ANALOG DEVICES

ADI_NL6448BC33_54
DEVICE DRIVER

DATE: 31 JAN 2006

Table of Contents

Table of Figures

Document Revision History

Date	Description of Changes
31.1.2006	Document created

Table 1 - Revision History

1. Overview

The adi_nl6448bc33_54 device driver is designed to provide an interface between A-V EZ Extender Card and NEC NL6448bc33-54 TFT Color LCD module for the Blackfin family of processors.
The device driver is designed based on the Analog Device Driver Model, utilizing the underlying PPI device driver and appropriate system services, this include Timer System Service for generation of the horizontal and vertical synchronization.
The commands, events and return codes in device driver can be used by the application programs to establish an effective interface with the LCD.

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/lcd/nec/adi_nl6448bc33_54.h> This file contains all definitions, function prototypes etc. for the NEC LCD device driver.

2.2. Source Files

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

- adi_nl6448bc33_54.c

3. Lower Level Drivers

The adi_nl6448bc33_54 driver uses the PPI.

3.1. PPI

The PPI device driver is used by the adi_nl6448bc33_54 to output the video data to the LCD.
The PPI device is fully configurable via the driver controls.
Conditionally compile code can be used to specify, on processor families that contain multiple PPI devices, which PPI device to use for the driver. Edinburgh and Braemar class devices have only 1 PPI, when the LCD driver is opened; default PPI-0 is used.
For Teton class device with 2 PPIs, a macro USE_PPI1 is provided to the user to select PPI-0 or PPI-1.

When nl6448bc33_54 driver is open, the PPI configuration is set to:

```
ADI_DEV_CMD_VALUE_PAIR PPI_Cfg[] = {
{ ADI_PPI_CMD_SET_PORT_DIRECTION,       (void *)1          },// transmit mode
{ ADI_PPI_CMD_SET_TRANSFER_TYPE,        (void *)3          },// output with 1,2 or 3 syncs
{ ADI_PPI_CMD_SET_PORT_CFG,             (void *)1          },// 2 or 3 syncs
{ ADI_PPI_CMD_SET_DATA_LENGTH,          (void *)7          },// DLEN = 16 bits
{ ADI_PPI_CMD_SET_FS_INVERT,            (void *)1          },// FS1 & FS2 polarity
{ ADI_PPI_CMD_SET_DELAY_COUNT_REG,      (void *)143        },// clk delay after frame sync
{ ADI_PPI_CMD_SET_TRANSFER_COUNT_REG,   (void *)639        },// 640 active pixels
{ ADI_DEV_CMD_END,                      (void *)0          }
};
```

With control command ADI_PPI_CMD_SET_TIMER_FRAME_SYNC_1, the frame sync timer 0 configuration (Hsync) is set to

```
pulse_hi = 0; //negative action pulse
emu_run = 1; // timer0 counter runs during emulation
period = 800;  // timer0 period (800clk = 1H)
width = 96;  // timer0 width (96clk)
```

and with control command ADI_PPI_CMD_SET_TIMER_FRAME_SYNC_2, the frame sync time 1 configuration (Vsync) is set to

```
pulse_hi = 0; //negative action pulse
emu_run = 1; // timer1 counter runs during emulation
period = 0x668A0; // timer1 period (525H = 525*800clk)
width = 1600; // timer1 width (2H = 2*800clk)
```

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. Memory for the Device Manager and System Services is provided in the initialization functions (adi_xxx_Init()).

The adi_nl6448bc33_54 driver uses one PPI port and DMA transmit channel.

3.2. Interrupts

This driver does not use any interrupts directly, please see PPI documentation for resources required by PPI driver.

3.3. DMA

This driver does not use any DMA directly; however check the PPI documentation for DMA resources required by this PPI driver.

3.4. Timers

This driver does not use any timers directly; however check the PPI documentation for timer resources required by PPI driver.

3.5. Real-Time Clock

This driver does not require the real-time clock.

3.6. Programmable Flags

This driver does not use any programmable flags.

3.7. Pins

This driver does not use any external pins.

4. Supported Features of the Device Driver

This section describes what features are supported by the device driver.

4.1. Directionality

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

ADI_DEV_DIRECTION	Description
ADI_DEV_DIRECTION_OUTBOUND	Supports the transmission of data out through the device.

Table 2 - Supported Dataflow Directions

4.2. Dataflow Methods

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

ADI_DEV_MODE	Description
ADI_DEV_MODE_CIRCULAR	Supports the circular buffer method
ADI_DEV_MODE_CHAINED	Supports the chained buffer method
ADI_DEV_MODE_CHAINED_LOOPBACK	Supports the chained buffer with loopback method

Table 3 - Supported Dataflow Methods

4.3. Buffer Types

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

- ADI_DEV_CIRCULAR_BUFFER
  - Circular buffer
  - pAdditionalInfo – ignored
- ADI_DEV_2D_BUFFER
  - Two-dimensional buffer
  - pAdditionalInfo – ignored

4.4. 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.

