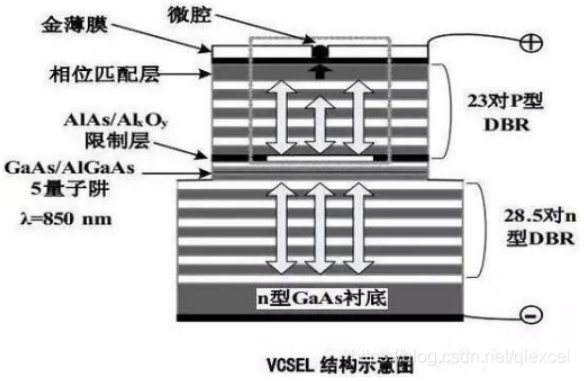
激光测距芯片VL53L0X的使用与代码

一、介绍

1、原理

采用940nm垂直腔面发射激光器（Vertical-Cavity Surface-Emitting Laser，简称VCSEL)发射出激光，激光碰到障碍物后反射 回来被VL53L0X接收到，测量激光在空气中的传播时间，进而得到距离。VCSEL相关知识



VCSEL 结构示意图

2、参数

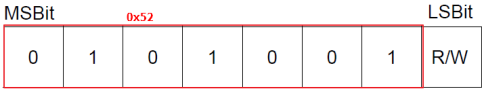
超小体积：4.4 x 2.4 x 1.0mm

最大测距：2m

发射的激光对眼镜安全，且完全不可见。

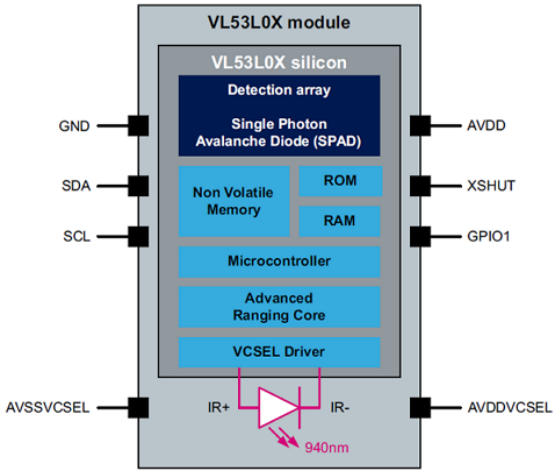
工作电压：2.6 to 3.5 V

通信方式 ：IIC，400KHz，设备地址0x52，最低位是读写标志位。0表示写，1表示读。

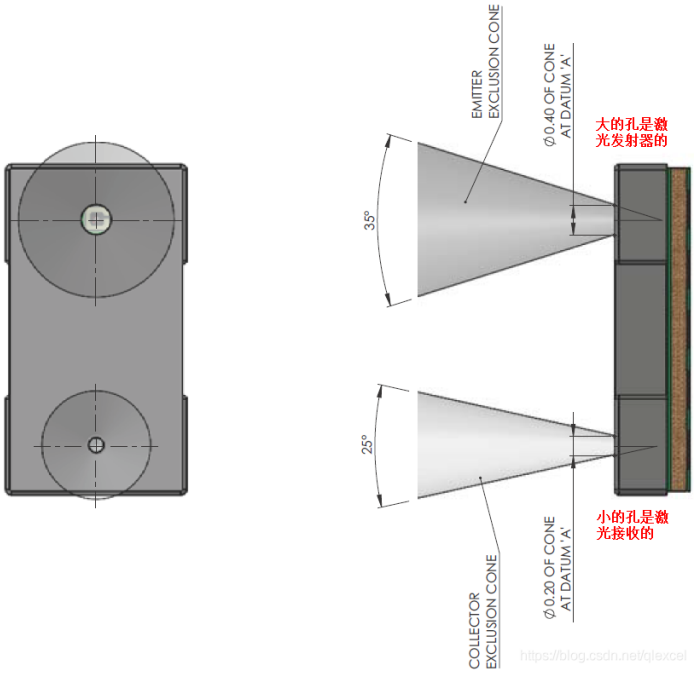


因此进行写的时候，此8位数据为：0101 0010，即0x52。进行读的时候，此8位数据为：0101 0011，即0x53。

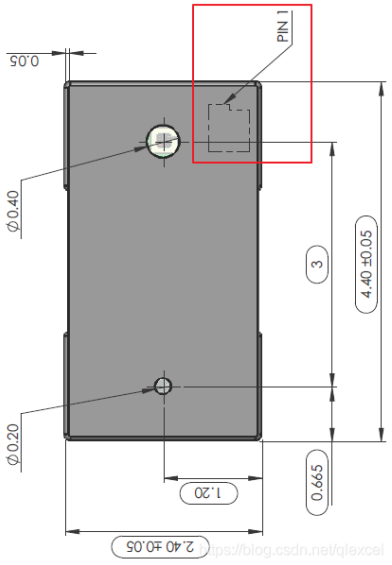
3、框图



VL53L0X上有两个孔，一个是VCSEL激光发射孔，一个是SPAD激光检测阵列的孔。

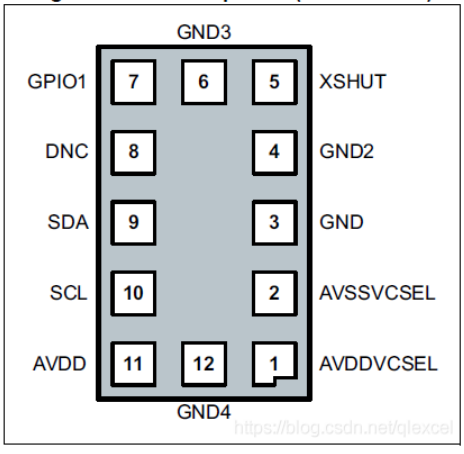


1脚在大孔那边。



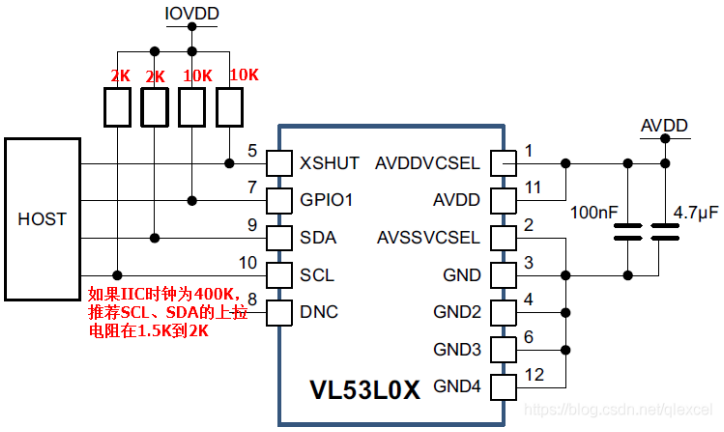
4、引脚图

Figure 2. VL53L0X pinout (bottom view)



1脚AVDDVCSEL：VCSEL电源正
2脚AVSSVCSEL：VCSEL电源地
3脚、4脚、6脚、12脚GND：地
5脚XSHUT：电源模式控制，如果不需要休眠功能，此脚可以直接接到AVDD上。
7脚GPIO1：中断输出。开漏输出，因此必须要外部上拉。
8脚DNC：悬空
9脚和10脚：IIC通信端口
11脚AVDD：电源正。

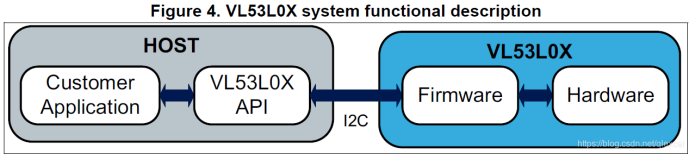
5、原理图



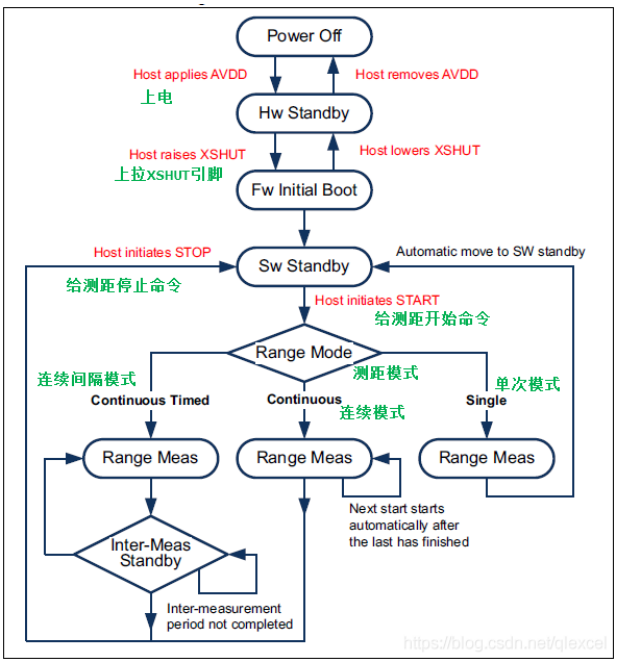
二、固件状态机

ST为VL53L0X专门配备了一套API，直接封装了各种功能比如：初始化/校准、测距开始/停止、精度选择、测距模式选择。用户的应用程序调用API中的函数，然后API通过IIC与VL53L0X中的固件（Firmware）进行通信，固件再操作硬件。

Figure 4. VL53L0X system functional description



VL53L0X中的固件是按照状态机来进行工作的：



当上电，VL53L0X会进入Hw Standby状态，这是待机状态，功耗很低。然后拉高XSHUT引脚，让VL53L0X进入Fw Boot状态，开始准备测距。如果不需要待机状态，可以把XSHUT接到AVDD上。VL53L0X处于休眠状态是不能进行IIC通信的。

VL53L0X有3种工作模式：

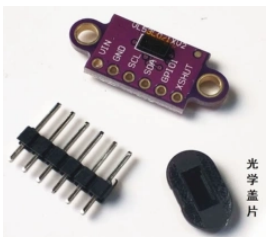
（1）单次模式：收到测距开始命令后，开始进行测量，测量完成后自动退出，进入Sw Standby状态。

