TEXAS INSTRUMENTS

TEXAS INSTRUMENTS THE WORLD LEADER IN DSP AND ANALOG

User Guide

BIOSUSB User Guide

01.10.03

This page has been intentionally left blank.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments

Post Office Box 655303

Dallas, Texas 75265

This page has been intentionally left blank.

TABLE OF CONTENTS

1 Introduction

The BIOS USB Package (based on JUNGO USB stack) provides capability to incorporate a USB host or USB device functionality for an embedded device. This document describes the USB host and device stack architecture, installation procedure and example use cases in TI BIOS environment.

The BIOSUSB package supports both CCS3 and CCS4 for the following platforms

- C6747/OMAPL137
- C6748/OMAPL138

1.1 Terms & Abbreviations

Term	Description
API	Application Programming Interface
Port	Platform and Operating system abstraction layer
JOS	USB general services module
USB Device	Platform capable of performing USB device functionality
USB Host	Platform capable of performing USB host functionality
MSC	Mass Storage Class
USB	Universal Serial Bus
USBware	Product name of Jungo's USB host and device stack

1.2 References

#	Document	Description
1	usb_device_stack.pdf	USBware device stack details
2	usb_host_stack.pdf	USBware host stack details
3	mass_storage_class_driver.pdf	USBware mass storage driver details
4	mass_storage_fd.pdf	USBware mass storage function driver details
5	hid_class_driver.pdf	HID Class driver details
6	hid_keyboard_fd.pdf	HID keyboard function driver details
7	hid_fd_lib.pdf	HID function driver library reference

2 Installation Guide

This chapter discusses the BIOS USB package installation procedure and how to configure and run the usb host or device sample example provided along with the package, refer to section-6.

2.1 System Requirements

The following products are required to be installed prior to using the BIOSUSB Package for C6747/OMAPL37 and C6748/OMAPL138 platform:

For CCS3 Environment

- CCS 3.3.80.11 (service release 10)
- DSP-BIOS versioned **5.41.03.17**
- Code Generation Tools **6.1.9**
- EVM C6747/OMAPL37 (Rev-D recommended) or C6748/OMAPL138 Board (Beta- version recommended)
- XDS 510 USB Emulator (Optional) – EVM has on board emulator

For CCS4 Environment

- CCS4 version 4.0.0.16000
- DSP-BIOS versioned **5.41.03.17**
- XDS 510 USB Emulator (Optional) – EVM has on board emulator

Dependent Packages

- EDMA 3 LLD – versioned **01.11.00.02**
- ERTFS File System package versioned **1.10.01.31**
- PSP_DRIVERS Package **-** The BIOSPSP GA Package with versioned **1.30.00**

2.2 Installation and configuration of dependent packages.

BIOS USB package is dependent on the following packages and as such expects these packages to be installed and configured correctly prior to its installation and usage.

I. **EDMA3 LLD** Driver package

II. **RTFS package** – Required for BIOS USB Host functionality. RTFS package is needed for supporting MSC devices such as USB sticks, USB HDD's etc.

III. PSP_DRIVERS Package

1. Install EDMA-3 LLD Driver into preferred drive/folder. Please refer the EDMA3 package installation guide. The environment variable ' EDMA3LLD_BIOS5_INSTALLDIR' is used in the sample application projects for referring to the EDMA3 driver libraries. This environment variable is created and updated by the EDMA3 LLD driver during its installation. Please ensure that this environment variable is pointing to the EDMA3 LLD install directory intended to be used along with this package. This is more important when there are multiple EDMA LLD installations as the installer updates this environment variable with latest installation version. (eg. If EDMA LLD driver is installed into c:\edma3_lld\ the environment variable is set to EDMA3LLD_BIOS5_INSTALLDIR=c:\edma3_lld).

2. Install the RTFS Driver into preferred drive/folder. Please refer to the RTFS package installation guide. The environment variable 'RTFS_INSTALL_DIR' is used in the sample application project for referring to RTFS driver libraries. Please ensure that this environment variable is pointing to the RTFS install directory intended to be used along with this package.
3. Install BIOSPSP Package Driver into preferred drive/folder. Please refer to PSP driver's installation procedure. The environment variable 'PSP_DRIVERS_INSTALL_DIR' is used in the sample application project for referring to libraries of PSP components like block media, PSC, platforms, MMC/SD, NAND, I2C etc. Please ensure that this environment variable is pointing to the install directory of PSP component intended to be used along with this package. (eg. If BIOSPSP package is installed into c:\pspdrivers_1_30_00_06\ the environment variable is set to PSP_DRIVERS_INSTALL_DIR = c:\pspdrivers_1_30_00_06).
4. Install the BIOS USB package using the self extracting installer. Figure-1 shows the directory structure of BIOS USB Package after the installation. The installer will create BIOSUSB_INSTALL_DIR environment variable pointing to BIOSUSB installation directory. This environment variable is used in USB Host MSC application examples of RTFS package.

3 Folder Structure

Figure 1 illustrates the folder structure and placement of the BIOS USB package.

