USB Synopsys® OTG Host Controller User's Guide

Version 1.20

For use with USBH Synopsys® OTG Host Controller versions 3.07 and above

Date: 24-Apr-2015 12:54

Table of Contents

System Overview 3
Introduction 3
Feature Check 4
Packages and Documents 4
Packages 4
Documents 4
Change History 5
Source File List 6
API Header File 6
Configuration File 6
Source Code Files 6
Version File 6
Platform Support Package (PSP) Files 7
Configuration Options 8
Starting the Host Controller 10
usbh_synopsys_otg_hc 10
Host Controller Task 10
Code Example 11
Integration 12
OS Abstraction Layer (OAL) 12
PSP Porting 13
synopsys_uh_hw_init 14
synopsys_uh_hw_start 15
synopsys_uh_hw_stop 16
synopsys_uh_hw_delete 17
synopsys_uh_hw_reset_start 18
synopsys_uh_hw_reset_end 19
synopsys_uh_hw_suspend 20
synopsys_uh_hw_resume 21
synopsys_uh_hw_state 22
synopsys_uh_hw_get_ms 23

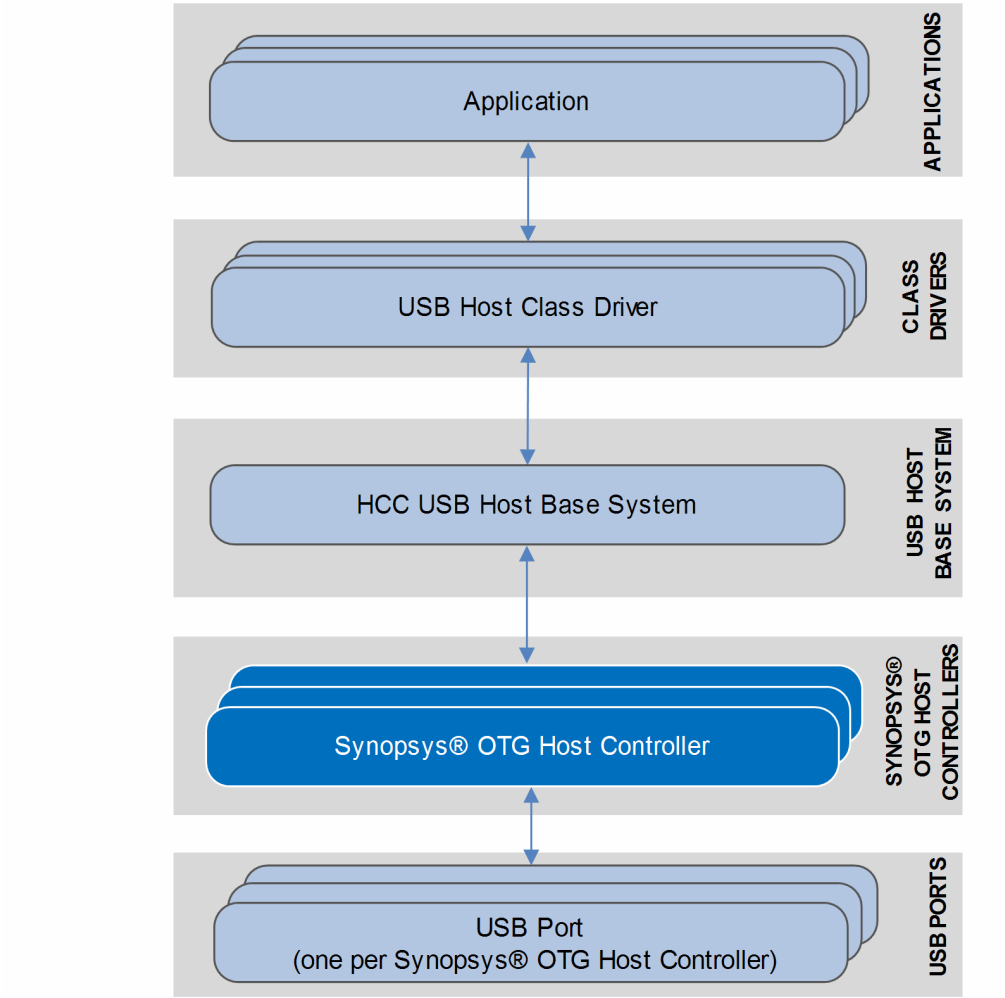
1 System Overview

1.1 Introduction

This guide is for those who want to implement HCC Embedded's Synopsys® OTG USB Host Controller module with the HCC USB host stack. This module provides a USB host controller for Synopsys® On The Go (OTG) microcontrollers; these include the STM32 connectivity line, STM32F20x, STM32F40x, Infineon XMC microcontrollers, the Silicon Labs EFM32™ family, and some Telit processors.