（2）连续模式：收到测距开始命令后，就一直进行测量，直到收到测距停止命令。收到测距停止命令时，会把最后一次测量完成才退出。

（3）连续间隔模式：收到测距开始命令后，开始进行测量，完成一次测量后，等待一段时间再进行下次测量，直到收到测距停止命令。测量等待间隔时间可调。

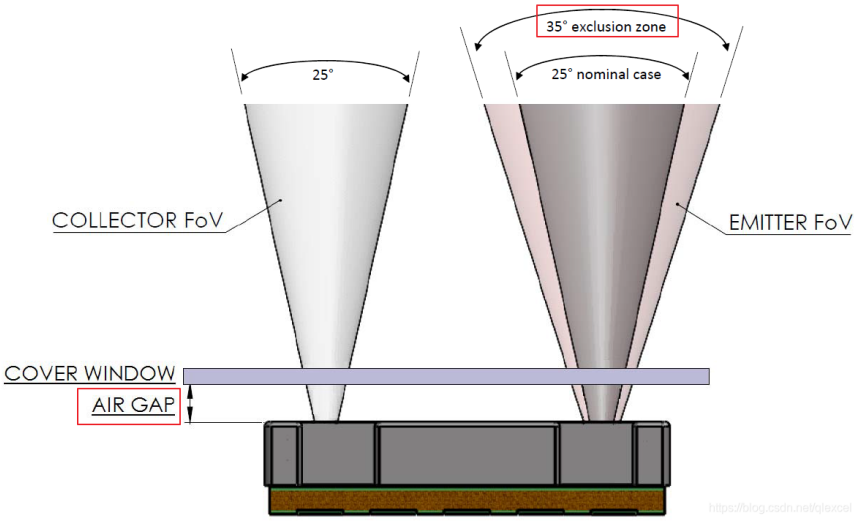
三、加盖玻片

VL53L0X一般会和盖玻片一起结合使用。盖玻片有两个作用：提供物理保护，防止灰尘；对光进行滤波。



盖玻片通常是不透明的，有两个圆孔或一个椭圆孔，以便发出和接收红外线。盖玻片必须满足一些光学要求，以保证测距能力。通过透射系数和雾度系数来测量盖玻片的质量。

有两个参数需要注意：VL53L0X和盖板窗口之间的气隙以及VL53L0X前面的扩展区（exclusion area），如下图：



当激光穿过盖玻片时，有部分会发生反射，我们应该尽量减少反射的光。嵌入的颗粒/孔洞和/或粗糙表面是盖玻片中光散射的主要因素。

理想的盖玻片有如下特点：

（1）塑料或玻璃材料无结构缺陷

（2）无可导致指纹光散射或污迹敏感的表面缺陷

（3）在近红外（940nm±10nm）和低雾度条件下，透过率>90%

（4）不降低指纹免疫性的外涂层（抗指纹或抗反射涂层）

（5）单一材料。使用双重材料可能会改变性能。

理想的结构设计（盖玻片盖在VL53L0X上的结构）有如下特点：

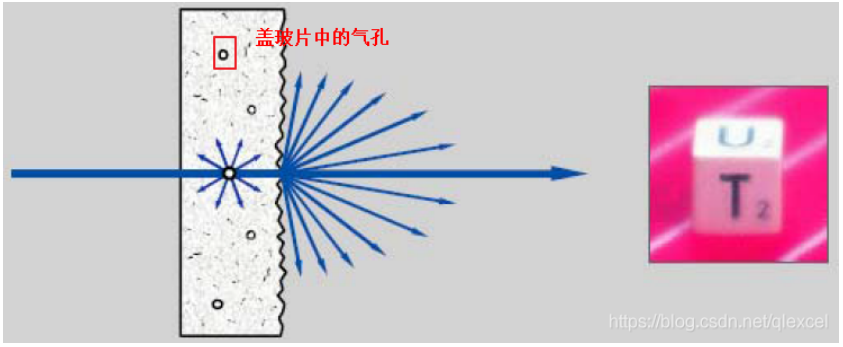
（1）气隙小（<0.5 mm）

（2）盖玻片薄

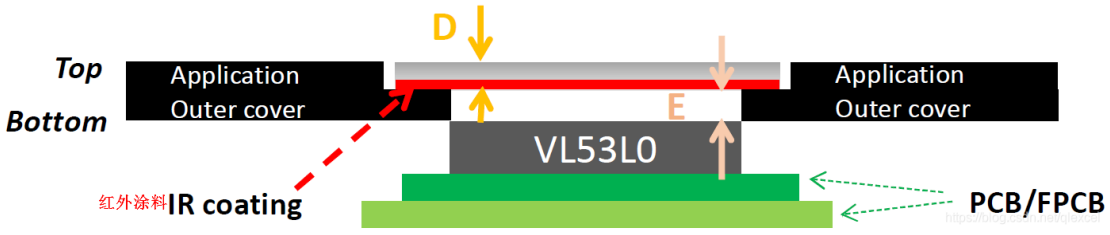
（3）盖玻片与VL53L0X表明的倾角低于2度

（4）严格的公差。

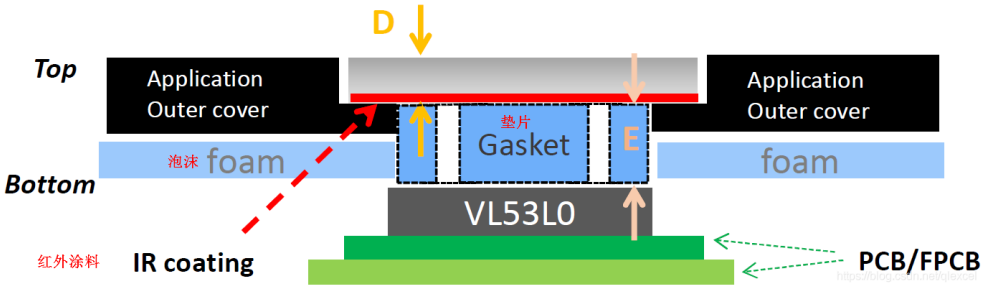
盖玻片的质量对激光传输的影响：



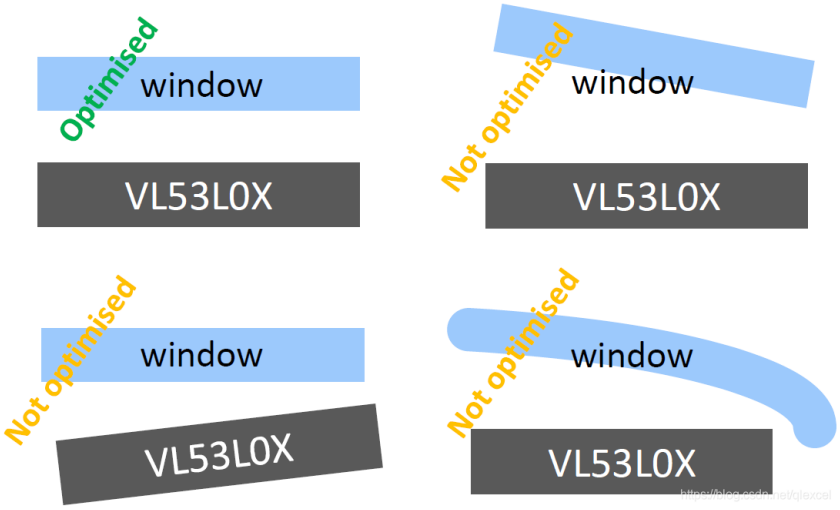
推荐的结构设计：尽量小的空气间隙（下图中的'E'），和薄的高透射系数的盖玻片（下图中的'D'）



如果空气间隙和盖玻片厚度已经不能再减小，可以在间隙中加入垫片，垫片可以帮助减少串扰。（串扰后面会讲）

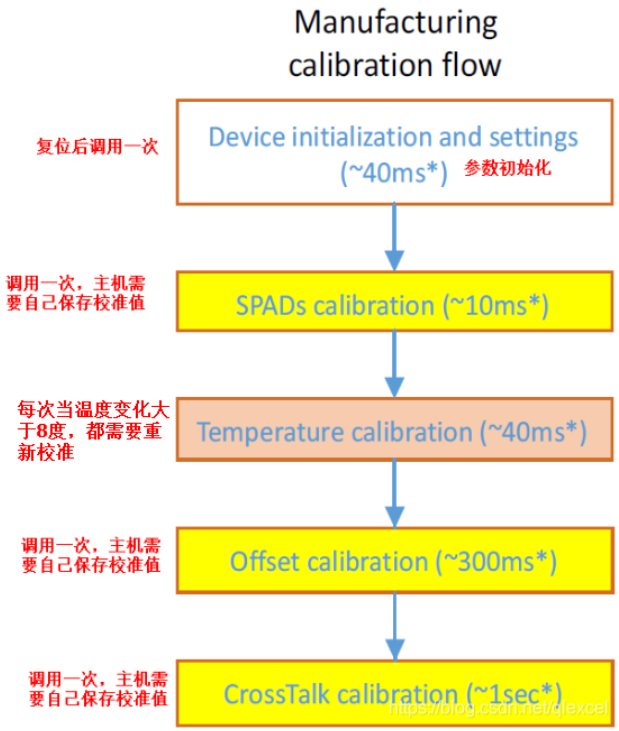


盖玻片还必须要和VL53L0X平行



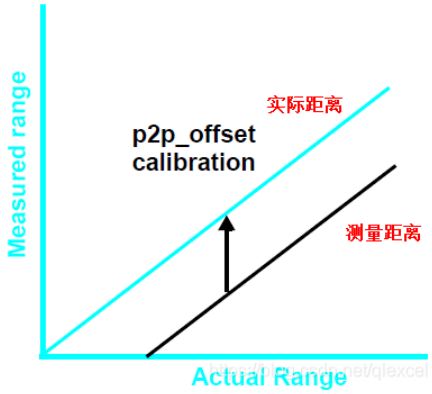
四、校准流程

为保证精度，用户确定好自己的使用环境（是否要盖玻璃盖、使用环境温度、供电电压等）后，要进行一次校准。流程如下：



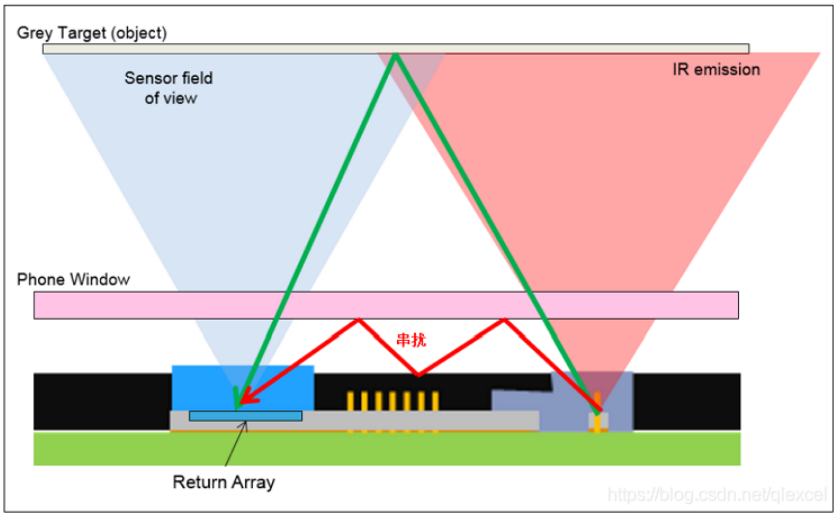
1、温度校准是确定两个与温度相关的参数：VHV 和 phase cal 。当每次VL53L0X的使用环境与校准环境之间温差大于8度时，都需要重新校准。

