ADI_LQ035Q1DH02
DEVICE DRIVER

DATE: MAR 24, 2010

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
Oct 07, 2008	Initial release
Nov 19, 2009	Added two new commands to support use of LCD on BF527 EZ-Kit Rev 2.1 and later.
Mar 24, 2010	Added buffer size requirements for each video frame.

Table 1 – Revision History

1. Overview

This document describes use of Sharp LQ035Q1DH02 LCD device driver.

Sharp LQ035Q1DH02 LCD is available on Blackfin Landscape LCD Ez-Extender and on the ADSP-BF527 EZ-KIT Lite from Revision 2.1 onwards. The driver for this display device is built on top of PPI/EPPI driver. The LCD driver configures PPI/EPPI control register and Frame Sync/Blank generation registers with Sharp LQ035Q1DH02 LCD specific values.

The Sharp LQ035Q1DH02 LCD driver has been tested with ADSP-BF526 Ez-Kit Lite, ADSP-BF527 Ez-Kit Lite and ADSP-BF548 EZ-Kit Lite development boards.

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/spi/adi_spi.h>`**
    - This file contains all definitions, function prototypes etc. specific to SPI device.
- **`<drivers/lcd/sharp/adi_lq035q1dh02.h>`**
    - This file contains all definitions, function prototypes etc. specific to the Sharp LQ035Q1DH02 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_lq035q1dh02.c>`**
    - This file contains all the source code for the Sharp LQ035Q1DH02 LCD driver. All source code is written in ‘C’. There are no assembly level functions in this driver.

3. Lower Level Drivers

The Sharp LQ035Q1DH02 LCD driver is built on top of SPI and PPI/EPPI driver.

3.1. SPI Device Driver

SPI device driver is used to configure Sharp LQ035Q1DH02 hardware registers.

3.2. PPI / EPPI Device Driver

Depending on the processor type, the Sharp LQ035Q1DH02 LCD driver leverages PPI or EPPI driver to control its video dataflow. The driver automatically configures PPI/EPPI control register and frame sync registers as per LCD requirements. Application can disable automatic configuration of PPI/EPPI control register using LCD specific command and use PPI/EPPI driver specific commands to pass its own configuration settings.

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 LQ035Q1DH02 LCD device requires three additional (driver) memory of size **`ADI_DEV_DEVICE_MEMORY`**, one for the LCD driver, Ope for SPI and one for PPI/EPPI driver.

4.1. Interrupts

The LCD driver requires three secondary interrupt memories, each of size **`ADI_INT_SECONDARY_MEMORY`**, to handle PPI/EPPI DMA Data interrupt, DMA error interrupt and SPI data interrupt. Additional memory of **`ADI_INT_SECONDARY_MEMORY`** size must be provided when the client decides to enable PPI/EPPI error reporting

4.2. DMA

The driver doesn't support DMA directly, but uses a DMA driven PPI/EPPI for its video dataflow. The LCD driver only supports outbound dataflow and memory of size **`ADI_DMA_CHANNEL_MEMORY`** must be allocated to PPI/EPPI DMA channel.

4.3. Timers

The LCD driver uses timer indirectly via PPI/EPPI driver to generate required Frame Sync signals.

4.4. Real-Time Clock

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

4.5. Programmable Flags

The LCD device driver does not use Flag control service.

4.6. Pins

The LCD driver requires pins used by SPI and PPI/EPPI device.

5. Buffer Size Requirement

The Sharp LQ035Q1DH02 LCD is a landscape display with a resolution of (QVGA) 320 x 240. The LCD requires 2 padding lines at the top and two padding lines at the bottom. The buffer size for each video frame depending upon using PPI or EPPI is:

If PPI is used:

320 x (240 + 4 padding lines) x 2 bytes per pixel = 156160 bytes.

