廣東工業大學

本科毕业设计（论文）

无线激光雷达采集端的设计与实现

学　　院	自动化学院
专　　业	自动化
年级班别	2018 级(3)班
学　　号	3118000954
学生姓名	郑思豪
指导老师	林靖宇教授

2022 年 6 月

无线激光雷达采集端的设计与实现

郑思豪

自动化程学院

摘　　要

激光雷达是一种精密的光学仪器，广泛应用在无人驾驶、场景扫描、工业检测等需要感知环境和物体的技术当中。

随着 IoT 技术的发展以及万物互联的前景，将设备进行 IoT 设计可以扩展其功能和应用场景。传统的激光雷达需要和 PC 主机之间传输数据需要进行有线连接。在有线连接的场景中，由于有线而导致激光雷达扫描过程中受到部分干扰，甚至限制原地距离和范围。激光雷达需要旋转扫描环境时，接口会由于旋转而导致松脱，甚至有线会阻挡激光雷达的扫描镜头导致扫描结果失真。

为了解决有线传输带来的问题，本文提出将激光雷达和 IoT 无线通信设备相结合的实现激光雷达无线通信的方案。本文首先通过激光雷达选型比较，选定合适的激光雷达。然后在选定激光雷达硬件结构基础上，采用合适的 IoT 无线通信设备进行通信连接设计，提出可行的硬件连接方案。经过设备比较，本文采用 USB 接口的激光雷达和同样具备 USB 接口 IoT 的嵌入式开发板的硬件连接方案。根据实际的硬件的连接，本文编写软件为通信提供支持，其中软件分为两个部分，即激光雷达和 IoT 无线通信设备间的 USB 通信，以及 IoT 无线通信设备和个人计算机间的 WiFi 无线通信。这样激光雷达就可以实现个人计算机远程控制。因此本文的主体内容是激光雷达和 IoT 嵌入式开发板的硬件连接和软件程序驱动。最后本文依据实际成品进行调试以及进行个人计算机和激光雷达无线通信的测试。解决通信存在的问题后，最终实现激光雷达无线传输。

关键词：IoT，激光雷达，无线通信，嵌入式开发板

Abstract

Lidar is a kind of precise optical instrument, which is used in unmanned driving, scene scanning, industrial detection and other technologies that need to perceive the environment and objects. With the development of IoT technology and the prospect of interconnection of all things, IoT design of equipment can expand its functions and application scenarios. The traditional lidar needs to be wired with the PC host to transmit data. In the scene of wired connection, the lidar scanning process is partially disturbed due to wired connection, and even the in-situ distance and range are limited. When the lidar needs to rotate the scanning environment, the interface will become loose due to rotation, and even the wire will block the scanning lens of the lidar, resulting in the distortion of the scanning results.

In order to solve the problems caused by wired transmission, this paper proposes a scheme of combining lidar and IOT wireless communication equipment to realize lidar wireless communication. Firstly, through the selection and comparison of lidar, this paper selects the appropriate lidar. Then, on the basis of selecting the hardware structure of lidar, appropriate IOT wireless communication equipment is used for communication connection design, and a feasible hardware connection scheme is proposed. After equipment comparison, this paper adopts the hardware connection scheme of lidar with USB interface and embedded development board with USB interface IOT. According to the actual hardware connection, this paper writes software to support communication. The software is divided into two parts, namely, USB communication between lidar and IOT wireless communication equipment, and WiFi wireless communication between IOT wireless communication equipment and personal computer. In this way, the lidar can realize the remote control of personal computer. Therefore, the main content of this paper is the hardware connection and software driver of lidar and IOT embedded development board. Finally, according to the actual finished products, this paper debugs and tests the wireless communication between personal computer and lidar. After solving the problems existing in communication, the wireless transmission of lidar is finally realized.

Key word： IoT, Lidar, Wireless Communication, Embedded Board

目　　录

1 引言

1.1 背景与意义

作为信息社会不可或缺而又日常所需的电子产品，个人计算机搭载操作系统、网络以及接口，因此个人计算机（PC）不仅能够作为和外部设备交互主机，还可以对外部设备进行管理。区别于更加专业化用于服务器的计算机，或者用于嵌入式的微型计算机，甚至用于超级计算的超级计算机，个人计算机更加注重人机交互，应用场景也更加多样化。在这种情况下许多设备都可以和个人计算机进行连接和交互，个人计算机的操作系统能够对所连接的设备进行驱动以及管理。

同时，从台式机计算机到笔记本计算机，个人计算机（PC）不断发展出不同的类型而且更加专业化。以笔记本计算机为例，为满足生活需求，笔记本电脑划分为了游戏笔记本电脑和轻薄笔记本电脑。虽然性能方面不及台式电脑，但笔记本计算机不仅由于其方便携带而且作为主机与外部设备交互便捷，更为重要的是具备市场上台式计算机所没有的无线通讯能力。笔记本计算机自带若干接口可以和外部设备连接，但是随着日常所需连接的外部设备的类型日益增多以及更加专业化，笔记本计算机自带的接口不能满足需求。扩展坞和集线器的出现可以缓解这一需求，同时笔记本计算机可以连接具备无线通讯的外部设备。在有线和无线的情况下，无需高传输速率和低丢失率的外部设备采用无线传输更为合适。

同样激光雷达不断普及在日常生活中，在模式识别和测量的场景中不断应用，如无人驾驶汽车应用激光雷达进行环境信息采集，工厂采用激光雷达检测器件是否符合标准从而剔除次品[1]。激光雷达通过发射激光束，将接收到的反射信号进行处理，生成数据。激光雷达高精度的性能需要良好的环境保障。然而部分作为外设的激光雷达有线连接到主机，在这种情况下会由于接线长度，激光雷达可能将接线和个人计算机一并扫描，甚至在某些需要激光雷达运动扫描时，与个人计算机接线会接触不良。另一方面在目前的有线设备端口不断更新时，老旧接口淘汰而让老旧的激光雷达端口无法适配新端口的设备而不得不换新激光雷达，甚至对于激光雷达所需的传输数据速度和数据量，选择老旧端口更有性价比，而更换新式端口则增加成本。

对比于使用树莓派等微型计算机实现激光雷达无线传输，IoT 嵌入式板可以真正实

现 IoT 的功能，不仅体积比微型计算机小，而且续航时间更加长。以树莓派为例，市场上销售的树莓派功率最低 Raspberry Pi Model A 的自身功耗是 1.5W，1000mAh 的电池能够运作 40 分钟，而嵌入式开发板以 Arduino 的功耗比较大的 Arduino UNO 为例，其自身功耗 0.25W，同样的容量电池可以达到 4 小时。因此树莓派也可以提供无线传输功能，但采用 IoT 嵌入式开发板才能真正 IoT 化。

为了让激光雷达脱离有线连接 PC 通信的方式，跟上 IoT 时代万物互联的要求，因此本文将具备 WiFi 无线通信的嵌入式开发板来扩展激光雷的功能。

1.2 国内外情况

在国外常采用的开发板是 Arduino。Arduino 较为便捷和开源的特性能够容易开发和激雷达连接的程序。

在 2013 年，Hokuyo URG-04LX-UG01 和 Vinculm 连接然后将数据通过嵌入式板的串口传输数据到 PC 的实例。Vinculm 是 FTDI（英商飞特帝亚）公司推出的搭载 USB 主机芯片的 USB 嵌入式板，主要是通用设备通信的抽象控制模型对 USB 外设进行交互和管理[2]。

在 2013 年也已经有在机器人进行设计的将 Arduino USB-Host-Shield 和 Hokuyo 激光雷达连接，进行无线通信的实例。同样采用通用设备通信的抽象控制模型和 USB 外设通信[2]。

同时激光雷达也出现带有无线通信功能的产品，有的激光雷达具备以太网接口，可以利用激光雷达的以太网端口和 Arduino 开发板进行连接的，或者将激光雷达的以太网端口和迷你 WiFi 或者路由器连接实现无线传输。

在国内比较流行的嵌入式板之一是意法半导体出品的 STM32，STM32 品种多样，部分产品具备 USB OTG 功能。在国内 STM32 的教程资料也同样丰富。以及有通过 STM32 和激光雷达连接，然后将数据通过无线通信模块传到 PC。在应用的网络方面有 TCP/IP 和 Lora。Lora 是比较新的无线网络技术

ESP32 是国内比较流行的无线网络集成芯片，内部搭载的微处理器可以进行数据的采集[3]。ESP32 同样具有 USB OTG 功能，目前有对激光雷达和 ESP32 进行连接的设计，但是案例并不多。

1.3 本论文构成以及设计与实现内容

通过前文对个计算机和激光雷达的概述，以及两者有线连接的问题，本文参考国内外的案例，设计个人计算机无线连接激光雷达，以实现对激光雷达进行管理和数据传输。因此本文构成分为 5 个结构，如图 1.1 所示。

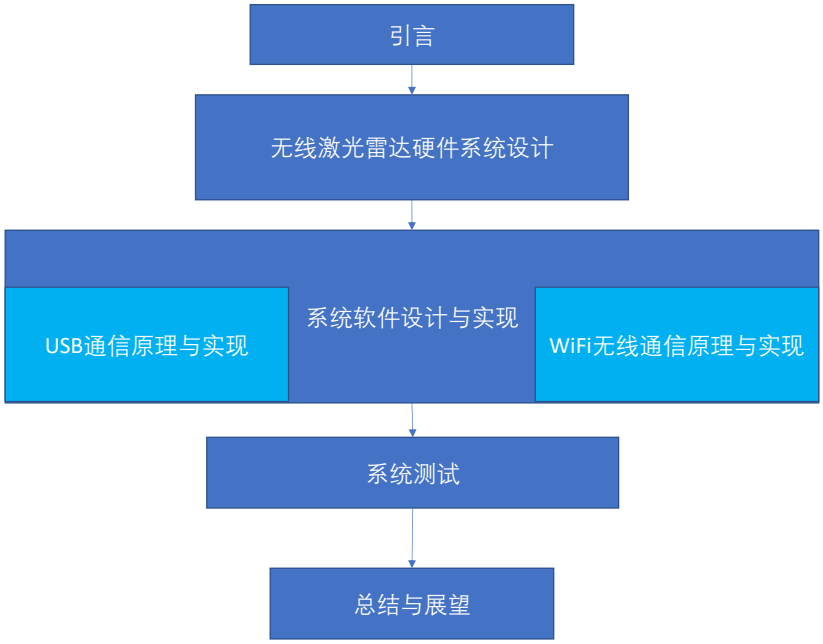

图 1.1 论文构成图

第一章是本文的引言。引言叙述了本文的对标的无线激光雷达现实需要的问题、国内外实现无线激光雷达的案例和本文的结构。

第二章是无线激雷达硬件系统设计。该章主要内容是对设备的选型以及设备连接的方案。本文采取 USB 连接和 WiFi 模块进行无线通信的方案，因此根据硬件的选择和连接方案引出第三章系统软件设计与实现。

第三章是系统软件设计与实现。该章是为硬件提供软件程序支持，根据本文的硬件方案分为两个部分，一部分是 USB 通信原理与实现，另一部分是 WiFi 通信原理与实现。软件程序根据 USB 和 WiFi 通信的原理合理编写，最后构造成为硬件进行通信的软件系统。

第四章是系统测试。系统测试包含通信调试以及系统集成测试。通信测试对本文的成品进行三方面的调试，即 USB、WiFi 和供电。最后系统集成测试是用成品和个人计

算机进行无线通信的测试。

第五章是总结与展望。第五章是论文最后的内容，总结论文和对未实现方案的构想。

2 无线激光雷达硬件系统设计

2.1 引言

激光雷达和物联网（IoT）嵌入式开发板进行硬件层面上连接是实现无线传输的基础，也是关键的一步。设备需要根据所选的激光雷达的硬件进行设计，同时也需要对供电进行设计。

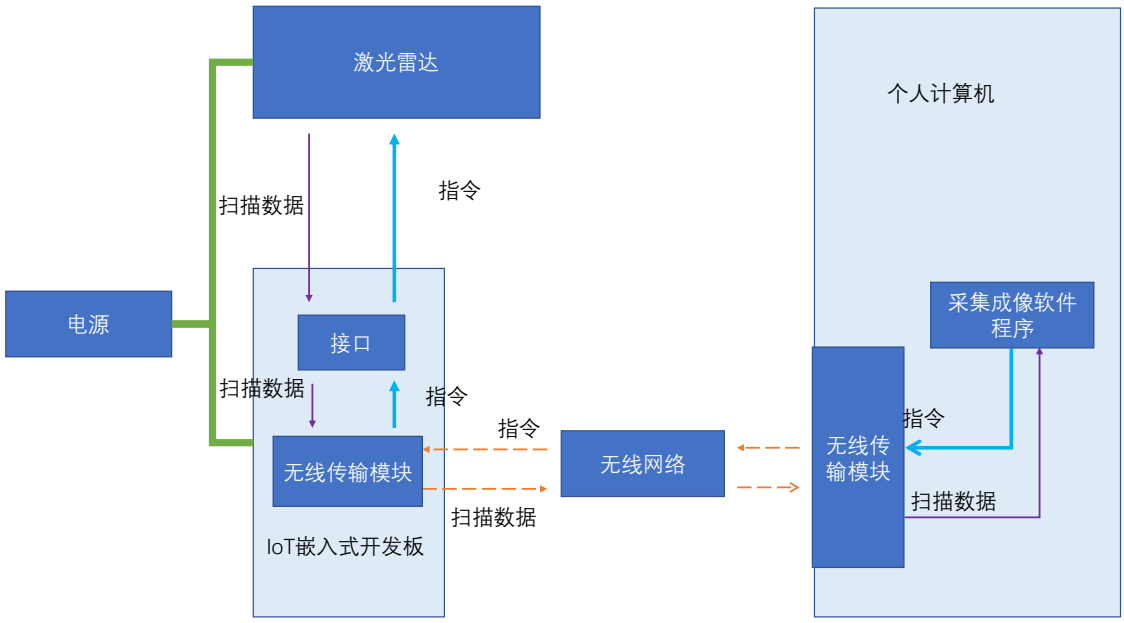

图 2.1 无线激光雷达硬件系统设计图

图 2.1 是激光雷达的硬件设计图，激光雷达和 IoT 嵌入式开发板通过进行物理接口连接，使得激光雷达和 IoT 嵌入式开发板可以进行通信。IoT 嵌入式开发板通过无线网络和个人计算机进行无线通信。

2.2 激光雷达

2.2.1 选型

不同的激光雷达需要不同的硬件进行连接，在对比市场上大多数激光雷达后，本文选择了 Hokuyo（北阳）激光雷达的 URG-04LX-UG01 型号激光雷达。

Hokuyo（北阳）激光雷达是日本的筑波大学参与研发的激光雷达。相比与其他种类的激光雷达，该激光雷达显著的特征是小巧（50mm×50mm×65mm，160 克）。

Hokuyo 的激光雷达有单个 USB 接口的 URG-04LX-UG01，也有同时具有以太网端

口和电源端口两种端口的 URG-05LX。URG-04LX-UG01 激光雷达通过单个 USB 接口进行供电和数据传输，而且输入的电压和电流是 USB2.0 版本所支持的 5V500mA。对比于采用以太网接口的激光雷达，使用 USB 接口的激光雷达传输速率更稳定而且更快，URG-04LX-UG01 的传输速率是 9Mbitps。

