HY16F3981
Infrared Thermopile Measurement Application Note

Table of Contents

1. INTRODUCTION 4

2. THEORY EXPLANATION 5

2.1. Introduction of Infrared Wavelength 5

2.2. Optics and Windows 5

2.3. Introduction of Sensors 6

2.4. Control IC 8

3. SYSTEM DESIGN 10

3.1. Hardware Description 10

3.2. Software Description 14

4. GUI SOFTWARE OPERATION 15

4.1. Hardware Connection Instructions 15

4.2. Software interface 16

4.3. Software Operation Instructions 17

4.4. Summary 19

5. DEMO CODE AND RELATED FILES 20

6. REFERENCES 47

7. REVISION 48

Attention:

1、 HYCON Technology Corp. reserves the right to change the content of this datasheet without further notice. For most up-to-date information, please constantly visit our website: http://www.hycontek.com .

2、 HYCON Technology Corp. is not responsible for problems caused by figures or application circuits narrated herein whose related industrial properties belong to third parties.

3、 Specifications of any HYCON Technology Corp. products detailed or contained herein stipulate the performance, characteristics, and functions of the specified products in the independent state. We does not guarantee of the performance, characteristics, and functions of the specified products as placed in the customer's products or equipment. Constant and sufficient verification and evaluation is highly advised.

4、 Please note the operating conditions of input voltage, output voltage and load current and ensure the IC internal power consumption does not exceed that of package tolerance. HYCON Technology Corp. assumes no responsibility for equipment failures that resulted from using products at values that exceed, even momentarily, rated values listed in products specifications of HYCON products specified herein.

5、 Notwithstanding this product has built-in ESD protection circuit, please do not exert excessive static electricity to protection circuit.

6、 Products specified or contained herein cannot be employed in applications which require extremely high levels of reliability, such as device or equipment affecting the human body, health/medical equipments, security systems, or any apparatus installed in aircrafts and other vehicles.

7、Despite the fact that HYCON Technology Corp. endeavors to enhance product quality as well as reliability in every possible way, failure or malfunction of semiconductor products may happen. Hence, users are strongly recommended to comply with safety design including redundancy and fire-precaution equipments to prevent any accidents and fires that may follow.

8、Use of the information described herein for other purposes and/or reproduction or copying without the permission of HYCON Technology Corp. is strictly prohibited.

1. Introduction

Common applications of infrared sensors are in medical, industrial, and consumer fields, such as ear thermometers, forehead thermometers, industrial temperature instrument, infrared thermometer…etc. In the application of ear thermometer, it is necessary to pay attention on the warming effect while inserting the ear, connection of wave guide and sensor...etc. As for the application of infrared thermometer, distances of the lens to object, and the lens focal distance should be noticed.

The HY16F3981 has built-in instrumentation amplifier with high input impedance and Sigma delta 24 Bit ADC with high precision signal measurement characteristics. Therefore, HY16F3981 is very suitable for measurement applications with high input impedance and low voltage (uV). This application note introduces the use of HY16F3981 chip and IR Sensor to complete infrared temperature measurement applications.

2. Theory Explanation

2.1. Introduction of Infrared Wavelength

According to infrared wavelength and radiant, there are 3 classifications as below.

Near Infra-red, NIR: 700~2,000nm

Middle Infra-red, MIR: 3,000~5,000nm

Far Infra-red, FIR: 8,000~14,000nm

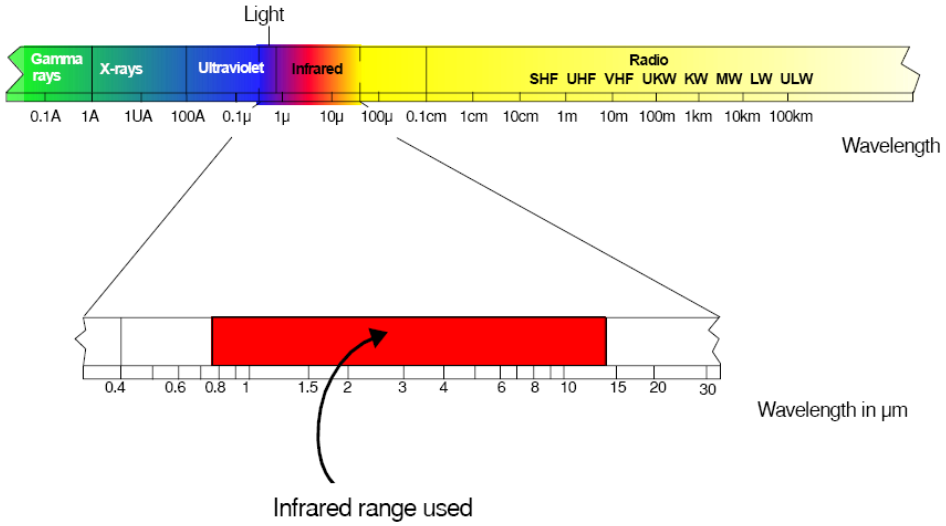


Figure 1 Wavelength Spectrum Map, Infrared Common Measurement Wavelength Range: 700nm~14000nm [1]

2.2. Optics and Windows

Measuring the energy emitted from the target is the only key point in an optical system. So the target must completely contain this spot. There are some certain relationship between energy and the measuring distance, the area of the spot. The farther the distance, the smaller the energy received by the sensor, the bigger the area of the spot.

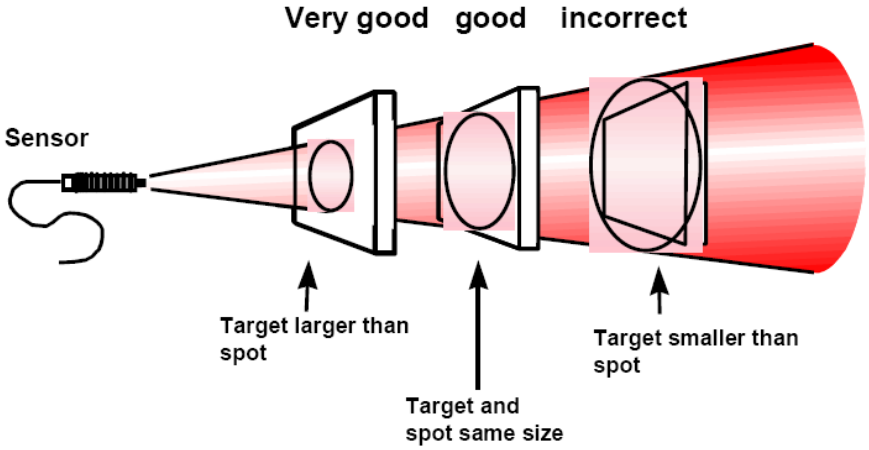


Figure 2 The target must completely contain the spot to be measured, otherwise the measured value will be incorrect [1]

2.3. Introduction of Sensors

IR sensor:

IR sensor is composed by two interface devices, Thermopile and Thermistor. Its package is as displayed in Figure 3.