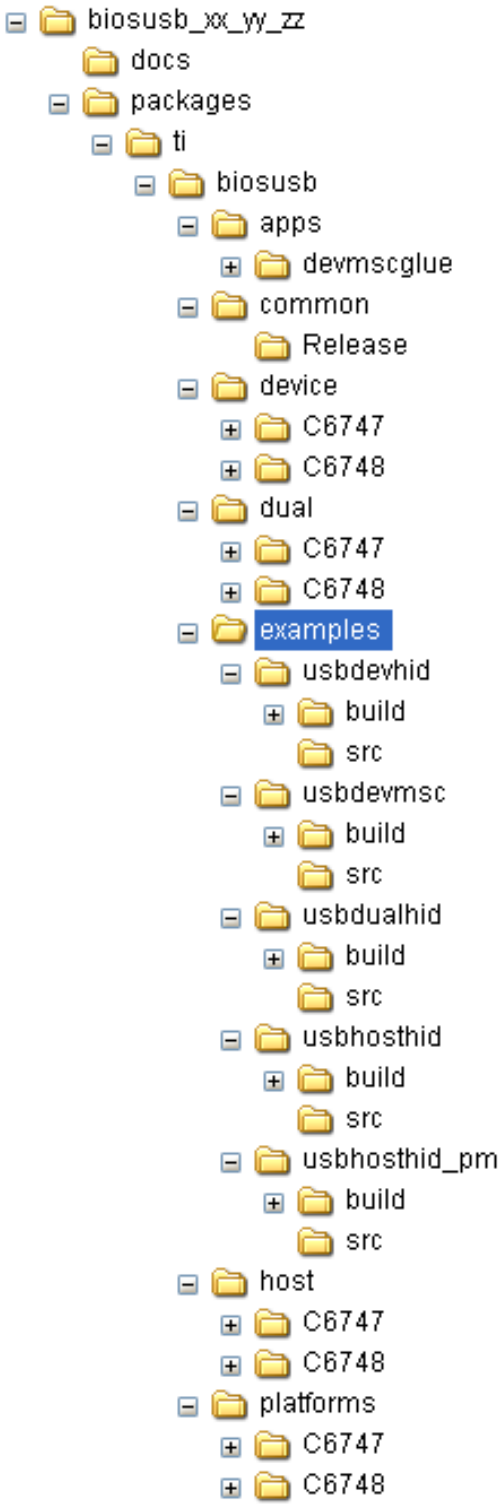

Figure 1: BIOSUSB directory structure

The folder 'biosusb' contains the following components

- USB stack for Device and/or Host mode as applicable
- Example application for demonstrating USB functionalities
- USB applications

Folder structure - detailed.

A. **common** : This folder contains

1. The component library in **release** mode which is common for biosusb device/host stack. For eg. this folder contains the power management library for c6748 platform.

B. **device**: This folder contains

1. Public header files relating to the device stack
2. Device stack libraries in **release** mode in for C6747/C6748 platforms. All the libraries will have a platform specific extension.

C. **host**: This folder contains

1. Public header files relating to the host stack
2. Host stack libraries in **release** mode for C6747/C6748 platforms. All the libraries will have a platform specific extension.

D. **dual**: This folder contains

1. Public header files relating to the dual stack
2. Dual mode stack libraries in **release** mode for C6747/C6748 platforms. All the libraries will have a platform specific extension.

E. **apps:** This folder contains the USB application for device/host.

1. **devmscglue** This folder contains the USB device MSC glue application source, documentation and binary (Debug and Release modes). MSC glue application supports storage device exposition over USB (Platform acting as MSC device to a USB host).

F. **example:** This folder contains the example for USB device/host or dual mode for C6747/C6748 platforms.

G. **Header Files:** The exported header files are distributed in following directory.

1. <BIOSUSB_INSTALL_DIR>\packages\ti\biosusb
2. <BIOSUSB_INSTALL_DIR>\packages\ti\biosusb\host
3. <BIOSUSB_INSTALL_DIR>\packages\ti\biosusb\device
4. <BIOSUSB_INSTALL_DIR>\packages\ti\biosusb\dual

For example the USB host Driver Interface prototype definitions (USBDI API) are available in biousb\host\usbdi.h. Please refer documents in section 1.2 for more information.

4 USB Host Stack Architecture

The USB host stack provides capability to support hot-pluggable interfaces for connecting various USB devices. Figure 2 illustrates the architecture of the USB host stack. Sections 3.1 to 3.9 describe the functionality of each of the components in the USB host architecture. Host stack component libraries are located in biosusb\host\Release folder.