该激光雷达精度在 60 到 1000 毫米范围的偏差在 30 毫米，1000 毫米到 4095 毫米则是 3%测量距离。测距分辨率高（1mm）。角度分辨率高角度分辨率在 0.36 度。扫描速度快，扫描时间是 100 毫秒一次。噪声在 25 分贝范围内。

本文采用 URG-04LX-UG01，进行设计实现。由于该激光雷达通讯和电源集成于单个端口，通过 USB 供电和通信，因此需要选择合适的电源可以驱动而且不影响嵌入式板。

2.2.2 激光雷达控制

SCIP 协议该协议是北阳激光雷达的通信协议之一，也是激光雷达的指令系统。SCIP 支持 USB 和 RS232C 协议通信，因此激光雷达通过接口可以和 RS232C 接口的设备或者 USB 接口的设备进行通信。对于 USB 的设备，SCIP 支持 CDC（通信设备类）标准，并且由 USB 主机进行驱动[4]。

表 2.1 部分 SCIP 指令表

指令	描述	发送示例
PP	查询激光雷达的规范	PP\n
VV	查询激光雷达的版本	VV\n
II	返回激光雷达运行时的状态	II\n
TM	设置激光雷达的时间	TM\n
BQ	开启激光雷达扫描	BQ\n
QT	关闭激光雷达扫描	QT\n
SS	设置激光雷达波特率	SS019200\n
MD	进行连续扫描	MD0044072501004\n

RS232C 通信需要 8 位数据和 1 位停止位，并且最高波特率为 750Kbps。USB 通信支持 2.0 版本但只有 9Mbps 的全速[4]。表 2.1 为激光雷达 SCIP 指令的示例。

SCIP 发送的指令需要添加“\n”进行结尾。PP、VV 和 II 指令是对激光雷达的描述，根据这些描述可以进行激光雷达初始化，包括发送“TM\n”进行时间同步以及“SS”指令设置波特率。由于 SCIP 协议对激光雷达进行返回数据进行了编码，在使用激光雷达时需要对数据进行解码。

2.3 物联网（IoT）嵌入式开发板

2.3.1 概述

URG-04LX-UG01 激光雷达通过一个 USB 口完成供电和数据传输，因此需要选择具备 USB 接口嵌入式开发板，并且开发板可以通过 USB 通信。

IoT 嵌入式开发板可以网络连接从而可以进行无线通信，市面上的较流行的具有 USB 的 IoT 嵌入式开发板有国内乐鑫出品的 ESP32-S 系列和意大利的 Arduino MKR 系列。

乐鑫的 ESP32-S3-DevkitM-1 搭载 WiFi 和蓝牙模组，具有两个 USB 口，一个用于调试，另一个是 USB-UART。但是由于乐鑫没有自己的 IDE，开发难度比较高。因此选用 Arduino 的开发板。

Arduino 是开源硬件和软件公司，并且 Arduino 与 Intel、Arm、Microchip 等供应商合作，在技术应用兼容方面基本不存在问题。

由于 Arduino 拥有自己的 IDE 以及用户极大的社区，拥有较完备的教程和较丰富的问题解决方案，可以对开发编程降低难度，而且 Arduino 也对友商的模块如乐鑫的 ESP32 一同融合到 IDE 中，进一步降低开发难度。在 IDE 方面，Arduino 的 IDE 可以用 C++语言进行开发，也可以采用 C 语言的基础语法进行编程，并且不需要深入理解的单片机基础和编程基础，能够快速进行开发。

2.3.2 Arduino 开发板

Arduino 的 IoT 开发板 MCU 分为两类，一种是 Atmega，另一种是 Cortex-M0®+ 32。两者都是低功耗类型的 MCU。

1、 Atmega

Arduino 的 IoT 系列开发板的 MCU 是来自 Atmega 公司提供的 Atmega 系列和

Comtex 系列。Atmega 系列单片机，型号有 Atmega48A/88A/168A/328。Atmega 系列是 AVR 指令集的一直单片机。AVR 指令集一种高速 8 位精简指令（RISC 指令集）。该系列的 Flash 是 4/8/16/32KB，SRAM 是 512/1K/1K/2KB。它还有 256/512/512/1KB 的 EEPROM。不同的电压它的速率分别为 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5.V, 0 - 20MHz @ 4.5 - 5.5V，它的工作电压也在 1.8 - 5.5V。

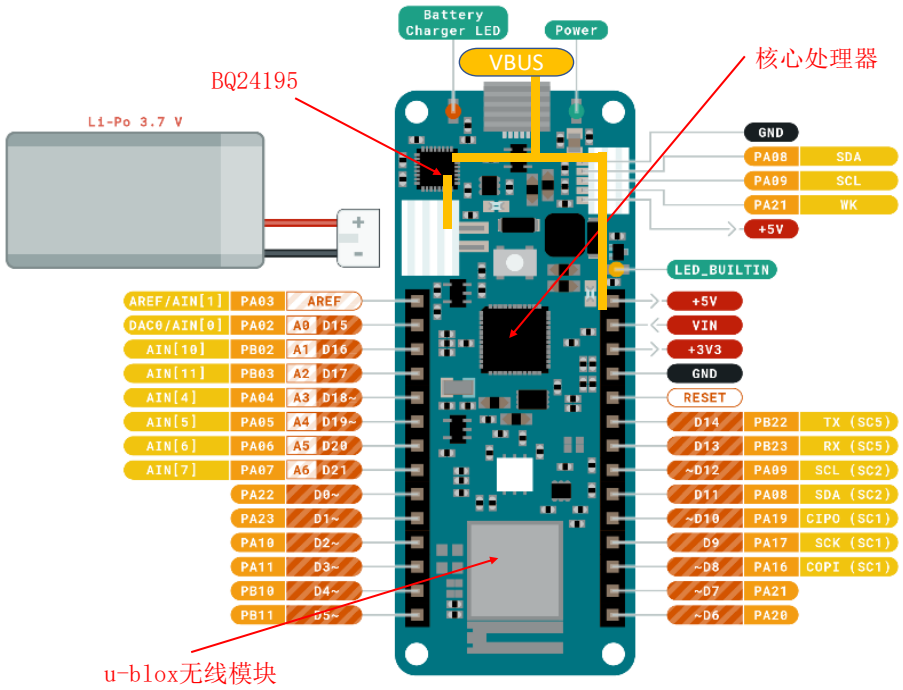

图 2.2 Arduino MKR WiFi 1010 引脚图

2、 Cortex

ARM 架构的 Cortex 系列。运用在 Arduino 的 IoT 功能的是 Cortex-M0®+ 32。它的 Flash 是 32/64/128/256 KB，SRAM 是 4/8/16/32 KB。最高可达到 48MHz。工作电压在 1.62~3.63V[5]。Cortex-M0®+ 32 位置在图 2.2 红色箭头指示的位置

对比 8 位的 Atmega，32 位的 Cortex-M0 处理速度更快，但是内存也比较小。在外围设备（Peripherals）方面, Cortex-M0 却更加丰富。

Arduino MKR 1010 开发板的微处理器是 SAMD21 Cortex-M0®+ 32 位微处理器。Arduino NaNo 33 IoT 也是用 SAMD21 Cortex-M0®+ 32 位低功耗 ARM 单片机。

本文选择 Cortex-M0 芯片的 MKR 系列的 Arduino MKR WiFi 1010 型号。MKR 系列开发板来自于 Microchip 提供的 SAMD 板，并且拥有 256KB 的 Flash 和 32KB 的 SRAM。

该型号的嵌入式开发板和 URG-04LX-UG01 激光雷达都是体积比较小型的（20.5mm×57.5mm×10mm）。

Arduino MKR WiFi 1010 嵌入式板可以通过板载的 u-blox 无线模块进行蓝牙和 WiFi 无线通信。板载的 u-blox 的 WiFi 模块支持 IEEE 802.11b/g/n 标准，可以实现 2.4GHz 频段 72Mbps 的传输速度。MKR 可以通过 WiFi 模块实现移动热点 AP 功能。网络设备可以连接到 MKR 的 AP 网络上，由 MKR 进行 IP 地址分配。蓝牙模块可以实现低功耗模式进行 1Mbps 的传输。

MKR 系列中开发板仅有一个全速的 USB 端口且能用于托管电源适配器和连接 USB 设备。而 IoT 系列中的 Zero 和 M0 有两个型号的开发板有 USB 端口，一个 USB 端口是编程调试端口，用于编程调试，另一个是带有 USB OTG 的 USB 端口，可以用于托管电源适配器和例如 Xbox 控制器的 USB 设备。

MKR 同时拥一个 I2C 接口和一个可以外接 3.3V 最低 1024mA 锂电池的 JST 接口。对与 JST 接口连接的电源进行管理模块的是 BQ24195 芯片，它使得 MKR 的电池接口可以通过 MKR 对电池进行充电，也通过电池对 MKR 进行供电。下图 2.3 来源 Arduino MKR WiFi 1010 datasheet，图 2.3 中显示了 Arduino MKR WiFi 1010 各模块需要的电流值，该开发板最高需要 5V600mA 的电流进行运转。

图 2.3 Arduino MKR WiFi 1010 模块供电图

MKR 板的问题是无法配置时钟速率和 USB 设备，无法对 MCU 的编程和更改程序，这就限制了 MKR 更加深层次的开发。

2.4 电源

URG-04LX-UG01 激光雷达的电源需要 5V500mA，和 Arduino MKR WiFi 1010 的 USB 所输出的电源相适应，可以进一步简化电路连接。

锂电池由于其轻便以及较高的放电倍率而选用。锂电池的单节电压为 3.7V，在市面

上用“S”表示一整电池由多少块锂电池串联，用“P”表示由“S”串联组成的电池分为一个单元。“3S1P”表示该电池仅有 1 个单元的 3 块锂电池组成，而“5S2P”表示该电池有 2 单元并联，每个单元有 5 块锂电池串联。放电倍率在市面上用“C”进行表示，放电倍率的高低是电池放电能力指标，放电倍率与电池容量的乘积值表示该电池最大输出电流值。由于 URG-04LX-UG01 激光雷达需要 5V500mA 的供电以及 Arduino MKR WiFi 1010 也需要进行供电，因此放电倍率和电池容量乘积值不能够低于 1100mA。

根据附录附录 A 的电池参数表，本文选择 ACE 1000mAh 15C 11.1V 3S1P 电池，因为该电池一方面价格便宜，另一方面和 Arduino MKR WiFi 1010 大小相近，适宜 URG-04LX-UG01 激光雷达的小巧体积使用，并且可以为 Arduino MKR WiFi 1010 和 URG-04LX-UG01 连续工作一天。

由于 ACE 800mAh 15C 11.1V 3S1P 电池的电压较高，需要进行降压处理来提供 5V 电源为 Arduino MKR WiFi 1010 中关于 USB 接口的 VBUS 供电，因此输出 5V 电压的电压模块是必不可少的。电压模块分为两类去电池电路（BEC）和外置去电池电路（UBEC），工作原理也分为线性稳压器和开关稳压两类。

去电池电路（BEC）是早期取代电池直接供电而采用间接供电的电路，节省了电池需求。外置去电池电路（UBEC）将设备中的内置 BEC 独立出来单独为电子设备供电，提供更高的输出功率。

线性稳压器主要采用稳压管或者稳压器来稳压。这种方式的功率消耗来自输入和输出电压差与流过稳压器的乘积，因此这种方式的问题是会导致功率消耗高，发热。若是输入输出电压差过大或者输出电流过高，会使得发热严重导致 BEC 关闭。

开关稳压的原理是斩波电路。斩波电路通过调制脉冲占空比，进行电压升降。但是由于这种方式类似开关的方式，会产生电磁波，从而干扰无线通信。图 2.4 中，输出电压 uo(t)的平均值 Uo 取决于开关 ton 和 toff，保持开关频率为一个恒定值，就能控制输出电压平均值，这种方法也称为脉宽调制方法。

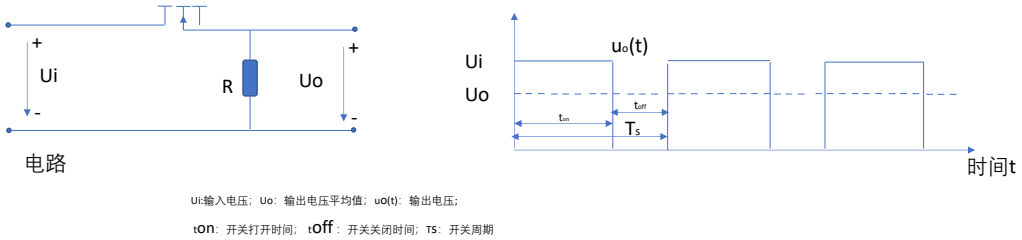

图 2.4 直流斩波原理图

本文采用 2.5 图的电源模块进行 5V 供电，该模块可以加入 6S 的电池输出 5V 电压和最大 1.5A 的电流。

图 2.5 BEC 电压模块图

2.5 硬件连接方案

本节对 Arduino MKR WiFi 1010 和 URG-04LX-UG01 进行通信硬件连接设计。URG-04LX-UG01 使用的 SCIP 通信协议支持 USB 通信，因此需要 Arduino MKR WiFI 1010 进行 USB 配置。

由于 Arduino MKR WiFi 1010 嵌入式板和 Hokuyo URG-04LX-UG01 激光雷达同是 USB 设备，而不是 USB 主机，因此需要 USB OTG 线进行连接（原因会在本文 3.3.1 节 USB 原理解释）。Arduino MKR WiFi 1010 具备 USB micro 接口，该端口 5 针脚，其中一针脚是对于 SAMD 板是 OTG 针脚，对于 USB 接口就是 ID 针脚。通过引脚接地将原本 USB 协议中识别为 USB 设备（USBDevice）的 MKR，识别为主机（USBHost）。

OTG 的功能使得原本的 USB 设备识别为主机，在本文中 Arduino MKR WiFi 1010 作为主机，通过 OTG 线缆将 ID 引脚接地。Arduino MKR WiFi 1010 嵌入式板识别为主机后，需要获取 Hokuyo URG-04LX-UG01 激光雷达的设备描述符才可以进行通信。也就是 Arduino MKR WiFi 1010 嵌入式板还需 USB 驱动。在本文中采用的 CDC 类的 ACM 描述符进行驱动 Hokuyo URG-04LX-UG01 激光雷达。本文采用 ACM 进行驱动激光雷达，对 MKR 进行 ACM 配置，实现 MKR 和激光雷达的通信。

USB 主机设置后，就是供电问题，USB 主机给 USB 设备 供电需要根据 USB 版本进行供电，由于 MKR 是全速（Full speed）的 USB2.0,版本，因此需要 5V 电压输入到 USB 接口。根据需要 Microchip 给出的 SAMD 板 USB 主机/从机供电图进行如图 2.6 的设置[6]。

