

南京宇微电子科技有限公司

VL53L0X 激光测距模块 V1.0

2025 年 9 月 23 日修订

作者：付坤

目录

| | |
|--|----|
| 客户须知..... | 2 |
| 一、概览..... | 3 |
| 二、板卡分区介绍 | 4 |
| 2.1 VL53L0X 传感器..... | 4 |
| 2.2 电源供电 | 4 |
| 2.3 IIC 主通信接口 | 4 |
| 2.4 多功能引脚 GPIO1 | 5 |
| 2.5 关断/复位引脚（Shut Down） | 5 |
| 三、应用..... | 6 |
| 3.1 测距 | 6 |
| 3.2 距离向低频震动测量 | 6 |
| 四、原理图与 PCB 布线 | 7 |
| 五、编程指南..... | 8 |
| 5.1 TOF 测距的数学原理..... | 8 |
| 5.2 基于 STSW-IMG005 API 移植的 USB 模拟串口输出测距值 | 8 |
| 5.2.1 STSW-IMG005 API..... | 8 |
| 5.2.2 校准 | 14 |
| 六、联系我们..... | 24 |

版本修订

| 时间 | 版本号 | 修订内容 |
|-----------------|------|------|
| 2025 年 9 月 23 日 | V1.0 | 初版 |

客户须知

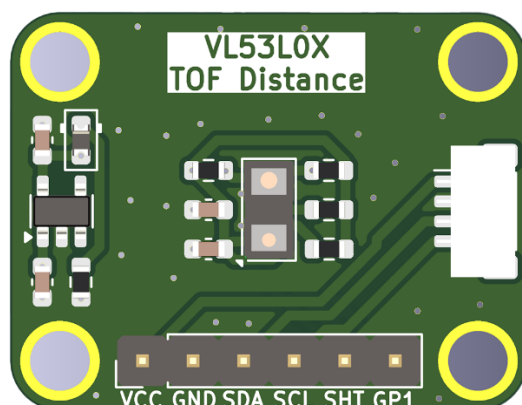
本文档为产品使用参考所编写，文档版本可能随时更新，恕不另行通知。本文中提供的所有使用方法、说明及建议仅供参考，不构成任何承诺或保证。使用本产品及本文档内容所产生的结果，由用户自行承担风险。本公司对因使用本文档或产品而导致的任何直接或间接损失，不承担任何责任。

一、概览

VL53L0X 激光测距模块是一种高精度飞行时间（Time-of-Flight, ToF）距离传感器模块，基于 STMicroelectronics 的 VL53L0X 芯片开发。它采用微型不可见红外激光（940nm VCSEL）和匹配传感器，通过测量光线往返时间实现距离检测，感测锥角仅 35 度，远窄于超声波传感器。

与传统 IR 传感器不同，VL53L0X 无线性误差或“双重成像”问题，能在各种表面和光照条件下提供稳定读数。默认测量范围 30mm 至 1200mm，长距离模式下可达 1.5-2 米，精度 3-12%（视环境而定）。模块集成 3.3V LDO 线性稳压器，支持 3-5V 微控制器（如 Arduino、Raspberry Pi），通过 I2C 接口通信。配备 I2C 专用连接器，即插即用，无需焊接。

本文档提供 STM32 HAL 库，便于编程。适用于机器人避障、手势识别、交互装置等 DIY 项目。尺寸 21x18x2.8mm，重 2.1g。

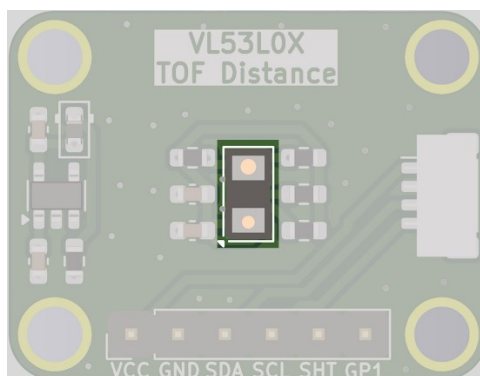


| 模块参数 | |
|------|---|
| 主传感器 | VL53L0X |
| 量程 | 30 mm 至 2000 mm |
| 精度 | 最佳条件下：±3 mm（高反射率目标、室内光照、充分校准）。 <ul style="list-style-type: none">• 典型精度：±3%（即 ±30 mm 在 1 米距离）。• 长距离模式：精度可能降至 ±5% 或更高，因信噪比降低。 |
| 供电 | 输入 VCC：3.3~6.5V |
| 接口 | IIC |
| 尺寸 | 约 20mm×26mm×1.6mm（具体以实物为准） |
| 重量 | 约 2.1g |
| 配件 | <ul style="list-style-type: none">• 板卡×1• 用户手册×1（电子版） |

二、板卡分区介绍

2.1 VL53L0X 传感器

数据手册: [VL53L01X](#)

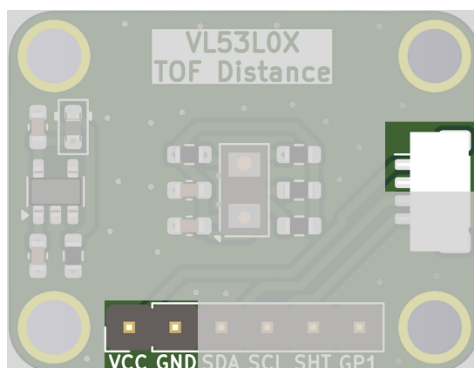


VL53L0X 是一款由 STMicroelectronics 开发的高精度 Time-of-Flight (ToF) 距离传感器，采用 940nm 红外激光和 SPAD 阵列，量程达 2 米（默认模式， $\pm 3\%$ 精度），长距离模式可扩展至约 4 米。发射波束宽度（FOV）约为 $15^\circ \sim 25^\circ$ ，接收波束宽度也约为 25° ，确保较宽的检测范围，适合室内避障、手势识别等场景。最小量程约 30 mm，受校准和盖板玻璃影响。传感器通过 I2C 接口通信，支持单次或连续测距，典型测量时间预算为 30 ms。需执行 SPAD、VHV/相位、偏移和串扰校准以优化精度。结合 STM32 HAL 库，需设置 I2C 超时（如 100 ms）确保通信稳定。校准数据可存至 Flash/EEPROM，支持嵌入式应用中的无校准运行。

