AN1703C ATK-VL53L0X 激光测距模块

使用说明

本应用文档（AN1703C）将教大家如何在 ALIENTEKT 探索者 STM32F407 开发板上使用 ATK-VL53L0X 激光测距模块。

本文档分为如下几部分：

1， ATK-VL53L0X 模块简介

2， 硬件连接

3， 软件实现

4， 验证

1、ATK-VL53L0X 模块简介

ATK-VL53L0X-V1.1(V1.1 是版本号，下面均以 ATK-VL53L0X 表示该产品)是 ALIENTEK 推出的一款高性能激光测距模块。ATK-VL53L0X 模块采用 ST 公司的 VL53L0X 芯片作为核心，该芯片内部集成了激光发射器和 SPAD 红外接收器，采用了第二代 FightSense™ 技术，通过接收器所接收到的光子时间来计算距离，测距的长度能扩至到两米，非常适合中短距离测量的应用。

ATK-VL53L0X 模块具有：体积小、测量精度高、多测量工作模式、支持从机地址设置和中断、兼容 3.3V/5V 系统、使用方便等特点，模块通过 6 个 2.54mm 间距的排针与外部连接，模块外观如图 1.1 所示：

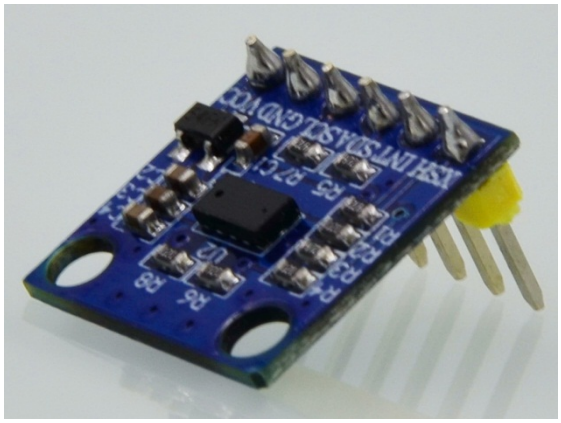

图 1.1 ATK-VL53L0X 模块外观图

1.1 VL53L0X 简介

VL53L0X 是 ST 公司推出的新一代 ToF 激光测距传感器，采用了第二代 FlightSense™ 技术，利用飞行时间（ToF）原理，通过光子的飞行来回时间与光速的计算，实现测距应用。较比上一代 VL6180X，新的器件将飞行时间测距长度扩展至 2 米，测量速度更快，能效更高。除此之外，为使集成度过程更加快捷方便，ST 公司为此也提供了 VL53L0X 软件 API（应用编程接口）以及完整的技术文档，通过主 IIC 接口，向应用端输出测距的数据，大大降低了开发难度。

VL53L0X 特点包括：

①，使用 940nm 无红光闪烁激光器，该频段的激光为不可见光，且不危害人眼。

②，系统视野角度(FOV)可达 25 度，传感器的感测有效工作直径扩展到 90 厘米。

③，采用脉冲式测距技术，避免相位式测距检测峰值的误差，利用了相位式检测中除波峰以外的光子。

④，多种精度测量和工作模式的选择。

⑤，测距距离能扩至到 2 米。

⑥，正常工作模式下功耗仅 20mW，待机功耗只有 5uA。

⑦，高达 400Khz 的 IIC 通信接口。

⑧，超小的封装尺寸：2.4mm × 4.4mm × 1mm。

关于 VL53L0X 传感器的介绍，我们就介绍到这里，详细的介绍，请看 ATK-VL53L0X 激光测距模块用户手册_V1.0.pdf。

1.2 VL53L0X 工作模式

VL53L0X 传感器提供了 3 种测量模式，Single ranging（单次测量）、Continuous ranging（连续测量）、以及 Timed ranging（定时测量），下面我们将简单介绍下：

（1）Single ranging（单次测量），在该模式下只触发执行一次测距测量，测量结束后，VL53L0X 传感器会返回待机状态，等待下一次触发。

（2）Continuous ranging（连续测量），在该模式下会以连续的方式执行测距测量。一旦测量结束，下一次测量就会立即启动，用户必须停止测距才能返回到待机状态，最后的一次测量在停止前完成。

（3）Timed ranging（定时测量），在该模式下会以连续的方式执行测距测量。测量结束后，在用户定义的延迟时间之后，才会启动下一次测量。用户必须停止测距才能返回到待机状态，最后的一次测量在停机前完成。

根据以上的测量模式，ST 官方提供了 4 种不同的精度模式，如 1.2.1 表格所示：

精度模式	测量时间预算范围（ms）	测距性能（ms）	典型应用
默认	30	1.2	标准
高精度	200	1.2 精度<±3%	精确测量
长距离	33	2	长距离，只适用于黑暗条件（无红外线）
高速	20	1.2 精度<±5%	高速，精度不优先

表 1.2.1 精度模式

从表格可以看到，针对不同的精度模式，测量时间也是有所区别的，测量时间最快为高速模式，只需 20ms 内就可以采样一次，但精度确存在有±5%的误差范围。而在长距离精度模式下，测距距离能达到 2m，测量时间在 33ms 内，但测量时需在黑暗条件（无红外线）的环境下。所以在实际的应用中，需根据当前的要求去选择合适的精度模式，以达到最佳的测量效果。关于 VL53L0X 精度的详细介绍，请看 VL53L0X 芯片手册文档（模块资料→芯片数据手册→VL53L0X.pdf）5.3 章节。

VL53L0X 简易的工作流程图，如图 1.2.2 所示：

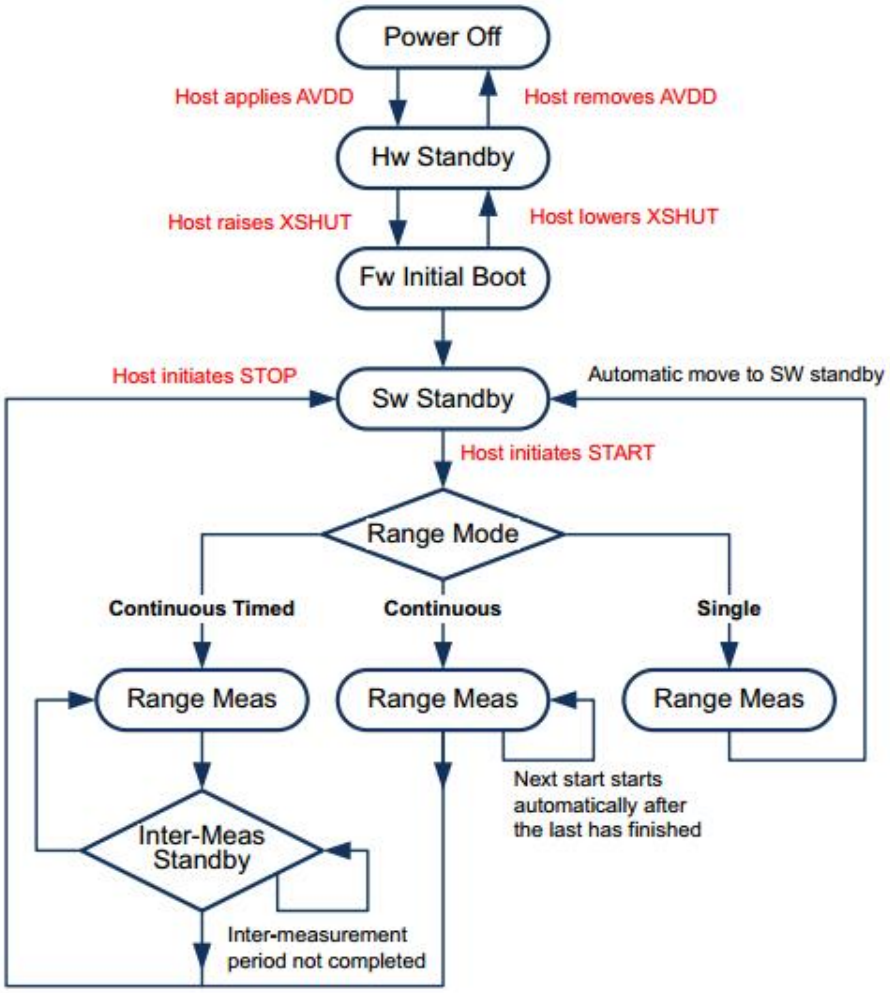

图 1.2.2 VL53L0X 工作流程图

针对测量数据的获取，VL53L0X 提供了轮询和中断两种工作方式，可根据实际情况进行选择。对于在 VL53L0X 实际测量时，可能会出现距离偏差，官方对此也有对 VL53L0X 传感器校准有一个详细的说明。关于测量数据的获取工作方式和校准的具体说明，请看 VL53L0X 芯片手册文档（VL53L0X.pdf）2.7 和 2.3 章节，在这里我们就不做讲解了。

关于 VL53L0X 的介绍，我们就介绍到这。VL53L0X 的详细资料介绍，请参考模块资料 →芯片数据手册→VL53L0X.pdf。

1.3 VL53L0X API 使用说明

经过 1.2 节的介绍，了解到 VL53L0X 工作流程，要对芯片进行使用，ST 并没有直接提供 VL53L0X 寄存器手册，而是提供 VL53L0X 软件 API（应用编程接口）以及完整的技术文档和例程源码，例程源码是 ST 官方的 X-NUCLEO-53L0A1 扩展板基于 STM32F401RE 和 STM32L476RG Nucleo 开发板进行开发的，我们需要将其移植一下才可以用到，官方例程驱动在：模块资料→4，VL53L0X 参考资料→en.X-CUBE-53L0A1.zip，en.X-CUBE-53L0A1.zip 就是官方的例程驱动，代码比较多，不过官方提供了 VL53L0X 软件 API（应用编程接口）文档供大家学习：VL53L0X_API_v1.0.2.4823_externalx.chm，这个文件在 VL53L0X 参考资料文件夹里面，具体位置在：模块资料→4，VL53L0X 参考资料-→VL53L0X_API_v1.0.2.4823_externalx.chm。大家可以阅读这个文件，来熟悉 VL53L0X 驱动库的使用。

官方 VL53L0X 驱动库移植起来，还是比较简单的，主要是实现 VL53L0X_WrByte、

VL53L0X_WrWord 、 VL53L0X_WrDWord 、 VL53L0X_RdByte 、 VL53L0X_RdWord 、 VL53L0X_RdDWord 、 VL53L0X_WriteMulti 、 VL53L0X_ReadMulti 、 VL53L0X_UpdateByte 和 VL53L0X_PollingDelay 的底层驱动函数。具体的细节，我们就不详细介绍了，移植后的驱动代码可以在：模块资料→程序源码→ATK-VL53L0X 模块实验→HARDWARE→VL53L0X 文件夹下找到，其中 core 文件，是 API 的功能应用函数。platform 文件，是我们需要实现的底层驱动函数，及一些宏定义设置。而 demo 文件，则是模块例程测试实验的全部代码。各文件夹文件如图 1.3.1 所示：