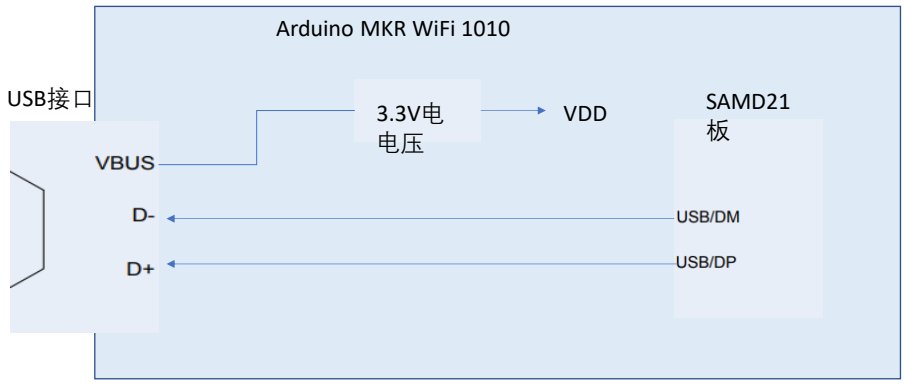

图 2.6 Arduino MKR WiFi 1010 主机模式供电图

这种连接不同于 MKR 系列的另一种型号为 Arduino NaNo 33 IoT 的开发板，NaNo 33 IoT 实现 ID 引脚接地后，USB 中的 VBUS 供电需要进行飞线并且进行焊接。而 MKR 电路板中的引脚分布如图 2.2，USB 接口的 VBUS 与图中的+5V 接口相连，由+5V 引脚外接 5V 电压模块进行供电[6]，电压模块正极连接图 2.6 示的+5V 引脚（由于是 USBHost，所以原本作为 USBDevice 的+5V 输出的变成输入，且 I2C 接口也可以作为主机连接外设，并输出+5V 电压）而负极连接 GND 引脚。值得注意的是 MKR 可以用 JST 接头外接 3.7V 最小输出 1024mA 电流甚至最大 6V 的锂电池供电，然后通过 BQ24195 升压芯片给 VBUS 输出 5V/1A 或者 5V/2A，并且可以进行编程进行操作（根据 BQ24195datasheet）。本文实现 USBHost 采用通过+5V 引脚接电源而不通过外接电池的 BQ24195 进行供电，因为 BQ24195 既可以通过 VBUS 进行给电池充电，也可以通过电池给 VBUS 供电，而通过+5V 引脚供电相当于充电。同时 MKR 的原本进行输出的+5V 引脚也连通 VBUS。故方法可行。

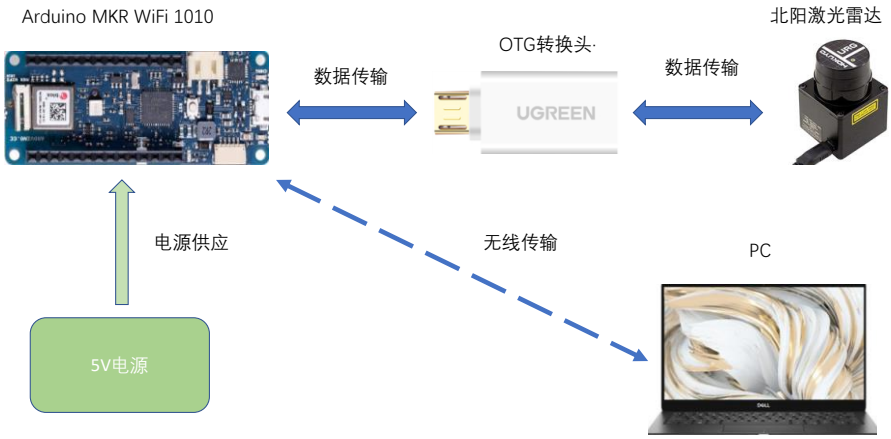

图 2.7 实物硬件连接示意图

在物理层实现 USB OTG 后，MKR 在 USB 协议中的地位也就是 USB 主机，但是仍然需要 USB 软件支持。USB 之间进行通讯需要两种设备，USB 主机和 USB 从机，USB 主机获取 USB 从机描述符，然后进行配置，找到支持 USB 从机功能实现的驱动。图 2.7 是无线激光雷达硬件连接完成的示意图。

在与个人计算机进行 WiFi 无线通信方面，本硬件设计可以通过调用 MKR 的 WiFiNiNa 库，配置 TCP，然后通过 WiFi 实现 PC 和 MKR 之间的通信，即个人计算机向 Arduino MKR WiFi 1010 发送 SCIP 指令，然后 Arduino MKR WiFi 1010 通过 USB 向将收到的指令激光雷达转发，而激光雷达通过 USB 和 WiFi 模块向个人计算机返回数据。

3 系统软件设计与实现

3.1 引言

根据硬件需要的进行的软件设计，也是基于北阳激光雷达的 SCIP 通信协议实现的。通过向激光雷达发送指定的指令，然后将激光雷达返回的数据进行数学建模和成像。

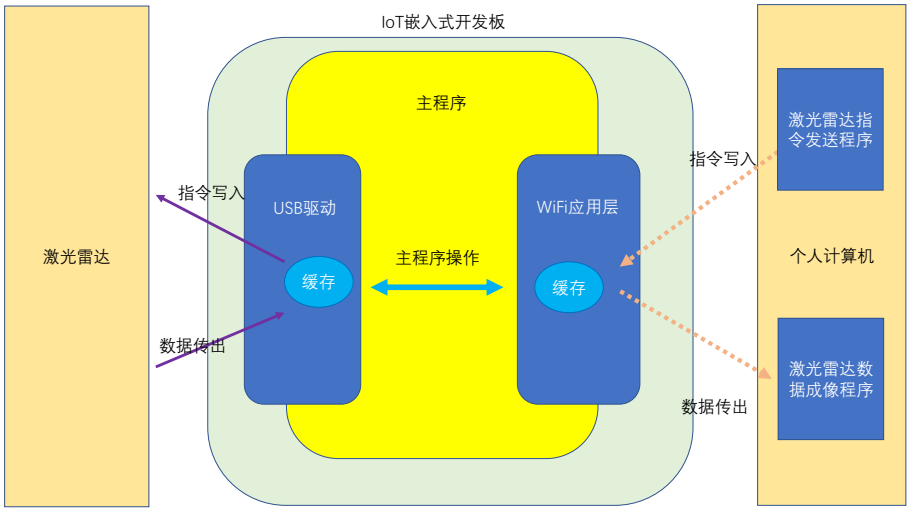

图 3.1 软件系统设计图

软件系统经过包括 WiFi 连接以及 USB 驱动在内的初始化，构建起激光雷达和个人计算机通信的通道。

图 3.1 展示的是系统软件设计图。个人计算机发送的指令通过 IoT 嵌入式开发板的 WiFi 模块和 USB，发送到激光雷达，激光雷达返回的数据根据路径返回到个人计算机中。主程序不仅构建 USB 驱动和 WiFi 应用层，而且在两者之间进行数据搬运。

根据 SCIP 协议，激光雷达根据发送的指令进行相应的状态变换从而实现相应的功能，因此需要设定好相关 SCIP 的指令进行控制。第一步是建立起个人计算机和激光雷达的 WiFi 无线连接，根据返回的状态进行判断是否连接成功并且返回激光雷达的型号和序列号。建立起连接后进行时钟同步，在测量的数据上加上时间戳，使得数据处理的顺序不会由于通信状态不好而导致错乱。

3.2 开发环境

对 Arduino MKR WiFi 1010 进行开发需要进行 IDE 的安装。在 Arduino 官网上进行 IDE 软件下载的版本分为离线版本的 1.8 版本 IDE 和 2.0 版本 IDE，以及可以再网页上进行开发的线上 IDE。离线的 1.8 版本 Arduino IDE 也可以在微软应用商店下载。本次实验使用的是微软应用商店下载的是离线 1.8 版本 IDE。

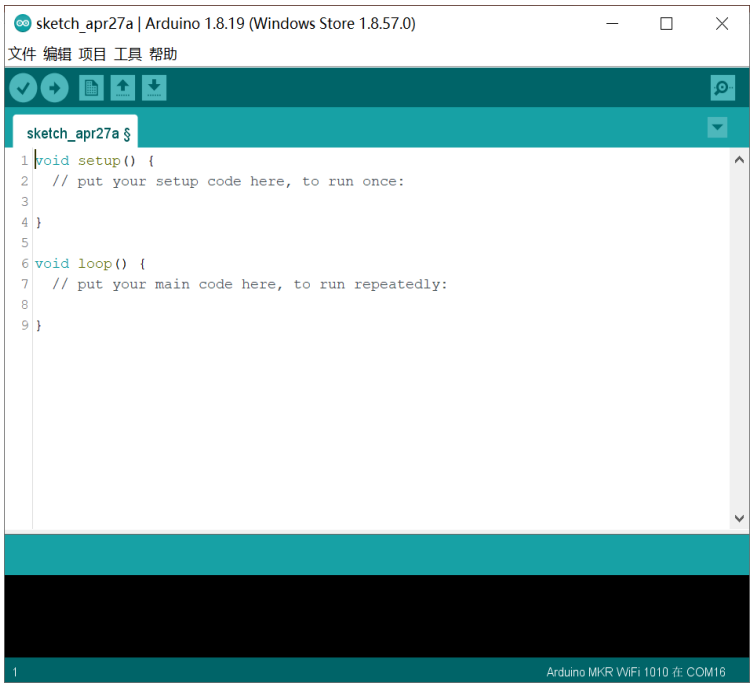

图 3.2 Arduino IDE 编程页面

图 3.2 是打开 IDE 后的编程页面。下载后，需要选择对应的嵌入式开发板进行安装驱动包。在图 3.3 中也变上栏点击“工具”可以显现出开发板选项。点击“开发板管理器”打开开发板管理器进行下载相应的开发板驱动。 根据不同型号的板需要安装不同的驱动。

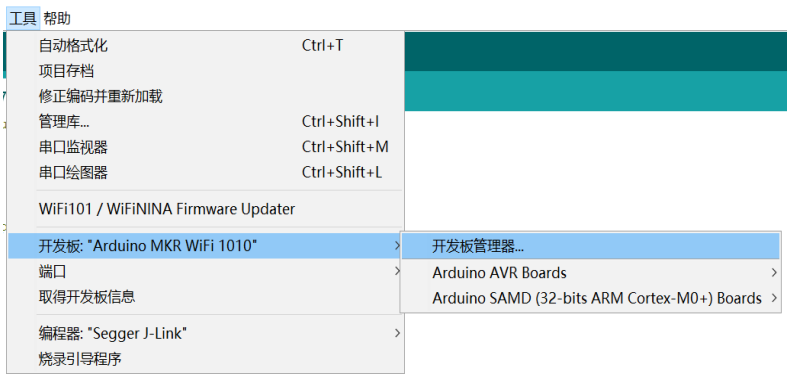

图 3.3 开发板选项示范图

由于需要进行 USB 以及 WiFi 开发，并且本文选择的是 SAMD 板。在开发板搜索栏写入“SAMD”，选择 Arduino SAMD Boards 进行安装。图 3.4 为已经安装完毕后的界面。

图 3.4 开发板选择页面

图 3.5 展示的是如何选择开发板型号。等待安装驱动完成之后会出现“Arduino SAMD (32-bits ARM Cortex-M0+ Boards”选项，点击该选项后会出现开发板型号。本文使用的是 MKR WiFi 1010，因此点击“Arduino MKR WiFi 1010”选项。

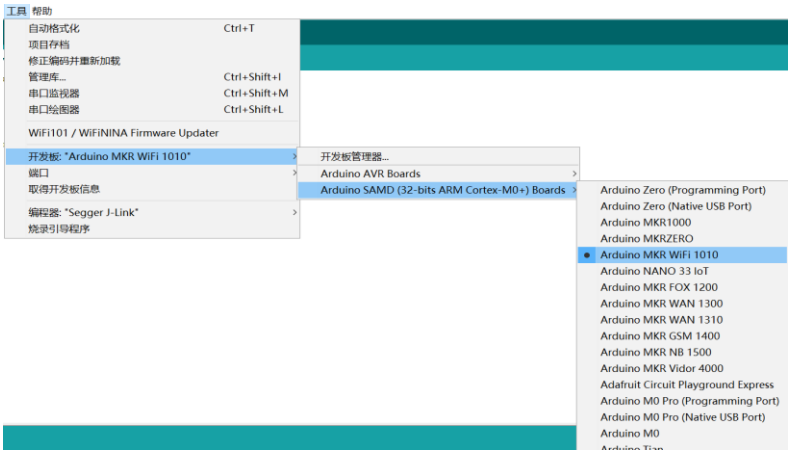

图 3.5 Arduino MKR WiFi 1010 选择示范图

选好之后可以点击上栏的“文件”打开示例，如图 3.6 所示。根据需要开发的功能查看示例，熟悉 Arduino 嵌入式开发板的使用。

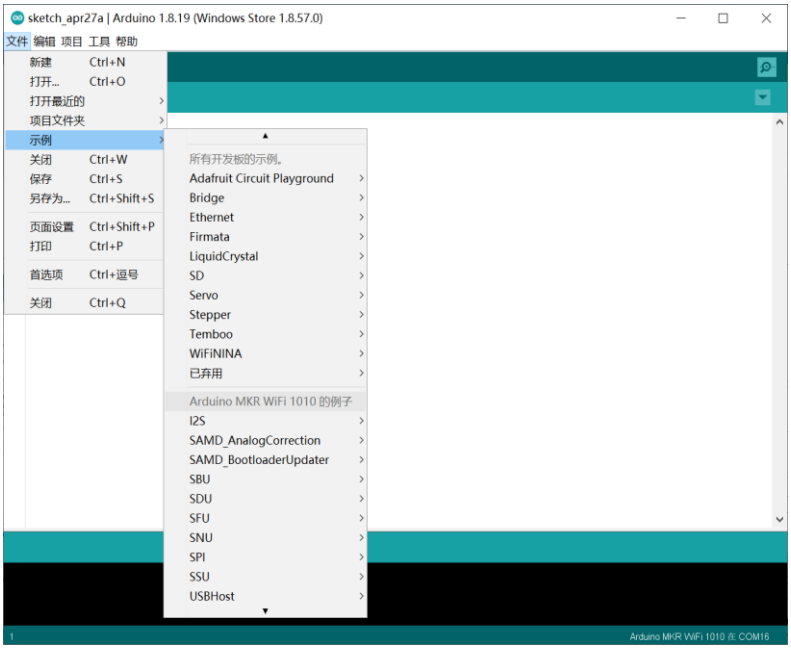

图 3.6 IDE 示例位置图

对 USB 通信和 WiFi 通信进行开发需要加载 USBHost 库和 WiFiNINA 库，图 3.7 的展示了如何进行加载操作。但是由于 Arduino 的 USBHost 只有键盘和鼠标的驱动，而没有对通信设备类（CDC）的驱动，因此需要根据 USBHost 库进行修改。

将适用于 SAMD 板的移植于 Arduino USB Host Shield 的 USBHost 库文件覆盖原本的 USBHost 文件作为示范（Demo），再根据移植的 CDC ACM 驱动进行修改，使得 CDC ACM 可以驱动 URG-04LX-UG01。