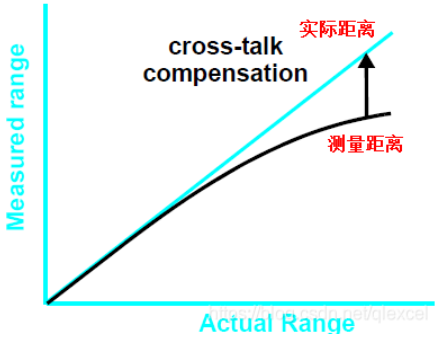
2、Offset校准是校准时间距离与测量距离之间的偏移量，一般推荐以10cm来进行校准。偏移量一般都是一个固定值，当确定好供电电压、环境温度、是否加玻璃盖等之后，读出测量值与实际值做差即可得到偏移量。



3、CrossTalk校准：CrossTalk即串扰，它被定义为从盖玻片弹回来的信号。如果加了玻璃盖，当激光射出玻璃盖的时候，一部分激光会被反射回来，成为干扰信号。干扰信号的大小取决于盖玻片类型和气隙大小。干扰信号产生的距离误差大小正比于串扰大小与目标返回信号大小的比值。

Figure 7. VL53L0X crosstalk





五、测距配置（profiles）

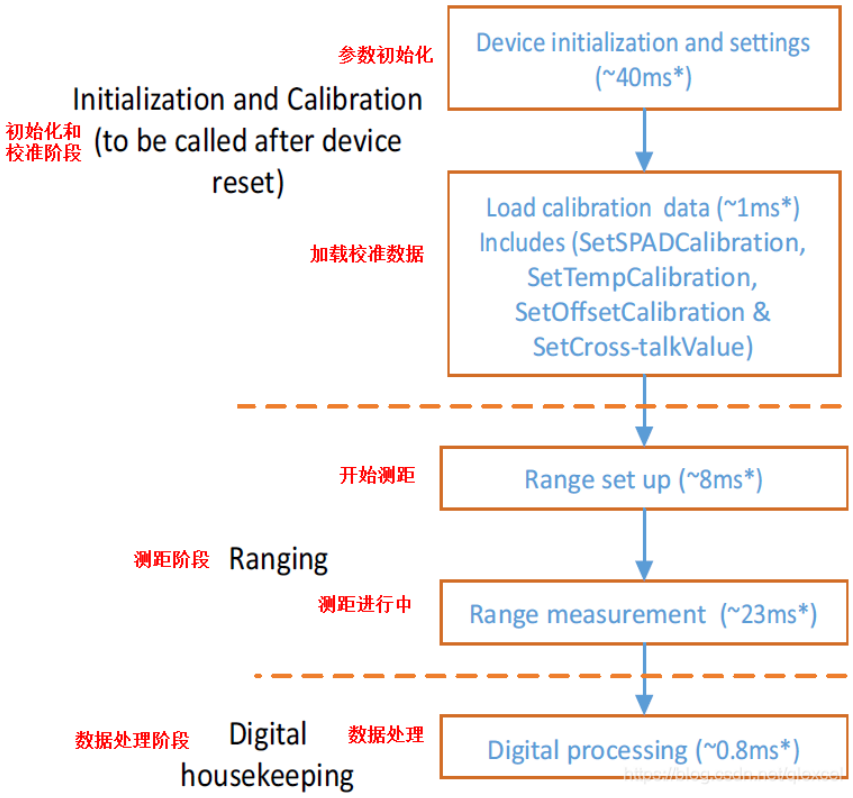
根据用户的应用，可以把VL53L0X配置为4种测距配置（ranging profiles）：默认配置、高速测距配置、高精度测距配置、长距离测距配置。VL53L0X以何种配置工作，并不是某一个寄存器决定的，而是以初始化的时候写入的众多初始参数决定的，因此把这些参数根据应用需求调整一下，又可以得到新的测距配置。

4种测距配置的特点如下：

Table 6. Example API range profiles

	Timing budget 测量时间	Typical max range 最大测量距离	Typical application 典型应用
Default mode	30ms	1.2m (white target)	standard
High accuracy	200ms	1.2m (white target)	precise measurement
Long range	33ms	2m (white target)	long ranging, only for dark conditions 长距离，仅能用于黑暗环境
High Speed	20ms	1.2m (white target)	high speed where accuracy is not priority 需要高速但精度不是很重要的场合

不管以何种配置工作，VL53L0X每次从上电到进入测距状态，总共需要3个阶段，如下：



1、初始化和校准阶段：处于设备复位后和开始第一次测量前，如果用户的使用环境温度变化大，需要周期性进行温度校准。

2、测距阶段：VL53L0X会发射几个激光脉冲，然后被目标反射回来，再被SPAD阵列接收到进行测距。

3、数据处理阶段：VL53L0X计算测量距离，如果信号太弱或没有目标，VL53L0X会返回错误码。VL53L0X会处理如下工作：信号强弱检查、Offset矫正、CrossTalk矫正、距离值计算。为了进一步提高精度，用户可以自己对读取到的距离值进行：多次取平均、迟滞处理、滤波处理。

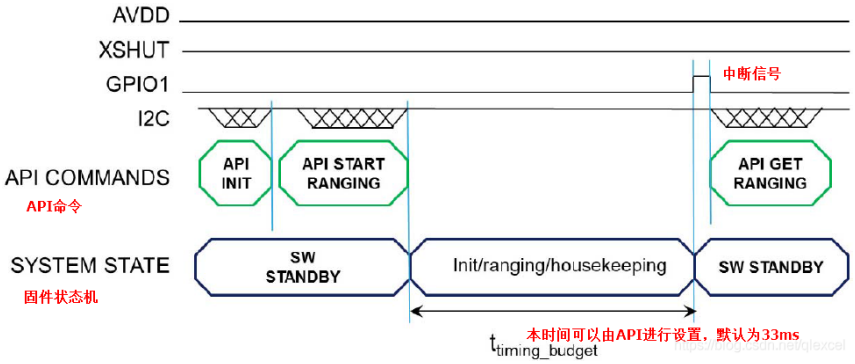
六、测距

1、用户可以通过轮询或中断的方式来获取数据

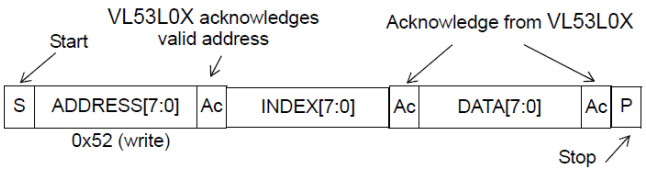
（1）轮询：需要主动去读API函数，获取测量状态。

（2）中断：当一次测量完成后，VL53L0X通过GPIO1引脚发送中断信号。

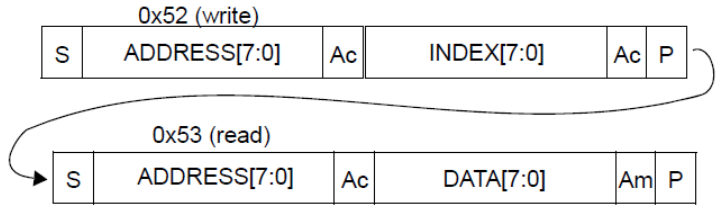
2、测距过程



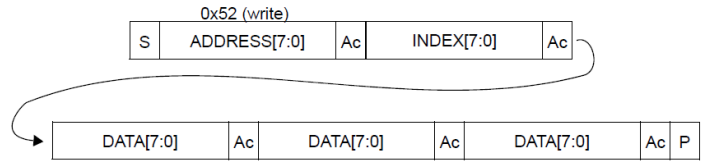
3、IIC写1字节数据



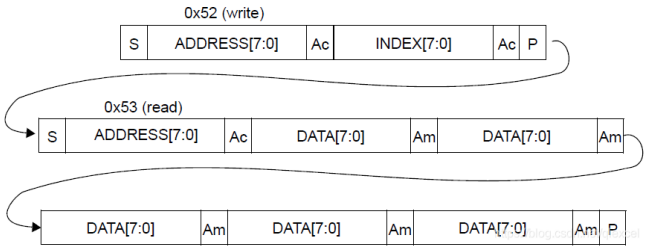
4、IIC读1字节数据



5、IIC写多个字节数据



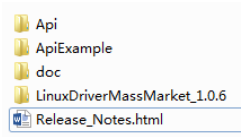
6、IIC读多个字节数据



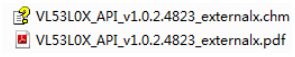
七、API移植

API包在此处下载：API。还需要一个编程手册：编程手册。

API包解压后如下：

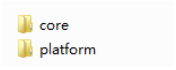


API包的doc文件夹下有对API中所有结构体、函数、宏定义的介绍，要使用哪些函数，怎么使用，都可以去查询，提供pdf和chm版本：

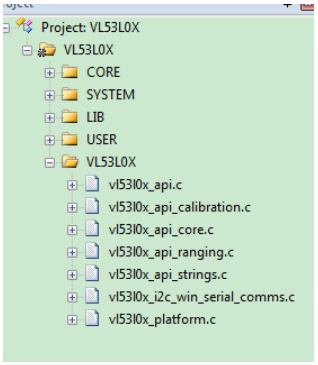


下面开始移植：

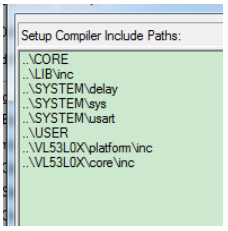
建立一个干净的工程，在工程文件夹下新建一个VL53L0X文件夹，把API包里的Api文件夹下的两个文件夹复制进去。



打开工程，新建一个VL53L0X目录，添加如下文件：



再把头文件包含进去：



编译一下，会在vl53l0x_i2c_win_serial_comms.c和vl53l0x_platform.c中报错。

vl53l0x_i2c_win_serial_comms.c是IIC的硬件实现层，其中实现了IIC的初始化、IIC关闭、读数据、写数据，共4个功能。现在移植到STM32平台后，要根据自己的硬件重新改。首先把相关的函数内容删除：

```
int VL53L0X_i2c_init(char *comPortStr, unsigned int baudRate) // mja
{
    unsigned int status = STATUS_FAIL;
    return status;
```

```
}
int32_t VL53L0X_comms_close(void)
{
    unsigned int status = STATUS_FAIL;
    return status;
}

int32_t VL53L0X_write_multi(uint8_t address, uint8_t reg, uint8_t *pdata, int32_t count)
{
    int32_t status = STATUS_OK;
    return status;
}

int32_t VL53L0X_read_multi(uint8_t address, uint8_t index, uint8_t *pdata, int32_t count)
{
    int32_t status = STATUS_OK;
    return status;
}
```

vl53l0x_platform.c中有一个延时函数报错，同样把它的内容删除：

```
VL53L0X_Error VL53L0X_PollingDelay(VL53L0X_DEV Dev){
    VL53L0X_Error status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    LOG_FUNCTION_END(status);
    return status;
}
```

