Embedded Technoloy and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(3), 149-160
Published Online June 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.23012

Hans汉斯

CherryUSB原理性分析和应用实践

吕家振

CherryUSB社区，江苏 南京

收稿日期：2025年8月11日；录用日期：2025年10月15日；发布日期：2025年10月27日

摘　要

CherryUSB是一个轻量级、高性能的开源USB主从协议栈，由国内开发者维护，专为资源受限的带USB外设的嵌入式系统设计。近年来，随着带USB外设的嵌入式设备逐渐增多，CherryUSB成为了一个可靠稳定的选择。相比其他USB协议栈，CherryUSB更注重用户阅读体验、驱动全面性、稳定性和高性能，降低了开发者入门的门槛，发挥出了嵌入式设备中USB的优势。本文对CherryUSB主机和从机代码进行原理性的分析，并基于rt-thread artpi2开源硬件平台，进行CherryUSB主从机的应用实践，为嵌入式USB开发提供参考和借鉴。

关键词

USB协议栈，嵌入式系统，CherryUSB项目

CherryUSB Principle Analysis and Application Practice

Jiazhen Lv

CherryUSB Community, Nanjing Jiangsu

Received: August 11, 2025; accepted: October 15, 2025; published: October 27, 2025

Abstract

CherryUSB is a lightweight, high-performance open source USB device and host stack, maintained by domestic developers, designed for resource-constrained embedded systems with USB peripherals. In recent years, with the increasing number of embedded devices with USB peripherals, CherryUSB has become a reliable and stable choice. Compared with other USB protocol stacks, CherryUSB focuses more on user reading experience, driver comprehensiveness, stability, and high performance, lowering the barrier for developers to get started and giving full play to the advantages of USB in embedded devices. This paper analyzes the CherryUSB host and slave code in principle, and conducts

文章引用：吕家振. CherryUSB 原理性分析和应用实践[J]. 嵌入式技术与智能系统, 2025, 2(3): 149-160.
DOI: 10.12677/etis.2025.23012

the application practice of CherryUSB host and slave based on the rt-thread open source hardware platform named art-pi2, and provides a reference for embedded USB development.

Keywords

USB Stack, Embedded System, CherryUSB Project

1. 引言

CherryUSB 项目于 2022 年发布，旨在为资源受限的嵌入式系统提供一套简洁易学且高效的 USB 协议栈[1]。在设计上，CherryUSB 采用树状化编程，所有类驱动和 USBIP 驱动模板化设计，没有复杂的 C 语言语法，方便用户学习和新增驱动。对用户常用的数据收发 API 采用类比法，将复杂 USB 通信类比为 UART + DMA 通信，简化了通信逻辑，方便用户使用。为了突出 USB 的性能，主从驱动直接对接寄存器操作，并且使用零拷贝、硬件 DMA 直通和中断快响应等技术，充分释放 USB 硬件带宽。

自发布以来，CherryUSB 得到了众多开发者和半导体企业的支持，包括博流智能、先楫、平头哥、飞腾、恩智浦、乐鑫、匠心创、博通、嘉楠等。并应用到众多嵌入式设备，包括鼠标、键盘、可视门铃、网络通信、图传、存储、调试、bootrom 等应用。并且 CherryUSB 采用 Apache 2.0 授权，可免费在商用解决方案中使用。

随着热度的提升，有越来越多的开源项目使用 CherryUSB，同样 CherryUSB 也对火热的开源项目进行了支持，包括 daplink，blackmagic，lvgl，lwip，nuttx，zephyr 等，并加入到 rt-thread 国产 OS 项目中，成为 rt-thread 标准 USB 协议栈。

USB 协议栈是一个庞大的体系，如何化繁为简，降低用户开发门槛，理解 USB 的实现原理，是一个难题。本文将对 CherryUSB 的主机和从机代码进行原理性分析，提取关键信息，帮用户理清脉络，并使用 rt-thread artpi2 开源硬件进行移植和主从应用的实践。

2. CherryUSB 从机协议栈概述

CherryUSB 从机协议栈主要包括从机控制器 IP 驱动，USB 设备枚举，USB 驱动加载。其中，设备枚举和驱动加载部分包括 USB 描述符注册，USB 接口驱动注册，USB 端点注册。CherryUSB 设备协议栈支持多种 USB 设备类驱动，包括 CDC、HID、MSC、VIDEO、AUDIO、RNDIS、MTP 等，基本涵盖所有常用类驱动。本章将从 IP 驱动设计切入，分析从机协议栈的整个实现原理。

