NXP Semiconductors
User's Guide

Document identifier: MCUXSDKUSBSHOSTCOMPUG
Rev. 2, 20 May 2020

MCUXpresso SDK USB Stack Composite Host User's Guide

Contents

Chapter 1
Overview

This document describes steps to implement a host that supports multiple devices based on the MCUXpresso SDK USB stack.

The USB Stack provides one host demo that supports HID mouse + HID keyboard. A user may need a host to meet its requirements, such as the ability to support different class devices like supporting an HID and an MSD device simultaneously. This document provides a step-by-step guide to create a customizable host that supports multiple devices.

Chapter 2
Introduction

Unlike the composite device that requires many steps, implementing a host that supports multiple devices is simple. The event callback function of host and class can handle attach, enumeration, and detach processing for all the devices. The process flow for this is shown in Figure 1. This figure shows a host supporting two classes, which is the same as a host supporting one class. All class-specific functionality for the devices is achieved in the class-specific task polling in the main function. The user only needs to focus on the modification of these two points.

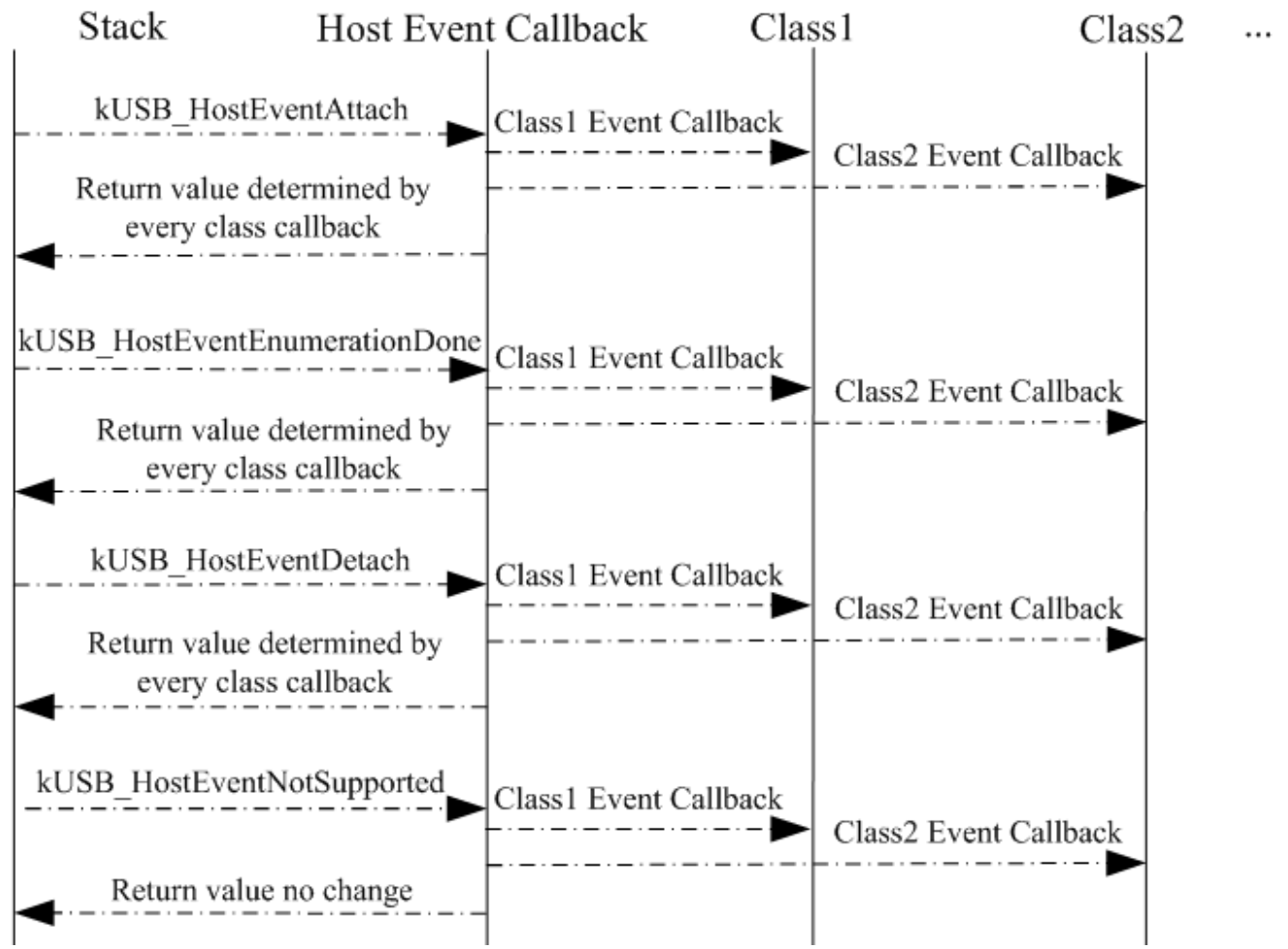

Figure 1. Process flow of event callback

Chapter 3
Detailed steps

Before developing the host that supports multiple devices, the user needs to determine:

1. How many classes this host needs to support.
2. How many subclasses for every class. For example, the HID class may include HID mouse and HID keyboard.

The code change for the host that supports HID mouse and HID keyboard is similar to that of the host supporting CDC virtual com and HID mouse.

3.1 Host event handle function

The USB_HostEvent is a common handle function for attach, unsupported device, enumeration, and detach event. This function needs to call the class-specific event handle function. When the host only supports CDC devices, the USB_HostEvent function is the following:

```
usb_status_t USB_HostEvent(usb_device_handle deviceHandle,
usb_host_configuration_handle configurationHandle,
uint32_t event_code)
{
usb_status_t status;