2.2 电源供电

功能：为模块提供电源，接受 3.3V 至 6.5V 的输入电压。模块内部集成了 3.3V 稳压器，为 VL53L0X 芯片供电，确保芯片在稳定电压下运行。典型工作电流约为 20mA，视测量频率和模式而定（高精度模式下可能略高）。

注意事项：确保电源稳定，避免电压突变损坏模块。VCC 支持常见的 3.3V 或 5V 微控制器电源。

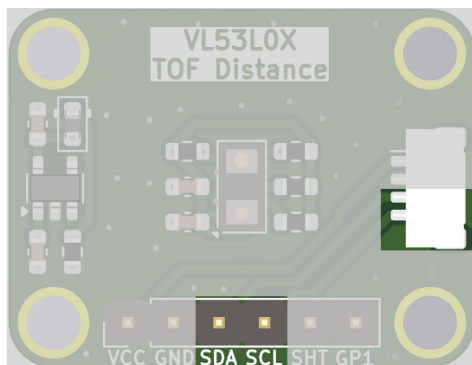


2.3 IIC 主通信接口

I2C 数据引脚（Serial Data）。用于双向数据传输，传输距离、配置命令等。逻辑电平：3.3V/5V 兼

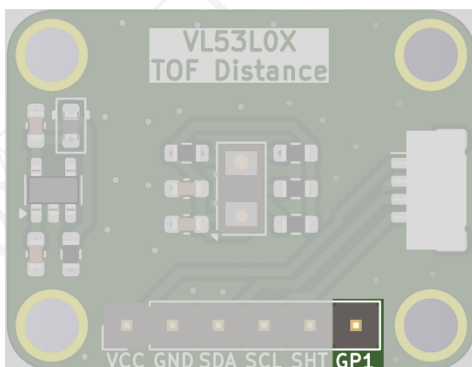
容（内置电平转换）。

- 上拉电阻：模块内置 4.7k Ω 上拉至 3.3V。
- 最大频率：400kHz（快速模式）。



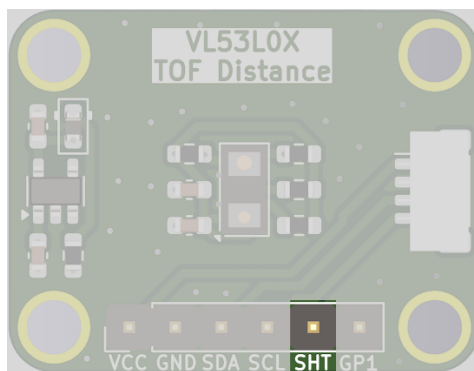
2.4 多功能引脚 GPIO1

模块内置 4.7k 上拉电阻。多功能 GPIO 引脚，主要用作中断输出（Interrupt），模块内置 4.7k Ω 上拉至 3.3V，可配置为距离阈值触发（新测量完成或超出阈值时拉低）。设置阈值为 500mm，当距离 <500mm 时触发中断。也可作为输入配置测量模式，具体操作查看手册 [VL53L01X](#)。



2.5 关断/复位引脚（Shut Down）

模块内置 4.7k 上拉电阻。当 XSHUT 引脚被拉低（电平 <0.4V）并保持至少 1ms 时，VL53L0X 进入**低功耗关断模式**：芯片停止所有操作（包括激光器发射、测量和 I2C 通信），功耗降至极低水平（<5 μ A），I2C 接口不可用，但内部寄存器状态保留。通过将 XSHUT 从高电平拉低（<0.4V）至少 1ms 后拉回高电平（>2.1V），触发芯片的**硬件复位**：芯片内部状态机重置，所有寄存器恢复默认值（包括 I2C 地址恢复为 0x29），激光器、测量模块和 I2C 接口重新初始化，相当于上电复位。



