UART_TXD,E_IO_PullHigh);
int_00=0x00000c00;                               //Clear UTxIF,URxIF
clk_08=0x20200000;                               //First, setting TUCKS=1b
clk_08=0x1F100000;                               //Second, setting ENUD=1b
ur_00=0xff00ff65;

//Step1. Specific Configuration Before Auto Baud Rate
ur_08 = 0x0;                                     // Clear BRG
DrvGPIO_Close(E_PT2,UART_RXD,E_IO_INPUT);
while(!(int_00 & 0x4));                          // Wait for RX IRQ
DrvGPIO_Open(UART_PORT,UART_RXD,E_IO_INPUT);
ur_00 = 0xff000000;                              // clear UART Flags
Delay(1000);                                     // Delay about 1000 NOP. OR less
int_00 = 0x00000400;                             // Clear UART Rx Interrupt flag

//Step2. Auto-Baud rate Enable and Detection Handshake
ur_04 =0xff08;                                  // Enable Auto-Baud rate detection
while(1)                                        // break only when Auto Baud Rate and Handshake Complete
{
    while(!(int_00 & 0x4))                       // Wait for RX IRQ. Wait Master to send 0x55
    int_00 = 0x00000400;                         // Clear UART Rx Interrupt flag
    if(Handshake())                              // If Handshake process(user defined) was completed
    {
        break;
    }
    ur_04 =0xff08;                               // Retry and Enable Auto-Baud rate detection
}
}

/* ----- */
/* Handshake */
/* ----- */
unsigned int Handshake(void)
{
    unsigned char ACK_MASTER = 0xa1;
    unsigned char ACK_SLAVE = 0xa2;
    unsigned char ACK_HANDSHAKE = 0xa3;
    unsigned char read_data;
    unsigned int Handshake_timeout = 1000;

    while(Handshake_timeout--)
    {

```

```

        ISP_UART_Write(&ACK_SLAVE,1);           // Slave sends Handshake signal as a slave
        if(!(ISP_UART_Read(&read_data,1)))      // If UART READ was completed
        {
            if(read_data==ACK_MASTER)          // Host sends back Handshake
                Character(ACK_MASTER)
                {
                    ISP_UART_Write(&ACK_HANDSHAKE,1); // Handshake is completed
                    return 1;
                }
        }
    }
    return 0;
}
/* ----- */
/* ISP_UART_Read */
/* ----- */
unsigned char ISP_UART_Read(unsigned char* ptr_data, unsigned int count)
{
    unsigned int Timeout=1000;
    while(Timeout && count)                    // while(timeout!=0 && count !=0)
    {
        if(int_00 & 0x4)
        {
            *ptr_data = ur_0c;                // get data
            Timeout=1000;                     // reset timeout
            ptr_data++;
            count--;
            int_00 = 0x00000400;              // clear interrupt flag
        }
        else
        {
            Timeout--;
        }
    }
    return count;                             // return 0: UART read was completed
                                           // return !=0: UART read timeout
}
/* ----- */
/* ISP_UART_Write */
/* ----- */
void ISP_UART_Write(unsigned char* ptr_data, unsigned int count)
{
    while(count)
    {
        if(int_00 & 0x8)
        {
            ur_0c=*ptr_data<<16;
            ptr_data++;
            count--;
            Delay(1000);                      // delay some NOP. for Master Rx to receive correct data if
need
        }
    }
}

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

```

/*-----*/



HY16F3981_IR_DemoCode_V00_UART.zip



HY16F3981_AM01_V00.zip

6. 参考文献

[1] http://www.hycontek.com/wp-content/uploads/APD-SD18009_TC.pdf , 紘康科技
HY11P13 红外线传感器应用说明书

[2] http://www.hycontek.com/hy_mcu/DS-HY16F3981_TC.pdf , 紘康科技 HY16F3981
Datasheet.

[3] http://www.hycontek.com/hy_mcu/UG-HY16F3981_TC.pdf , 紘康科技 HY16F3981 User
Guide.

7. 修订记录

以下描述本文件差异较大的地方，而标点符号与字形的改变不在此描述范围。

文件版次	页次	日期	摘要
V01	All	2022/8/30	初版发行