USB MASS STORAGE CLASS DEVICE DRIVER

DATE: AUGUST 13, 2007

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
July 27, 2007	Initial Version
July 28, 2007	Updates and revisions
August 13, 2007	Added new IOCTL commands

Table 1 – Revision History

1. Overview

This document describes the usage of the peripheral mode of the USB Mass Storage Class Driver.   This driver has been tested on the BF533 EZKit Lite with a USBLAN-EZEXT and the BF548 EZKit Lite development boards.

The EZKit is connected to a PC via a USB cable. Once connected the PC detects the device, enumerates it. When enumeration is completed the device appears as a disk drive that can be accessed by the PC.

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 various device objects that are buliding blocks for the usb system.
- <drivers\usb\usb_core\adi_usb_core.h> This file contains usb core layer definitions.
- <drivers/usb/class/pheripheral/mass_storage/adi_usb_msd_class.h> This file contains all commands, event and return codes specific to the Mass Storage Class Driver.
- <drivers/usb/class/pheripheral/mass_storage/adi_usb_msd_scsi.h> This file contains the Mass Storage SCSI command definition and structures
- <drivers/usb/class/pheripheral/mass_storage/adi_usb_msd_support.h> This file contains the Mass Storage support definitions

2.2. Source Files

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

- <Blackfin/lib/src/drivers/usb/class/pheripheral/mass_storage/adi_usb_msd_class.c> This file contains source code for the Mass Storage Class Driver. All code is written in 'C'.
- <Blackfin/lib/src/drivers/usb/class/pheripheral/mass_storage/adi_usb_msd_scsi.c> This file contains source code for Mass Storage Class Driver SCSI support. All code is written in 'C'.
- <Blackfin/lib/src/drivers/usb/class/pheripheral/mass_storage/adi_usb_msd_support.c> This file contains the support code for the Mass Storage Class Driver. All code is written in 'C'.

3. Lower Level Drivers

The Mass Storage Class Driver is part of the ADI USB stack. Please see the USB Core and USB Device driver documentation or more information.

4. Resources Required

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

4.1 Interrupts

The USB Mass Storage Device Driver does not use any Interrupt resources.

4.2 DMA

The USB Mass Storage Device Driver does not use any DMA resources.

4.3 Timers

The USB Mass Storage Device Driver does not use any Timer resources.

4.4 Real-Time Clock

The USB Mass Storage Device Driver does not use any RTC resources.

4.5 Programmable Flags

The USB Mass Storage Device Driver does not use any Programmable Flag resources.

4.6 Pins

The USB Mass Storage Device Driver does not use any Pin resources.

5 Supported Features of the Device Driver

This section describes what features are supported by the device 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

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.

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 5.2.
  - 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_ENABLE_USB
  - This command enables USB communication with the host PC.
  - Value – NULL

- ADI_MSD_CMD_SET_MEMORY_SIZE
  - If Mass Storage Device is going to use onboard memory this command sets the memory size to use.
  - Value – This parameter is a void * whose value is the size of memory to use. (currently this value can only be set to 0x800000 which will allocate 8MB)

- ADI_MSD_CMD_SCSI_INIT
  - This command initializes the SCSI command handlers.
  - Value – NULL

- ADI_MSD_CMD_IS_DEVICE_CONFIGURED
  - This command is called to determine if the device was configured by the host PC.
  - Value – This parameter is a pointer to a u32 which will upon completion contain the returned value TRUE for device configured or FALSE device not configured.

- ADI_MSD_CMD_SET_VID
    - This command is used to set a user defined Vendor ID.
    - Value – This value should be the user defined Vendor ID for this device.

- ADI_MSD_CMD_SET_PID
    - This command is used to set a user defined Product ID.
    - Value – This value should be the user defined Product ID.

- ADI_MSD_CMD_SET_SERIAL_NUMBER
    - This command is used to set a user defined serial number.
    - Value – This value should contain a pointer to a string containing the user defined serial number.

- ADI_MSD_CMD_INIT_RAW_PID
    - This command is used to initialize the device that communicates with a onboard Hard Disk. This command should only be used with EZKits that have support for a ATAPI Hard Disk.
    - Value – This value should be a pointer to a u32 the will receive the result. If the command returns TRUE an ATAPI Hard Disk is available and ready to use. If it returns FALSE no ATAPI Hard Disk can be found.

- ADI_MSD_CMD_SET_BUFFER
    - This command is used to pass the buffer down to the Mass Storage Class Driver.
    - Value – This a void * whose value is the address of the buffer.

- ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE
    - This command is used to pass the controller device handle to the class driver.
    - Value – This is a void * whose value is the handle of the controller device.

- ADI_USB_CMD_CLASS_CONFIGURE
    - This command is to configure the class driver.
    - Value – This value should be zero.

5.5 Callback Events

This Mass Storage Class driver does not send callback events back to the client application. All callbacks are handled by the class driver. Section 5.5.1 defines the events handled.

5.5.1 Device Driver Specific Events

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

- ADI_USB_EVENT_SET_CONFIG
    - Notifies callback function that device enumeration by the host is complete. Device is ready to use. This is an Endpoint Zero event.
    - Value – This value is the CallbackParameter value that was supplied in the buffer that was passed.

- ADI_USB_EVENT_SETUP_PKT
    - This is a device setup packet event. Here the host can request a reset or it can ask the device to return the logical unit number LUN. This is an Endpoint Zero event.
    - Value - This value is the CallbackParameter value that was supplied in the buffer that was passed.

- ADI_USB_EVENT_ROOT_PORT_RESET
    - Notifies callback function that host is signalling a reset
    - Value - This value is the CallbackParameter value that was supplied in the buffer that was passed.

- ADI_USB_EVENT_VBUS_FALSE

- Notifies callback function that the cable is unplugged.
- Value - This value is the CallbackParameter value that was supplied in the buffer that was passed.

- ADI_USB_EVENT_RESUME
  - Notifies callback function that host is initiating a resume.
  - Value - This value is the CallbackParameter value that was supplied in the buffer that was passed.

- ADI_USB_EVENT_SUSPEND
  - Notifies callback function that host is initiating a suspend.
  - Value - This value is the CallbackParameter value that was supplied in the buffer that was passed.

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.

- 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

There are no Mass Storage Class specific return codes.

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_Device_MassStorageClass_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 5.2	ADI_DEV_CMD_SET_DATAFLOW_METHOD
Enable Dataflow	See section 5.2	ADI_DEV_CMD_SET_DATAFLOW
Detect Hard Disk	See section 5.4.3	ADI_MSD_CMD_INIT_RAW_PID
Using onboard Memory	See section 5.4.3	ADI_MSD_CMD_SET_MEMORY_SIZE
Init SCSI Handlers	See section 5.4.3	ADI_MSD_CMD_SCSI_INIT
Set user defined vendor ID	See section 5.4.3	ADI_MSD_CMD_SET_VID
Set user defined product ID	See section 5.4.3	ADI_MSD_CMD_SET_PID
Set user defined serial number	See section 5.4.3	ADI_MSD_CMD_SET_SERIAL_NUMBER
Enable USB	See section 5.4.3	ADI_USB_CMD_ENABLE_USB
Pass the handle of the controller to the class driver	See section 5.4.3	ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE
Configure the class driver	See section 5.4.3	ADI_USB_CMD_CLASS_CONFIGURE
Pass the user buffer to the class driver	See section 5.4.3	ADI_MSD_CMD_SET_BUFFER

Table 5 – Additional Required Settings

7 Hardware Considerations

If using the BF548 EZKit Lite with an ATAPI hard disk, be sure to format the disk using the supplied disk format utility.