三、应用

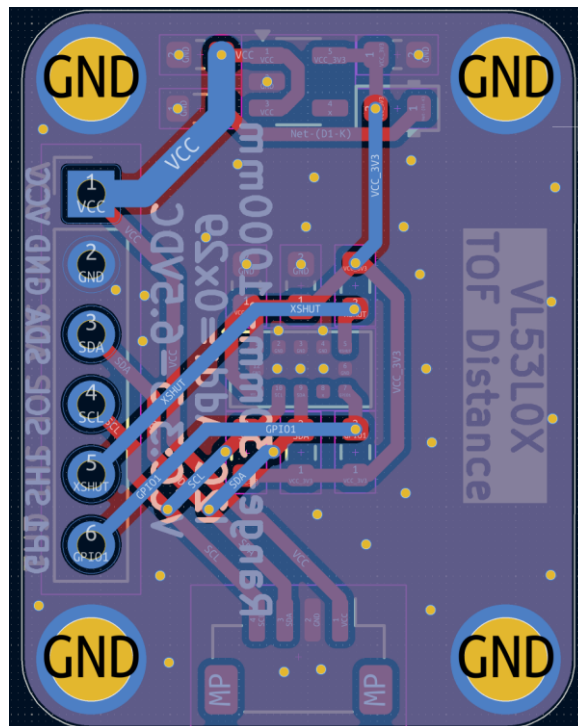
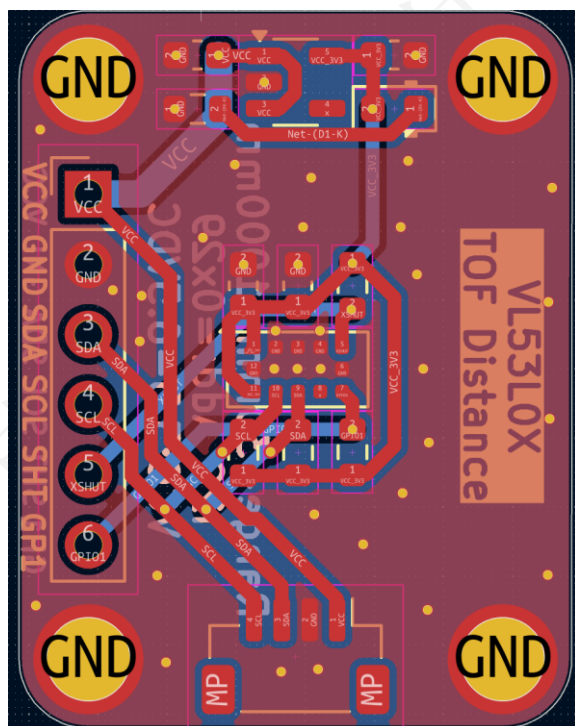
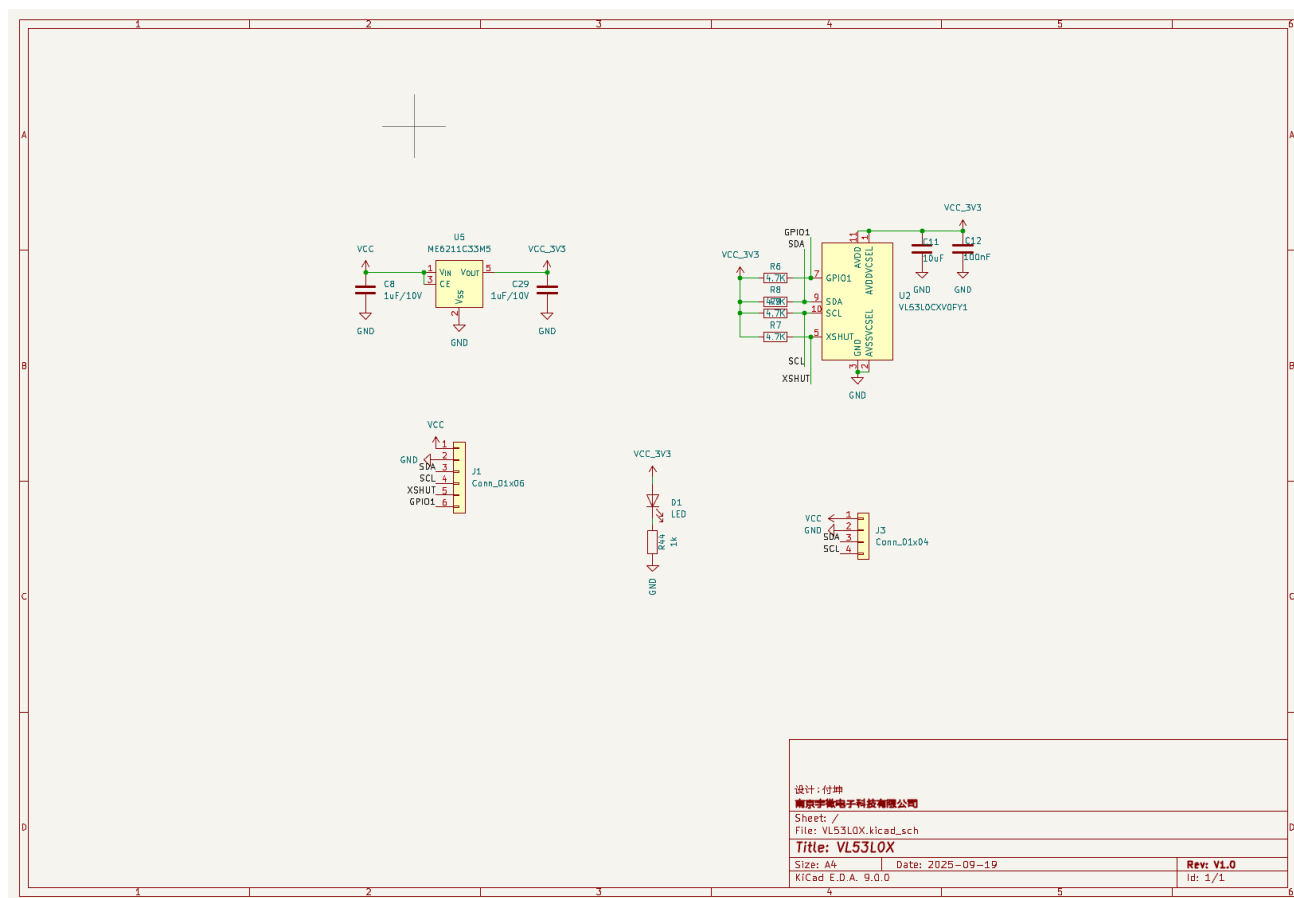
3.1 测距

机器人导航中，它用于避障和路径规划，检测 30mm 至 2m 内的障碍物。无人机可利用其进行高度控制和着陆辅助，确保精准悬停。手势识别系统通过检测手部距离变化实现非接触交互，常用于智能家居或交互艺术装置。自动化设备中，它监测物体位置，如流水线上的零件检测或液位测量。物联网项目可结合其 I2C 接口，集成到智能门锁或安防系统，检测接近物体。教育与 DIY 领域，配合 Arduino 或 Raspberry Pi，用于学习传感器原理或开发创意项目，如距离触发装置。其小巧尺寸和高精度（ $\pm 3\text{-}12\%$ ）使其适合多种表面和光照条件，应用灵活，易于集成，是嵌入式系统和创新设计的理想选择。

3.2 距离向低频震动测量

利用模块的 I2C 接口，结合微控制器（如 STM32），以高帧率（最高 50Hz）连续采集目标物体的距离数据。通过分析距离数据的微小变化，计算目标的位移幅度和频率，推导震动特性。推荐在固定距离（30mm-1200mm）内，针对平滑表面（如金属板）进行测量，以确保精度（ $\pm 1\text{-}10\text{mm}$ ）。软件上，可用上位机处理数据，绘制位移-时间曲线或进行 FFT 分析提取震动频率。需注意，环境光和表面反射会影响精度，需校准。适用于低频震动监测，如机械设备振动或结构健康检测，但不适合高频或大幅度震动。

四、原理图与 PCB 布线



五、编程指南

5.1 TOF 测距的数学原理

ToF 测距通过测量光脉冲从传感器发射到目标物并反射回传感器的时间，结合光速计算距离。其核心公式为

$$D = \frac{c \times t}{2}$$

其中： D 传感器到目标物的距离（单位：米）， c 光速，约为 $3 \times 10^8 m/s$ （在空气中）， t 光脉冲往返时间（单位：秒），实际单程距离需除以 2。

