ANALOG DEVICES

ADI_LQ043T1DG01
DEVICE DRIVER

DATE: MAR 24, 2010

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
May 16, 2007	Initial release
Oct 06, 2008	Added new command to select EPPI control register configuration mode
Feb 25, 2009	EPPI Control register configuration is now disabled by default
Mar 24, 2010	Added buffer size requirements for each video frame

Table 1 – Revision History

1. Overview

This document describes use of Sharp LQ043T1DG01 LCD device driver.

Sharp LQ043T1DG01 LCD is available on ADSP-BF548 Ez-Kit Lite. This driver is built on top of EPPI driver and configures EPPI windowing and Frame Sync/Blank generation registers with Sharp LQ043T1DG01 LCD specific values. In addition to that, the driver also generates/controls the LCD DISP signal (used to ON/OFF the LCD). Terminating the LCD dataflow abnormally locks the display, which can be fixed only by power-cycling the hardware.

The Sharp LQ043T1DG01 LCD driver has been tested on the ADSP-BF548 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/sharp/adi_lq043t1dg01.h>**
  - This file contains all definitions, function prototypes etc. specific to the Sharp LQ043T1DG01 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/sharp/adi_lq043t1dg01.c>**
  - This file contains all the source code for the Sharp LQ043T1DG01 LCD driver. All source code is written in 'C'. There are no assembly level functions in this driver.

3. Lower Level Drivers

The Sharp LQ043T1DG01 LCD driver is built on top of EPPI driver.

3.1. EPPI Device Driver

Sharp LQ043T1DG01 LCD driver only supports EPPI based devices and is tested for ADSP-BF548 Ez-Kit. The driver configures EPPI control and frame sync registers as per LCD requirements. Application can choose not to configure EPPI control register or select a particular configuration mode using a driver specific command. Application can change EPPI register settings using EPPI driver specific commands.

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

Each Sharp LQ043T1DG01 LCD device requires two additional (driver) memory of size **`ADI_DEV_DEVICE_MEMORY`**, one for the LCD driver and one for EPPI driver.

4.1. Interrupts

The LCD driver requires two additional memory of size **`ADI_INT_SECONDARY_MEMORY`** for each EPPI DMA channel – one for DMA Data interrupt handler and one for DMA error interrupt handler. Additional memory of **`ADI_INT_SECONDARY_MEMORY`** size must be provided when the client decides to enable EPPI error reporting

4.2. DMA

The driver doesn't support DMA directly, but uses a DMA driven EPPI for its video dataflow. The LCD driver only supports outbound dataflow and enough memory should be allocated for EPPI DMA channels depending on the EPPI operating mode (single or multi-channel DMA mode). Each DMA channel requires memory of size **`ADI_DMA_CHANNEL_MEMORY`**.

4.3. Timers

The LCD driver uses timer service to generate LCD DISP signal. Client must configure the driver with a Timer ID that can be used to generate the DISP signal and Flag ID to which the DISP signal is connected to.

4.4. Real-Time Clock

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

4.5. Programmable Flags

The LCD device driver uses flag service to control the LCD DISP signal status. Client must configure the driver with a Flag ID to which the LCD DISP signal is connected.

4.6. Pins

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

5. Buffer Size Requirement

The Sharp LQ043T1DG01 LCD is a landscape display with a resolution of 480 x 272. The buffer size required for each video frame is:

480 x 272 x 3 bytes per pixel = 391680 bytes.

6. Supported Features of the Device Driver

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

6.1. Directionality

ADI_DEV_DIRECTION	Description
ADI_DEV_ DIRECTION_OUTBOUND	Supports the transmission of data out through the device.

Table 2 – Supported Dataflow Directions

6.2. 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

6.3. 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

6.4. 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.