status = kStatus_USB_Success;
switch (event_code)
{
case kUSB_HostEventAttach:
status = USB_HostCdcEvent(deviceHandle, configurationHandle, event_code);
/* here add the new device’s event handle function */
break;
case kUSB_HostEventNotSupported:
usb_echo("device not supported.\r\n");
break;
case kUSB_HostEventEnumerationDone:
status = USB_HostCdcEvent(deviceHandle, configurationHandle, event_code);
/* here add the new device’s event handle function */
break;
case kUSB_HostEventDetach:
status = USB_HostCdcEvent(deviceHandle, configurationHandle, event_code);
/* here add the new device’s event handle function */
break;
default:
break;
}
return status;
}
```

To support other devices, add the corresponding class-specific event handle function. Additionally, it is necessary to add the local variable to receive the return value of every event handle function. The return value of USB_HostEvent should be changed according to the following occasions:

1. kUSB_HostEventAttach: if the return values for all of the class-specific event handle functions are kUSB_HostEventNotSupported, the return value of USB_HostEvent is kUSB_HostEventNotSupported.
2. kUSB_HostEventNotSupported: no change.
3. kUSB_HostEventEnumerationDone: if the return values for all of the class-specific event handle functions are not kStatus_USB_Success, the return value of USB_HostEvent is kStatus_USB_Error.

4. kUSB_HostEventDetach: if the return values for all of the class-specific event handle functions are not kStatus_USB_Success, the return value of USB_HostEvent is kStatus_USB_Error.

3.2 Class-specific device task

The main function needs to schedule every supported device's task. If the host only supports CDC devices, the class-specific task in the main function is as follows:

```
int main(void)
{
BOARD_InitHardware();
APP_init();
while (1)
{
USB_HostTaskFn(g_hostHandle);
/* cdc class task */
USB_HosCdcTask(&g_cdc);
/* here add the new device's task */
}
}
```

Chapter 4
Host MSD command + CDC virtual com example

This section provides a step-by-step example for how to implement a host that supports CDC virtual com and MSD command. This example is based on the existing host CDC virtual com example.