If EPPI is used:

When EPPI is used, EPPI is configured to add the required padding lines. So the video frame buffer size does not include padding line. The buffer size required in this case is:

320 x 240 x 2 bytes per pixel = 153600 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_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 PPI/EPPI driver specific commands. Refer to PPI/EPPI driver manual for list of commands supported by PPI/EPPI.

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

SPI related commands

- **`ADI_LQ035Q1DH02_CMD_SET_SPI_DEVICE_NUMBER`**
  - Sets SPI device number to use to configure LCD registers
  - Value – u8 (SPI device number to use)
  - Default – 0xFF (SPI device number invalid)
- **`ADI_LQ035Q1DH02_CMD_SET_SPI_CHIP_SELECT`**
  - Sets SPI Chip select Number connected to LCD CSB pin
  - Value – u8 (SPI Chip select number to use)
  - Default – 0 (SPI Chip select invalid)

PPI / EPPI related commands

- **`ADI_LQ035Q1DH02_CMD_OPEN_PPI`**
  - Opens PPI device number connected to LCD
  - Value – u8 (PPI device number to open)
  - Default – None
- **`ADI_LQ035Q1DH02_CMD_OPEN_PPI`**
  - Closes PPI device previously opened for this LCD device
  - Value – NULL
- **`ADI_LQ035Q1DH02_CMD_ENABLE_AUTO_PPI_CONTROL_CONFIG`**
  - Enable/Disable Automatic PPI control register configuration
  - Value – true/false
  - Default – true
  - Note: 'true' to allow the driver to automatically configure PPI control register as per LCD requirements, 'false' to configure PPI / EPPI control register from the application.

- **`ADI_LQ035Q1DH02_CMD_SET_PPI_CTL_VALUE`**
  - Supplies value for PPI_CONTROL register if default is not suitable.
  - Value = the appropriate PPI_CONTROL value
  - Default = suitable value for LCD Extender card
- **`ADI_LQ035Q1DH02_CMD_SET_PPI_CLK_PER_DOT`**
  - Supplies ratio of PPI clocks to LCD 'dot' clocks.
  - Value = the appropriate ratio (e.g. 3 on BF527 EZ-Kit)
  - Default = 1 (Suitable for the LCD Extender card)

Sharp LQ035Q1DH02 LCD query commands

- **`ADI_LQ035Q1DH02_CMD_GET_TOP_PAD_LINES_PER_FRAME`**
  - Gets Number of padding lines required at the start/top of each frame
  - Value – u32 * (address of client location to store top padding value)
  - Default – requires 2 lines of top padding for PPI. Padding not required for EPPI.

- **`ADI_LQ035Q1DH02_CMD_GET_BOTTOM_PAD_LINES_PER_FRAME`**
  - Gets Number of padding lines required at the end/bottom of each frame
  - Value – u32 * (address of client location to store bottom padding value)
  - Default – requires 2 lines of bottom padding for PPI. Padding not required for EPPI.

Sharp LQ035Q1DH02 LCD configuration commands

- **`ADI_LQ035Q1DH02_CMD_SET_OUTPUT_SHIFT_DIRECTION`**
  - Sets Output shift direction (Top to Bottom and Left to Right) of the LCD display
  - Value – Enumeration of type **`ADI_LQ035Q1DH02_OUT_SHIFT_MODE`**
  - Default – **`ADI_LQ035Q1DH02_OUT_SHIFT_TB_RL`**

```
/*
** ADI_LQ035Q1DH02_OUT_SHIFT_MODE
**  - Enumerations of Output shift direction modes supported by Sharp LQ035Q1DH02 LCD driver
*/

typedef enum __AdiLq035q1dh02OutShiftMode
{

        /* Top Shifts to Bottom, Left Shifts to Right */
        ADI_LQ035Q1DH02_OUT_SHIFT_TB_LR,

        /* Top Shifts to Bottom, Right Shifts to Left */
        ADI_LQ035Q1DH02_OUT_SHIFT_TB_RL,

        /* Bottom Shifts to Top, Left Shifts to Right */
        ADI_LQ035Q1DH02_OUT_SHIFT_BT_LR,

        /* Bottom Shifts to Top, Right Shifts to Left */
        ADI_LQ035Q1DH02_OUT_SHIFT_BT_RL

} ADI_LQ035Q1DH02_OUT_SHIFT_MODE;
```