6.4.1. 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

6.4.2. 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 6.2.
  - 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_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)

6.4.3. Device Driver Specific Commands

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

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

- **ADI_LQ043T1DG01_CMD_SET_DISP_TIMER_FLAG**
  - Sets the Timer ID that can be used to generate LCD DISP signal & Flag ID to which the DISP signal is connected to.
  - Value – **ADI_LQ043T1DG01_TIMER_FLAG ***
  - Note: Refer section 9.4 for example

```
/*
** ADI_LQ043T1DG01_TIMER_FLAG
**  - Data Structure to pass Timer & Flag ID to generate DISP signal
*/

typedef struct
{

        /* Variable: DispTimerId
                - Timer ID to generate DISP signal for the LCD */
         u32                DispTimerId;

        /* Variable: DispFlagId
              - Flag ID to which the DISP signal is connected to */
        ADI_FLAG_ID      DispFlagId;

} ADI_LQ043T1DG01_TIMER_FLAG;
```

- **ADI_LQ043T1DG01_CMD_SET_EPPI_DEV_NUMBER**
  - Sets the EPPI Device number to use
  - Value – u8
- **ADI_LQ043T1DG01_CMD_SET_OPEN_EPPI_DEVICE**
  - Open/Close EPPI Device connected to this LCD
  - Value –TRUE/FALSE, TRUE to open & FALSE to close EPPI
- **ADI_LQ043T1DG01_CMD_SET_EPPI_CTRL_REG_MODE**
  - Configure EPPI Control Register to pre-defined mode
  - Value – enumerated value of type **ADI_LQ043T1DG01_EPPI_CTRL_REG_MODE**
  - Default – **ADI_LQ043T1DG01_EPPI_CTRL_REG_AUTO_CONFIG_DISABLED**
  - Note: Automatic configuration of EPPI control register is now disabled by default. Selecting a valid, pre-defined, EPPI control register configuration value captured in the below table will stop the programmer from configuring EPPI control register fields directly from the application. If the programmer intends to configure the EPPI Control register fields from the application, then the Automatic configuration of EPPI control register should be disable by passing '**ADI_LQ043T1DG01_EPPI_CTRL_REG_AUTO_CONFIG_DISABLED**' as command value parameter. Refer section 9.4 for example.

```
/*
** ADI_LQ043T1DG01_EPPI_CTRL_REG_MODE
**  - Enumerations of EPPI Control register configuration modes supported by the LCD driver
*/

typedef enum __AdiLq043t1dg01EppiCtrlRegMode
{

        /* Mode to disable driver from automatically configuring EPPI control register */
        ADI_LQ043T1DG01_EPPI_CTRL_REG_AUTO_CONFIG_DISABLED = 0,

        /* Mode to configure EPPI Control register for RGB 24-bit out

                - EPPI as Output, GP 2 frame sync mode, Internal Clock generation disabled,
                  Internal FS generation enabled, Receives samples on EPPI_CLK raising edge,
                  Transmits samples on EPPI_CLK falling edge, FS1 & FS2 are active high,
                  24-bits for RGB888 out, DMA unpacking enabled, Swapping Disabled,
                  One (DMA) Channel Mode, RGB Formatting disabled,
                  Regular watermark - 75% full, Urgent watermark - 25% full */
        ADI_LQ043T1DG01_EPPI_CTRL_REG_RGB24 = 0x68136E2E,

        /* Mode to configure EPPI Control register to
           format RGB 24-bit data and output as RGB 18-bit on-the-fly

                - EPPI as Output, GP 2 frame sync mode, Internal Clock generation disabled,
                  Internal FS generation enabled, Receives samples on EPPI_CLK raising edge,
                  Transmits samples on EPPI_CLK falling edge, FS1 & FS2 are active high,
                  16-bits for RGB666 out, DMA unpacking enabled, Swapping Disabled,
                  One (DMA) Channel Mode, RGB Formatting enabled,
                  Regular watermark - 75% full, Urgent watermark - 25% full */
        ADI_LQ043T1DG01_EPPI_CTRL_REG_RGB24_TO_RGB18 = 0x6C12EE2E,

        /* Mode to confgiure EPPI Control register for RGB 18-bit out

                - EPPI as Output, GP 2 frame sync mode, Internal Clock generation disabled,
                  Internal FS generation enabled, Receives samples on EPPI_CLK raising edge,
                  Transmits samples on EPPI_CLK falling edge, FS1 & FS2 are active high,
                  16-bits for RGB666 out, DMA unpacking enabled, Swapping Disabled,
                  One (DMA) Channel Mode, RGB Formatting disabled,
                  Regular watermark - 75% full, Urgent watermark - 25% full */
        ADI_LQ043T1DG01_EPPI_CTRL_REG_RGB18 = 0x6812EE2E,
```

```
} ADI_LQ043T1DG01_EPPI_CTRL_REG_MODE;
```