4.1 USB component files

Add the usb_host_msd component files, the usb_host_msd_ufi source file, and the host_msd_command component files into the current project. Normally, the host_msd_command component should be in the source folder, shown in Figure 2. The usb_host_msd component and the usb_host_msd_ufi source file should be located in the class folder showing in the Figure 3.

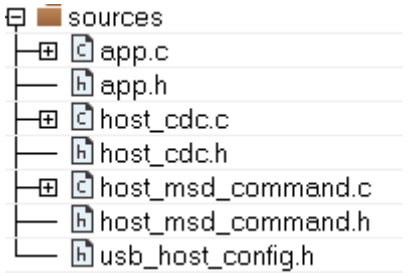

Figure 2. Source folder

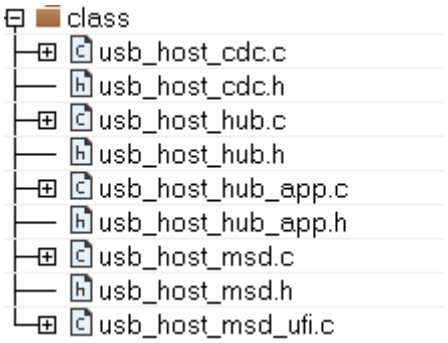

Figure 3. Class folder

4.2 USB_HostEvent function

Add the USB_HostMsdEvent function into the USB_HostEvent function.

```
usb_status_t USB_HostEvent(usb_device_handle deviceHandle,
                           usb_host_configuration_handle
configurationHandle,
                           uint32_t event_code)
{
    usb_status_t status1;
    usb_status_t status2;
    usb_status_t status = kStatus_USB_Success;

    switch (event_code)
    {
        case kUSB_HostEventAttach:
            status1 = USB_HostCdcEvent(deviceHandle, configurationHandle, event_code);
```

```
            status2 = USB_HostMsdEvent(deviceHandle, configurationHandle, event_code);
            if ((status1 == kStatus_USB_NotSupported) && (status2 ==
kStatus_USB_NotSupported))
            {
                status = kStatus_USB_NotSupported;
            }
            break;
        case kUSB_HostEventNotSupported:
            usb_echo("device not supported.\r\n");
            break;

        case kUSB_HostEventEnumerationDone:
            status1 = USB_HostCdcEvent(deviceHandle, configurationHandle, event_code);
            status2 = USB_HostMsdEvent(deviceHandle, configurationHandle, event_code);
            if ((status1 != kStatus_USB_Success) && (status2 != kStatus_USB_Success))
            {
                status = kStatus_USB_Error;
            }
            break;

        case kUSB_HostEventDetach:
            status1 = USB_HostCdcEvent(deviceHandle, configurationHandle, event_code);
            status2 = USB_HostMsdEvent(deviceHandle, configurationHandle, event_code);
            if ((status1 != kStatus_USB_Success) && (status2 != kStatus_USB_Success))
        {
            status = kStatus_USB_Error;
        }
        break;

    default:
        break;
    }
    return status;
}
```

4.3 Main function task

Add the USB_HostMsdTask function into the main function. The modified code should look like this:

```
int main(void)
{
    gpio_pin_config_t pinConfig;
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    /* enable usb host vbus */
    pinConfig.pinDirection = kGPIO_DigitalOutput;
    pinConfig.outputLogic = 1U;

    GPIO_PinInit(PTD, 8U, &pinConfig);

    APP_init();

    while (1)
    {
        USB_HostTaskFn(g_HostHandle);
        /* cdc class task */
        USB_HosCdcTask(&g_cdc);
        /* msd class task */
```

```
        USB_HostMsdTask(&g_MsdCommandInstance);
    }
}
```

Chapter 5
Revision history

The following table summarizes the changes done to this document since the initial release.

Table 1. Revision history

Revision number	Date	Substantive changes
0	11/2018	Initial release
1	12/2018	2.4.0 vs 2.5.0
2	05/2020	Updated for MCUXpresso SDK v2.8.0

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 20 May 2020

Document identifier: MCUXSDKUSBSHOSTCOMPUG

arm