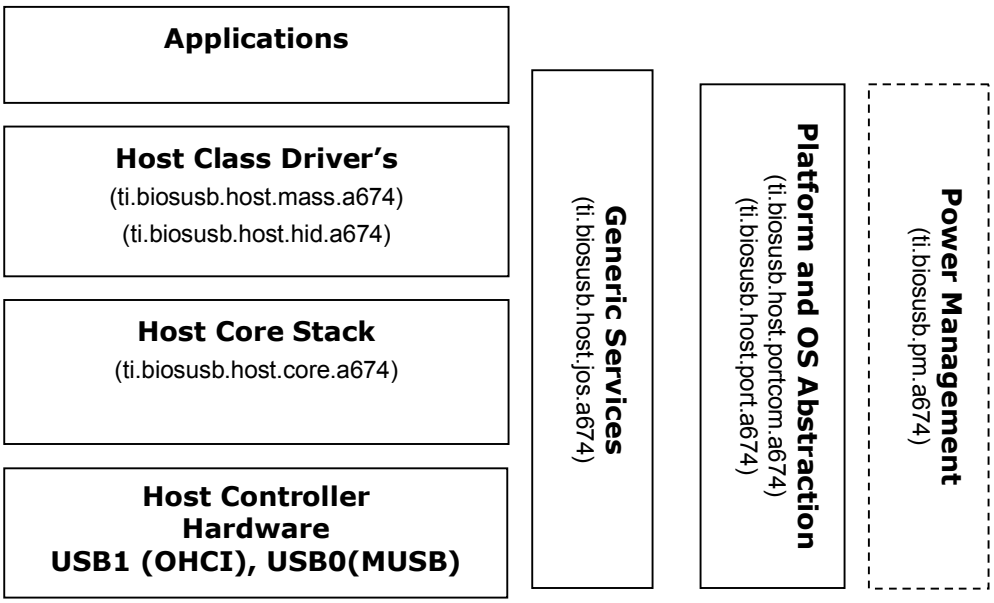

Figure 2: USB Host Stack Architecture

4.1 Host Core Stack

The host core stack component (ti.biosusb.host.core.a674) provides an implementation of the USB host functionality. It provides communication facility to its clients (class drivers) such as the mass storage class driver. This component also implements the hub driver and the host controller driver functionality for both USB1.1 Host Controller (OHCI) and USB2.0 controller (MUSB).

4.2 Host Class (MSC) Driver

The host mass storage class driver component (ti.biosusb.host.mass.a674) provides the capability to communicate with external USB mass storage devices such as thumb drives. It supports the SCSI command protocol for communication with the external USB mass storage devices.

4.3 Host Class (HID) Driver

The HID class driver component (ti.biosusb.host.hid.a674) provides the support for HID input devices like keyboard, mouse etc

4.4 Host Generic Services

The USB host generic services component (ti.biosusb.host.jos.a674) provides the services to the other components in the USB host stack architecture. It provides services for memory allocation, task management and initialization.

4.5 Platform Abstraction and Native OS Abstraction

The platform abstraction and native OS abstraction provides abstraction to the platform features and operating system interfaces. This component provides isolation to the other components of the host stack architecture from platform, native operating system and allows portability of the host stack components.

4.6 Power Management

Refer to section 7.

4.7 Host Stack Component Libraries and Dependencies

Table 1 lists the host stack components and the corresponding libraries included in the BIOSUSB deliverable. Note: The host stack component libraries are located at biosusb\host.

Host Stack Component	Library Name	Depends On
Host Core Stack	ti.biosusb.host.core.a674	ti.biosusb.host.jos.a674
Host MSC Class Driver	ti.biosusb.host.mass.a674	ti.biosusb.host.core.a674 ti.biosusb.host.jos.a674
Host HID Class Driver	ti.biosusb.host.hid.a674	ti.biosusb.host.core.a674 ti.biosusb.host.jos.a674
Host Generic Services	ti.biosusb.host.jos.a674	ti.biosusb.host.portcom.a674 ti.biosusb.host.port.a674
Platform and Native OS Abstraction	ti.biosusb.host.portcom.a674 ti.biosusb.host.port.a674	None

Table 1: USB Host Stack Component Libraries and Dependency

4.8 Host Stack Component Build Options

Multiple build options can be specified for the compilation of the USB host stack components. Table 2 lists the build options that have been used to build the (binary) host stack components that are available in the BIOS USB package.

#	Configuration Option	Configuration Value
1	CONFIG_PORT	dsp_bios
2	CONFIG_MEMPOOL	1
3	CONFIG_MEMPOOL_SIZE	500000
4	CONFIG_MEMPOOL_DMABLE	1
5	CONFIG_MUSBHSHC	1
6	CONFIG_JHOST	1
7	CONFIG_BYTE_ORDER	CPU_LITTLE_ENDIAN
8	CONFIG_ALIGN_CRITICAL	1
9	CONFIG_EXTERNAL_VBUS_CONTROL	1
10	CONFIG_LONG_64B	1
11	CONFIG_EXTERNAL_DMA	1

Table 2: USB Host Stack Component Build Options

4.9 Platform specific library

The usb platform specific library support of platform specific initialization for usb module for C6747/C6748 platforms.

4.10 Use Case: USB Host Mass Storage Functionality

The following libraries are required to build a USB host mass storage functionality. These libraries are available in Release mode in the directory biosusb/host/Release.

