

### 1 目的

该文档的主要目的是描述怎样实现单片机和红外温度计 MLX90614 的 SMBus 通信，并简略介绍利用单片机对 MLX90614 的 PWM 输出脉冲进行测量并计算相关温度。代码是基于 STC12C5604AD 单片机的。实例给出了由 C 语言实现 MLX90614 读取物体温度，写入数据到 MLX90614、以及对 PWM 脉冲的测量和计算等功能。

### 2 C 代码

#### 2.1 由 MLX90614 读取温度部分

实例给出了应用 C 语言来实现从 MLX90614 中读取物体温度的程序。为了使程序操作和运行简单起见，整个项目被分为几个子项目。在该文档中整个 C 程序分为主文件 **SMBus.c**（用来实现对物体温度的读取、改变 SMBus 地址、改变发射率、以及改变 PWMCTRL 配置参数功能）；子文件 **SMBus\_CM.c**（具体描述 SMBus 通信的起始状态，终止状态，发送和接受数据）；子文件 **SMBus\_OP.c**（包含了利用 SMBus 通信由 MLX90614 读取数据，对 MLX90614 写入数据和 PEC 校验码计算的程序）；子文件 **Delay.c**（调用延迟函数）；子文件 **dec2hex.c**（将十进制转换为十六进制）；子文件 **CalTem.c**（根据十六进制数值计算温度读数）；子文件 **digitalLED.c**（在数字 LEDs 上显示温度读数）。在主文件的头文件中引用相应的子文件.h 文件，以将所有功能综合、链接起来。

```
//-----//
```

```
//-----//
```

#### 主文件 SMBus.c

```
//-----//
```

```
//-----//
```

红外温度计—MLX90614 中读取物体温度

描述：该文件是基于单片机 STC12C5604AD 编写的 C 语言程序，可通过 SMBus 实现对 MLX90614 温度数据的读取，对 MLX90614 写入数据的功能。实例给出了读取物体温度的程序，以及可根据十六进制数值计算温度数值并在数字 LEDs 上显示温度的程序，但是此为可选项，用户可根据自己的应用另行选择其他方式。

```
//-----//
```

头文件

```
//-----//
```

```
#include <STC12C5410AD.H>
```

```
#include "stdio.h"
```

```
#include "intrins.h"
```

```
#include "string.h"
```

```
#include "SMBus_CM.h" //引用 SMBus_CM.h 文件(包含起始状态，终止状态，发送，接收字节等)
```

```
#include "SMBus_OP.h" //引用 SMBus_OP.h 文件(包含由 MLX90614 中读取数据和写入数据)
```

```
#include "digitalLED.h" //引用 digitalLED.h 文件以在数字 LEDs 上显示温度读数
```

```
#include "CalTem.h" //引用 CalTem.h 文件来根据十六进制数值计算温度
```

```
#include "dec2hex.h" //引用 dec2hex.h 文件将十进制转换为十六进制
```

```
#include "Delay.h" //引用 Delay.h 文件
```

```
//-----//
```

宏定义 I/O 端口和 SMBus 信号输入输出方向

```
//-----//
```

```
#define _SDA_OUTPUT P1M0=0x10; P1M1=0x10; //设置 SDA 为开漏输出
```

```
#define _SDA_INPUT P1M0=0x10; P1M1=0x00; //设置 SDA 为高阻输入
```

```
#define _SCL_IO P1M0=0x08; P1M1=0x08; //设置 SCL 为开漏输出的 I/O 端口
```

```
sbit SDA = P1^4; //指定 SDA 线给 P14
```

```

sbit SCL = P1^3;                                     //指定 SCL 线给 P13
//-----//
主函数功能
作用：读取物体温度
//-----//
void main()
{
    unsigned char    slaveaddress;
    unsigned long int DATA;
    unsigned int      *mahm;

    _SCL_IO;                                     //引用宏定义-设置 SCL 为开漏式 I/O 口
    _SDA_OUTPUT;                                 //引用宏定义-设置 SDA 为开漏式输出

    SCL=0;                                       //
    Delay(1200);                                //SMBus 请求时间，将 PWM 模式转换为 SMBus 模式(至少为 2ms)
    SCL=1;                                       //
    while(1)
    {
        slaveaddress=MEM_READ(0x00,0x2E);
        //读取存于 MLX90614 EEPROM "0Eh"地址中的 SMBus 地址
        DATA=MEM_READ(slaveaddress,0x07);
        //基于上述地址由 MLX90614 的内存 07h 中读取物体温度
        mahm=CALTEMP(DATA);
        //基于所得的十六进制温度格式计算实际温度
        show(mahm,5);
        //在数字 LEDs 上显示计算所得温度
    }
}
//-----//
//-----//
子文件 SMBus_CM.c
//-----//
//-----//
该文档包含了 SMBus 通信的起始状态，终止状态，发送和接受字节等
//-----//
头文件
//-----//
#include <STC12C5410AD.H>
#include "SMBus_CM.h"                               //头文件中引用本身
#include "intrins.h"
#include "Delay.h"
//-----//
宏定义 I/O 端口和 SMBus 信号的方向
//-----//
#define _SDA_OUTPUT P1M0=0x10; P1M1=0x10;           //设置 SDA 为开漏输出
#define _SDA_INPUT P1M0=0x10; P1M1=0x00;            //设置 SDA 为高阻输入
#define _SCL_IO P1M0=0x08; P1M1=0x08;              //设置 SCL 为开漏式 I/O 端口
sbit SDA = P1^4;                                     //指定 SDA 线给 P14
sbit SCL = P1^3;                                     //指定 SCL 线给 P13
//-----//