6.5. 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.

6.5.1. 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.

6.5.2. Device Driver Specific Events

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

6.6. 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 */
}
```

6.6.1. 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_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.

6.6.2. 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_LQ043T1DG01_RESULT_PPI_STATE_INVALID**
  - Results when the client tries to submit buffer(s) when EPPI is disabled.
- **ADI_LQ043T1DG01_RESULT_DISP_TIMER_FLAG_INVALID**
  - Results when client tries to enable LCD dataflow with invalid DISP generation Timer ID or Flag ID
- **ADI_LQ043T1DG01_RESULT_CMD_NOT_SUPPORTED**
  - Command not recognised or supported by this driver
- **ADI_LQ043T1DG01_DISP_GENERATION_ALREADY_IN_PROGRESS**
  - Results when client tries to enable/disable dataflow while DISP signal generation already in progress

7. 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.

7.1. 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.

- **`ADILQ043T1DG01EntryPoint`**

7.2. 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
Lines per Frame Register	286	286	**`ADI_EPPI_CMD_SET_LINES_PER_FRAME`**
Samples Per Line Register	525	525	**`ADI_EPPI_CMD_SET_SAMPLES_PER_LINE`**
Vertical Delay Register	12	12	**`ADI_EPPI_CMD_SET_VERTICAL_DELAY`**
Vertical Transfer Count Register	272	272	**`ADI_EPPI_CMD_SET_VERTICAL_TX_COUNT`**
Horizontal Delay Register	43	43	**`ADI_EPPI_CMD_SET_HORIZONTAL_DELAY`**
Horizontal Transfer Count Register	480	480	**`ADI_EPPI_CMD_SET_HORIZONTAL_TX_COUNT`**
Clock Divide Register	Sets for 8MHz	7.86 to 9.26 MHz	**`ADI_EPPI_CMD_SET_CLOCK_DIV`**
Frame Sync 1 Width Register	41	41	**`ADI_EPPI_CMD_SET_FS1_WIDTH`**
Frame Sync 2 Width Register	5250	5250	**`ADI_EPPI_CMD_SET_FS2_WIDTH`**
Frame Sync 1 Period Register	525	525	**`ADI_EPPI_CMD_SET_FS1_PERIOD`**
Frame Sync 2 Period Register	150150	150150	**`ADI_EPPI_CMD_SET_FS2_PERIOD`**
Clipping Register	0xFF00FF00	0xFFFFFFFF to 0	**`ADI_EPPI_CMD_SET_CLIPPING`**

EPPI Control Register
Default value : **`ADI_LQ043T1DG01_EPPI_CTRL_REG_AUTO_CONFIG_DISABLED`**
Possible values : enumerated value of type **`ADI_LQ043T1DG01_EPPI_CTRL_REG_MODE`**
Configuration command : **`ADI_LQ043T1DG01_CMD_SET_EPPI_CTRL_REG_MODE`**

Table 4 – Default Settings

7.3. 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
DISP Signal Timer and Flag ID	Hardware Dependent	**ADI_LQ043T1DG01_CMD_SET_DISP_TIMER_FLAG**

Table 5 – Additional Required Settings

8. Hardware Considerations

The driver uses flag service to configure DISP signal Flag ID in output mode. In addition to that, all hardware considerations specific to EPPI applies to this driver. Refer to EPPI driver manual for more information.

9. Using Sharp LQ043T1DG01 LCD Driver in Applications

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

9.1. 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 EPPI DMA Data interrupt handler and DMA error interrupt handler. Also, additional secondary interrupt memory must be provided when EPPI error reporting is enabled. Refer to corresponding driver documents for memory requirements

9.2. DMA Manager Data memory allocation

This section explains DMA manager memory allocation requirements for applications using this driver. The application should allocate base memory + memory for EPPI DMA channel(s) (one or two DMA channel depending on EPPI operating mode). Each DMA channel requires memory of size **`ADI_DMA_CHANNEL_MEMORY.`**

9.3. 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 SHARP LQ043T1DG01 LCD driver + memory for 1 EPPI device + memory for other devices used by the application. Each device require memory of size **`ADI_DEV_DEVICE_MEMORY.`**

9.4. Typical usage of SHARP LQ043T1DG01 LCD device driver

a. SHARP LQ043T1DG01 LCD (driver) initialization

Step 1: Open LCD Device driver with device specific entry point (refer section 7.1 for valid entry point) and data direction

Step 2: Configure LCD with Timer ID to generate DISP signal and FLAG ID to control DISP.

