ANALOG DEVICES

ADI_USB_BULKADI DEVICE DRIVER

DATE: AUGUST 10, 2007

Table of Contents

Table of Figures

Document Revision History

Date	Description of Changes
2007-08-10	Initial version

Table 1 - Revision History

1. Overview

This document describes the use of the BULKADI USB class driver. The BULKADI class driver implements the vendor-specific BULKADI class created by ADI. BULKADI is a simple class that defines bulk endpoints to allow the movement of data in both directions over USB. The BULKADI driver works within the VisualDSP++ USB Driver Stack framework.

BULKADI does not make any assumptions regarding the controller hardware or the user application that sits on top of it. It is a fairly generic class implementation to facilitate basic USB transfers. It has been tested with both the Blackfin USB-LAN EZ-EXTENDER (with PLX NET2272 peripheral USB controller) and the ADI BF54x On-The-Go USB controller.

2. Files

The files listed below comprise the device driver API and source files.

2.1. Include Files

The driver sources include the following include files:

- <services/services.h> This file contains all definitions, function prototypes etc. for all the System Services.
- <drivers/adi_dev.h> This file contains all definitions, function prototypes etc. for the Device Manager and general device driver information.
- <drivers\usb\usb_core\adi_usb_objects.h> This file contains all definitions for USB core objects.
- <drivers\usb\usb_core\adi_usb_core.h> This file contains all definitions and function prototypes for the USB core.
- <drivers\usb\class\peripheral\vendor_specific\adi\bulkadi\adi_usb_bulkadi.h> This file contains all definitions and function prototypes for the BULKADI driver.
- <ccblkfn.h> This file contains all definitions, function prototypes etc. for the Blackfin compiler.
- <cplb.h> This file contains all definitions, function prototypes etc. for the cache protection lookaside buffers (CPLB)s.

2.2. Source Files

The driver sources are contained in the following files, as located in the default installation directory:

- adi_usb_bulkadi.c This file contains all the source code for the BULKADI driver written in C.

3. Lower Level Drivers

The BULKADI driver works within the USB driver stack model. As a class level driver, applications sit on top of it, while both the USB core and the controller driver are below it.

3.1. USB Core

The USB core handles all the USB protocol details. It manages the USB system and is responsible for enumeration, handling standard requests, and passing endpoint zero information to the appropriate upper level software. It also provides upper and lower level APIs. Please refer to the USB Core documentation for more information.

3.2. Controller Driver

The controller driver is the lowest level driver in the USB stack. It is responsible for communicating with the USB controller hardware (peripheral, OTG, or host). Please refer to the specific controller driver documentation for more information.

4. Resources Required

Device drivers typically consume some amount of system resources. This section describes the resources required by the BULKADI driver.

Unless explicitly noted in the sections below, this device driver uses the System Services to access and control any required hardware. The information in this section may be helpful in determining the resources this driver requires, such as the number of interrupt handlers or number of DMA channels etc., from the System Services.

Because dynamic memory allocations are not used in the Device Drivers or System Services, all memory used by the Device Drivers and System Services must be supplied by the application. The Device Drivers and System Services supply macros that can be used by the application to size the amount of base memory and/or the amount of incremental memory required to support the needed functionality. Memory for the Device Manager and System Services is provided in the initialization functions (adi_xxx_Init()).

Wherever possible, this device driver uses the System Services to perform the necessary low-level hardware access and control.

USB class drivers are usually hardware independent and they typically don't access hardware resources directly. Therefore the only resources required are those used by the USB core or controller drivers.

4.1. Interrupts

This driver does not use this resource directly, please check with the USB core and specific controller driver documentation for more information.

4.2. DMA

This driver does not use this resource directly, please check with the USB core and specific controller driver documentation for more information.

4.3. Timers

This driver does not use this resource directly, please check with the USB core and specific controller driver documentation for more information.

4.4. Real-Time Clock

This driver does not use this resource directly, please check with the USB core and specific controller driver documentation for more information.

4.5. Programmable Flags

This driver does not use this resource directly, please check with the USB core and specific controller driver documentation for more information.

4.6. Pins

This driver does not use this resource directly, please check with the USB core and specific controller driver documentation for more information.

5. Supported Features of the Device Driver

This section describes what features are supported by the device driver. As previously stated, most USB class drivers are hardware independent. This means they often defer many specifics to the USB core or lower level controller driver.

5.1. Directionality

The driver supports the dataflow directions listed in the table below.