A. ti.biosusb.host.core.a674
B. ti.biosusb.host.mass.a674
C. ti.biosusb.host.jos.a674
D. ti.biosusb.host.portcom.a674
E. ti.biosusb.host.port.a674

4.11 Use Case: USB Host HID Functionality

The following libraries are required to build a USB host HID functionality. These libraries are available for both in Debug and Release mode in the directory biosusb/host/Release.

F. ti.biosusb.host.core.a674
G. ti.biosusb.host.jos.a674
H. ti.biosusb.host.portcom.a674
I. ti.biosusb.host.port.a674
J. ti.biosusb.host.hid.a674

5 USB Device Stack Architecture

The USB device stack enables the platform to act as a USB device to a USB host. Figure 3 illustrates the architecture of the USB device stack. Sections 5.1 to 5.7 describes the functionality of each of the components in the USB device architecture.

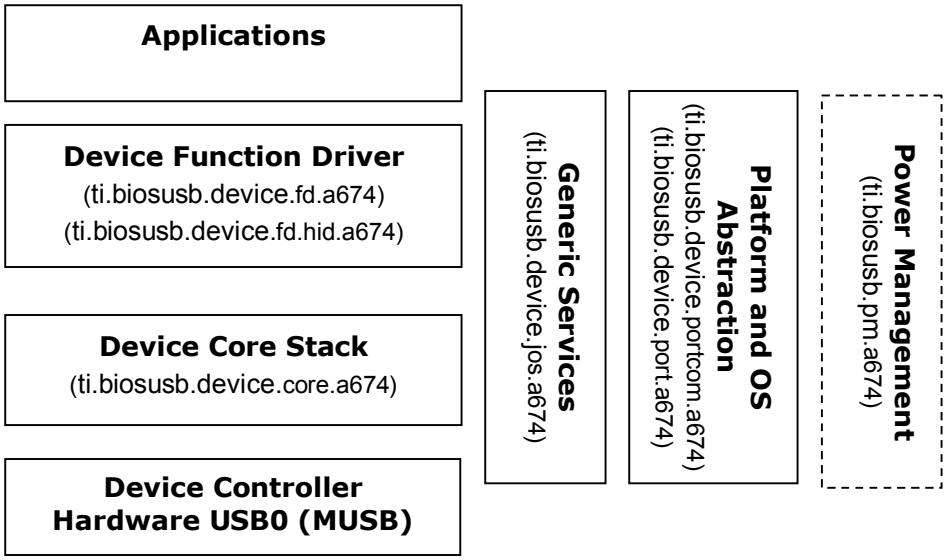

Figure 3: USB Device Stack Architecture

5.1 Device Core Stack

The device core stack (ti.biosusb.device.core.a674) component provides an implementation of the USB device functionality and device controller driver for USB2.0 MUSB controller. It provides communication facility to its clients (Class function drivers) such as the mass storage function driver.

5.2 Device (MSC) Function Driver

The device mass storage function driver component provides the capability to expose mass storage device functionality to the USB host. It supports the SCSI command protocol for communication with the USB host.

5.3 Device (HID) Function Driver

The HID function driver component provides the capability to expose HID device interface functionality to the USB host.

5.4 Device Generic Services

The USB device generic services component provides the services to the other components in the USB device stack architecture. It provides services for memory allocation, task management and initialization.

5.5 Power Management

Refer to section 7.0.

5.6 Platform Abstraction and Native OS Abstraction

The platform abstraction and native OS abstraction provides abstraction to the platform features and operating system interfaces. This component provides isolation to the other components of the device stack architecture from platform and native operating system and allows portability of the device stack components.

5.7 Device Stack Component Libraries and Dependencies

Table 3 lists the device stack components and the corresponding libraries included in the PSP deliverable.

Device Stack Component	Library Name	Depends On
Device Core Stack	ti.biosusb.device.core.a674	ti.biosusb.device.jos.a674 ti.biosusb.device.port.a674
Device MSC/HID Function Driver	ti.biosusb.device.fd.a674/ ti.biosusb.device.fd.hid.a674	ti.biosusb.device.core.a674 ti.biosusb.device.jos.a674
Device Generic Services	ti.biosusb.device.jos.a674	ti.biosusb.device.port.a674 ti.biosusb.device.portcom.a674
Platform and Native OS Abstraction	ti.biosusb.device.port.a674 ti.biosusb.device.portcom.a674	None

Table 3: USB Device Stack Component Libraries and Dependency

5.8 Device Stack Component Build Options

Multiple build options can be specified for compilation of the USB device stack components. Table 4 lists the build options that have been used to build the device stack components that are available in the BIOS USB product.

#	Configuration Option	Configuration Value
1	CONFIG_PORT	Dsp_bios
2	CONFIG_MEMPOOL	1
3	CONFIG_MEMPOOL_SIZE	500000
4	CONFIG_MEMPOOL_DMABLE	1
5	CONFIG_MUSBHSFC	1

6	CONFIG_ JSLAVE	1
7	CONFIG_BYTE_ORDER	CPU_LITTLE_ENDIAN
8	CONFIG_ALIGN_CRITICAL	1
9	CONFIG_EXTERNAL_VBUS_CONTROL	1
10	CONFIG_LONG_64B	1
11	CONFIG_EXTERNAL_DMA	1

Table 4: USB Device Stack Component Build Options

5.9 Use Case: USB Device Mass Storage Functionality

The following libraries are required to build USB device mass storage functionality. These libraries are available in Release mode in the directory biosusb/device/Release.