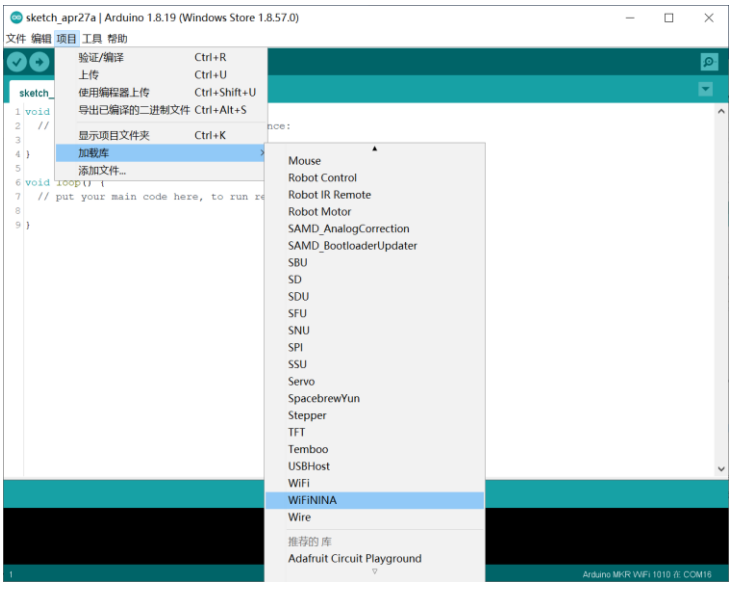

图 3.7 IDE 加载库位置图

Arduino 的库可以从上图 3.7 中进行查看。以 Win10 为例，Arduino 库存储在电脑的位置是在“User\Documents\ArduinoData\packages\arduino\hardware\samd\1.8.13\libraries。”在需要使用库的时候可以如上图 3.7 进行加载库。

3.3 USB 通信原理与实现

3.3.1 USB 原理

上一章，讲述了选定的设备以及硬件的连接方案。根据连接方案，本节是实现激光雷达和 IoT 嵌入式开发板间的通信。IoT 嵌入式开发板 Arduino MKR WiFi 1010 和激光雷达 Hokuyo URG-04LX-UG01 通过 USB OTG 线简单相连便可实现激光雷达无线通信的功能。但是激光雷达和嵌入式板进行最为关键一点就是 USB 通信。USB OTG 线将 Arduino MKR WiFi 1010 进行连接之后，嵌入式板 Arduino MKR WiFi 1010 如何实现 USB 通信。

在日常生活中将 U 盘直接插入个人计算机，便可以在个人计算机上进行读取，但是将 U 盘插入嵌入式开发板却无法直接对 U 盘进行读取。原因不是嵌入式板缺少操作系统或者嵌入式板的 MCU 性能比较弱，而是 USB 的原理使然。在早期，手机需要连接电脑才可以进行照片传输，而且手机无法直接插入 U 盘对照片进行读取。同时手机转移数据需要 SD 卡，而不是将数据转移至 U 盘。同一时代的 MP3 也是采用 SD 卡而不是 U 盘。因此为了解决 Arduino MKR WiFi 1010 通过 USB 实现和激光雷达传输的问题，本节内容为介绍 USB 通信的原理，包括 OTG、USBHost 以及 USB 的传输。

USB 全名为 Universal Serial BUS，通用串行总线，其中串行接口（Serial）采用 串行方式通讯，其中包括 RS-232 接口在内。串行接口将数据为一位一位按顺序发送。而 USB 是广泛应用的通讯标准中的一种，并且广泛应用在个人计算机中。但是 USB 设备一旦脱离个人计算机后也就无法正常工作。值得一提的是 USB 不需要设置波特率，但是需要配置时钟。根据通讯速率的不同,需要进行配置，低速在 1.5Mbit/s 至少需要 32KHz，而全速、高速需要至少 8MHz 的频率。

USB 通讯传输的方式主要有三种，分别为控制传输、中断传输和同步传输。

控制传输是双向传输，将一次传输分三段，也可以描述成三个阶段。第一阶段是主机和从机初始化配置，也就是接收描述符。第二阶段是用于特殊设备的配置。第三阶段

是根据状态进行传输，也就是主机发送数据或者接收数据。

中断传输是单向的轮询传输。主机每隔一段中断时间就对中断端点进行查询状态，若有数据传输就接收发送数据。

同步传输是实时的单向传输，并且没有握手操作，不保证数据的准确性。

目前 USB 已经发展到了 USB4.0 版本，其接口由 Micro、Mini 等形状逐渐统一为 Type-C 形状，消除由于接口不统一而无法物理连接的问题。同时 USB 也出现了 OTG 功能，使得 USB 设备也可以进行通信。

OTG 全名 On the Go。可以将外设识别为主机，通过 USB 接口 ID 针脚接地识别为主机，并且不影响个人计算机是否为主机。

USB 协议以及端口解决的个人计算机和外围设备的交互，通过 USB 线对 PC 和外围设备进行连接，然后通过个人计算机端的驱动就可以实现个人计算机和外围设备的交互。在 USB 协议中，有驱动的个人计算机作为 USB 主机（USBHost），而外设作为从机。主机和从机的区别是，拥有驱动的主机对外设进行管理的分配通信令牌，而从机不可以主动向主机或其他从机进行通讯。两个普通 USB 设备无法识别主机和从机，因此当脱离 PC 后，其他外设之间就不可以进行交互。

USB OTG 使得外设也可以作为主机，OTG 的端口有 5 个针脚如图 3.8 所示，通过 ID 针脚接地来识别主机。原本是从机的外设通过 OTG 的 ID 针脚接地被识别为主机，但是识别为主机后没有 USB 驱动同样无法和其他外设进行通讯。USB 协议需要识别外设的描述符、解析数据包等才可以通讯，但是由于没有驱动，主机无法构建通讯通道，因此无法进行通讯。

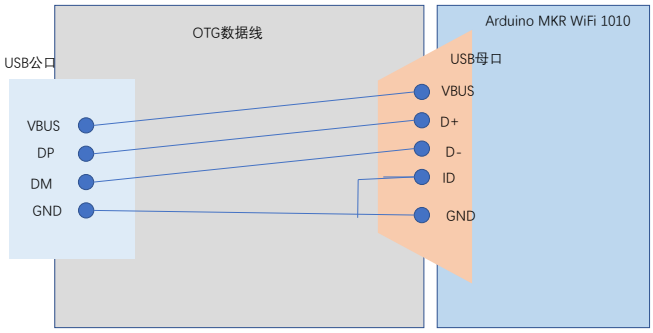

图 3.8 OTG 物理连接示意图

由于嵌入式开发板一般没有操作系统进行支持，实现 OTG 功能还需要烧录 USB 驱

动。OTG 使用的 USB 驱动与一般操作系统使用的驱动的基本功能并没有太大的区别，仅仅区别于 OTG 使用的 USB 驱动需要提前烧录，甚至嵌入式开发板只支持一种 USB 类设备，如 CDC 类进行驱动，而个人计算机的操作系统可以区分不同类的 USB 设备接入并且提供相应的驱动。同时 OTG 使用的 USB 驱动还需要处理嵌入式开发板识别为 USB 主机或者识别为 USB 从机的转换工作。USB 主机类的驱动程序并非由 USB 接口制造商提供，而是由芯片制造商提供，如 Arduino MKR 系列的 MCU 提供商 Atmel 为 Arduino MKR 系列提供相应的 USB 驱动。主机工作时，USB 主机栈工作，其中的主机控制器驱动程序负责 USB 主机栈与硬件端点的数据交换，USB 协议层负责处理 USB 协议规范[10]。设备类驱动程序的功能取决于该两用设备的功能。

USB 协议规定 USB 通信需要两种设备，一种设备是 USB 主机（USBHost），即个人计算机或者具备 OTG 的 IoT 嵌入式开发板，另一种则是 USB 从机（USBDevice），即具有 USB 接口的激光雷达等设备。USB 主机和 USB 从机之间的通讯需要 USB 主机驱动，主机要求从机描述符，从机收到命令后就向主机发送设备描述符。主机根据设备描述符进行识别配置，配置也需要配置描述符去获取接口和端点描述符，进行分配端点（Endpoint，EP）和通道（Pipe）以及 USB 包的大小（Packet）。在此之后驱动根据描述符中描述的 USB 设备的类，为设备的功能找到合适的程序进行支持。当完成配置后，主机通过 USB 包进行对通讯的调节和数据传输。接下来就是内容这些 USB 组成的概述。

在 USB 通信中，基本通信的传递信息是设备描述符和 USB 包这两种类型。前者是最开始完成配置工作时 USB 所传递的信息，后者是 USB 通信传递的基本单元。

下面是 USB 传输信息的概述：

1、 描述符

（1） 设备描述符

代码 设备描述符数据结构

```
typedef struct {
    uint8_t     bLength;                //以字节为单位的设备描述符长度
    uint8_t     bDescriptorType;        //设备描述符类型
    uint16_t bcdUSB;                    // USB 版本号,如 USB3.0
    uint8_t     bDeviceClass;           // USB 设备类型
    uint8_t     bDeviceSubClass;        // USB 设备子类
```

续代码 设备描述符数据结构

```
    uint8_t     bDeviceProtocol;          //设备协议
    uint8_t     bMaxPacketSize0;          // USB 端点 0 最大包大小
    uint16_t idVendor;            //厂家标识
    uint16_t idProduct;           //设备 ID
    uint16_t bcdDevice;           //设备版本号
    uint8_t     iManufacturer;            //厂家描述字符串索引
    uint8_t     iProduct;                 //产品描述字符串索引
    uint8_t     iSerialNumber;            //串行描述索引
    uint8_t     bNumConfigurations;       //可配置数
} USB_DEVICE_DESCRIPTOR;
```

USB 通过类进行描述了设备的描述，USB 的类也可以与接口描述符关联。

bcdUSB 表示的是 USB 规范发布版本编号，它采用以 BCD 码进行表示。这个字段确定了设备及其描述符所遵循的 USB 规范发布版本。

描述符字符串索引是用字符串描述一个设备的一些属性，通过人类使用的字符描述设备的属性，描述的属性包括设备厂商名字、产品名字、产品序列号、各个配置名字、各个接口名字以及用户自己定义的字符串。

（2） 配置描述符

代码 配置描述符数据结构

```
typedef struct {
    uint8_t bLength;                  //描述符长度
    uint8_t bDescriptorType;          // 描述符类型
    uint16_t wTotalLength;            //配置返回的数据总长度
    uint8_t bNumInterfaces;           //配置支持的接口数量
    uint8_t bConfigurationValue;      //配置值
    uint8_t iConfiguration;           //字符串描述符索引
    uint8_t bmAttributes;             //配置特性
    uint8_t bMaxPower;                //设备从总线获取的最大功耗
    } USB_CONFIGURATION_DESCRIPTOR;
```

配置描述符说明了一个特定配置的相关信息。取得设备描述符后，主机就可以继续去获取设备的配置、接口和端点描述符。当主机请求配置描述符时，返回的是所有相关的接口和端点描述符。

其中值得注意的是 wTotalLength 配置返回的数据总长度，它包括该配置返回的所有描述符的总长度，。

对于 USB2.0，bMaxPower 以 2mA 为单位。如果设备要求 200mA，则 bMaxPower 等于 100 (0x64)，设备可请求的最大总线电流 500mA。对于增强型超高速(Enhanced SuperSpeed)设备，bMaxPower 以 8mA 为单位。如果设备要求 200mA，则 bMaxPower 等于 25(0x19)，设备可请求的最大总线电流 900mA。当设备和主机支持 USB Power Delivery Rev. 2.0，主机可以从 PD Class Specific Descriptors 检索设备的电源需求。

（3） 接口描述符

代码 接口描述符数据结构

```
typedef struct {
    uint8_t bLength;                //以字节为单位的描述符大小
    uint8_t bDescriptorType;        //接口描述符类型
    uint8_t bInterfaceNumber;       //接口的编号
    uint8_t bAlternateSetting;      //用来确认接口编号的代替设置编号
    uint8_t bNumEndpoints;          //接口使用的端点数量
    uint8_t bInterfaceClass;        //接口的类
    uint8_t bInterfaceSubClass;     //子类
    uint8_t bInterfaceProtocol;     // 协议
    uint8_t iInterface;             //接口字符串描述符的索引
} USB_INTERFACE_DESCRIPTOR;
```

接口描述符描述的是接口的信息，如 USB 接口类型，同时接口描述符对接口所支持的端点进行描述，例如 Arduino MKR WiFi 1010 仅仅支持 8 个端点，接口描述符中的端点数量为 8。

同时接口描述符也对接口的类进行配置，在 bInterfaceClass 中，类有 CDC 接口、HID 接口等。接口描述符根据这些类进行配置。

（4） 端点描述符

代码 端点描述符数据结构

```
typedef struct {
    uint8_t bLength;                //描述符长度
    uint8_t bDescriptorType;        //描述符类型
    uint8_t bEndpointAddress;       //端点地址
    uint8_t bmAttributes;           //端点属性
    uint16_t wMaxPacketSize;        //端点承载的最大包大小
    uint8_t bInterval;              //查询端点进行数据传输的间隔
    } USB_ENDPOINT_DESCRIPTOR;
```

端点描述了 USB 规范定义的端点信息。每一个端点都有自己的端点描述符。主机

端获取端点描述符 (Endpoint Descriptor)，总是作为配置描述符的一部分返回，不能直接用 Get Descriptor 或者 Set Descriptor 请求访问。

2、 USB 包（USB Packet）

USB 包（Packet）是 USB 通信的主要单元，并且分为四类。用于区分包（Packet）的类型的包标识 PID 分为令牌包（Token 包）、数据包、握手包和特殊包。图 3.9 是 USB 包的基本构造图，它由同步域、包标识（PID）、数据和 EOP（包尾）四个部分组成。同步域用于主机同步 USB 设备的时钟，并且向 USB 设备表示准备数据传输。EOP（包尾）是包的结束符。

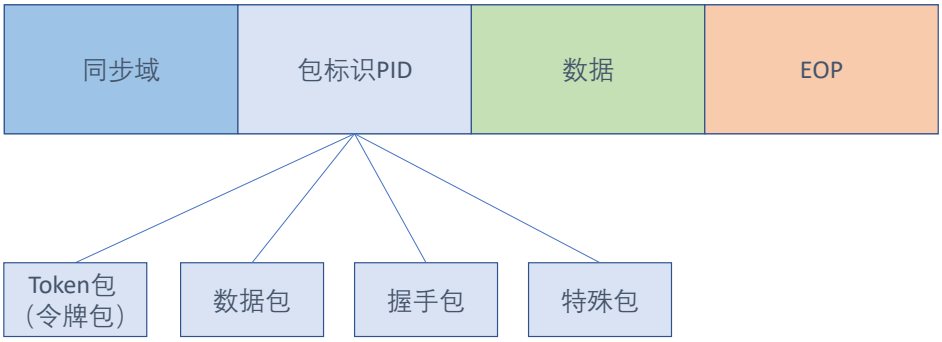

图 3.9 USB 包一般结构图

（1） 令牌包（Token 包）

令牌包是主机进行传输用的状态标识以及添加时标的 SOF 功能。Token 包发送的是 IN 令牌表示，则主机接收数据，OUT 令牌表示是主机发送数据。图 3.10 中 IN 和 OUT 的 7 位地址意味着 USB 总线最大只能挂载 127 个 USB 设备，4 位端点号表示最大 16 个端点号。