ADI_DEV_DIRECTION	Description
ADI_DEV_ DIRECTION_BIDIRECTIONAL	Supports both the reception of data and transmission of data through the device.

Table 2 – Supported Dataflow Directions

5.2. Dataflow Methods

The driver supports the dataflow methods listed in the table below.

ADI_DEV_MODE	Description
ADI_DEV_MODE_CHAINED	Supports the chained buffer method

Table 3 – Supported Dataflow Methods

5.3. Buffer Types

The driver supports the buffer types listed in the table below.

- ADI_DEV_1D_BUFFER
  - Linear one-dimensional buffer
  - pAdditionalInfo – ignored
  - **Reserved[4]** field of the buffer should have the destination endpoint number. Applications can obtain the endpoint information via the ADI_USB_CMD_ENUMERATE_ENDPOINTS control command to the associated class driver.

5.4. Command IDs

This section enumerates the commands that are supported by the driver. The commands are divided into three sections. The first section describes commands that are supported directly by the Device Manager. The next section describes common commands that the driver supports. The remaining section describes driver specific commands.

Please note that the BULKADI driver may pass any command that it does not handle down to the lower level drivers. For that reason, please check with the USB core and specific controller driver documentation for more information.

Commands are sent to the device driver via the adi_dev_Control() function. The adi_dev_Control() function accepts three arguments:

- DeviceHandle – This parameter is a ADI_DEV_DEVICE_HANDLE type that uniquely identifies the device driver. This handle is provided to the client in the adi_dev_Open() function call.
- CommandID – This parameter is a u32 data type that specifies the command ID.
- Value – This parameter is a void * whose value is context sensitive to the specific command ID.

The sections below enumerate the command IDs that are supported by the driver and the meaning of the Value parameter for each command ID.

5.4.1. Device Manager Commands

The commands listed below are supported and processed directly by the Device Manager. As such, all device drivers support these commands.

- ADI_DEV_CMD_TABLE
  - Table of command pairs being passed to the driver
  - Value – ADI_DEV_CMD_VALUE_PAIR *
- ADI_DEV_CMD_END
  - Signifies the end of a command pair table
  - Value – ignored
- ADI_DEV_CMD_PAIR
  - Single command pair being passed
  - Value – ADI_DEV_CMD_PAIR *
- ADI_DEV_CMD_SET_SYNCHRONOUS
  - Enables/disables synchronous mode for the driver
  - Value – TRUE/FALSE

5.4.2. Common Commands

The command IDs described in this section are common to many device drivers. The list below enumerates all common command IDs that are supported by this device driver.

- ADI_DEV_CMD_SET_DATAFLOW_METHOD
  - Specifies the dataflow method the device is to use. The list of dataflow types supported by the device driver is specified in section 0.
  - Value – ADI_DEV_MODE enumeration
- ADI_DEV_CMD_SET_DATAFLOW
  - Enables/disables dataflow through the device
  - Value – TRUE/FALSE

5.4.3. Device Driver Specific Commands

The command IDs listed below are supported and processed by the device driver. These command IDs are unique to this device driver.

- ADI_USB_CMD_ENUMERATE_ENDPOINTS
  - This command gets the endpoint information from the driver. This information includes information the caller needs to use the endpoints.
  - Value – ADI_ENUM_ENDPOINT_INFO* (pointer to endpoint info structure)

5.5. Callback Events

The BULKADI driver does not generate any callback events itself although it may be used to pass callbacks back to the application. Please check with the USB core and specific controller driver documentation for more information.

5.6. Return Codes

All API functions of the device driver return status indicating either successful completion of the function or an indication that an error has occurred. This section enumerates the return codes that the device driver is capable of

returning to the client. A return value of ADI_DEV_RESULT_SUCCESS indicates success, while any other value indicates an error or some other informative result. The value ADI_DEV_RESULT_SUCCESS is always equal to the value zero. All other return codes are a non-zero value.

The return codes are divided into two sections. The first section describes return codes that are common to many device drivers. The next section describes driver specific return codes. The client should prepare to process each of the return codes described in these sections.

Typically, the application should check the return code for ADI_DEV_RESULT_SUCCESS, taking appropriate corrective action if ADI_DEV_RESULT_SUCCESS is not returned. For example:

```
if (adi_dev_Xxxx(…) == ADI_DEV_RESULT_SUCCESS) {
      // normal processing
} else {
      // error processing
}
```

5.6.1. Common Return Codes

The return codes described in this section are common to many device drivers. The list below enumerates all common return codes that are supported by this device driver.

