ADI_LCD
DEVICE DRIVER

DATE: 02 JUNE 2006

Table of Contents

Table of Figures

Document Revision History

Date	Description of Changes
02/06/2006	Document created
07/27/2007	Typos/formatting and document name

Table 1 - Revision History

1. Overview

The lcd driver allows the client to control NEC NL6448BC33-54 or SHARP LQ10D368 TFT LCD.
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. specific to NEC NL6448BC33-54 LCD device
- <drivers/lcd/sharp/adi_lq10d368.h > This file contains all definitions, function prototypes etc. specific to SHARP LQ10D368 LCD device

2.2. Source Files

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

- <drivers/lcd/adi_lcd.c >
- <drivers/lcd/nec/adi_nl6448BC33_54.c >
- <drivers/lcd/sharp/adi_lq10d368.c >

Application must not include the adi_lcd.c file in directly to the project but either one of the files with the complete device name such as adi_nl6448BC33_54.c or adi_lq10d368.c.

3. Lower Level Drivers

The lcd driver is layered on the PPI driver.

3.1. PPI

The PPI device driver is used to output the video data to the LCD.
The PPI device is fully configurable via the driver controls.

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. 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.

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

4.1. Interrupts

No specific interrupts or interrupt handlers are used by this driver.

4.2. DMA

The driver doesn't support DMA directly, but uses a DMA driven PPI for its outbound video dataflow.

4.3. Timers

The driver uses following timers to generate PPI Frame Sync signals (Vsync & Hsync).

Processor	PPI Device Number	Timers
BF533	0	Timer 0 for FS1, Timer 1 for FS2
BF537	0	Timer 0 for FS1, Timer 1 for FS2
BF561	0	Timer 8 for FS1, Timer 9 for FS2
BF561	1	Timer 10 for FS1, Timer 11 for FS2

4.4. Real-Time Clock

This driver does not require the real-time clock.

4.5. Programmable Flags

This driver does not use any programmable flags.

4.6. Pins

This driver does not use any external pins.

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_OUTBOUND	Supports the transmission of data out 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_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

5.3. Buffer Types

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

- ADI_DEV_CIRCULAR_BUFFER
  - Circular buffer
  - pAdditionalInfo – ignored
- ADI_DEV_1D_BUFFER
  - Linear one-dimensional buffer
  - pAdditionalInfo – ignored
- ADI_DEV_2D_BUFFER
  - Two-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
- 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)

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.

Commands specific to **NEC NL6448BC33-54** LCD device

- ADI_NL6448BC3354_CMD_OPEN_PPI
  - Open PPI device and sets PPI device number to be used for dataflow
  - Value – u32

- ADI_NL6448BC3354_ CMD_GET_TOP_PADDING
  - Query about number of padding preceding active data
  - Value – u32* (location where number of padding is stored)

Commands specific to **SHARP LQ10D368** LCD device

- ADI_LQ10D368_CMD_OPEN_PPI
  - Open PPI device and sets PPI device number to be used for dataflow
  - Value – u32

- ADI_LQ10D368_ CMD_GET_TOP_PADDING
    - Query about number of padding preceding active data
    - Value – u32* (location where number of padding is stored)

5.5. 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 client should prepare its callback function to process each event described in these two 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.

5.5.1. 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.

5.5.2. 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.

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.

Return codes specific to TWI/SPI Device access service

- ADI_DEV_RESULT_TWI_LOCKED
  - Indicates the present TWI device is locked in other operation
- ADI_DEV_RESULT_REQUIRES_TWI_CONFIG_TABLE
  - Client need to supply a configuration table for the TWI driver
- ADI_DEV_RESULT_CMD_NOT_SUPPORTED
  - Command not supported by the Device Access Service
- ADI_DEV_RESULT_INVALID_REG_ADDRESS
  - The client attempting to access an invalid register address
- ADI_DEV_RESULT_INVALID_REG_FIELD
  - The client attempting to access an invalid register field location
- ADI_DEV_RESULT_INVALID_REG_FIELD_DATA
  - The client attempting to write an invalid data to selected register field location
- ADI_DEV_RESULT_ATTEMPT_TO_WRITE_READONLY_REG
  - The client attempting to write to a read-only location
- ADI_DEV_RESULT_ATTEMPT_TO_ACCESS_RESERVE_AREA
  - The client attempting to access a reserved location
- ADI_DEV_RESULT_ACCESS_TYPE_NOT_SUPPORTED
  - Device Access Service does not support the access type provided by the driver

5.6.2. 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.