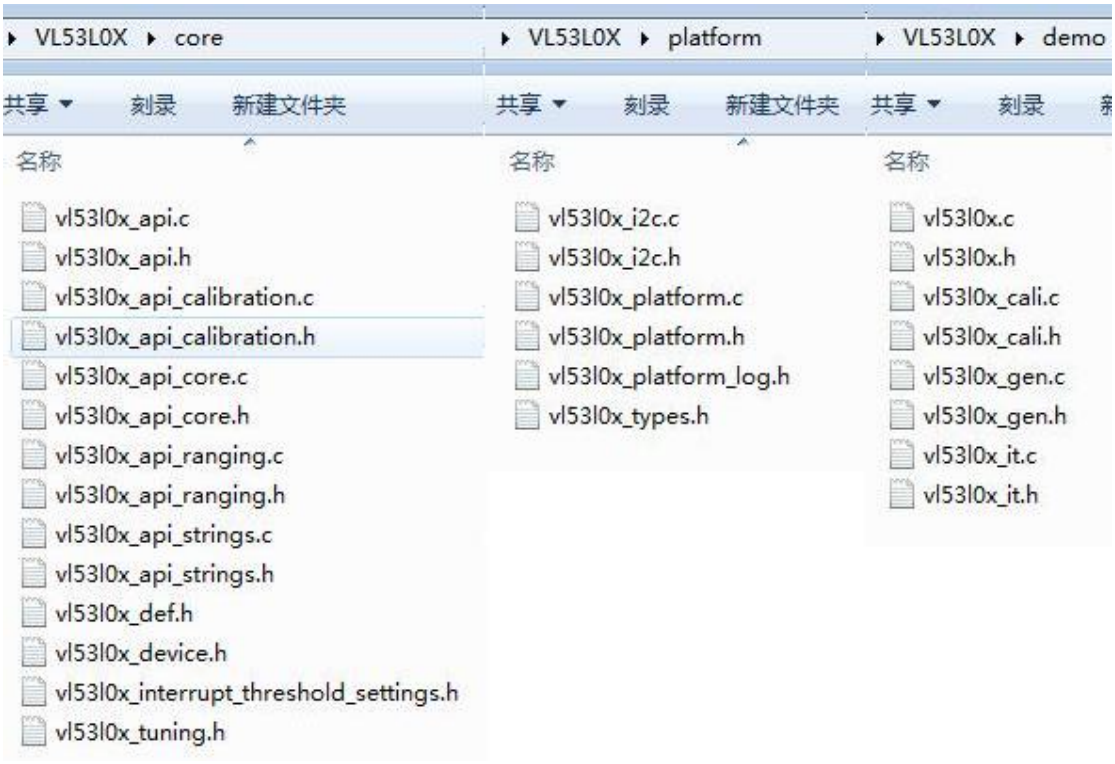

图 1.3.1 文件夹文件

其中，core 文件夹下有四个重点.c 文件，分别是：vl53l0x_api.c、vl53l0x_api_calibration.c、vl53l0x_api_core.c 和 vl53l0x_api_strings.c，这四个 C 文件的关系可以用以下关系图来表示，如图 1.3.2 所示：

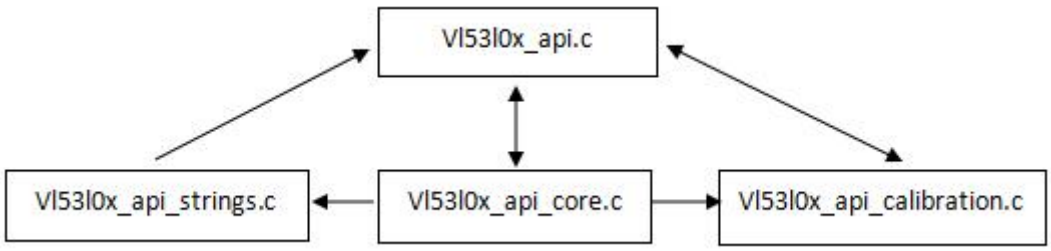

图 1.3.2 文件关系图

vl53l0x_api_core.c 文件，提供有 vl53l0x 传感器的底层操作核心函数。

vl53l0x_api_calibration.c 文件，提供有 vl53l0x 传感器校准操作的底层函数。

vl53l0x_api_strings.c 文件，它实现信息内容的获取，提供有获取 vl53l0x 传感器设备 ID 信息的底层函数，以及提供根据功能函数返回的状态值去获得其状态信息（字符串）的底层函数。

vl53l0x_api.c 文件，则对以上的底层函数进行封装，提供包括系统初始化、测量模式配

置、参数配置、校准功能、状态信息获取、以及中断配置等 API 函数。在一般的使用时，直接调取 vl53l0x_api.c 提供的 API 函数就可以了。

这里我们列出部分常用的 API 函数：

函数	作用
VL53L0X_DataInit()	设备初始化
VL53L0X_StaticInit()	参数值恢复默认
VL53L0X_SetDeviceAddress()[1]	设置设备 IIC 地址
VL53L0X_SetDeviceMode()	设置测量工作模式
VL53L0X_PerformSingleMeasurement()	执行单次测量
VL53L0X_StartMeasurement()	启动测量
VL53L0X_StopMeasurement()	暂停测量
VL53L0X_GetRangingMeasurementData()	获取测量数据
VL53L0X_PerformRefCalibration()	执行参考校准
VL53L0X_PerformXTalkCalibration()	执行串扰校准
VL53L0X_PerformOffsetCalibration()	执行偏移校准
VL53L0X_PerformRefSpadManagement()	执行参考 SPAD 管理
VL53L0X_SetGpioConfig()	设置 GPIO 引脚工作模式中断配置
VL53L0X_SetInterruptThresholds()	设置中断触发距离上下限值
VL53L0X_ClearInterruptMask()	清除中断标志位
VL53L0X_SetMeasurementTimingBudgetMicroSeconds()	设定完整测距的最长时间
VL53L0X_SetVcselPulsePeriod()	设定 VCSEL 脉冲周期
VL53L0X_SetReferenceSpads()	设置 SPAD 参考值
VL53L0X_SetRefCalibration()	设置参考校准值
VL53L0X_SetOffsetCalibrationDataMicroMeter()	设置偏移校准值
VL53L0X_SetXTalkCompensationRateMegaCps()	设置串扰校准值
VL53L0X_GetRangeStatusString()	根据 Range 状态值读取范围状态字符串
VL53L0X_GetPalErrorString()	根据 PAL 状态值读取错误字符串

表 1.3.3 常用的 API 函数

以上表格就是常用的 API 函数，这里我们将挑部分 API 函数简单介绍下：

1、VL53L0X_DataInit()函数

该函数实现传感器初始化，函数为 VL53L0X_DataInit(VL53L0X_DEV Dev)，入口参数类型为 VL53L0X_DEV 结构体，结构体变量为设置传感器的 I2C 地址等重要信息。在失能传感器设备，到再重新使能设备，需调用该函数进行初始化。

2、VL53L0X_StaticInit()函数

该函数实现传感器寄存器值恢复默认初值和工作状态切换为待机状态。函数为 VL53L0X_StaticInit(VL53L0X_DEV Dev)，入口参数为 VL53L0X_DEV 结构体变量，VL53L0X_DEV 结构体变量为设置传感器的 I2C 地址等重要信息。该函数在初始化时进行调用。

3、VL53L0X_SetDeviceAddress()函数

该函数实现修改传感器设备的 I2C 地址，函数为 VL53L0X_SetDeviceAddress(VL53L0X_DEV Dev,uint8_t DeviceAddress)，入口参数为 VL53L0X_DEV 结构体变量和 DeviceAddress 变量。

VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，DeviceAddress 变量为新设置的 I2C 地址。（注意：重新使能设备后，I2C 地址为会恢复为默认的 0x52 地址）。若修改设备 I2C 地址，需在使能设备后进行调用。设置的 I2C 地址必须为偶数，否则会出现通讯出错。

4、VL53L0X_SetDeviceMode()函数

该函数实现传感器测量的工作模式配置，函数为 VL53L0X_SetDeviceMode(VL53L0X_DEV Dev,VL53L0X_DeviceModes DeviceMode)，入口参数为 VL53L0X_DEV 结构体类型变量和 VL53L0X_DeviceModes 类型变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，VL53L0X_DeviceModes 类型变量为设置的工作模式，其中包含 SINGLE_RANGING 单次测量、CONTINUOUS_RANGING 连续测量和 TIMED_RANGING 定时测量等其他模式。该函数在测量前初始化进行调用。

5、VL53L0X_PerformSingleMeasurement()函数

该函现实现启动传感器一次测量，函数为 VL53L0X_PerformSingleRangingMeasurement (VL53L0X_DEV Dev,VL53L0X_RangingMeasurementData_t *pRangingMeasurementData)。入口参数为 VL53L0X_DEV 结构体类型变量和 VL53L0X_RangingMeasurementData_t 结构体类型变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，VL53L0X_RangingMeasurement Data_t 结构体变量为返回测量距离的数据和测量范围状态等数据。该函数一般在传感器工作在单次测量模式下进行调用。

6、VL53L0X_StartMeasurement()和 VL53L0X_StopMeasurement()函数

该函数分别实现了传感器启动测量和暂停测量，函数为 VL53L0X_StartMeasurement (VL53L0X_DEV Dev) 和 VL53L0X_StopMeasurement(VL53L0X_DEV Dev)，入口参数都为 VL53L0X_DEV 结构体类型变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，这两个函数一般在测量时进行使用。

7、VL53L0X_GetRangingMeasurementData()函数

该函数实现获取传感器采样的测量数据，函数为 VL53L0X_PerformSingleRanging Measurement(VL53L0X_DEV Dev,VL53L0X_Ranging MeasurementData_t *pRangingMeasure mentData)。入口参数为 VL53L0X_DEV 结构体类型变量和 VL53L0X_RangingMeasurementData_t 结构体类型变量，VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，VL53L0X_RangingMeasurementData_t 结构体变量为返回测量距离的数据和测量范围状态等数据。该函数一般在传感器工作在连续测量模式下进行使用。

8、VL53L0X_SetGpioConfig()函数

该函数实现测量中断模式的配置功能，函数为 VL53L0X_SetGpioConfig(VL53L0X_DEV Dev, uint8_t Pin,VL53L0X_DeviceModes DeviceMode, VL53L0X_GpioFunctionality Functionality, VL53L0X_InterruptPolarity Polarity)。入口参数为 VL53L0X_DEV 结构体类型变量、Pin 变量、VL53L0X_DeviceModes 类型变量，VL53L0X_GpioFunctionality 类型变量和 VL53L0X_InterruptPolarity 类型变量。其中 VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，VL53L0X_DeviceModes 类型变量为设置的测量工作模式（单次测量、连续测量、定时测量等），Pin 变量为设置 GPIO 引脚口（使用时默认配置为 0）、VL53L0X_DeviceModes 类型变量为设置测量触发中断模式的，包含 CROSSED_LOW（采样距离<下限值）、CROSSED_HIGH（采样距离>上限值）以及 CROSSED_OUT（采样距离<下限值或采样距离>上限值）模式，VL53L0X_InterruptPolarity 类型变量为设置引脚的触发模式，包含上升沿和下降沿。函数一般在传感器工作在连续测量模式初始化配置时进行使用。