2.1. 从机控制器 IP 设计

USB 从机控制器 IP 是一种专用集成电路设计模块，用于实现 USB 协议栈的数据链路层功能，使嵌入式系统能够作为 USB 从设备与主机建立通信。负责管理 PHY 交互、数据包处理、端点配置和传输调度等底层操作，同时向上层软件提供标准寄存器定义。而从机控制器 IP 驱动则是根据这些寄存器定义，进行一些配置，使得 USB 设备能够与主机进行通信。目前，在 CherryUSB 中，支持多种从机控制器 IP，商业性的 IP 包括 MUSB、DWC2、FOTG210 等。下面我们以 DWC2 为例，分析该 IP 的设计思路。

1) DWC2 IP 寄存器定义

DWC2 IP 寄存器包含全局寄存器、从机寄存器、主机寄存器三类。从机寄存器中又包含端点寄存器组，包括 IN 端点寄存器组和 OUT 端点寄存器组，各支持最大 16 组。寄存器组的定义如下：

```
typedef struct
{
__IO uint32_t DIEPCTL;          /*!< dev IN Endpoint Control Reg       900h + (ep_num * 20h) + 00h */
uint32_t Reserved04;            /*!< Reserved                          900h + (ep_num * 20h) + 04h */
__IO uint32_t DIEPINT;          /*!< dev IN Endpoint Itr Reg           900h + (ep_num * 20h) + 08h */
uint32_t Reserved0C;            /*!< Reserved                          900h + (ep_num * 20h) + 0Ch */
__IO uint32_t DIEPTSIZ;         /*!< IN Endpoint Txfer Size            900h + (ep_num * 20h) + 10h */
__IO uint32_t DIEPDMA;          /*!< IN Endpoint DMA Address Reg       900h + (ep_num * 20h) + 14h */
__IO uint32_t DTXFSTS;          /*!< IN Endpoint Tx FIFO Status Reg    900h + (ep_num * 20h) + 18h */
uint32_t Reserved18;           /*!< Reserved    900h + (ep_num*20h) + 1Ch-900h + (ep_num * 20h) + 1Ch
*/
} DWC2_INEndpointTypeDef;

typedef struct
{
__IO uint32_t DOEPCTL;          /*!< dev OUT Endpoint Control Reg      B00h + (ep_num * 20h) + 00h */
uint32_t Reserved04;            /*!< Reserved                          B00h + (ep_num * 20h) + 04h */
__IO uint32_t DOEPINT;          /*!< dev OUT Endpoint Itr Reg          B00h + (ep_num * 20h) + 08h */
uint32_t Reserved0C;           /*!< Reserved                          B00h + (ep_num * 20h) + 0Ch */
__IO uint32_t DOEPTSIZ;         /*!< dev OUT Endpoint Txfer Size       B00h + (ep_num * 20h) + 10h */
__IO uint32_t DOEPDMA;          /*!< dev OUT Endpoint DMA Address      B00h + (ep_num * 20h) + 14h */
uint32_t Reserved18[2];         /*!< Reserved B00h + (ep_num * 20h) + 18h - B00h + (ep_num * 20h) +
1Ch */
} DWC2_OUTEndpointTypeDef;
```

从上述寄存器总结如下表 1，从中得出，数据的传输，主要是依靠端点，并配置端点相关的寄存器，下表 1 中 IN 代表发送，OUT 代表接收：

Table 1. DWC2 IP endpoint register description
表 1. DWC2 IP 端点寄存器说明

寄存器	功能
DIEPCTL\DOEPCTL	配置端点属性，类型，最大数据包长度
DIEPTSIZ\DOEPTSIZ	配置端点发送或者接收长度
DIEPDMA\DOEPDMA	配置端点发送或者接收 buffer 地址
DIEPINT\DOEPINT	配置发送或者接收端点中断

2) DWC2 IP 中断

DWC2 的中断标志有很多，这里只列举常用的一些中断，包括：

USB_OTG_GINTSTS_OEPINT
USB_OTG_GINTSTS_IEPINT
USB_OTG_GINTSTS_USBRST
USB_OTG_GINTSTS_USBSUSP
USB_OTG_GINTSTS_WKUINT

USB_OTG_GINTSTS_OEPINT 和 USB_OTG_GINTSTS_IEPINT 代表是端点中断，其余几个表示 USB 的一些状态。在端点中断中，又进一步划分

- USB_OTG_GINTSTS_OEPINT

Table 2. DWC2 IP OUT endpoint interrupt register description
表 2. DWC2 IP OUT 端点中断标志说明

中断标志	功能
USB_OTG_DOEPINT_XFRC	接收完成中断
USB_OTG_DOEPINT_STUP	接收 SETUP 包完成中断

- USB_OTG_GINTSTS_IEPINT

Table 3. DWC2 IP IN endpoint interrupt register description
表 3. DWC2 IP IN 端点中断标志说明

中断标志	功能
USB_OTG_DIEPINT_XFRC	发送完成中断
USB_OTG_DIEPINT_TXFE	发送空中断，用于持续发送数据，直到发送完成

根据表 2 和表 3，可以分析出，从机控制器 IP 的特点包括：多组端点寄存器、发送(IN)完成中断、接收(OUT)完成中断、复位\挂起\恢复等状态中断。同样地，其余从机控制器 IP 也是包含这些特点，根据这些特点，从而能够帮助我们定义从机控制器驱动的 API。

