ANALOG DEVICES

USB APPLICATION PORTING GUIDE

DATE: AUGUST 24, 2007

Table of Contents

Document Revision History

Date	Description of Changes
2007-07-23	Initial version
2007-08-17	Added changes for opening controller driver from app
2007-08-24	Added section on “Things to Watch Out For”

Table 1 - Revision History

1. Overview

This document is intended to help users port their bulk transport USB-based applications created with VisualDSP++ 4.5 to VisualDSP++ 5.0. Note that there is not audio class support in the initial release of 5.0.

With VisualDSP++ 5.0, Analog Devices Inc. (ADI) has developed a brand new USB stack from the ground up. The new USB stack breaks down the responsibilities of handling USB transfers into layers allowing for a much more dynamic, customizable, and easier to understand system. The USB stack is in contrast to the earlier USB offerings which combined the NET2272 controller, USB protocol, and class level details all in one library.

The USB stack uses the ADI System Services and Device Driver models exclusively which allows ADI or other users to create components that will easily plug into the stack. If a different USB controller is desired (for which a driver is not already available) a user may write a driver just for that USB controller and insert it into the stack. This way a user will be able to leverage the application, class, and core layers already available and face minimal integration issues in the process. Likewise new class drivers may be added into the stack without worrying about affecting the system as a whole.

Note that VDSP 4.5 only supported a single USB controller, the PLX NET2272 Peripheral Controller. With the initial release of VDSP 5.0 there is also support for the ADI BF54x Dual-Role On-The-Go USB Controller as well.

For more information on the USB stack please refer to the documentation for the individual stack components or refer to the USB examples within VDSP 5.0.

Please note that within this document <VDSP45> and <VDSP50> refer to the installation folders for VDSP 4.5 and VDSP 5.0 respectively.

2. Files

This section will give a brief overview as to where USB files can be found in VDSP 5.0 as opposed to VDSP 4.5. USB follows the hierarchical folder system used by other drivers within VDSP 5.0 whether they're include files, source files, examples, or documentation files. This simplifies the task of finding what the user is looking for.

2.1. Include Files

In 4.5 the NET2272 driver include files were located here:

- <VDSP45>\Blackfin\include\drivers\usb\net2272
- <VDSP45>\Blackfin\include\drivers\usb\net2272\plx

In 5.0 there are several different drivers available. You will find their include files located here:

- Class drivers
  - Host
    - <VDSP50>\Blackfin\include\drivers\usb\class\host\mass_storage
  - Peripheral
    - <VDSP50>\Blackfin\include\drivers\usb\class\peripheral\mass_storage
    - <VDSP50>\Blackfin\include\drivers\usb\class\peripheral\vendor_specific\adi\bulkadi
- Controller drivers
  - On-The-Go
    - <VDSP50>\Blackfin\include\drivers\usb\controller\otg\adi\bf54x
  - Peripheral
    - <VDSP50>\Blackfin\include\drivers\usb\controller\plx\net2272

- USB core
  - <VDSP50>\Blackfin\include\drivers\usb\usb_core

2.2. Source Files

In 4.5 the NET2272 driver source files were located here:

- <VDSP45>\Blackfin\lib\src\drivers\usb\net2272
- <VDSP45>\Blackfin\lib\src\drivers\usb\net2272\plx

In 5.0 there are several different drivers available. You will find their source files located here:

- Class drivers
  - Host
    - <VDSP50>\Blackfin\lib\src\drivers\usb\class\host\mass_storage
  - Peripheral
    - <VDSP50>\Blackfin\lib\src\drivers\usb\class\peripheral\mass_storage
    - <VDSP50>\Blackfin\lib\src\drivers\usb\class\peripheral\vendor_specific\adi\bulkadi
- Controller drivers
  - On-The-Go
    - <VDSP50>\Blackfin\lib\src\drivers\usb\controller\otg\adi\bf54x
  - Peripheral
    - <VDSP50>\Blackfin\lib\src\drivers\usb\controller\plx\net2272

- USB core
  - <VDSP50>\Blackfin\lib\src\drivers\usb\usb_core

2.3. Blackfin Examples

In 4.5 the NET2272 examples were located here:

- <VDSP45>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB

In 5.0 the examples are sorted by USB controller hardware, so NET2272 examples are still located in the same location (with new example names) while BF548 examples are in their own folder as follows:

- NET2272 examples
  - <VDSP50>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB
- BF548 examples
  - <VDSP50>\Blackfin\Examples\ADSP-BF548 EZ-KIT Lite\drivers\usb

2.4. Windows Host Software

In 4.5 the Windows host software (host application and Windows drivers) was located here:

- <VDSP45>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB\host

In 5.0 the host software (host application and Windows drivers) is located here:

- <VDSP50>\Blackfin\ Examples\usb\host\windows

2.5. Documentation

In 4.5 the only documentation was this readme file:

- <VDSP45>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB\USB Software Readme.txt

In 5.0 each example project has its own readme file (look in each example project folder) and each stack component has a documentation file such as:

- <VDSP50>\Blackfin\Examples – check each example project folder for the readme
- <VDSP50>\Blackfin\docs\drivers\usb – check each subfolder for the documentation files

3. Porting USB Applications from VDSP 4.5 to VDSP 5.0

There have been numerous improvements and changes to the System Services and Driver Model in 5.0 that will very likely result in 4.5 USB applications not working as expected when rebuilt with 5.0 even when using the 4.5 USB libraries. For this reason alone it is not recommended to use the 4.5 USB driver with 5.0.

Instead we recommend that you use the new 5.0 USB stack which will be supported by VDSP going forward. In order to simplify the process of porting 4.5 applications to work with 5.0 we offer the following points that may be followed in order to complete the port. Additionally, it is recommended to compare a 4.5 USB example (such as the NET2272 loopback application) with its 5.0 counterpart to give more insight into the changes necessary. It's a good idea to look at a "diff" of these example files while performing your own port.

Note that this list of changes assumes that the user used the 4.5 NET2272 driver "as is" and did not make any modifications to it. We also assume the user has moderate experience with VDSP as each step will not be explained in extensive detail. While the NET2272 was the only supported controller in 4.5, this porting guide is equally valid for users wishing to port their NET2272 application to use another controller such as the BF54x.

Also note that this list was compiled by porting both VDSP and Blackfin Software Development Kit (SDK) examples based on 4.5 to 5.0. Depending on how many changes users have made to their projects, it may not be complete for all user applications, but it should serve as a good guide. Please make a copy of your existing application before starting this port.

3.1. Application Project File Changes

The following changes are required to the application project (dpj) file:

- Remove support for the old USB library such as:
  - Remove paths to the old USB driver include directories either on the *Compile->Preprocessor* page of the *Project Options* dialog or within the LDF
  - Remove references to the old USB library (libnet2272_*.dlb) either on the *Link* page of the *Project Options* dialog or within the LDF
  - Remove the older NET2272 header file (adi_net2272.h) from your project

- Add support for the USB controller your wish to use:
  - For NET2272 support, add the following file to your project: <VDSP50>\Blackfin\lib\src\drivers\usb\controller\plx\net2272\adi_usb_net2272.c
  - For BF548 support, you must link with the libdrv548.dlb library. If you are using a standard LDF this library should be automatically included but if you have a custom LDF you may have to manually add this library.

- Add support for the USB stack:
  - You must link with one of the USB libraries which contain the class and core functionality. If you are using a standard LDF this library should be automatically included but if you have a custom LDF you may have to manually add this library. For BF533 use libusb532.dlb, for BF537 use libusb537.dlb, for BF548 use libusb548.dlb, and for BF561 use libusb561.dlb.

3.2. Application Source File Changes

The following changes are required to the application source files:

- Replace old include files with new ones:

  - Remove any include files that reference the old "*net2272*" and "*plx*" folders. You will likely want to replace them with some or all of these files (depending on what the file needs):

        - USB core headers
            - #include <drivers\usb\usb_core\adi_usb_objects.h>
            - #include <drivers\usb\usb_core\adi_usb_core.h>
            - #include <drivers\usb\usb_core\adi_usb_ids.h>
        - USB bulk class header
            - #include <drivers\usb\class\peripheral\vendor_specific\adi\bulkadi\adi_usb_bulkadi.h>
        - USB controller headers
            - #include <drivers\usb\controller\peripheral\plx\net2272\adi_usb_net2272.h>
            - #include <drivers\usb\controller\otg\adi\bf54x\adi_usb_bf54x.h>
        - USB hostapp interface header
            - #include <..\..\..\..\..\Examples\usb\host\windows\hostapp\hostapp.h>

    - The EZ-KIT utility files (.c and .h) have been renamed and moved from a common folder into each project folder that uses them shown below. You will need to update the filename and/or path anywhere that you reference these files.