```

函数名: start\_bit

功能: 在 SMBus 总线上产生起始状态

注解: 参考“系统管理总线说明书-版本 2.0”

```
//-----//
void start_bit()
{
    _SDA_OUTPUT;                //设置 SDA 为输出
    SDA=1;                      //设置 SDA 线为高电平
    _nop();_nop();
    SCL=1;                      //设置 SCL 线为高电平
    Delay(5);                   //在终止和起始状态之间产生总线空闲时间(Tbuf=4.7us 最小值)
    SDA=0;                      //设置 SDA 线为低电平
    Delay(5);
    //（重复）开始状态后的保持时间，在该时间后，产生第一个时钟信号
    //Thd:sta=4us 最小值
    SCL=0;                      //设置 SCL 线为低电平
    _nop();_nop();
}
//-----//
```

函数名: stop\_bit

功能: 在 SMBus 总线上产生终止状态

注解: 参考“系统管理总线说明书-版本 2.0”

```
//-----//
void stop_bit()
{
    _SDA_OUTPUT;                //设置 SDA 为输出
    SCL=0;                      //设置 SCL 线为低电平
    Delay(5);
    SDA=0;                      //设置 SDA 线为低电平
    Delay(5);
    SCL=1;                      //设置 SCL 线为高电平
    Delay(5);                   //终止状态建立时间(Tsu:sto=4.0us 最小值)
    SDA=1;                      //设置 SDA 线为高电平
}
//-----//
```

函数名: send\_bit

功能: 在 SMBus 总线上发送一位数据

```
//-----//
void send_bit(unsigned char bit_out)
{
    _SDA_OUTPUT;                //设置 SDA 为开漏输出以在总线上传送数据

    if(bit_out==0)              //核对字节的位
                                //如果 bit_out=1，设置 SDA 线为高电平
        SDA=0;

    else                          //如果 bit_out=0，设置 SDA 线为低电平
        SDA=1;

    _nop();                     //
    _nop();                     //Tsu:dat=250ns 最小值
    _nop();                     //
    SCL=1;                      //设置 SCL 线为高电平
}
```

```
Delay(4);           //时钟脉冲高电平脉宽(10.6us)
SCL=0;              //设置 SCL 线为低电平
Delay(4);           //时钟脉冲低电平脉宽
}
//-----//
函数名: receive_bit
功能: 在 SMBus 总线上接收一位数据
//-----//
unsigned char receive_bit()
{
    unsigned char bit_in;
    _SDA_INPUT;      //设置 SDA 为高阻输入
    SCL=1;           //设置 SCL 线为高电平
    Delay(2);
    if(SDA==1)        //从总线上读取一位, 赋给 bit_in
        bit_in=1;
    else
        bit_in=0;
    Delay(2);
    SCL=0;            //设置 SCL 线为低电平
    Delay(4);
    return bit_in;    //返回 bit_in 值
}
//-----//
函数名: slave_ack
功能: 由受控器件 MLX90614 中读取确认位
返回值: unsigned char ack
        1 - ACK
        0 - NACK
//-----//
unsigned char slave_ack()
{
    unsigned char ack;
    ack=0;
    _SDA_INPUT;      //设置 SDA 为高阻输入
    SCL=1;           //设置 SCL 线为高电平
    Delay(2);
    if(SDA==1)        //从总线上读取一位, 赋给 ack
        ack=0;
    else
        ack=1;
    Delay(2);
    SCL=0;            //设置 SCL 线为低电平
    Delay(4);
    return ack;
}
//-----//
发送一个字节
函数名: TX_byte
功能: 在 SMBus 总线上发送一个字节
参数: unsigned char TX_buffer (将要在总线上发送的字节)
注解: 先发送字节的高位
//-----//
```

```
void TX_byte(unsigned char TX_buffer)
{
    unsigned char Bit_counter;
    unsigned char bit_out;

    for(Bit_counter=8;Bit_counter;Bit_counter--)
    {
        if(TX_buffer&0x80)
            bit_out=1;           //如果 TX_buffer 的当前位是 1,设置 bit_out 为 1
        else
            bit_out=0;           //否则, 设置 bit_out 为 0
        send_bit(bit_out);       //发送 SMBus 总线上的当前位
        TX_buffer<<=1;           //核对下一位
    }
}
//-----//
接收一个字节
函数名: RX_byte
功能: 在 SMBus 总线上接收一个字节
参数: unsigned char ack_nack (确认位)
0 - 主控制器发送 ACK
1 - 主控制器发送 NACK
返回值: unsigned char RX_buffer (总线接收的字节)
注解: 先接收字节的高位
//-----//
unsigned char RX_byte(unsigned char ack_nack)
{
    unsigned char RX_buffer;
    unsigned char Bit_counter;
    for(Bit_counter=8;Bit_counter;Bit_counter--)
    {
        if(receive_bit()==1)     //由 SDA 线读取一位
        {
            RX_buffer<<=1;       //如果位为“1”, 赋“1”给 RX_buffer
            RX_buffer|=0x01;
        }
        else                     //如果位为“0”, 赋“0”给 RX_buffer
        {
            RX_buffer<<=1;
            RX_buffer&=0xfe;
        }
    }

    send_bit(ack_nack);          //发送确认位
    return RX_buffer;
}
//-----//
//-----//
子文件 SMBus_OP.c
//-----//
//-----//
该文档包含了 SMBus 通信时从 MLX90614 读取数据, 写入数据和 PEC 校验码计算的程序
//-----//
```