9、VL53L0X_SetInterruptThresholds()函数

该函数实现测量中断模式上下限距离值的设置，函数为 VL53L0X_SetInterruptThresholds (VL53L0X_DEV Dev,VL53L0X_DeviceModes DeviceMode, FixPoint1616_t ThresholdLow,FixPoint

1616_t ThresholdHigh)。入口参数 VL53L0X_DEV 结构体类型变量、VL53L0X_DeviceModes 类型变量、ThresholdLow 和 ThresholdHigh 变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，VL53L0X_DeviceModes 类型变量为设置的测量工作模式（单次测量、连续测量、定时测量等），而 ThresholdLow 和 ThresholdHigh 为设置中断触发的下限距离值和上限距离值。该函数一般在传感器工作在连续测量模式初始化配置时进行使用。

10、VL53L0X_ClearInterruptMask()函数

该函数实现清除测量中断模式的中断标志位，函数为 VL53L0X_ClearInterruptMask(VL53L0X_DEV Dev,uint32_t InterruptMask)。入口参数为 VL53L0X_DEV 结构体类型变量，InterruptMask 变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，InterruptMask 变量为设置中断清除，默认设置为 0。该函数在传感器触发中断后时进行使用。

11、VL53L0X_SetMeasurementTimingBudgetMicroSeconds()函数

该函数实现配置测量工作模式的周期采样时间的功能，函数为 VL53L0X_SetMeasurementTimingBudgetMicroSeconds(VL53L0X_DEV Dev, uint32_t MeasurementTimingBudgetMicroSeconds)。入口参数为 VL53L0X_DEV 结构体类型变量、MeasurementTimingBudgetMicroSeconds 变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，MeasurementTimingBudgetMicroSeconds 变量为设置测量工作模式的周期采样时间，单位为 ms。该函数在工作模式初始化配置时进行使用。

12、VL53L0X_PerformOffsetCalibration()函数

该函数实现距离偏移的校准的功能，函数为 VL53L0X_PerformOffsetCalibration VL53L0X_DEV Dev,FixPoint1616_t CalDistanceMilliMeter, int32_t *pOffsetMicroMeter)。入口参数为 VL53L0X_DEV 结构体类型变量、CalDistanceMilliMeter 变量和 pOffsetMicroMeter 指针变量。VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，CalDistanceMilliMeter 变量为设置校准时与传感器设备的相对距离值，单位为 mm，pOffsetMicroMeter 指针变量为相对距离校准后传感器返回的修正距离变量的指针。该函数在校准时进行使用。

13、VL53L0X_SetOffsetCalibrationDataMicroMeter()函数

该函数实现设置距离补偿值的功能，函数为 VL53L0X_SetOffsetCalibrationDataMicroMeter(VL53L0X_DEV Dev, int32_t OffsetCalibrationDataMicroMeter)。入口参数为 VL53L0X_DEV 结构体类型变量和 OffsetCalibrationDataMicroMeter 变量，VL53L0X_DEV 结构体变量为设置传感器 I2C 地址等重要信息，OffsetCalibrationDataMicroMeter 变量为设置的距离补偿值。该函数在补偿时进行使用。

以上就是部分的 API 函数的说明，关于校准功能及更多的 API 函数的详细说明及应用，请查看官方 VL53L0X 软件 API（应用编程接口）文档：VL53L0X_API_v1.0.2.4823_externalx.chm。文档路径：模块资料→ VL53L0X 参考资料→ VL53L0X_API_v1.0.2.4823_externalx.chm。

2、硬件连接

2.1 功能介绍

本实验功能简介：本实验用于测试 ATK-VL53L0X 激光测距模块，共包括 3 项测试：

1，校准测试-通过 KEY_UP 按键进入此项测试。该功能实现对传感器测量误差的校准，进入测试后，会看到 LCD 屏幕提示，提示需要一个白色的目标（可以是白纸），且白色的目标需与模块保持在 100mm 的距离，确认目标和距离无误后，这时可按下 LCD 屏幕提示的 KEY1 按键执行校准操作，若不想执行校准可按下 KEY_UP 返回主菜单。按下 KEY1 按键后，LCD 屏幕会显示开始校准，这时校准需要一定的时间，若校准成功，则会将校准的数据存入到 24C02 上。否则，将不会进行数据保存，校准过程结束后，会自动返回主菜单页面。

2，普通测量测试-此测试是使用 Single ranging（单次测量）工作模式，采用轮询方式，读取测量数据，通过 KEY1 按键进入此项测试。此项测试包含 4 个精度模式子项，其中包括：默认、高精度、长距离、高速。通过屏幕提示的 KEY1 按键操作，进行精度模式的切换。选择好精度模式后，按下 KEY0 按键可进入此模式测试。进入测试后，若之前成功校准过，会将保存在 24C02 的校准数据写进模块上，测量的数据通过轮询方式采集。数据采集结束后，数据显示在屏幕上、同时也打印在串口调试助手上。单击按下 KEY_UP，可返回精度模式选项页面，双击按下，则返回主菜单页面。

3，中断测量测试-此测试是使用 Continuous ranging（连续测量）工作模式，采用中断方式，读取测量数据。通过 KEY0 按键进入此项测试。此项测试包含 4 个精度模式子项，其中包括：默认、高精度、长距离、高速。通过 LCD 屏幕提示的 KEY1 按键操作，进行精度模式的切换。选择好精度模式后，按下 KEY0 按键可进入此模式测试。进入测试后，若之前成功校准过，会将保存在 24C02 的校准数据写进模块上。在 LCD 屏幕上，会显示距离上下限阀值，当测出的距离在上下限阀值范围之间，模块不触发测距工作，中断没有输出，若在阀值外，则启动测距工作，测量结束会触发一次中断，通过中断读取测量数据，距离数据会显示在屏幕上，同时也会打印在串口调试助手上。单击按下 KEY_UP，可返回精度模式选择页面，双击按下，则返回到主菜单页面。

2.2 硬件准备资源

本实验所需要的硬件资源如下

1，ALIENTEK 探索者 STM32F407 开发板 1 个

2，TFTLCD 模块

3，ATK-VL53L0X 模块 1 个

4，USB 线一条（用于供电和模块与电脑串口调试助手通信）

2.3 模块与开发板连接

ATK-VL53L0X 模块可直接与 ALIENTEKT 探索者 STM32F407 开发板板载的 ATK 模块接口（ATK MODULE）进行连接，ATK MODULE 与 MCU 连接原理图如图 2.3.1 所示：

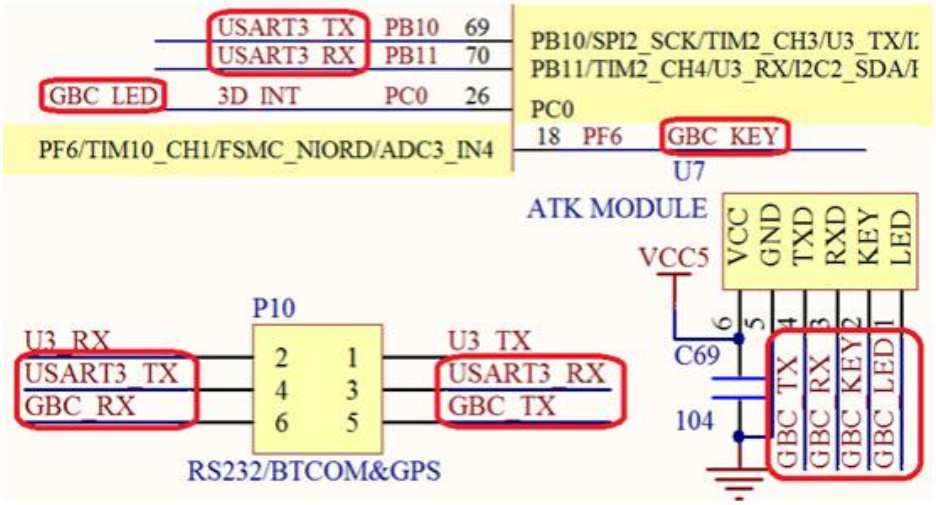

图 2.3.1 ATK-MODULE 接口与 MCU 连接关系

从上图看出，ATK MODULE 接口，使用时必须将 P10 的 USART3_TX（PB10）和 GBC_RX 以及 USART3_RX（PB11）和 GBC_TX 连接，才能完成和 STM32 的连接。探索者 F407 只需要用跳线帽去短接就可以了，连接好后，探索者 F407 开发板与 ATK-VL53L0X 模块的连接关系如表 2.3.2 所示：

ATK-VL53L0X 激光测距模块与开发板连接关系						
ATK-VL53L0X 模块	VCC	GND	SCL	SDA	INT	XSH
探索者 STM32F407 开发板	5V	GND	PB11	PB10	PF6	PC0

