MLX90640 Thermal Camera

用 户 手 册

产品特点

本模块一款红外热像仪模块，32×24 像素，I2C 接口通信，兼容 3.3V/5V 电平，支持 Raspberry Pi、Arduino 等主控。我采用 MLX90640 远红外热传感器阵列，可精确检测特定区域和温度范围内的目标物体，尺寸小巧，可方便集成到各种工业或智能控制应用中。

- 采用 MLX90640 远红外热传感器阵列，32×24 像素
- 支持 I2C 接口通信，可设置为快速模式(速率可达 1MHz)
- 噪声等效温差(NETD)仅为 0.1K RMS@1Hz 刷新率，噪声性能好
- 板载电平转换电路，可兼容 3.3V/5V 的工作电平

[照片]

参数：

工作电压:3.3V/5V
工作电流:＜23mA
通信接口:I2C (地址为 0x33)
视场角(水平视角×垂直视角):

- MLX90640-D55 Thermal Camera：55° ×35° （角度小，适合远距离测量）
- MLX90640-D110 Thermal Camera：110° ×75° （角度大，适合近距离测量）

工作温度:-40℃～85℃
目标温度:-40℃～300℃
检测精度:±1℃
刷新速率:0.5Hz~64Hz (可编程设置)
产品尺寸:28mm×16 mm
固定孔尺寸:2.0mm

主要用途：

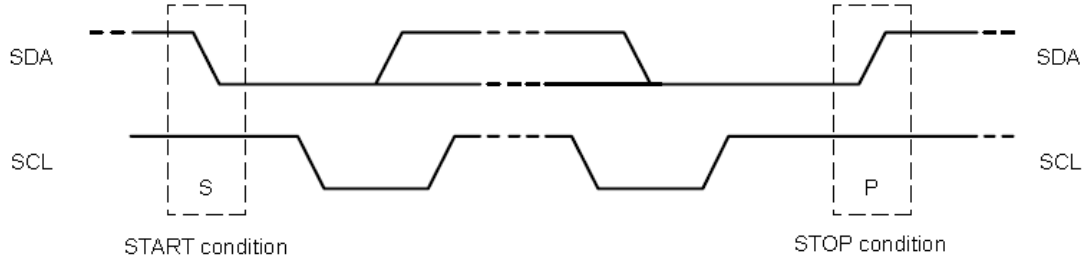
- 高精度非接触性物体温度检测
- 红外热像仪、红外测温仪
- 智能家居、智能楼宇、智能照明
- 工业温度控制、安防、入侵/移动检测

接口说明(以接入 MCU 为例)：

VCC：接 3.3V
GND：接 GND
SDA：接 MCU.I2C 数据线
SCL：接 MCU.I2C 时钟线

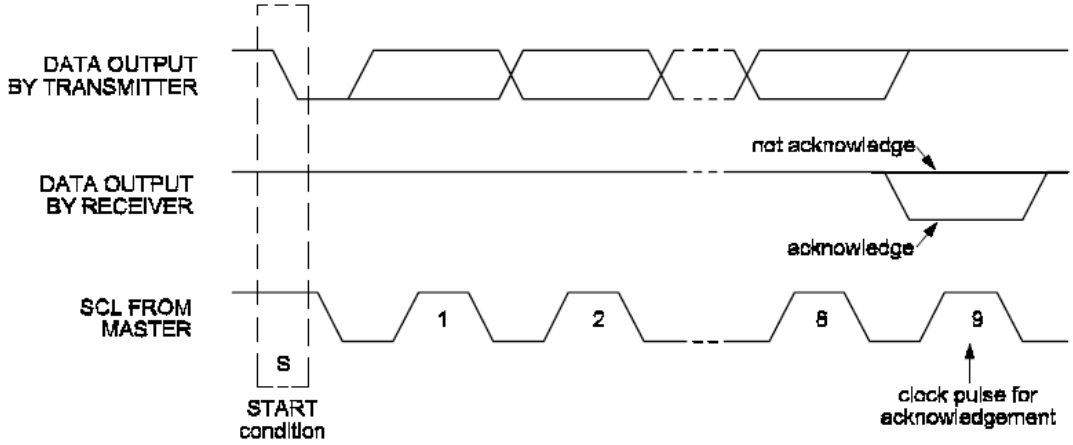
通讯方式：

本模块的通讯方式为 I2C，支持 I2C 高速模式(最高可达 1MHz)，只能作为 I2C 总线上的从设备，SDA 和 SCL 端口可以承受 5V 电压，可以直接接入到 5V I2C 总线中，模块的设备地址是可以编程的，最多可以有 127 个地址，出场默认值为 0x33。
与一般 I2C 总线一样，在传送数据过程中共有三种类型信号：开始信号、结束信号和应答信号。



起始和停止条件

开始信号：SCL 为高电平，SDA 由高电平转换为低电平。
结束信号：SCL 为高电平，SDA 由低电平转换为高电平。
可以看出开始信号和结束信号都是在 SCL 总线为高电平时刻完成的。



I^2C 总线的响应

应答信号：在每个字节传输之后的第 9 个时钟期间内，发送数据端设备释放 SDA 总线，接受数据端设备拉低 SDA 总线表示收到字节(ACK)，或者是 SDA 总线为高电平不应答(NoACK)。
设备地址：
主机通过在 START 条件后发送 7 位从机地址来寻址从机。 前七个位是该地址专用，第 8 个是读/写（R / W）位。 该位指示传输方向：
Read（1）表示主机将从从机读取数据
Write（0）表示主机将向从机发送数据

