USB LM3S and TM4C Host Controller User Guide

Version 1.10

For use with USBH LM3S and TM4C Host Controller versions 2.03 and above

Date: 18-Jul-2017 12:58

Table of Contents

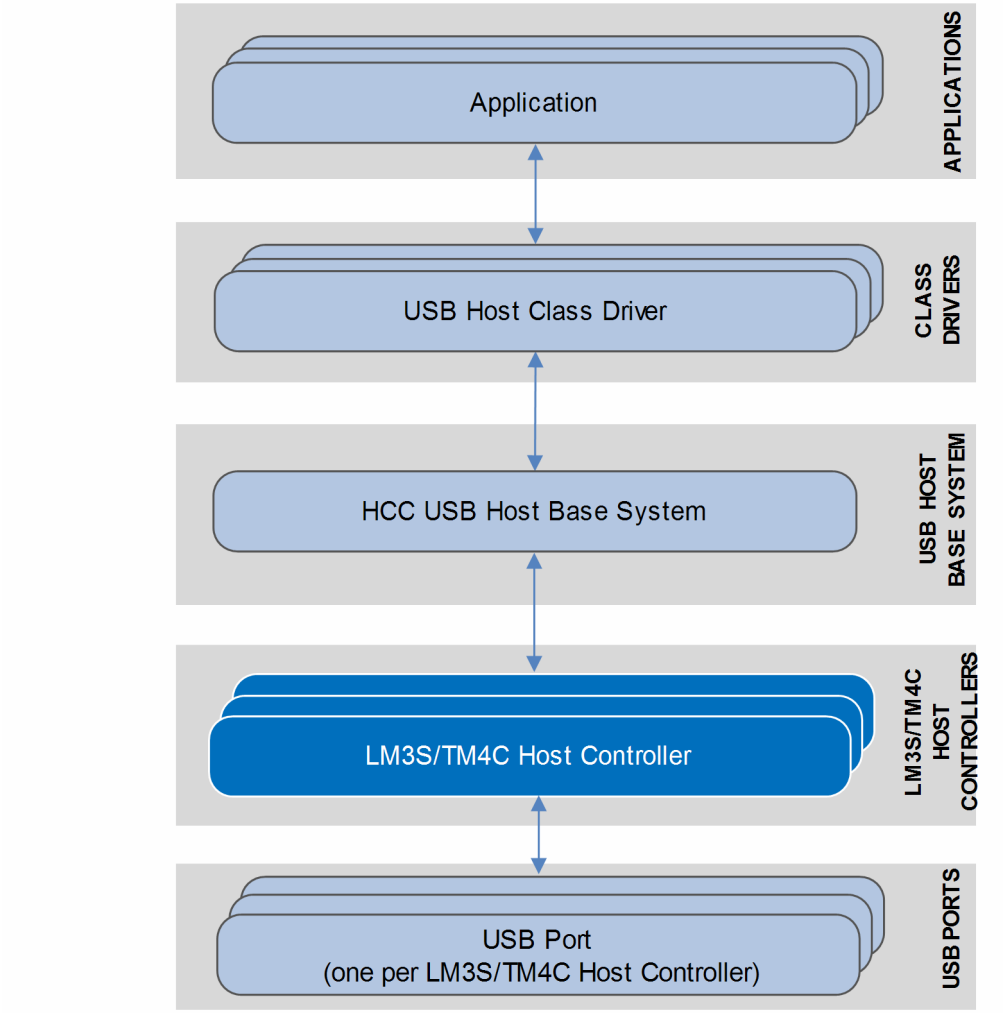
1 System Overview

1.1 Introduction

This guide is for those who want to implement HCC Embedded's LM3S/TM4C USB Host Controller with the HCC USB host stack. This supports the LM3Sxxx/TM4Cxxx MCUs from Texas Instruments Incorporated.

The LMS module provides a high speed USB 2.0 host controller which provides both full and low speed USB functions. The controller can handle all USB transfer types and, in conjunction with the USB host stack, can be used with any USB class driver.

The position of the host controller within the USB stack is shown below:



1.2 Device Overview

The TM4Cxxx devices have replaced the LM3Sxxx series, which is now obsolete. This table summarizes the properties of the various device types:

	LM3Sxxx	TM4C123xxx	TM4C129xxx
Flash	256 KB	Up to 256 KB	1 MB
SRAM	96 KB	32 KB	256 KB
		Full speed USB 2.0 OTG/Host /Device	Full speed USB 2.0 OTG/Host/Device plus USB ULPI interface

1.3 Feature Check

The main features of the host controller are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Integrated with the HCC USB Host stack and all its class drivers.
- Supports LM3Sxxx and TM4Cxxx MCUs from Texas Instruments Incorporated.
- Supports multiple simultaneous LMS controllers, each with multiple devices attached.
- Supports all USB transfer types: Control, Bulk, Interrupt, and Isochronous.

1.4 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbh_base	The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package.
usbh_drv_lms	The USB host controller package described by this document.

Documents

For an overview of HCC's embedded USB stacks, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC USB Host Base System User Guide

This document defines the USB host base system upon which the complete USB stack is built.

HCC USB LM3S and TM4C Host Controller User Guide

This is this document.

1.5 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: USB LMS and TM4C Host Controller User Guide.
- For the history of changes made to the package code itself, see History: usbh_drv_lms.

The current version of this manual is 1.10. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.10	2017-07-18	2.03	New *Change History* format.
1.00	2017-04-13	2.03	First release.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any of these files except the configuration file and PSP files.