Figure 3 IR Temperature Sensors [2]

Thermopile:

Thermopile outputs small voltage. The voltage value depends on object temperature and the ambience temperature where the thermopile is located, as shown in Figure 4. The recommended precision is 0.01°C. Its absolute error is within ±0.03°C when the thermopile is used at ambient 25°C. Due to the fact that the sensor is made up of semiconductor materials, the measured result is easily influenced by temperature. A good IR sensor, the mathematical formula of the thermopile should be as below:

$$V_{out} = K \times [(T_t + 273.13)^4 - (T_a + 273.13)^4]$$
$$= K \times f(T_t, T_a) \quad \text{.....Equation (1)}$$
$$= K \times [f(T_t, T_{ref}) - f(T_a, T_{ref})]$$

Vout: Thermopile Voltage Output
K: Sensitivity of Thermopile
Tt: Target Temperature (°C)
Ta:Ambient Temperature (°C)

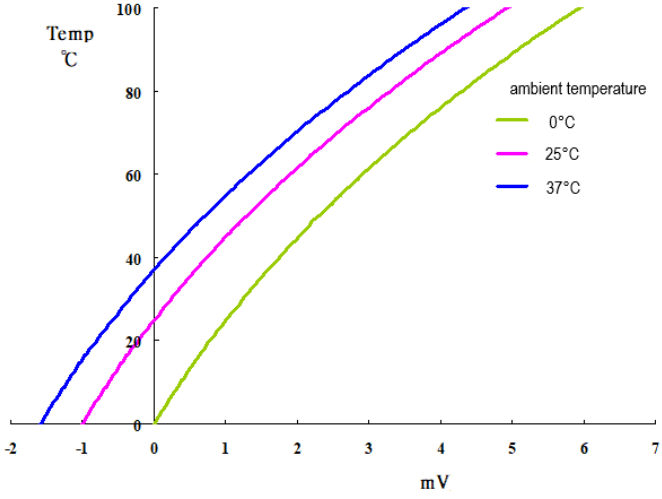


Figure 4 Thermopile Voltage and Temperature Curve

Good Thermopile equips with the following features:

- When Tt=Ta, Vout=0
- K is constant, will not change with ambience temperature.

Thermistor:

Thermistor resistance changes according to the temperature where it is located (As shown in Figure 5) and can be used to monitor the internal temperature of IR sensor. Here it is also called ambient temperature when measuring. It is suggested the measure error and repeatability should be less than 0.05°C.

Thermistor mathematical formula is given as below

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

$R_{th}(T)$: Thermistor resistance change value

B: Sensitivity of Thermistor

R_{25}: 25℃ resistance

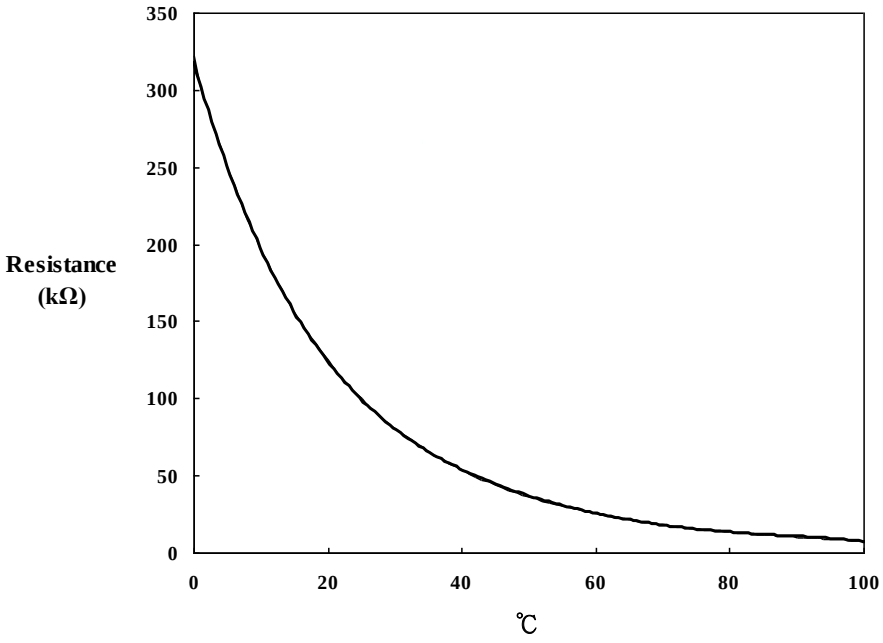


Figure 5 Thermistor Resistance and Temperature Curve

2.4. Control IC

Measure temperature signal via IR sensor and transform it to electrical signals (resistance and output small voltage). Utilize HY16F3981 of HYCON Technology Corp to measure electrical signals, to operate and to display digitally, accomplished non-touchable IR temperature measurement solution in minimum components (as Figure 6 illustrated).

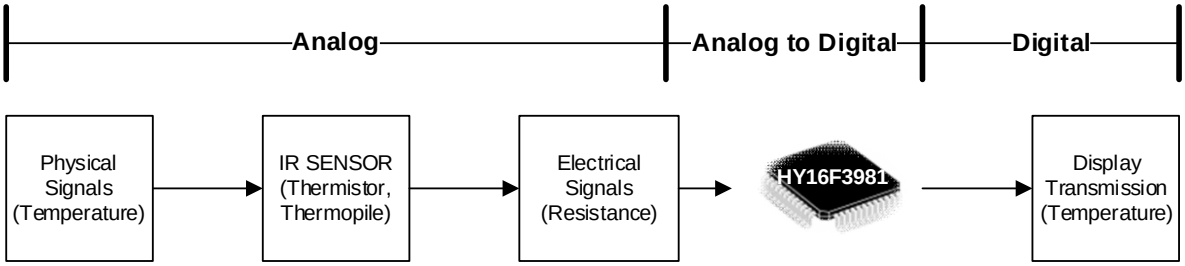


Figure 6 Analog and Digital Signal Conversion

HY16F3981 Chip Introduction:

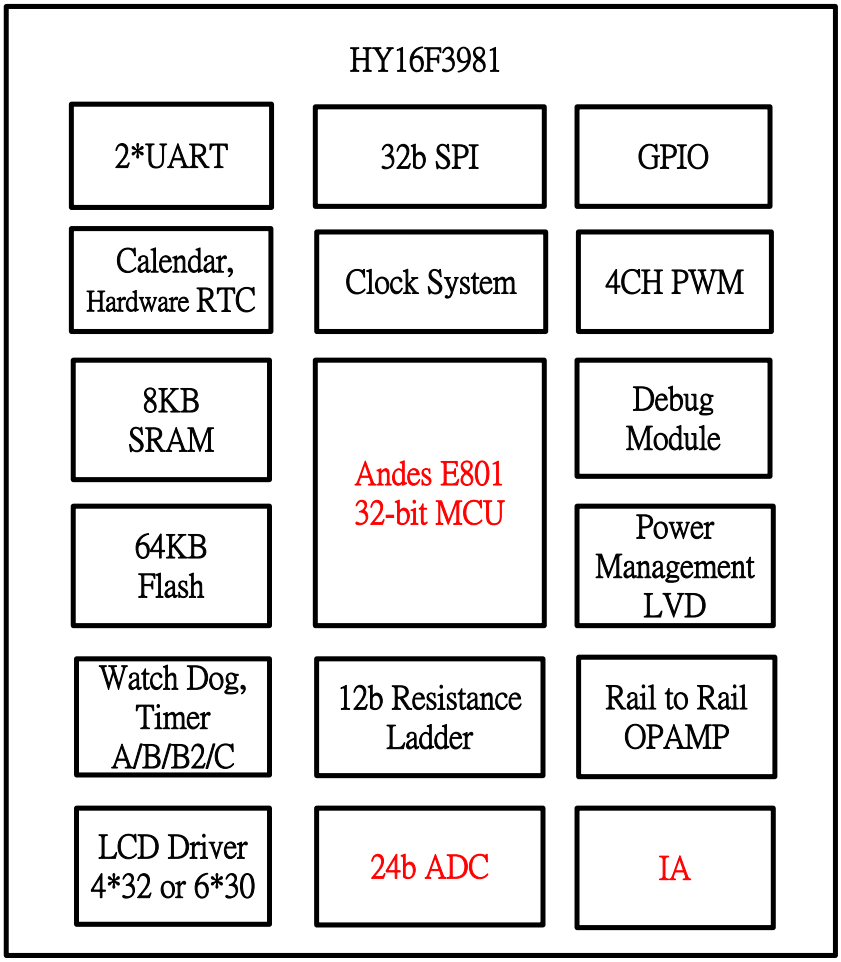


Figure 7 HY16F3981 Internal Block Diagram

(01) The core of the CPU is Andes E801 32-bit MCU.
(02) Operating voltage range: 2.2V to 3.6V(Analog power off), Operating temperature range: -40°C ~85°C.
(03) External High Speed Oscillator Max 16MHz, Internal High Speed Oscillator Max 16MHz.
(04) 64K-Byte Flash ROM.
(05) 8K-Byte SRAM.
(06) Brownout Detector and Watch dog Timer prevents CPU from crash.
(07) High-performance 24-bit A/D converter (24-bitΣΔADC)
(ADC Clock=1MHz@30sps, gain=256 (IA*ADGN), IA HW Chopper On, 0.1uV input RMS noise)
(7.1) Built-in Instrumentation Amplifier (IA), the maxi-mum input magnification up to 256.
(7.2) Built-in absolute temperature sensor (TPS).
(08) Built-in an Rail to Rail OPAMP.
(09) Built-in 12-bit Resistance Ladder(DAC).
(10) 16-bit Timer A.
(11) 16-bit Timer B/B2, the TMB can be also used to generate the waveform of the PWM.
(12) 16-bit Timer C, the TMC has capture function.
(13) 32-bit SPI/ UART(x2) communication Hardware IP.
(14) RTC Hardware IP.
(15) LCD Driver.

3. System Design

3.1. Hardware Description

The HY16F3981 infrared thermopile measurement application provides a complete development kit. The system structure diagram is as follows.

