Embedded Linux Device Drivers

Agenda

- USB Device Driver

USB Protocol Fundamentals

1. USB Architectural Overview

Key Components:

Component	Description
Host	Controller (e.g., xHCI in BBB) managing the bus
Device	Peripheral (mouse, keyboard, etc.)
Hub	Multi-port repeater (BBB has 1 root hub)

Speed Standards:

Type	Speed	Linux Identifier
Low Speed (LS)	1.5 Mbps	`USB_SPEED_LOW`
Full Speed (FS)	12 Mbps	`USB_SPEED_FULL`
High Speed (HS)	480 Mbps	`USB_SPEED_HIGH`
SuperSpeed (SS)	5 Gbps	`USB_SPEED_SUPER`

2. USB Protocol Layers

Electrical Layer:

- Differential signaling (D+/D- lines)
- NRZI encoding with bit stuffing

Packet Structure:

```
[SYNC][PID][ADDR/ENDP][DATA][CRC][EOP]
```

- **PID Types**:
  - Token (IN, OUT, SETUP)
  - Data (DATA0, DATA1)
  - Handshake (ACK, NAK, STALL)

Transfer (Endpoint) Types:

Type	Characteristics	Linux Macro	Use Case
Control	Bidirectional, setup phase	`USB_ENDPOINT_XFER_CONTROL`	Device configuration
Interrupt	Periodic, small payload	`USB_ENDPOINT_XFER_INT`	HID devices
Bulk	Reliable, large transfers	`USB_ENDPOINT_XFER_BULK`	Mass storage
Isochronous	Time-sensitive, no retries	`USB_ENDPOINT_XFER_ISOC`	Audio/video

3. USB Descriptors

Hierarchical structure defining device capabilities:

Descriptor Types:

```
struct usb_descriptor_header {
    __u8 bLength;
    __u8 bDescriptorType;
};
```

1. **Device Descriptor**:

```
struct usb_device_descriptor {
    __u8 bLength;            // 18 bytes
    __u8 bDescriptorType;    // USB_DT_DEVICE
    __le16 idVendor;         // Important for driver binding
    __le16 idProduct;
    // ... other fields
};
```

2. **Configuration Descriptor**:

```
struct usb_config_descriptor {
    __u8 bLength;            // 9 bytes
    __u8 bDescriptorType;    // USB_DT_CONFIG
    __le16 wTotalLength;     // Total length of all descriptors
    __u8 bNumInterfaces;     // Number of interfaces
    // ... other fields
};
```

3. **Interface Descriptor**:

```
struct usb_interface_descriptor {
    __u8 bLength;            // 9 bytes
```

```
    __u8 bDescriptorType;    // USB_DT_INTERFACE
    __u8 bInterfaceNumber;   // Used in driver binding
    __u8 bAlternateSetting;
    __u8 bNumEndpoints;
    __u8 bInterfaceClass;    // Key for class drivers
    // ... other fields
};
```

4. **Endpoint Descriptor**:

```
struct usb_endpoint_descriptor {
    __u8 bLength;            // 7 bytes
    __u8 bDescriptorType;    // USB_DT_ENDPOINT
    __u8 bEndpointAddress;   // Bit 7: direction (IN=1, OUT=0)
    __u8 bmAttributes;       // Transfer type
    __le16 wMaxPacketSize;
    // ... other fields
};
```

4. USB Device Classes

A. HID (Human Interface Device)

Class Code: `0x03`
Examples: Keyboards, Mice, Game Controllers
Descriptor Structure:

```
struct hid_descriptor {
    __u8  bLength;
    __u8  bDescriptorType;  // USB_DT_HID
    __le16 bcdHID;          // HID spec version
```

```
    __u8  bCountryCode;
    __u8  bNumDescriptors;  // Number of subordinate descriptors
    __u8  bDescriptorType0; // Report descriptor type
    __le16 wDescriptorLength0;
};
```