2.2. 从机控制器驱动设计

根据从机控制器 IP 的特点，CherryUSB 中制定了通用标准的从机控制器驱动 API (cherryusb/common/usb_dc.h)。如下表 4 所示。

Table 4. CherryUSB device controller API
表 4. CherryUSB 从机控制器 API

函数	功能
usb_dc_init	初始化从机控制器
usb_dc_deinit	反初始化从机控制器
usbd_ep_open	配置端点相关属性

续表

usbd_ep_close	关闭端点
usbd_ep_start_write	端点启动端点发送
usbd_ep_start_read	端点启动端点接收
usbd_event_ep0_setup_complete_handler	setup 包完成中断处理
usbd_event_ep_in_complete_handler	端点发送完成中断处理
usbd_event_ep_out_complete_handler	端点接收完成中断处理

API 和寄存器中大量提到了 USB 的一个名词：端点(endpoint) [2]。端点是 USB 设备内部的基本通信结构单元，本质上是一块缓冲区存储器，用作数据传输的源或目的地，并通过管道(pipe)进行传输。端点具有方向性，包括 IN 和 OUT 方向；端点支持一种传输方式，包括控制传输、批量传输、中断传输、同步传输[3]；端点传输一个包时有最大数据包长度限制，称为 MPS。这里，我们采用类比法，将端点比作 DMA 通道，端点的发送与接收看作是外设发送和接收，端点的传输方式看作是 UART\SPI\I2C 传输等等，端点的完成中断看作是 DMA 通道完成中断。当我们采用类比法以后，可以更快地理解端点的含义和作用，更快的理解从机控制器 IP 和驱动的设计思想。

2.3. 从机协议栈设计

在 USB 中，另一个重要的概念叫做 setup 包[4]，它是主机和从机枚举所定义的一个格式，如图 1 所示。主机和从机通过端点 0 进行控制传输，并发送 setup 包给到从机，从机会执行到 usbd_event_ep0_setup_complete_handler 函数，该函数则是用于处理获取的 setup 包，并按照不同的 bRequest 执行不同的处理，最终完成 USB 的枚举以及接口驱动的加载工作。