头文件

```
//-----//
#include <STC12C5410AD.H>
#include "SMBus_CM.h"           //引用 SMBus_CM.h 文件
#include "intrins.h"
#include "SMBus_OP.h"           //头文件中引用本身
#include "Delay.h"
//-----//

sbit SDA = P1^4;                //指定 MLX90614 的 SDA 线给单片机 P14 引脚
sbit SCL = P1^3;                //指定 MLX90614 的 SCL 线给单片机 P13 引脚
//-----//

计算 PEC 包裹校验码
函数名: PEC_cal
功能: 根据接收的字节计算 PEC 码
参数: unsigned char pec[], int n
返回值: pec[0] - 该字节包含计算所得 crc 数值
注解: 参考“系统管理总线说明书-版本 2.0”和应用指南“MCU 和 MLX90614 的 SMBus 通信”
//-----//
unsigned char PEC_cal(unsigned char pec[], int n)
{
    unsigned char crc[6];
    unsigned char Bitposition=47;
    unsigned char shift;
    unsigned char i;
    unsigned char j;
    unsigned char temp;
    do{
        crc[5]=0;                //载入 CRC 数值 0x000000000107
        crc[4]=0;
        crc[3]=0;
        crc[2]=0;
        crc[1]=0x01;
        crc[0]=0x07;
        Bitposition=47;          //设置 Bitposition 的最大值为 47
        shift=0;
        //在传送的字节中找出第一个“1”

        i=5;                     //设置最高标志位 (包裹字节标志)
        j=0;                     //字节位标志, 从最低位开始
        while((pec[i]&(0x80>>j))==0 && (i>0))
        {
            Bitposition--;
            if(j<7)
            {
                j++;
            }
            else
            {
                j=0x00;
                i--;
            }
        }
        //while 语句结束, 并找出 Bitposition 中为“1”的最高位位置
        shift=Bitposition-8;      //得到 CRC 数值将要左移/右移的数值“shift”
        //对 CRC 数据左移“shift”位

        while(shift)
```

```

{
for(i=5;i<0xFF;i--)
{
if((crc[i-1]&0x80) && (i>0)) //核对字节的最高位的下一位是否为"1"
{ //是 - 当前字节 + 1
temp=1; //否 - 当前字节 + 0
} //实现字节之间移动 "1"
else
{
temp=0;
}
crc[i]<=1;
crc[i]+=temp;
}

shift--;
}
//pec 和 crc 之间进行异或计算
for(i=0;i<=5;i++)
{
pec[i]^=crc[i];
}
}while(Bitposition>8);
return pec[0]; //返回计算所得的 crc 数值
}
//-----//
由 MLX90614 RAM/EEPROM 读取的数据
函数名: MEM_READ
功能: 给定受控地址和命令时由 MLX90614 读取数据
参数: unsigned char slave_addR (受控地址)
      unsigned char cmdR (命令)
返回值: unsigned long int Data
//-----//
unsigned long int MEM_READ(unsigned char slave_addR, unsigned char cmdR)
{
    unsigned char DataL; //
    unsigned char DataH; //由 MLX90614 读取的数据包
    unsigned char PEC; //
    unsigned long int Data; //由 MLX90614 返回的寄存器数值
    unsigned char Pecreg; //存储计算所得 PEC 字节
    unsigned char arr[6]; //存储已发送字节的缓冲器
    unsigned char ack_nack;
    unsigned char SLA;
    SLA=(slave_addR<<1);
begin:
    start_bit(); //发送起始位
    TX_byte(SLA); //发送受控器件地址, 写命令
    if(slave_ack()==0)
    {
        stop_bit();
        goto begin;
    } //发送命令
    TX_byte(cmdR);
    if(slave_ack()==0)
    {