4.4.1. 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

4.4.2. 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 4.2.
  - Value – ADI_DEV_MODE enumeration

- ADI_DEV_CMD_SET_DATAFLOW
  - Enables/disables dataflow through the device
  - Value – TRUE/FALSE

- ADI_DEV_CMD_GET_PERIPHERAL_DMA_SUPPORT
  - Determines if the device driver is supported by peripheral DMA
  - Value – u32 * (location where TRUE or FALSE is stored)

4.4.3. 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.

This driver does not have any specific commands.

4.5. Callback Events

This section enumerates the callback events the device driver is capable of generating. The events are divided into two sections. The first section describes events that are common to many device drivers. The next section describes driver specific event IDs. The callback function of the client should be prepared to process each of the events in these sections.

The callback function is of the type ADI_DCB_CALLBACK_FN. The callback function is passed three parameters. These parameters are:

- ClientHandle – This void * parameter is the value that is passed to the device driver as a parameter in the adi_dev_Open() function.

- EventID – This is a u32 data type that specifies the event ID.
- Value – This parameter is a void * whose value is context sensitive to the specific event ID.

The sections below enumerate the event IDs that the device driver can generate and the meaning of the Value parameter for each event ID.

4.5.1. Common Events

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

- ADI_DEV_EVENT_BUFFER_PROCESSED
  - Notifies callback function that a chained or sequential I/O buffer has been processed by the device driver. This event is also used to notify that an entire circular buffer has been processed if the driver was directed to generate a callback upon completion of an entire circular buffer.
  - Value – For chained or sequential I/O dataflow methods, this value is the CallbackParameter value that was supplied in the buffer that was passed to the adi_dev_Read(), adi_dev_Write() or adi_dev_SequentialIO() function. For the circular dataflow method, this value is the address of the buffer provided in the adi_dev_Read() or adi_dev_Write() function.
- ADI_DEV_EVENT_SUB_BUFFER_PROCESSED
  - Notifies callback function that a sub-buffer within a circular buffer has been processed by the device driver.
  - Value – The address of the buffer provided in the adi_dev_Read() or adi_dev_Write() function.
- ADI_DEV_EVENT_DMA_ERROR_INTERRUPT
  - Notifies the callback function that a DMA error occurred.
  - Value – Null.

4.5.2. 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.

This driver does not have any specific events.

4.6. 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. Wherever functions in the device driver API are called, the application should be prepared to process any of these return codes.

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
}
```

4.6.1. 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.

4.6.2. Device Driver Specific Return Codes

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

This driver does not have any specific return codes.

5. 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.

5.1. 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_NL6448bc3354_EntryPoint

5.2. 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
PPI device	0	0,1	See section 3.1.PPI

Table 4 - Default Settings

5.3. Additional Required Configuration Settings

No additional configuration settings are required for this driver.

6. Hardware Considerations

On the A-V EZ-Extender board, R2(10kOhm) is pulling the LCD's DPS(pin31) signal to Vdd, but there is already an internal resistor inside the LCD pulling DPS(pin31) signal to ground. To avoid a voltage divider at DPS make sure the R2 at the A-V EZ-Extender Card is removed, for location detail see yellow arrow at AV_Extender.jpg

Two example programs are provided to demonstrate how to generate an RGB 5-6-5 color bar pattern in SDRAM memory, sets up the interface to an LCD panel, and transmits the color-bar pattern in memory to the LCD.

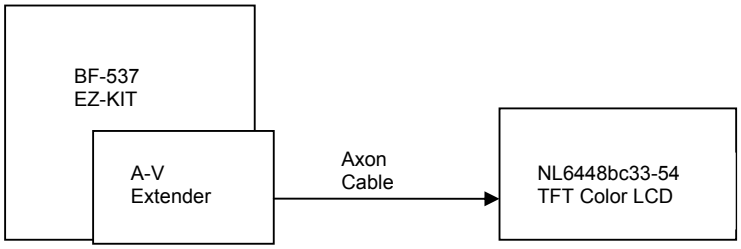
To test a chained buffer method:
- load LCDexamp_ch.c
- Comment out the macro USE_LOOPBACK.

To test a chained with loopback buffer method:
- Load LCDexamp_ch.c
- Include the macro USE_LOOPBACK.

To test a circular buffer method:
- Load LCDexamp_circ.c

The number of video frames to be created and send to LCD driver can be changed. (Default 4 color pattern)
For example, to display 2 different video frames, change the value of NUM_OF_VIDEO_FRAME to 2.

The value of NUM_BUFFERS defines the display period, by default each color pattern is displayed for about ½ to 1 second.



Hardware Requirements:

ADSP-BF537 EZ-KIT Board Rev 1.1
A-V EZ-Extender Card Rev. 1.2
NL6448bc33-54 TFT Color LCD Module.
Axon cable.

Switch settings on the ADSP-BF537 EZ-KIT Lite board:

Set all to default value. (EZ-Kit Lite Hardware Reference)

Switch settings on the A-V EZ-Extender board:

- JP2: Jump pins 1 and 2 together.
- JP4: Jump pins 1 and 2 together & Jump pins 3 and 4 together
- JP8: Jump pins 1 and 3 together, 2 and 4 together, and 7 and 8 together.
- JP5: Jump pins 3 and 4 together.

Connect the LCD panel to the A-V EZ-Extender board using the Axon cable.