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

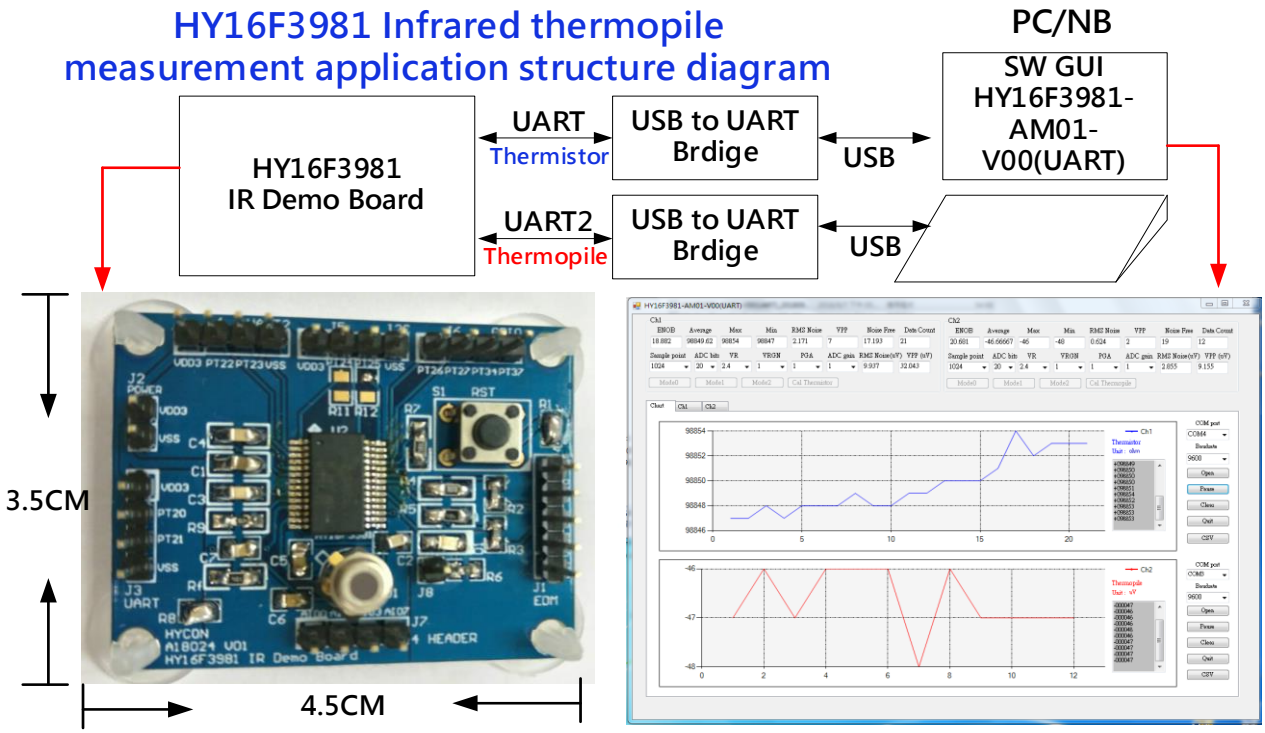


Figure 8 HY16F3981 Infrared thermopile measurement application structure diagram

HY16F3981 IR Demo Board circuit diagram:

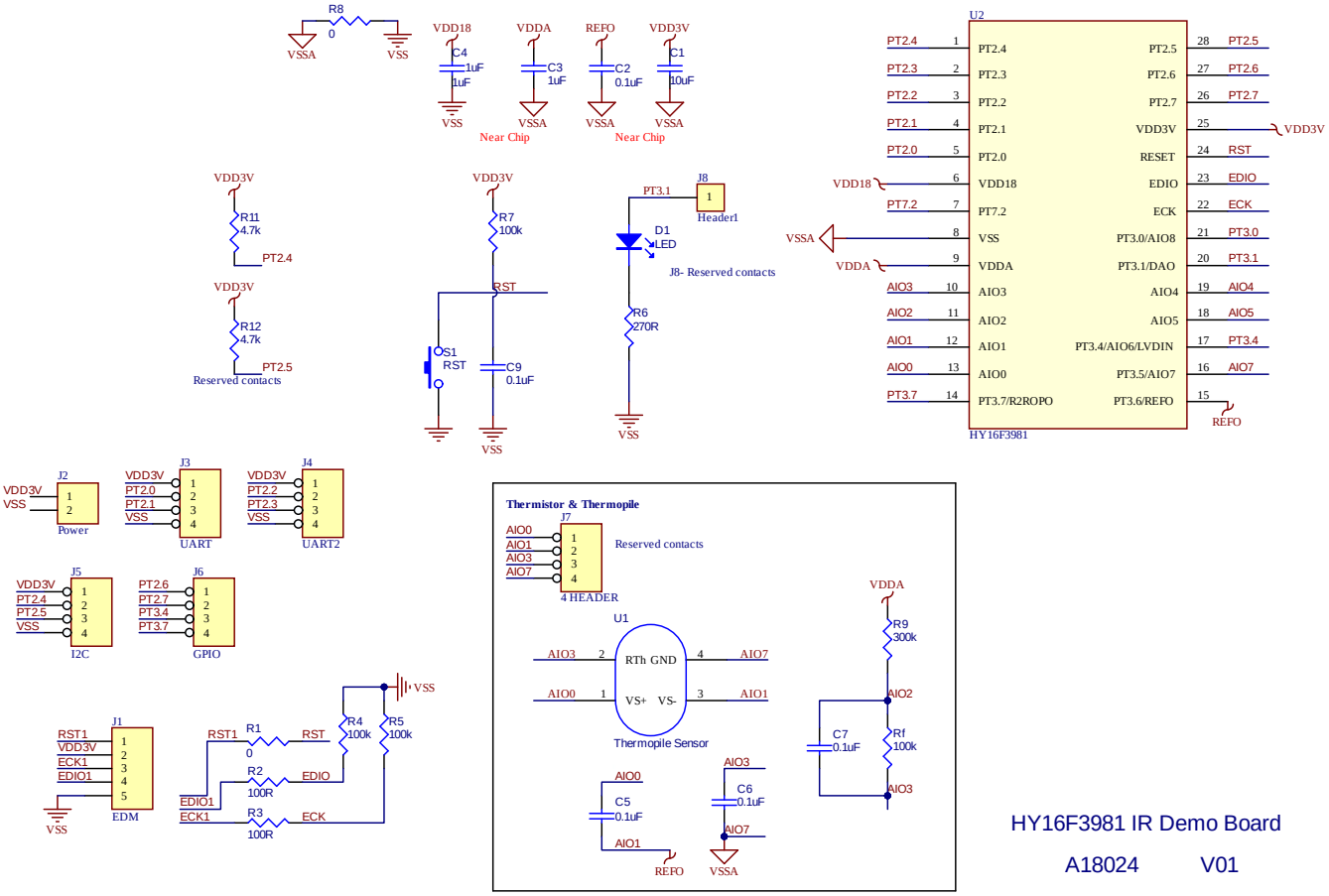


Figure 9 HY16F3981 IR Demo Board circuit diagram

Introduction of main components:

(1) MCU: HY16F3981 (Andes 32-bit MCU Core + HYCON 24-bit ΣΔADC & IA + MCU 64K Flash).

(2) J1 Port: Connected to the HY16F Mini Link for debug.

(3) J3&J4 Port: Transmit the measured Thermistor and Thermopile data through UART and UART2..

(4) J5&J6 Port: I2C&GPIO reserved for transmission.

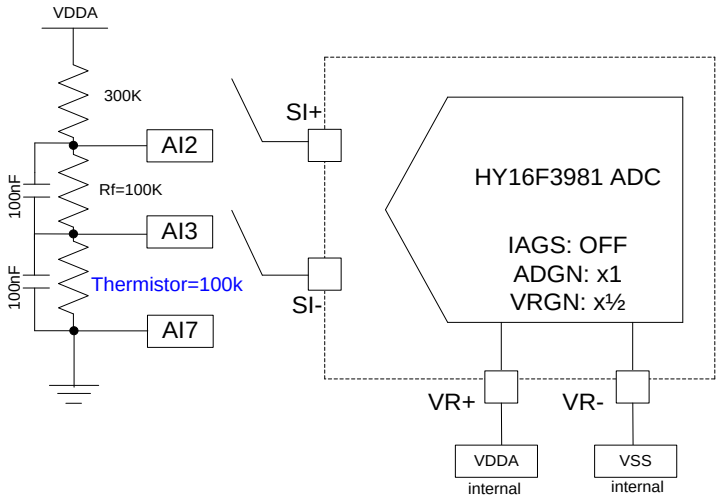
(5) J7 Port: Connect to Thermistor and Thermopile.