把其他的小错误排除后，再次编译，就可以通过了。

接下来把IIC的4个函数根据自己的硬件重新写好，新建vl53l0x_IIC.c和vl53l0x_IIC.h文件添加进工程中。vl53l0x_IIC.c内容如下：

```
#include "vl53l0x_IIC.h"

void VL53L0X_IIC_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd( VL_SDA_RCC, ENABLE );
    GPIO_InitStructure.GPIO_Pin = VL_SDA_PIN;               //端口配置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP ;       //推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       //50Mhz速度
    GPIO_Init(VL_SDA_IOx, &GPIO_InitStructure);

    RCC_APB2PeriphClockCmd( VL_SCL_RCC, ENABLE );
    GPIO_InitStructure.GPIO_Pin = VL_SCL_PIN;               //端口配置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP ;       //推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       //50Mhz速度
    GPIO_Init(VL_SCL_IOx, &GPIO_InitStructure);

    GPIO_SetBits(VL_SDA_IOx,VL_SDA_PIN);//SDA输出高
    GPIO_SetBits(VL_SCL_IOx,VL_SCL_PIN);//SCL输出高
}

void VL_IIC_Start(void)
{
    VL_SDA_OUT();//sda线输出
    VL_IIC_SDA=1;
    VL_IIC_SCL=1;
    delay_us(4);
    VL_IIC_SDA=0;//START:when CLK is high,DATA change form high to low
    delay_us(4);
    VL_IIC_SCL=0;//钳住I2C总线，准备发送或接收数据
}

//产生IIC停止信号
void VL_IIC_Stop(void)
{
    VL_SDA_OUT();//sda线输出
    VL_IIC_SCL=0;
    VL_IIC_SDA=0;//STOP:when CLK is high DATA change form low to high
    delay_us(4);
    VL_IIC_SCL=1;
    VL_IIC_SDA=1;//发送I2C总线结束信号
    delay_us(4);
}

u8 VL_IIC_Wait_Ack(void)
{
    u8 ucErrTime=0;
    VL_SDA_IN();  //SDA设置为输入
    VL_IIC_SDA=1;delay_us(1);
```

```
    VL_IIC_SCL=1;delay_us(1);
    while(VL_READ_SDA)
    {
        ucErrTime++;
        if(ucErrTime>250)
        {
            VL_IIC_Stop();
            return 1;
        }
    }
    VL_IIC_SCL=0;//时钟输出0
    return 0;
}

//产生ACK应答
void VL_IIC_Ack(void)
{
    VL_IIC_SCL=0;
    VL_SDA_OUT();
    VL_IIC_SDA=0;
    delay_us(2);
    VL_IIC_SCL=1;
    delay_us(2);
    VL_IIC_SCL=0;
}

//不产生ACK应答
void VL_IIC_NAck(void)
{
    VL_IIC_SCL=0;
    VL_SDA_OUT();
    VL_IIC_SDA=1;
    delay_us(2);
    VL_IIC_SCL=1;
    delay_us(2);
    VL_IIC_SCL=0;
}

//IIC发送一个字节
//返回从机有无应答
//1，有应答
//0，无应答
void VL_IIC_Send_Byte(u8 txd)
{
    u8 t;
    VL_SDA_OUT();
    VL_IIC_SCL=0;//拉低时钟开始数据传输
    for(t=0;t<8;t++)
    {
        if((txd&0x80)>>7)
            VL_IIC_SDA=1;
        else
            VL_IIC_SDA=0;
        txd<<=1;
        delay_us(2);
        VL_IIC_SCL=1;
        delay_us(2);
        VL_IIC_SCL=0;
        delay_us(2);
    }
}

//读1个字节，ack=1时，发送ACK，ack=0，发送nACK
u8 VL_IIC_Read_Byte(void)
{
    unsigned char i,receive=0;
    VL_SDA_IN();//SDA设置为输入
    VL_IIC_SDA = 1;
    delay_us(4);
    for(i=0;i<8;i++ )
    {
        receive<<=1;
        VL_IIC_SCL=0;
        delay_us(4);
        VL_IIC_SCL=1;
        delay_us(4);
        if(VL_READ_SDA)
            receive |= 0x01;
        delay_us(4); //1
    }
    VL_IIC_SCL = 0;
    return receive;
}

//IIC写一个字节数据
```

```
u8 VL_IIC_Write_1Byte(u8 SlaveAddress, u8 REG_Address,u8 REG_data)
{
    VL_IIC_Start();
    VL_IIC_Send_Byte(SlaveAddress);
    if(VL_IIC_Wait_Ack())
    {
        VL_IIC_Stop();//释放总线
        return 1;//没应答则退出

    }
    VL_IIC_Send_Byte(REG_Address);
    VL_IIC_Wait_Ack();
    delay_us(5);
    VL_IIC_Send_Byte(REG_data);
    VL_IIC_Wait_Ack();
    VL_IIC_Stop();

    return 0;
}

//IIC读一个字节数据
u8 VL_IIC_Read_1Byte(u8 SlaveAddress, u8 REG_Address,u8 *REG_data)
{
    VL_IIC_Start();
    VL_IIC_Send_Byte(SlaveAddress);//发写命令
    if(VL_IIC_Wait_Ack())
    {
         VL_IIC_Stop();//释放总线
         return 1;//没应答则退出
    }
    VL_IIC_Send_Byte(REG_Address);
    VL_IIC_Wait_Ack();
    delay_us(5);
    VL_IIC_Start();
    VL_IIC_Send_Byte(SlaveAddress|0x01);//发读命令
    VL_IIC_Wait_Ack();
    *REG_data = VL_IIC_Read_Byte();
    VL_IIC_Stop();

    return 0;
}

//I2C读多个字节数据
uint8_t VL_I2C_Read_nByte(uint8_t SlaveAddress, uint8_t REG_Address, uint8_t *buf, uint16_t len)
{
    VL_IIC_Start();
    VL_IIC_Send_Byte(SlaveAddress);//发写命令
    if(VL_IIC_Wait_Ack())
    {
        VL_IIC_Stop();//释放总线
        return 1;//没应答则退出
    }
    VL_IIC_Send_Byte(REG_Address);
    VL_IIC_Wait_Ack();
    delay_us(5);
    VL_IIC_Start();
    VL_IIC_Send_Byte(SlaveAddress|0x01);//发读命令
    VL_IIC_Wait_Ack();
    while(len)
    {
        *buf = VL_IIC_Read_Byte();
        if(1 == len)
        {
            VL_IIC_NAck();
        }
        else
        {
            VL_IIC_Ack();
        }
        buf++;
        len--;
    }
    VL_IIC_Stop();

    return STATUS_OK;
}

//I2C写多个字节数据
uint8_t VL_I2C_Write_nByte(uint8_t SlaveAddress, uint8_t REG_Address, uint8_t *buf, uint16_t len)
{
    VL_IIC_Start();
    VL_IIC_Send_Byte(SlaveAddress);//发写命令
    if(VL_IIC_Wait_Ack())
    {
```

```
        VL_IIC_Stop();//释放总线220          return 1;//没应答则退出
    }
    VL_IIC_Send_Byte(REG_Address);
    VL_IIC_Wait_Ack();
    while(len--)
    {
        VL_IIC_Send_Byte(*buf++);
        VL_IIC_Wait_Ack();
    }
    VL_IIC_Stop();

    return STATUS_OK;
}

```

vl53l0x_IIC.h内容如下：

```
#ifndef __VL53L0X_IIC_H
#define __VL53L0X_IIC_H

#include "stdint.h"
#include "sys.h"
#include "delay.h"

#define VL_SDA_RCC  RCC_APB2Periph_GPIOB
#define VL_SDA_PIN  GPIO_Pin_11
#define VL_SDA_IOx  GPIOB
#define VL_SCL_RCC  RCC_APB2Periph_GPIOB
#define VL_SCL_PIN  GPIO_Pin_10
#define VL_SCL_IOx  GPIOB

#define VL_SDA_IN()  {GPIOB->CRH&=0xFFFF0FFF;GPIOB->CRH|=8<<12;}
#define VL_SDA_OUT() {GPIOB->CRH&=0xFFFF0FFF;GPIOB->CRH|=3<<12;}
#define VL_IIC_SCL    PBout(10)      //SCL
#define VL_IIC_SDA    PBout(11)      //SDA
#define VL_READ_SDA   PBin(11)       //输入SDA

//Status
#define STATUS_OK       0x00
#define STATUS_FAIL     0x01

void VL53L0X_IIC_Init(void);
uint8_t VL_I2C_Read_nByte(uint8_t SlaveAddress, uint8_t REG_Address, uint8_t *buf, uint16_t len);
uint8_t VL_I2C_Write_nByte(uint8_t SlaveAddress, uint8_t REG_Address, uint8_t *buf, uint16_t len);

#endif