表 2.3.2 ATK-VL53L0X 模块与探索者 STM32F407 开发板连接关系图

ATK-VL53L0X 模块插入到开发板的 ATK MODULE 接口，如图 2.3.3 所示：

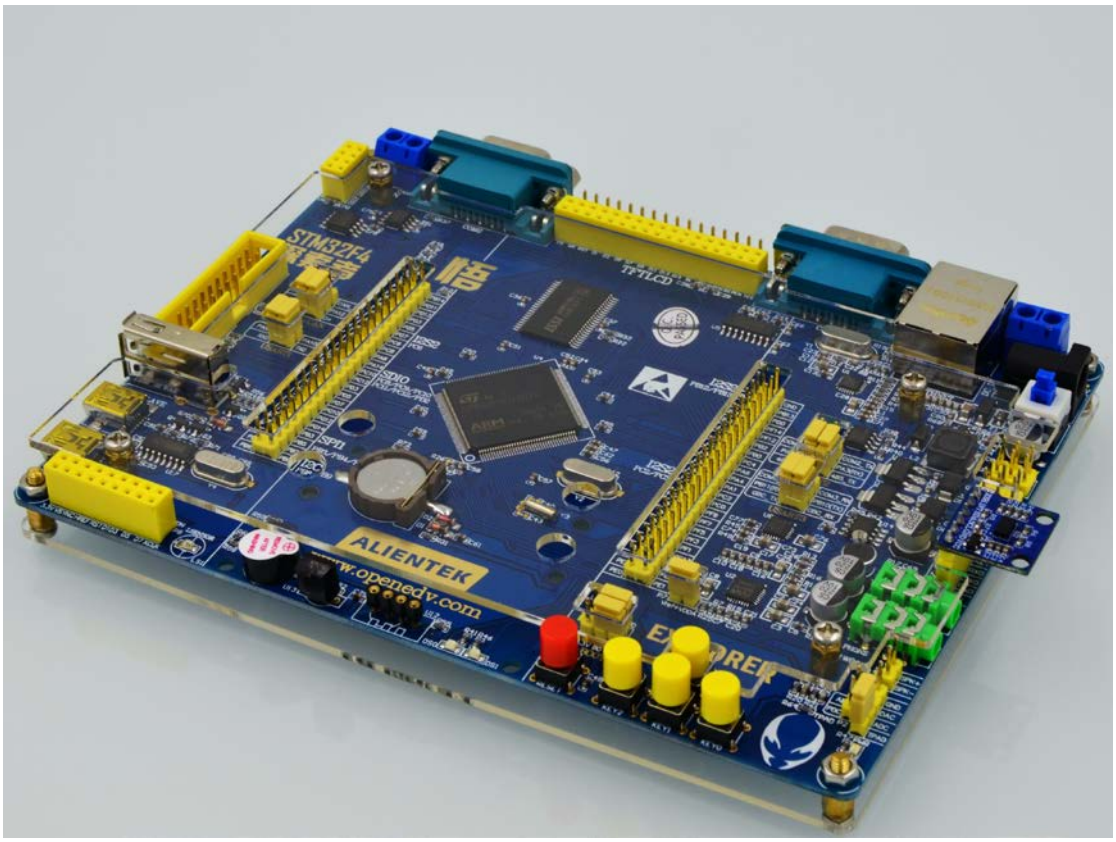

图 2.2.4 ATK-VL53L0X 模块与开发板对接实物图

3、软件实现

本实验在探索者 STM32F407 开发板的 IIC 实验基础上进行修改，在 HARDWARE 文件夹内新建了 VL53L0X 文件夹，VL53L0X 文件夹内新建 core 文件夹，存放官方移植的 VL53L0X API 驱动文件，vl53l0x_api.c、vl53l0x_api_calibration.c、vl53l0x_api_core.c 等文件。创建 platform 文件夹，存放 VL53L0X 底层接口驱动文件，vl53l0x_i2c.c、vl53l0x_platform.c 等文件。最后创建 demo 文件夹，创建 vl53l0x.c、vl53l0x_it.c、vl53l0x_cali.c、vl53l0x_gen.c 等文件

在工程目录添加 VL53L0X 和 VL53L0X_demo 分组，将 core 和 platform 文件夹内的 C 文件添加到 VL53L0X 分组内，接着将 demo 文件夹内的 C 文件添加到 VL53L0X_demo 分组内，最后添加 core、platform 和 demo 文件夹到头文件包含路径。最终的工程如图 3.1 所示：

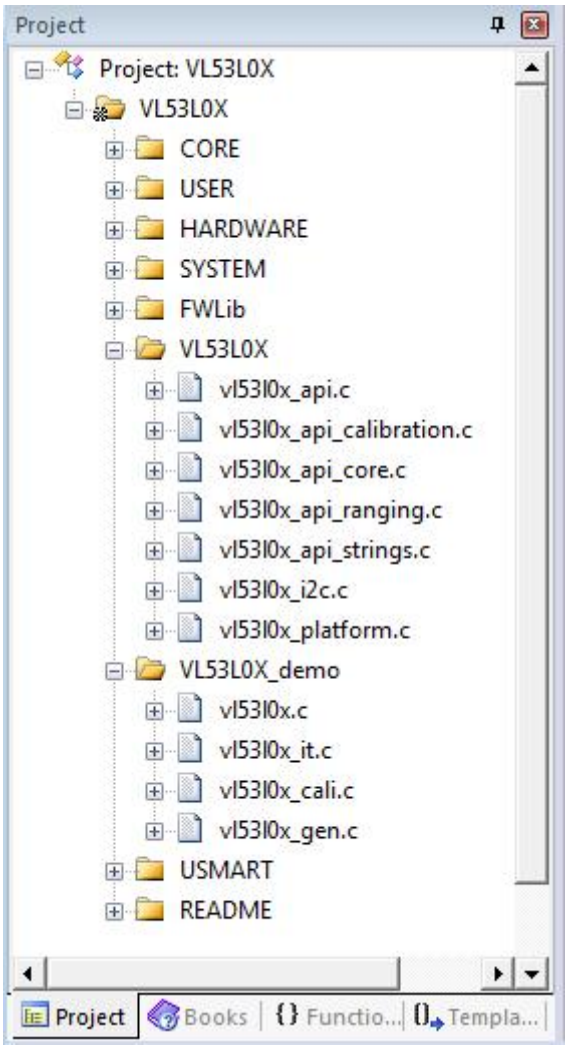

图 3.1 ATK-VL53L0X 模块测试实验工程截图

本例程由于代码量较多，我们仅对部分代码（vl53l0x_platform.c、vl53l0x.c、vl53l0x_it.c、vl53l0x_cali.c、vl53l0x_gen.c），以及 main 函数进行讲解。

（1）vl53l0x_platform.c，该文件为底层驱动函数，它调用了 vl53l0x_i2c.c 文件的 iic 读写数据函数，实现了对 VL53L0X 读写操作,同时文件也实现对底层延时的功能，具体代码如下：

```
//VL53L0X 连续写数据
//Dev:设备 I2C 参数结构体
```

```
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//          其他:错误
VL53L0X_Error VL53L0X_WriteMulti(VL53L0X_DEV Dev, uint8_t index,
                                 uint8_t *pdata,uint32_t count)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int = 0;
    uint8_t deviceAddress;
    if(count >=VL53L0X_MAX_I2C_XFER_SIZE)
    {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_write_multi(deviceAddress, index, pdata, count);
    if(status_int !=0)
       Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 连续读数据
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//          其他:错误
VL53L0X_Error VL53L0X_ReadMulti(VL53L0X_DEV Dev, uint8_t index,
                                uint8_t *pdata,uint32_t count)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    if(count >=VL53L0X_MAX_I2C_XFER_SIZE)
    {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_read_multi(deviceAddress, index, pdata, count);
    if(status_int!=0)
       Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
```

```
}

//VL53L0X 写单字节寄存器
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//        其他:失败
VL53L0X_Error VL53L0X_WrByte(VL53L0X_DEV Dev, uint8_t index, uint8_t data)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_write_byte(deviceAddress,index,data);
    if(status_int!=0)
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 写字（2 字节）寄存器
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//        其他:失败
VL53L0X_Error VL53L0X_WrWord(VL53L0X_DEV Dev, uint8_t index, uint16_t data)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_write_word(deviceAddress,index,data);
    if(status_int!=0)
      Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 写双字（4 字节）寄存器
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
```

```
//count:长度
//返回值: 0:成功
//          其他:失败
VL53L0X_Error VL53L0X_WrDWord(VL53L0X_DEV Dev, uint8_t index, uint32_t data)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_write_dword(deviceAddress, index, data);
    if (status_int != 0)
      Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 威胁安全更新(读/修改/写)单字节寄存器
//Dev:设备 I2C 参数结构体
//index:偏移地址
//AndData:8 位与数据
//OrData:8 位或数据
//返回值: 0:成功
//          其他:错误
VL53L0X_Error VL53L0X_UpdateByte(VL53L0X_DEV Dev, uint8_t index,
                                 uint8_t AndData, uint8_t OrData)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    uint8_t data;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_read_byte(deviceAddress,index,&data);
    if(status_int!=0)
       Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    if(Status == VL53L0X_ERROR_NONE)
    {
       data = (data & AndData) | OrData;
       status_int = VL53L0X_write_byte(deviceAddress, index,data);
       if(status_int !=0)
            Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    }
    return Status;
}

//VL53L0X 读单字节寄存器
```

```
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//        其他:错误
VL53L0X_Error VL53L0X_RdByte(VL53L0X_DEV Dev, uint8_t index, uint8_t *data)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_read_byte(deviceAddress, index, data);
    if(status_int !=0)
    Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 读字（2 字节）寄存器
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//        其他:错误
VL53L0X_Error VL53L0X_RdWord(VL53L0X_DEV Dev, uint8_t index, uint16_t *data)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_read_word(deviceAddress, index, data);
    if(status_int !=0)
       Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 读双字（4 字节）寄存器
//Dev:设备 I2C 参数结构体
//index:偏移地址
//pdata:数据指针
//count:长度
//返回值: 0:成功
//        其他:错误
```

```
VL53L0X_Error    VL53L0X_RdDWord(VL53L0X_DEV Dev, uint8_t index, uint32_t *data)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;
    deviceAddress = Dev->I2cDevAddr;
    status_int = VL53L0X_read_dword(deviceAddress, index, data);
    if (status_int != 0)
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;
    return Status;
}

//VL53L0X 底层延时函数
//Dev:设备 I2C 参数结构体
//返回值: 0:成功
//        其他:错误
#define VL53L0X_POLLINGDELAY_LOOPNB    250
VL53L0X_Error VL53L0X_PollingDelay(VL53L0X_DEV Dev)
{
    VL53L0X_Error status = VL53L0X_ERROR_NONE;
    volatile uint32_t i;
    for(i=0;i<VL53L0X_POLLINGDELAY_LOOPNB;i++){
        //Do nothing
    }
    return status;
}
```

以上就是 vl53l0x_platform.c 文件的全部代码。都是 VL53L0X 底层操作的函数，比较简单，这里我们只挑 VL53L0X_WriteMult 和 VL53L0X_ReadMulti 两个函数说下，对于其他的函数这里我们就不做讲解了。VL53L0X_WriteMulti()函数，该函数用于指定器件和地址，实现连续写数据，被 VL53L0X 的 API 函数所调用。而 VL53L0X_ReadMulti()函数，该函数用于指定器件和地址，实现连续读数据，同样也是被 VL53L0X 的 API 函数所调用。

（2）vl53l0x.c 文件，该文件主要实现 vl53l0x 设备初始化、I2C 地址设置等函数。代码比较多，这里就不全部列出来了，仅介绍几个重要的函数。

首先是：print_pal_error 函数，该函数代码如下：

```
//API 错误信息打印
//Status：详情看 VL53L0X_Error 参数的定义
void print_pal_error(VL53L0X_Error Status)
{
    char buf[VL53L0X_MAX_STRING_LENGTH];

    VL53L0X_GetPalErrorString(Status,buf);//根据 Status 状态获取错误信息字符串

    printf("API Status: %i : %s\r\n",Status, buf);//打印状态和错误信息
}
```

该函数调用了 VL53L0X_GetPalErrorString() API 函数，该函数可根据传入的 Status 状态值，获取状态值的信息字符串。根据字符串分析，可以快速定位错误问题点，方便调试。