A. ti.biosusb.device.core.a674
B. ti.biosusb.device.fd.a674
C. ti.biosusb.device.jos.a674
D. ti.biosusb.device.port.a674
E. ti.biosusb.device.portcom.a674

5.10 Use Case: USB Device HID Functionality

The following libraries are required to build USB device HID function driver . These libraries are available in Release mode in the directory biosusb/device/Release.

A. ti.biosusb.device.core.a674
B. ti.biosusb.device.jos.a674
C. ti.biosusb.device.port.a674
D. ti.biosusb.device.portcom.a674
E. ti.biosusb.device.fd.hid.a674

6 USB Dual Mode (Host + Device) Stack Architecture

The USB dual mode stack enables the platform to act as a USB device or a USB host, and switch between the modes without reloading the stack image on the platform. Figure 4 illustrates the architecture of the USB dual mode stack. Sections 6.1 to 6.4 describes the functionality of each of the components in the USB dual mode stack architecture.

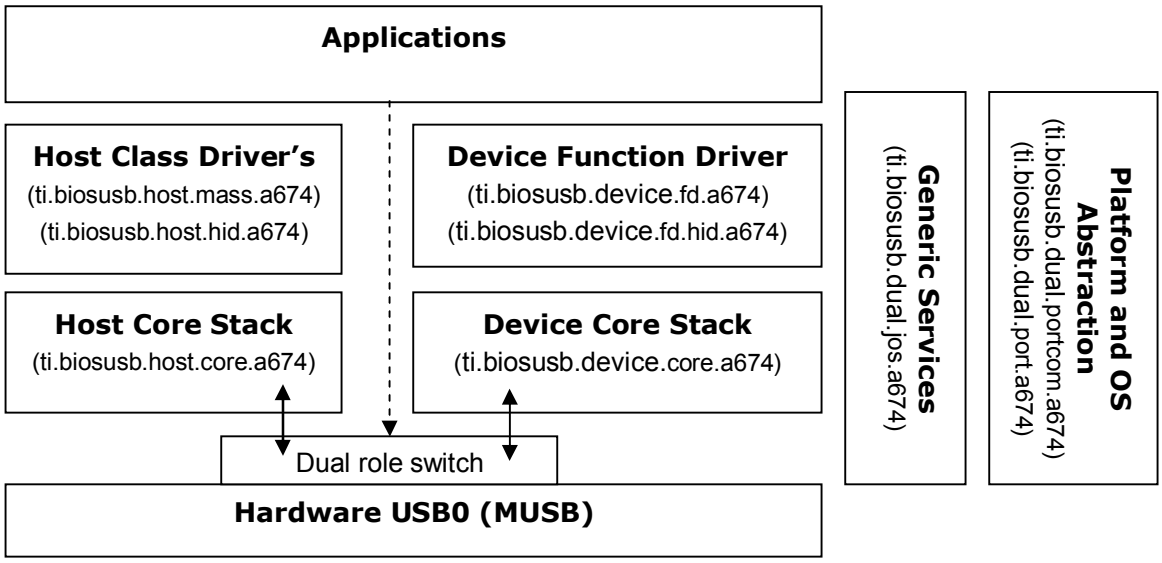

Figure 4: USB Dual Mode Stack Architecture

6.1 Dual Mode Stack

The dual stack has both the host and device core stacks. It also has the necessary functional drivers required by both the host and device mode to function. In this package we have put together host and device core stacks along with the HID class Drivers for Host and device side. Now the application can switch the mode of operation of the platform from Host to Device or vice-versa using an API call. The usage of the dual mode stack and the API call to switch between the Host and the Device mode are implemented in the Dual mode HID example provided along with the package.

6.2 Dual Mode Stack Component Libraries

Table 5 lists the dual mode stack components and the corresponding libraries included in the PSP deliverable.

Host Stack Component	Library Name
Host Core Stack	ti.biosusb.host.core.a674
Host HID Class Driver	ti.biosusb.host.hid.a674
Host MSC Class Driver	ti.biosusb.host.mass.a674
Device Stack Component	**Library Name**
Device Core Stack	ti.biosusb.device.core.a674
Device MSC/HID Function Driver	ti.biosusb.device.fd.a674/ ti.biosusb.device.fd.hid.a674
Dual Mode Stack Component	**Library Name**
Device Generic Services	ti.biosusb.device.jos.a674
Platform and Native OS Abstraction	ti.biosusb.device.port.a674 ti.biosusb.device.portcom.a674

Table 5: USB Dual Mode Stack Component Libraries

6.3 Dual Mode Stack Component Build Options

Multiple build options can be specified for compilation of the USB dual stack components. Table 6 lists the build options that have been used to build the dual stack components that are available in the BIOS USB product.