通信格式：

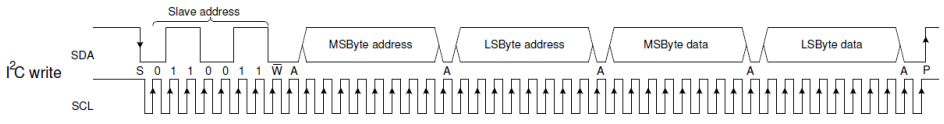


Figure 4 I^2C write command format (default SA=0x33 is used)

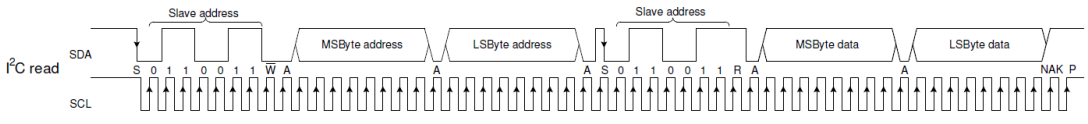
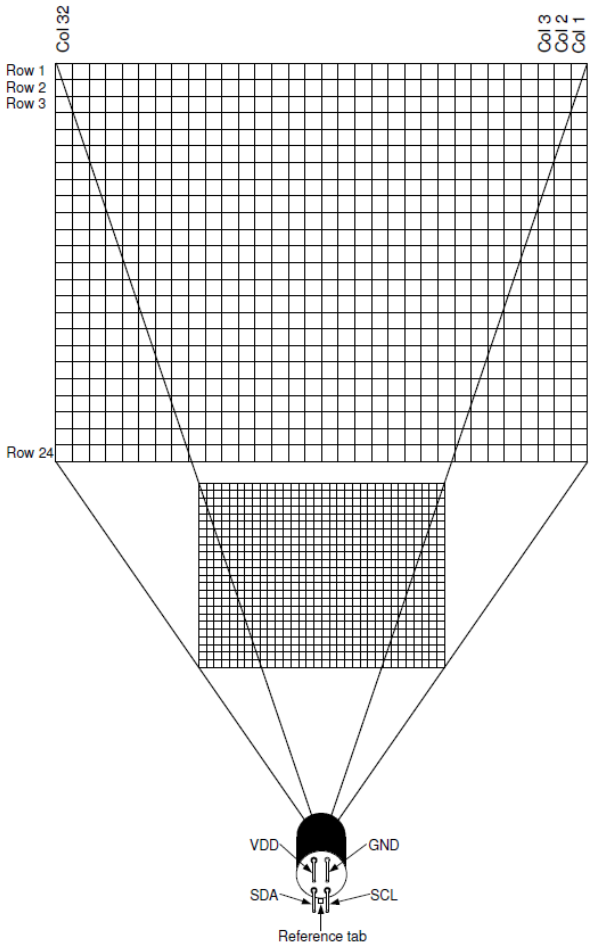


Figure 5 I^2C read command format (default SA=0x33 is used)

本模块共具有 768 个 IR 传感器（也称为像素）组成。 每个像素的行和列位置标识为 Pix（i，j），其中 i 是其行号（从 1 到 24），j 是其列号（从 1 到 32），像素具体到某一平面可以参照下图。



*需要说明的是传感器原厂在传感器出厂时允许有 4 个以内的坏点，每个坏点都在 EEPROM 表中有标识，所以模块可能会有一定几率存在坏点，也就是说这不能作为退换货的依据，对此原厂的建议是使用相邻像素的平均值代替。