然后我们看下 vl53l0x_Addr_set 函数，该函数代码如下：

```
//配置 VL53L0X 设备 I2C 地址
//dev:设备 I2C 参数结构体
//newaddr:设备新 I2C 地址
VL53L0X_Error vl53l0x_Addr_set(VL53L0X_Dev_t *dev,uint8_t newaddr)
{
    uint16_t Id;
    uint8_t FinalAddress;
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    u8 sta=0x00;

    FinalAddress = newaddr;
    if(FinalAddress==dev->I2cDevAddr)//新设备 I2C 地址与旧地址一致,直接退出
        return VL53L0X_ERROR_NONE;
    //在进行第一个寄存器访问之前设置 I2C 标准模式(400Khz)
    Status = VL53L0X_WrByte(dev,0x88,0x00);
    if(Status!=VL53L0X_ERROR_NONE)
    {
        sta=0x01;//设置 I2C 标准模式出错
        goto set_error;
    }
    //尝试使用默认的 0x52 地址读取一个寄存器
    Status = VL53L0X_RdWord(dev, VL53L0X_REG_IDENTIFICATION_MODEL_ID, &Id);
    if(Status!=VL53L0X_ERROR_NONE)
    {
        sta=0x02;//读取寄存器出错
        goto set_error;
    }
    if(Id == 0xEEAA)
    {
        //设置设备新的 I2C 地址
        Status = VL53L0X_SetDeviceAddress(dev,FinalAddress);
        if(Status!=VL53L0X_ERROR_NONE)
        {
            sta=0x03;//设置 I2C 地址出错
            goto set_error;
        }
        //修改参数结构体的 I2C 地址
        dev->I2cDevAddr = FinalAddress;
        //检查新的 I2C 地址读写是否正常
        Status = VL53L0X_RdWord(dev, VL53L0X_REG_IDENTIFICATION_MODEL_ID, &Id);
        if(Status!=VL53L0X_ERROR_NONE)
```

```
            {
                sta=0x04;//新 I2C 地址读写出错
                goto set_error;
            }
        }
        set_error:
        if(Status!=VL53L0X_ERROR_NONE)
        {
            print_pal_error(Status);//打印错误信息
        }
        if(sta!=0)
          printf("sta:0x%x\r\n",sta);
        return Status;
    }
```

该函数实现对 vl53l0x 设备 I2C 地址的修改。设备初次上电默认 I2C 地址为 0X52，在修改前，我们先用旧地址读取寄存器的值，读取寄存器成功，保证通讯正常了，然后再调用 VL53L0X_SetDeviceAddress() API 函数设置新的地址，最后用新设置的 I2C 地址去读取寄存器的值，若读取寄存器成功，则表示 vl53l0x 新的 I2C 地址修改成功。

最后看下 vl53l0x_init 函数，该函数代码如下：

```
//初始化 vl53l0x
//dev:设备 I2C 参数结构体
VL53L0X_Error vl53l0x_init(VL53L0X_Dev_t *dev)
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_Dev_t *pMyDevice = dev;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC,ENABLE);
                                    //先使能外设 IO PORTC 时钟
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;//端口配置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;//普通输出模式
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;//推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;//100MHz
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//上拉
    GPIO_Init(GPIOC, &GPIO_InitStructure);//初始化

    pMyDevice->I2cDevAddr = VL53L0X_Addr;//I2C 地址(上电默认 0x52)
    pMyDevice->comms_type = 1;              //I2C 通信模式
    pMyDevice->comms_speed_khz = 400;       //I2C 通信速率

    VL53L0X_i2c_init();//初始化 IIC 总线
    VL53L0X_Xshut=0;//失能 VL53L0X
    delay_ms(30);
    VL53L0X_Xshut=1;//使能 VL53L0X,让传感器处于工作
```

```
    delay_ms(30);

    vl53l0x_Addr_set(pMyDevice,0x54);//设置 VL53L0X 传感器 I2C 地址
    if(Status!=VL53L0X_ERROR_NONE) goto error;
    Status = VL53L0X_DataInit(pMyDevice);//设备初始化
    if(Status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
    Status = VL53L0X_GetDeviceInfo(pMyDevice,&vl53l0x_dev_info);//获取设备 ID 信息
    if(Status!=VL53L0X_ERROR_NONE) goto error;

    AT24CXX_Read(0,(u8*)&Vl53l0x_data,sizeof(_vl53l0x_adjust));
      //读取 24c02 保存的校准数据,若已校准  Vl53l0x_data.adjustok==0xAA
    if(Vl53l0x_data.adjustok==0xAA)//已校准
      AjustOK=1;
    else //没校准
      AjustOK=0;

    error:
    if(Status!=VL53L0X_ERROR_NONE)
    {
        print_pal_error(Status);//打印错误信息
        return Status;
    }
    return Status;
}
```

该函数为 vl53l0x 设备初始化函数，函数入口参数为 dev，该参数用来保存 I2C 的设备信息，如地址和速度大小。在函数中，先对用到的引脚进行时钟等功能配置初始化，然后保存设备默认 I2C 地址（0X52）和其他的参数值到 pMyDevice 结构体中，接着初始化 VL53L0X 设备 I2C 引脚和使能 vl53l0x 设备，让设备处于工作状态，然后修改设备 I2C 地址，修改成功后，调用设备初始化 API 函数对 VL53L0X 设备进行初始化，等初始化结束后，调用信息获取 API 函数获取其设备 ID 信息，将 ID 信息保存到 vl53l0x_dev_info 信息结构体中。由于测试实验使用到 24C02 保存传感器校准的数据，所以在初始化最后会对 24C02 进行校准数据的读取，读取到标志为 0XAA，表示传感器上次已校准过，并且标志 AjustOK 变量为 1。

（3）vl53l0x_cali.c，该文件主要实现传感器的校准，该文件下代码比较多，这里就不全部列出来了，仅介绍传感器校准函数 vl53l0x_adjust()。

vl53l0x_adjust()函数，代码如下：（注：由于代码较多，省略了部分非重要代码）

```
_vl53l0x_adjust Vl53l0x_adjust; //校准数据 24c02 写缓存区
                                (用于在校准模式下校准数据写 24c02)
_vl53l0x_adjust Vl53l0x_data; //校准数据 24c02 读缓存区
                              （用于系统初始化时向 24C02 读取数据）
//VL53L0X 校准函数
//dev:设备 I2C 参数结构体
VL53L0X_Error vl53l0x_adjust(VL53L0X_Dev_t *dev)
{
```

```
VL53L0X_DeviceError Status = VL53L0X_ERROR_NONE;
uint32_t refSpadCount = 7;
uint8_t    isApertureSpads = 0;
uint32_t XTalkCalDistance = 100;
uint32_t XTalkCompensationRateMegaCps;
uint32_t CalDistanceMilliMeter = 100<<16;
int32_t    OffsetMicroMeter = 30000;
uint8_t VhvSettings = 23;
uint8_t PhaseCal = 1;
u8 i=0;

VL53L0X_StaticInit(dev);//数值恢复默认,传感器处于空闲状态
LED1=0;
/******************SPADS 校准************************/
spads:
delay_ms(10);
printf("The SPADS Calibration Start...\r\n");
Status = VL53L0X_PerformRefSpadManagement(dev,&refSpadCount,
                                          &isApertureSpads);//执行参考 Spad 管理
if(Status == VL53L0X_ERROR_NONE)
{
    //保存校准值
    Vl53l0x_adjust.refSpadCount = refSpadCount;
    Vl53l0x_adjust.isApertureSpads = isApertureSpads;
}else{}
/******************设备参考校准**********************/
ref:
delay_ms(10);
printf("The Ref Calibration Start...\r\n");
Status = VL53L0X_PerformRefCalibration(dev,&VhvSettings,&PhaseCal);//Ref 参考校准
if(Status == VL53L0X_ERROR_NONE)
{
    //保存校准值
    Vl53l0x_adjust.VhvSettings = VhvSettings;
    Vl53l0x_adjust.PhaseCal = PhaseCal;
}else{}
/******************偏移校准**************************/
offset:
delay_ms(10);
printf("Offset Calibration:need a white target,in black space,and the
       distance keep 100mm!\r\n");
printf("The Offset Calibration Start...\r\n");
Status= VL53L0X_PerformOffsetCalibration(dev,CalDistanceMilliMeter,
                                         &OffsetMicroMeter);//偏移校准
```

```
        if(Status == VL53L0X_ERROR_NONE)
        {
            //保存校准值
            Vl53l0x_adjust.CalDistanceMilliMeter = CalDistanceMilliMeter;
            Vl53l0x_adjust.OffsetMicroMeter = OffsetMicroMeter;
        }else{}
        /******************串扰校准**********************/
        xtalk:
        delay_ms(20);
        printf("Cross Talk Calibration:need a grey target\r\n");
        printf("The Cross Talk Calibration Start...\r\n");
        Status = VL53L0X_PerformXTalkCalibration(dev,XTalkCalDistance,
                                                 &XTalkCompensationRateMegaCps);//串扰校准
        if(Status == VL53L0X_ERROR_NONE)
        {
            //保存校准值
            Vl53l0x_adjust.XTalkCalDistance = XTalkCalDistance;
            Vl53l0x_adjust.XTalkCompensationRateMegaCps=
            XTalkCompensationRateMegaCps;
        }else{}
        LED1=1;
        printf("All the Calibration has Finished!\r\n");
        printf("Calibration is successful!!\r\n");

        Vl53l0x_adjust.adjustok = 0xAA;//校准成功
        //将校准数据保存到 24c02
        AT24CXX_Write(0,(u8*)&Vl53l0x_adjust,sizeof(_vl53l0x_adjust));
        //将校准数据复制到 Vl53l0x_data 结构体
        memcpy(&Vl53l0x_data,&Vl53l0x_adjust,sizeof(_vl53l0x_adjust));
        return Status;
    }
```

该函数为 vl53l0x 传感器校准函数，实现对传感器的 SPAD（接收端）、Ref 参考设备、距离偏移、以及串扰进行校准。这里我们定义了 Vl53l0x_adjust 校准数据写缓冲区和 Vl53l0x_data 校准数据读缓冲区的全局变量结构体，将每项校准后的数据保存在 Vl53l0x_adjust 结构体变量中，同时标志 Vl53l0x_adjust.adjustok 标志位为 0XAA，以表示校准成功。校准的数据会保存在 24C02 上，同时也将当前的校准数据同步更新到 Vl53l0x_data 校准数据读缓冲区中。