(6) J8 Port: Use the PT3.1 pin to control to get a complete Thermistor & Thermopile data. PT3.1 will make high and low changes, which can distinguish the data transmission output rate.

(7) R9&Rf: Divider resistance and Reference resistance. The voltage divider circuit is mainly used to measure the Thermistor channel data.

(8)Thermopile Sensor: AI0-AI1 channels are connected to Thermopile. AI3-AI7 channels are connected to Thermistor.

Thermistor resistance measurement:



The VDDA output of HY16F3981 is 2.4V. Through the 300k divider resistor, the voltage of the RF reference resistor 100k and the Thermistor resistor 100k can be controlled within the voltage signal range that the ADC can measure. Use the resistance value of Thermistor to look up the table to convert the ambient temperature. The resistance value of Thermistor can be compared with the RF reference resistance.

The following is the ADC RAW DATA calculation process for ADC measurement of RF and Thermistor resistance:

ADC_RF_P = AI2-AI3

Comb Filter Reset

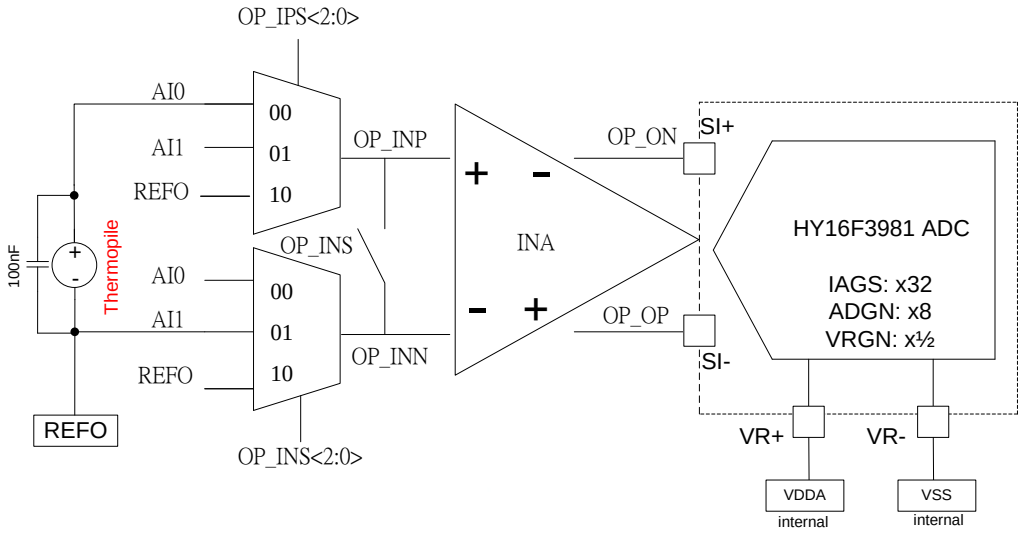
ADC_RF_N =AI3-AI2

Do ADC Chopper deduction offset

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
Do ADC Chopper deduction offset
ADC_RS(Thermistor)=(ADC_RS_P- ADC_RS_N)/2
First use the ratio of ADC_RF and ADC_RS to convert the resistance value, and then use the resistance value to look up the table to calculate the ambient temperature.

Thermopile Low Voltage Measurement:

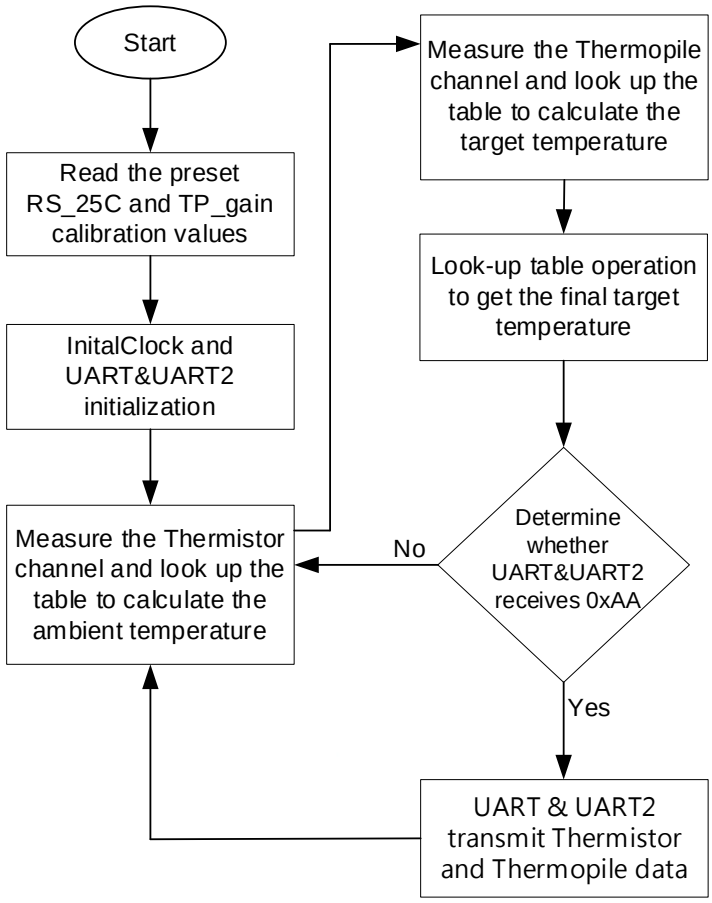


If IA chopper on is used for signal measurement, each ADC data obtained is already the ADC data processed by hardware Chopper, so there is no need to do Chopper through software.
Thermopile's ADC RAW DATA data measurement process is as follows:
ADC_TP=AI0-AI1 (IA chopper On)
However, when the IA chopper on is turned on, if the input impedance of the measurement signal source exceeds 10k ohms, there will be a problem of attenuation of the accuracy of the ADC.
In order to pursue more accurate Thermopile measurement, the following provides an alternative method of IA Chopper Off, which needs to be processed by software to make Chopper, which is also a way to use speed to exchange accuracy.
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)

Using ADC_TP, first convert the measured tiny voltage value, and then look up the obtained tiny voltage value (uV) to do a look-up table to convert the target temperature.