The Synopsys® OTG Host Controller 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 Feature Check

The main features of the host controller are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It can be used with or without an RTOS.
- It is integrated with HCC USB Host stack and all its class drivers.
- It supports microcontrollers with the Synopsys® On The Go (OTG) core. These include the STM32 connectivity line, STM32F20x, STM32F40x, Infineon XMC microcontrollers, the Silicon Labs EFM32™ family, and some Telit processors.
- It supports multiple simultaneous Synopsys® OTG controllers, each with multiple devices attached.
- It supports all USB transfer types: control, bulk, interrupt and isochronous.

1.3 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_synopsys_otg	The USB Synopsys® OTG host controller package described by this document.

Documents

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's Guide

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

HCC USB Synopsys® OTG Host Controller User's Guide

This is this document.

1.4 Change History

This section includes recent changes to this product. For a list of all the changes, refer to the file **src/history/usb-host/usbh_hc_synopsys_otg.txt** in the distribution package.

Version	Changes
3.07	Added support for Infineon XMC4500 devices. Fixed a problem that meant the core could unintentionally start acting as device after device disconnection, preventing further connections on the port. Eliminated several GCC warnings.
3.06	Fixed a problem that meant STM32's OTG_HS core could hang during initialization when used in full speed mode with internal PHY.
3.05	Added software workaround for further port connection issues. Introduced an external timer instead of using the internal frame counter of the Synopsys core.
3.04	The hardware accidentally reported a successful USB reset when a low speed device was connected, but did not start communication with the device. Introduced a software workaround for this case; this can be enabled with the new configuration option SYNOPSYS_UH_CHECK_FS_RESET. Replaced **reset()** with **synopsys_uh_hw_reset_start()** and **synopsys_uh_hw_reset_end()** in PSP files.

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_synopsys_otg.h** is the only file that should be included by an application using this module. For details, see Starting the Host Controller.

2.2 Configuration File

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

2.3 Source Code Files

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

File	Description
usbh_synopsys_otg.c	Source file for Synopsys® OTG code.
usbh_synopsys_otg.h	Header file for Synopsys® OTG public functions.
usbh_synopsys_otg_hc.c	Source file for HC descriptor.
usbh_synopsys_otg_hc.h	Descriptor header file.
usbh_synopsys_otg_hub.c	Source file for Synopsys® OTG hub.
usbh_synopsys_otg_hub.h	Header file for hub public functions.

2.4 Version File

The file **src/version/ver_usbh_synopsys_otg.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

There are three sets of files, located in directories named **src/psp_stm32f20x/target**, **src/psp_stm32f40x/target** and **src/psp_xmc4500/target**. 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 development board; you may need to modify these to work with a different microcontroller and/or board. See PSP Porting for details.

The files are as follows:

File	Description
include/hcc_stm32f20x_regs.h **include/hcc_stm32f4xx_regs.h** **include/hcc_xmc4500_regs.h**	Register definitions for the different device types.
usbh_synopsys_otg/psp_usbh_synopsys_otg.c	Hardware functions code.
usbh_synopsys_otg/psp_usbh_synopsys_otg.h	Hardware functions header file.

3 Configuration Options

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

SYNOPSYS_UH_TRANSFER_TASK_STACK_SIZE

The transfer task stack size. The default is 1024.

MAX_DEVICE

The maximum number of devices supported. The default is 8.

MAX_EP

The maximum number of bulk and interrupt endpoints. The default is 20.

ONE_NAK_PER_FRAME

Set this to 1 (the default) to allow only one NAK for an endpoint in a frame.

MAX_NP_TRANSFERS

The maximum number of NP transfers. The default is 10.

MAX_ISO_TRANSFERS

The maximum number of isochronous transfers. The default is 10.

MAX_INT_TRANSFERS

The maximum number of interrupt transfers. The default is 10.

SYNOPSYS_UH_FS_USED

Keep the default of 1 if Full Speed (FS) OTG is used. Otherwise set it to zero.

SYNOPSYS_UH_HS_USED

Set this to 1 if High Speed (HS) OTG is used. The default is zero.