```

于是vl53l0x_i2c_win_serial_comms.c中的4个底层函数改写为：

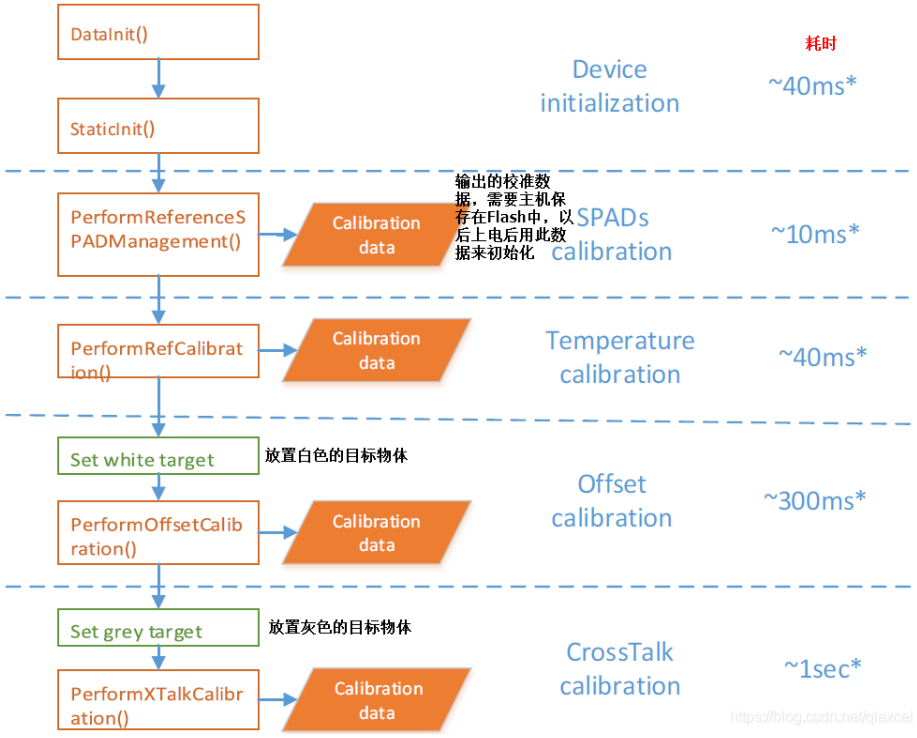
```
int VL53L0X_i2c_init(char *comPortStr, unsigned int baudRate) // mja
{
    unsigned int status = STATUS_FAIL;
    return status;
}
int32_t VL53L0X_comms_close(void)
{
    unsigned int status = STATUS_FAIL;
    return status;
}

int32_t VL53L0X_write_multi(uint8_t address, uint8_t reg, uint8_t *pdata, int32_t count)
{
    int32_t status = STATUS_OK;
    if(VL_I2C_Write_nByte(address,reg,pdata,count))
       status  = STATUS_FAIL;
    return status;
}

int32_t VL53L0X_read_multi(uint8_t address, uint8_t index, uint8_t *pdata, int32_t count)
{
    int32_t status = STATUS_OK;
    if(VL_I2C_Read_nByte(address,index,pdata,count))
      status  = STATUS_FAIL;
    return status;
}
```

八、API使用

1、校准

为获得更好的精度，当使用环境确定后，要校准一次。校准的过程中，需要主机把校准的数据保存下来，完成校准后在后面的使用过程中只要环境变化不大，都可以直接使用校准数据即可，不需要每次都校准。

下面的校准过程必须按照顺序一步一步来，不能颠倒。



(1) Data init

调用VL53L0X_DataInit()函数一次，设备上电后调用一次。把VL53L0X_State从VL53L0X_STATE_POWERDOWN改为 VL53L0X_STATE_WAIT_STATICINIT。VL53L0X_State是初始化状态机，看此变量的值可以就可以知道当前的初始化进度。

(2) Static Init

调用VL53L0X_StaticInit()函数来做基本的设备初始化。把VL53L0X_State从VL53L0X_STATE_WAIT_STATICINIT改为VL53L0X_STATE_IDLE.

(3) 参考SPADs校准（Reference SPADs calibration ）

参考SPADs校准会在从ST出厂的最后一次模块测试时在裸VL53L0X（没有盖玻片）上进行，校准数据（SPAD数量和类型）会被存入VL53L0X的Flash。

如果用户要加盖玻片使用，就需要重新校准参考SPADs。

校准的数据（SPAD数量和类型），VL53L0X会自动保存到自己的Flash中，用户可以读出来，然后直接赋值给VL53L0X来初始化。用户也可以把校准数据保存到单片机的Flash中，然后赋值给VL53L0X来初始化。总之，参考SPADs只需要校准一次，然后把校准数据保存好，以后直接赋值即可，不需要每次都校准。

校准过程不需要任何条件，直接调用VL53L0X_PerformRefSpadManagement()函数即可，此函数会输出校准数据（SPADs的数量和类型），当获取到这两个值后，VL53L0X会自动编程到自己的Flash中。同时主机也要保存这两个参数用于以后的参数初始化。

虽然校准过程不需要目标物或者光照条件，但是如果放置一个高反射率的目标物体在VL53L0X前面，导致过多的激光被检测到，会导致校准失败，报'-50'状态码。出现这种情况，要把目标物体移开再校准。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_PerformRefSpadManagement (VL53L0X_DEV Dev, uint32_t * refSpadCount, uint8_t * isApertureSpads)

函数功能：参调用此函数确定要启用的参考SPAD的最小数量，以实现对目标的最佳测量。此函数同时也应该在初始化期间执行一次。

Parameters:

Dev	Device Handle 设备句柄
refSpadCount	Reports ref Spad Count 参考SPAD数量
isApertureSpads	Reports if spads are of type aperture or non-aperture. 1:=aperture, 0:=Non-Aperture SPAD类型。aperture：有光圈。Non-Aperture：无光圈

(4) 温度校准（Ref (temperature) calibration)

温度校准会输出与温度相关的两个参数：VHV 和 phase cal。这两个参数还可以用来设置VL53L0X的灵敏度。

当温度变化大于8度，建议重新校准。如果温度没变，校准数据可以直接加载使用。

温度校准也不需要任何条件，直接调用VL53L0X_PerformRefCalibration()函数即可。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_PerformRefCalibration (VL53L0X_DEV Dev, uint8_t * pVhvSettings, uint8_t * pPhaseCal)

Parameters:

Dev	Device Handle
pVhvSettings	Pointer to vhv settings parameter.
pPhaseCal	Pointer to PhaseCal parameter.

（5）偏移校准（Offset calibration）

偏移校准也是在从ST出厂的最后一次模块测试时在裸VL53L0X（没有盖玻片）上进行的，校准数据（偏移值）会被存入VL53L0X的Flash。

当VL53L0X的使用环境改变（温度、加了盖玻片），可能会出现很大的偏差，就需要重新校准偏移值。

偏移校准所需条件：使用白色目标物体（88%反射率）、把目标物体放置在100mm处（实际的距离）、在黑暗环境中。100mm的校准距离可以根据用户的应用作出调整，但必须在测距曲线的线性部分进行选择，因为偏移值应该是一个常数，如果处于非线性区，偏移值就不好确定了。

准备好条件后，调用VL53L0X_PerformOffsetCalibration()函数。函数需要输入校准距离。函数输出的就是偏移值，单位是mm。把校准值保存到单片机的Flash中，以后直接赋值即可。

需要注意的是，函数的输入校准距离要左移16位，比如校准距离值为100mm：

```
1    Status=VL53L0X_PerformOffsetCalibration(pMyDevice,100<<16,&OffsetMicroMeter);          //偏移校准，把白色物体放在100mm处进行
2    if(Status!=VL53L0X_ERROR_NONE) { return Status; }
```

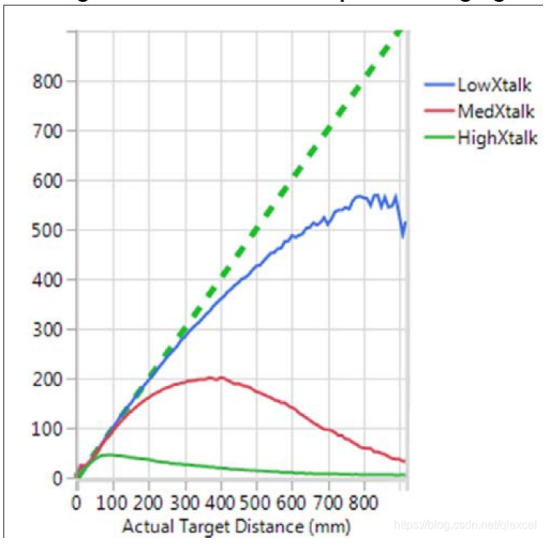
校准完成后，OffsetMicroMeter变量的值就是偏移值，偏移校准完成后，再去读测量距离值就变成100了，即传入函数的校准距离值，可以根据这个来判断偏移校准是否成功。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_PerformOffsetCalibration (VL53L0X_DEV Dev, FixPoint1616_t CalDistanceMilliMeter, int32_t * pOffsetMicroMeter)

Dev	Device Handle
CalDistanceMilli Meter	Calibration distance value used for the offset compensation. 用于偏移补偿的校准距离值
pOffsetMicroMeter	Pointer to new Offset value computed by the function. 由本函数计算的新偏移值

（6）盖玻片校准（Cross-talk calibration）

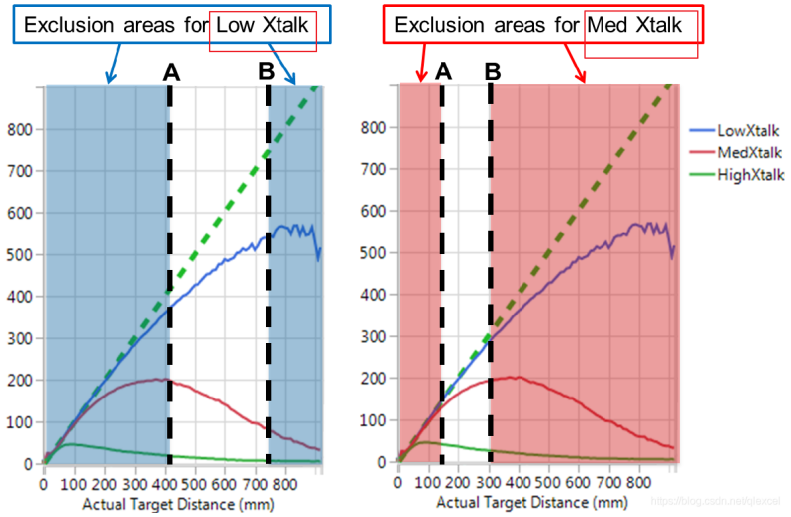
Figure 3. Cover window impact on ranging



盖玻片对测距的影响如上，LowXtalk是指VL53L0X与盖玻片之间空气间隙的大小，可见间隙越大，影响越大。上图中的绿点线是理想测距线。

对于盖玻片的校准就是对测量距离值进行加权放大，气隙越小矫正效果越好，如果已经是上图绿线那样了，就难了。

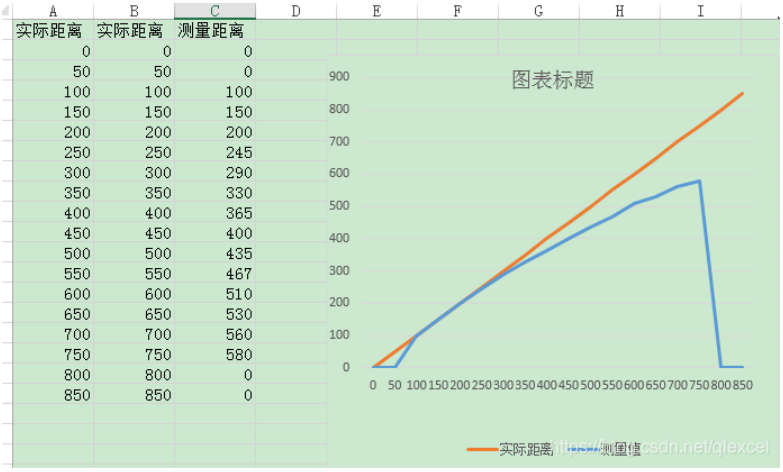
要想开始校准过程，就先要确定好校准距离。校准距离区间的开始点一般是测量值开始偏离出理想测距线的时候。结束点一般是信号太弱，虽然实际距离在增加，测量距离却开始减小的时候。如下：