（4）vl53l0x_gen.c 文件，该文件主要实现单次测量、精度模式配置等函数，文件代码比较多，这里就不全部列出来了，仅介绍几个重要的函数。

vl53l0x_set_mode()函数，该函数代码如下：

```
//VL53L0X 单次测量精度模式配置
//dev:设备 I2C 参数结构体
//mode: 0:默认;1:高精度;2:长距离;3:高速
VL53L0X_Error vl53l0x_set_mode(VL53L0X_Dev_t *dev,u8 mode)
```

```
{
    VL53L0X_Error status = VL53L0X_ERROR_NONE;
    uint8_t VhvSettings;
    uint8_t PhaseCal;
    uint32_t refSpadCount;
    uint8_t isApertureSpads;

    vl53l0x_reset(dev);//复位 vl53l0x(频繁切换精度模式容易导致采集距离数据不准，
                       需加上这一代码)
    status = VL53L0X_StaticInit(dev);//参数恢复默认值
    if(AjustOK!=0)//已校准好了,写入校准值
    {
        status= VL53L0X_SetReferenceSpads(dev,Vl53l0x_data.refSpadCount,
                                    Vl53l0x_data.isApertureSpads);//设定 Spads 校准值
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status= VL53L0X_SetRefCalibration(dev,Vl53l0x_data.VhvSettings,
                                          Vl53l0x_data.PhaseCal);//设定 Ref 校准值
        if(status!=VL53L0X_ERROR_NONE) goto error;
         delay_ms(2);
         status=VL53L0X_SetOffsetCalibrationDataMicroMeter
                              (dev,Vl53l0x_data.OffsetMicroMeter);//设定偏移校准值
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status=VL53L0X_SetXTalkCompensationRateMegaCps
                (dev,Vl53l0x_data.XTalkCompensationRateMegaCps);//设定串扰校准值
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
    }else//没校准
    {
        status = VL53L0X_PerformRefCalibration(dev, &VhvSettings,
                                               &PhaseCal);//Ref 参考校准
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status = VL53L0X_PerformRefSpadManagement(dev, &refSpadCount,
                                          &isApertureSpads);//执行参考 SPAD 管理
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
    }
    status = VL53L0X_SetDeviceMode(dev,
                        VL53L0X_DEVICEMODE_SINGLE_RANGING);//使能单次测量模式
    if(status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
    status = VL53L0X_SetLimitCheckEnable(dev,
```

```
                VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,1);//使能 SIGMA 范围检查
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status = VL53L0X_SetLimitCheckEnable(dev,
         VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,1);//使能信号速率范围检查
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status = VL53L0X_SetLimitCheckValue(dev,
         VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,Mode_data[mode].sigmaLimit);
                                                        //设定 SIGMA 范围
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status= VL53L0X_SetLimitCheckValue(dev,
                VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
                        Mode_data[mode].signalLimit);//设定信号速率范围范围
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status= VL53L0X_SetMeasurementTimingBudgetMicroSeconds(dev,
                        Mode_data[mode].timingBudget);//设定完整测距最长时间
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status = VL53L0X_SetVcselPulsePeriod(dev, VL53L0X_VCSEL_PERIOD_PRE_RANGE,
                        Mode_data[mode].preRangeVcselPeriod);//设定 VCSEL 脉冲周期
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status = VL53L0X_SetVcselPulsePeriod(dev, VL53L0X_VCSEL_PERIOD_FINAL_RANGE,
                Mode_data[mode].finalRangeVcselPeriod);//设定 VCSEL 脉冲周期范围

        error://错误信息
        if(status!=VL53L0X_ERROR_NONE)
        {
            print_pal_error(status);
            LCD_Fill(30,140+20,300,300,WHITE);
            return status;
        }
        return status;
    }
```

vl53l0x_set_mode()函数为单次测量精度模式配置函数，该函数在测量前，对 VL53L0X 传感器进行误差的校正，在使能单次测量模式后，配置工作精度模式。精度模式的配置根据函数参数 mode 变量决定，mode 值对应，0：默认，1：高精度，2：长距离，3：高速，根据全局数组 Mode_data[]变量的写入，从而实现不同精度模式的配置，该全局数组 Mode_data[]定义位置在 vl53l0x.c 文件下。在函数的 VL53L0X 运行前，我们做了复位，由于频繁精度模式的切换，容易导致采集的距离数据不准，所以才加上复位。

接着我们看下 vl53l0x_start_single_test()函数，该函数代码如下：

```
//VL53L0X 单次距离测量函数
//dev:设备 I2C 参数结构体
//pdata:保存测量数据结构体
VL53L0X_Error vl53l0x_start_single_test(VL53L0X_Dev_t *dev,
                                VL53L0X_RangingMeasurementData_t *pdata,char *buf)
{
    VL53L0X_Error status = VL53L0X_ERROR_NONE;
    uint8_t RangeStatus;

    status = VL53L0X_PerformSingleRangingMeasurement(dev, pdata);
                                        //执行单次测距并获取测距测量数据
    if(status !=VL53L0X_ERROR_NONE) return status;
    RangeStatus = pdata->RangeStatus;//获取当前测量状态
    memset(buf,0x00,VL53L0X_MAX_STRING_LENGTH);
    VL53L0X_GetRangeStatusString(RangeStatus,buf);//根据测量状态读取状态字符串
    Distance_data = pdata->RangeMilliMeter;//保存最近一次测距测量数据

    return status;
}
```

该函数为单次测量函数，通过调用 VL53L0X_PerformSingleRangingMeasurement ()API 函数启动单次测量，该 API 函数为阻塞函数，测量数据返回到 pdata 测量数据结构体上，最后将获取测量结构体的测量数据赋值给 Distance_data 全局变量。

（5）vl50l0x_it.c 文件，该文件主要实现连续测量（中断模式）、精度配置等函数。文件代码比较多，这里就不全部列出来了，仅介绍连续测量函数：vl53l0x_interrupt_start()：

vl53l0x_interrupt_start()函数，该函数代码如下：

```
//vl53l0x 连续测量（中断模式）测试
//dev:设备 I2C 参数结构体
//mode: 0:默认;1:高精度;2:长距离;3:高速
void vl53l0x_interrupt_start(VL53L0X_Dev_t *dev,uint8_t mode)
{
    uint8_t VhvSettings;
    uint8_t PhaseCal;
    uint32_t refSpadCount;
    uint8_t isApertureSpads;
    VL53L0X_RangingMeasurementData_t RangingMeasurementData;
    static char buf[VL53L0X_MAX_STRING_LENGTH];//测试模式字符串字符缓冲区
    VL53L0X_Error status=VL53L0X_ERROR_NONE;//工作状态
    u8 key;
    exti_init();//中断 IO 初始化
    LED0=1;
    mode_string(mode,buf);//显示当前配置的模式
    LCD_Fill(30,170,300,300,WHITE);
    POINT_COLOR=RED;            //设置字体为红色
    LCD_ShowString(30,140+30,300,16,16,"Interrupt Mode                    ");
```

```
POINT_COLOR=BLUE;           //设置字体为蓝色
LCD_ShowString(30,140+50,200,16,16,"KEY_UP: Exit the test          ");
LCD_ShowString(30,140+70,200,16,16,"Mode:              ");
LCD_ShowString(80,140+70,200,16,16,(u8*)buf);
sprintf((char*)buf,"Thresh Low:    %d mm ",Thresh_Low);
LCD_ShowString(30,140+90,300,16,16,(u8*)buf);
sprintf((char*)buf,"Thresh High: %d mm",Thresh_High);
LCD_ShowString(30,140+110,300,16,16,(u8*)buf);
LCD_ShowString(30,140+130,300,16,16,"Now value:         mm");

vl53l0x_reset(dev);//复位 vl53l0x(频繁切换精度模式容易导致采集距离数据不准，
                        需加上这一代码)
status = VL53L0X_StaticInit(dev);
if(status!=VL53L0X_ERROR_NONE) goto error;

if(AjustOK!=0)//已校准好了,写入校准值
{
    status= VL53L0X_SetReferenceSpads(dev,Vl53l0x_data.refSpadCount,
                                    Vl53l0x_data.isApertureSpads);//设定 Spads 校准值
    if(status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
    status= VL53L0X_SetRefCalibration(dev,Vl53l0x_data.VhvSettings,
                                        Vl53l0x_data.PhaseCal);//设定 Ref 校准值
    if(status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
    status=VL53L0X_SetOffsetCalibrationDataMicroMeter(dev,
                                    Vl53l0x_data.OffsetMicroMeter);//设定偏移校准值
    if(status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
    status=VL53L0X_SetXTalkCompensationRateMegaCps(dev,
                Vl53l0x_data.XTalkCompensationRateMegaCps);//设定串扰校准值
    if(status!=VL53L0X_ERROR_NONE) goto error;
}else
{
    status = VL53L0X_PerformRefCalibration(dev, &VhvSettings, &PhaseCal);
                                                                //Ref 参考校准
    if(status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
    status = VL53L0X_PerformRefSpadManagement(dev,
                            &refSpadCount, &isApertureSpads);//执行参考 SPAD 管理
    if(status!=VL53L0X_ERROR_NONE) goto error;
    delay_ms(2);
}
status = VL53L0X_SetDeviceMode(dev,
```

```
                VL53L0X_DEVICEMODE_CONTINUOUS_RANGING);//使能连续测量模式
  if(status!=VL53L0X_ERROR_NONE) goto error;
  delay_ms(2);
  status = VL53L0X_SetInterMeasurementPeriodMilliSeconds(dev,
                        Mode_data[mode].timingBudget);//设置内部周期测量时间
  if(status!=VL53L0X_ERROR_NONE) goto error;
  delay_ms(2);
  status=VL53L0X_SetLimitCheckEnable(dev,
          VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,1);//使能 SIGMA 范围检查
  if(status!=VL53L0X_ERROR_NONE) goto error;
  delay_ms(2);
  status = VL53L0X_SetLimitCheckEnable(dev,
  VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,1);//使能信号速率范围检查
  if(status!=VL53L0X_ERROR_NONE) goto error;
  delay_ms(2);
  status = VL53L0X_SetLimitCheckValue(dev,
  VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,Mode_data[mode].sigmaLimit);
                                                        //设定 SIGMA 范围
  if(status!=VL53L0X_ERROR_NONE) goto error;
  delay_ms(2);
  status= VL53L0X_SetLimitCheckValue(dev,
VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,Mode_data[mode].signalLimit);
                                                    //设定信号速率范围范围
if(status!=VL53L0X_ERROR_NONE) goto error;
delay_ms(2);
status= VL53L0X_SetMeasurementTimingBudgetMicroSeconds
                    (dev,Mode_data[mode].timingBudget);//设定完整测距最长时间
if(status!=VL53L0X_ERROR_NONE) goto error;
delay_ms(2);
status = VL53L0X_SetVcselPulsePeriod(dev,VL53L0X_VCSEL_PERIOD_PRE_RANGE,
                    Mode_data[mode].preRangeVcselPeriod);//设定 VCSEL 脉冲周期
if(status!=VL53L0X_ERROR_NONE) goto error;
delay_ms(2);
status = VL53L0X_SetVcselPulsePeriod(dev, VL53L0X_VCSEL_PERIOD_FINAL_RANGE,
              Mode_data[mode].finalRangeVcselPeriod);//设定 VCSEL 脉冲周期范围
if(status!=VL53L0X_ERROR_NONE) goto error;
delay_ms(2);
  status = VL53L0X_StopMeasurement(dev);//停止测量
  if(status!=VL53L0X_ERROR_NONE) goto error;
  delay_ms(2);
  status= VL53L0X_SetInterruptThresholds
  (dev,VL53L0X_DEVICEMODE_CONTINUOUS_RANGING,
  AlarmModes.ThreshLow, AlarmModes.ThreshHigh);//设定触发中断上、下限值
  if(status!=VL53L0X_ERROR_NONE) goto error;
```

```
        delay_ms(2);
        status=VL53L0X_SetGpioConfig
        (dev,0,VL53L0X_DEVICEMODE_CONTINUOUS_RANGING,
        AlarmModes.VL53L0X_Mode,
        VL53L0X_INTERRUPTPOLARITY_LOW);//设定中断模式、 GPIO1 下降沿触发
        if(status!=VL53L0X_ERROR_NONE) goto error;
        delay_ms(2);
        status = VL53L0X_ClearInterruptMask(dev,0);//清除 VL53L0X 中断标志位

        error://错误信息
        if(status!=VL53L0X_ERROR_NONE)
        {
            print_pal_error(status);
            return ;
        }
        alarm_flag = 0;
        VL53L0X_StartMeasurement(dev);//启动测量
        while(1)
        {
            key = KEY_Scan(0);
            if(key==WKUP_PRES)
            {
                VL53L0X_ClearInterruptMask(dev,0);//清除 VL53L0X 中断标志位
                status = VL53L0X_StopMeasurement(dev); //停止测量
                LED1=1;
                break;//返回上一菜单
            }
            if(alarm_flag==1)//触发中断
            {
                alarm_flag=0;
                VL53L0X_GetRangingMeasurementData
                (dev,&RangingMeasurementData);//获取测量距离,并且显示距离
                printf("d: %3d mm\r\n",RangingMeasurementData.RangeMilliMeter);
                LCD_ShowxNum(110,140+130,
                RangingMeasurementData.RangeMilliMeter,4,16,0);
                delay_ms(70);
                VL53L0X_ClearInterruptMask(dev,0);//清除 VL53L0X 中断标志位
            }
            delay_ms(30);
            LED1=!LED1;
        }
    }
```

vl53l0x_interrupt_start()函数为连续测量（中断模式）测试函数，该函数实现了连续测量（中断模式）距离。运行过程，首先对 VL53L0X 误差进行校正，然后配置工作的精度模式，

