MLX90640 驱动库说明文档

目录

目录

1. 概述

为了正确使用 MLX90640 的 API 库，下面的 4 个文件应该被包含（include）到您的 C 工程内：

- MLX90640_I2C_Driver.h - 与 I2C 接口有关的函数、变量等定义声明头文件
- MLX90640_I2C_Driver.cpp 或者 MLX90640_SWI2C_Driver.cpp - I2C 接口相关函数具体实现文件
- MLX90640_API.h -MLX90640 专用功能头文件
- MLX90640_API.cpp -MLX90640 专用功能（函数）实现文件

2. I2C 接口驱动

实现标准 I2C 接口通讯。用户应该根据实际驱动器（MCU 或者 DSP、PLC 等）修改此驱动文件内容。因 MLX90640 的驱动 API 需要调用接口驱动函数，故此在修改过程中不要改变函数体声明样式。
与 I2C 接口相关的有 2 个文件：

2.1. MLX90640_I2C_Driver.cpp

如果您使用的是硬件 I2C 接口（非用 IO 模拟）时，请将此文件添加到 C 工程。用户应该修改文件内容以适应具体的 MCU。大部分的 MCU 都具有硬件 I2C 的功能以及官方的驱动程序。

2.2. MLX90640_SWI2C_Driver.cpp

- 此文件通过 2 根标准的 GPIO 管脚，采用软件模拟的方式实现 I2C 接口驱动功能。用户应定义 IO 引脚（SDA 和 SCL）并且保证符合 I2C 的通讯时序要求。
- 定义 IO：I2C 的数据管脚应定义为标准输入输出（InOut）类型，名称为“sda”。I2C 的时钟管脚应定义为输出（Out），名称为“scl”。

定义 IO 电平：为了实现正确的 I2C 通讯，数据管脚和时钟管脚的电平应满足以下定义要求：

```
#define LOW 0; -管脚输出低电平
#define HIGH 1; -管脚输出高电平
#define SCL_HIGH scl = HIGH; - I2C 时钟管脚输出高电平定义
#define SCL_LOW scl = LOW; - I2C 时钟管脚输出高电平定义
#define SDA_HIGH sda.input(); - I2C 数据输出高电平定义
#define SDA_LOW sda.output();sda = LOW; - I2C 数据输出低电平定义
```

管脚 sda 在此文件中被定义为不具备开漏输出功能的 IO，这样的定义使此管脚功能近似于开漏管脚的特性（以符合 I2C 接口要求）。如果您的 MCU 支持开漏功能，则此管脚的高和低不用特殊处理。

- 设置 I2C 通讯速率：这是软件模拟 I2C 时专用功能，用于时钟方波延时。不同的 MCU 以及指令周期会直接影响此延时函数的实际延时时长，所以用户应该修改此延时函数，以便执行正确的期望的通讯速率。默认的延时函数如下：

```
void Wait(int freqCnt)
{
  int cnt;
  for(int i = 0;i<freqCnt;i++)
  {
    cnt = cnt++;
  }
}
```

应修改延时函数 Wait()的内容以实现期望通讯频率，通过修改全局变量 freqCnt 可以对函数延时进行调整，freqCnt 值越小时通讯频率越高。

2.3. I2C 驱动函数

I2C 驱动函数有 4 个，这 4 个函数完成 MCU 与 MLX90640 的通讯。用户应该修改这几个函数，当然修改过程中不要改变函数声明定义。

2.3.1. I2C 初始化函数

void MLX90640_I2CInit(void)

此函数用于初始化 I2C 的两根管脚（sda 和 scl）以及相关的硬件模块。I2C 总线的两根线初始状态应为高电平。默认状态下还会在此函数中发送一个 I2C 停止信号。

例如：

1.在正式开始与 MLX90640 通讯以前，首先要完成的是 MLX90640 的 I2C 接口初始化

```
main.c
…一些定义…
..MCU 初始化
MLX90640_I2CInit();
…
…MLX90640 通讯指令//通讯前必须已经完成了 MLX90640_I2CInit();
…用户其它代码
```

2.3.2. 通讯速率设置函数

void MLX90640_I2CFreqSet(int freq)

此函数用于动态的调整 I2C 通讯速率。此函数有一个整型参数，此参数用来设置硬件 I2C 的通讯速率或者软件模拟 I2C 接口中的 Wait()函数周期数。当使用硬件 I2C 时，此函数中实际执行的是硬件 I2C 的标准库函数（一般由 MCU 厂家提供），而当使用软件模拟