左图中可以看到蓝线在A点开始大幅度偏离绿点线，在B点实际距离在增加，测量距离却要开始减小了。右图同样。因此选择A点和B点间的任何一个距离作为校准距离即可。

如果校准距离选在了线性区域，校准因数会太小，导致矫正没有效果。

要进行盖玻片校准，用户需要在校准前把测量距离与实际距离的关系图绘制出来，然后选择好校准距离。目标物体要使用17%反射率的灰色物体。下图为笔者测得的数据，校准距离选择450或500。



实际距离	实际距离	测量距离
0	0	0
50	50	0
100	100	100
150	150	150
200	200	200
250	250	245
300	300	290
350	350	330
400	400	365
450	450	400
500	500	435
550	550	467
600	600	510
650	650	530
700	700	560
750	750	580
800	800	0
850	850	0

做好准备后调用VL53L0X_PerformXTalkCalibration()函数，函数的输入参数就是校准距离，输出值是盖玻片校准因数，要把校准因数保存到主机中，以便于以后使用。

函数的输入参数依然要左移16位，比如校准距离450：

```
    Status=VL53L0X_PerformXTalkCalibration(pMyDevice,450<<16,&XTalkCompensation);      //盖玻片校准，把灰色物体放在校准距离处进行
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }
```

校准完成后，把目标物体放在450mm处，测量的距离就应该是450mm了，也可以以此来判断校准是否成功。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_PerformXTalkCalibration (VL53L0X_DEV Dev, FixPoint1616_t XTalkCalDistance, FixPoint1616_t * pXTalkCompensationRateMegaCps)

XTalkCalDistance	XTalkCalDistance value used for the XTalk computation. 校准距离
pXTalkCompensationRateMegaCps	Pointer to new XTalkCompensation value. 校准因数

总结：

1、VL53L0X有4种校准：参考SPADs校准、温度校准、偏移校准、盖玻片校准。

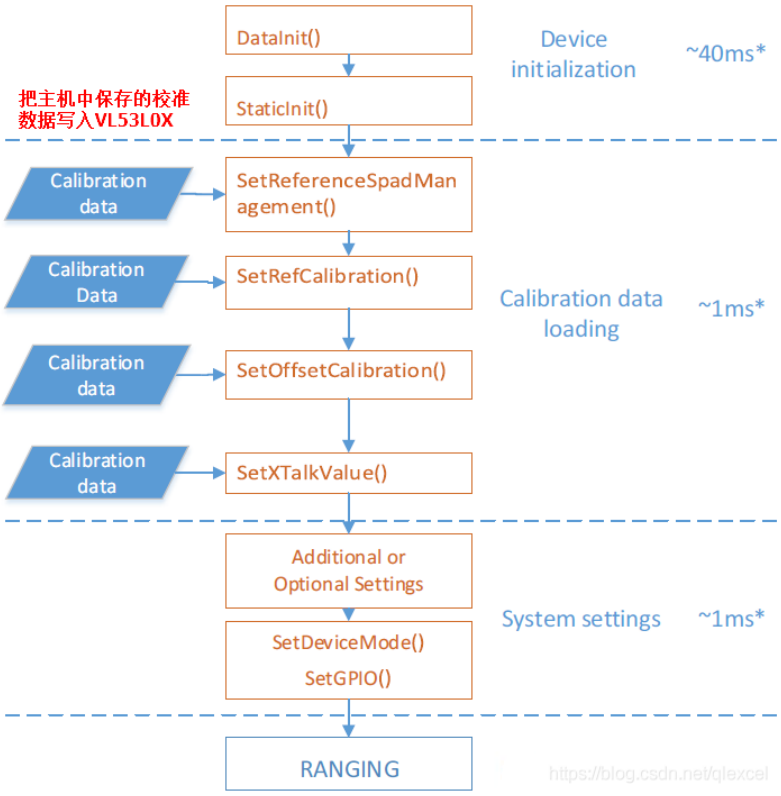
2、温度校准不需要任何条件，而且温度最容易改变，因此可以每次上电都校准一次。

3、确定好自己产品上给VL53L0X提供的电压值、需不需要盖玻片后，可以拿一台出来进行参考SPADs校准、偏移校准、盖玻片校准，然后把校准参数取出来做成常量直接放到程序中，给其他的产品直接赋值即可，这样就不用每台都进行繁琐的校准过程了。

2、上电开始测距

校准完成后，以后每次上电只需要3个阶段来进入测距状态：初始化和加载校准数据阶段、测距阶段、数据处理阶段。

1）、初始化和加载校准数据阶段



(1) Data init

调用VL53L0X_DataInit()函数一次。

(2) Static Init

调用VL53L0X_StaticInit()函数做基本初始化。

(3) 设置参考SPADs校准值（Reference SPADs）

调用VL53L0X_SetReferenceSpads() 用于设置参考SPADs的数量和类型。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_SetReferenceSpads (VL53L0X_DEV Dev, uint32_t refSpadCount, uint8_t isApertureSpads)

Parameters:

Dev	Device Handle
refSpadCount	Number of ref spads.
isApertureSpads	Defines if spads are of type aperture or non-aperture. 1:=aperture, 0:=Non-Aperture

VL53L0X_GetReferenceSpads()用于读取SPADs的数量和类型，如果设备参数结构体没有值，就会去VL53L0X的Flash中去读。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_GetReferenceSpads (VL53L0X_DEV Dev, uint32_t * refSpadCount, uint8_t * isApertureSpads)

(4) 设置温度校准值（Ref calibration）

调用VL53L0X_SetRefCalibration()函数，设置温度校准数据。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_SetRefCalibration (VL53L0X_DEV Dev, uint8_t VhvSettings, uint8_t PhaseCal)

(5) 设置偏移校准值（Offset calibration）

调用VL53L0X_SetOffsetCalibrationDataMicroMeter()函数设置偏移值，单位mm。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_SetOffsetCalibrationDataMicroMeter (VL53L0X_DEV Dev, int32_t OffsetCalibrationDataMicroMeter)

Parameters:

Dev	Device Handle
OffsetCalibration DataMicroMeter	Offset (microns) 偏移值，mm

(6) 设置盖玻片校准值（Cross-talk correction）

调用VL53L0X_SetXTalkCompensationRateMegaCps()设置盖玻片校准因数，再调用VL53L0X_SetXTalkCompensationEnable()函数来使能盖玻片矫正测量距离值。

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_SetXTalkCompensationRateMegaCps (VL53L0X_DEV Dev, FixPoint1616_t XTalkCompensationRateMegaCps)

Parameters:

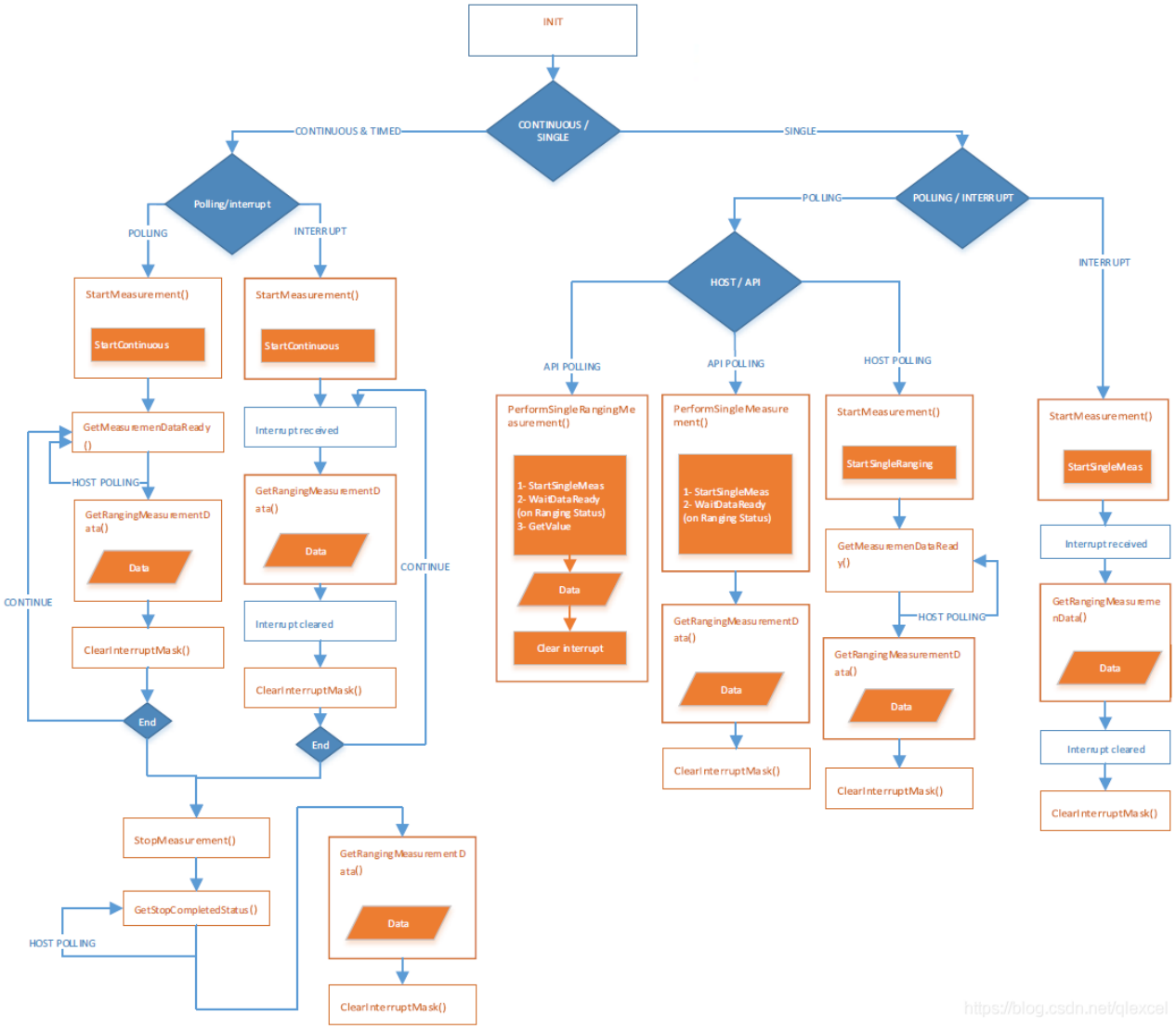
Dev	Device Handle
XTalkCompensationRateMegaCps	Compensation rate in Mega counts per second (16.16 fix point) see datasheet for details 盖玻片校准因数