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

Figure 1. Format of setup packet
图 1. setup 包格式

下图 2 完整描述了 CherryUSB 从机协议栈的执行过程和调用关系，和从机控制器 IP 驱动设计紧密结合。

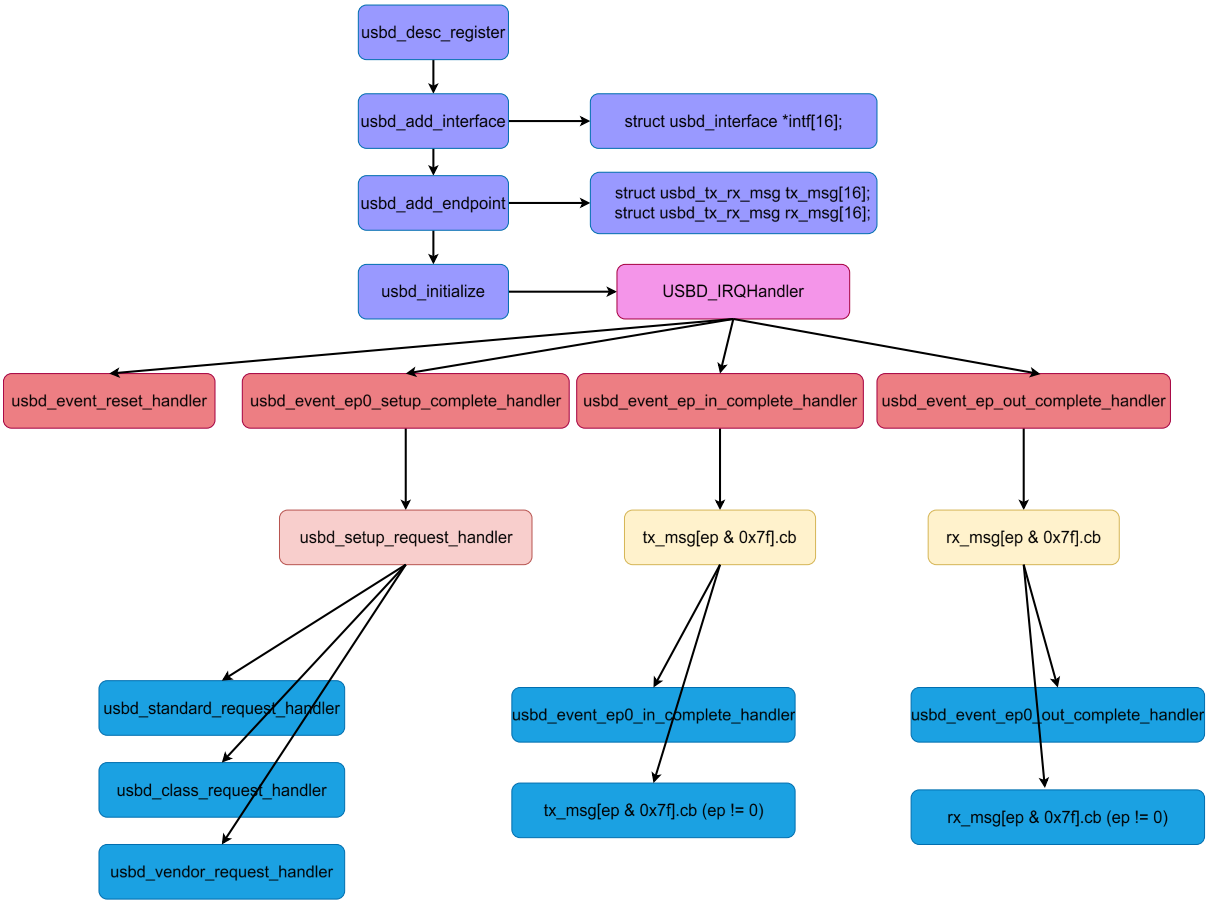

Figure 2. CherryUSB device stack framework
图 2. CherryUSB 设备协议栈框图

3. CherryUSB 主机协议栈概述

CherryUSB 主机协议栈主要包括主机控制器 IP 驱动，USB 设备枚举和驱动加载。相比于从机的一对一的通信方式，主机更具复杂性，由于是一对多的方式，同时需要支持多个 USB 设备，因此，在 CherryUSB 中，采用了 RTOS 的接管，如果使用 bare mental 的方式管理，增加例如死等，状态机的代码，会让用户在阅读和使用上变得非常麻烦，并且也让代码的健壮性变得不可控。CherryUSB 主机协议栈支持多种 USB 设备类驱动，包括 CDC、HID、MSC、VIDEO、AUDIO、RNDIS、HUB 等，可以支持 1 托多个 USB 设备，每个设备还可以是复合设备(支持多种 USB 类接口的设备)。本章同从机一样，从主机控制器 IP 设计入手进行分析。

3.1. 主机控制器 IP 设计

USB 主机控制器 IP 是一个为嵌入式系统提供实现 USB 主机功能的硬件模块，主要负责总线仲裁，数据传输调度，USB 协议的处理，并向上层软件提供标准寄存器定义。而主机控制器 IP 驱动则是根据这些寄存器定义，进行配置并能够与 USB 设备进行通信。CherryUSB 支持多个主机控制器 IP，商业性的 IP

包括 DWC2、MUSB、OHCI、EHCI、XHCI 等。其中 MUSB 和 DWC2 部分模式采用寄存器配置，OHCI/EHCI/XHCI 和部分 DWC2 部分模式采用描述符链表配置，前者更多是把功能交给软件做，而后者则是将复杂功能交给硬件做，降低了软件的负载率，但是同时也增加了软件的编写难度。OHCI/EHCI/XHCI 是 intel 制定的一套主机控制器标准，目前也是市面上最流行也是最通用的，也是最推荐使用的。由于主机控制器设计较为复杂，本章只简单介绍中断设计相关。如下表 5 所示。

Table 5. Interrupt of USB host controller
表 5. USB 主机控制器中断种类

IP	中断标志	功能
DWC2	USB_OTG_GINTSTS_HCINT	中断完成/错误中断
MUSB	MUSB_TXIS/MUSB_RXIS	所有 pipe 中断完成/错误中断
OHCI	XHCI_USBSTS	所有 pipe 中断完成/错误中断
EHCI	EHCI_USBSTS_INT/EHCI_USBSTS_ERR	所有 pipe 中断完成/错误中断
XHCI	OHCI_INT_WDH	所有 pipe 中断完成/错误中断

从上述寄存器功能可以看出，无论主机控制器如何设计，都是具备相似点的，无非只是软件编写方式的不同。在这里 pipe 和从机的端点是对应的，一个端点对应一个 pipe，自然一个 pipe 对应一个端点，但是由于主机是一对多的方式，因此存在多个相同的端点，但是属于不同的 USB 设备，因此，在主机中，统称为 pipe。借助中断，我们可以定义出主机控制器 IP 的驱动。