```

```

        stop_bit();
        goto begin;
    }
    start_bit();
    TX_byte(SLA+1);
    if(slave_ack()==0)
    {
        stop_bit();
        goto begin;
    }
    DataL=RX_byte(0);

    DataH=RX_byte(0);
    PEC=RX_byte(ack_nack);
    if(ack_nack==1)
    //取决于 pec 计算, 如果 PEC 是不正确的, 发送 nack 并返回到 goto begin
    {
        stop_bit();
        goto begin;
    }
    stop_bit();
    arr[5]=(SLA);
    arr[4]=cmdR;
    arr[3]=(SLA+1);
    arr[2]=DataL;
    arr[1]=DataH;
    arr[0]=0;
    Pecreg=PEC_cal(arr,6);
    if(PEC==Pecreg)
    {
        ack_nack=0;
    }
    else
    {
        ack_nack=1;
    }
    Data=(DataH*256)+DataL;
    return Data;
}

//-----//
MLX90614 EEPROM 中写入数据
函数名: EEPROM_WRITE
功能: 根据命令写入相关数据到给定受控器件地址的 MLX90614
参数: unsigned char slave_addW (受控器件地址)
      unsigned char cmdW (命令)
      unsigned char DataL
      unsigned char DataH
//-----//
void EEPROM_WRITE(unsigned char slave_addW,unsigned char cmdW,unsigned char DataL,unsigned char DataH)
{
    unsigned char Pecreg;
    unsigned char SLA;
    unsigned char arr[6];

```



```

SLA=(slave_addW<<1);
arr[5]=0;
arr[4]=SLA;
arr[3]=cmdW;
arr[2]=DataL;
arr[1]=DataH;
arr[0]=0;
Pecreg=PEC_cal(arr,6);

begin:
start_bit();
TX_byte(SLA);
if(slave_ack()==0)
{
    stop_bit();
    goto begin;
}
TX_byte(cmdW);
if(slave_ack()==0)
{
    stop_bit();
    goto begin;
}
TX_byte(DataL);
if(slave_ack()==0)
{
    stop_bit();
    goto begin;
}
TX_byte(DataH);
if(slave_ack()==0)
{
    stop_bit();
    goto begin;
}
TX_byte(Pecreg);
if(slave_ack()==0)
{
    stop_bit();
    goto begin;
}
stop_bit();
Delay(200);

}
//-----//
//-----//
子文件 Delay.c
//-----//
//-----//
延迟函数
//-----//