VL53L0X 使用直接 ToF（Direct Time-of-Flight）技术，通过激光脉冲（VCSEL，垂直腔面发射激光器）和 SPAD（单光子雪崩二极管）阵列实现高精度测距。其工作流程和数学原理如下：

- 发射脉冲：VL53L0X 的 VCSEL 发射短脉冲红外激光（波长约 940nm）。脉冲宽度通常在纳秒级，频率由传感器配置（如测量时间预算）决定。
- 接收反射光：SPAD 阵列检测从目标物反射回来的光子。SPAD 的高灵敏度允许检测单个光子，生成时间戳。
- 时间测量：传感器内部的高精度计时器（TDC，Time-to-Digital Converter）测量发射和接收之间的时间差 t 。
- 距离计算：使用公式计算距离 D 。结果以毫米为单位输出，考虑校准参数（如偏移和串扰）进行修正。

5.2 基于 STSW-IMG005 API 移植的 USB 模拟串口输出测距值

STSW-IMG005 API 是意法半导体提供的库，用于驱动 VL53L0X 激光测距传感器。我们本次将这个 API 移植到基于 STM32f103c8t6 的主控板卡，通过 USB 模拟串口实时输出测距值。

5.2.1 STSW-IMG005 API

| 名称 | 修改日期 | 类型 | 大小 |
|-----------------------------|-------------------|----------------------|-------|
| Api | 1/2/2017 4:40 PM | 文件夹 | |
| ApiExample | 1/3/2017 12:37 PM | 文件夹 | |
| doc | 1/2/2017 4:43 PM | 文件夹 | |
| LinuxDriverMassMarket_1.0.7 | 1/5/2017 7:44 PM | 文件夹 | |
| Release_Notes.html | 1/5/2017 7:45 PM | Microsoft Edge HT... | 72 KB |

这个 API 文件目录如上图所示，STSW-IMG005 API 文件夹下包含四个子文件夹：Api、ApiExample、doc 和 LinuxDriverMassMarket_1.0.7，以及一个文件 ReleaseNotes.html，它们的作用分别是

- Api：包含 VL53L0X 应用的编程接口（API）源代码，用于初始化和控制 VL53L0X 传感器（如测距功能），适用于不同平台的开发。

- ApiExample: 提供使用 API 的示例代码, 展示如何集成和调用 API 进行测距等操作, 以 Nucleo F401 为例。
- doc: API 文档, 包括用户手册 (.chm 或.html 格式), 解释 API 功能和使用方法。
- LinuxDriverMassMarket_1.0.7: 包含适用于 Linux 平台的驱动程序或支持文件。
- ReleaseNotes.html: 发布说明文件, 记录软件的更新日志、已知问题和版本信息。

我们本次使用最原始的 Api 来构建项目, 展开文件会发现有 core 和 platform 两个文件夹, 其子目录分别有两个文件夹存放头文件和源文件。意法半导体官方提供这套 Api 时考虑到在多平台使用 VL53L0X 这款传感器, 所以将跟硬件有关的底层驱动(如 IIC 通信)与业务层面的逻辑驱动(如测距)剥离, 所以我们使用不同的板卡驱动 VL53L0X, 需要将 platform 中的硬件驱动改写为适合这张板卡的。

| 名称 | 修改日期 | 类型 | 大小 |
|----------|------------------|-----|----|
| core | 1/2/2017 4:40 PM | 文件夹 | |
| platform | 1/2/2017 4:40 PM | 文件夹 | |

展开 core 的文件, 可以看到一组源文件和头文件, 我们来一一说明。

名称

vl53l0x_api.c
vl53l0x_api_calibration.c
vl53l0x_api_core.c
vl53l0x_api_ranging.c
vl53l0x_api_strings.c

名称





vl53l0x_api.h
vl53l0x_api_calibration.h
vl53l0x_api_core.h
vl53l0x_api_ranging.h
vl53l0x_api_strings.h
vl53l0x_def.h
vl53l0x_device.h
vl53l0x_interrupt_threshold_settings.h
vl53l0x_tuning.h

| 名称 | 作用 |
|---------------------------|---|
| vl53l0x_api.h | 该文件定义了 STMicroelectronics VL53L0X 飞行时间 (ToF) 传感器的编程接口 (API) 头文件, 适用于测距应用。API 提供了一系列函数, 用于初始化设备、配置参数 (如电源模式、测量时间预算、GPIO 设置等)、执行校准 (偏移、交叉干扰等)、进行测距测量 (单次或连续模式) 以及管理中断和 SPAD 设置。这些函数支持开发者通过 C 语言与 VL53L0X 传感器交互, 获取距离数据和设备状态。文件还定义了错误代码和数据结构, 确保开发过程中的一致性与可靠性。 |
| vl53l0x_api_calibration.h | 该文件是 VL53L0X 飞行时间传感器的校准工具包, 专门用来优化测距精度。它能帮你自动调整交叉干扰和偏移误差, 确保测量结果更可靠, 还能优化传感器内部的参考点配置, |