        - In 4.5:
            - <VDSP45>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB\common\ezkitutilities.*

        - In 5.0, one example:
            - <VDSP50>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB\ bulk_loopback_app\usb_ezkit_utils.*

- Allocate more memory to System Services:

    - In 5.0 the SS managers need more memory than in 4.5. How much more memory depends on your application, this is what we allocated in the example applications:

        - In 4.5:

```
static u8 IntMgrData[                       (ADI_INT_SECONDARY_MEMORY * 0)];
static u8 DMAMgrData[ADI_DMA_BASE_MEMORY + (ADI_DMA_CHANNEL_MEMORY * 2)]
static u8 DevMgrData[ADI_DEV_BASE_MEMORY + (ADI_DEV_DEVICE_MEMORY * 1)];
```

        - In 5.0:

```
static u8 IntMgrData[                         (ADI_INT_SECONDARY_MEMORY * 8)];
static u8 DMAMgrData[ADI_DMA_BASE_MEMORY + (ADI_DMA_CHANNEL_MEMORY * 8)];
static u8 DevMgrData[ADI_DEV_BASE_MEMORY + (ADI_DEV_DEVICE_MEMORY * 4)];
```

- USB events (such as those handled by the application's callback handler) have had their names changed. All standard USB events which used to be named ADI_NET2272_EVENT_xxx are now named ADI_USB_EVENT_xxx. For example, ADI_NET2272_EVENT_DATA_TX is now ADI_USB_EVENT_DATA_TX.

- You must initialize the USB core before attempting to open any USB drivers. The prototype for the core initialization function is this:

```
s32_t adi_usb_CoreInit(void *pConfig);
```

- You must open both the USB class driver and the USB controller driver. You will get different handles back for each opened driver. Both drivers also contain new entry points in 5.0.

    - Only one entry point existed in 4.5:
        - NET2272 controller - ADINET2272EntryPoint

  - All entry points are new in 5.0:
    - bulk class - ADI_USB_VSBulk_Entrypoint
    - NET2272 controller - ADI_USB_NET2272_Entrypoint
    - BF54x controller - ADI_USBDRC_Entrypoint

- Two additional adi_dev_Control() calls must be made after opening the USB drivers. The first one supplies the class driver with the handle to the controller you want to use and the second one tells the class driver that it is time to configure. Note that these commands are issued to the class driver, not the controller driver. The code will look something like the following but please refer to the examples as well:

```
// give USB controller driver handle to the class drivers.
Result = adi_dev_Control(ClassDevHandle, ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE,
(void*)PeripheralDevHandle);

// configure the class driver
Result = adi_dev_Control(ClassDevHandle, ADI_USB_CMD_CLASS_CONFIGURE, (void*)0);
```

- The application is now responsible to supply the vendor and device IDs (VIDs and PIDs). These are the IDs which identify your product to the host system. They used to be contained within the NET2272 library. Now the user has access to them via the *adi_usb_GetDeviceDescriptor()* USB core API function. After opening the USB device you may call this function to get a pointer to the device descriptor. All the fields within the descriptor may be updated but most importantly you will need to update the wIdProduct and wIdVendor fields. The IDs used by ADI are supplied in the <VDSP50>\Blackfin\include\drivers\usb\usb_core\adi_usb_ids.h header file. Look at the bulk USB examples for exactly how to do this.