#	Configuration Option	Configuration Value
1	CONFIG_PORT	Dsp_bios
2	CONFIG_MEMPOOL	1
3	CONFIG_MEMPOOL_SIZE	500000
4	CONFIG_MEMPOOL_DMABLE	1
5	CONFIG_MUSBHSHC	1
6	CONFIG_JHOST	1
7	CONFIG_MUSBHSFC	1
8	CONFIG_ JSLAVE	1
9	CONFIG_BYTE_ORDER	CPU_LITTLE_ENDIAN

10	CONFIG_ALIGN_CRITICAL	1
11	CONFIG_EXTERNAL_VBUS_CONTROL	1
12	CONFIG_LONG_64B	1
13	CONFIG_EXTERNAL_DMA	1

Table 6: USB Dual Mode Stack Component Build Options

6.4 Use Case: USB Dual Mode Stack Functionality

The following libraries are required to build USB dual mode stack functionality. These libraries are available in Release mode in the directory biosusb/device/Release or biosusb/host/Release or biosusb/dual/Release

A. ti.biosusb.dual.jos.a674
B. ti.biosusb.dual.port.a674
C. ti.biosusb.dual.portcom.a674
D. ti.biosusb.host.core.a674
E. ti.biosusb.host.hid.a674
F. ti.biosusb.host.mass.a674
G. ti.biosusb.device.core.a674
H. ti.biosusb.device.fd.hid.a674
I. ti.biosusb.device.fd.a674

7 Power Management

The BIOSUSB support Supend/Resume of USB Bus through the following API's.

- Int32_t PSP_usb_suspend(int32_t clk_disable)
- Int32_t PSP_usb_resume(void)

The power management library (ti.biosusb.pm.a674) is available in biosusb/ common/ <platform>/ directory, this library should be linked with BIOSUSB stack along with PWRM libraries to enable power management capabilities in the USB stack.

Please refer to steps in example **usbhosthid_pm**, (for usage of PM API's) for incorporating power management support in host/device mode.

Note: The BIOSUSB stack does not support PWRM events.

8 API Reference

Following are the two new APIs have been added to support dual role mode in this release

8.1 switch_host_device_mode(bool_t host_mode)

This API is used to switch between the host mode and the device mode, when an application is running over the dual mode stack. To switch to host mode pass parameter *host_mode* as 1 and to switch to device mode pass parameter *host_mode* as 0. The API re-initializing the usb controller to switch to either device or host mode based on the parameter.

This API returns '0' when the switch is done successfully.

The usage of this API is illustrated in the 'usbdualhid' example present in the package. Please refer to example.

8.2 PSP_musbEpTrafficConfig(ep_traffic_type_t *ep_cfg);

The API is used to configure the DMA mode based on the application endpoint traffic characteristic. The API need to be called once in the startup code. The proper usage of this API is illustrated in all the examples present in this package. Please refer to usbdevmsc example in this package.

The definition of ep_traffic_type_t is as follows

```
typedef struct {
    uint8_t         is_tx;  /* is_tx =1 for tx channel, 0- for rx channel*/
    uint8_t         epnum;  /* endpoint number 1..4 */
    /** whether usb i/o request size on rx endpoint is of same size for every request
            1 - request size is same for every request
            0 - request size vary
    */
    uint8_t         is_req_size_same;
    /** whether usb i/o request is short packet terminated
            1 - shorpkt termination is valid
            0 - shorpkt termination is invalid
    */
    uint8_t         is_req_shortpkt;
}ep_traffic_type_t;
```

For example, in usb storage device application (refer usbdevmsc example) where the short packet termination is not allowed, hence the is_req_shortpkt field must be set to zero and is_req_size_same should be set to zero.

For RNDIS/CDC application where short packet termination is valid, hence the is_req_shortpkt field must be set to to one and is_req_size_same must be set to one.

8.3 Int32_t ConfigureUSB(void)

This API will perform the platform specific initialization for USB stack. User has to include the usb_evmInit.h file available in the installation directory <BIOSUSB_INSTALL_DIR>/packages/ti/platforms<C6747/C6748>/.

This API need to be called as part of standard USB initialization sequence (host and device).

8.4 Int32 dsp_bios_entry(void *arg)

This is entry point for usb stack. This API performs the initialization of generic components of USB stack. Refer to application example for usage of this API.

8.5 Int32_t PSP_usb_suspend(int32_t clk_disable)

This API is used to suspend the USB bus of OHCI and/ MUSB host controller. If input parameter clk_disable is set to true, then usb clock (controller IP clock) will be disabled after usb bus suspend operation is completed. If parameter clk_disable is set to false, the usb clock (controller IP clock) will not be disabled and only the USB bus alone is suspended. The function prototype is defined in <BIOSUSB_INSTALL>/packages/ti/biosusb/psp_usb.h

Please refer to section 9.6 usbhosthid_pm example to usage of this API.