| | |
|--|--|
| | 同时提供相位和参考校准功能，让设备在不同环境下都能保持最佳性能。 |
| vl53l0x_api_core.h | 该文件包含数据处理工具（如字节反转、超时计算）和设备配置功能（如 VCSEL 脉冲周期设置、测量时间预算调整），还能优化测量性能，通过计算信号率、噪声估计和最大距离等参数，确保传感器在各种条件下的准确性。 |
| vl53l0x_api_ranging.h | 目前内容为空，但它旨在定义与传感器测距功能相关的编程接口（API），适用于需要精确距离检测的开发项目。 |
| vl53l0x_api_strings.h | 提供了传感器的状态和错误信息字符串定义。它包含函数来获取设备信息、错误代码描述、测距状态、PAL 状态、序列步骤和限制检查的文字说明，帮助开发者通过人性化的文本理解传感器的工作状态和潜在问题，尤其在调试或日志记录时非常实用。 |
| vl53l0x_def.h | 定义了传感器的核心类型和参数，用于支持其 API 开发。它包含版本信息、设备模式（如单次或连续测距）、电源模式、状态定义、测距数据结构（如距离、信号率、环境光率）以及校准和配置参数（如偏移量、交叉干扰补偿）。 |
| vl53l0x_device.h | 定义了传感器的设备特定参数和寄存器映射。它包含设备错误代码、GPIO 功能选项、寄存器地址（如系统配置、测量结果、校准设置等），以及与光速和信号处理相关的常数。 |
| vl53l0x_interrupt_threshold_settings.h | 定义了传感器的中断阈值设置数组。它包含一系列预定义的字节数据，用于配置中断触发条件，如距离阈值或信号强度，通过硬件中断实现实时监测和响应传感器数据变化，适合需要动态调整测距触发的应用场景。 |
| vl53l0x_tuning.h | 定义了传感器的默认调优设置数组。这些设置包括寄存器配置（如 VCSEL 参数、信号阈值和时序调整），旨在优化传感器的性能和测距精度，帮助开发者根据具体应用需求进行微调，确保设备在不同环境下的稳定运行。 |

展开 platform 的文件，可以看到一组源文件和头文件，我们来一一说明。

名称

 vl53l0x_i2c_platform.c
 vl53l0x_i2c_win_serial_comms.c
 vl53l0x_platform.c
 vl53l0x_platform_log.c

名称

 vl53l0x_i2c_platform.h
 vl53l0x_platform.h
 vl53l0x_platform_log.h
 vl53l0x_types.h

| 名称 | 作用 |
|------------------------|---|
| vl53l0x_i2c_platform.h | 定义了传感器的平台层接口原型，旨在支持与硬件的通信和控制。它提供了初始化和关闭通信（支持 I2C 和 SPI）、读写寄存器数据（单字节、双字节、四字节）、电源循环、GPIO 操作以及延时等待等功能，可根据目标平台调整实现，方便与传 |

| | |
|--------------------------------|---|
| | 传感器进行低级交互和时间戳管理。 |
| vl53l0x_platform.h | 定义了传感器的平台抽象层(PAL)接口,连接 API 与硬件实现。它包含设备结构定义(如 I2C 地址和通信类型)、寄存器读写函数(单字节、双字节、四字节)、序列访问锁机制以及轮询延时功能,允许开发者根据具体操作系统或平台定制实现,确保与传感器的高效交互和线程安全操作。 |
| vl53l0x_platform_log.h | 定义了传感器的平台日志功能。它支持可配置的跟踪级别(错误、警告、信息、调试)、模块和函数过滤,通过宏实现函数入口/出口日志记录和时间戳输出,当启用 VL53L0X_LOG_ENABLE 时可将日志定向到文件或控制台,便于开发者调试和监控传感器运行状态。 |
| vl53l0x_types.h | 定义了传感器的基本类型,属于平台移植部分。它通过包含 stdint.h 和 stddef.h 提供标准整数类型(如 uint8_t、uint16_t、uint32_t 等),并定义了 16.16 固定点类型 FixPoint1616_t 用于处理小数,开发者需根据目标平台调整类型定义,确保跨平台兼容性。 |
| vl53l0x_i2c_win_serial_comms.c | 提供了传感器的 Windows 平台层实现。它基于 Windows API(如互斥锁、事件处理)和串口通信库(SERIAL_COMMS)实现 I2C 通信初始化、关闭、读写寄存器(单字节、双字节、四字节)以及延时、GPIO 操作等功能,包含日志支持(可选启用),并针对非 4 字节对齐数据进行处理,确保与传感器的可靠交互。 |

这里进一步说明, vl53l0x_platform.h 是上层抽象, vl53l0x_i2c_platform.h 是下层实现,两者形成了一个分层架构。vl53l0x_platform.h 包含了 vl53l0x_i2c_platform.h, 确保上层接口能够调用下层通信函数。例如, vl53l0x_platform.h 的 VL53L0X_WriteMulti 依赖 vl53l0x_i2c_platform.h 中的 VL53L0X_write_multi 实现。vl53l0x_platform.h 提供规范, vl53l0x_i2c_platform.h 提供可定制的实现基础。

现在我们来分析并修改一些文件

vl53l0x_i2c_win_serial_comms.c 这个文件被设计用于基于 windows 平台下的串口通信, ST 官方为 VL53L0X 和其他传感器提供上位机软件,这个库与 ST 提供的上位机工具相关,用于 Windows PC 通过串口/U-Boot 协议调试 VL53L0X 等传感器。所以对于用 MCU 直接通过 IIC 控制传感器,然后通过 usb 模拟串口传输到自己的上位机,是不需要这个源文件的。其它上层文件中可能会调用这个文件中的一些函数设置,可以选择直接删除相关的冲突函数。

我们在 core 文件下涉及配置传感器的函数, IIC 通信首先是跳转到 vl53l0x_platform.c/h 文件,然后再跳转到 vl53l0x_i2c_platform.c/h 文件结束,这里面的函数多是占位空函数。所以需要我们自己用 HAL 库(或其它)编写逻辑实现,因此我们需要编写必须的逻辑实现,填充在这些空的占位函数中去,使其能

真正用于通信。我们需要填充的 IIC 通信函数有以下几个

| 名称 | 作用 |
|--|--|
| <code>int32_t VL53L0X_write_multi(uint8_t address, uint8_t index, uint8_t *pdata, int32_t count);</code> | 这是 I2C 通信的多字节写操作, 允许一次性写入 <code>count</code> 个字节到指定寄存器 |
| <code>int32_t VL53L0X_read_multi(uint8_t address, uint8_t index, uint8_t *pdata, int32_t count);</code> | 这是 I2C 通信的多字节读操作, 从指定寄存器读取 <code>count</code> 个字节 |
| <code>int32_t VL53L0X_write_byte(uint8_t address, uint8_t index, uint8_t data);</code> | 这是单字节写操作的便捷封装, 通常用于修改单个寄存器值。 |
| <code>int32_t VL53L0X_write_word(uint8_t address, uint8_t index, uint16_t data);</code> | 向 VL53L0X 设备写入一个 16 位无符号整数 (word), 遵循大端序, 高字节先写入。 |
| <code>int32_t VL53L0X_write_dword(uint8_t address, uint8_t index, uint32_t data);</code> | 向 VL53L0X 设备写入一个 32 位无符号整数 (dword), 遵循大端序, 高字节先写入。 |
| <code>int32_t VL53L0X_read_byte(uint8_t address, uint8_t index, uint8_t *pdata);</code> | 从 VL53L0X 设备读取单个字节。 |
| <code>int32_t VL53L0X_read_word(uint8_t address, uint8_t index, uint16_t *pdata);</code> | 从 VL53L0X 设备读取一个 16 位无符号整数。遵循大端序, 高字节先读取。 |
| <code>int32_t VL53L0X_read_dword(uint8_t address, uint8_t index, uint32_t *pdata);</code> | 从 VL53L0X 设备读取一个 32 位无符号整数。遵循大端序, 高字节先读取。 |