<Do not delete any entries is this section. This list contains return codes that the Device Manager is capable of returning regardless as to whether or not the device driver supports them.>

- ADI_DEV_RESULT_SUCCESS
  - The function executed successfully.
- ADI_DEV_RESULT_NOT_SUPPORTED
  - The function is not supported by the driver.
- ADI_DEV_RESULT_DEVICE_IN_USE
  - The requested device is already in use.
- ADI_DEV_RESULT_NO_MEMORY
  - There is insufficient memory available.
- ADI_DEV_RESULT_BAD_DEVICE_NUMBER
  - The device number is invalid.
- ADI_DEV_RESULT_DIRECTION_NOT_SUPPORTED
  - The device cannot be opened in the direction specified.
- ADI_DEV_RESULT_BAD_DEVICE_HANDLE
  - The handle to the device driver is invalid.
- ADI_DEV_RESULT_BAD_MANAGER_HANDLE
  - The handle to the Device Manager is invalid.
- ADI_DEV_RESULT_BAD_PDD_HANDLE
  - The handle to the physical driver is invalid.
- ADI_DEV_RESULT_INVALID_SEQUENCE
  - The action requested is not within a valid sequence.
- ADI_DEV_RESULT_ATTEMPTED_READ_ON_OUTBOUND_DEVICE
  - The client attempted to provide an inbound buffer for a device opened for outbound traffic only.
- ADI_DEV_RESULT_ATTEMPTED_WRITE_ON_INBOUND_DEVICE
  - The client attempted to provide an outbound buffer for a device opened for inbound traffic only.
- ADI_DEV_RESULT_DATAFLOW_UNDEFINED
  - The dataflow method has not yet been declared.
- ADI_DEV_RESULT_DATAFLOW_INCOMPATIBLE
  - The dataflow method is incompatible with the action requested.
- ADI_DEV_RESULT_BUFFER_TYPE_INCOMPATIBLE
  - The device does not support the buffer type provided.
- ADI_DEV_RESULT_CANT_HOOK_INTERRUPT

    - The Interrupt Manager failed to hook an interrupt handler.
- ADI_DEV_RESULT_CANT_UNHOOK_INTERRUPT
    - The Interrupt Manager failed to unhook an interrupt handler.
- ADI_DEV_RESULT_NON_TERMINATED_LIST
    - The chain of buffers provided is not NULL terminated.
- ADI_DEV_RESULT_NO_CALLBACK_FUNCTION_SUPPLIED
    - No callback function was supplied when it was required.
- ADI_DEV_RESULT_REQUIRES_UNIDIRECTIONAL_DEVICE
    - Requires the device be opened for either inbound or outbound traffic only.
- ADI_DEV_RESULT_REQUIRES_BIDIRECTIONAL_DEVICE
    - Requires the device be opened for bidirectional traffic only.

5.6.2. Device Driver Specific Return Codes

The BULKADI driver does not have any driver specific return codes. Please check with the USB core and specific controller driver documentation for more information.

6. Opening and Configuring the Device Driver

This section describes the default configuration settings for the device driver and any additional configuration settings required from the client application.

6.1. Entry Point

When opening the device driver with the adi_dev_Open() function call, the client passes a parameter to the function that identifies the specific device driver that is being opened. This parameter is called the entry point. The entry point for this driver is listed below.

- ADI_USB_VSBulk_Entrypoint

6.2. Default Settings

The table below describes the default configuration settings for the device driver. If the default values are inappropriate for the given system, the application should use the command IDs listed in the table to configure the device driver appropriately. Any configuration settings not listed in the table below are undefined.

Item	Default Value	Possible Values	Command ID
None			

Table 4 - Default Settings

6.3. Additional Required Configuration Settings

In addition to the possible overrides of the default driver settings, the device driver requires the application to specify the additional configuration information listed in the table below.

Item	Possible Values	Command ID
Dataflow method	See section 0	ADI_DEV_CMD_SET_DATAFLOW_METHOD

Table 5 – Additional Required Settings

7. Hardware Considerations

USB class drivers are usually hardware independent and they typically don't access hardware resources directly. Specifically, the BULKADI driver is generic in that it may be used with different controller hardware without major modifications. One area of concern would be in regards to the number of available endpoints. The default BULKADI driver defines only two bulk endpoints and any peripheral USB controller should be capable of supporting at least two bulk endpoints.