MT793X IoT SDK for USB

User Guide

Version:	0.2
Release date:	2023-01-05

Use of this document and any information contained therein is subject to the terms and conditions set forth in Exhibit 1. This document is subject to change without notice.

Version History

Version	Date	Description
0.1	2021-04-01	Initial draft
0.2	2023-01-05	USB audio Class driver Usage

Table of Contents

List of Figures

List of Tables

1 Overview

The software architecture of the USB (Universal Serial Bus) module is as follows:

- PHY Driver
- Host Controller Drivers
- Gadget Class Driver: supports VCOM class driver
- Gadget Controller Driver
- Dual Role Switch Driver

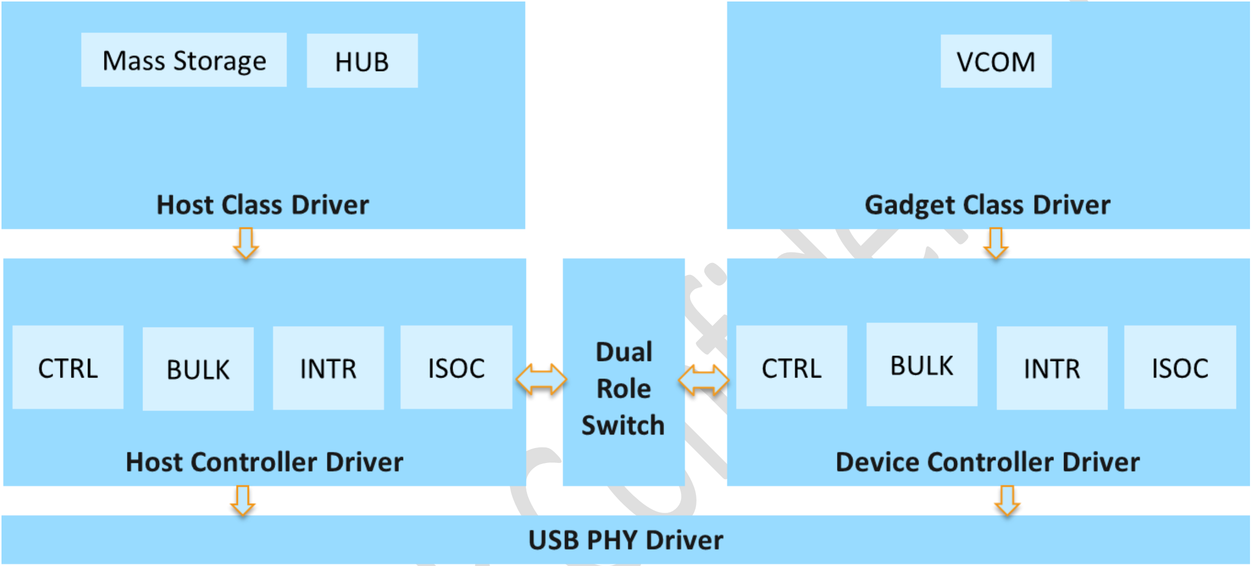


Figure 1. USB Driver Architecture

2 USB Mode Configuration

USB Mode	Configurable

Host Only	Can be **Configured** by feature.mk under project folder
	(e.g. project\mt7933_hdk\apps\bga_sdk_demo\GCC)
	MTK_SSUSB_HOST_ENABLE = y
	MTK_SSUSB_GADGET_ENABLE = n

Device Only	Can be **Configured** by
	1. feature.mk under project\mt7933_hdk\apps\bga_sdk_demo\GCC folder
	MTK_SSUSB_HOST_ENABLE = n
	MTK_SSUSB_GADGET_ENABLE = y
	2. header file (driver\chip\mt7933\inc\hal_usb_mtu3_rscs.h)
	dr_mode = USB_DR_MODE_PERIPHERAL(hal_usb_mtu3_rscs.h)

Dual Role	Can be **Configured** by
	1. feature.mk under project\mt7933_hdk\apps\bga_sdk_demo\GCC folder
	MTK_SSUSB_HOST_ENABLE = y
	MTK_SSUSB_GADGET_ENABLE = y
	2. header file (driver\chip\mt7933\inc\hal_usb_mtu3_rscs.h)
	dr_mode = USB_DR_MODE_OTG(hal_usb_mtu3_rscs.h)

Table 1. USB Mode Configuration

3 USB PHY Driver

3.1 PHY Driver Introduction

Both the host and device need to describe PHY resource in header file hal_usb_mtu3_rscs.h and hal_usb_xhci_rscs.h. The host and device drivers initialize the PHY based on the resource.