3.2. 主机控制器驱动设计

主机控制器 IP 驱动，CherryUSB 参考了 linux 的 urb 设计[5]，也是采用 urb (usb request block)的方式，urb 内容如下：

```
struct usbh_urb {
    usb_slist_t list;
    void *hcpriv;
    struct usbh_hubport *hport;
    struct usb_endpoint_descriptor *ep;
    uint8_t data_toggle;
    uint32_t interval;
    struct usb_setup_packet *setup;
    uint8_t *transfer_buffer;
    uint32_t transfer_buffer_length;
    int transfer_flags;
    uint32_t actual_length;
    uint32_t timeout;
    int errorcode;
    uint32_t num_of_iso_packets;
    uint32_t start_frame;
    usbh_complete_callback_t complete;
```

```
    void *arg;
#if defined(__ICCARM__) || defined(__ICCRISCV__) || defined(__ICCRX__)
    struct usbh_iso_frame_packet *iso_packet;
#else
    struct usbh_iso_frame_packet iso_packet[0];
#endif
};
```

- hport 对应一个 USB 设备，当前 urb 将会在哪个 USB 设备上使用
- ep 对应一个 USB 设备上的一个端点，当前 urb 会和该 USB 设备的哪个端点通信
- setup 表示控制传输对应的 setup 包
- transfer_buffer 则表示传输的地址
- transfer_buffer_length 表示传输的长度
- actual_length 表示最终发送或者接收的实际长度

虽然 urb 看上去内容很多，但是实际上，和从机协议栈一样，最终都是端点通信，并且都会触发端点的完成中断，只不过在主机叫做 pipe 完成中断，同样也是可以类比成 UART + DMA 的形式，将 pipe 比作 DMA 通道，传输比作一个外设，比如 UART/SPI/I2C，完成中断比作是 DMA 通道完成中断。

3.3. 主机协议栈设计

主机协议栈主要是对接入的 USB 设备进行枚举，并且解析描述符，再根据描述符找到对应的 USB 类驱动，并初始化类驱动，最后提供用户可用的 API 供调用，对拔出的设备，进行资源的释放。关于主机协议栈整体调用，如图 3 所示。

在主机中，有一个叫做 hub 的概念，USB hub (USB 集线器)是一种允许多个 USB 设备连接到一个单个 USB 端口的设备。它基本上是 USB 接口的分线器或扩展器。每个 hub 上的 USB 接口，称为 hubport，每个 hubport 可以接入一个 USB 设备，因此 USB 设备的枚举就是通过 hub 上的 hubport 进行。而 hub 又分 roothub 和 external hub，roothub 代表主机控制器本身，external hub 自身是一个 USB 设备，具备连接多个 USB 设备的功能，如图 4 所示。因此在 CherryUSB 中会创建一个 hub 守护线程，作用就是探测 hub 上所有 USB 接口是否有插入和拔出事件，如果是插入事件，则进行枚举和驱动加载，如果是拔出事件则进行资源释放。如果接入的是 hub 设备，则通过 hub 通信继续检测插拔事件，最终都会通过统一的一个 hub 守护线程去操作 hub 上的所有 USB 端口。

4. 基于 CherryUSB 的应用开发

本章中，我们将基于 RT-Thread ART-PI2 开源硬件，进行 CherryUSB 的主机和从机移植和应用开发。

4.1. 使用从机枚举虚拟串口

首先根据 CherryUSB 官方文档(https://cherryusb.cherry-embedded.org/quick_start/transplant#usb-device)完成从机的基础移植，主要包含非 USBIP 相关的内容，包括 USB 引脚，USB 时钟，USB 中断等，这些不属于 USBIP 驱动中的内容。由于我们使用 RT-Thread ART-PI2，可以直接使用 ART-PI2 官方仓库(https://github.com/RT-Thread-Studio/sdk-bsp-stm32h7r-realthread-artpi2)，并选择 art_pi2_cherryusb_usbdev_cdc_acm 工程编译即可。具体则是在 env 环境中使用 menuconfig 勾选 CherryUSB 模块，选择 USBIP 和 cdc acm 类和 demo 并保存，如图 5 所示。