3.2. Software Description

FW : HY16F3981 IR Demo Code flowchart:



The following is as the UART transmission communication format:

1. When UART and UART2 receive 0xAA respectively, they will start to individually drop the channel data of Thermistor and Thermopile to the Host side.
2. UART and UART2 transmission mode command switching is as follows.

   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 software operation

4.1. Hardware Connection Instructions

SW: HY16F3981-AM01-V00(UART) GUI

FW: HY16F3981_IR_DemoCode_V00

UART transmits Thermistor data, UART2 transmits Thermoile data, and the hardware connection of the HY16F3981 IR Demo Board is shown in Figure 10.

In this article, the FTDI FT232R USB to UART Bridge is used to connect to J3 Port to receive Thermopile data. The chip can output 3.3V and directly supply it to the HY16F3981 chip. J4 port is connected to the second set of USB to UART Bridge to receive Thermopile data.

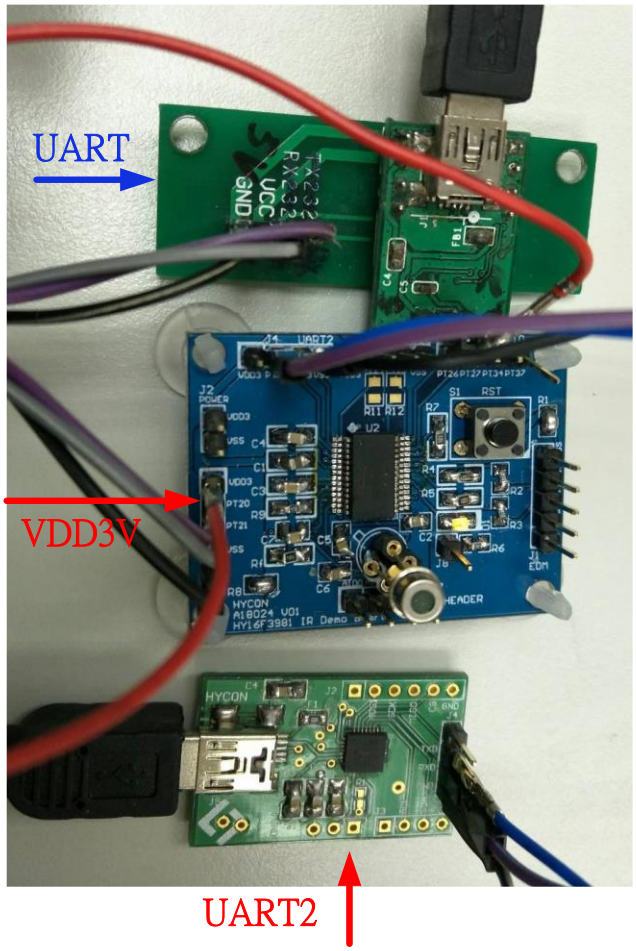


Figure 10 HY16F3981 IR Demo Board hardware connection diagram

4.2. Software interface

The GUI operation screen is shown in Figure 11. It can scan two channels of data of Thermistor and Thermopile at the same time, and the data is displayed in the graphical interface.

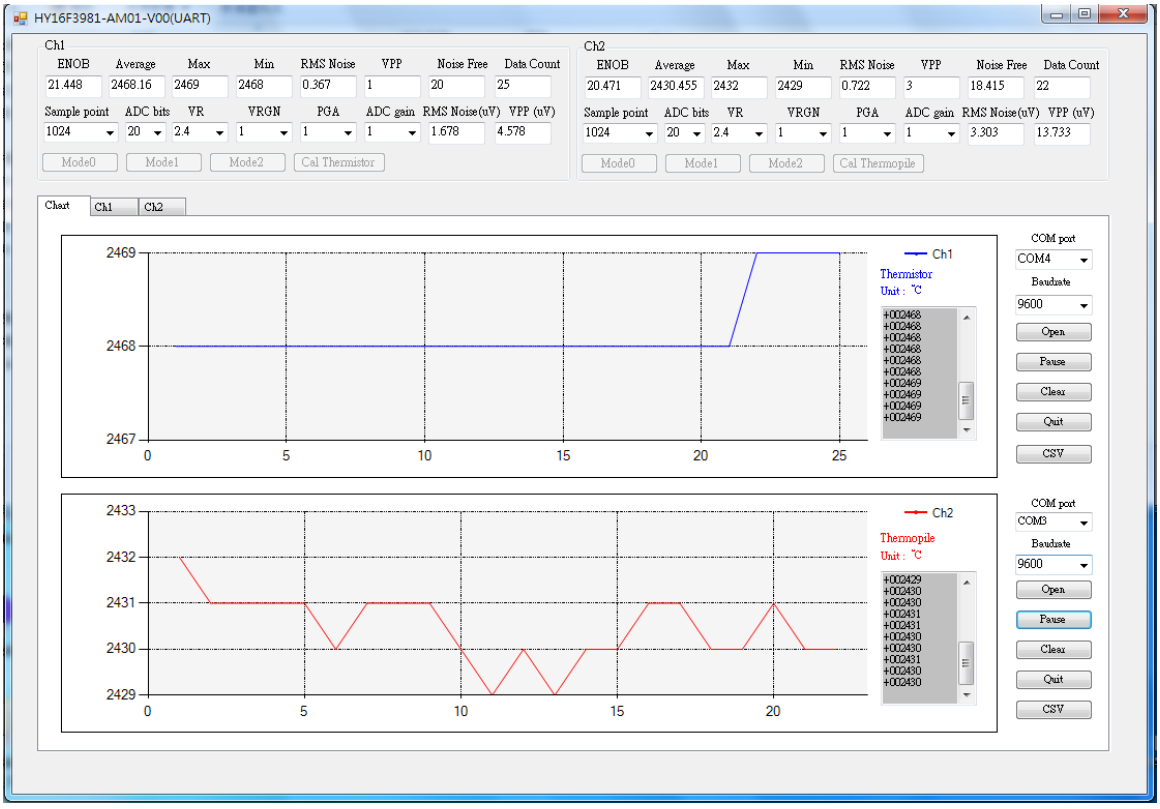


Figure 11 GUI HY16F3981-AM01-V00(UART) operation screen

Note: The COM port number corresponding to the open COM port needs to be the same as the port of the thermistor and the thermopile. As shown in Figure 12, from the device administrator, you can see two sets of COM ports, COM4 is connected to UART, corresponding to the Thermopile channel, COM3 is connected to UART2, corresponding to the Thermopile channel.