I2C 时（使用两个标准 GPIO），I2C 通讯速率受控于 Wait 函数使用的一个全局变量（Wait 的周期数）。

例如：

1.使用硬件 I2C，通讯速率设置为 1MHz：

```
MLX90640_I2CFreqSet(1000); //实际执行的是 MCU 硬件 I2C 的标准函数
//参数即是通讯速率值，（
1000 表示 1000kHz，即 1MHz）
```

2.使用软件 I2C，通讯速率设置为 400kHz

```
MLX90640_I2CFreqSet(20); //参数值表示的是 Wait 等待的周期数，
//实际时长依赖具体的 MCU 性能，用户可以调整 Wait 函数内容，以
实现 400kHz
```

2.3.3. 连续字读取函数

```
int MLX90640_I2CRead(uint8_t slaveAddr,
uint16_t startAddress,
uint16_t nMemAddressRead,
uint16_t *data)
```

此函数从指定地址的 MLX90640 设备中读取若干个字的数据。需要注意的是在连续读取的过程中，读取的指针在 MLX90640 内部是自增的，即：读取完一个字后会自动指向下一个字，此时继续读取即可（无需重新设定要读取的地址值），这样有最快的读取效率。但对于一次新的读取操作，则必须指定首个要读取的字的地址。

此函数返回 0 表示读取成功，如果返回-1 表示 MLX90640 在被读取过程中返回了 NACK（非应答）信号，读取失败。此函数的参数说明如下：

- **uint8_t slaveAddr** – MLX90640 的设备地址 (默认地址为 0x33)
- **uint16_t startAddress** – 要读取的存储于 MLX90640 内部的首个字的地址值. 计算校正参数存储于 EEPROM 内，地址为：0x2400 to 0x273F，实时测量数据保存于 RAM 内，地址为 0x0400 to 0x073F。
- **uint16_t nMemAddressRead** – 要读取多少个字（每个字 2 字节）
- **uint16_t *data** – 读取到的若干字保存的字指针

例如：

1. 从 EEPROM 中读取单个字:

```
uint16_t eeValue;
MLX90640_I2CRead (0x33, 0x240C, 1, &eeValue);
//EEPROM 0x240C 单元格的数据被读取
//到 eeValue 变量中
```

2. 从 RAM 中读取完整的一帧实时数据:

```
static uint16_t frameData[832];
MLX90640_I2CRead(0x33, 0x0400, 832, frameData); //完整的一帧数据被读取到了
//frameData 数据中
```

2.3.4. 修改单个字函数

int MLX90640_I2CWrite (uint8_t slaveAddr,

uint16_t writeAddress, uint16_t data)

此函数向指定存储器地址写入一个字（即：修改指定字的内容）。返回 0 表示写入成功，返回-1 表示 MLX90640 返回了非应答信号 NACK，返回-2 表示写入后重新再读取出来后发现与写入值不同。

函数参数说明如下：

- **uint8_t slaveAddr** – MLX90640 的设备地址（默认地址为 0x33）
- **uint16_t writeAddress** – 要向设备存储器内的哪个字地址写入数据
- **uint16_t data** – 要写入的数据

例如：

int status;

status = MLX90640_I2CWrite(0x33, 0x800D, 0x0901); //向地址 0x800D（控制寄存器）

//写入新值 0x0901

返回值保存于 status 中，若为 0 则表示写入成功

3. MLX90640 API

用户不要修改此文件内容。

3.1. MLX90640 API 函数

3.1.1. 设置测量分辨率函数

int MLX90640_SetResolution(uint8_t slaveAddr, uint8_t

resolution)

此函数用于设置 MLX90640 的测量分辨率值（0~3 表示分辨率为 16~19 位）。需要注意的是在下次重新上电时，此函数修改的值会自动恢复为默认值（即：此函数修改后不会保存，掉电即失）。返回 0 表示设置成功，-1 表示设备未应答，-2 表示重新读取后发现不是预期的值。

例如：

设置分辨率为 19 位：*MLX90640_SetResolution(0x33,0x03);*

3.1.2. 读取测量分辨率函数

int MLX90640_GetCurResolution (uint8_t slaveAddr)