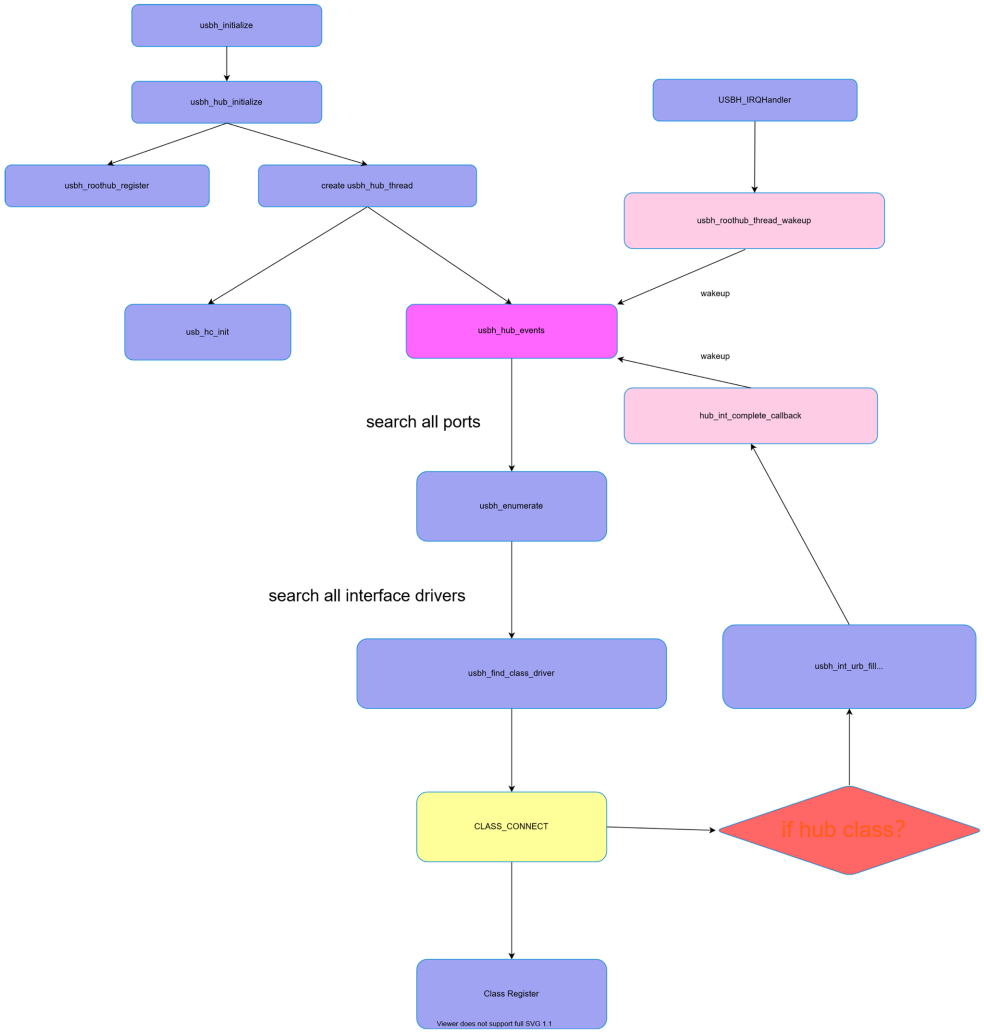

Figure 3. CherryUSB host stack framework
图 3. CherryUSB 主机协议栈框图

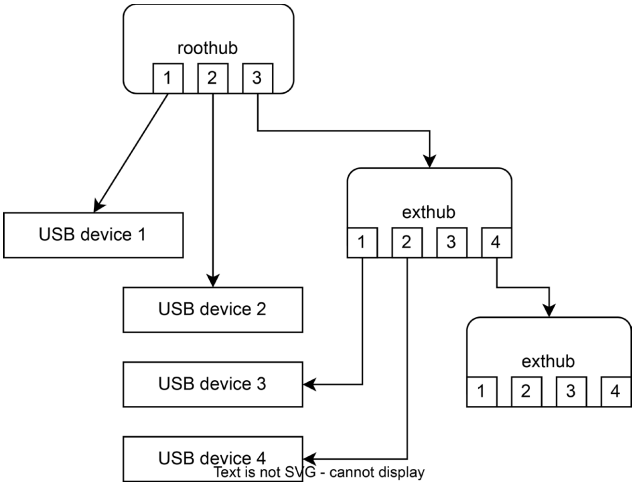

Figure 4. USB hub topological structure
图 4. USB hub 拓扑结构

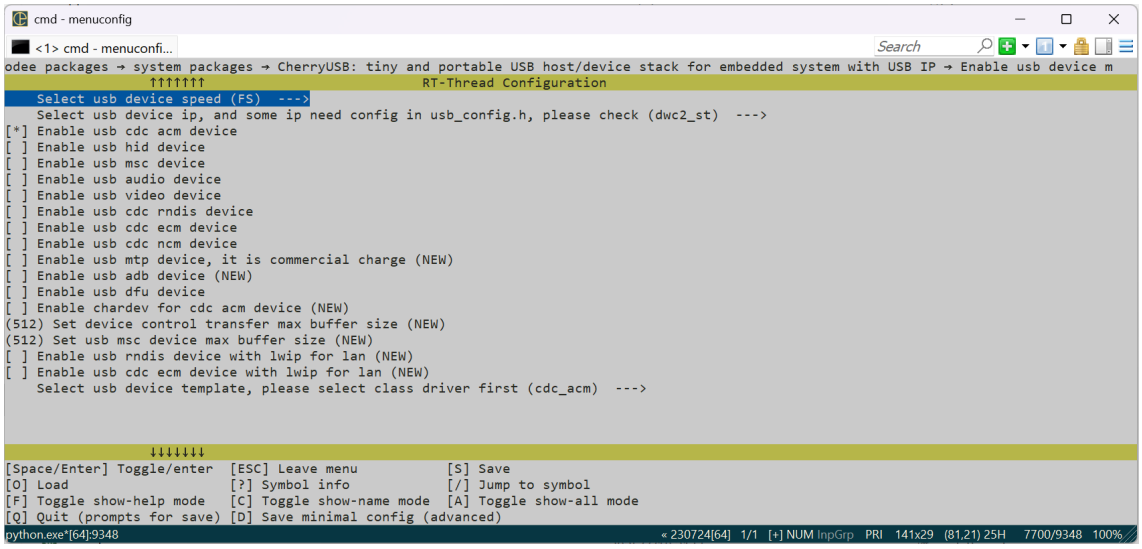

Figure 5. CherryUSB device menuconfig
图 5. CherryUSB 从机 menuconfig 配置

然后执行 scons --target=mdk5，并在 main 函数中调用 cdc_acm_init 初始化 cdc acm 设备，然后编译构建即可。最终将会在电脑上显示一个 USB COM 口，并可以进行数据的通信。如图 6 所示。