```

```
#include <STC12C5410AD.H>
#include "intrins.h"
#include "Delay.h"
//-----//
函数名: Delay
功能: 延迟一定的时间
参数: N 表示延迟时间为 MCU 机器周期的 N 倍
注解: 机器周期是时钟周期(1/Fosc)的 12 倍,对于 STC12C5604AD, Fosc=11.0592MHz
//-----//
void Delay(unsigned int N)
{
    unsigned int i;
    for(i=0;i<N;i++)
        _nop_();
}
//-----//
//-----//
子文件 dec2hex.c
//-----//
//-----//
十进制转换为十六进制
//-----//
#include "dec2hex.h" //头文件中引用自身
#include "stdio.h"
//-----//
函数名: dec2hex
功能: 将十进制代码转换为十六进制代码
参数: float e (发射率数值)
返回值: unsigned int c
        c 是数组 c[4]的首地址
//-----//
unsigned int *dec2hex(float e)
{
    long int a=(e*65535)+0.5;
    int j,m=0,i=0;
    unsigned int b[16];
    unsigned int c[4],t;
    b[15]=0x0f;
    b[14]=0x0e;
    b[13]=0x0d;
    b[12]=0x0c;
    b[11]=0x0b;
    b[10]=0x0a;
    b[9]=0x09;
    b[8]=0x08;
    b[7]=0x07;
    b[6]=0x06;
    b[5]=0x05;
    b[4]=0x04;
    b[3]=0x03;
    b[2]=0x02;
    b[1]=0x01;
    b[0]=0;
```

```

while(a!=0)
{
    j=a%16;                //求模运算
    c[i++]=b[j];
    a=a/16;                //求余运算
    m++;
}
for(i=0;i<m/2;i++)
{
    t=c[i];                //
    c[i]=c[m-i-1];        //将数组 c[4]倒序
    c[m-i-1]=t;            //
}
return c;                //返回数组 c[4]的首地址
}
//-----
//-----
子文件 CalTem.c
//-----
//-----
根据十六进制数值计算温度
//-----
#include "CalTem.h"        //头文件中引用自身
//-----
函数名: CALTEMP
功能: 计算温度
参数: unsigned long int TEMP (由 MLX90614 中读到的数据)
返回值: unsigned int mah
mah 是数组 mah[5]的首地址
注解: 将十六进制代码转换为温度数据的公式为 T=(Data)*0.02-273.15
//-----
unsigned int *CALTEMP(unsigned long int TEMP)
{
    unsigned long int T;
    unsigned int a,b;
    unsigned int A4,A5,A6,A7,A8;
    unsigned int mah[5];
    T=TEMP*2;
    if(T>=27315)
    {
        T=T-27315;
        a=T/100;
        b=T-a*100;
        if(a>=100)
        {
            A4=a/100;
            a=a%100;
            A5=a/10;
            a=a%10;
            A6=a;
        }
        else if(a>=10)
        {

```

```
        A4=0;
        A5=a/10;
        a=a%10;
        A6=a;
    }
    else
    {
        A4=0;
        A5=0;
        A6=a;
    }
    if(b>=10)
    {
        A7=b/10;
        b=b%10;
        A8=b;
    }
    else
    {
        A7=0;
        A8=b;
    }
}
else
{
    T=27315-T;
    a=T/100;
    b=T-a*100;
    A4=9;
    if(a>=10)
    {
        A5=a/10;
        a=a%10;
        A6=a;
    }
    else
    {
        A5=0;
        A6=a;
    }
    if(b>=10)
    {
        A7=b/10;
        b=b%10;
        A8=b;
    }
    else
    {
        A7=0;
        A8=b;
    }
}

mah[4]=A4;
mah[3]=A5;
mah[2]=A6;
mah[1]=A7;
```

```

mah[0]=A8;
return mah;
}
//-----//
//-----//
子文件 digitalLED.c
//-----//
//-----//
根据计算所得数据在数字式 LEDs 上显示
//-----//
#include <STC12C5410AD.H>
#include "digitalLED.h" //头文件引用自身
//-----//
//定义 I/O 端口去控制数字式 LEDs
//-----//
sbit a0 = P1^7; //指定 a0 线给 P17
sbit a1 = P1^5; //指定 a1 线给 P15
sbit a2 = P1^6; //指定 a2 线给 P16
//-----//
函数名: show
功能: 在数字式 LEDs 上显示温度读数
参数: unsigned int mahh[],int number
注解: 显示存储在数组 mahh[5]的数据
//-----//
void show(unsigned int mahh[],int number)
{
    unsigned char yanshicon;
    unsigned char weicon;
    unsigned char code_LED_tab[10]={
        0xFC, //0
        0x60, //1
        0xDA, //2
        0xF2, //3
        0x66, //4
        0xB6, //5
        0xBE, //6
        0xE0, //7
        0xFE, //8
        0xF6, //9
    };
    P2=0; //关显示
    a0=1;
    a1=1;
    a2=1;
    //a2,a1,a0 控制显示位置, 可由 0-7 之间变化
    //控制显示位置, 当其值为 0-7 时, 对应数码管位 1-8 位
    while(1)
    {
        if(++yanshicon>200) //每 200 个扫描周期执行一次
        {
            yanshicon=0;
            if(++weicon>7) weicon=0; //显示位置轮流, 需要严格限制范围为 0-7
        }
    }
}

