ANALOG DEVICES

ADI_T350MCQB01 DEVICE DRIVER

DATE: MAY 16, 2007

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
Nov 1, 2007	Initial release

Table 1 – Revision History

1.Overview

This document describes use of Varitronix T350MCQB01 LCD device driver.

Varitronix T350MCQB01 LCD is available on ADSP-BF527 Ez-Kit Lite. This driver is built on top of PPI driver and configures PPI windowing and Frame Sync/Blank generation registers with Varitronix T350MCQB01 LCD specific values.

The Varitronix T350MCQB01 LCD driver has been tested on the ADSP-BF527 EZ-Kit Lite development board.

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/varitronix/adi_t350mcqb01.h>
- This file contains all definitions, function prototypes etc. specific to the Varitronix T350MCQB01 LCD

2.2.Source Files

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

< Blackfin/lib/src/drivers/ lcd/varitronix/adi_t350mcqb01.c>
- This file contains all the source code for the Varitronix T350MCQB01 LCD driver. All source code is written in ‘C’. There are no assembly level functions in this driver.

3.Lower Level Drivers

The Varitronix T350MCQB01 LCD driver is built on top of PPI driver.

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.

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

The Varitronix T350MCQB01 LCD device requires two additional (driver) memory of size ADI_DEV_DEVICE_MEMORY (one for the LCD driver and one for the PPI driver).

4.1.Interrupts

The LCD driver does not require any additional interrupt memory. But additional memory is required for the PPI driver. Please refer to PPI driver manual for memory requirements.

4.2.DMA

The driver doesn’t support DMA directly, but uses a DMA driven PPI for its video dataflow. Please refer to PPI driver manual for DMA support and memory requirements.

4.3.Timers

The LCD driver uses timer service to generate the LCD sync signals.

4.4.Real-Time Clock

The LCD driver does not use any real-time clock services.

4.5.Programmable Flags

The LCD driver does not use any programmable flag services.

4.6.Pins

In addition to DISP signal pin, the LCD driver requires pins used by allocated PPI device. Refer to PPI driver manual for pin requirements specific to PPI.

5.Supported Features of the Device Driver

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

5.1.Directionality

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 loop back 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 – optional

ADI_DEV_1D_BUFFER
- One-dimensional buffer
- pAdditionalInfo – optional

ADI_DEV_2D_BUFFER
- Two-dimensional buffer
- pAdditionalInfo – optional

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_GET_2D_SUPPORT
- Determines if the driver can support 2D buffers
- Value – u32 * (location where TRUE/FALSE is stored)

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_STREAMING
- Enables/disables the streaming mode of the driver.
- Value – TRUE/FALSE

ADI_DEV_CMD_GET_INBOUND_DMA_CHANNEL_ID
- Returns the DMA channel ID value for the device driver's inbound DMA channel
- Value – u32 * (location where the channel ID is stored)

ADI_DEV_CMD_GET_OUTBOUND_DMA_CHANNEL_ID
- Returns the DMA channel ID value for the device driver's outbound DMA channel
- Value – u32 * (location where the channel ID is stored)

ADI_DEV_CMD_SET_INBOUND_DMA_CHANNEL_ID
- Sets the DMA channel ID value for the device driver's inbound DMA channel
- Value – ADI_DMA_CHANNEL_ID (DMA channel ID)

ADI_DEV_CMD_SET_OUTBOUND_DMA_CHANNEL_ID
- Sets the DMA channel ID value for the device driver's outbound DMA channel
- Value – ADI_DMA_CHANNEL_ID (DMA channel ID)

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)

ADI_DEV_CMD_SET_ERROR_REPORTING
- Enables/Disables error reporting from the device driver
- Value – TRUE/FALSE

ADI_DEV_CMD_GET_MAX_INBOUND_SIZE
- Returns the maximum number of data bytes for an inbound buffer
- Value – u32 * (location where the size is stored)

ADI_DEV_CMD_GET_MAX_OUTBOUND_SIZE
- Returns the maximum number of data bytes for an outbound buffer
- Value – u32 * (location where the size is stored)

ADI_DEV_CMD_FREQUENCY_CHANGE_PROLOG
- Notifies device driver immediately prior to a CCLK/SCLK frequency change
- Value – ADI_DEV_FREQUENCIES * (new frequencies)

ADI_DEV_CMD_FREQUENCY_CHANGE_EPILOG
- Notifies device driver immediately following a CCLK/SCLK frequency change
- Value – ADI_DEV_FREQUENCIES * (new frequencies)

ADI_DEV_CMD_GET_INBOUND_DMA_INFO
- Gets Inbound DMA channel Information
- Value – ADI_DEV_DMA_INFO * (DMA channel information table)

