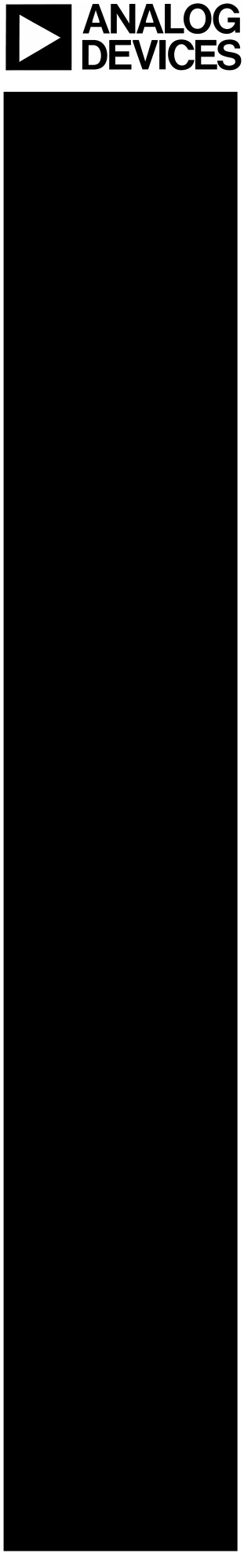

ADI_EPPI
DEVICE DRIVER

DATE: JANUARY 25, 2007

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
Jan 25, 2007	Initial release
May 14, 2007	Added configuration table examples for ADSP-BF548 Ez-Kit Lite

Table 1 – Revision History

1. Overview

This document describes use of the Enhanced Parallel Peripheral Interface (EPPI) device driver.

EPPI is a half-duplex, bidirectional port accommodating up to 24 bits of data, and has a dedicated clock pin and three frame sync (FS) pins. A dedicated DMA channel is also connected to the EPPI and can also be setup in different configurations. The EPPI supports direct connection to LCD panels, parallel A/D and D/A converters, video encoders and decoders, CMOS sensors and other general purpose peripherals.

The EPPI device driver is not interrupt driven, but uses the Direct Memory Access (DMA) services, as explained later in this document.

The EPPI device 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/eppi/adi_eppi.h>
  - This file contains all definitions, function prototypes etc. specific to the EPPI device

2.2. Source Files

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

- < Blackfin/lib/src/drivers/eppi/adi_eppi.c>
  - This file contains all the source code for the EPPI device driver. All source code is written in ‘C’. There are no assembly level functions in this driver.

3. Lower Level Drivers

The EPPI device driver does not use any lower level device drivers.

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

Each EPPI device requires an additional (driver) memory of size ADI_DEV_DEVICE_MEMORY

4.1. Interrupts

For each EPPI driver that is opened, only one interrupt is used: the error interrupt.

Unless overridden with the appropriate “SetIVG” commands, the error interrupt for the EPPI device driver uses the default, power-up, Interrupt Vector Group (IVG) mapping for the specific processor.

The EPPI device driver hooks or unhooks the error interrupt handler when the client calls the ‘adi_dev_Control()’ function, with the command: ADI_DEV_CMD_SET_ERROR_REPORTING. If the command is accompanied by an argument of TRUE, the error interrupt is enabled, and the error interrupt handler is hooked into the IVG chain. If the command is accompanied by an argument of FALSE, the error interrupt is disabled, and the interrupt handler is unhooked from the IVG chain. When the client closes the driver by calling ‘adi_dev_close()’, the error interrupt, if enabled and hooked, is automatically disabled and unhooked.

This driver requires two additional memory of size ADI_INT_SECONDARY_MEMORY for each DMA channel – one for DMA Data interrupt handler and one for DMA error interrupt handler. One additional memory of above size must be provided for each EPPI device when the client decides to enable EPPI error reporting.

4.2. DMA

This section will explain how to use the EPPI device driver in conjunction with the Direct Memory Access (DMA) services, to pass data to and from peripheral devices.

One DMA channel should be allocated for each EPPI driver that is opened. If the processor has three EPPI ports, and they are used simultaneously, then three DMA channels should be allocated and initialized. Some EPPI port can be configured to use a second DMA channel, provided that the second DMA channel is not used by the any other EPPI port. In that case, two DMA channels should be allocated and initialized.