内存地址分布：

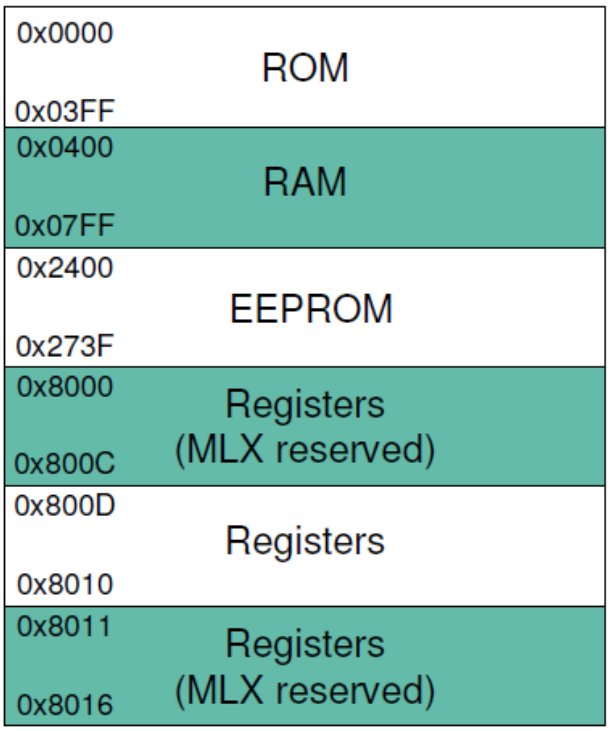


Figure 10 MXL90640 memory map

两种 RAM 数据模式：

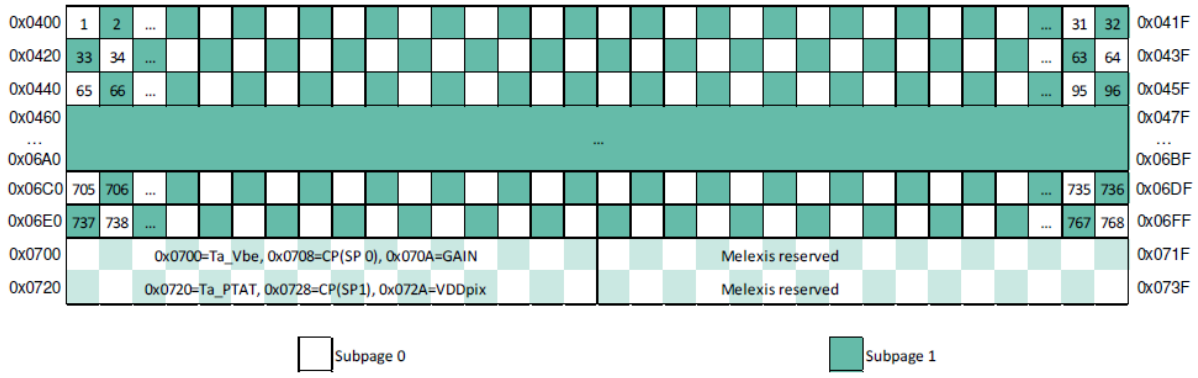


Figure 14 RAM memory map (Chess pattern mode) – factory default mode

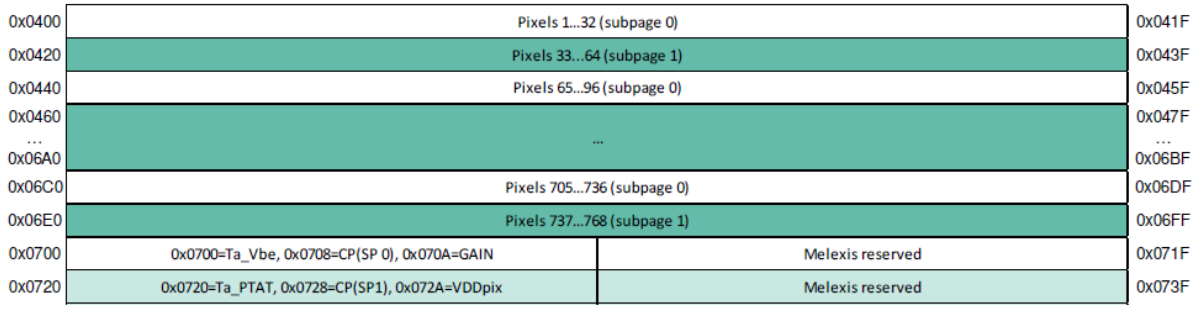


Figure 15 RAM memory map (Interleaved mode)

EEPROM 用于存储校准常数和设备的配置参数：

EEPROM address	Access	Meaning
0x2400	Melexis	Melexis reserved
0x2401	Melexis	Melexis reserved
0x2402	Melexis	Melexis reserved
0x2403	Melexis	Configuration register
0x2404	Melexis	Melexis reserved
0x2405	Melexis	Melexis reserved
0x2406	Melexis	Melexis reserved
0x2407	Melexis	Device ID1
0x2408	Melexis	Device ID2
0x2409	Melexis	Device ID3
0x240A	Melexis	Device Options
0x240B	Melexis	Melexis reserved
0x240C	Customer	Control register_1
0x240D	Customer	Control register_2
0x240E	Customer	I2CConfReg
0x240F	Customer	Melexis reserved / I2C_Address

Table 7 Configuration parameters memory

刷新率：

本模块共支持 8 种刷新率，最高可达 64Hz，刷新率由控制寄存器 1-0x800D 控制，如下图：

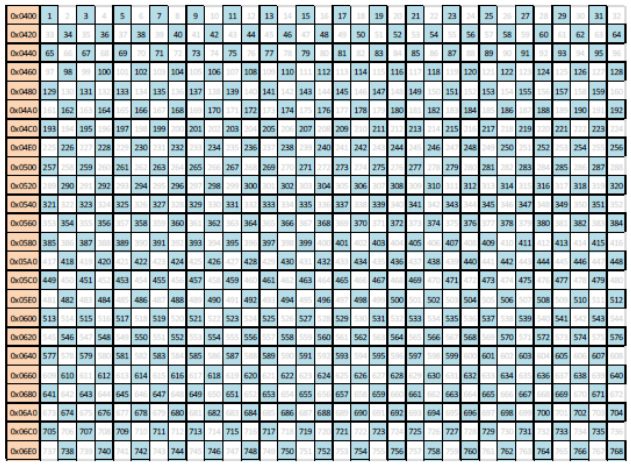
B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	Control register 1 - 0x800D
Melexis reserved			Reading pattern	Resolution control		Refresh rate control			Select subpage			Enable subpages repeat	Enable data hold	Melexis reserved	Enable subpages mode	
															0	No subpages, only one page will be measured
															1	Subpade mode is activated (default)
														0		Keep this bit = "0" (default)
													0			Transfer the data into storage RAM at each measured frame (default)
													1			Transfer the data into storage RAM only if en_overwrite = 1 (check 0x8000)
												0				Toggles between subpage "0" and subpage "1" if Enable subpages mode = "1" (default)
												1				Select subpage determines which subpage to be measured if Enable subpages mode = "1"
									0	0	0					Subpage 0 is selected (default)
									0	0	1					Subpage 1 is selected
									0	1	0					Not Applicable
									0	1	1					Not Applicable
									1	0	0					Not Applicable
									1	0	1					Not Applicable
									1	1	0					Not Applicable
									1	1	1					Not Applicable
						0	0	0								IR refresh rate = 0.5Hz
						0	0	1								IR refresh rate = 1Hz
						0	1	0								IR refresh rate = 2Hz (default)
						0	1	1								IR refresh rate = 4Hz
						1	0	0								IR refresh rate = 8Hz
						1	0	1								IR refresh rate = 16Hz
						1	1	0								IR refresh rate = 32Hz
						1	1	1								IR refresh rate = 64Hz
				0	0											ADC set to 16 bit resolution
				0	1											ADC set to 17 bit resolution
				1	0											ADC set to 18 bit resolution (default)
				1	1											ADC set to 19 bit resolution
			0													Interleaved (TV) mode
			1													Chess pattern (default)
-	-	-	Melexis reserved													