8.6 Int32_t PSP_usb_resume (void)

This API is used to resume the USB bus (OHCI and/ MUSB) from suspended state. The function prototype is defined in <BIOSUSB_INSTALL>/packages/ ti/biosusb/psp_usb.h

Please refer to section 9.6 usbhosthid_pm example to usage of this API.

9 Building and Running USB Application Example

This chapter discusses how to configure and run the USB host or device example provided along with the package.

Note : Before running any USB example, user should perform system reset or reset the target board or EVM . The system reset can be performed through code composer studio IDE. (From CCS IDE, select menu item Debug->"Advanced Resets"->"System Reset"). Note that CPU Reset from CCS does only the resetting the CPU, it does not reset the entire system.

Setup Procedure for CCS4 Environment (Building and Loading the executable)

1. Invoke the CCS4 and perform Target configuration to configure C6747/OMAPL137 or C6748/OMAPL138 EVM platform and use the appropriate DSP gel file and Launch the target by selecting Debug-> "TI Launch Debugger", then connect to the target.
2. Select C/C++ perspective window (click on window->"open perspective"->C/C++) and Open the USB example's appropriate CCS4 project file. Select Project->"Import existing CCS/CCSE eclipse project" and provide project directory path from "<INSTALL_DIR> \biosusb_xx_yy_zz\ packages\ ti\ biosusb\ examples\ <usbdevmsc, usbdevhid, usbhosthid >\build\ <C6747/ C6748>\ ccs4". This will load the selected project. Set Project configuration to Debug/Release mode.
3. Build the project in Release/Debug mode. Switch to Debug window by selecting Debug Perspective window (click on window->"open perspective"->Debug) and load the executable.
4. Run the application program by pressing F8 key.

9.1 Building the USB MSC Device example for NAND OR MMC/SD OR SATA.

Below steps details how to build the USB device mass-storage application, where the target platform acts as USB mass-storage device with NAND or MMC/SD or SATA (only on C6748/OMAPL138 platform) as local storage media. User can choose the storage media to be either NAND or MMCSD or SATA with this application. Make sure all the dependent packages/components are installed and the environment variables are created and the values are initialized correctly.

For CCS3 Environment

1. Setup the CCSV3 to configure C6747/OMAPL137 or C6748/OMAPL138 EVM platform and use the appropriate DSP gel file and connect to the target.
2. Open the project (PJT file) for the USB device sample application from "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbdevmsc\build\<C6747/C6748>\ccs3\usb_dev_msc_sample.pjt". Set Project configuration to Release mode.
3. User can chose either NAND or MMC/SD or SATA (only C6748 paltform) as storage media. Refer to block media user guide from BIOSPSP package for further details with respect to media selection to expose over USB.

4. Build the project in Release mode, the executable file will be created in the "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples \usbdevmsc\build\<C6747/C6748>\ccs3\bin\<Debug or Release> folder.
5. Load the 'devmsc_sample.out' file from the Release folder to target and Run.

For CCS4 Environment

Please follow the procedure explained in section 7.

6. Connect the micro-B USB cable between the target (USB0 port) and Host PC. The PC enumerates the mass storage device and new removable drive will be appear in the explorer.
7. Format the drive, now the device is ready for file read/write operation.

9.2 Building the USB HID (keyboard) Device example.

Below steps details how to build the USB device HID application, where the target platform acts HID Keyboard device. Make sure all the dependent packages/components are installed and the environment variables are created and the values are initialized correctly.

For CCS3 Environment

1. Setup the CCSV3 to configure C6747/OMAPL137 or C6748/OMAPL138 EVM platform and use the appropriate DSP gel file and connect to the target.

2. Open the project (PJT file) for the USB HID device sample application from "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbdevhid\build\<C6747/C6748>\ccs3\usb_dev_hid_sample.pjt". Set Project configuration to Release mode.

3. Build the project in Release mode, the executable file will be created in the "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbdevhid\build\<C6747/C6748>\bin\Release folder.

4. Load the 'devhid_sample.out' file from the Release folder to target and Run.

For CCS4 Environment

Please follow the procedure explained in section 7.

5. Connect the micro-B USB cable between the target (USB0 port) and host-pc. The PC enumerates the HID device.

Note : In this example, in order to simulate the key press events, open the Notepad application in host-pc. Press "Shift" key and "Scroll lock" key together from the host-pc keyboard, you will see the "Jungo Demo!" string displayed on the notepad.

9.3 Building the USB Host Mass Storage example.

This example will details how to build the USB host mass-storage application. Please refer section 3.8.4 in RTFS User's Guide. The section 3.8.4 of RTFS User's guide will explain how to configure and build this example and its usage.

Note: This example dependent on the block media with RTFS file system support. The block media file system project from the PSP_DRIVERS packages needs to be built. Refer the block media documentation for further information.

9.4 Building the USB Host HID (keyboard/mouse) example.

Below steps details how to build the USB host HID example application.