由于使用了中断模式，需设置 GPIO1 中断引脚的触发模式和上下限距离值，这里我们是使用了窗口模式（即“测量距离<下限距离”或“测量距离>上限距离”时触发中断，获取测量数据。关于其他模式使用请看 API 文档），中断配置好后，启动距离测量。当测量距离在窗口模式范围时，GPIO1 会产生中断，在中断服务函数（中断服务函数代码这里没有截取出来）将 alarm_flag 标志位置 1，最后根据标志位去读取测量的数据。

关于 VL53L0X 的例程代码，我们就介绍到这来，我们再来看看 main.c，该文件里面就一个 main 函数，main 函数代码如下：

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置系统中断优先级分组 2
    delay_init(168);                                //初始化延时函数
    uart_init(115200);                              //初始化串口波特率为 115200
    usmart_dev.init(84);                            //初始化 USMART
    LED_Init();                                     //初始化 LED
    LCD_Init();                                     //LCD 初始化
    KEY_Init();                                     //按键初始化
    AT24CXX_Init();                                 //IIC 初始化
    POINT_COLOR=RED;                                //设置字体为红色
    LCD_ShowString(30,50,200,16,16,"Explorer STM32F4");
    LCD_ShowString(30,70,200,16,16,"Sensor VL53L0X TEST");
    LCD_ShowString(30,90,200,16,16,"ATOM@ALIENTEK");
    LCD_ShowString(30,110,200,16,16,"2017/6/25");
    POINT_COLOR=BLUE;//设置字体为蓝色
    while(AT24CXX_Check())//检测不到 24c02
    {
        LCD_ShowString(30,150,200,16,16,"24C02 Check Failed!");
        delay_ms(500);
        LCD_ShowString(30,150,200,16,16,"Please Check!      ");
        delay_ms(500);
        LED0=!LED0;//DS0 闪烁
    }
    while(1)
    {
        vl53l0x_test();//vl53l0x 测试
    }
}
```

此部分代码比较简单，对用到的外设进行初始化，然后通过调用 vl53l0x_test()，进入 ATK-VL53L0X 模块的主测试程序，对 ATK-VL53L0X 的各项功能（校准测试，普通测量测试、中断测量测试）进行测试。

另外，为了方便大家调试，我们在本例程的 USMART 添加了 vl53l0x_info 和 One_measurement 两个函数。通过调用 vl53l0x_info 函数获取 vl53l0x 设备的 ID 信息、地址等信息内容，通过 One_measurement 函数实现对 vl53l0x 设备执行一次单次距离测量，调用时需输入精度模式的配置，0:默认、1：高精度、2：长距离、3：高速。（注意：使用 One_measurement 函数时，需退出普通测量测试和中断测量测试，避免发生通信冲突）通

过这两个函数，电脑可以通过串口 1，间接控制 ATK-VL53L0X 模块，从而获得设备信息内容和测量数据了。USMART 调试效果，如图 3.2 所示。

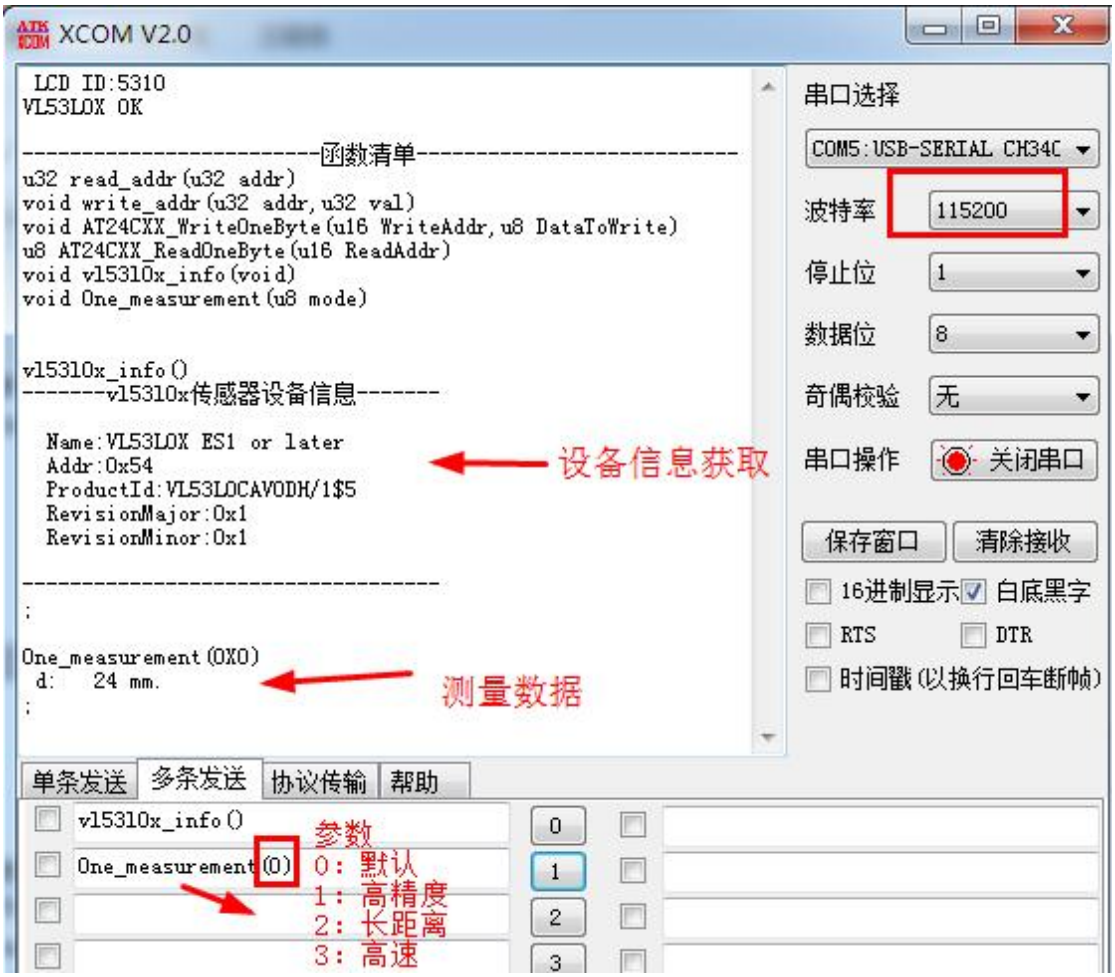

图 3.2 USMART 调试效果

至此，软件实现部分介绍完了，我们接下来看代码验证。

4、验证

首先，请先确保硬件都已经连接好了：

1，ATK-VL53L0X 模块与 ALIENTEK 探索者 STM32F407 开发板连接（连接方式见 2.3 小节）

2，ALIENTEK 探索者 STM32F407 开发板插上 TFTCLD 液晶。

3，給 ALIENTEK 探索者 STM32F407 开发板供电。