Endpoints:

- **Mandatory**: 1 Interrupt IN (Device → Host)
- **Optional**: 1 Interrupt OUT (Host → Device)
  Linux Driver: `drivers/hid/usbhid/`

B. MSC (Mass Storage Class)

Class Code: `0x08`
Subclasses:

- `0x06` (SCSI Transparent)
- `0x04` (UFI)
  Protocols:
- `0x50` (Bulk-Only Transport - BOT)
- `0x62` (USB Attached SCSI - UAS)

Descriptor Requirements:

- 1 Interface
- 2 Bulk Endpoints (IN/OUT)
  Data Flow:

```
[CBW (31 bytes)] → [DATA (optional)] → [CSW (13 bytes)]
```

Linux Driver: `drivers/usb/storage/`

C. CDC (Communications Device Class)

Subclass Examples:

1. **CDC-ACM** (Modems, Serial Converters)
    - 2 Interfaces:
        - Control (COMM)
        - Data (DATA)
    - Endpoints:
        - 1 Interrupt IN (optional)
        - 2 Bulk (IN/OUT)
2. **CDC-ECM** (Ethernet Networking)
    - 1 Data Interface
    - 2 Bulk Endpoints

Linux Drivers:

- `drivers/usb/class/cdc-acm.c`
- `drivers/net/usb/`

D. Audio Class (UAC)

Versions:

- UAC1 (Class Code `0x01`)
- UAC2 (Class Code `0x01` with different descriptors)

Typical Configuration:

- 1 Audio Control Interface
- 1+ Audio Streaming Interfaces

Endpoints:

- 1 Isochronous IN (Recording)
- 1 Isochronous OUT (Playback)
- 1 Interrupt IN (Status)

Descriptor Complexity:

```
struct uac1_as_header_descriptor {
    __u8  bLength;            // 7+ bytes
    __u8  bDescriptorType;    // USB_DT_CS_INTERFACE
    __u8  bDescriptorSubtype; // UAC_AS_GENERAL
    __u8  bTerminalLink;
    __u8  bDelay;
    __le16 wFormatTag;        // PCM/MPEG/etc
};
```

Linux Driver: `sound/usb/`

E. USB Video Class (UVC)

Class Code: `0x0E`
Descriptor Hierarchy:

1. Video Control Interface
    - Input Terminal
    - Output Terminal
    - Processing Unit
2. Video Streaming Interface
   Endpoints:

- 1 Bulk/Interrupt IN (Control)
- 1 Isochronous IN (Video Data)

Packet Structure:

```
[UVC Header (12B)][Payload][Footer (2B)]
```

Linux Driver: `drivers/media/usb/uvc/`

Class Comparison Table

Class	Interfaces	Required Endpoints	Transfer Types	Typical Use
HID	1	1 INT IN	Interrupt	Input Devices
MSC	1	2 BULK (IN/OUT)	Bulk	Storage
CDC-ACM	2	1-2 BULK, 0-1 INT	Bulk+Interrupt	Serial Ports
Audio	2+	1-2 ISO, 1 INT	Isochronous	Audio I/O
Video	2	1 ISO, 1 INT	Isochronous	Webcams
DFU	1	0	Control	Firmware Update

- Device Classes:
  - https://community.renesas.com/the_vault/renesas_usb/drivers/m/mediagallery/3194

5. USB Request Block (URB)

Core data structure for USB communication:

URB Lifecycle:

1. Allocation (`usb_alloc_urb()`)
2. Initialization (transfer-specific)
3. Submission (`usb_submit_urb()`)
4. Completion (callback or synchronous wait)
5. Deallocation (`usb_free_urb()`)