Each DMA channel requires an additional memory of size ADI_DMA_CHANNEL_MEMORY

4.3. Timers

The EPPI device driver does not use timer service.

4.4. Real-Time Clock

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

4.5. Programmable Flags

The EPPI device driver does not use flag services.

4.6. Pins

The EPPI has a dedicated clock pin, three frame sync pins, and 8 to 24 dedicated data pins. On processors where pin multiplexing is used, the programmable flag (PF) pins can be reconfigured to enable EPPI data pins. The EPPI can be supplied with an external clock, or the clock can be generated internally and supplied to external devices.

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_INBOUND	Supports the reception of data in through the device.
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

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

Commands to configure EPPI registers

- ADI_EPPI_CMD_SET_CONTROL_REG
  - Sets EPPI control Register
  - Value – u32
- ADI_EPPI_CMD_SET_LINES_PER_FRAME
  - Sets Lines per Frame Register
  - Value – u16
- ADI_EPPI_CMD_SET_SAMPLES_PER_LINE
  - Sets EPPI Samples Per Line Register
  - Value – u16
- ADI_EPPI_CMD_SET_VERTICAL_DELAY
  - Sets EPPI Vertical Delay Register
  - Value – u16
- ADI_EPPI_CMD_SET_VERTICAL_TX_COUNT
  - Sets EPPI Vertical Transfer Count Register
  - Value – u16
- ADI_EPPI_CMD_SET_HORIZONTAL_DELAY
  - Sets EPPI Horizontal Delay Register
  - Value – u16
- ADI_EPPI_CMD_SET_HORIZONTAL_TX_COUNT
  - Sets EPPI Horizontal Transfer Count Register
  - Value – u16
- ADI_EPPI_CMD_SET_CLOCK_FREQ
  - Sets EPPI Clock Frequency
  - Value – u32
- ADI_EPPI_CMD_SET_CLOCK_DIV
  - Sets EPPI Clock Divide Register
  - Value – u16

- ADI_EPPI_CMD_SET_FS1_WIDTH
  - Sets EPPI Frame Sync 1 Width Register
  - Value – u32
- ADI_EPPI_CMD_SET_FS2_WIDTH
  - Sets EPPI Frame Sync 2 Width Register
  - Value – u32
- ADI_EPPI_CMD_SET_FS1_PERIOD
  - Sets EPPI Frame Sync 1 Period Register
  - Value – u32
- ADI_EPPI_CMD_SET_FS2_PERIOD
  - Sets EPPI Frame Sync 2 Period Register
  - Value – u32
- ADI_EPPI_CMD_SET_CLIPPING
  - Sets EPPI Clipping Register
  - Value – u32

Commands to configure individual bits/fields of EPPI registers

- ADI_EPPI_CMD_SET_HORIZONTAL_BLANK_PER_LINE
  - Sets Horizontal Blanking Samples per Line
  - Value – u16
- ADI_EPPI_CMD_SET_VERTICAL_BLANK_PER_LINE
  - Sets Vertical Blanking Samples per Line
  - Value – u16

Commands to configure individual bits/fields of EPPI control register

- ADI_EPPI_CMD_SET_PORT_DIRECTION
  - Sets EPPI Direction
  - Value – u8
- ADI_EPPI_CMD_SET_TRANSFER_TYPE
  - Sets EPPI Transfer type
  - Value – u8
- ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG
  - Sets EPPI Frame sync configuration
  - Value – u8
- ADI_EPPI_CMD_SET_FIELD_SELECT_TRIGGER
  - Sets EPPI Field select/trigger
  - Value – u8
- ADI_EPPI_CMD_SET_ITU_TYPE
  - Sets EPPI ITU Type
  - Value – u8
- ADI_EPPI_CMD_ENABLE_BLANKGEN
  - Enable/Disable EPPI Blank/preamble generation
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN
  - Enable/Disable EPPI Internal clock generation
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_ENABLE_INTERNAL_FS_GEN
  - Enable/Disable EPPI Internal Frame Sync generation
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_SET_CLOCK_POLARITY
  - Sets EPPI clock polarity
  - Value – u8
- ADI_EPPI_CMD_SET_FRAME_SYNC_POLARITY
  - Sets EPPI Frame sync polarity
  - Value – u8