函数详情：VL53L0X_API VL53L0X_Error VL53L0X_SetXTalkCompensationEnable (VL53L0X_DEV Dev, uint8_t XTalkCompensationEnable)

Parameters:

Dev	Device Handle
XTalkCompensationEnable	Cross talk compensation to be set 0=disabled else = enabled 使能盖玻片矫正测量到的距离值

2）、测距阶段



（1）测距模式

调用VL53L0X_SetDeviceMode()来设置工作模式：单次、连续、间隔连续。

调用VL53L0X_GetDeviceMode()来获取当前工作模式。

（2）轮询或中断

一旦测距完成，VL53L0X会输出一个中断信号或置位状态寄存器。

VL53L0X_SetGPIOConfig()用于设置中断模式。

（3）开始测距

A、调用VL53L0X_StartMeasurement()函数，VL53L0X会按照之前设置的测距模式开始测距。

B、调用VL53L0X_PerformSingleMeasurement()函数，VL53L0X会开始测距并且等待测距完成。此函数内部调用了VL53L0X_StartMeasurement()和VL53L0X_GetMeasurementDataReady()。

C、调用VL53L0X_PerformSingleRangingMeasurement()函数，VL53L0X会开始测距，然后等待测距完成，接着清除中断，最后输出测量数据退出。此函数内部调用了VL53L0X_PerformSingleMeasurement()、VL53L0X_GetRangingMeasurementData()

和VL53L0X_ClearInterruptMask()。

（4）停止测距

调用VL53L0X_StopMeasurement()函数来停止测距。

（5）获取测量数据

调用VL53L0X_GetMeasurementDataReady()函数来获取测量状态。

调用VL53L0X_GetRangingMeasurementData()函数来获取测量数据，此函数返回一个buffer，包含如下内容：

RangeMilliMeter：以mm为单位的测量距离。
RangeDMaxMilliMeter:最大测量距离，mm为单位。
SignalRateRtnMegaCps：返回的信号大小（signal rate (MCPS)），由目标物的反射率决定。
AmbientRateRtnMegaCps：返回的环境信号大小（ambient rate (MCPS)），由环境光决定。
EffectiveSpadRtnCount
RangeStatus：测量状态

其中RangeStatus有如下取值：

Table 1. Range Status

RangeStatus value	RangeStatus String	Comment
0	Range Valid	Ranging measurement is valid 距离测量有效
1	Sigma Fail	Sigma fail will trigger particularly in ambient light, when the amount of ambient light is adding too much noise onto the ranging measurement. 当环境光噪声对于测量干扰太大时触发
2	Signal Fail	Signal fail will trigger when the return signal is too low to give enough confidence on the range measured. The limit will be given by either the signal limit or the RIT (Range Ignore Threshold). 当接收到的返回信号太弱时触发
3	Min Range Fail	Not enabled as default. 默认不使能
4	Phase Fail	Phase fail will trigger when wraparound conditions are detected or when noise on signal is too high. 当检测到VL53L0X被覆盖住了或信号中的噪声太强时触发
5	Hardware Fail	Hardware Fail will trigger if a VCSEL failure, or VHV fail are detected. 当激光发射器出错或VHV出错时触发
255	No Update	This error should not trigger.

要获取测距状态的字符串（即上表中的第二列），可以调用VL53L0X_GetRangeStatusString()函数。

L53L0X_GetDeviceErrorStatus()函数只能在调试过程中使用，一般不要用。

测量数据中的 SignalRateRtnMegaCps 是经过盖玻片矫正后的数据（如果使能了盖玻片矫正），如果要得到没矫正的数据，调用VL53L0X_GetTotalSignalRate()函数。

3、API的其他功能

（1）调整总体测量时间

调用VL53L0X_SetMeasurementTimingBudgetMicroSeconds()和VL53L0X_GetMeasurementTimingBudgetMicroSeconds()来设置或获取总体测量时间。

总体测量时间默认为33ms，最小值为20ms。如：

Status =VL53L0X_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,66000)就是把总体测量时间设置为66ms。

增加测量时间会增加测量精度。

（2）限制值设置（Limit settings）

用户可以启用/禁用限制检查和限制值。

禁用或放宽这些限制可以允许更长的测量范围，在这种情况下，标准偏差将增加，主机将接收测量异常值。

有3个限制值可以调整：

A、Sigma：VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE

Sigma是参考和返回SPAD阵列之间的时差（移位）。Sigma本来表示的是光的传播时间，在芯片内会转换成距离，所以这个参数是用mm表示的。

B、Return Signal Rate：VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE

返回信号大小的测量值，用MCPS表示。这表示从目标反射并由设备检测到的信号的幅度。

C、Range Ignore Threshold：VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD

接收到信号的最小阈值。信号大小低于此值的测量将被忽略。这可确保不会由于外壳反射而进行错误测量。

使用VL53L0X_SetLimitCheckEnable() 和 VL53L0X_GetLimitCheckEnable() 来设置使能或禁止限制值。

使用VL53L0X_SetLimitCheckValue() 和VL53L0X_GetLimitCheckValue()来设置和获取限制值大小。

使用VL53L0X_GetLimitCheckCurrent() 和 VL53L0X_GetLimitCheckStatus()可以得到当前值和与限制值进行比较的状态。

限制值的默认使能状态和值：

Table 2. Default limit states and values

Limit ID	Default limit state	Default limit value
Sigma	Enabled	18mm
Return Signal	Enabled	0.25Mcps
Range Ignore Threshold	Disabled	N x Xtalk Mcps/spad where N = 1.5 by default

使用举例：

```
Status = VL53L0X_SetLimitCheckEnable(pMyDevice,VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 1);
Status = VL53L0X_SetLimitCheckValue(pMyDevice,VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 0.40*65536);
```

读取当前的sigma值：

```
Status = VL53L0X_GetLimitCheckCurrent(pMyDevice,VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE, &Sigma);
```

（3）连续间隔模式的测量间隔时间设置

VL53L0X_SetInterMeasurementPeriodMilliSeconds()和VL53L0X_GetInterMeasurementPeriodMilliSeconds()

（4）API版本和产品版本读取

VL53L0X_GetVersion()
VL53L0X_GetPalSpecVersion()
VL53L0X_GetProductRevision()

4、API的状态码和错误码

调用VL53L0X_GetPalState()来获取API的状态值（下表的第一列）。VL53L0X_GetPalStateString()返回API的状态字符串（下表的第二列）。

Table 3. API state description

API state value	API state string	Comment
0	POWERDOWN state	Device is in HW reset
1	Wait for StaticInit State	Device is initialized and wait for static initialization
2	STANDBY State	Device is in low power Standby mode
3	IDLE State	Not used
4	RUNNING State	Device is performing measurement
98	UNKNOWN State	Device is in unknown state and needs to be rebooted
99	ERROR State	Device is in error state and needs to be rebooted

调用API函数都会返回API的错误码。VL53L0X_GetPalErrorString()函数返回错误字符串。

Table 4. API error values and error strings description

API error value	API error string	Occurrence	Possible root cause
0	No Error	-	-
-1	Calibration warning error	Not implemented, cannot happen	N/A
-2	Min clipped error	Not implemented, cannot happen	N/A
-3	Undefined error	Not implemented, cannot happen	N/A
-4	Invalid parameters error	Parameter passed is invalid or out of range	Wrong API programming (wrong setting used), or I2C transaction corrupted
-5	Not supported error	Function is not supported in current mode or configuration	Wrong API programming (non implemented function called), or I2C transaction corrupted
-6	Range error	Device reports a ranging error interrupt status	Wrong API programming (syntax error), or no target present during offset calibration, or I2C transaction corrupted
-7	Time out error	Aborted due to time out	Device is functionally failing, or I2C transaction corrupted
-8	Mode not supported error	Requested mode is not supported by the device	Wrong API programming (non existent mode called), or I2C transaction corrupted
-9	Buffer too small	Cannot happen	N/A

API error value	API error string	Occurrence	Possible root cause
-10	GPIO not existing	User tried to set up a non-existing GPIO pin	Wrong API programming (wrong setting used), or I2C transaction corrupted
-11	GPIO functionality not supported	Unsupported GPIO functionality	Wrong API programming (wrong setting used), or I2C transaction corrupted
-20	Control Interface Error	Error reported from IO functions (comm error)	I2C comm error
-30	Invalid Command Error	Cannot happen	N/A
-40	Division by zero Error	Cannot occur in cut1.1	N/A
-50	Reference SPAD Init Error	Error during reference SPAD initialization	Bad NVM programming, module aperture blocked with high reflective target, I2C transaction corrupted (wrong programming)
-99	Not implemented error	Not implemented	N/A

5、IIC设备地址设置

VL53L0X_SetDeviceAddress()函数可以改变IIC的设备地址

6、复位

VL53L0X_ResetDevice()

7、中断设置

使用VL53L0X_SetGpioConfig()和VL53L0X_GetGpioConfig()来设置和获取中断功能。有如下选项：

A、无中断。

B、当测量值小于下限时发生中断
C、当测量值大于上限时发生中断
D、当测量值小于下限或大于上限时发生中断

当中断阈值被编程为大于254mm并且也被设置为连续或连续定时模式时，在API中嵌入了一个专用的过程。在这种情况下，API在每次测距开始时加载一些特定的调谐参数，这将给第一次测距测量带来几毫秒的延迟（取决于主机I2C的性能）。

在单次测量模式中，最大的门限值为254mm。

Table 5. Interrupt threshold behaviour

Interrupt options	Ranging distance	Threshold value	Ranging mode	GPIO state
Level Low	> thresh_low	-	all	no interrupt
	< thresh_low	thresh_low < 254mm	all	interrupt at GPIO
	< thresh_low	thresh_low > 254mm	continuous or continuous timed	interrupt at GPIO
	< thresh_low	thresh_low > 254mm	single	no interrupt (limitation)
Level High	< thresh_high	-	all	no interrupt
	> thresh_high	thresh_high < 254mm	all	interrupt at GPIO
	> thresh_high	thresh_high > 254mm	continuous or continuous timed	interrupt at GPIO
	> thresh_high	thresh_high > 254mm	single	no interrupt (limitation)

VL53L0X_SetInterruptThresholds()和VL53L0X_GetInterruptThresholds()函数用于设置或获取中断门限值。

8、4种测距配置（range profiles）的设置

Table 6. Example API range profiles

	Timing budget	Typical max range	Typical application
Default mode	30ms	1.2m (white target)	standard
High accuracy	200ms	1.2m (white target)	precise measurement
Long range	33ms	2m (white target)	long ranging, only for dark conditions
High Speed	20ms	1.2m (white target)	high speed where accuracy is not priority