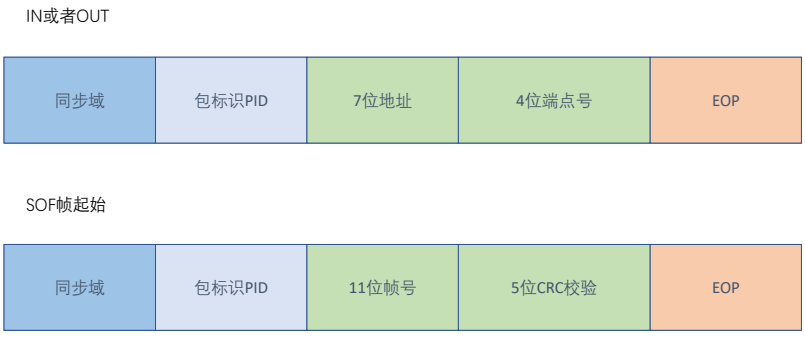

图 3.10 令牌包结构图

SOF 全名 Start of Frame，即起始帧，SOF 时标标记的帧存储在设备帧编号寄存器（FNUM.FNUM）的帧编号字段中，每一个设备都能接收到 SOF 包，判断当前时间。CRC 是 5 位循环冗余码。

（2） 数据包

数据包结构如图 3.11 所示。数据包的数据以字节为单位，存储的最大字节数由配置描述符而定。低速、全速和高速一般的最大包字节分别为 8、64 和 1024。

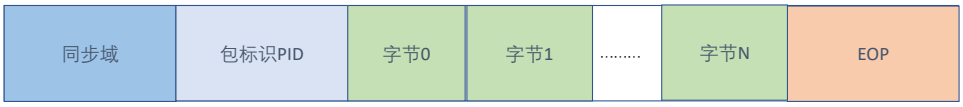

图 3.11 数据包结构图

（3） 握手包

握手包的用于返回传输的状态，它的结构如图 3.12 所示。握手包同样根据 PID 分为确认字符（ACK）包、非应答（NAK）包和 STALL 包。ACK 包表示正确接收数据，NAK 包表示没有数据发送或者接收，STALL 表示传输出错。

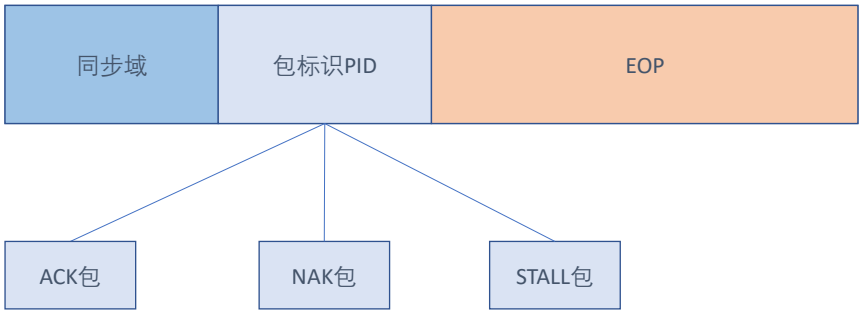

图 3.12 握手包结构图

（4） 特殊包

特殊包用于一些特殊的功能，如通知集线器打开低速端口。它的结构如图 3.13 所示。

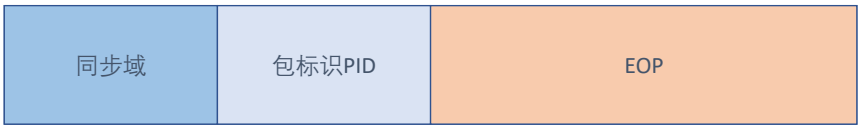

图 3.13 特殊包结构图

在 USB 通信的信息了解之后，接下来的内容便是这些信息的起点和终点。USB 对描述符和 USB 包的接收和发送受到 USB 驱动程序的管理和处理。USB 驱动程序根据描

述符识别 USB 设备的类型，然后为 USB 设备提供相应的程序支持，而驱动程序最基础的功能是发送和接收 USB 包。同时 USB 驱动受到 USB 栈的管理，而 USB 栈为用户的应用程序提供支持。下列是对 USB 的类、驱动以及栈的介绍。

1、 USB 类（USB Class）

USB 定义类的代码信息，也就是描述符数据结构。然后 USB 根据这些信息描述的功能加载设备驱动程序。这些信息通常包含名为“基类”、“子类”和“协议”的 3 字节长度中。表 3.1 是部分类的基类和子类以及协议。

表 3.1 部分 USB 类表

基类	描述符位置	描述
00h	设备描述符	在接口描述符使用类信息
01h	接口描述符	Audio 音频
02h	设备和接口描述符	CDC 通信设备控制类
03h	接口	HID 人机交互设备
05h	接口	Physical

在一个设备中有两个位置放置类的代码信息，一个在接口描述符中，另一个在设备描述符。注意，某些类的代码只允许放在设备描述符，而有些只能在接口描述符

2、 USB 驱动（USB Driver）

USB 驱动（Driver）是一个允许硬件设备和电脑通信的文件。USB 驱动包含一定范围的电子条目，但是仅限于键盘、鼠标、照相机、监控器等等，也就是有这些设备的描述符结构.h 文件,就像 Arduino MKR WiFi samd 文件 library USBHost 中的那些头文件。

在个人计算机中 USB 驱动安装在注册表目录中。驱动程序的特定功能包括对操作系统的支持、加测各种数据和传感器属性的方法与兼容外围设备配合使用的能力。计算机的设备管理器提供有关程序的信息，如是否更新、删除或者重新安装驱动程序。

USB 驱动程序栈（USB Driver stack）是操作系统连接到 USB 控制器设备加载 USB 驱动的栈。USB 驱动的栈底部是 USB 主机控制器的驱动、USB 接口驱动、若干个微型接口驱动。当操作系统寻找到主机控制器硬件，微型接口驱动就会工作，然后接口驱动进行工作。而接口驱动处理主机控制器中的特立的协议部分工作。USB 主机控制器在 Window 操作系统中的可以找到。

关于 USB 驱动程序日期问题，Window 的 USB 驱动程序日期是 2006 年 6 月，但是这并不意味这驱动老旧，而是类似该驱动程序的纪念日期，并且 USB 驱动程序一直在更新。

3、 USB 栈（USB Stack）

USB 栈属于 USB 主机的 vendor class，也就是厂家类（vendor 在设备描述符出现过，即 idVendor），而且该类还包含另一类是它的同名类 vendor class。也就是说 USB host vendor class 有且仅有 USB stack 和 vendor class。与像 CDC 和 HID 等 USB 类不同的是，vendor class 是为先进 USB 开发商而设计的[7]。

USB 栈可以对 USB 进行 USB 主机与 USB 设备之间进行角色转换（通过 USB 设备 OTG 进行识别为 USB 主机）、对 VBUS 进行供电管理、USB 设备连接管理以及 USB 设备枚举管理。以 USB 主机栈为例，如图 3.14，它分成三个部分，USB 主机控制器、若干的 USB 主机接口以及 USB 驱动。

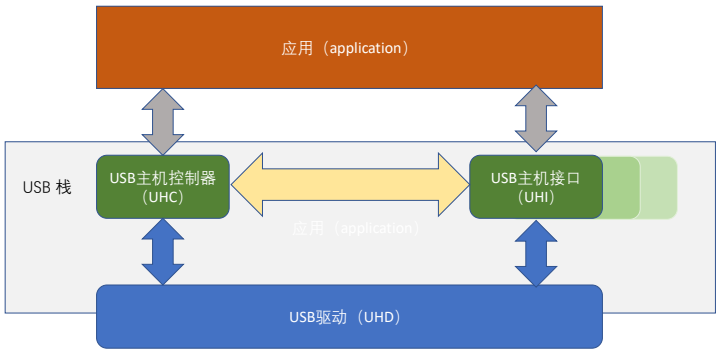

图 3.14 USB 栈结构图

USB 主机控制器可以对从接口接入的设备进行管理。又由于 USB 是拓扑结构，与 USB 主控制器连接的总线因此也称为根集线器（Root hub）。一台 PC 可有有多个 USB 系统，但每个 USB 只有一个 USB 主机控器及其根集线器。控制器还提供检测已经连接的设备数量、分配设备 ID、检测是否支持高速传输、发送 SOF、获取设备描述符等功能。

USB 主机堆栈中的所有 USB 主机接口（UHI）都基于 UHC，以支持 USB 枚举。USB 主机接口为不同的 USB 类如 CDC 类 HID 类的设备进行配置 USB 主机。

USB 驱动为控制器和接口提供程序支持。

在了 USB 解基本的软件的组成后，接下来的内容就是 USB 整体上通信的工作流程。

1、 抽象层传输

USB 设备的通信依靠端点间的通道完成如图 3.15。同时由于不同的 USB 速度，需要时钟的频率也不同，USB2.0 的速度主要分为低速、全速和高速，低速最高 1.5Mbit/s，全速最高 12Mbit/s ,高速可达 480Mbit/s。全速和高速至少 8M（百万）Hz，不相匹配的时钟频率会导致数据丢失[8]。

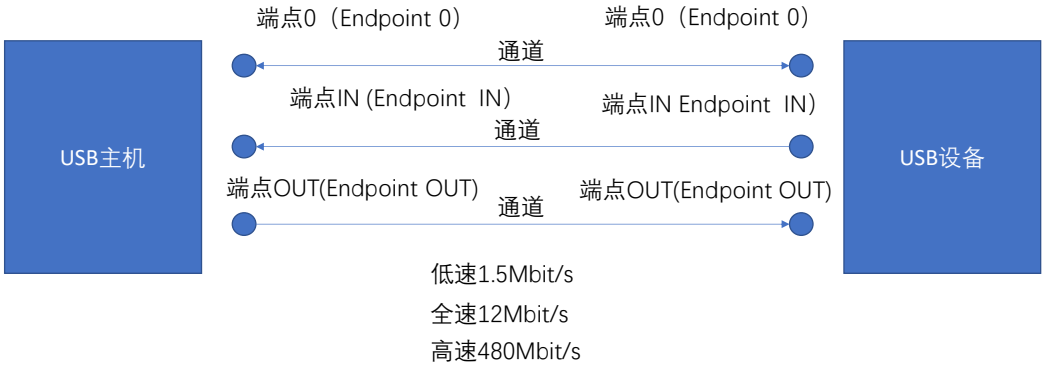

图 3.15 USB 通信概念图

端点的配置是通过端点描述符进行，但是由于 Arduino MKR WiFi 1010 设备自身的限制，它支持的端点数为 8。端点 0 是双向的，用于接收描述符初始化也可以用于传输数据，而其他端点在未完成配置之前是不可用的而且是单向的。图 3.15 中的 IN 和 OUT 端点分别用于主机接收和发送数据。

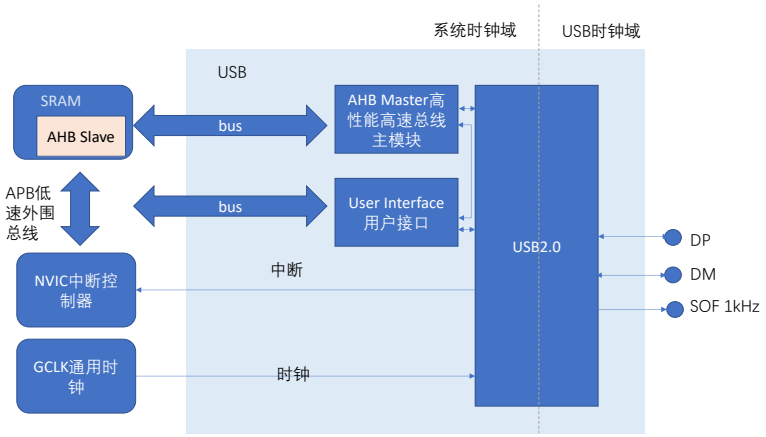

图 3.16 USB 工作概念图

在 CPU 内部，系统时钟域是 CPU 的时钟，而 USB 时钟域不同于系统时钟域，需要经过系统时钟分频或者外接时钟源进行输出。该图 3.16 展示的 CPU 和 USB2.0 进行的交互，USB 的数据在 SRAM 进行中继。而 AHB 系统由主模块和从模块组成。AHB 总

线上的传输的工作模式是从模块受到主模块支配，数据只能由主模块将数据传输出去，从模块进行数据接收。每隔一段时间，USB 主机都会发送 SOF 包进行时钟时间调整，以避免时钟问题导致数据丢失。USB 接收到的差分数据经过 USB2.0 传输到 SRAM 进行缓存。当发送时，数据也是从 SRAM 取出进行发送。

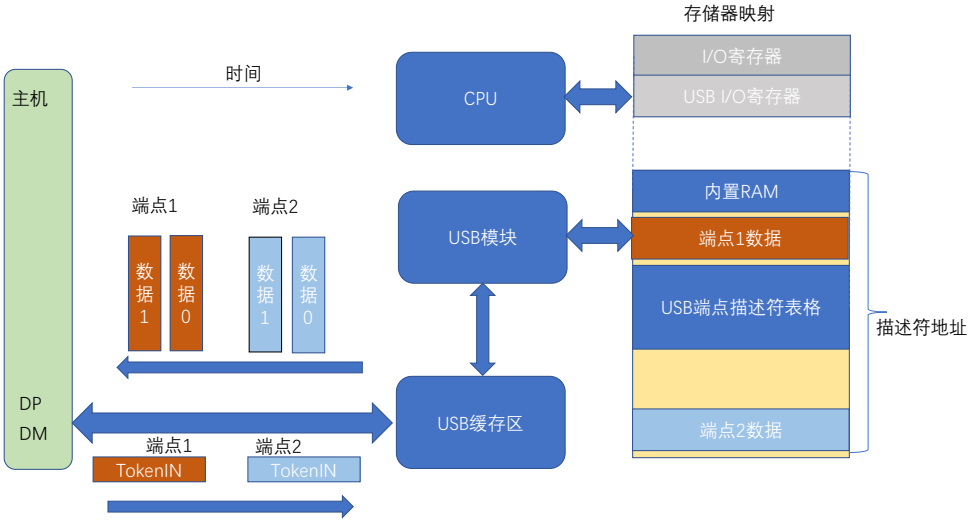

图 3.17 USB 主机发送数据概念图

该图 3.17 展示的是 In Intransfer，是主机接收从机发送的数据。CPU 对寄存器进行管理，进行 USB 的时钟和中断等配置。端点 1 和端点 2 接收到令牌包 Token IN 才开始工作，数据先进入 USB 缓存区，然后根据描述符的地址进入到相匹配的端点