此函数用于读取当前的测量分辨率（0~3），若此函数返回了-1 则表示读取失败。

3.1.3. 设置测量速率函数

int MLX90640_SetRefreshRate (uint8_t slaveAddr,

uint8_t refreshRate)

此函数用于设置 MLX90640 的测量速率（即：每秒测量几帧数据），参数值可以是 0~7 代表 0.5、1、2、4、8、16、32 和 64Hz。返回 0 表示设置成功，-1 表示设备未应答，-2 表示重新读取后发现不是预期的值。

例如：

设置为每秒测量 16 帧： MLX90640_SetRefreshRate *(0x33,0x05);//5 的意思就是 16Hz*

3.1.4. 读取测量分辨率函数

int MLX90640_GetRefreshRate (uint8_t slaveAddr)

此函数用于读取当前的测量速率值（0~7），若此函数返回了-1 则表示读取失败。

3.1.5. 读取完整实时数据

int MLX90640_GetSubPageNumber (uint16_t *frameData)

此函数用于读取一次完整的测量数据（832 个字），返回值表示当前帧是哪个子页（0 或 1）。

例如：

```
static int mlx90640Frame[834];
int subPage;
subPage = MLX90640_GetSubPageNumber(mlx90640Frame);
```

3.1.6. 设置子页模式为 TV

int MLX90640_SetInterleavedMode (uint8_t slaveAddr) 6 / 9

此函数用于设置测量分布模式为 TV 模式(行交错模式)。返回 0 表示设置成功,-1 表示设备未应答,-2 表示重新读取后发现不是预期的值。

3.1.7. 设置子页模式为棋盘

int MLX90640_SetChessMode (uint8_t slaveAddr)

此函数用于设置测量分布模式为棋盘模式(像素交错模式)。返回 0 表示设置成功,-1 表示设备未应答,-2 表示重新读取后发现不是预期的值。

3.1.8. 读取子页测量模式

int MLX90640_GetCurMode (uint8_t slaveAddr)

此函数用于读取当前的测量分布模式,返回 0 表示工作于 TV 模式,返回 1 表示工作于棋盘模式。

3.1.9. 读取全部校正参数

int MLX90640_DumpEE(uint8_t slaveAddr, uint16_t *eeData)

此函数用于读取保存于 EEPROM 内的全部 832 个字的温度计算校准数据,保存于 eeData 数组内。

例如:

```
static uint16_t eeMLX90640[832];
int status;
status = MLX90640_DumpEE (0x33, eeMLX90640); //eeMLX90640 数组内是读取得到的
//EEPROM 参数数据
```

3.1.10. 解析校正参数为计算参数

int MLX90640_ExtractParameters(uint16_t * eeData,

paramsMLX90640 *mlx90640)

此函数用来将 3.1.9 中得到的所有 EEPROM 数据解析到定义于 MLX90640_API.h 中的自定义结构体变量中,此变量特别重要,是完成各种计算的必需参数(以下简称“计算参数”)。当解析完成后,原来的 EEPROM 数据基本再没有意义了(可以根据需要释放,也可以不管)。此函数返回-7 表示提供的 EEPROM 参数错误。

例如:

```
static uint16_t eeMLX90640[832];
paramsMLX90640 mlx90640;
```

```
int status;
status = MLX90640_DumpEE (0x33, eeMLX90640);//将 EEPROM 数据读取到
eeMLX90640 数组中
status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);//将
eeMLX90640 数组中
//的数据解析计算参数变量中
```

3.1.11. 读取一帧计算用实时数据

int MLX90640_GetFrameData(uint8_t slaveAddr, uint16_t *frameData)

此函数用于读取完整的一帧实时测量数据。计算所需要的完整的一帧数据为 834 个字（包括 832 个字 RAM 数据+控制寄存器+状态寄存器）。如果返回-1 表示 MLX90640 未应答，-8 表示读取异常（最可能的情况是读取速率太低了），若返 回 0 或者 1 则表示读取到了刚刚测量完成的子页 0 或者子页 1（读取成功），此时参数 frameData 数 组内即是 834 个字的实时数据。

例如：

```
static uint16_t mlx90640Frame[834];
int status;
status = MLX90640_GetFrameData (0x33, mlx90640Frame);
```

3.1.12. 计算 Vdd

float MLX90640_GetVdd(uint16_t *frameData, const paramsMLX90640 *params)