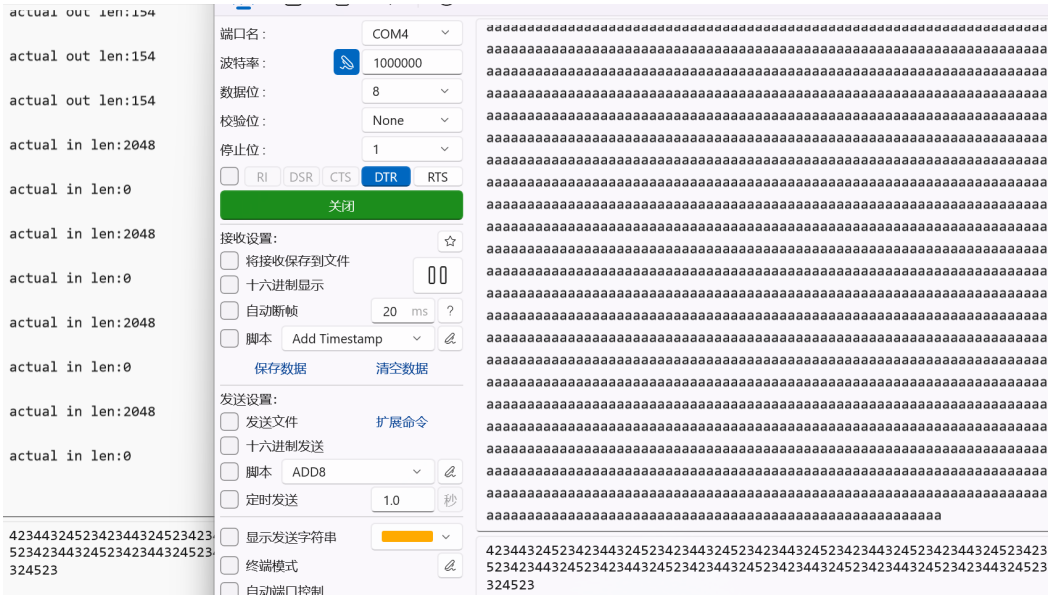

Figure 6. CDC ACM read write test
图 6. CDC ACM 读写测试

4.2. 使用主机枚举 U 盘

主机的移植可以参考 CherryUSB 官方文档 (https://cherryusb.cherry-embedded.org/quick_start/transplant#usb-host)，同样在 ART-PI2 官方仓库中也包含主机的例程，选择 art_pi2_cherryusb_usbhost 工程[6]，并使用 menuconfig 勾选 USBIP 和 MSC 类设备并保存，如图 7 所示。CherryUSB 将会对接到 RT-Thread 的 DFS (虚拟文件系统)组件中。