8 种刷新率的设置是取决于控制寄存器 1-0x800D 的位 7、位 8、位 9。

像素排列模式：

国际象棋模式（出厂默认设置）

Subpage 0 --> 0x8000 = 0xXXX8



Subpage 1 --> 0x8000 = 0xXXX9

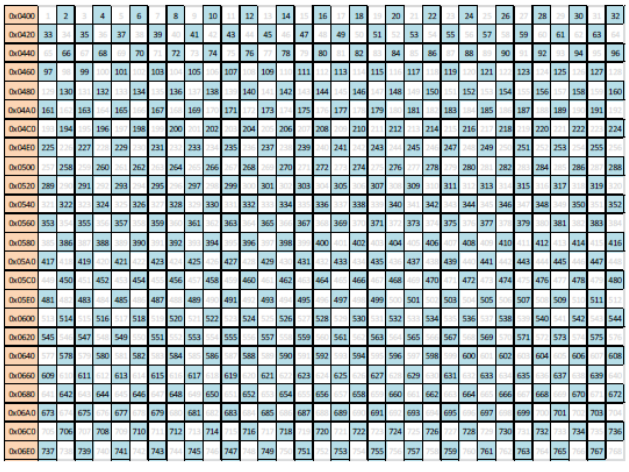


Figure 9 Chess reading pattern (only highlighted cells are updated)

电视交错模式

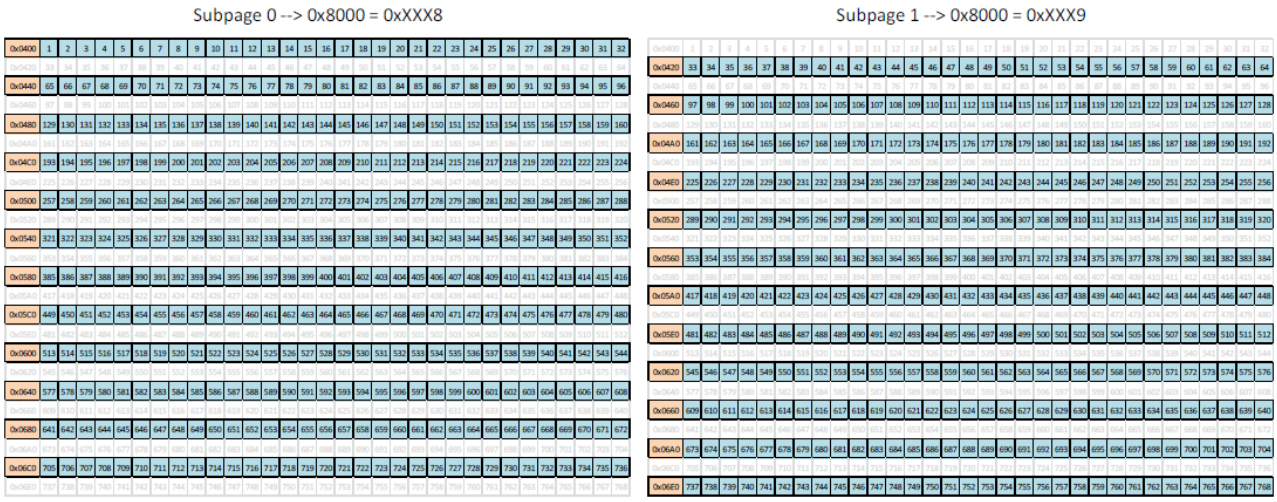


Figure 8 TV mode reading pattern (only highlighted cells are updated)

两种模式在子页面的更新方式上不同，这里需要注意的是传感器仅在国际象棋模式下进行过出厂校准，因此在国际象棋模式下可以获得更好的固定图案噪声行为，因此为了获得最佳效果建议使用国际象棋棋盘模式，两种模式的设定取决于控制寄存器 1-0x800D 的位 12。

测量原理：

对于非接触式红外测温模块，很重要的一个概念是“视场 (FOV)”。 视场是由温差电堆接收到 50%的辐射信号来确定的，并且和传感器的主轴线相关。测得的温度是视场内被测物体的温度加权平均值，所以当被测物体完全覆盖 FOV 视场时的准确度是最高的。