```

```
P2=0; //关显示
switch(weicon)
{
    case 0: //第 1 位显示内容
        P2=0; //关显示
        a0=0;
        a1=0;
        a2=0;
        break;
    case 1: //第 2 位显示内容
        P2=0; //关显示
        a0=1;
        a1=0;
        a2=0;
        break;
    case 2: //第 3 位显示内容
        P2=0; //关显示
        a0=0;
        a1=1;
        a2=0;
        break;
    case 3: //第 4 位显示内容
        P2=LED_tab[mahh[4]]; //显示数组 mahh[4]的数值
        a0=1;
        a1=1;
        a2=0;
        break;
    case 4: //第 5 位显示内容
        P2=LED_tab[mahh[3]]; //显示数组 mahh[3]的数值
        a0=0;
        a1=0;
        a2=1;
        break;
    case 5: //第 6 位显示内容
        P2=LED_tab[mahh[2]]; //显示数组 mahh[2]的数值
        a0=1;
        a1=0;
        a2=1;
        break;
    case 6: //第 7 位显示内容
        P2=LED_tab[mahh[1]]; //显示数组 mahh[1]的数值
        a0=0;
        a1=1;
        a2=1;
        break;
    case 7: //第 8 位显示内容
        P2=LED_tab[mahh[0]]; //显示数组 mahh[0]的数值
        a0=1;
        a1=1;
        a2=1;
        break;
    default:
        break;
}
```

```

    }
}
//-----//

```

### 2.2 写入数据到 MLX90614 部分(改变发射率、改变 SMBus 地址、改变 PWMCTRL 配置)

主文件 SMBus.c 中同样可以实现改变发射率、改变 SMBus 地址、改变 PWMCTRL 配置等功能。以下亦给出了对应的 C 程序，应用时只需简单地将现行主文件中的主函数（读取温度读数）替换以新的主函数（改变发射率，SMBus 地址等），并在头文件里加入或减去需要或不需要的子头文件即可。

```

//-----//
//-----//
功能：改变发射率
//-----//
//-----//
void main()
{
    unsigned char slaveaddress;
    unsigned int *Emv;           //定义指向整型变量的指针
    unsigned int EmvLO;
    unsigned int EmvHI;
    float Emissivity=0.5;        //给定一个发射率数值-用户可自己选择
    Emv=dec2hex(Emissivity);      //调用子函数(十进制转换为十六进制) 返回数列首地址
    EmvLO=*(Emv+2)<<4)+*(Emv+3); //载入发射率低字节
    EmvHI=*(Emv+0)<<4)+*(Emv+1);  //载入发射率高字节

    _SCL_IO;
    _SDA_OUTPUT;

    SCL=0;                       //
    Delay(1200);                 //SMBus 请求时间，将 PWM 模式转换为 SMBus 模式(至少为 2ms)
    SCL=1;                       //

    slaveaddress=MEM_READ(0x00,0x2E); //得到存于 EEPROM "0Eh"中的受控器件地址
    EEPROM_WRITE(slaveaddress,0x24,0x00,0x00); //首先写入数据 0x0000 到 EEPROM "04h"
    EEPROM_WRITE(slaveaddress,0x24,EmvLO,EmvHI); //写入新发射率到 EEPROM "04h"
                                           //重启以激活
}
//-----//
//-----//

```

### 功能：改变 SMBus 地址

```

//-----//
void main()
{
    unsigned char slaveaddress;
    unsigned int DataLO;
    unsigned int DataHI;
}

```

```

DataLO=0x5A;           //载入受控器件地址低字节 (用户需自己选择)
DataHI=0x00;           //载入受控器件地址高字节
_SCL_IO;
_SDA_OUTPUT;

SCL=0;                 //
Delay(1200);           //SMBus 请求时间, 将 PWM 模式转换为 SMBus 模式(至少为 2ms)
SCL=1;                 //

slaveaddress=MEM_READ(0x00,0x2E); //得到存于 EEPROM "0Eh"的旧的受控器件地址

EEPROM_WRITE(slaveaddress,0x2E,0x00,0x00); //写入数据 0x0000 到 EEPROM "0Eh"

EEPROM_WRITE(slaveaddress,0x2E,DataLO,DataHI); //写入新的受控地址到 EEPROM "0Eh"
//需要重启以激活
}
//-----//
//-----//
功能: 改变 PWMCTRL 配置, 配置为 PWM 输出格式
//-----//
//-----//
void main()
{
    unsigned char slaveaddress;
    unsigned char PWMCTRLO;
    unsigned char PWMCTRHI;
    PWMCTRLO=0x07;
    //PWMCTRL 低字节 (0x07 代表选定单个 PWM 模式, 使能 PWM, 并且 SDA 引脚设置为 Push-Pull 推挽式)
    PWMCTRHI=0x02;
    //PWMCTRL 高字节 (0x02 代表 PWM 周期为 1.024ms*1, 并且无 PWM 重复)

    _SCL_IO;
    _SDA_OUTPUT;

    SCL=0;                 //
    Delay(1200);           //SMBus 请求时间, 将 PWM 模式转换为 SMBus 模式(至少为 2ms)
    SCL=1;                 //

    slaveaddress=MEM_READ(0x00,0x2E); //读取存于 MLX90614 EEPROM "0Eh"中的 SMBus 地址
    EEPROM_WRITE(slaveaddress,0x22,0x00,0x00); //写入数据 0x0000 到 EEPROM "02h"

    EEPROM_WRITE(slaveaddress,0x22,PWMCTRLO,PWMCTRHI);
    //写入新的 PWMCTRL 数值到 EEPROM "02h"
    //需要重启以激活
}
//-----//
//-----//
功能: 改变 PWMCTRL 配置, 配置为 SMBus 输出格式
//-----//
//-----//