我们使用 HAL 库编写 I2C 的业务逻辑, 实现上述函数功能, 并填充。我们发现其它 6 个函数均使用到批量读写的函数, 因此我们仅需更改批量读写的函数即可。

```
int32_t VL53L0X_write_multi(uint8_t address, uint8_t index, uint8_t *pdata, int32_t count)
{
    int32_t status = STATUS_OK;

#ifdef VL53L0X_LOG_ENABLE
    int32_t i = 0;
    char value_as_str[VL53L0X_MAX_STRING_LENGTH_PLT];
    char *pvalue_as_str = value_as_str;

    // 格式化写入数据为字符串
    for (i = 0; i < count && (pvalue_as_str - value_as_str) < VL53L0X_MAX_STRING_LENGTH_PLT - 3; i++) {
        sprintf(pvalue_as_str, "%02X", *(pdata + i));
        pvalue_as_str += 2;
    }
    *pvalue_as_str = '\0'; // 确保字符串终止
    trace_i2c("Write reg: 0x%04X, Val: 0x%s\n", index, value_as_str);
#endif

    status = HAL_I2C_Mem_Write(&hi2c1, (address << 1), index, 1, pdata, count, I2C_TIMEOUT_MS);
    return HAL_OK ? STATUS_OK : 1;
}
```



```

    if (status != STATUS_OK) {
        //send_data_to_pc(66666); // 错误时发送调试信息
    }

    return status;
}

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

#ifdef VL53L0X_LOG_ENABLE
    int32_t i = 0;
    char value_as_str[VL53L0X_MAX_STRING_LENGTH_PLT];
    char *pvalue_as_str = value_as_str;
#endif

    status = HAL_I2C_Mem_Read(&hi2c1, (address << 1), index, 1, pdata, count, I2C_TIMEOUT_MS)
== HAL_OK ? STATUS_OK : 1;

    if (status != STATUS_OK) {
        //send_data_to_pc(88888); // 错误时发送调试信息
    }

#ifdef VL53L0X_LOG_ENABLE
    // 格式化读取的数据为字符串
    for (i = 0; i < count; i++) {
        sprintf(pvalue_as_str, "%02X", *(pdata + i));
        pvalue_as_str += 2;
    }
    trace_i2c("Read reg: 0x%04X, Val: 0x%s\n", index, value_as_str);
#endif

    return status;
}

```

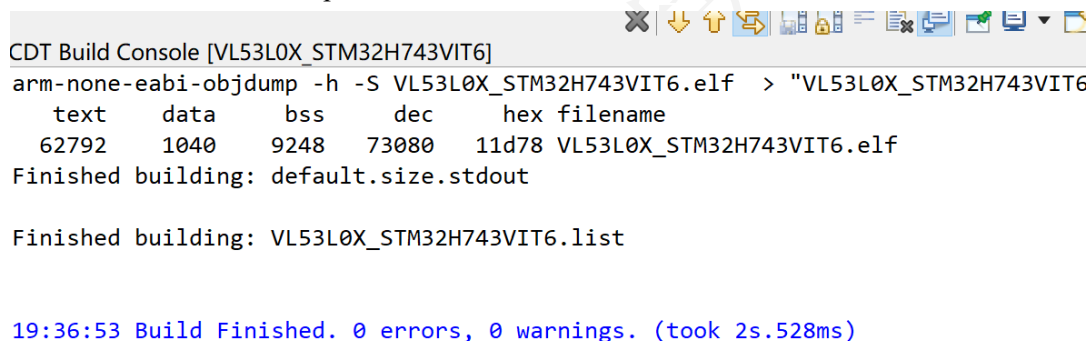
I2C 地址存在高 7 为，最后一位为读写标志位，因此需要将地址左移一位。在编写主程序时，涉及到的地址均使用原地址，而不是左移一位后的地址。

对于文件延迟 1ms 的函数做如下更改，或直接删除弃用。

| vl53l0x_i2c_platform.c 原函数 | 替换函数 |
|--|--|
| VL53L0X_Error | #include "stm32h7xx_hal.h" |
| VL53L0X_PollingDelay(VL53L0X_DEV Dev){ | #include "vl53l0x_i2c_platform.h" // 包含平台延 |

| | |
|---|---|
| <pre> VL53L0X_Error status = VL53L0X_ERROR_NONE; LOG_FUNCTION_START(""); const DWORD cTimeout_ms = 1; HANDLE hEvent = CreateEvent(0, TRUE, FALSE, 0); if(hEvent != NULL) { WaitForSingleObject(hEvent,cTimeout_ms); } LOG_FUNCTION_END(status); return status; } </pre> | <pre> 时函数声明 VL53L0X_Error VL53L0X_PollingDelay(VL53L0X_DEV Dev) { VL53L0X_Error status = VL53L0X_ERROR_NONE; LOG_FUNCTION_START(""); // 使用 HAL 延时, 1ms HAL_Delay(1); LOG_FUNCTION_END(status); return status; } </pre> |
|---|---|