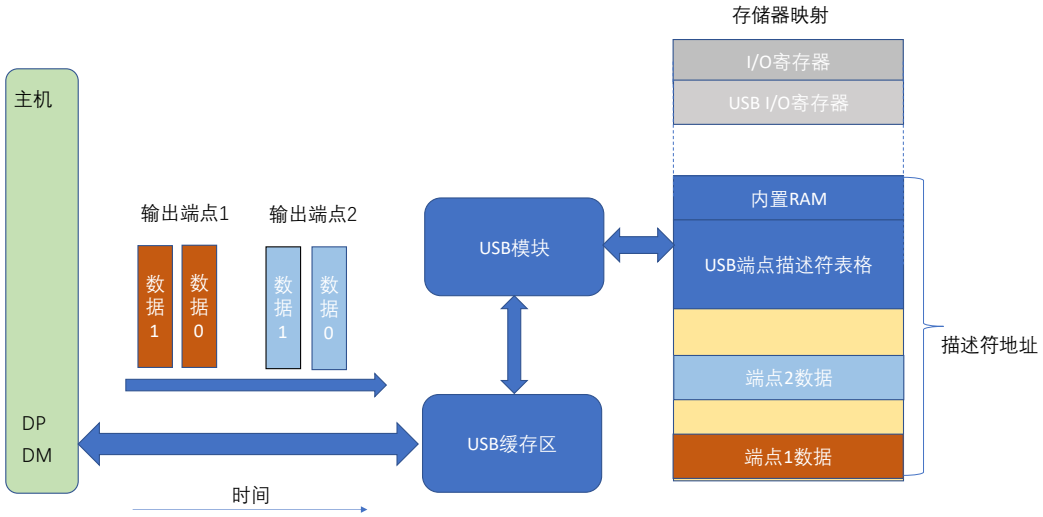

图 3.18 USB 主机接收数据概念图

该图 3.18 展示的是 Out Intransfer，主机向从机发送数据。主机从端点 1 发送数据，

经过从机的 USB 缓存区，然后进入 RAM，通过 USB 端点描述符进入从机的端点 1。由于从机受到主机的支配，从机没有 CPU 进行时钟配置，而是受到主机时钟的控制。

2、 物理层传输

在物理实现上是通过 FIFO（First In First Out）先进先出存储器实现的。它的结构如图 3.19 所示。USB 应用该存储器进行数据的读取和发送，并且 FIFO 存储器通过两边设备的 CPU 的控制，即 USB 主机和 USB 设备的 CPU。FIFO 当数据满和数据为空时会分别向两边发送满标志位和空标志位，这样就可以让数据不会丢失，并且让 CPU 合理使用空闲时间。

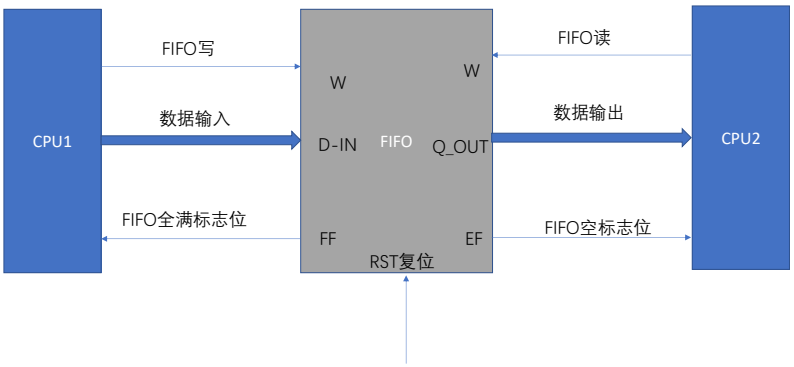

图 3.19 FIFO 存储器结构图

USB 一般采用两片 FIFO 存储器进行数据收发。通过交替存储数据的方式，实现不断接收或者发出数据，也就是乒乓球存储方式。乒乓模式的工作就像 3.20 所示的那样。

图 3.20 展示的了乒乓模式配置端点时，它同时使用输入和输出数据缓冲区在单个方向上配置端点，通过使用 Token IN 和 OUT 来选择方向。

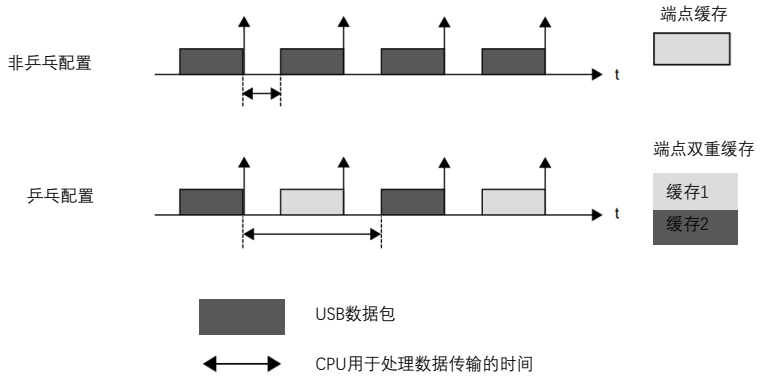

图 3.20 乒乓模式工作图

在非乒乓缓存时，由于只能才用单个缓存区缓存数据，CPU 处理时间短，容易丢包。而双重缓存可以缓存更多数据，预留更多时间给 CPU。

3.3.2 USB 通信实现

本文使用是 CDC ACM 实现 USB 通信。CDC 全称是 communication Device Class（通信设备类）。CDC 是 USB 定义的一类给各种通信设备使用的子类。它的组成如图 3.21 所示。CDC 有 3 个基本任务，设备管理是控制配置特定设备并通知 USB 主机某些事件，呼叫管理是建立和终止呼叫或其他连接，数据传输是发送和接收应用程序数据。平常使用的 USB 转串口使得的协议是 CDC 协议。CDC 按设备类型分为网络设备和电信设备。

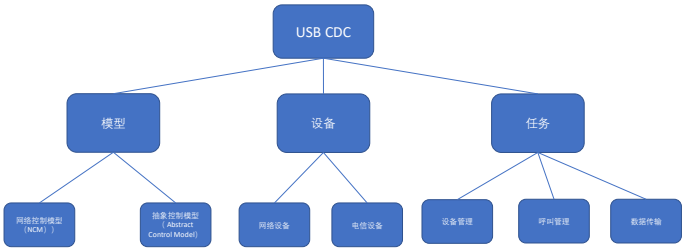

图 3.21 CDC 类组成示意图

网络设备主要指以太网适配器和集线器等网络拓扑设备，电信设备有数字电话以及 COM 端口、ISDN（综合业务数字模型）终端适配器，其中 ISDN 是用于交换机和传真机等设备。

表 3.2 CDC 描述符示例

ACM 描述符	NCM 描述符
标准设备描述符	标准设备描述符
标准配置描述符	标准配置描述符
接口关联描述符	接口关联描述符
CDC 标头功能描述符	CDC 标头功能描述符
CDC 联合功能描述符	CDC 联合功能描述符
呼叫管理功能描述符	CDC 以太网网络功能描述符
抽象 控制管理 功能描述符	NCM 功能描述符
CDC 类通信接口的标准接口描述符	CDC 类通信接口的标准接口描述符
中断 IN 端点的标准端点描述符	中断 IN 端点的标准端点描述符
CDC 类数据接口的标准接口描述符	CDC 类数据接口的标准接口描述符
批量输入和批量输出终结点的标准终结点描述符	批量输入和批量输出终结点的标准终结点描述符

按模型分类，CDC 有抽象控制模型（ACM）和网络控制模型（NCM）。ACM 可以虚拟 COM 端口，也可以通过 RNDIS（远程网络驱动接口规范）和以太网适配器和集线器连接[9]。NCM 是用来虚拟以太网适配器的。ACM 和 NCM 的描述符如表 3.2 所示。

Arduino MKR WiFI 1010 采用 USBHost 模式和 CDC ACM 对 URG-04LX-UG01 进行驱动，实现抽象传输图 3.22 的两者之间的 USB 通信。

图 3.22 USB 通信示意图

用 USBHost 声明 USB 主机类 USBH，再用 ACMAsyncOper 声明一个设备 AsyncOper，最后声明 ACM 类将 USB 主机和设备进行关联起来 Acm(&UsbH, &AsyncOper)

代码 声明类

```
USBHost      UsbH;          //声明一个 USB 主机类 UsbH
ACMAsyncOper AsyncOper;
ACM          Acm(&UsbH, &AsyncOper);
```

根据 USB 的组成进行 USB 程序编写，包括在 USB 原理中的描述符的数据结构。当获取设备描述符后，USB 根据描述符进行 USB 配置，以及 USB 主机的初始化。

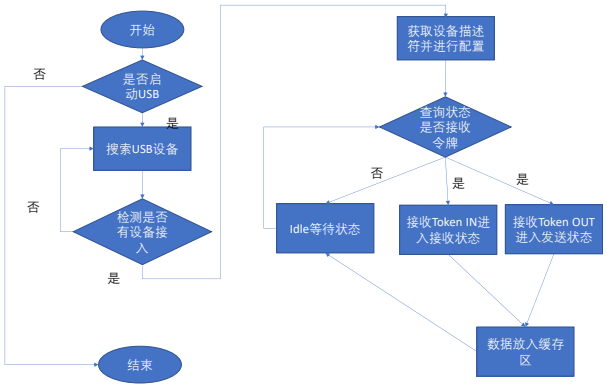

图 3.23 USB 通信流程图

由于激光雷达仅需要 USB 传输数据的功能，因此 USB 驱动需要 IN Intransfer 和 OUT Intransfer 的功能，最后实现流程图 3.23 的功能。

下列是程序的主要代码。UsbH.Init()是 Arduino MKR WiFi 1010 对 USB 进行初始化，Usb.Task()进行 USB 连接状态的查询。当获取设备描述符之后进行 USB 配置，如建立传输端点 IN 和 OUT。

Acm.isReady()用于返回的是为作为 ACM 设备的激光雷达完成好配置后的状态。Acm.SndData()发送数据时，USB 主机通过使用 OUT 端点进行传输，Acm.RcvData()接收数据也是通过端点 IN 进行接收。无论是 IN 还是 OUT 都使用在原理中讲述过的的令牌包。因此在 inIntransfer 和 outIntransfer 都需要标明设备的地址和端口号。

代码 USB 主程序

```
  UsbH.Init();//USB 初始化
  delay( 200 );
  UsbH.Task();//搜索 USB 设备
while (!Acm.isReady())//搜索到 ACM 设备
  {
  Acm.SndData(rcvd_wifi, buf_wifi);//USB 主机发送存储在 buf_wifi 的 rcvd_wifi 长度数据
  Acm.RcvData(&rcvd, buf_acm); // USB 接收主机存储在 buf 的 rcvd 长度数据
  }
uint32_t ACM::SndData(uint16_t nbytes, uint8_t *dataptr)
{
        pUsb->outTransfer(bAddress, epInfo[epDataOutIndex].epAddr, nbytes, dataptr);
}//USB 主机指针 pUsb 向激光雷达地址 bAddress 中发送号端点地址 epAddr 发送数据
uint32_t ACM::RcvData(uint16_t *bytes_rcvd, uint8_t *dataptr)
{
          pUsb->inTransfer((uint32_t)bAddress, epInfo[epDataInIndex].epAddr, bytes_rcvd, dataptr);
}//USB 主机指针 pUsb 接收来自激光雷达地址 bAddress 中接收号的端点地址 epAddr 的数据
```

当令牌包使用后，接下来的传输的数据包就进入 FIFO 存储器，受到 MCU 的管理和读取。在核心程序中使用缓存 buf 和 buf_WiFi，进行从 FIFO 读取数据和并且在程序中使用。

3.4 WiFi 无线通信原理与实现

3.4.1 WiFi 通信原理

本节介绍如何通过配置 Arduino MKR WiFi 1010 的无线传输功能以及对个人计算机 WiFi 无线通信的进行通讯配置。

对比于 Zigbee、蓝牙等无线通信，WiFi 兼容性强，支持方便无线设备连接，无论是 Zigbee 还是蓝牙，都只能配备有 Zigbee 和蓝牙的设备间通信。另一方面 WiFi 拥有比 2Mbitps 的蓝牙和 25Kbitps 的 Zigbee 更快的传输速度。

随着技术的进步，WiFi 的协议 IEEE802.11 已经发展出了多个版本，并且可以在多个频段进行无线通信。发布于 2022 年的最新的 WiFi 版本 WiFi7 所使用的 IEEE802.11be 最高的传输速率可达 30Gbitps，甚至可以在 60GHz 频段工作。

Wi-Fi 没有全名，仅仅是个称呼。创建 WiFi 标准的无线局域网技术的 IEEE802.11，是 IEEE 在 1997 年最初制定的一个 WLAN（无线局域网 Wireless Local Area Net）标准，工作在 2.4GHz 开发频段，支持 1Mbit/s 和 2Mbit/s 的速率传输，并且定义了物理层和 MAC 层规范。

WiFi 无线网络技术通过无线电波将数据传到支持 WiFi 的设备。该技术是澳大利亚悉尼大学发明的。因此每购买使用 WiFi 的设备就需要付给澳大利亚专利费。

图 3.24 是网络模型的构造图。网络模型分为 7 层的 OSI 和 4 层 TCP/IP。WiFi 设备采用 TCP/IP 网络模型进行通信。WiFi 位于图中的网络互连成和网络链路层的底层，并且为应用层和传输层提供支持。

当使用 WiFi 进行网络通信时，所使用的运输层是 TCP 或者 UDP，而在应用层使用 Socket 套接字。套接字含有 IP 地址和端口号，由运输层的 TCP 为提供支持，最后通过位于底层的 WiFi 发送出去。

图 3.24 网络模型结构图

因此根据网络模型，下面讲述位于应用层的套接字和传输层的 TCP 和 UDP。

1、 套接字

在应用层 socket 字进行传输的。套接字就是对网络不同主机上的应用层之间进行双向通信的端点抽象，一个套接字就是网络进程的一端，提供了应用层进程利用网络协议交换数据的的机制。从位置上说，套接字上联应用进程下联网络协议栈，是应用程序通过网络协议进行通讯的接口，也是应用程序与网络协议栈进行交互的接口[11]。

（1） 表示方法

套接字在 IP 地址后面加上端口号进行表示，其中的 IP 地址和端口号用逗号或者冒号进行分隔。若是 IP 地址是 192.168.1.100，而端口号是 10940，那么套接字表示为(192.168.1.100:10940)。

（2） 工作流程

互联网进行通信，至少需要一对套接字，其中一个运行于客户端，称之为 Client Socket，另一个运行于服务器端，称之为 Server Socket[11]。

使用套接字进行网络传输时，TCP Client 和 TCP Server 两者套接字之间的连接过程可以分为三个步骤：

第一步是 TCP Server 监听。TCP Sever 端套接处于等待连接的状态，实时监控网络状态，知道 TCP Client 发送套接字进行请求连接。

第二步是 TCP Client 请求。由 TCP Client 的套接字提出连接请求，,通过指定的 TCPServer 的 IP 地址和端口号向指定的 TCP Server 进行连接。

第三步是连接确认。当 TCP Server 端套接字监听到或者说接收到 TCP Client 套接字

的连接请求，就会响应 TCP Client 套接字的请求，建立一个新的线程，并把 TCP Server 端套接字的描述发送给 TCP Client。一旦 TCP Client 确认了此描述，连接就建立好了。而 TCP Server 端套接字继续处于监听状态，接收其他 TCP Client 套接字的连接请求。

2、 网络端口（Port）和 IP 地址

网络端口处于位于运输层，用于一台主机的进程和另一台主机的进程交换数据。区别于硬件端口，网络端口是软件端口是应用层的各种协议进程与运输实体进行层间交互的一种地址。

TCP/IP 的运输层用一个 16 位的只具有本地计算机有意义道的端口号来表示一个端口，16 位端口号数值范围在 0~49151.。

服务端的端口号分为 0~1023 的系统端口号和 1024~65535 端口号的登记端口号，前者用于系统应用使用，后者为非系统应用使用。而客户端使用的端口号数值为 49151~65535，这类端口号仅仅在客户进程运行时才动态使用。