此函数用于计算并返回 Vdd 电压值。需要的参数说明如下：

uint16_t *frameData：读取到的完整的一帧实时数据（3.1.11 中得到的 834 个字的数组）

const paramsMLX90640 *params：由 EEPROM 解析得到的自定义结构体变量（8.1.10 得到的“计算参数”）此函数的返回值是浮点数，单位为 V。若不是 3.3V 左右，则说明发生了较为严重的问题。Vdd 在计算点阵温度时用做参考电压，故此计算结果必须为实际电压，否则无法得到正确的温度值。（Vdd 的值在 3.1.14 MLX90640_CalculateTo 函数体中会用到）

例如：

为了得到 Vdd 电压值，需要如下步骤：

```
float vdd;
uint8_t slaveAddress;
static int eeMLX90640[832];
static int mlx90640Frame[834];
paramsMLX90640 mlx90640;
int status;
```

```
status = MLX90640_DumpEE (slaveAddress, eeMLX90640);
//读取 EEPROM
status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640); //由 EEPROM
计算得到计
算参数
status = MLX90640_GetFrameData (0x33, mlx90640Frame);
//读取一帧实时数据
vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
//由计算参数和
//实时数据计算得到 Vdd
```

3.1.13. 计算 Ta

```
float MLX90640_GetTa(uint16_t *frameData,

const paramsMLX90640 *params)
```

此函数计算得到 Ta（MLX90640 外壳温度）。
所需要的参数与 3.1.12 相同。
此函数的返回值是浮点数，单位为℃。若与环境温度相差甚远，则说明发生了较为严重的问题。
例如：

为了得到 Ta 温度值，需要如下步骤：

```
float Ta;
uint8_t slaveAddress;
static int eeMLX90640[832];
static int mlx90640Frame[834];
paramsMLX90640 mlx90640;
int status;
status = MLX90640_DumpEE (slaveAddress, eeMLX90640);
//读取 EEPROM
status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640); //由 EEPROM
计算
//得到计算参数
status = MLX90640_GetFrameData (0x33, mlx90640Frame);
//读取一帧实时数据
Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640); //由计算参数和实时数据计
算得到 Ta
```

3.1.14. 计算物体绝对温度数据

```
void MLX90640_CalculateTo(uint16_t *frameData,
```

```
const paramsMLX90640 *params,
float emissivity, float tr,
float *result)
```

此函数计算一帧温度值（768 像素点）。计算结果为浮点型，保存于 result 数组内。参数说明如下：

- **uint16_t *frameData**：读取到的一帧实时数据
- **const paramsMLX90640 *params**：由 EEPROM 计算得到的计算参数
- **float emissivity**：被测物体的辐射率（人体为 0.95，其它材料的辐射率请上网查）
- **float tr**：校正温度，一般取 Ta-8
- **float *result**：计算结果，768 个浮点数

例如：

计算一帧刚刚读取到的实时数据为温度值，假设被测物体的辐射率为 0.95。

```
float emissivity = 0.95;
float Ta,tr;
unsigned char slaveAddress;
static uint16_t eeMLX90640[832];
static uint16_t mlx90640Frame[834];
paramsMLX90640 mlx90640;
static float mlx90640To[768];
int status;
status = MLX90640_DumpEE (slaveAddress, eeMLX90640);
status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
status = MLX90640_GetFrameData (0x33, mlx90640Frame);
Ta= MLX90640_GetTa(mlx90640Frame, &mlx90640);
tr = Ta-8.0;
MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr, mlx90640To);
```

计算完成后，mlx90640To 数组中保存的即是 768 个单位为℃温度值。

3.1.15. 计算热图像数据

```
void MLX90640_GetImage(uint16_t *frameData,
const paramsMLX90640 *params,
float *result)
```

此函数的功能与上一节计算温度几乎完全相同，不同点仅为计算结果中的数值没有规划为温度单位，而是一些仅有数值大小意义的数值，用这些数值大小来绘图是足够的，这个函数的优点就是速度要比计算温度要快很多（仅需要绘图而不关心绝对温度值时可以使用这个函数来计算完成）。

例如：

```
static uint16_t eeMLX90640[832];
static uint16_t mlx90640Frame[834];
paramsMLX90640 mlx90640;
static float mlx90640Image[768];
int status;
status = MLX90640_DumpEE (0x33, eeMLX90640);
status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
status = MLX90640_GetFrameData (0x33, mlx90640Frame);
MLX90640_GetImage(mlx90640Frame, &mlx90640, mlx90640Image);
```