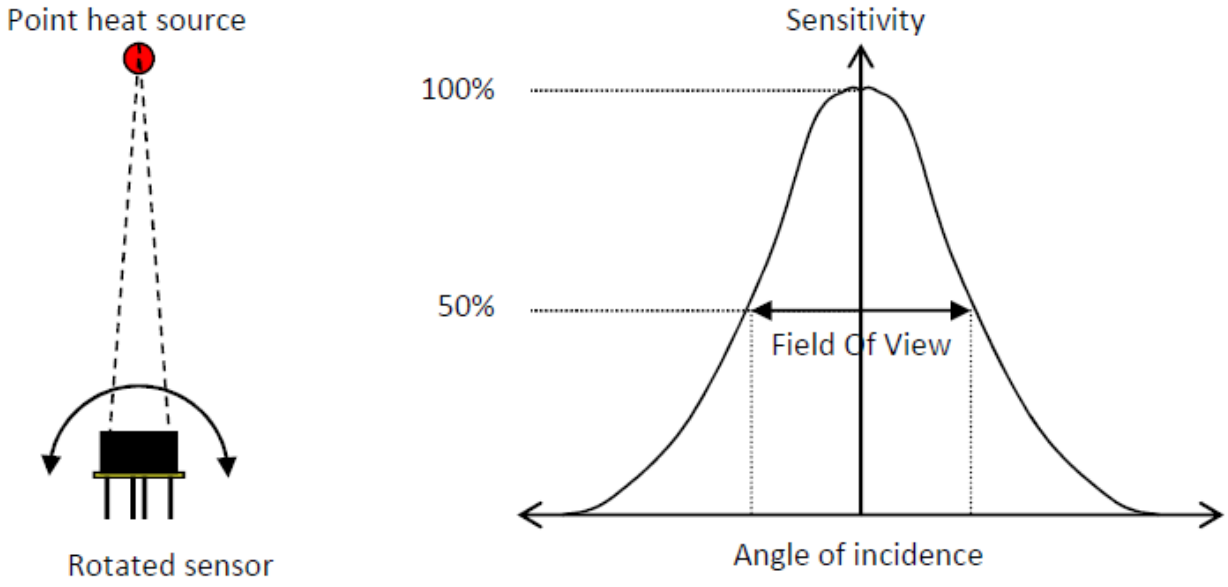


Figure 24: Field Of View measurement

其中被测物体的面积与模块测量距离满足以下关系：

$$S : D = 2 \tan\left(\frac{FOV}{2}\right)$$

测量流程：

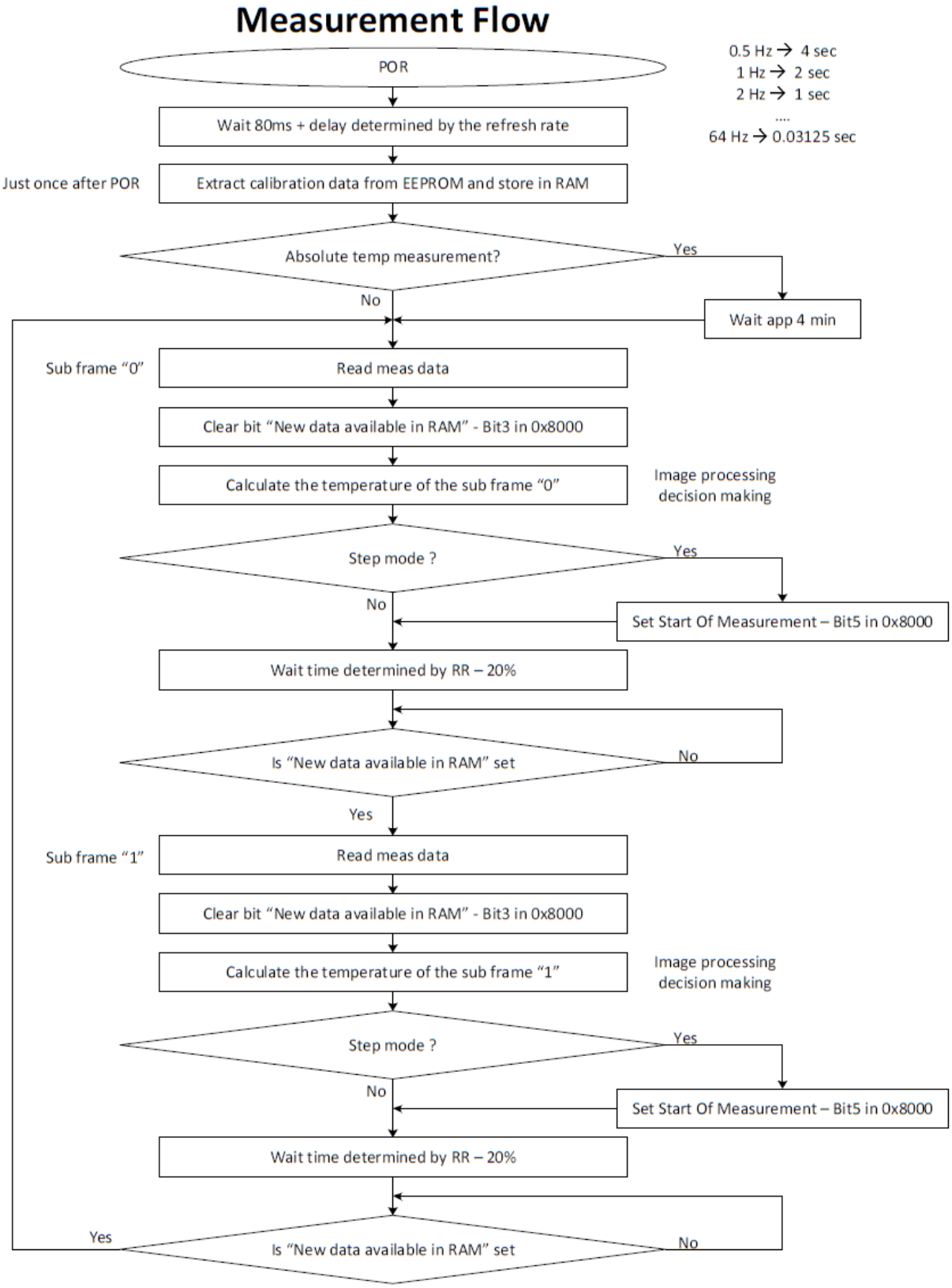


Figure 7 Recommended measurement flow

可以看出整个测量流程还是比较麻烦的，其中还包含了很多数学计算，比如计算供电电压：

11.2.2.2. Supply voltage value calculation (common for all pixels)

$$V_{dd} = \frac{Resolution_{corr} * RAM[0x072A] - V_{dd_{25}}}{K_{Vdd}} + V_{dd_0}$$