3.2 PHY Usage Example

3.2.1 Host PHY Usage Example

File path: driver/chip/mt7933/inc/hal_usb_xhci_rscs.h

```
static struct u2phy_banks xhci_u2_banks[U2_PHY_NUM] = {
        {
                .misc = (void *)SSUSB_SIFSLV_U2PORT0_MISC_BASE,
                .fmreg = (void *)SSUSB_SIFSLV_U2PORT0_U2FREQ_BASE,
                .com = (void *)SSUSB_SIFSLV_U2PORT0_COM_BASE,
        },
};
static struct u2phy_banks *u2_banks[U2_PHY_NUM] = {
        &(xhci_u2_banks[0]),
};
```

3.2.2 Device PHY Usage Example

File path: driver/chip/mt7933/inc/hal_usb_mtu3_rscs.h

```
static struct u2phy_banks mtu3_u2_banks[U2_PHY_NUM] = {
        {
                .misc = (void *)SSUSB_SIFSLV_U2PORT0_MISC_BASE,
                .fmreg = (void *)SSUSB_SIFSLV_U2PORT0_U2FREQ_BASE,
                .com = (void *)SSUSB_SIFSLV_U2PORT0_COM_BASE,
        },
};
static struct u2phy_banks *u2_banks[U2_PHY_NUM] = {
        &(mtu3_u2_banks[0]),
};
```

4 USB Host Driver

4.1 Host Controller Driver Introduction

The host needs to describe xHCI resource in header file hal_usb_xhci_rscs.h. The host drivers initializes the xHCI hardware based on the resource.

The initialization of the host driver only requires the calling of the API, mtk_usb_host_init.

4.1.1 xHCI Usage Example

```
/*
 * XHCI_INSTANCE: indicate HW IP number
 * usb resource
 */
#define XHCI_INSTANCE 1
#define XHCI_U2_PHY_NUM 1
#define XHCI_PHY_TYPE MTK_TPHY
#define XHIC_INT_ID 48

#define VBUS_NUM 1
#define PORT0_VBUS_GPIO 31

/*
 * xhci power domain
 */
#define XHCI_USB_POWER_DOMAIN

/*
 * xhci clk resource
 */
#define XHCI_REF_CLK
#define XHCI_SYS_CLK
#define XHCI_CLK CLK
#define XHCI_MCU_CLK
#define XHCI_DMA_CLK

/*
 * xhci ip sleep setting
 */
#define PERI_SSUSB_SPM_GLUE_CG 0x300d0700
#define RG_SSUSB_SPM_INT_EN BIT(1)
#define RG_SSUSB_IP_SLEEP_EN BIT(4)
```

```
static u32 xhci_vbus[VBUS_NUM] = {
                PORT0_VBUS_GPIO,
};

static u32 *vbus_gpio[VBUS_NUM] = {
        &(xhci_vbus[0]),
};

static struct xhci_hcd_mtk mtk_hcd[XHCI_INSTANCE] = {
        {
                .xhci_base = (void *)(USB_BASE),
                .ippc_base = (void *)SSUSB_SIFSLV_IPPC_BASE,
                .xhci_irq = XHIC_INT_ID,
                .vbus_gpio = (u32 **)&vbus_gpio,
                /* phy resource */
                .phy.u2_phy_num = XHCI_U2_PHY_NUM,
                .phy.u3_phy_num = XHCI_U3_PHY_NUM,
                .phy.type = XHCI_PHY_TYPE,
                .phy.u2_banks = (struct u2phy_banks **)&u2_banks,
        },
};
```

4.2 Host Class Driver Introduction

At present, the host supports mass storage and hub class driver.
In the enumeration phase, different devices are matched to the corresponding class driver.

4.2.1 Mass Storage Class Driver Usage

You need to operate the mass storage (Udisk) device through the file system.

Example:

- Mount Udisk Usage command
    - ff mount USB
- List Udisk root directory File Usage command
    - ff ls USB:/
- Read/write Udisk File Usage command
    - Write Usage: ff write <file name> <content>
      ff write USB:/test.txt 123456789abcd
    - Read Usage: ff read <file name>
      ff read USB:/test.txt

4.2.2 USB audio Class driver Usage

MT7933 Currently supports UAC 1.0.

Please use the below sequence to initial the uac playback and capture function:

- uac_get_device () return the current uac device
  For playback, the uac streaming is streaming [0]. For capture, the uac streaming is streaming [1].

[Playback]:

- uac_reinit_ring_buffer (), use to re-initial the playback ring buffer before initial playback.
- uac_get_pl_buffer_write_pos (), use to get the write position and length of the playback ring buffer.
- uac_set_pl_buffer_write_done (), use to update the write position of the ring buffer.
- uac_register_playback_callback (), **optional**, register the callback to receive the notification when the data is not enough when playback.