3、 协议

协议分为 UDP 和 TCP 两类。

（1） UDP

UDP 协议属于 TCP/IP 的一种[12]。对比与比较复杂的 TCP，UDP 只在 IP 的数据报服务进行改动，但是 UDP 是在发送数据之前不需要建立连接，不保证可靠交付，并且由于 UDP 无需进行拥塞控制[13]，因此发送主机在网络出现拥塞情况时不会降低发送速率。在 Arduino MKR 1010 的数据库 WIFININA 包含 UDP 的 API，并且可以用 UDP 的 API 进行数据无线传输。

（2） TCP

TCP Client 是用于 TCP 协议中的的客户端。客户端一般用于向服务端发送消息，当向 TCP Server 发送消息之前，TCP Client 需要知道需要发送的 TCP Server 的 IP 地址和可用端口[14]。值得注意的是客户端的 IP 地址往往由服务端或者网络自行分配。

TCP Server 是用于 TCP/IP 协议的服务端，用于处理客户端发送的信息，也会传回处理结果。TCP Server 需要设定 IP 地址和可用端口用于 TCP 客户端识别。

表 3.3 UDP 和 TCP 特点表

UDP	UDP 是无连接的。UCP 是在发送数据之前不需要建立连接，因此开销和时延相对于 TCP 较少。
	UDP 使用最大努力交付。UDP 不保证可靠交付。，因此主机不需要维持复杂的连接状态表。
	UDP 是面向报文的。发送方的 UDP 对应用程序交下来的报文，在添加首部后就向 IP 层交付。
	UDP 没有拥塞控制。由于 UDP 无拥塞控制，发送主机在网络出现拥塞时不会降低发送速率。
	UDP 支持一对一、一对多、多对一和多对多通信
	UDP 首部开销小。UDP 首部只有 8 字节，而 TCP 是 20 字节。其报文格式为源端口、目的端口，长度和校验和，且每个字段长度为 2 字节
TCP	TCP 是面向连接的运输层协议。应用层在使用 TCP 协议之间，需要建立 TCP 连接
	每一条 TCP 连接只能有两个端点，每一条 TCP 连接只能是点对点的[15]
	TCP 提供全双工通信。TCP 允许通信双方的应用进程在任何时候都能发送数据，应用程序在把数据传送到 TCP 缓存后就可以做自己的事
	面向字节流。TCP 的流指的是流到进程或从进程流出的字节序列。TCP 交互是一次一个数据块，但不知道传输字节流的含义，因此不保证接收方收到的数据块与发送时的一致

3.4.2 WiFi 通信实现

WiFI 通信的设计通过在 Arduino IDE 调用 WiFININA 库实现 WiFi 的配置和网络通信。程序流程图如 3.29。

根据 WiFi 的传输原理，WiFi 通信的底层是 WiFi，因此第一步是构建好网络互连层和网络链路层，让设备连接上 WiFi。然后是配置传输层的协议，本文采用的 TCP 协议因此需要 TCP Client 和 TCP Server，Arduino MKR WiFi 1010 作为 TCP Client 而个人计算机作为 TCP Server。最后在主程序应用层使用 Socket 套接字进行数据传输。至此 WiFi 通信已经构建好了应用层、传输层和网络互连层以及网络链路层。

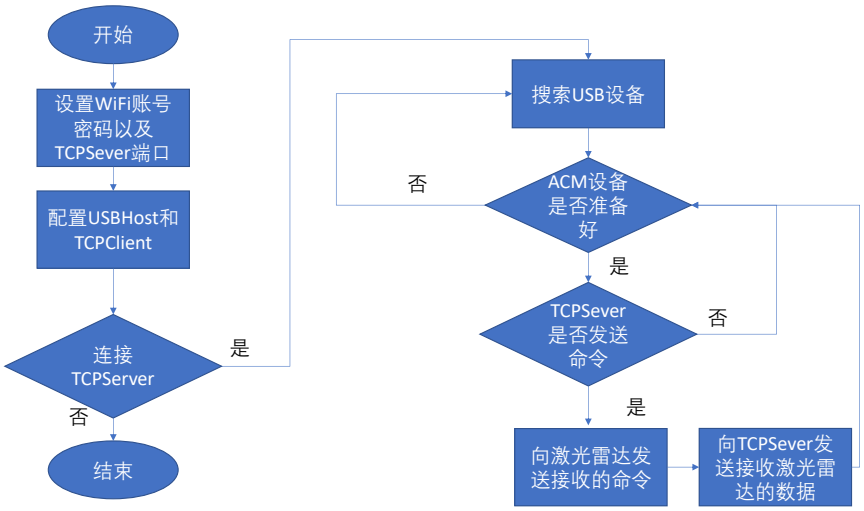

图 3.25 WiFi 通信流程图

图 3.25 左边主要是进行初始化无线通信和 USB 驱动，第一步是在程序中写入需要连接 WiFi 账号和密码，通过指定 WiFi 避免连接错误，之后是通过 WiFiNINA 的 Client 类声明 TCP Client。将激光雷达和个人计算机的数据通过 WiFiNINA 库的套接字函数实现 WiFI 无线传输。同时，在个人计算机连接同一 WiFi 并开启 TCP Server。

右边进行嵌入式的死循环操作，搜索 USB 设备就是查询作为 ACM 设备的激光雷达是否根据设备描述符进行配置完成。当个人计算机或者激光雷达传入消息，Arduino MKR WiFi 1010 就起到中转站功能，将消息转发到相应的设备。

核心程序如下。在连接上网络后，核心程序通过死循环完成通信工作，若是有来自的个人计算机的指令，就将指令写入 USB 进行发送。否则 USB 处于接收状态，并且数据接收到为 NULL。

代码 WiFi 通信主程序

```
char ssid[] = "Visson";//连接名为 Visson 的 WiFi
char pass[] = "0000000";
WiFiClient client; //声明一个客户端对象，用于与服务器进行连接
WiFi.begin(ssid, pass);//连接 WiFi
if (client.connect(serverIP, serverPort)) //  连接位于 Vission 的 Server
  {
    client.print("Hello world!"); //连接 Server，在 Server 上显示 Hello world 否则
```

续代码 WiFi 通信主程序

```
  }
   Loop{                          //嵌入式死循环
       if (client.available()) {   //若有指令发送到 Arduino MKR WiFi 1010 就将指令发送到 ACM 设备
          client.write(line.);      //将来自于个人计算机的 Socket 写入到 line
         Acm.SndData(line.length,line.);
       }
Acm.RcvData(&rcvd, buf);//接收来自于 ACM 设备的数据，若是没有数据发送接收的数据为 NULL
client.write( buf，rcvd);//将接收的数据采用 Socket 发送出去
}
size_t WiFiClient::write(const uint8_t *buf, size_t size) {
ServerDrv::sendData(_socket, buf, size);
}
int WiFiClient::read(uint8_t* buf, size_t size) {
  return    WiFiSocketBuffer.read(_sock, buf, size);
}
```

4 系统测试

4.1 引言

成品不仅需要通信调试，同样还要对成品需要进行供电调试。通过对比使用的不同容量电池，检测成品耗电情况。

实现 Arduino MKR WiFI 1010 和 URG-04LX-UG01 的 USB 通信，以及 Arduino MKR WiFI 1010 和个人计算机的 WiFi 无线通信后，需要对通信进行调试。针对 USB 通信，主要测试的 USB 传输数据是否丢包。USB 调试是基于激光雷达 SCIP 指令进行的。对于 WiFi 通信测试主要测试的方面是延时问题。该调试是基于 WiFiNINA 库进行的。

在经过调试之后，本文还进行系统集成测试，即激光雷达和个人计算机无线通信的同时，个人计算机能够将接收到的数据进行成像。

4.2 供电调试

在使用过程中，由于电池消耗导致电压不足，会出现供电能力下降会导致 Arduino MKR WiFi 1010 连接不上 WiFi 的问题。此时需要更换电源充足的电池。在不同的使用场景中，成品对电池的容量有着不同的使用需求。经过测试 ACE 1000mAh 15C 11.1V 3S1P 可以使用连续使用一天，而 ACE 5300mAh 15C 11.1V 3S1P 可以持续三天。同样若是连接的杜邦线会导致电阻太大，使得电压不能够达到需求也导致嵌入式开发板不能正常工作。

本文为剔除由于电池电量消耗到不足以正常工作所带来的影响，在接下来的调试中需要将只能满足一天使用需求的 ACE 1000mAh 15C 11.1V 3S1P 更换为容量更大的 ACE 5300mAh 15C 11.1V 3S1P，来为激光雷达和 IoT 嵌入式开发板提供更久的续航。

4.3 通信调试

4.3.1 USB 通信调试

USB 的调试是通过网络串口调试助手进行调试的，通过 Arduino MKR WiFi 1010 发送命令，然后在串口调试助手返回 USB 的数据。根据返回的数据进行调试与问题解决。

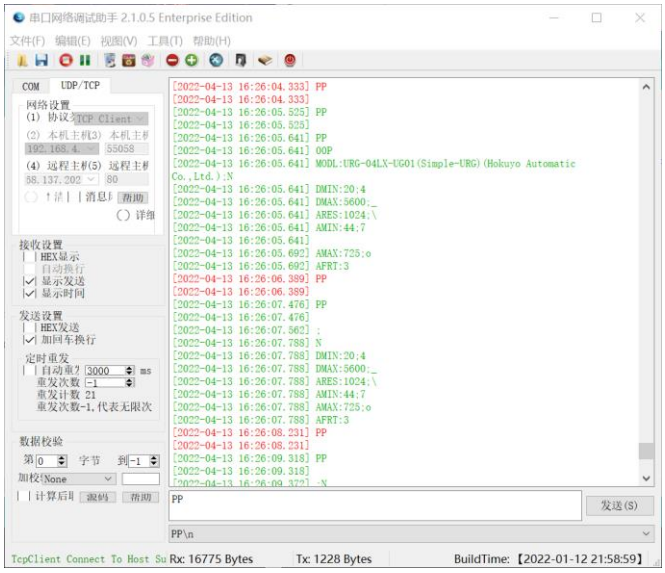

图 4.1 USB 丢包图

在通信时出现如图 4.1 所示 USB 丢包情况，在 Arduino MKR WiFi 1010 无法配置 USB 时钟配置情况下，本调试采用分包的方式将激光雷达的包分为几段进行发送，保证缓存区有足够的空间接收激光雷达的数据而不至于由于满载而丢弃未缓存的数据，得到和图 4.2 激光雷达有线连接电脑一样的结果。值得注意的是 SCIP 指令“SS”是对波特率的调试并不可以对 USB 速率进行改动，而是针对 RS232C 接口的。

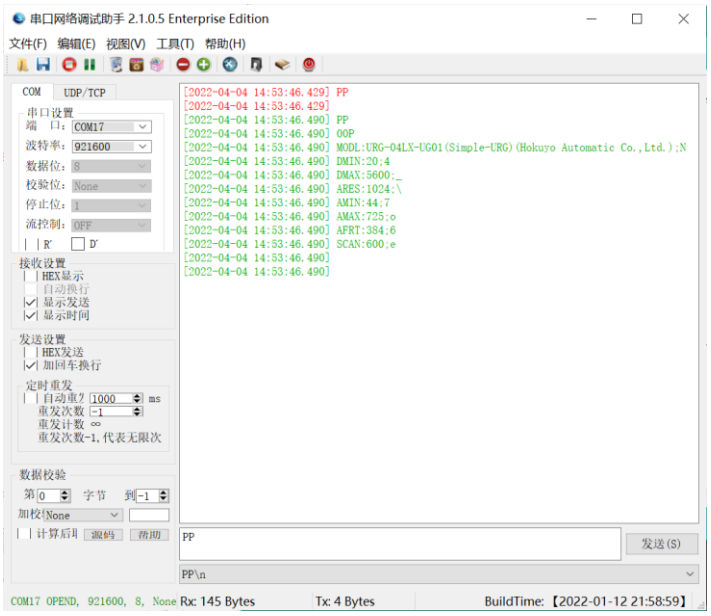

图 4.2 激光雷达串口传输结果图

4.3.2 WiFi 通信调试

延时问题比较影响系统工作速度，根据串口调试助手返回的数据会有好几段的分段并且是间隔较长时间。

WiFiNINA 的客户端发送数据的函数有 Client.write（b）和 Client.write（buffer，length），前者的 b 指的是 1 字符，也就是每次调用发送 1 字符，可以利用循环将收到的字符一一发送，而后者是用 buffer 进行缓存，然后发送 length 长度的 buffer。对比之下，后者可以降低延迟，即由 PP 命令接收的每 64 字节 150ms，变为 1ms 就可以将将整个返回数据接收。

WiFiNINA 进行接收个人计算机发送的指令时，使用 Client.readString，由于读取字符串会延长向激光雷达写入数据的时间，改用 Client.read（buffer,length）将指令直接写入送到 USB 进行处理，可以将指令快速写入激光雷达。

在更换 WiFi 后 IP 地址会发生改变，由于 Arduino MKR WiFi 1010 使用的程序是 TCP Client 需要获取 TCP Server 的 IP 地址，因此个人计算机需要在命令提示符使用“ipcinfig”进行地址查询，然后设置好 IP 地址和端口号。

4.4 系统集成测试

系统集成测试是实现个人计算机和激光雷达 WiFi 无线通信程序封装，并且进行数据成像。本测试以个人计算机程序是否能够对激光雷达进行控制以及能否正确将激光雷达发送的数据进行成像作为指标。

图 4.3 激光雷达扫描侧视图

如图 4.3，该测试以激光雷达扫描墙面作为测试对象。通过对墙面的扫描对系统进

行集成测试。

激光雷达放置在距墙面 0.15 米的位置，也就是图 4.4 所示激光雷达最前端。墙的距离是 0.15 米，在根据扫描的坐标原点距离 0.05 米的激光雷达本身身长，可得理论距离 0.2 米。

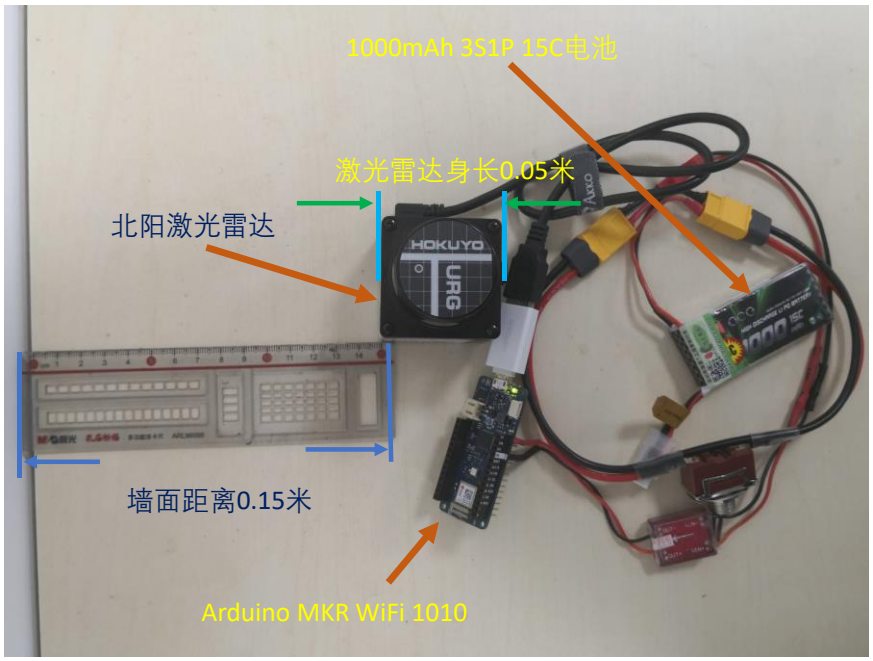

图 4.4 墙距扫描俯视图

图 4.5 程序扫描图结果墙面呈现的是一条直线，角度为 0 度时距离墙面的距离为 0.1980 米，误差在 2 毫米左右。考虑到激光雷达采用无线传输进行成像，能够完成计算机数据接收并且程序可对数据处理呈像是本实验的指标，而本结果符合预计的实验指标。