```



## 2.3 子文件对应的.h 文件介绍

17 / 23

```
//-----//
SMBus_OP.h
//-----//
头文件保护
//-----//
#ifndef SMBUS_OP_H
#define SMBUS_OP_H
//-----//
函数声明
//-----//
unsigned char PEC_cal(unsigned char pec[],int n);
unsigned long int MEM_READ(unsigned char slave_addR,unsigned char cmdR);
void EEPROM_WRITE(unsigned char slave_addW,unsigned char cmdW,unsigned char DataL,unsigned char
DataH);

#endif
//-----//

//-----//
Delay.h
//-----//
函数声明
//-----//
void Delay(unsigned int N);
//-----//

//-----//
dec2hex.h
//-----//
函数声明
//-----//
unsigned int *dec2hex(float e);
//-----//

//-----//
CalTem.h
//-----//
函数声明
//-----//
extern unsigned int *CALTEMP(unsigned long int TEMP);
//-----//

//-----//
digitalLED.h
//-----//
函数声明
//-----//
extern void show(unsigned int mahh[],int number);
//-----//
//-----//
```

### 3 根据 PWM 输出模式计算温度读数

MLX90614 可以设置为 PWM 输出模式，实例给出用 STC MCU 定时器功能实现对温度计 PWM 格式的读出，并计算和显示相关温度。(注意，本例中 MLX90614 的输出方式为单个 PWM 格式，PWM 的周期为 1.024ms，显示为物体温度读数。) 关于 PWM 输出方式的介绍，具体请参考应用指南-由 MLX90614 读取 PWM 格式数据并利用 PIC 18 MCU 实现温度计算。

```
//-----//
//-----//
主文件 PWM.c
//-----//
//-----//
头文件
//-----//
#include <STC12C5410AD.H>
#include "stdio.h"
#include "math.h"
#include "intrins.h"
#include "string.h"
#include "PWM_display.h" //引用 PWM_display.h (在数字式 LEDs 上显示温度读数)
//-----//
宏定义 I/O 端口
//-----//
sbit P3_3=P3^3; //利用单片机外部中断源 1 (INT1) 端口测量 SDA 引脚 PWM 脉冲
//-----//
功能：根据 PWM 脉冲宽度和周期（占空比）来计算需要显示的温度数据
//-----//
int Calculate(unsigned int t1pwm,unsigned int T1pwm)
{
    int DC;
    long int T;
    int T1;
    int K;
    int T1max=120; //物体温度的最大值，存于 EEPROM 00h 地址，用户可自行设置
    int T1min=-20; //物体温度的最小值，存于 EEPROM 01h 地址，用户可自行设置
    K=2*(T1max-T1min);

    DC=t1pwm*100000/T1pwm; //计算占空比，乘以 100,000 将小数点右移 5 位
    T=(DC-0.125*100000)*K+T1min*100000;
    // 根据公式  $T_{out} = [2(DC - 0.125)(T_{max} - T_{min})] + T_{min}$  计算单个 PWM 模式下物体温度，DC 为占空比
    T1=T/1000; //为实现分辨率为 0.01℃，结果需除以 1000
    return T1;
}
//-----//
函数名：主函数
功能：利用 STC MCU 外部中断源端口 1、定时器 0 和 1 实现对 PWM 脉冲宽度和周期的测量
//-----//

void main(void)
```

```

{
    unsigned int A;
    unsigned int B;
    unsigned int C;
    unsigned int D;
    unsigned int Data1;
    unsigned int Data2;
    int Data;

    TMOD=0x19;           //设置定时器 0 (GATE=1) 和定时器 1 (GATE=0) 都为定时方式 1
    TH0=0x00;            //TH0,TL0 清零
    TL0=0x00;

    ET0=1;               //开启定时器 0 中断允许, 允许定时器 0 中断
    ET1=1;               //开启定时器 1 中断允许, 允许定时器 1 中断
    EA=1;                //开启全局中断允许, 允许所有中断
    TL1=0x00;            //TH1,TL1 清零
    TH1=0x00;

    EX1=0;               //关 INT1 中断

    while(P3_3==1)        //等待 INT1 引脚低电平
    {}

    while(P3_3==0)        //等待 INT1 引脚高电平
    {}

    TR0=1;                //开启定时器 0 开始计数
    TR1=1;                //开启定时器 1 开始计数
    while(P3_3==1)        //等待 INT1 引脚低电平
    {}

    TR0=0;                //停止定时器 0 计数
    C=TL0;                //T0 低字节计数值送到 C
    D=TH0;                //T0 高字节计数值送到 D
    while(P3_3==0)        //等待 INT1 引脚高电平
    {}

    TR1=0;                //停止定时器 1 计数
    A=TL1;                //T1 低字节计数值送到 A
    B=TH1;                //T1 高字节计数值送到 B

    Data2=(D<<8)+C;
    Data1=(B<<8)+A;

    Data=Calculate(Data2,Data1); //调用函数, 根据两个计数器数值计算温度
    display(Data);               //显示温度数据
}

//-----//
//-----//
子文件 PWM_display.c
//-----//
//-----//
头文件
//-----//
#include <STC12C5410AD.H>
#include "Intrins.h"
#include "PWM_display.h"