2.1 API Header File

The file **src/api/api_usbh_lms.h** is the only file that should be included by an application using this module. It declares the usbh_lms_hc() function. For details, see Starting the Host Controller.

2.2 Configuration File

The file **src/config/config_usbh_lms.h** contains all the configurable parameters. Configure these as required. For details of these options, see Configuration Options.

2.3 Source Code

The source code files are in the directory **src/usb-host/usb-driver/lms**. **These files should only be modified by HCC**.

File	Description
lms.c	Source file for code.
lms.h	Header file for code.
lms_hc.c	Source file for the HC descriptor.
lms_hc.h	HC descriptor header file.
lms_hub.c	Source file for hub.
lms_hub.h	Header file for hub public functions.
lms_regs.h	Register file.

2.4 Version File

The file **src/version/ver_usbh_lms.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

2.5 Platform Support Package (PSP) Files

These files are in the directory **src/psp/target/usb-host-lms**. These provide functions and elements the core code may need to use, depending on the hardware.

> **Note:** These are PSP implementations for the specific microcontroller and board; you may need to modify these to work with a different microcontroller and/or development board. See PSP Porting for details.

The files are as follows:

File	Description
target/usb-host-lms/lms_hw.c	Functions source code.
target/usb-host-lms/lms-hw.h	Header file for functions.

3 Configuration Options

Set the system configuration options in the file **src/config/config_usbh_lms.h**. This section lists the available configuration options and their default values.

LMS_CHIP_TYPE

The device type. Select from the possible values in the file **api_usbh_lms.h**. The default is LMS_CHIP_TYPE_TM4C1294.

LMS_TRANSFER_TASK_STACK_SIZE

The stack size of the transfer task(s). The default is 1024.

LMS_ISR_ID

The ISR ID of the host controller. The default is INT_USB0.

LMS_INT_PRIO

The ISR priority of the host controller. The default is configKERNEL_INTERRUPT_PRIORITY.

LMS_MAX_EP

The maximum number of host channels that can be assigned to endpoints, excluding EP0s. The default is 14.

The actual number depends on the USB core implementation used. Usually this number of channels cannot be distributed between all connected devices' endpoints freely, but the the same number of IN (Host Rx) and OUT (Host Tx) endpoints is available. Consult the MCU's data sheet for the actual number of available channels.

Some example chip implementations follow:

Device	IN and OUT endpoints
LM3S3748, LM3S3749, LM3S3768	3 IN + 3 OUT
TM4C1294	7 IN + 7 OUT
LM3S9B92, LM3S9D92, LM3S9B96	15 IN + 15 OUT

LMS_MAX_DEVICE

The maximum number of Bulk and Interrupt endpoints, including EP0s. The default is 5.

4 Starting the Host Controller

This section shows how to start the host controller and describes the task created. It includes a code example.

4.1 usbh_lms_hc

This external interface function provides the host controller descriptor required by the **usbh_hc_init()** function.

Format

```
extern void * const  usbh_lms_hc
```

4.2 Host Controller Task

The host controller task handles all completed transfers. The callback requested for the transfer is executed from this task.

The task has the following attributes:

Attribute	Description
Entry point	*lms_transfer_task*
Priority	USBH_TRANSFER_TASK_PRIORITY
Stack size	LMS_TRANSFER_TASK_STACK_SIZE. The default is 1024.

4.3 Code Example

This example shows how to initialize the host controller. Note the following:

- There is only one external interface function, **usbh_lms_hc()**. To link this host controller to the system, you call the **usbh_hc_init()** function with this function as a parameter.
- The last parameter in the **usbh_hc_init()** call is the number of the host controller.

```
void start_usb_host_stack ( void )

{
int  rc;
rc = hcc_mem_init();


    if ( rc == 0 )
    {
        rc = usbh_init();    /* Initialize USB host stack */
    }

    if ( rc == 0 )
    {
        /* Attach LMS host controller */
        rc = usbh_hc_init( 0, usbh_lms_hc, 0 );
    }

    if ( rc == 0 )
    {
        rc = usbh_start();  /* Start USB host stack */
    }


    if ( rc == 0 )
    {
        rc = usbh_hc_start( 0 );  /* Start LMS Host controller */
    }

.....

}
```

5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

5.1 OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

This module requires the following OAL elements:

OAL Resource	Number Required
Tasks	1
Mutexes	1
Events	1
ISRs	1

5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The module makes use of the following standard PSP function:

Function	Package	Element	Description
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The module makes use of the following functions that must be provided by the PSP. These are designed to allow you to port them easily to work with your hardware solution. The package includes samples in the **psp_usbh_lms.c** files.

Function	Description
lms_hw_init()	Initializes the device.
lms_hw_start()	Starts the device.
lms_hw_stop()	Stops the device.
lms_hw_delete()	Deletes the device, releasing the associated resources.

These functions are described in the following sections.

Note: HCC can provide samples for different configurations; contact support@hcc-embedded.com.

lms_hw_init

This function is provided by the PSP to initialize the device.

Format

```
int lms_hw_init ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

lms_hw_start

This function is provided by the PSP to start the device.

Format

```
int lms_hw_start ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

lms_hw_stop

This function is provided by the PSP to stop the device.

Format

```
int lms_hw_stop ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

lms_hw_delete

This function is provided by the PSP to delete the device, releasing associated resources.

Format

```
int lms_hw_delete ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.