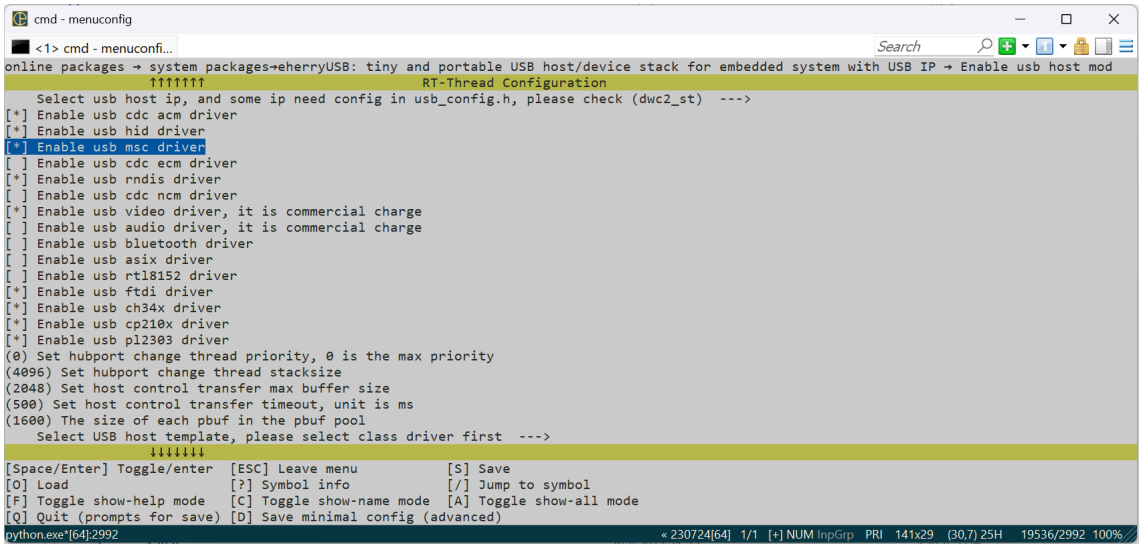

Figure 7. CherryUSB host menuconfig
图 7. CherryUSB 主机 menuconfig 配置

然后执行 scons--target=mdk5，并在 main 函数中使用 usbh_initialize 初始化主机协议栈初始化，然后编译构建即可。

当我们插入一个 U 盘时，协议栈会识别并进行枚举，加载 MSC 驱动，最终注册到 DFS 组件中，可以使用‘ls’，‘cat’，‘echo’相关 API 进行访问。如图 8 所示。

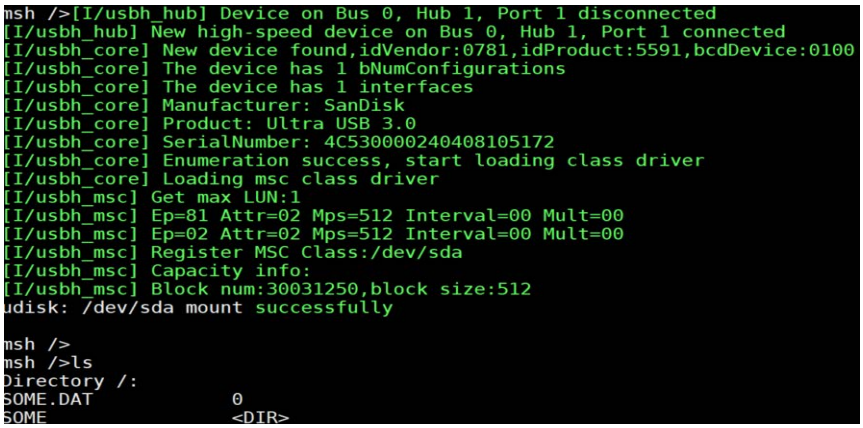

Figure 8. CherryUSB host flash drive test
图 8. CherryUSB 主机接入 U 盘测试

5. 结语

CherryUSB 的设计是为了让用户更好地理解和使用 USB，将复杂的功能采用简易代码和类比等形式简化，并实现了众多 USB 类设备，统一了 USBIP 驱动，降低了用户移植到新嵌入式系统中的难度，也方便了嵌入式开发者进行应用开发或者是 IP 验证。CherryUSB 希望通过这些来吸引更多的开发者和企业用户。希望通过本文，读者对 CherryUSB 的原理有个初步的了解，不必拘泥于复杂的 USB 概念，而是从 USBIP 设计入手去发现规律，总结规律，从而形成一个大体的框架，最终对 USB 的整体会有一个很深入的理解。

CherryUSB 近年来得到了很多开发者和企业的支持，在很多产品当中都有 CherryUSB 的身影，项目也是趋于高度稳定，期待 CherryUSB 在未来能够支持更多的硬件支持，支持更多的开源项目，成为一个主流标准的 USB 协议栈[7]。

参考文献

[1] CherryUSB 官方文档[EB/OL]. https://cherryusb.cherry-embedded.org, 2025-08-04.

[2] The Micriμm USB Team. 嵌入式协议栈 uc/USB-Device [M]. 何小庆, 译. 北京: 航空航天大学出版社, 2015.

[3] 刘荣. 圈圈教你玩 USB [M]. 第 3 版. 北京: 航空航天大学出版社, 2022.

[4] USB Implementers Forum (2000) Universal Serial Bus Specification Revision 2.0. https://www.usb.org/document-library/usb-20-specification

[5] (2000) Linux Project USB Request Block. https://docs.linuxkernel.org.cn/driver-api/usb/URB.html

[6] (2025) RT-Thread Artpi2 Project. https://github.com/RT-Thread-Studio/sdk-bsp-stm32h7r-realthread-artpi2

[7] 王小强. 多种嵌入式平台通用 USB2.0 协议栈的研究与设计[D]: [硕士学位论文]. 成都: 电子科技大学, 2009.