Where: Constants calculation of the EEPROM stored values (can be done just once after POR)

$$K_{Vdd} = \frac{EE[0x2433] \,\&\, 0xFF00}{2^8} = \frac{0x9D68 \,\&\, 0xFF00}{2^8} = 0x009D = 157$$

If $157 > 127 \rightarrow K_{Vdd} = 157 - 256 = -99$

$$K_{Vdd} = K_{Vdd} * 2^5 = -99 * 32 = -3168$$

$$Vdd_{25} = EE[0x2433] \,\&\, 0x00FF = 0x9D68 \,\&\, 0x00FF = 0x0068 = 104$$

$$Vdd_{25} = (Vdd_{25} - 256) * 2^5 - 2^{13} = -152 * 32 - 8192 = -13056$$

VDD calculations:

$$RAM[0x072A] = 0xCCC5 = 52421$$

If $52883 > 32767 \rightarrow RAM[0x072A] = 52421 - 65536 = -13115$ LSB

$$V_{dd} = \frac{1 * -13115 - (-13056)}{-3168} + 3.3 = \frac{-59}{-3168} + 3.3 \approx 0.0186 + 3.3 \approx 3.319V$$

从上图可以看到公式还是比较繁琐的，如果用代码写出来有可能还会出错，当然这仅仅是一个参数的计算，还有很多参数都是需要我们计算的，不过不用担心，迈来芯的工程师已经为我们写好驱动了，我们只需要移植一下驱动就可以很简单方便的计算出每一个像素点的温度值了，驱动可以从下面链接上获取到最新版本：

https://github.com/melexis/mlx90640-library

代码分析：

下面我们通过 STM32 移植一下驱动代码去实际分析一下工作流程，代码以 STM32F405RGT6 最小系统板为例：

实验现象

1） 用杜邦线将传感器模块按照以下接线方式连接起来：

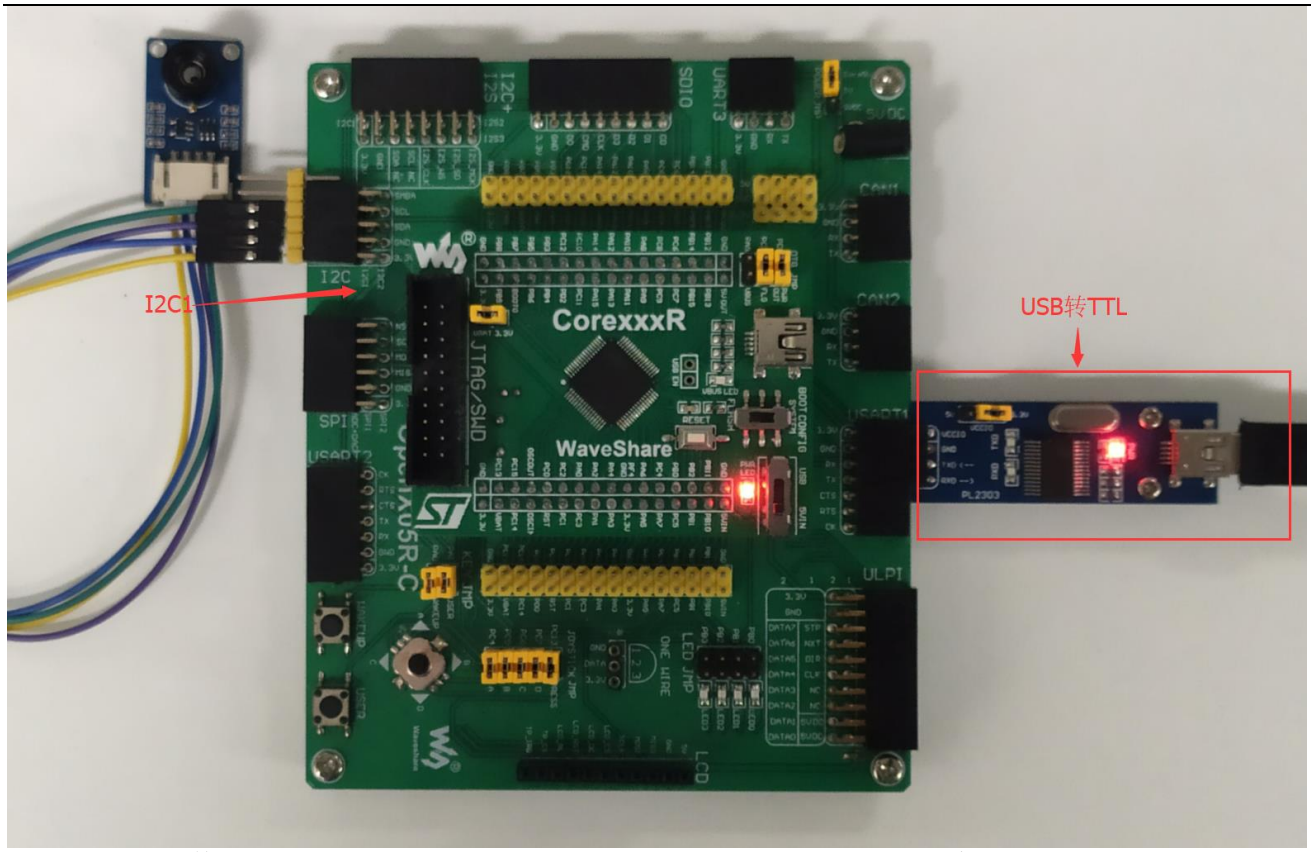
STM32F405RGT6 最小系统板<------>MLX90640 Thermal Camera