For CCS3 Environment

1. Setup the CCSV3 to configure C6747/OMAPL137 or C6748/OMAPL138 EVM platform and use the appropriate DSP gel file and connect to the target.
2. Open the project (PJT file) for the USB host sample HID application from "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbhosthid\build\<C6747/C6748>\ccs3\usb_host_hid_sample.pjt". Set Project configuration to Release mode.
3. Build the project in Release mode, the executable file will be created in the"<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbhosthid\build\<C6747/C6748>\bin\Release folder.
4. Connect the USB mouse/keyboard to USB port (USB0 or USB1) directly or through HUB.
5. Load the 'hosthid_sample.out' file from the Release folder to target and Run.

For CCS4 Environment

Please follow the procedure explained in section 7.

6. The target will enumerate the USB mouse/keyboard device, when you move mouse or press any key on keyboard, keyboard/mouse key press event message will be displayed on the console window.

9.5 Building the USB Dual role HID example.

Below steps details how to build the USB dual HID example application.

For CCS3 Environment

1. Setup the CCSV3 to configure C6747/OMAPL137 or C6748/OMAPL138 EVM platform and use the appropriate DSP gel file and connect to the target.
2. Open the project (PJT file) for the USB dual sample HID application from "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbdualhid\build\<C6747/C6748>\ccs3\usb_dual_hid_sample.pjt". Set Project configuration to Release mode.
3. Build the project in Release mode, the executable file will be created in the"<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbdualhid\build\<C6747/C6748>\bin\Release folder.
4. Load the 'dualhid_sample.out' file from the Release folder to target and Run.

For CCS4 Environment

Please follow the procedure explained in section 7.

5. The example keeps switching between the host and device mode. The mode in which, the platform presently is and the mode to which it is switching to are printed in the output console window.

Switching to Host Mode:

6. Connect the USB mouse/keyboard to USB port (USB0 port) directly or through HUB if the example is in host mode at the moment.

7. The target will enumerate the USB mouse/keyboard device, when you move mouse or press any key on keyboard, keyboard/mouse key press event message will be displayed on the console window.

Switching to Device Mode:

8. Connect the micro-B USB cable between the target (USB0 port) and host-pc if the example is in device mode at the moment.
9. The PC enumerates the HID device.

Note : In this example, in order to simulate the key press events, open the Notepad application in host-pc. Press "Shift" key and "Scroll lock" key together from the host-pc keyboard, you will see the "Jungo Demo!" string displayed on the notepad.

9.6 Building the USB Power management host HID example.

This application demonstrates capability of suspending of USB bus and putting the system into deep sleep state (using PWRM_sleep call) and resume into operational state after 30 seconds. The RTC is configured to wakeup from sleep state after 30sec.

Below steps details how to build the USB host HID PM example application.

For CCS3 Environment

1. Setup the CCSV3 to configure C6748/OMAPL138 EVM platform and use the appropriate DSP gel file and connect to the target.
2. Open the project (PJT file) for the power management host hid sample HID application example from "<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\ usbdualhid_pm\build\C6748\ccs3\usb_dual_hid_sample_pm.pjt". Set Project configuration to Release mode.
3. Build the project in Release mode, the executable file will be created in the"<INSTALL_DIR>\biosusb_xx_yy_zz\packages\ti\biosusb\examples\usbdualhid_pm\build\C6748\bin\Release folder.
4. Load the 'hosthid_sample_pm.out' file from the Release folder to target and open Message log window (select DSP/BIOS->Message log) and Run the application.

For CCS4 Environment

Please follow the procedure explained in section 7.

5. Before running the application it is recommended that usb mouse be connected through hub or directly to USB0 port or USB1 port of EVM to observe the suspend/resume operation.
6. The power management example does the following of sequence steps, refer to bios_sample_pm.c for more details.
    i. The usbhosthid application task initializes the usb stack and enumerates the hub/hid devices connected and invokes pm_test() and waits for 10seconds.
    ii. During this time user can verify the hid devices (movement or button click the mouse), observe the mouse LED is ON.

iii. After the 10seconds delay the usb bus will suspended and clock is disabled (if requested). Observe the mouse LED is OFF. Wait for approximately 30 seconds.

iv. The CCS3 will be disconnected,

v. After 30seconds has elapsed the system will wake up from deep sleep state through wakeup RTC alarm and resume the USB bus. One can observe mouse LED turning ON.

vi. Connect the CCS3, the output will be displayed on message log window.

10 Technical Support BIOSUSB

To submit questions about issues with this BIOSUSB drivers release please go to the external forums at http://community.ti.com/ or to http://support.ti.com.

We currently support only the MSC and HID application in host/device mode, that are packaged along with this stack.

Revision History

Date	Author	Revision History	Version
May 2009	Ravi Babu	Support of C6747 platform on CCS3	01_00_00_04
Nov 2009	Ravi Babu	Support for C6747/C6748 platform on both CCS3/CCS4	01_10_01_01
Feb 2010	Sundaram Raju	Support dual mode feature with additional fixes	01_10_02_01
Apr 2010	Sundaram Raju	Support for Power Management	01_10_03_01