Return codes specific to NEC NL6448BC33-54 LCD device

- ADI_NL6448BC3354_RESULT_BAD_PPI_DEVICE
  - Results when the client provides a wrong PPI device number

Return codes specific to SHARP LQ10D368 LCD device

- ADI_LQ10D368_RESULT_BAD_PPI_DEVICE
  - Results when the client provides a wrong PPI device number

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_NL6448BC3354_EntryPoint
- ADI_LQ10D368_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
PPI device	0	0,1	ADI_NL6448BC3354_CMD_OPEN_PPI or ADI_LQ10D368_CMD_OPEN_PPI

Table 4 - Default Settings

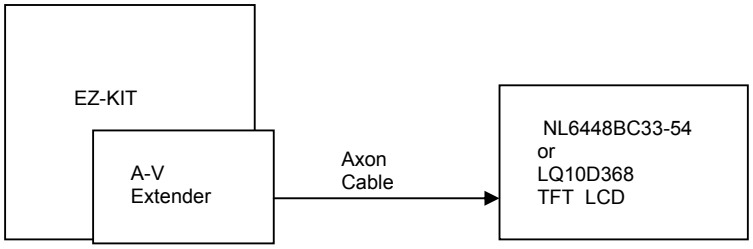
6.3. Additional Required Configuration Settings

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
PPI Device	0 (for BF533, BF537) 0, 1 (for BF561)	ADI_NL6448BC3354_CMD_OPEN_PPI or ADI_LQ10D368_CMD_OPEN_PPI

Table 5 – Additional Required Settings

7. Hardware Considerations



Hardware Requirements:
ADSP EZ-KIT Board
A-V EZ-Extender Card Rev. 1.2
NEC NL6448BC33-54 TFT LCD or SHARP LQ10D368 TFT LCD
Axon cable.

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.

NOTE: specific to NEC NL6448BC33-54 LCD device

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.

NOTE: specific to SHARP LQ10D368 LCD device

On the A-V EZ-Extender board, ENAB signal(pin27) of Flat panel display interface P6 must be pulled "low" all the time.

8. Appendix

8.1. Using LCD Device Driver in Applications

This section explains how to use adi_lcd device driver with an application.

Initialize Ez-Kit, Interrupt manager, Deferred Callback Manager, DMA Manager, Device Manager (all application dependent)

a. LCD driver initialization

Step 1: Open LCD Device driver with device specific entry point.

- ADI_NL6448BC3354_EntryPoint for NEC LCD or
- ADI_LQ10D368_EntryPoint for SHARP LCD

Example:
// Open an SHARP LQ10D368 device driver.
adi_dev_Open(DeviceManagerHandle, ADI_LQ10D368_EntryPoint, 0, NULL, &LcdDriverHandle,
ADI_DEV_DIRECTION_OUTBOUND, DMAMgrHandle, DCBMgrHandle, ClientCallback)

Step 2: Open PPI and select PPI device number to be used for video output

Example:
// Open and set PPI 1 for SHARP LQ10D368 the AD_BF561
adi_dev_Control (LcdDriverHandle, ADI_LQ10D368_CMD_OPEN_PPI (void*)1);

Step 3: Set video dataflow method

Example:
// Set device dataflow method ADI_DEV_MODE_CHAINED_LOOPBACK
adi_dev_Control (LcdDriverHandle , ADI_DEV_CMD_SET_DATAFLOW_METHOD,
(void *)ADI_DEV_MODE_CHAINED_LOOPBACK)

Step 4: Load video output buffers

Example:
// Load OutBuffer for video processing
adi_dev_Write(LcdDriverHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *)&OutBuffer)

Step 5: Get the number of buffer padding

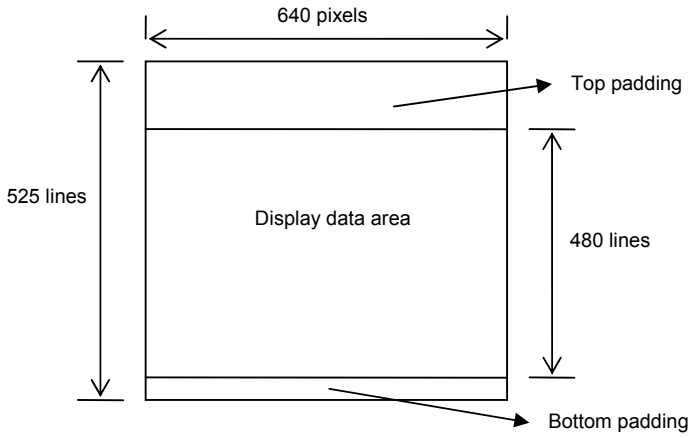
Example:
// Get Sharp LCD top padding
adi_dev_Control(LcdDriverHandle, ADI_LQ10D368_CMD_GET_TOP_PADDING, (void*)&padding)

Step 6: Start video dataflow

Example:
// start outputting LCD video data
adi_dev_Control(LcdDriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void*)TRUE)

b. Display Position on the LCD screen

LCD rows and columns are synchronized to Horizontal sync. signal and Vertical sync. signal generated by the timer. Vertical sync. signal cycles 525 lines (inc. top padding lines , active data and bottom padding lines) and the active data display starts at the top line of the display data area.
Application program shall provide video output buffer of the size of 525 lines * 640 pixels and places the video data within the display data area.
NEC NL6448BC3354 LCD's display data area starts from the 33rd lines (top padding) of the buffer and
Sharp LQ10D368 LCD's display data area starts from the 34th lines (top padding) of the buffer.
Example:
Application provides an Output_buffer[640*525] located at "SDRAM" address 0x00800000.
To get the top padding value, control command is used
adi_dev_Control(LcdDriverHandle, ADI_LQ10D368_CMD_GET_TOP_PADDING, (void*)&padding).
Offset from the start of the buffer to the display data area of the buffer is 0x00800000 + (padding*640).



c. Terminating LCD driver

Step 6: Terminate driver with adi_dev_Terminate( )

Terminate DMA Manager, Deferred Callback etc.., (application dependent)