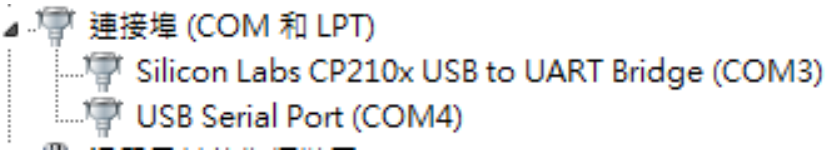


Figure 12 COM Port status observed by the computer device administrator

4.3. Software Operation Instructions

The HY16F3981-AM01-V00(UART) GUI provides the data display of Thermistor and Thermopile channels in several different Modes, as shown in Figure 13 below.

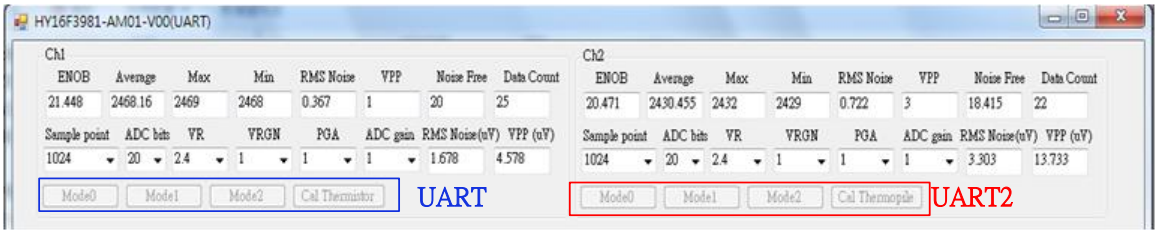


Figure 13 Thermistor and Thermopile switch different modes

Thermistor (Ch1) function description:

Mode0: Displays the measured Thermistor resistance. unit: ohm.

Mode1: Displays the measured ambient temperature of Thermistor. unit: temp.

Mode2: Displays the measured AIO3-AIO7 ADC Count. unit: Count.

Cal Thermistor: Correct the resistance of the thermistor. (Calibration temperature 25°C)

Thermopile (Ch2) function description:

Mode0: Displays the measured Thermopile voltage. unit: uV.

Mode1: Displays the measured target temperature of the Thermopile. unit: temp.

Mode2: Displays the measured AIO0-AIO1 ADC Count. unit: Count.

Cal Thermopile: Correct the voltage of the Thermopile. (External input voltage)

As shown in Figure 14, the user sets both the Thermistor and Thermopile to work in Mode2 in the transmission data. The temperature value display value 2500 represents 25.00°C.

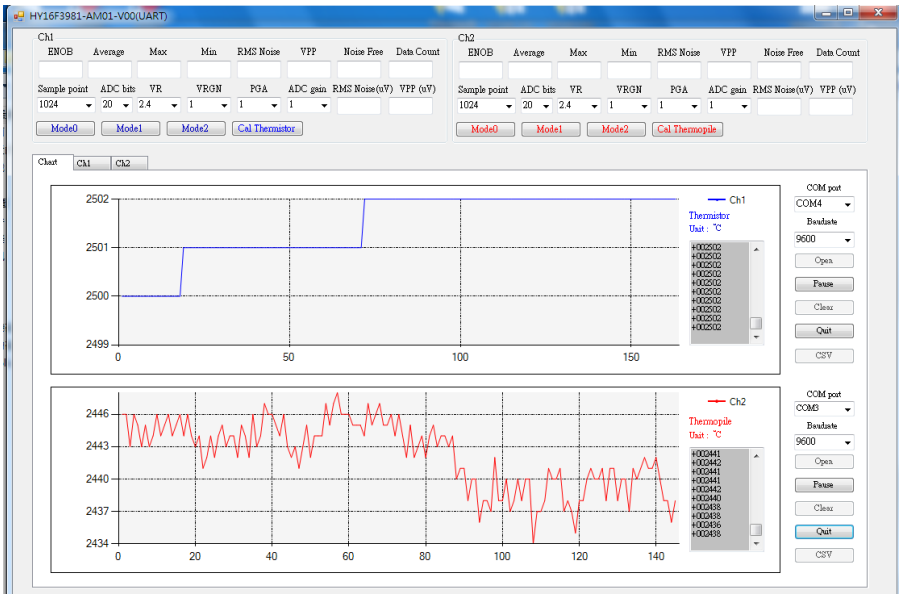


Figure 14 Thermistor channel and Thermopile channel display temperature (unit: °C)

As shown in Figure 15, the user touches the IR Sensor while transmitting data, and the IR Sensor is affected by temperature changes. Set both Thermistor and Thermopile to work

in Mode0. Thermistor channel displays resistance value (unit: ohm), Thermopile channel displays voltage value (unit: uV).

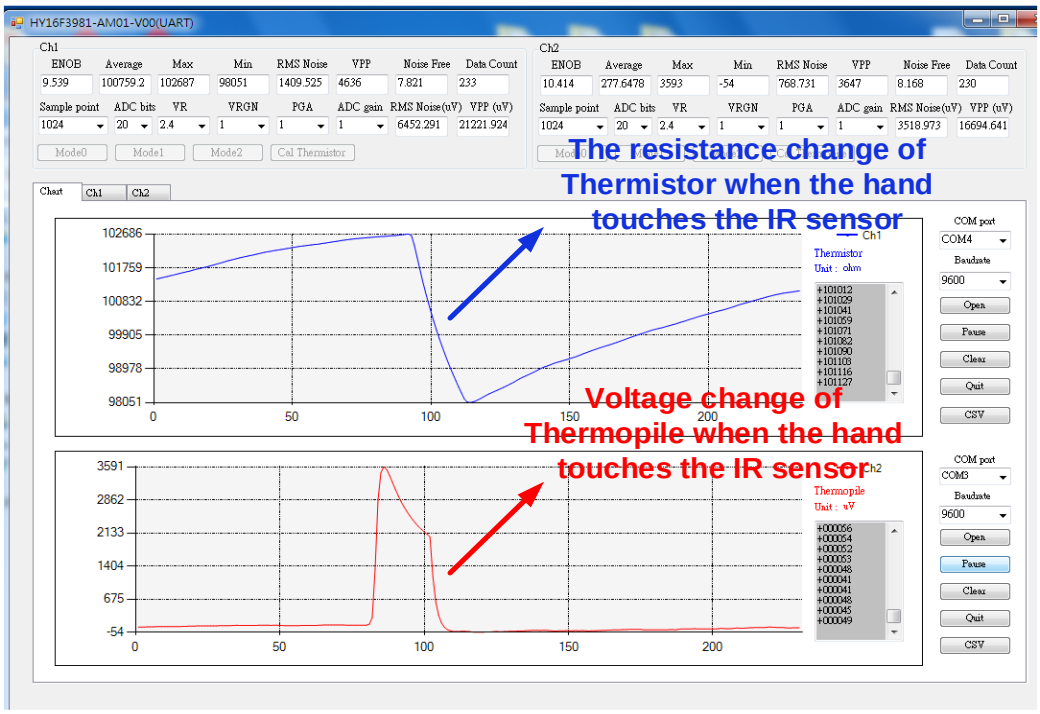


Figure 15 Thermistor channel and Thermopile channel of IR Sensor due to temperature changes

The data captured by the user can be saved as a csv file, which can be opened by Excel for data analysis.