ADI_DEV_CMD_GET_OUTBOUND_DMA_INFO
- Gets Outbound DMA channel Information
- Value – ADI_DEV_DMA_INFO * (DMA channel information table)

ADI_DEV_CMD_OPEN_PERIPHERAL_DMA
- Device manager opens a DMA channel for the peripheral
- Value – ADI_DEV_DMA_INFO * (DMA channel information table)

ADI_DEV_CMD_CLOSE_PERIPHERAL_DMA
- Device manager closes a DMA channel used by a peripheral
- Value – ADI_DEV_DMA_INFO * (DMA channel information table)

5.4.3.Device Driver Specific Commands

In addition to the commands listed below, the LCD driver supports all PPI driver specific commands. Refer to PPI driver manual for list of commands supported by PPI.

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

ADI_T350MCQB01_CMD_SET_PPI_DEV_NUMBER
- Sets the PPI Device number to use
- Value – u8

ADI_T350MCQB01_CMD_SET_OPEN_PPI_DEVICE
- Open/Close PPI Device connected to this LCD
- Value –TRUE/FALSE, TRUE to open & FALSE to close PPI

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 the PPI 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 driver supports all events generated by PPI and passes them to client callback function without any change. There are no event codes specific to this driver.

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

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

ADI_T350MCQB01_RESULT_PPI_STATE_INVALID
- Results when the client submits outbound buffer(s) with PPI device in closed.

ADI_T350MCQB01_RESULT_CMD_NOT_SUPPORTED
- Command not recognised or supported by this driver

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.

ADIT350MCQB01EntryPoint

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
Vertical Transfer Count Register	241	241	ADI_PPI_CMD_SET_VERTICAL_TX_COUNT
Horizontal Delay Register	204	204	ADI_PPI_CMD_SET_HORIZONTAL_DELAY
Horizontal Transfer Count Register	480	480	ADI_PPI_CMD_SET_HORIZONTAL_TX_COUNT
Frame Sync 1 Width Register	90	90	ADI_PPI_CMD_SET_FS1_WIDTH
Frame Sync 2 Width Register	3672	3672	ADI_PPI_CMD_SET_FS2_WIDTH
Frame Sync 1 Period Register	1224	1224	ADI_PPI_CMD_SET_FS1_PERIOD
Frame Sync 2 Period Register	294984	294984	ADI_PPI_CMD_SET_FS2_PERIOD

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
Control Register	Application dependent	ADI_PPI_CMD_SET_CONTROL_REG

Table 5 – Additional Required Settings

7.Hardware Considerations

All hardware considerations specific to PPI applies to this driver. Refer to PPI driver manual for more information.

8.Using Varitronix T350MCQB01 LCD Driver in Applications

This section explains how to use PPI device driver in an application.

8.1.Device Manager Data memory allocation

This section explains device manager memory allocation requirements for applications using this driver. The application should allocate base memory + memory for 1 VARITRONIX T350MCQB01 LCD driver+ memory for 1 PPI device + memory for other devices used by the application

8.2.Interrupt Manager Data memory allocation

This section explains Interrupt manager memory allocation requirements for applications using this driver. The application should allocate at least two secondary interrupt memory of size **ADI_INT_SECONDARY_MEMORY** – one for PPI DMA Data interrupt handler and DMA error interrupt handler. Also, additional secondary interrupt memory must be provided in case the client uses Pixel compositor of decides to enable PPI error reporting. Refer to corresponding driver documents for memory requirements.

8.3.Typical usage of VARITRONIX T350MCQB01 LCD device driver

a. VARITRONIX T350MCQB01 LCD (driver) initialization

Step 1: Open LCD Device driver with device specific entry point (refer section 6.1 for valid entry point) and data direction

Step 2: Configure driver with PPI device number to use

Step 3: Open the PPI device allocated to this driver

Step 4: Configure PPI control register specific LCD operating mode (refer section (page) for examples)

Step 5: Enable PPI error reporting (if required)

Step 6: Set LCD video dataflow method

b. Submitting buffers

Step 7: Queue outbound buffers to LCD (PPI DMA) using adi_dev_Write( )

Step 8: Enable LCD Dataflow

Step 9: Respond to callbacks

c. Terminating VARITRONIX T350MCQB01 LCD driver

Step 10: Disable LCD dataflow

Step 11: Disable PPI error reporting (if already enabled)

Step 12: Wait until DISP signal END generation is complete.

Step 13: Terminate LCD driver with adi_dev_Close( )

Terminate DMA Manager, Deferred Callback, Flag Manager, DMA Manager, Device Manager (application dependent)