最后 build，根据错误情况调整，platform 相关的函数该删的删，头文件该加的加，经过漫长的挣扎。



```

CDT Build Console [VL53L0X_STM32H743VIT6]
arm-none-eabi-objdump -h -S VL53L0X_STM32H743VIT6.elf > "VL53L0X_STM32H743VIT6
text    data    bss    dec    hex filename
62792   1040    9248   73080  11d78 VL53L0X_STM32H743VIT6.elf
Finished building: default.size.stdout

Finished building: VL53L0X_STM32H743VIT6.list

19:36:53 Build Finished. 0 errors, 0 warnings. (took 2s.528ms)

```

5.2.2 校准

必须按以下顺序执行：SPAD 校准 → VHV/相位校准 → 偏移校准 → 串扰校准。原因是后续校准依赖于前一步的设置（如串扰校准需要正确的 SPAD 和 VHV 参数）。SPAD 和 VHV/相位校准通常在出厂或初始化时执行一次。偏移和串扰校准可能需要定期重新执行，特别是在以下情况下：

- 温度变化 >10°C（VL53L0X 对温度敏感）。
- 更换盖板玻璃或传感器安装位置。
- 测距精度下降。

5.2.2.1 SPAD 校准 (Reference SPAD Management)

VL53L0X 使用 SPAD 阵列检测返回的光子，不同芯片的 SPAD 灵敏度可能存在差异。SPAD 校准优化有效的 SPAD 的数量和类型（Aperture 或 Non-Aperture），以提高信噪比和测距精度。API 函数 VL53L0X_Error VL53L0X_PerformRefSpadManagement(VL53L0X_DEV Dev, uint32_t *refSpadCount,

uint8_t *isApertureSpads), 执行时无需特定目标物, 建议在无强反射物 (如白墙) 的环境下执行。

```
//SPAD 校准
HAL_Delay(5000);
status = VL53L0X_PerformRefSpadManagement(Drv, &refSpadCount, &isApertureSpads);
sprintf(usb_buffer, "status = %ld,refSpadCount = %ld,isApertureSpads_
= %ld\r\n",status,refSpadCount,isApertureSpads);
CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
if (status != VL53L0X_ERROR_NONE) {
    Error_Handler();
}
```

5.2.2.2 VHV 和相位校准 (Reference Calibration)

VHV 校准: 调整激光发射器的参考电压 (VHV), 以优化 VCSEL (垂直腔面发射激光器) 的发射功率。相位校准: 校正信号的相位偏移, 确保 ToF 计算的准确性。这两个校准共同确保传感器在不同环境下的信号质量。API 函数 VL53L0X_Error VL53L0X_PerformRefCalibration(VL53L0X_DEV Dev,uint8_t *pVhvSettings, uint8_t *pPhaseCal), 无需特定目标物, 建议在正常光照、无强反射物环境下执行。

```
//VHV 和相位校准
HAL_Delay(5000);
status = VL53L0X_PerformRefCalibration(Drv, &VhvSettings, &PhaseCal);
sprintf(usb_buffer, "status = %ld,VhvSettings = %ld,PhaseCal
= %ld\r\n",status,VhvSettings,PhaseCal);
CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
if (status != VL53L0X_ERROR_NONE) {
    Error_Handler();
}
```

5.2.2.3 偏移校准 (Offset Calibration)

校正测距结果中的固定偏差 (零点漂移), 通常由传感器硬件或安装环境 (如盖板玻璃) 引起。偏移校准确保测距值为实际距离。API 函数 VL53L0X_Error VL53L0X_PerformOffsetCalibration(VL53L0X_DEV Dev, FixPoint1616_t CalDistanceMilliMeter, int32_t *pOffsetMicroMeter), 需要一个已知距离的平坦目标物 (如白色卡纸, 推荐距离 100mm)。目标物应为漫反射表面 (避免镜面反射)。

```
// 偏移校准 (需要 100mm 已知距离目标)
HAL_Delay(5000);
status = VL53L0X_PerformOffsetCalibration(Drv, 100000, &VL53L0X_Offset);
sprintf(usb_buffer, "status = %ld,VL53L0X_Offset = %ld\r\n",status,VL53L0X_Offset);
CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
```

```
if (status != VL53L0X_ERROR_NONE) {
    Error_Handler();
}
```

5.2.2.4 串扰校准 (Crosstalk Calibration)

校正由盖板玻璃或其他光学组件引起的信号串扰（即反射光干扰）。串扰会导致测距结果偏小，校准后可提高精度。API 函数 VL53L0X_Error VL53L0X_PerformXTalkCalibration(VL53L0X_DEV Dev, FixPoint1616_t XTalkCalDistance, FixPoint1616_t *pXTalkCompensationRateMegaCps)，需要一个已知距离的非高反射目标（如灰色卡纸，推荐距离 100mm）。确保盖板玻璃清洁且正确安装。

```
// 串扰校准（需要 100mm 已知距离目标和盖板玻璃）
HAL_Delay(5000);
status = VL53L0X_PerformXTalkCalibration(Dev, 100000, &VL53L0X_XtalkCompensationRate);
sprintf(usb_buffer, "status = %ld,VL53L0X_XtalkCompensationRate
= %ld\r\n",status,VL53L0X_XtalkCompensationRate);
CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
if (status != VL53L0X_ERROR_NONE) {
    Error_Handler();
}
```

注意事项：串扰校准必须在 SPAD 和 VHV 校准之后执行。保存 xtalkCompensationRate，以便在需要时恢复（使用 VL53L0X_SetXTalkCompensationRateMegaCps）。如果没有盖板玻璃，串扰校准可能不需要，但建议执行以确认。通过 STM32CUBEMX 生成工程文件，并执行关键的初始化流程，整合完成的主函数为

```
/* USER CODE BEGIN Header */
/**
*****
* @file      : main.c
* @brief     : Main program body
*****
* @attention
*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*/
```

```

*****
*/
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"
#include "i2c.h"
#include "rtc.h"
#include "usb_device.h"
#include "gpio.h"
#include "vl53l0x_platform.h"
#include "vl53l0x_api.h"
#include "usbd_cdc_if.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */
#define Calibration//Calibration/Running
#define VL53L0X_I2C_ADDRESS 0x29 // VL53L0X 默认 I2C 地址
#define WHO_AM_I_REG 0xC0
/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/

/* USER CODE BEGIN PV */
VL53L0X_Dev_t MyDevice;
VL53L0X_DEV Dev = &MyDevice;

VL53L0X_RangingMeasurementData_t RangingData; // 测距数据结构
char usb_buffer[50];
VL53L0X_Error status;