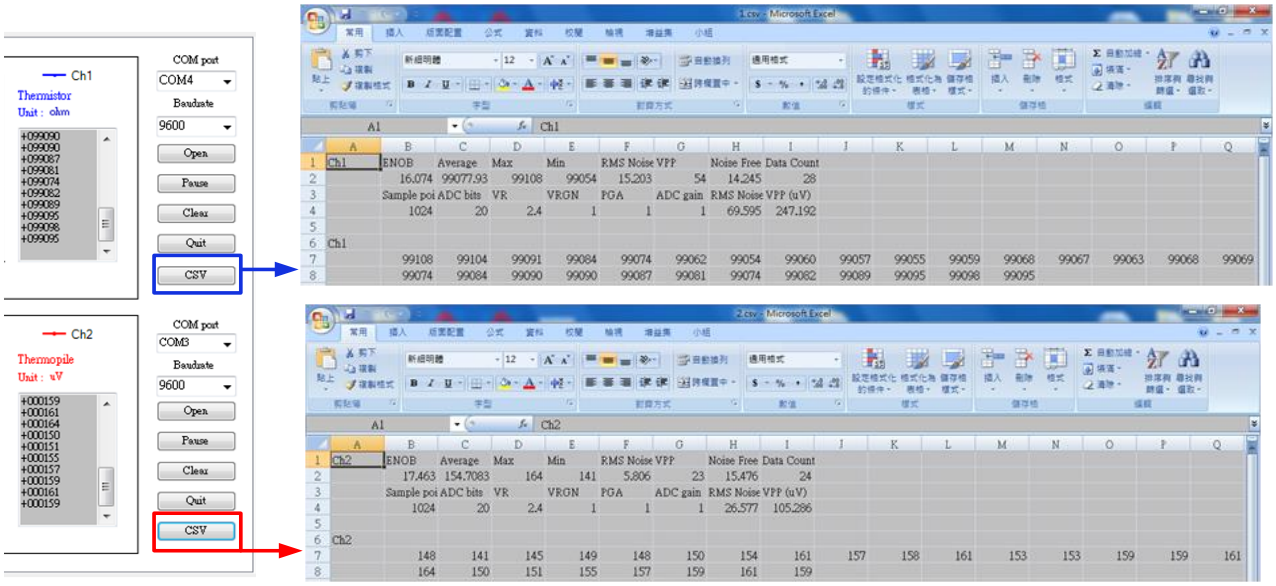


Figure 16 Thermistor and Thermopile channel are saved as csv files

4.4. Summary

In this article, a complete development kit for infrared temperature measurement applications is introduced for users' reference. According to the ADC Raw Data change of Thermopile and Thermopile, users can transmit it to the Host PC through UART for real-time dynamic data display, which is convenient for subsequent program development and evaluation in infrared applications.

5. Demo Code and related files

```
/*******************************************************************************
 *
 * 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 ;
/*-------------------------------------------------------------------------------*/
/*  Data Flash                                                                   */
/*-------------------------------------------------------------------------------*/
//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)
};


/*-------------------------------------------------------------------------------*/
/*  Function PROTOTYPES                                                          */
/*-------------------------------------------------------------------------------*/
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();
#if defined(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 Tempurature
      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;
  //可以從變數 Exception_link_pointer 找出進入 expection 的程式段
  asm volatile("add %0, %1, $lp":"=&r"(Exception_link_pointer):"r"(Exception_link_pointer_temp));

  int   Exception_ITYPE;
  Exception_ITYPE=0;
  //以下語句可以將 ITYPE(ir6)資料從暫存器中搬出到 C 環境中.
  asm volatile("mfsr %0,$ITYPE":"=&r"(Exception_ITYPE)); //ITYPE 資料存放在變數 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_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)
#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_SetIAInputChannel(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_SetIAInputChannel(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_SetIAInputChannel(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_SetIAInputChannel(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_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  代表兩個點之間分成 100 階數, count1 代表每一階的數值大小
    count2=(RS_kth-tempL)/count1;               //count2  代表總共的階數為多少
    TH_temperature=TH[a]*100-count2;            //Note : *100  代表兩個點之間分成 100 階數, 十位數與個位數代表小數點後兩位
    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]=ADCData2%10 | '0';    //unit
   ADCData2=ADCData2/10;
   Uart2TxBuffer[5]=ADCData2%10 | '0';    //ten
   ADCData2=ADCData2/10;
   Uart2TxBuffer[4]=ADCData2%10 | '0';    //hundred
   ADCData2=ADCData2/10;
   Uart2TxBuffer[3]=ADCData2%10 | '0';    //thousand
   ADCData2=ADCData2/10;
   Uart2TxBuffer[2]=ADCData2%10 | '0';    //ten thousand
   ADCData2=ADCData2/10;
   Uart2TxBuffer[1]=ADCData2%10 | '0';    //hundred thousand
   ADCData2=ADCData2/10;
   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_SelectIHOSC(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. References

[1] http://www.hycontek.com/wp-content/uploads/APD-SD18009_TC.pdf , HYCON Technology HY11P13 IR Measurement Application Note.

[2] http://www.hycontek.com/hy_mcu/DS-HY16F3981_TC.pdf , HYCON Technology HY16F3981 Datasheet.

[3] http://www.hycontek.com/hy_mcu/UG-HY16F3981_TC.pdf , HYCON Technology HY16F3981 User Guide.

7. Revision

The following describes the major changes made to the document, excluding the font and punctuation changes.

Version	Page	Data	Revision Summary
V01	All	2022/10/04	First edition