```

```
//-----//
I/O 端口的宏定义
//-----//
sbit a0 = P1^7;           //指定 P17 给 a0 线
sbit a1 = P1^5;           //指定 P15 给 a1 线
sbit a2 = P1^6;           //指定 P16 给 a2 线
//-----//

unsigned char yanshicon;
unsigned char weicon;

unsigned char code LED_tab[10]={

                                0xFC,           //0
                                0x60,           //1
                                0xDA,           //2
                                0xF2,           //3
                                0x66,           //4
                                0xB6,           //5
                                0xBE,           //6
                                0xE0,           //7
                                0xFE,           //8
                                0xF6,           //9

};

void display(int Data)
{
    int A4,A5,A6,A7,A8;
    if(Data>=10000)
    {
        A4=Data/10000;
        Data=Data%10000;
        A5=Data/1000;
        Data=Data%1000;
        A6=Data/100;
        Data=Data%100;
        A7=Data/10;
        Data=Data%10;
        A8=Data;
    }
    else if(Data>=1000)
    {
        A4=0;
        A5=Data/1000;
        Data=Data%1000;
        A6=Data/100;
        Data=Data%100;
        A7=Data/10;
        Data=Data%10;
        A8=Data;
    }
    else if(Data>=100)
    {
        A4=0;
    }
}
```

```

    A5=0;
    A6=Data/100;
    Data=Data%100;
    A7=Data/10;
    Data=Data%10;
    A8=Data;
}
else if(Data>=10)
{
    A4=0;
    A5=0;
    A6=0;
    A7=Data/10;
    Data=Data%10;
    A8=Data;
}
else
{
    A4=0;
    A5=0;
    A6=0;
    A7=0;
    A8=Data;
}

P2=0; //关显示
a0=1;
a1=1;
a2=1;
//a2,a1,a0 控制显示位置, 当其值为 0-7 时, 对应数码管位 1-8 位
while(1)
{
    if(++yanshicon>200) //每 200 个扫描周期执行一次
    {
        yanshicon=0;
        if(++weicon>7) weicon=0; //显示位置轮流, 需要严格限制范围为 0-7
        P2=0; //关显示
        switch(weicon)
        {
            case 0: //第 1 位显示内容
                P2=0; //关闭显示
                a0=0;
                a1=0;
                a2=0;
                break;
            case 1: //第 2 位显示内容
                P2=0; //关闭显示
                a0=1;
                a1=0;
                a2=0;
                break;
            case 2: //第 3 位显示内容
                P2=0; //关闭显示
                a0=0;
                a1=1;
                a2=0;
        }
    }
}

```

```
break;
case 3:           //第 4 位显示内容
    P2=LED_tab[A4]; //显示 A4
    a0=1;
    a1=1;
    a2=0;
break;
case 4:           //第 5 位显示内容
    P2=LED_tab[A5]; //显示 A5
    a0=0;
    a1=0;
    a2=1;
break;
case 5:           //第 6 位显示内容
    P2=LED_tab[A6]; //显示 A6
    a0=1;
    a1=0;
    a2=1;
break;
case 6:           //第 7 位显示内容
    P2=LED_tab[A7]; //显示 A7
    a0=0;
    a1=1;
    a2=1;
break;
case 7:           //第 8 位显示内容
    P2=LED_tab[A8]; //显示 A8
    a0=1;
    a1=1;
    a2=1;
break;
default:
break;
    }
}

}

}

//-----//
//-----//
PWM_display.h
//-----//
//-----//
函数声明
//-----//
void display(int Data);
//-----//
//-----//
```