下面列出要把VL53L0X设置为这4种配置所需要的函数，都要在进入测量前执行：

A、高精度测距配置

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
             VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
             (FixPoint1616_t)(0.25*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
             VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
             (FixPoint1616_t)(18*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
            200000);
    }
```

B、长距离测距配置

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
             VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
             (FixPoint1616_t)(0.1*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
             VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
             (FixPoint1616_t)(60*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
            33000);
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetVcselPulsePeriod(pMyDevice,
             VL53L0_VCSEL_PERIOD_PRE_RANGE, 18);
```

```
    }
    if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetVcselPulsePeriod(pMyDevice,
             VL53L0_VCSEL_PERIOD_FINAL_RANGE, 14);
    }
```

C、高速测距配置

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
             VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
             (FixPoint1616_t)(0.25*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
             VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
             (FixPoint1616_t)(32*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
             20000);
    }
```

九、代码示例

完整工程在这儿：简单的测距。

依照上面介绍的思想，我们专门写一个校准函数，完成4个校准，然后把校准数据保存到程序中的常量中，以后直接赋值即可。

校准函数：

```
uint8_t  Debug_Ref=0;
uint32_t refSpadCount;
uint8_t  isApertureSpads;
uint8_t  VhvSettings;
uint8_t  PhaseCal;
int32_t  OffsetMicroMeter;
uint32_t XTalkCompensation;
VL53L0X_Error VL53L0x_Ref(VL53L0X_Dev_t *dev)   //校准函数
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    dev->I2cDevAddr = 0x52;        //I2C地址(上电默认0x52)
    dev->comms_type = 1;           //I2C通信模式
    dev->comms_speed_khz = 400;    //I2C通信速率

    VL53l0x_GPIO_Init();
    VL53L0X_IIC_Init();

    XSHUT_LOW;
    delay_ms(30);
    XSHUT_HIGH;                    //拉高XSHUT引脚
    delay_ms(30);

    Status = VL53L0X_DataInit(dev);
    if(Status!=VL53L0X_ERROR_NONE) goto MyError;

    Status = VL53L0X_StaticInit(dev);
    if(Status!=VL53L0X_ERROR_NONE) goto MyError;

    Status=VL53L0X_PerformRefSpadManagement(dev, &refSpadCount, &isApertureSpads); //参考SPADs校准
    if(Status!=VL53L0X_ERROR_NONE) goto MyError;

    Status=VL53L0X_PerformRefCalibration(dev, &VhvSettings, &PhaseCal);            //温度校准
    if(Status!=VL53L0X_ERROR_NONE) goto MyError;

    while(1){ if(Debug_Ref==1) { Debug_Ref=0; break; } }                          //等待校准条件准备好
    Status=VL53L0X_PerformOffsetCalibration(dev,100<<16,&OffsetMicroMeter);            //偏移校准，把白色物体放在100mm处进行
    if(Status!=VL53L0X_ERROR_NONE) goto MyError;

    while(1){ if(Debug_Ref==1) { Debug_Ref=0; break; } }                          //等待校准条件准备好
    Status=VL53L0X_PerformXTalkCalibration(dev,450<<16,&XTalkCompensation);            //盖玻片校准，把灰色物体放在校准距离处进行
    if(Status!=VL53L0X_ERROR_NONE) goto MyError;

MyError:
    while(1);
}
```

校准后把校准函数注释掉，使用下面函数来直接初始化即可：

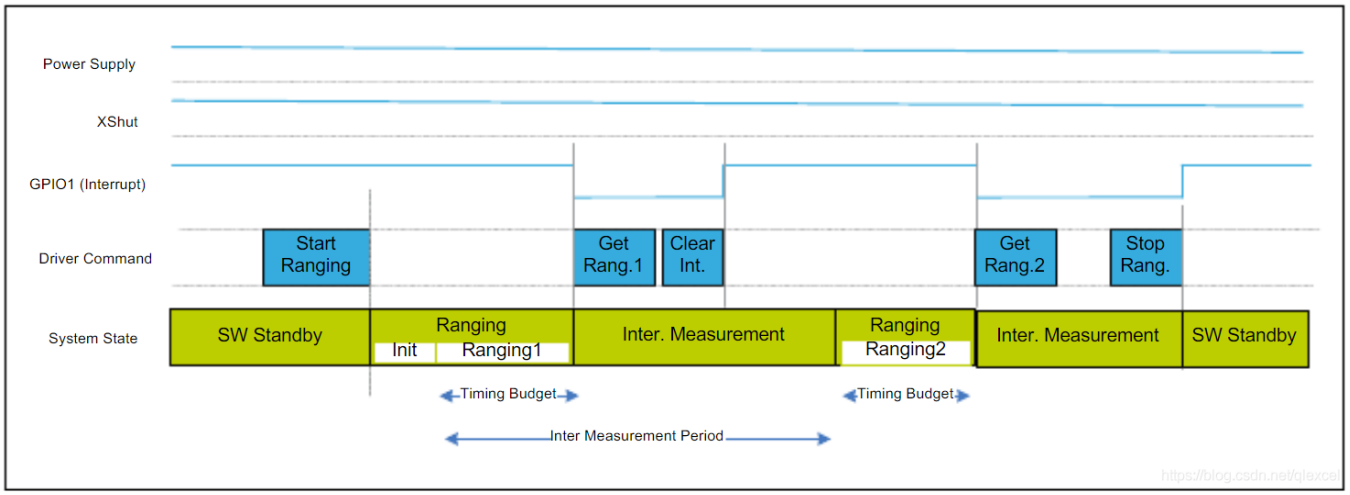
```
const uint32_t refSpadCount=4;
const uint8_t  isApertureSpads=0;
const uint8_t  VhvSettings=26;
const uint8_t  PhaseCal=1;
const int32_t  OffsetMicroMeter=15000;
const uint32_t XTalkCompensation=99;
VL53L0X_Error VL53L0x_Init(VL53L0X_Dev_t *dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    dev->I2cDevAddr = 0x52;        //I2C地址(上电默认0x52)
    dev->comms_type = 1;           //I2C通信模式
    dev->comms_speed_khz = 400;    //I2C通信速率

    VL53l0x_GPIO_Init();
    VL53L0X_IIC_Init();

    XSHUT_LOW;
    delay_ms(30);
    XSHUT_HIGH;
    delay_ms(30);

    Status = VL53L0X_DataInit(dev);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_StaticInit(dev);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetReferenceSpads(dev,refSpadCount,isApertureSpads);   //设定参考SPADs的校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status=VL53L0X_SetRefCalibration(dev,VhvSettings,PhaseCal);             //设定温度校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status=VL53L0X_SetOffsetCalibrationDataMicroMeter(dev,OffsetMicroMeter);//设定偏移校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status=VL53L0X_SetXTalkCompensationRateMegaCps(dev,XTalkCompensation);  //设定盖玻片校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetXTalkCompensationEnable(dev,1);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetDeviceMode(dev,VL53L0X_DEVICEMODE_CONTINUOUS_RANGING);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_StartMeasurement(dev);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    return Status;
}
```

上面是 默认模式的测距配置（range profiles），如果要使用长距离测距配置，初始化函数为：

```
VL53L0X_Error VL53L0x_Init(VL53L0X_Dev_t *dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    dev->I2cDevAddr = 0x52;        //I2C地址(上电默认0x52)
    dev->comms_type = 1;           //I2C通信模式
    dev->comms_speed_khz = 400;    //I2C通信速率

    VL53l0x_GPIO_Init();
    VL53L0X_IIC_Init();

    XSHUT_LOW;
    delay_ms(30);
    XSHUT_HIGH;
    delay_ms(30);

    Status = VL53L0X_DataInit(dev);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_StaticInit(dev);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetReferenceSpads(dev,refSpadCount,isApertureSpads);   //设定参考SPADs的校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status=VL53L0X_SetRefCalibration(dev,VhvSettings,PhaseCal);             //设定温度校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status=VL53L0X_SetOffsetCalibrationDataMicroMeter(dev,OffsetMicroMeter);//设定偏移校准值
```

```
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }
                                                    31
    Status=VL53L0X_SetXTalkCompensationRateMegaCps(dev,XTalkCompensation);  //设定盖玻片校准值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetXTalkCompensationEnable(dev,1);                     //使能盖玻片矫正
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetDeviceMode(dev,VL53L0X_DEVICEMODE_CONTINUOUS_RANGING);  //连续测量模式
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }


    Status = VL53L0X_SetLimitCheckValue(dev,VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,(FixPoint1616_t)(0.1*65536)); //返回信号大小的测量值
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetLimitCheckValue(dev,VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,(FixPoint1616_t)(60*65536));  //参考和返回SPAD阵列之间的时差
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status =VL53L0X_SetMeasurementTimingBudgetMicroSeconds(dev,33000);
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetVcselPulsePeriod(dev,VL53L0X_VCSEL_PERIOD_PRE_RANGE, 18);  //设定VCSEL脉冲周期
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    Status = VL53L0X_SetVcselPulsePeriod(dev,VL53L0X_VCSEL_PERIOD_FINAL_RANGE, 14);  //设定VCSEL脉冲周期范围
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }


    Status = VL53L0X_StartMeasurement(dev);                                 //开始测量
    if(Status!=VL53L0X_ERROR_NONE) { return Status; }

    return Status;
}
```

每次读取测量数据后，都要清除中断，不然中断引脚不会更新：



其他的一些初始化函数：

```
VL53L0X_SetLimitCheckEnable(dev,VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,1);   //开启SIGMA范围检查

//SIGMA表示是激光飞行时间

VL53L0X_SetLimitCheckEnable(dev,VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,1);    //使能信号速率范围检查

VL53L0X_SetLimitCheckValue(dev,VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,sigmaLimit);//设定SIGMA范围

VL53L0X_SetLimitCheckValue(dev,VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,signalLimit);//设定信号速率范围范围

VL53L0X_SetMeasurementTimingBudgetMicroSeconds(dev,timingBudget);//设定完整测距最长时间

VL53L0X_SetVcselPulsePeriod(dev, VL53L0X_VCSEL_PERIOD_PRE_RANGE, preRangeVcselPeriod);//设定VCSEL脉冲周期

VL53L0X_SetVcselPulsePeriod(dev, VL53L0X_VCSEL_PERIOD_FINAL_RANGE, finalRangeVcselPeriod);//设定VCSEL脉冲周期范围

//sigmaLimit, signalLimit, timingBudget, preRangeVcselPeriod, finalRangeVcselPeriod参数的典型值可以在API手册第7部分 Example API range profiles 找到
```