- ADI_EPPI_CMD_SET_DATA_LENGTH
  - Sets EPPI Data length
  - Value – u8
- ADI_EPPI_CMD_SET_SKIP_ENABLE
  - Enable EPPI data skipping
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_SET_SKIP_EVEN_ODD
  - Sets EPPI to Skip even or odd elements
  - Value – u8
- ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE
  - Enable EPPI DMA packing/unpacking
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_SET_SWAP_ENABLE
  - Enable Data swapping
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_SET_SIGN_EXT_SPLIT16
  - Sets EPPI sign extension/split 16
  - Value – u8
- ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD
  - Sets EPPI split even/odd samples
  - Value – u8
- ADI_EPPI_CMD_ENABLE_SUBSPLIT_ODD
  - Enable sub-split odd samples
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_SET_DMA_CHANNEL_MODE
  - Sets EPPI DMA channel mode
  - Value – u8
- ADI_EPPI_CMD_ENABLE_RGB_FORMATING
  - Enable RGB Formatting
  - Value – TRUE/FALSE
- ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK
  - Sets EPPI FIFO regular watermark
  - Value – u8
- ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK
  - Sets EPPI FIFO urgent watermark
  - Value – u8

Commands to configure individual bits/fields of vertical blanking register

- ADI_EPPI_CMD_SET_FIELD1_PRE_ACTIVE_DATA_VBLANK
  - Sets number of lines of vertical blanking before Field 1 active data
  - Value – u8
- ADI_EPPI_CMD_SET_FIELD1_POST_ACTIVE_DATA_VBLANK
  - Sets number of lines of vertical blanking after Field 1 active data
  - Value – u8
- ADI_EPPI_CMD_SET_FIELD2_PRE_ACTIVE_DATA_VBLANK
  - Sets number of lines of vertical blanking before Field 2 active data
  - Value – u8
- ADI_EPPI_CMD_SET_FIELD2_POST_ACTIVE_DATA_VBLANK
  - Sets number of lines of vertical blanking after Field 2 active data
  - Value – u8

Commands to configure individual bits/fields of Lines of active video per frame register

- ADI_EPPI_CMD_SET_FIELD1_ACTIVE_DATA_LINES
  - Sets number of lines of active data in Field 1
  - Value – u8
- ADI_EPPI_CMD_SET_FIELD2_ACTIVE_DATA_LINES
  - Sets number of lines of active data in Field 2
  - Value – u8

Commands to configure individual bits/fields of clipping register

- ADI_EPPI_CMD_SET_CHROMA_LOW_CLIP_LIMIT
  - Sets Lower clipping limit for odd bytes (Chroma)
  - Value – u8
- ADI_EPPI_CMD_SET_CHROMA_HIGH_CLIP_LIMIT
  - Sets Higher clipping limit for odd bytes (Chroma)
  - Value – u8
- ADI_EPPI_CMD_SET_LUMA_LOW_CLIP_LIMIT
  - Sets Lower clipping limit for even bytes (Luma)
  - Value – u8
- ADI_EPPI_CMD_SET_LUMA_HIGH_CLIP_LIMIT
  - Sets Higher clipping limit for even bytes (Luma)
  - Value – u8

Commands to sense EPPI register bits/fields

- ADI_EPPI_CMD_GET_FIELD_RECEIVED_STATUS
  - Gets Field Received status (FLD bit value in EPPI_STATUS register)
  - Value – u8*
- ADI_EPPI_CMD_GET_CONTROL_REG
  - Gets present control register value
  - Value – u32*