```
 /*
** Example: Set LCD DISP Signal generation Timer and Flag for ADSP-BF548 Ez-Kit
*/

/* Define instance to hold LCD DISP Timer and Flag ID */
ADI_LQ043T1DG01_TIMER_FLAG              oLcdDisp;

/* Timer ID to be used to generate DISP signal */
oLcdDisp.DispTimerId = ADI_TMR_GP_TIMER_3;
/* Flag port to with LCD DISP pin is connected to */
oLcdDisp.DispFlagId  = ADI_FLAG_PE3;

/* Pass DISP signal generation Timer and Flag information */
adi_dev_Control (LcdDriverHandle, ADI_LQ043T1DG01_CMD_SET_DISP_TIMER_FLAG, (void *) &oLcdDisp);
```

Step 3: Configure driver with EPPI device number to use

```
/* Example: Use EPPI 0 for ADSP-BF548 Ez-Kit */
adi_dev_Control (LcdDriverHandle, ADI_LQ043T1DG01_CMD_SET_EPPI_DEV_NUMBER, (void*) 0);
```

Step 4: Open the EPPI device allocated to this driver

```
/* Example: Open EPPI device allocated for this LCD */
adi_dev_Control (LcdDriverHandle, ADI_LQ043T1DG01_CMD_SET_OPEN_EPPI_DEVICE, (void*) true);
```

Step 5: Configure EPPI control register specific LCD operating mode (refer page 14 for other configuration modes)

```
/* Example: Configure EPPI Control register for 24-bit RGB output */
adi_dev_Control (LcdDriverHandle,
                 ADI_LQ043T1DG01_CMD_SET_EPPI_CTRL_REG_MODE,
                 (void*)ADI_LQ043T1DG01_EPPI_CTRL_REG_RGB24);
```

Step 6: Enable EPPI error reporting (only if required)

Step 7: Set LCD video dataflow method

```
/* Example: Configure LCD in chained loopback dataflow mode */
adi_dev_Control (LcdDriverHandle,
                 ADI_DEV_CMD_SET_DATAFLOW_METHOD,
                 (void *)ADI_DEV_MODE_CHAINED_LOOPBACK);
```

b. Submitting buffers

Step 8: Queue outbound buffers to LCD (EPPI DMA) using adi_dev_Write( )

Step 9: Enable LCD Dataflow

```
/* Example: Enable LCD dataflow */
adi_dev_Control (LcdDriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void *) true);
```

Step 10: Respond to callbacks

c. Terminating SHARP LQ043T1DG01 LCD driver

Step 11: Disable LCD dataflow

```
/* Example: Disable LCD dataflow */
adi_dev_Control (LcdDriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void *) false);
```

Step 12: Disable EPPI error reporting (if already enabled)

Step 13: Wait until DISP signal END generation is complete.

Step 14: Terminate LCD driver with adi_dev_Close( )

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