```



```

uint32_t refSpadCount=0; // 有效 SPAD 数量（典型值 10-50
uint8_t isApertureSpads=0; // SPAD 类型（0 = Non-Aperture, 1 = Aperture）
uint8_t VhvSettings=0; // 电压校准值
uint8_t PhaseCal=0; // 相位校准值
int32_t VL53L0X_Offset=0; // 偏移值（单位：微米）
FixPoint1616_t VL53L0X_XtalkCompensationRate=0; // 串扰补偿率（单位：MCPS）

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
void VL53L0X_Init(void);
void VL53L0X_GetCalibration(void);
void VL53L0X_SetCalibration(void);
void VL53L0X_DeviceMode(void);
void send_data_to_pc(int32_t distance);
void VL53L0X_CheckWhoAmI(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */
/* VL53L0X 初始化 */
void VL53L0X_Init(void) {

    // 初始化设备结构
    Dev->I2cDevAddr = VL53L0X_I2C_ADDRESS ;
    Dev->comms_type = I2C;
    Dev->comms_speed_khz = 400;
    // 调用 VL53L0X API 初始化
    status = VL53L0X_DataInit(Dev); // 数据初始化
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }

    status = VL53L0X_StaticInit(Dev); // 静态初始化
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }
}
#ifdef Calibration
VL53L0X_GetCalibration(); // 校准后键入校准参数，再运行前可以注释掉
#endif
#ifdef Running

```

```

VL53L0X_SetCalibration();//校准后键入校准参数，再运行前可以注释掉
#endif
VL53L0X_DeviceMode();

}

/* 调用 VL53L0X API 校准 */
void VL53L0X_GetCalibration(void) {

    //SPAD 校准
    HAL_Delay(5000);
    status = VL53L0X_PerformRefSpadManagement(Dev, &refSpadCount, &isApertureSpads);
    sprintf(usb_buffer, "status = %ld,refSpadCount = %ld,isApertureSpads
= %ld\r\n",status,refSpadCount,isApertureSpads);
    CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }
    //VHV 和相位校准
    HAL_Delay(5000);
    status = VL53L0X_PerformRefCalibration(Dev, &VhvSettings, &PhaseCal);
    sprintf(usb_buffer, "status = %ld,VhvSettings = %ld,PhaseCal
= %ld\r\n",status,VhvSettings,PhaseCal);
    CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }
    // 偏移校准（需要 100mm 已知距离目标）
    HAL_Delay(5000);
    status = VL53L0X_PerformOffsetCalibration(Dev, 100000, &VL53L0X_Offset);
    sprintf(usb_buffer, "status = %ld,VL53L0X_Offset = %ld\r\n",status,VL53L0X_Offset);
    CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }
    // 串扰校准（需要 100mm 已知距离目标和盖板玻璃）
    HAL_Delay(5000);
    status = VL53L0X_PerformXTalkCalibration(Dev, 100000, &VL53L0X_XtalkCompensationRate);
    sprintf(usb_buffer, "status = %ld,VL53L0X_XtalkCompensationRate
= %ld\r\n",status,VL53L0X_XtalkCompensationRate);
    CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }
}

```

```
}
/* 调用 VL53L0X API 校准 */
void VL53L0X_SetCalibration(void) {

    // 应用校准数据
    VL53L0X_SetReferenceSpads(Dev, refSpadCount, isApertureSpads);
    VL53L0X_SetRefCalibration(Dev, VhvSettings, PhaseCal);
    VL53L0X_SetOffsetCalibrationDataMicroMeter(Dev, VL53L0X_Offset);
    VL53L0X_SetXTalkCompensationRateMegaCps(Dev, VL53L0X_XtalkCompensationRate);

}
/* VL53L0X 测量模式设置 */
void VL53L0X_DeviceMode(void) {

    status = VL53L0X_SetDeviceMode(Dev, VL53L0X_DEVICEMODE_SINGLE_RANGING); // 设置
    单次测距模式
    if (status != VL53L0X_ERROR_NONE) {
        Error_Handler();
    }

}

/* 发送数据到 PC */
void send_data_to_pc(int32_t distance) {
    sprintf(usb_buffer, "Distance: %ld mm\r\n", distance);
    CDC_Transmit_FS((uint8_t*)usb_buffer, strlen(usb_buffer));
}

void VL53L0X_CheckWhoAmI(void) {
    uint8_t who_am_i = 0;
    HAL_StatusTypeDef status;

    // 写入寄存器地址
    status = HAL_I2C_Mem_Read(&hi2c1, 0x29<<1, WHO_AM_I_REG, I2C_MEMADD_SIZE_8BIT,
    &who_am_i, 1, 100);
    if (status == HAL_OK) {

        send_data_to_pc(who_am_i);

    } else {
        send_data_to_pc(88);
    }
}
```

```
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_USB_DEVICE_Init();
    MX_RTC_Init();

    VL53L0X_CheckWhoAmI();
    /* USER CODE BEGIN 2 */
    /* VL53L0X 初始化 */
    VL53L0X_Init();
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
```

```
{
    /* USER CODE END WHILE */
    /* 启动单次测距 */
    VL53L0X_PerformSingleRangingMeasurement(Dev, &RangingData);

    /* 获取距离数据 */
    uint16_t distance = RangingData.RangeMilliMeter;

    /* 通过 USB 虚拟串口发送数据 */
    send_data_to_pc(distance);

    /* 延时 500ms */
    HAL_Delay(500);
    /* USER CODE BEGIN 3 */

}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

    /** Initializes the RCC Oscillators according to the specified parameters
     * in the RCC_OscInitTypeDef structure.
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
}
```



```

/** Initializes the CPU, AHB and APB buses clocks
 */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_RTC|RCC_PERIPHCLK_USB;
PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
{
    Error_Handler();
}
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
        HAL_Delay(5000);
        send_data_to_pc(888);
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

```

```
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

六、联系我们

若需任何帮助，请邮件联系我们: info@fukunlab.com

样品购买: [淘宝店铺-宇微电子](#)