PB10	SCL
PB11	SDA
3V3	VCC
GND	GND

STM32F405RGT6 最小系统板<------>USB 转 TTL 串口助手

PA9	RX
GND	GND

如图：

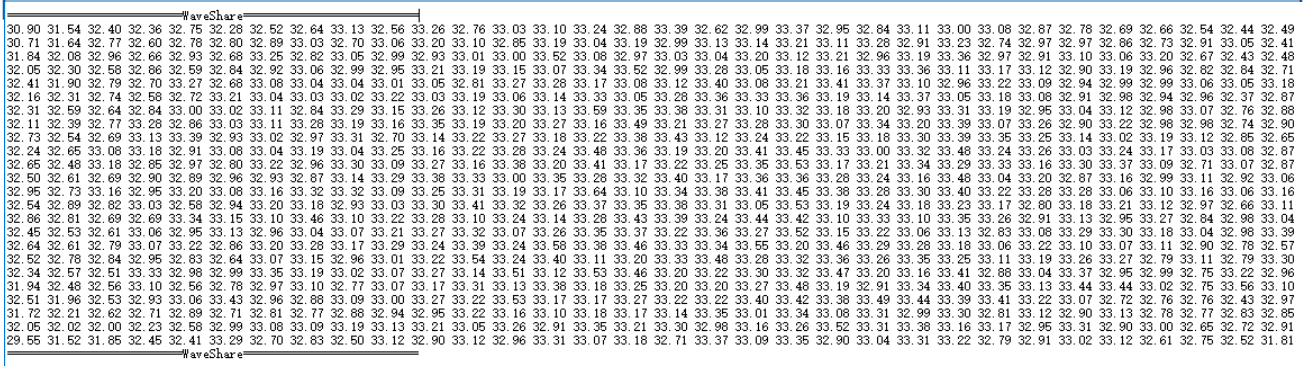


2) 用 keil 软件打开程序 .\MDK-ARM\F405_HAL_HWI2C_MLX90640.uvprojx，编译、下载。

3) 打开串口监视软件，选择正确的串口号，并设置如下：波特率：115200；数据位：8；停止位：1；校验位：None；控制流：None。

预期结果：

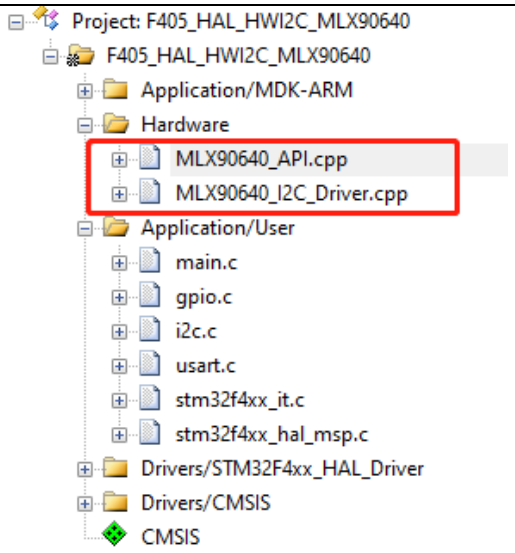
将手掌正对着传感器镜头，距离尽量近一点，可以看到串口打印的温度值大多数都接近人的体温：



接下来我们分析一下迈来芯给的驱动是如何移植到我们的工程中去的，以及大体工作流程是怎样的。

首先我们先看一下驱动程序都有哪些文件，这里只看一下 C 文件，头文件可以自己去看一下，如下图：

其中 MLX90640_API.cpp 主要就是前面我们讲的主要负责计算的部分，这一部分也是我们唯一不需要改动的地方，我们需要改动的地方只有其余的两个文件，这里我们使用的是 STM32 的硬件 I2C，所以第三个文件我们不需要，所以可以不用加载到工程中去，最后我们的工程树如下图就可以：



添加好以后我们还需要适配 I2C 驱动，这样才能使得 MLX90640_API.cpp 文件可以正确的通过 MLX90640_I2C_Driver.cpp 文件提供的方法读取传感器的值，如下图，将 MLX90640_I2CRead()方法和 MLX90640_I2CWrite()关联 STM32 的 I2C 读写函数即可：

读函数

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

  uint8_t* bp = (uint8_t*) data;

    int ack = 0;
    int cnt = 0;

    ack = HAL_I2C_Mem_Read(&hi2c2, (slaveAddr<<1), startAddress, I2C_MEMADD_SIZE_16BIT, bp, nMemAddressRead*2, 500);

    if (ack != HAL_OK)
    {
        return -1;
    }

    for(cnt=0; cnt < nMemAddressRead*2; cnt+=2) {
      uint8_t tmpbytelsb = bp[cnt+1];
      bp[cnt+1] = bp[cnt];
      bp[cnt] = tmpbytelsb;
    }


    return 0;
}
```

写函数

```
int MLX90640_I2CWrite(uint8_t slaveAddr, uint16_t writeAddress, uint16_t data)
{

    uint8_t sa;
    int ack = 0;
    uint8_t cmd[2];
    static uint16_t dataCheck;

    sa = (slaveAddr << 1);

    cmd[0] = data >> 8;
    cmd[1] = data & 0x00FF;

    ack = HAL_I2C_Mem_Write(&hi2c2, sa, writeAddress, I2C_MEMADD_SIZE_16BIT, cmd, sizeof(cmd), 500);

    if (ack != HAL_OK)
    {
        return -1;
    }

    MLX90640_I2CRead(slaveAddr,writeAddress,1, &dataCheck);

    if ( dataCheck != data)
    {
        return -2;
    }

    return 0;
}
```

接下来我们就可以通过 API 来进行数据读取了，按照前面讲的测量流程，上电复位后我们首先要读取 EEPROM 中的参数：

```
paramsMLX90640 mlx90640;
status = MLX90640_DumpEE(MLX90640_ADDR, eeMLX90640);
if (status != 0) printf("\r\nload system parameters error with code:%d\r\n",status);
status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
if (status != 0) printf("\r\nParameter extraction failed with error code:%d\r\n",status);
```

然后我们可以设置一下刷新速率：