- **`ADI_LQ035Q1DH02_CMD_ENABLE_REVERSE_DISPLAY`**
  - Enables/Disables Reverse display
  - Value – true/false
  - Default – false (reversal disabled)
  - Note: ‘true’ to display all characters and graphics in reversal, ‘false’ to display in normal mode

- **`ADI_LQ035Q1DH02_CMD_ENABLE_256K_COLOR`**
  - Enables/Disables 256k color mode (OR) Disables/Enables 8-color mode
  - Value – true/false

   - Default – true (256k color mode enabled)
   - Note: 'true' to select 256k color, 'false' to select 8-color

- **`ADI_LQ035Q1DH02_CMD_SET_NUM_DUMMY_DOT_CLOCKS_PER_LINE`**
   - Sets number of dummy dot-clocks per line
   - Value – u8 (range between 2 and 64)
   - Default – 2

- **`ADI_LQ035Q1DH02_CMD_SET_NUM_DUMMY_LINES_PER_FRAME`**
   - Sets number of dummy lines per frame
   - Value – u8 (range between 1 and 240)
   - Default – 2

- **`ADI_LQ035Q1DH02_CMD_ENTER_SLEEP`**
   - Enters sleep mode by writing to LCD SHUT register
   - Value – NULL

- **`ADI_LQ035Q1DH02_CMD_EXIT_SLEEP`**
   - Enters sleep mode by writing to LCD SHUT register
   - Value – NULL

- **`ADI_LQ035Q1DH02_CMD_HARDWARE_RESET`**
   - Resets LCD display by pulling down LCD reset line
   - Value – enumeration of type ADI_FLAG_ID (Flag ID connected to the LCD Reset line)

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_LQ035Q1DH02_RESULT_PPI_NOT_OPEN`**
  - Results when client tries to submit buffers or enable dataflow when the underlying PPI device in closed state.
- **`ADI_LQ035Q1DH02_RESULT_CMD_NOT_SUPPORTED`**
  - Given command is not recognised or supported by the driver
- **`ADI_LQ035Q1DH02_RESULT_OUT_SHIFT_MODE_INVALID`**
  - Given Output Shift mode is invalid
- **`ADI_LQ035Q1DH02_RESULT_PIXELS_PER_LINE_INVALID`**
  - Given Valid Pixels per line count is invalid
- **`ADI_LQ035Q1DH02_RESULT_DUMMY_DOT_CLOCKS_PER_LINE_INVALID`**
  - Given Dummy Dot clocks per line count is invalid
- **`ADI_LQ035Q1DH02_RESULT_DUMMY_LINES_PER_FRAME_INVALID`**
  - Given Dummy Lines per Frame count is invalid
- **`ADI_LQ035Q1DH02_RESULT_SPI_DEVICE_INVALID`**
  - Given SPI Device number is invalid
- **`ADI_LQ035Q1DH02_RESULT_SPI_CS_INVALID`**
  - Given SPI chip select is invalid
- **`ADI_LQ035Q1DH02_RESULT_PPI_ALREADY_ALLOCATED`**
  - Results when client tries to open (allocate) more than one PPI / EPPI device for a LCD device instance

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.

- **`ADILQ035Q1DH02EntryPoint`**

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.