代码编译成功之后，我们将代码下载到 STM32 开发板上。LCD 界面显示如图 4.1 所示：

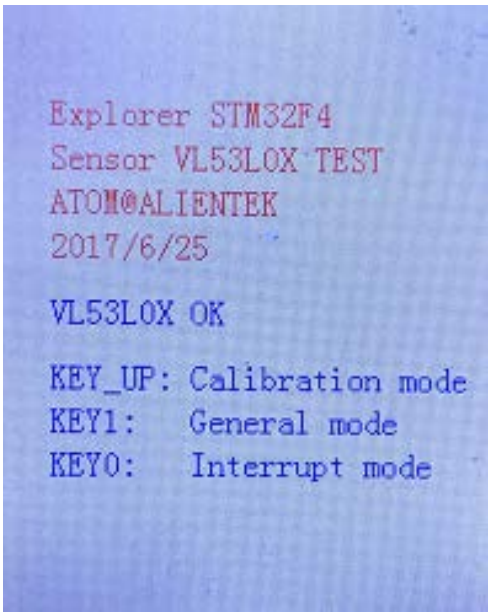

图 4.1 本测试实验界面

可以看到，LCD 屏幕显示 VL53L0X OK，表示 VL53L0X 传感器初始化通过，同时显示了 KEY_UP/KEY1/KEY0 测试功能选项，KEY_UP 校准测试、KEY1 普通测量测试、以及 KEY0 中断测量测试。

4.1 校准测试

在主菜单界面，按 KEY_UP，则可进入此项测试，此项测试为校准 VL53L0X 传感器，校准测试主界面如图 4.1.1 所示：

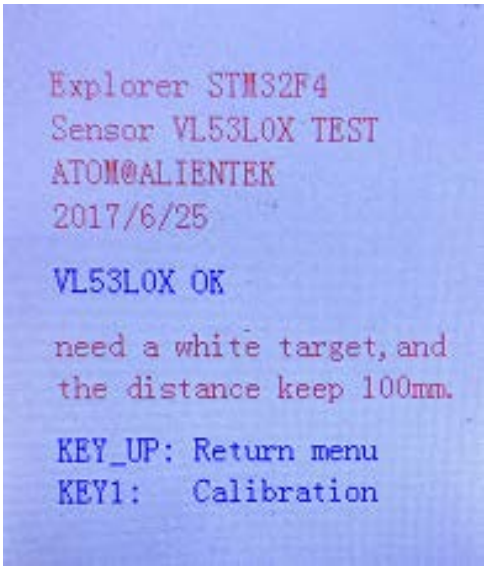

图 4.1.1 校准测试主界面

LCD 屏幕显示校准需要一个白色目标物体，而且 VL53L0X 传感器与目标距离需保持 10 厘米（100mm）。当确认目标和距离无误后，按下 KEY1 执行校准。若不执行校准，则按下 KEY_UP 返回主菜单页面。按下 KEY1 按键后，这时会提示 Start Calibration，表示校准开始，如图 4.1.2 所示：

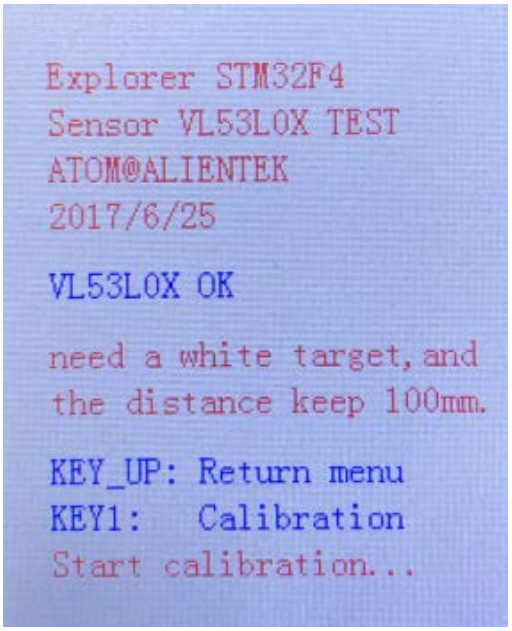

图 4.1.2 校准开始

校准结束后，LCD 屏幕会显示校准状态，若显示 Calibration is complete!，则表示校准结束（成功），显示 Calibration is error，则表示校准失败，同时开发板 DS1 绿灯保持常亮以提示校准失败。（注意：校准失败有可能是当前 VL53L0X 传感器接收不到返回的激光信号，所以请检查模块与目标距离是否为 10 厘米）。校准状态的显示，如图 4.1.3 所示：

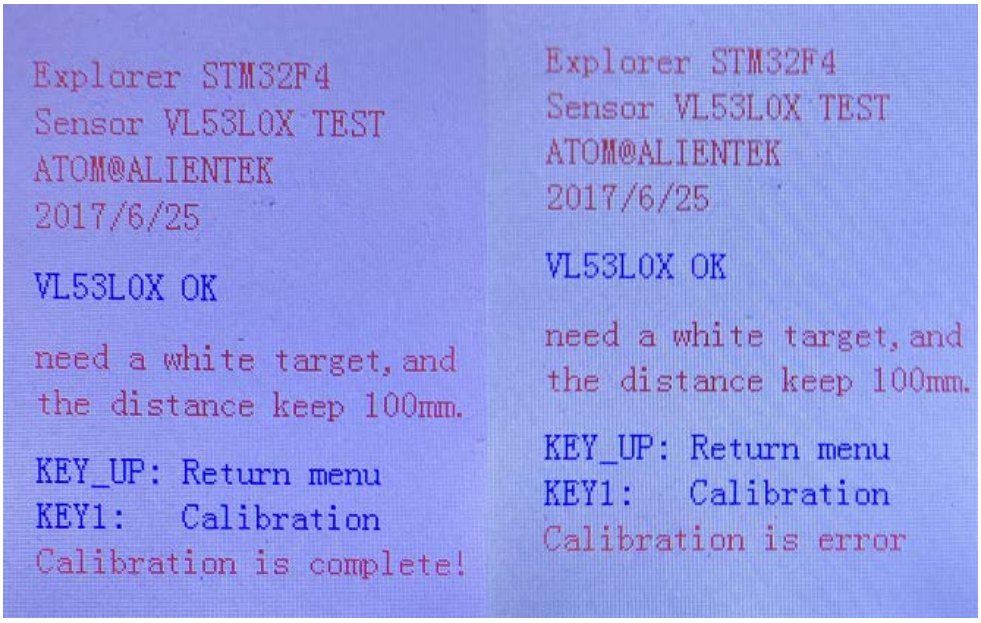

图 4.1.3 校准状态显示

校准状态显示后，LCD 屏幕显示会自动返回主菜单页面。

4.2 普通测量测试

在主菜单页面，按 KEY1，可进入此项测试，此项可测试在单次测量工作模式下，实现在不同精度模式下进行距离的测量。普通测量测试主界面如图 4.2.1 所示：

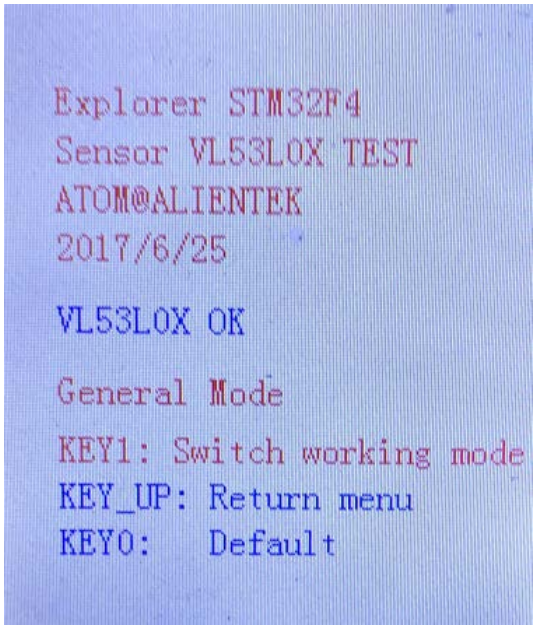

图 4.2.1 普通测量测试主界面

按 KEY1 可切换精度模式，从图 4.2.1 可以看到，当前设置为 Default 默认精度，按 KEY0 进入测试，测试的界面如图 4.2.2 所示。

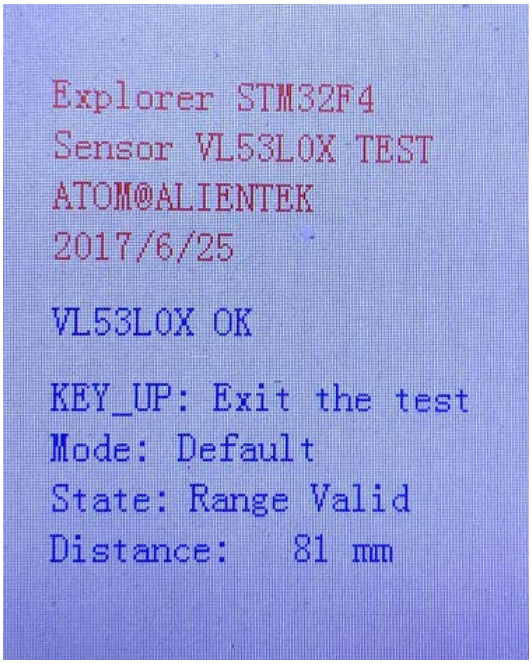

图 4.2.2 默认精度测试

进入测试后，LCD 屏幕会显示当前工作的精度模式，测量状态，以及测量的距离。当测量的距离在有效的测量范围内时，State 状态会显示 Range Valid（范围有效），超出范围则显示 Phase Fail（相位失效）。从图 4.2.2 中看到，当前测量的距离为 8.1 厘米（81mm），测量数据在有效范围内。单击 KEY_UP 会返回精度选择页面，双击 KEY_UP 则会返回主菜单页面。

4.3 中断测量测试

在主菜单页面，按 KEY0，可进入此项测试，此项可测试在连续测量工作模式下，利用中断触发，实现在不同精度模式下进行距离的测量。中断测量测试主界面如图 4.3.1 所示：

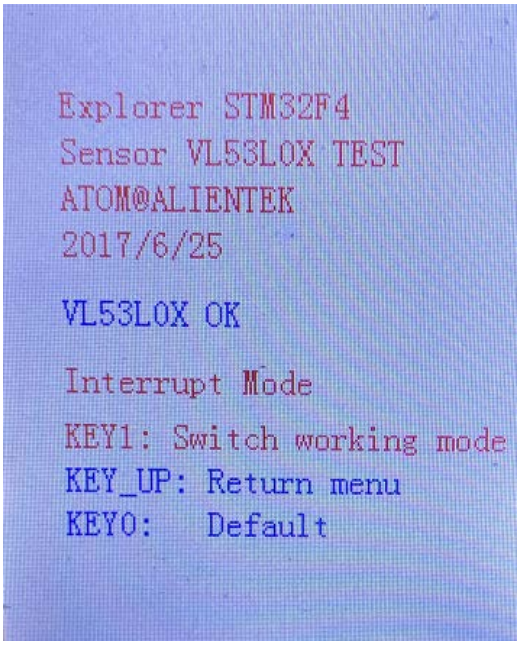

图 4.3.1 中断测量测试主界面

按 KEY1 可切换精度模式，从图 4.3.1 可以看到，当前设置的是 Default 默认精度，按 KEY0 可进入测试，测试的界面如图 4.3.2 所示。

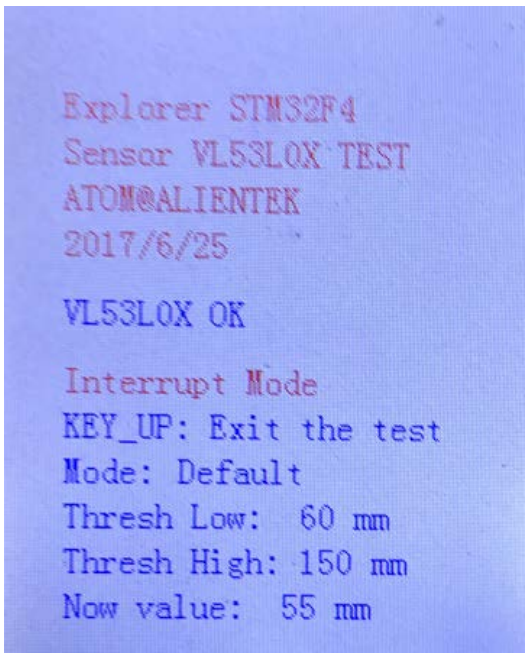

图 4.3.2 默认精度测试

进入测试后，LCD 屏幕会显示当前工作的精度模式，触发中断的上下限距离值，以及测量的距离。当测量的距离在上下限距离范围内时，测量中断不触发，距离数据不做更新变化。当距离小于下限距离值或大于上限距离值时，测量中断触发，测量的距离数据发生变化。从图 4.3.2 中看到，当前测量的距离为 5.5 厘米（55mm）。单击 KEY_UP 会返回精度选择页面，双击 KEY_UP 则返回主菜单页面。

至此，关于 ATK-VL53L0X 模块的使用介绍，我们就讲完了，本文档介绍了 ATK-VL53L0X 模块的使用，有助于大家快速学会 ATK-VL53L0X 模块的使用。

1、 购买地址：

官方店铺 1：https://eboard.taobao.com

官方店铺 2：https://openedv.taobao.com

2、 资料下载

模块资料下载地址：http://www.openedv.com/thread-133995-1-1.html

3、 **技术支持**

公司网址：www.alientek.com

技术论坛：www.openedv.com

传真：020-36773971

电话：020-38271790