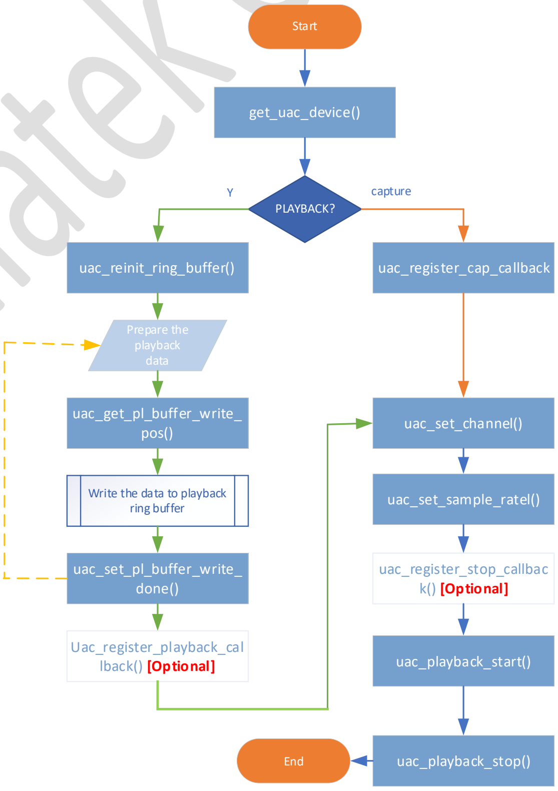
[Capture]

- uac_register_cap_callback (), register the callback function to receive the capture data from speaker.

- uac_set_channel (), use to set the playback or capture streaming channel parameter.
- uac_sel_sample_rate (), use to set the playback or capture streaming sample rate parameter.
- uac_register_stop_callback (), **optional**, register stop callback to uac driver, and receive the stop notification.
- uac_playback_start (), when all resource is ready, use this API to start the playback or capture process.

If you want to stop the playback or capture process, please call the uac_playback_stop () API to trigger it stop.

The flow chart as below:


Start
get_uac_device()
Y
PLAYBACK?
capture
uac_reinit_ring_buffer()
uac_register_cap_callback
Prepare the playback data
uac_get_pl_buffer_write_pos()
uac_set_channel()
Write the data to playback ring buffer
uac_set_sample_ratel()
uac_set_pl_buffer_write_done()
uac_register_stop_callback() [Optional]
Uac_register_playback_callback() [Optional]
uac_playback_start()
End
uac_playback_stop()

5 USB Device Driver

5.1 Device Controller Driver Introduction

The device needs to describe mtu3 resource in header file hal_usb_mtu3_rscs.h. The device drivers initialize the mtu3 hardware based on the resource.

5.1.1 MTU3 Usage Example

```
/*
 * SSUSB_IP: indicate HW IP number
 * usb resource
 */
#define SSUSB_IP 1
#define U2_PHY_NUM 1
#define PHY_TYPE MTK_TPHY
#define VBUS_VALID 33
#define IDDIG_GPIO 34
#define IDDIG_EINT HAL_EINT_NUMBER_26

#define P0_VBUS_GPIO 31
#define SSUSB_DEV_INT_ID 50
/*
 * device power domain
 */
#define SSUSB_POWER_DOMAIN

/*
 * device clk resource
 */
#define SSUSB_REF_CLK
#define SSUSB_SYS_CLK
#define SSUSB_XHCI_CLK
#define SSUSB_MCU_CLK
#define SSUSB_DMA_CLK

static struct ssusb_mtk ssusb_rscs[SSUSB_IP] = {
        {
                .u3d.mac_base = (void *)(SSUSB_DEV_BASE),
                .u3d.dev_irq = SSUSB_DEV_INT_ID,
                .ippc_base = (void *)SSUSB_SIFSLV_IPPC_BASE,
                .iddig_gpio = IDDIG_GPIO,
                .vbus_gpio = P0_VBUS_GPIO,
                .dr_mode = USB_DR_MODE_OTG,
                /* phy resource */
                .phy.u2_phy_num = U2_PHY_NUM,
                .phy.u3_phy_num = U3_PHY_NUM,
                .phy.type = PHY_TYPE,
                .phy.u2_banks = (struct u2phy_banks **)&u2_banks,
        },
```

5.2 Gadget Class Driver Introduction

At present, the device only supports CDC class driver.

5.2.1 CDC Class Driver Usage

Please use the API, serial_usb_init, to initialize device controller driver and bind CDC class driver. When serial_configured returns true, you can use the API, serial_usbtty_putcn/serial_usbtty_getcn, to write/read data.

5.2.2 CDC Class Driver API

```
int serial_usb_init(void)
int serial_configured(void)
void serial_usbtty_putcn(char *buf, int count)
void serial_usbtty_getcn(char *buf, int count)
```

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.