LCD Output Shift Direction

Default Value	: **`ADI_LQ035Q1DH02_OUT_SHIFT_TB_LR`** (Top shifts to Bottom, Left shifts to Right)
Possible values	: Enumeration of type **`ADI_LQ035Q1DH02_OUT_SHIFT_MODE`**
Command ID	: **`ADI_LQ035Q1DH02_CMD_SET_OUTPUT_SHIFT_DIRECTION`**

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
SPI Device Number	Processor dependent	**`ADI_LQ035Q1DH02_CMD_SET_SPI_DEVICE_NUMBER`**
SPI Chip select	Processor dependent	**`ADI_LQ035Q1DH02_CMD_SET_SPI_CHIP_SELECT`**
PPI Device Number	Processor dependent	**`ADI_LQ035Q1DH02_CMD_OPEN_PPI`**

Table 4 – Additional Required Settings

8. Hardware Considerations

The driver uses SPI driver to configure LCD hardware registers. Application must pass the SPI device number and chip select number connected to the LCD SPI port. 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 LQ035Q1DH02 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 three secondary interrupt memory** of size **`ADI_INT_SECONDARY_MEMORY`** – one for SPI Data interrupt handler, one for PPI/EPPI DMA Data interrupt handler and one for PPI/EPPI DMA error interrupt handler. Also, additional secondary interrupt memory must be provided when PPI/EPPI error reporting is enabled.

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 one DMA channel for PPI based device. For EPPI based device, application should allocate base memory + memory for number of DMA channels used by EPPI (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 three devices (one SHARP LQ035Q1DH02 LCD driver + one SPI device + one PPI/EPPI device) + memory for other devices used by the application. Each device requires memory of size **`ADI_DEV_DEVICE_MEMORY.`**

9.4. Typical usage of SHARP LQ035Q1DH02 LCD device driver

a. SHARP LQ035Q1DH02 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: Enable LCD Backlight if required (hardware specific)

Step 3: Configure driver with SPI device number to use

```
/* Example: Use SPI on Port G for ADSP-BF526 Ez-Kit */
adi_dev_Control (LcdDriverHandle, ADI_LQ035Q1DH02_CMD_SET_SPI_DEVICE_NUMBER, (void*) 1);
```

Step 4: Configure driver with SPI chip select number to use

```
/* Example: Use SPI chip select 1 for ADSP-BF526 Ez-Kit */
adi_dev_Control (LcdDriverHandle, ADI_LQ035Q1DH02_CMD_SET_SPI_CHIP_SELECT, (void*) 1);
```

Step 5: Select LCD data output shift direction (if required)

```
/* Example: Set Output Shift Direction as Top to Bottom and Left to Right */
adi_dev_Control (LcdDriverHandle,
                 ADI_LQ035Q1DH02_CMD_SET_OUTPUT_SHIFT_DIRECTION,
                 (void*) ADI_LQ035Q1DH02_OUT_SHIFT_TB_RL);
```

Step 6: Open PPI /EPPI device connected to the LCD data and frame sync port

```
/* Example: Open PPI device allocated for this LCD, Device 0 for ADSP-BF526 Ez-Kit */
adi_dev_Control (LcdDriverHandle, ADI_LQ035Q1DH02_CMD_OPEN_PPI, (void*) 0);
```

Step 7: Enable/Disable driver from configuring PPI / EPPI control register automatically

```
/* Example: Enable driver to configure PPI / EPPI Control register */
adi_dev_Control (LcdDriverHandle,
                 ADI_LQ035Q1DH02_CMD_ENABLE_AUTO_PPI_CONTROL_CONFIG,
                 (void*) true);
```

Step 8: Enable PPI / EPPI error reporting (only if required)

Step 9: 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 10: Queue outbound buffers to LCD (EPPI DMA) using adi_dev_Write( )

Step 11: Enable LCD Dataflow

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

Step 12: Respond to callbacks

c. Terminating SHARP LQ035Q1DH02 LCD driver

Step 13: Disable LCD dataflow

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

Step 14: Disable PPI / EPPI error reporting (if already enabled)

Step 15: Terminate LCD driver with adi_dev_Close( )

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