- To enable USB communications you must issue the ADI_USB_CMD_ENABLE_USB command via the *adi_dev_Control()* function to the controller driver. This should be done as the last step before you are expecting data transfer. Look at the bulk USB examples for exactly how to do this, here is an example:

```
adi_dev_Control(ControllerDevHandle, ADI_USB_CMD_ENABLE_USB, (void*)0 ));
```

- The USB system (class drivers and core) now assign endpoint addresses based on resources available in the system. This means there are no longer any hard coded endpoint addresses specified in the application. You will need to issue the ADI_USB_CMD_CLASS_ENUMERATE_ENDPOINTS command via the *adi_dev_Control()* function to get the assigned addresses after the system is configured. The example applications have done this with a *ConfigureEndpoints()* routine. You may copy this implementation or come up with your own. Please refer to the example bulk USB applications for how this is done. The basic steps to copy the example implementation are as follows:

  - Define global read and write endpoint IDs in your source file such as this:

```
static s32 g_ReadEpID;
static s32 g_WriteEpID;
```

  - Copy the *ConfigureEndpoints()* from one of the 5.0 example bulk USB applications (such as loopback) to your source file.
  - Call *ConfigureEndpoints()* after your system is configured.

- In 5.0 you now need to supply the endpoint address to any read or write call. In 4.5 we provided *usb_Read()* and *usb_Write()* wrapper functions which wrapped the *adi_dev_Read()* and *adi_dev_Write()* routines. They are still there in 5.0 and they make a convenient central location to place this information. Look at the bulk USB examples for how to do this, but the basic change is to add the following code to the beginning of the wrapper functions:

  - In *usb_Read()*:

```
ADI_DEV_1D_BUFFER                   *p1DBuff;
```

```
p1DBuff = (ADI_DEV_1D_BUFFER*)pBuffer;
p1DBuff->Reserved[BUFFER_RSVD_EP_ADDRESS] = g_ReadEpID;
```

- In *usb_Write()*:

```
ADI_DEV_1D_BUFFER                    *p1DBuff;
p1DBuff = (ADI_DEV_1D_BUFFER*)pBuffer;
p1DBuff->Reserved[BUFFER_RSVD_EP_ADDRESS] = g_WriteEpID;
```

- In 4.5 the common header file shared by the Windows hostapp and the Blackfin examples was called usbcmd.h. This file had been renamed, moved, and modified in 5.0. It is not necessary to port to the 5.0 version of this file for many users as basically only a few names have changed. This file lives at the application level so as long as you use the same file when you build your examples as when you build hostapp you will be fine. For those who wish to use the 5.0 hostapp and the 5.0 version of this file you will have to make some source modifications. Check the example bulk USB applications for how this is done, but the basic changes are as follows:
  - Update references to the header file:
    - In 4.5 the file was here: <VDSP45>\Blackfin\Examples\USB-LAN EZ-EXTENDER\USB\common\usbcmd.h
    - In 5.0 the file is here: <VDSP50>\Blackfin\Examples\usb\host\windows\hostapp\hostapp.h
  - The USBCB structure has been redefined so you will need to update any references to the fields from this in 4.5:

```
typedef struct _USBCB             // USB command block
{
  ULONG ulCommand;                // command to execute
  ULONG ulData;                   // generic data field
  ULONG ulCount;                  // number of bytes to transfer
} USBCB, *PUSBCB;
```

    to this in 5.0:

```
typedef struct _USBCB             /* USB command block */
{
  unsigned int u32_Command;       /* command to execute */
  unsigned int u32_Data;          /* generic data field */
  unsigned int u32_Count;         /* number of bytes to transfer */
} USBCB, *PUSBCB;
```

  - MAX_DATA_BYTES_EZEXTENDER and MIN_DATA_BYTES_EZEXTENDER have been renamed to MAX_DATA_BYTES_BULKADI and MIN_DATA_BYTES_BULKADI respectively.

3.3. Things to Watch Out For

This is a short list of items that the user should watch out for while performing their port.

- As with any type of data, take special care when placing USB data buffers into cached memory. If it appears some data is not arriving from the host as expected it may be because you are using an area of cached memory that was not properly managed.
- If you are using the hostapp utility to perform file IO, remember that file paths are relative to the working directory of hostapp. Since directories have moved around in 5.0 you may need to adjust your file paths to account for this.
- In 5.0 the default LDFs bring in the USB libraries (USB stack, BF54x controller) but not the NET2272 driver. Be careful if you want make changes to the USB library sources that you are properly linking with the correct

objects. If you do not remove the old object from the default LDF or add the new source or object file to your project you may continue to link with the old object.