SYNOPSYS_UH_CHECK_FS_RESET

Ports using the core's internal FS transceiver may fail to reset a port when a Low Speed (LS) device is connected, even though it is connected properly. Set this option to 1 if connection of LS devices is not always recognized correctly.

SYNOPSYS_UH_BASE_ADDRESS_FS

The host controller base address. The default is 0x50000000.

SYNOPSYS_UH_HOST_ISR_ID_FS

The ISR ID of the Full Speed host controller. The default is 67.

SYNOPSYS_UH_HOST_INT_PRIO_FS

The ISR priority of the Full Speed host controller. The default is 5.

FORCE_FSLS_ON_HS

Set this to a non-zero value to force the host to operate in Full Speed/Low Speed-only mode, even if the used port is capable of High Speed and the attached device is High Speed. The default is zero.

USE_INTERNAL_FS_PHY

Set this to 1 if an OTG_HS port is used with its internal FS PHY (with no external ULPI present). The default is zero.

SYNOPSYS_UH_BASE_ADDRESS_HS

The host controller base address. The default is 0x40040000.

SYNOPSYS_UH_HOST_ISR_ID_HS

The ISR ID of the High Speed host controller. The default is 77.

SYNOPSYS_UH_HOST_INT_PRIO_HS

The ISR priority of the High Speed host controller. The default is 6.

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_synopsys_otg_hc

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

Format

```
extern void * const usbh_synopsys_otg_hc
```

4.2 Host Controller Task

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

The task has the following attributes:

Attribute	Description
Entry point	One of the following: • *synopsys_uh_transfer_task_fs* for a Full Speed transfer. • *synopsys_uh_transfer_task_hs* for a High Speed transfer
Priority	OAL_HIGHEST_PRIORITY (USBH_TRANSFER_TASK_PRIORITY)
Stack size	SYNOPSYS_UH_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_synopsys_otg_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 the USB host stack */
    }
    if ( rc == 0 )
    {
        /* Attach the Synopsys host controller */
        rc = usbh_hc_init( 0, usbh_synopsys_otg_hc, 0 );
    }

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

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 host controller makes use of the following functions that must be provided by the PSP. These are designed for you to port them easily to work with your hardware solution. The package includes samples for the STM32F20x and STM32F4xx families in the **psp_usbh_synopsys_otg.c** files.

Function	Description
synopsys_uh_hw_init()	Initializes the device.
synopsys_uh_hw_start()	Starts the device.
synopsys_uh_hw_stop()	Stops the device.
synopsys_uh_hw_delete()	Deletes the device, releasing the associated resources.
synopsys_uh_hw_reset_start()	Starts a reset.
synopsys_uh_hw_reset_end()	Ends a reset.
synopsys_uh_hw_suspend()	Suspends a device.
synopsys_uh_hw_resume()	Resumes a suspended device.
synopsys_uh_hw_state()	Gets the state of a device.
synopsys_uh_hw_get_ms()	Gets the device timer value in ms.

These functions are described in the following sections.

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

synopsys_uh_hw_init

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

Format

```
int synopsys_uh_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.

synopsys_uh_hw_start

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

Format

```
int synopsys_uh_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.

synopsys_uh_hw_stop

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

Format

```
int synopsys_uh_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.

synopsys_uh_hw_delete

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

Format

```
int synopsys_uh_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.

synopsys_uh_hw_reset_start

This function is provided by the PSP to start a reset.

Format

```
void synopsys_uh_hw_reset_start ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

None.

synopsys_uh_hw_reset_end

This function is provided by the PSP to end a reset.

Format

```
void synopsys_uh_hw_reset_end ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

None.

synopsys_uh_hw_suspend

This function is provided by the PSP to suspend a device.

Format

```
void synopsys_uh_hw_suspend ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

None.

synopsys_uh_hw_resume

This function is provided by the PSP to resume a device.

Format

```
void synopsys_uh_hw_resume ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

None.

synopsys_uh_hw_state

This function is provided by the PSP to get the state of a device.

Format

```
int synopsys_uh_hw_state ( 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.

synopsys_uh_hw_get_ms

This function is provided by the PSP to get the device timer value in ms.

> **Note:** This is only used if full speed OTG is used and SYNOPSYS_UH_CHECK_FS_RESET is enabled.

Format

```
uint16_t synopsys_uh_hw_get_ms ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
Time	The timer value.