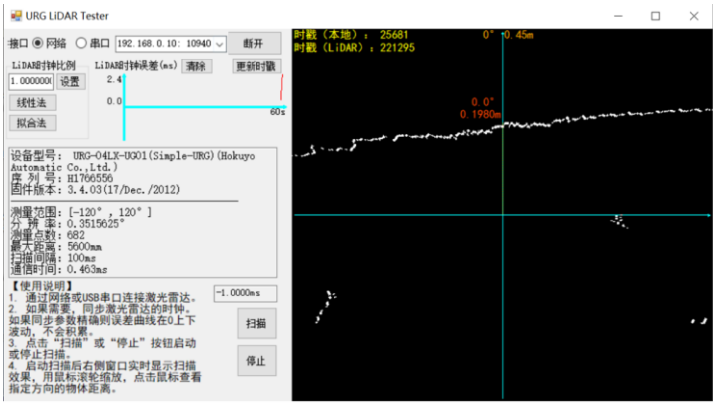

图 4.5 计算机程序扫描图

总结与展望

本文总结

本文实现了具备 WiFi 无线通信的嵌入式板以 USB 主机模式对激光雷达进行控制的功能，从而使得激光雷达可以进行 WiFi 无线通信。本文同时选择来自日本的北阳激光雷达进行设计，北阳公司的激光雷达都是小巧的外形。

本文最为关键的是通过 USB 实现嵌入式板对激光雷达的控制。日常生活中的 USB 数据线和实验使用的 USB OTG 数据线极为不同，普通的 USB 数据线连接电脑即可传输数据而除了用于供电以外不能用于 USB 设备与 USB 设备之间的通信。在阅读相关 OTG 的资料了解了 USB OTG 的原理后， 发现 USB 设备之间的通信不仅需要通过 USB OTG 线解决 ID 引脚接地的问题，还需要 USB 驱动，甚至需要 USB OTG 线为设备提供充足的电源。这打破了起初认为通过 USB 线将激光雷达和 IoT 嵌入式开发连接就即可采用串口的方式进行数据传输的想法。在发现这些 USB 问题后，一些硬件设计问题也接踵而至，尤其是 OTG 线供电的问题，由于 Arduino MKR WiFi 1010 的使用说明仅仅描述可以通过飞线为 USB 接口电源引脚 VBUS 供电，但是并未说明在何处进行飞线，甚至只介绍接入的电池 3.7V 而未说明如何用 3.7V 电池为 VBUS 提供 5V 电源。面对一系列的模棱两可的使用说明的描述，通过在别的开发板寻找到解决 OTG 问题的方法，查询 BQ24195 芯片的使用说明和比较 Arduino MKR WiFi 1010 电路图找到 OTG 供电的可行性方案。解决供电问题后，IoT 嵌入式开发板和激光雷达如何利用 USB 进行通信传输的问题又是一大难关。由于在国内并无案例可以参考，因此需要到国外网站寻找案例，所幸的是存在用采用 USB 在激光雷达和嵌入式开发板之间通信的例子。这些例子虽然是采用 USB 连接，但是需要用到 FTDI 开发板作为主机，然后通过串口传输到电脑，而不是直接采用 IoT 嵌入式开发板进行无线传输。在案例中了解到了激光雷达可以采用 USB CDC ACM 进行驱动，因此本文也采用 ACM 进行驱动激光雷达。解决问题之后，所有问题都集中在通信是否发送和接收的数据完整以及通信延时的问题上。经过调试，虽然还是存在一些问题，但还是能够进行通信。

至此利用 IoT 嵌入式板对激光雷达进行控制，并且利用自身搭载的 WiFi 无线通信模块进行 IoT 实现。虽然已经有具备以太网接口的激光雷达出现，但是本毕业设计采用

USB 不仅简化了供电设计，而且由于越来越多设备采用 USB 接口，使得一方面激光雷达可以和更多的设备连接，而 IoT 嵌入式开发板也更能使得更多 USB 设备具备 IoT 功能。

未来展望

本实验仍然存在一些问题，成品不能十分稳定地进行无线传输，在无线传输一段时间后会出现数据丢失问题，但是由于时间问题本文不能够进行完善和优化。同时在实验的实践中也认识到了许多新的激光雷达无线传输方案。

Arduino MKR WiFi 1010 有移动热点功能，可以通过网络进行初始化网络配置。本次毕业设计是采用 WiFi 路由器以及电脑移动热点进行 WiFi 连接的，但是由于开发板的 WiFi 热点能力较弱，会小概率出现丢包现象，由于时间问题不能深入研究故此放在未来展望。

激光雷达通过连接可以外接电源的 USB 集线器，连接 GPS 等设备。集线器可以让更多 USB 设备连接起来，而且市面上出现自带电接口的集线器可以直接为 USB 设备供电，因此将激光雷达和其他设备连接上集线器可以扩展激光雷达功能，甚至将历史数据送到 U 盘当中保存。

虽然已经出现以太网接口的激光雷达，利用以太网接口的激光雷达进行无线通信。但是本文并没有对以太网接口的激光雷达进行设计。

参 考 文 献

[1] J. Lin, S. Li, W. Dong, T. Matsumaru and S. Xie, "Long-Arm Three-Dimensional LiDAR for Antiocclusion and Antisparsity Point Clouds,"[J] in IEEE Transactions on Instrumentation and Measurement, vol. 70, pp. 1-10, 2021, Art no. 4506610, doi: 10.1109/TIM.2021.3104019.

[2] SOURCEFORGE. Arduino を USB ホストシールドを用いてプログラムを作成する.[EB/OL]. [2022-4-26]. https://sourceforge.net/p/urgnetwork/discussion/general/thread/7cb3dc2e/?msclkid=e35051c2a82111ec845c341ff5929c87.

[3] CSDN. ESP32+激光传感器 VL53L1x 移植与调试.[EB/OL]. [2022-4-26]. https://blog.csdn.net/qq_20515461/article/details/102912857.

[4] HOKUYO AUTOMATIC CO,.LTD . Communication Protocol Specification For SCIP2.0 Standard.[EB/OL] (2006.10.10). [2022-4-26]. https://www.hokuyo-aut.jp/

[5] Arduino.Cortex-M0 32-bit SAMD21 datasheet.[M/OL]. [2022-4-26]. https://content.arduino.cc/assets/mkr-microchip_samd21_family_full_datasheet-ds40001882d.pdf?_gl=1*r1yegm*_ga*NDI0Njg5MDQwLjE2NDg4MDQ0NDE.*_ga_NEXN8H46L5*MTY1MDk1ODI0MC4zNS4xLjE2NTA5NTgyNjguMA.

[6] Atmel. AT06475: SAM D21 USB . [EB/OL]. [2022-4-26]. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42261-SAM-D21-USB_Application-Note_AT06475.pdf.

[7] Atmel. Atmel AVR4950: ASF - USB Host Stack.[EB/OL]. [2022-4-26]. https://ww1.microchip.com/downloads/en/DeviceDoc/doc8486.pdf.

[8] Atmel. AT09331: ASF USB Stack Manual. [M/OL]. [2022-4-26]. https://ww1.microchip.com/downloads/en/Appnotes/Atmel-42336-ASF-USB-Stack-Manual_ApplicationNote_AT09331.pdf

[9] armKEIL. USB Component Version 6.15.4 MDK Middleware for USB Device and Host Communication.[EB/OL]. [2022-4-26]. https://www.keil.com/pack/doc/mw/USB/html/_c_d_c.html?msclkid=a553b9c6bb2511ec94ae7f5070b80828.

[10]赵海涛.《嵌入式系统 USB 打印机驱动程序的设计与实现》[D]. 北京邮电大学,2010

[11]创客诚品，刘慧欣，孟令一编著,C 语言从入门到精通 全新精华版[M]. 北京希望电子出版社, 2017：第 377 页~第 378 页

[12]汪国辉.《基于浏览器双向连接的研究与实现》[D] . 北京.北京邮电大学，2013.

[13]郭剑锋.《工业以太网技术的研究与实现》 [D] .机械科学研究院，2004

[14]董继刚. 《水声通信网仿真与实现》[D]. 哈尔滨工程大学，2010

[15]刘桦. 基于 Web 技术的嵌入式网络图像监控系统的研究和设计》[D].河海大学，2007

致　　谢

本论文完成之时，也意味着大学完篇。

在无线激光雷达的设计与实现中，经过林靖宇教授不断地描述激光雷达的概念，本人从对实验设计的一片空白，逐渐找清楚需要实现的目标。在进行实现过程，本人遇到了一系列从所未见的难点，不过大学学的基础知识还是给了有限的帮助。在每天实验室和师兄们的谈笑中，本人逐渐把 USB OTG 弄懂、实现嵌入式板 OTG 以及找到激光雷达所属的 USB 类。虽然目前还是存在一些问题，但是本人收获丰富，毕竟劳动者最富有了。同时林靖宇老师传授给本人一些珍贵的人生经验以及知识整理方法，并且悉心指导本实验与论文的完成。

本人对大学有过迷茫以及自我否定。大学可以用屈原的《离骚》的三句进行总结。首先庆幸的是“回朕车以复路兮，及行迷之未远，”真正认识到在大学学习的不仅只是专业知识而且还有哲学思想。以马克思基本原理作为指导，本人对学习生活的感悟进入到新的境界，尤其是对《政治经济学批判》前三卷的阅读。经过阅读《政治经济批判学》后，本人对事物以及学习的知识能够比较有不同以往的思考，那一刻可以用《离骚》的“步余马于兰皋兮，驰椒丘且焉止息”表述舒畅的心情。同样在大学中，本人也认识到自身的不足，对自己进行否定之否定后，“进不入以离尤兮，退将复修吾初服。” 在称之为“内卷”的浪潮中，本人既然能力有限，那么有限就有限吧，毕竟劳动者才是最富有的，岂能“学成文武艺，货于帝王家。”

最后感谢广东工业大学教授过本人的老师，以及陪伴本人度过大学的同学和朋友。

附录 A　格式电池参数表

表 A 1 格式电池参数表

序号	型号	尺寸(毫米 mm)	重　量 (克 g)	电池容量 （mAh）	放电倍率(C)	电压 (V)	价　格 （元）
1	ACE 800mAh 15C 11.1V 3S1P	20*30*58	75	800	15	11.1	54.4
2	ACE 1000mAh 15C 11.1V 3S1P	17*35*73	90	1000	15	11.1	68.8
3	ACE 1300mAh 25C 11.1V 3S1P	22*35*71	110	1300	25	11.1	75.2
4	ACE 1550mAh 25C 11.1V 3S1P	23*30*92	134	1550	25	11.1	89.6
5	ACE 1800mAh 20C 11.1V 3S1P	23*30*92	135	1800	20	11.1	89.6
6	ACE 2000mAh 5C 11.1V 3S1P	22*30*94	128	2000	5	11.1	95.2
7	ACE 2200mAh 20C 11.1V 3S1P	22*34*105	169	2200	20	11.1	75.2
8	ACE 2200mAh 25C 11.1V 3S1P	22*34*106	168	2200	25	11.1	99.2
9	ACE 2200mAh 30C 11.1V 3S1P	22*34*117	186	2200	30	11.1	109.6
10	ACE 2200mAh 40C 11.1V 3S1P	23*34*107	182	2200	40	11.1	139.2
11	ACE 2600mAh 25C 11.1V 3S1P	25*34*116	208	2600	25	11.1	142.2
12	ACE 3300mAh 25C 11.1V	20*43*138	255	3300	25	11.1	196

	3S1P								
13	ACE	4000mAh	25C	11.1V	24*42*138	302	4000	25	11.1	219.2
	3S1P								
14	ACE	4000mAh	30C	11.1V	27*41*138	330	4000	30	11.1	229.2
	3S1P								
15	ACE	5000mAh	40C	11.1V	29*42*138	371	5000	40	11.1	299.6
	3S1P								
16	ACE	5300mAh	30C	11.1V	29*42*136	367	5300	30	11.1	309.2
	3S1P								
17	ACE	5500mAh	25C	11.1V	45*26*189	448	5500	25	11.1	322
	3S1P								
18	ACE	7000mAh	40C	11.1V	43*28*193	525	7000	40	11.1	582.4
	3S1P								

附录 B　实物图

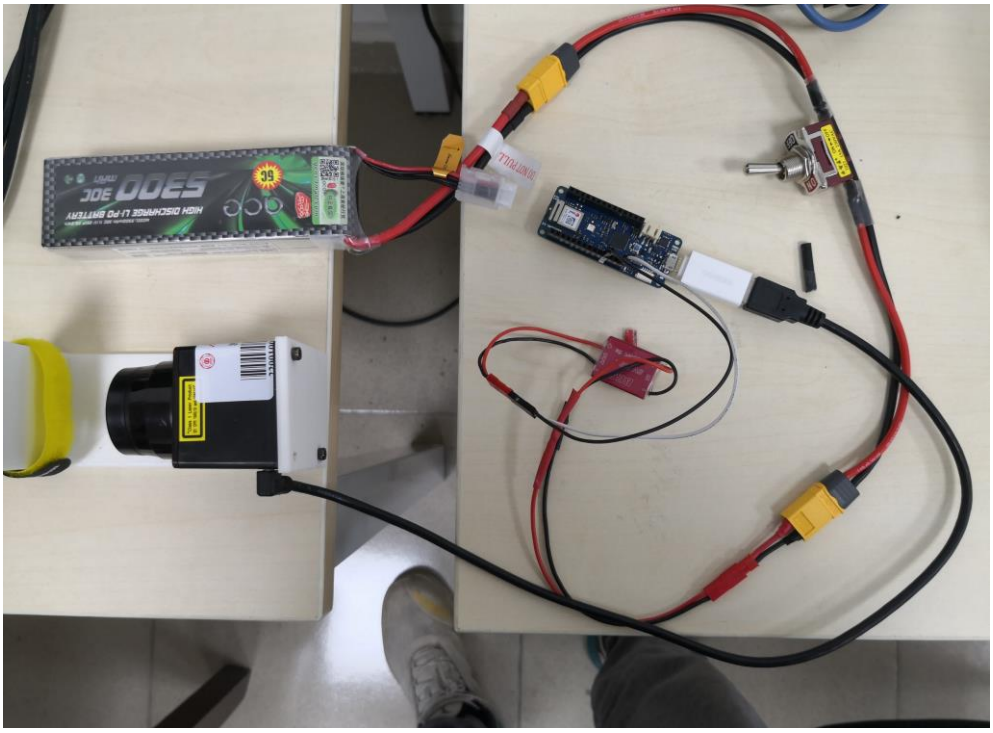

图 B 1 实物图