```
MLX90640_SetRefreshRate(MLX90640_ADDR, RefreshRate);
```

接下来我们就可以读取传感器的温度值了，刚开始可能会有一点不准，一般 30S 内数据就会稳定下来：

```
    for(int i = 0; i < 2; i ++){
      int status = MLX90640_GetFrameData(MLX90640_ADDR, frame);
      if (status < 0)
      {
        printf("GetFrame Error: %d\r\n",status);
      }
      float vdd = MLX90640_GetVdd(frame, &mlx90640);
      float Ta = MLX90640_GetTa(frame, &mlx90640);

      float tr = Ta - TA_SHIFT; //Reflected temperature based on the sensor ambient temperature
      MLX90640_CalculateTo(frame, &mlx90640, emissivity , tr, mlx90640To);
    }
```

MLX90640_GetFrameData()是获取一帧数据，此时还不是我们直观上的温度值，还需要通过计算来获得真正的温度值，计算的时候有几个参数是需要我们注意，首先是发射率(emissivity)，这是物体的一种属性，根据材料的不同而不同，一般非金属（如塑料、油漆、皮革、纸张等）发射率通常为 0.95，常见物体发射率可以参考下图：

材料名称	规格	发射率	材料名称	规格	发射率
铝	氧化	0.20-0.40	人体皮肤		0.98
	抛光	0.02-0.04	石墨	氧化	0.20-0.60
铜	氧化	0.40-0.80	塑胶	透明度>0.5mm	0.95
	抛光	0.02-0.05			
黄金		0.01-0.10	橡胶		0.95
铁	氧化	0.60-0.09	塑胶		0.85-0.95
钢	氧化	0.70-0.90	混凝土		0.95
石棉		0.95	水泥		0.96
石膏		0.80-0.90	土壤		0.90-0.98
沥青		0.95	灰泥		0.89-0.91
陶器		0.95	砖		0.93-0.96
木材		0.90-0.95	大理石		0.94
木炭	粉末	0.96	纺织品	各种	0.90
漆器		0.80-0.95	纸	颜色	0.94
漆器	无光泽	0.97			
碳胶		0.90	沙子		0.90
肥皂泡		0.75-0.80	泥土		0.92-0.96
水		0.93	沙砾	餐具	0.95
雪		0.83-0.90	玻璃	规格	0.85-0.92
冰		0.96-0.98	纺织品		0.95

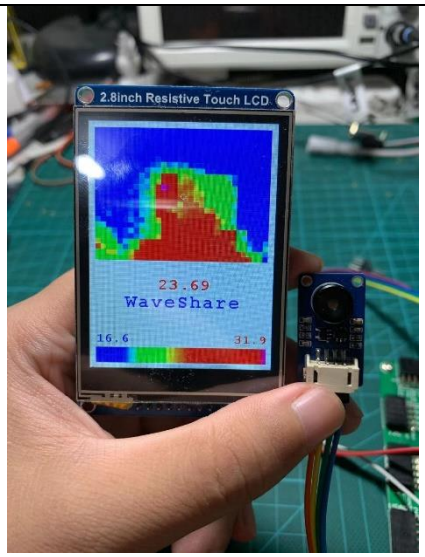
另外一个就是 tr(反射温度基于传感器环境温度)的值这个参数是通过 Ta - TA_SHIFT 计算出来的，其中 TA_SHIFT 默认值为 8，MLX90640_CalculateTo()就是将获得的一帧数据与这几个值进行运算从而得到我们所理解的温度值。计算完成后我们就可以通过串口打印查看具体的温度值了，如下图，可以将 768 个像素的温度值全部打印出来，经过计算的温度值储存在 mlx90640To 数组中：

```
printf("\r\n=========================WaveShare=========================\r\n");
for(int i = 0; i < 768; i++){
  if(i%32 == 0 && i != 0){
    printf("\r\n");
  }
  printf("%2.2f ",mlx90640To[i]);
}
printf("\r\n=========================WaveShare=========================\r\n");
```

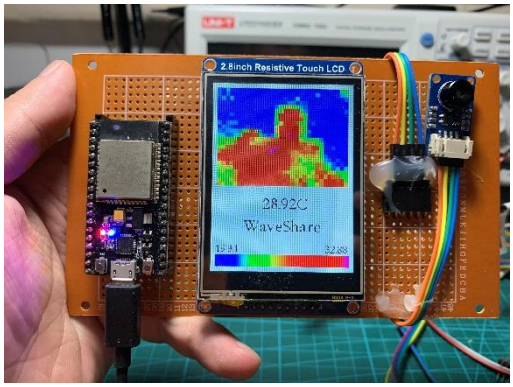
至此，我们就成功移植了官方的驱动到我们的工程中，更多例程请查看示例程序的其他部分。

比如：

① STM32+LCD 图像显示：



② Arduino+ESP32+LCD 图像显示：



③ 树莓派 HDMI 显示屏显示：