Example URB for Bulk Transfer:

```
struct urb *urb = usb_alloc_urb(0, GFP_KERNEL);
unsigned char *buf = kmalloc(BUF_SIZE, GFP_KERNEL);

usb_fill_bulk_urb(urb, udev,
                  usb_sndbulkpipe(udev, ep_addr),
                  buf, BUF_SIZE,
                  urb_completion_cb, NULL);

int status = usb_submit_urb(urb, GFP_KERNEL);
```

6. Linux USB Subsystem

Key Components:

- **Host Controller Driver** (e.g., `xhci-hcd`)
- **USB Core** (`drivers/usb/core/`)
- **Class Drivers** (hid, storage, etc.)
- **Device Drivers**

Driver Types:

Type	Description	Example
Host Controller	Manages USB host hardware	`xhci-hcd`
Hub Driver	Handles USB hubs	`hub.c`
Class Driver	Generic for device classes	`usb-storage`
Vendor Driver	Device-specific	Custom drivers

7. Practical USB Driver Development

-

Probing Sequence:

1. Match based on IDs or class

```
static struct usb_device_id my_table [] = {
    { USB_DEVICE(VENDOR_ID, PRODUCT_ID) },
    { } // Terminate
};
MODULE_DEVICE_TABLE(usb, my_table);
```

2. Register driver:

```
static struct usb_driver my_driver = {
    .name = "my_usb_drv",
    .probe = my_probe,
    .disconnect = my_disconnect,
    .id_table = my_table,
};
```

3. Handle endpoints in probe():

```
struct usb_endpoint_descriptor *epdesc;
struct usb_host_interface *interface = intf->cur_altsetting;

for (i = 0; i < interface->desc.bNumEndpoints; i++) {
    epdesc = &interface->endpoint[i].desc;
    if (usb_endpoint_is_bulk_in(epdesc))
        bulk_in_ep = usb_rcvbulkpipe(udev, epdesc->bEndpointAddress);
}
```

USB Skeleton Device Driver

1. Create USB device

- Create Simple USB CDC device on STM32
  - https://medium.com/@pasindusandima/stm32-usb-virtual-com-port-vcp-bc7cb1bd5f5

2. Driver Framework

```
#include <linux/module.h>
#include <linux/usb.h>

#define VENDOR_ID  0x1234  // Replace with your device's VID
#define PRODUCT_ID 0x5678  // Replace with your device's PID

/* Device-specific structure */
struct usb_skel {
    struct usb_device *udev;
    struct usb_interface *interface;
    unsigned char *bulk_in_buffer;
    size_t bulk_in_size;
    __u8 bulk_in_endpointAddr;
    __u8 bulk_out_endpointAddr;
};

/* Table of devices supported by this driver */
static struct usb_device_id skel_table[] = {
    { USB_DEVICE(VENDOR_ID, PRODUCT_ID) },
    { } /* Terminating entry */
};
MODULE_DEVICE_TABLE(usb, skel_table);

/* USB probe function */
```

```c
static int skel_probe(struct usb_interface *interface,
                      const struct usb_device_id *id)
{
    struct usb_skel *dev;
    struct usb_host_interface *iface_desc;
    struct usb_endpoint_descriptor *endpoint;
    int i;
    int retval = -ENOMEM;

    /* Allocate memory for our device structure */
    dev = kzalloc(sizeof(*dev), GFP_KERNEL);
    if (!dev)
        return -ENOMEM;

    dev->udev = usb_get_dev(interface_to_usbdev(interface));
    dev->interface = interface;

    /* Find endpoints */
    iface_desc = interface->cur_altsetting;
    for (i = 0; i < iface_desc->desc.bNumEndpoints; ++i) {
        endpoint = &iface_desc->endpoint[i].desc;

        if (usb_endpoint_is_bulk_in(endpoint)) {
            dev->bulk_in_endpointAddr = endpoint->bEndpointAddress;
            dev->bulk_in_size = le16_to_cpu(endpoint->wMaxPacketSize);
            dev->bulk_in_buffer = kmalloc(dev->bulk_in_size, GFP_KERNEL);
            if (!dev->bulk_in_buffer) {
                retval = -ENOMEM;
                goto error;
            }
        }

        if (usb_endpoint_is_bulk_out(endpoint))
            dev->bulk_out_endpointAddr = endpoint->bEndpointAddress;
    }
```

```
    /* Save our device structure in the interface */
    usb_set_intfdata(interface, dev);

    /* Device is ready */
    dev_info(&interface->dev, "USB Skeleton Device now attached\n");
    return 0;

error:
    if (dev) {
        kfree(dev->bulk_in_buffer);
        usb_put_dev(dev->udev);
        kfree(dev);
    }
    return retval;
}

/* USB disconnect function */
static void skel_disconnect(struct usb_interface *interface)
{
    struct usb_skel *dev = usb_get_intfdata(interface);

    usb_set_intfdata(interface, NULL);

    /* Clean up */
    kfree(dev->bulk_in_buffer);
    usb_put_dev(dev->udev);
    kfree(dev);

    dev_info(&interface->dev, "USB Skeleton Device disconnected\n");
}

/* USB driver structure */
static struct usb_driver skel_driver = {
    .name = "usb-skel",
    .probe = skel_probe,
    .disconnect = skel_disconnect,
```

```
    .id_table = skel_table,
};

module_usb_driver(skel_driver);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("USB Skeleton Driver");
```

3. Key Components Explained

1. Device Identification

```
static struct usb_device_id skel_table[] = {
    { USB_DEVICE(VENDOR_ID, PRODUCT_ID) },
    { }
};
```

- Replace `VENDOR_ID` and `PRODUCT_ID` with your device's identifiers
- Can match by class using `USB_INTERFACE_INFO()`

2. Probe Function Essentials

1. **Device Allocation**:

   ```
   dev = kzalloc(sizeof(*dev), GFP_KERNEL);
   ```

2. **Endpoint Discovery**:

```
for (i = 0; i < iface_desc->desc.bNumEndpoints; ++i) {
    endpoint = &iface_desc->endpoint[i].desc;
    if (usb_endpoint_is_bulk_in(endpoint)) {
        /* Setup input endpoint */
    }
}
```

3. **Interface Data Attachment**:

```
usb_set_intfdata(interface, dev);
```

3. Disconnect Handling

- Must clean up all allocated resources
- Release USB device reference:

```
usb_put_dev(dev->udev);
```

4. Adding Basic I/O Operations

Bulk Write (Host → Device)

```
static int skel_write(struct usb_skel *dev, void *data, size_t len)
{
    int actual_length;
    int retval;

    retval = usb_bulk_msg(dev->udev,
```

```
                        usb_sndbulkpipe(dev->udev,
                                        dev->bulk_out_endpointAddr),
                        data, len, &actual_length,
                        HZ * 5); /* 5 second timeout */

    if (retval < 0)
        dev_err(&dev->interface->dev, "Write error: %d\n", retval);

    return retval;
}
```

Bulk Read (Device → Host)

```
static int skel_read(struct usb_skel *dev, void *data, size_t len)
{
    int actual_length;
    int retval;

    retval = usb_bulk_msg(dev->udev,
                        usb_rcvbulkpipe(dev->udev,
                                        dev->bulk_in_endpointAddr),
                        data, len, &actual_length,
                        HZ * 5); /* 5 second timeout */

    if (retval < 0)
        dev_err(&dev->interface->dev, "Read error: %d\n", retval);

    return retval ? retval : actual_length;
}
```

5. Building & Testing

Compilation:

```
obj-m := usb-skel.o
KDIR := /lib/modules/$(uname -r)/build
PWD := $(shell pwd)

all:
    make -C $(KDIR) M=$(PWD) modules
```

Load/Unload:

```
sudo insmod usb-skel.ko
# Plug in your USB device
dmesg | tail # Check probe messages

sudo rmmod usb-skel
# Device disconnect messages appear
```