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

- ADI_EPPI_EVENT_CHROMA_FIFO_ERROR
    - Indicates that Chroma FIFO overflow/underflow error has occurred
    - Value – NULL
- ADI_EPPI_EVENT_LUMA_FIFO_ERROR
    - Indicates that Luma FIFO overflow/underflow error has occurred
    - Value – NULL
- ADI_EPPI_EVENT_LINE_TRACK_OVERFLOW_ERROR
    - Indicates that Line track overflow error has occurred
    - Value – NULL
- ADI_EPPI_EVENT_LINE_TRACK_UNDERFLOW_ERROR
    - Indicates that Line track underflow error has occurred
    - Value – NULL
- ADI_EPPI_EVENT_FRAME_TRACK_OVERFLOW_ERROR
    - Indicates that Frame track overflow error has occurred
    - Value – NULL
- ADI_EPPI_EVENT_FRAME_TRACK_UNDERFLOW_ERROR
    - Indicates that Frame track underflow error has occurred
    - Value – NULL
- ADI_EPPI_EVENT_PREAMBLE_ERROR_NOT_CORRECTED
    - Indicates that a preamble error has been detected, but not corrected
    - Value – NULL
- ADI_EPPI_EVENT_PREAMBLE_ERROR
    - Indicates that a preamble error has been detected
    - Value – NULL

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_EPPI_RESULT_PORT_SHARING_ERROR
  - Occurs when the selected EPPI port (hardware) is unavailable due to other EPPI device(s) operating state and port usage
- ADI_EPPI_RESULT_DMA_SHARING_ERROR
  - Occurs when the selected EPPI device is configured to use a shared/extensible DMA and the shareable DMA channel is already in use
- ADI_EPPI_RESULT_PORT_SHARING_ERROR
  - Occurs when client provides invalid Clock Divide value
- ADI_EPPI_RESULT_PORT_SHARING_ERROR
  - Occurs when client provides invalid EPPI clock frequency value

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.

- ADIEPPIEntryPoint

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
Control Register	0	Application dependent	ADI_EPPI_CMD_SET_CONTROL_REG
Lines per Frame Register	0	0xFFFF to 1	ADI_EPPI_CMD_SET_LINES_PER_FRAME
Samples Per Line Register	0	0xFFFF to 1	ADI_EPPI_CMD_SET_SAMPLES_PER_LINE
Vertical Delay Register	0	0xFFFF to 0	ADI_EPPI_CMD_SET_VERTICAL_DELAY
Vertical Transfer Count Register	0	0xFFFF to 0	ADI_EPPI_CMD_SET_VERTICAL_TX_COUNT
Horizontal Delay Register	0	0xFFFF to 0	ADI_EPPI_CMD_SET_HORIZONTAL_DELAY
Horizontal Transfer Count Register	0	0xFFFF to 0	ADI_EPPI_CMD_SET_HORIZONTAL_TX_COUNT
Clock Divide Register	0	0xFFFE to 0	ADI_EPPI_CMD_SET_CLOCK_DIV
Frame Sync 1 Width Register	0	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS1_WIDTH
Frame Sync 2 Width Register	0	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS2_WIDTH
Frame Sync 1 Period Register	0	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS1_PERIOD
Frame Sync 2 Period Register	0	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS2_PERIOD
Clipping Register	0xFF00FF00	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_CLIPPING

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_EPPI_CMD_SET_CONTROL_REG
Lines per Frame Register	0xFFFF to 1	ADI_EPPI_CMD_SET_LINES_PER_FRAME
Samples Per Line Register	0xFFFF to 1	ADI_EPPI_CMD_SET_SAMPLES_PER_LINE
Vertical Delay Register	0xFFFF to 0	ADI_EPPI_CMD_SET_VERTICAL_DELAY
Vertical Transfer Count Register	0xFFFF to 0	ADI_EPPI_CMD_SET_VERTICAL_TX_COUNT
Horizontal Delay Register	0xFFFF to 0	ADI_EPPI_CMD_SET_HORIZONTAL_DELAY
Horizontal Transfer Count Register	0xFFFF to 0	ADI_EPPI_CMD_SET_HORIZONTAL_TX_COUNT
Clock Divide Register	0xFFFE to 0	ADI_EPPI_CMD_SET_CLOCK_DIV
Frame Sync 1 Width Register	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS1_WIDTH
Frame Sync 2 Width Register	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS2_WIDTH
Frame Sync 1 Period Register	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS1_PERIOD
Frame Sync 2 Period Register	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_FS2_PERIOD
Clipping Register	0xFFFFFFFF to 0	ADI_EPPI_CMD_SET_CLIPPING

Table 5 – Additional Required Settings

7. Hardware Considerations

On processors where pin multiplexing is used, depending on the control register value, the driver automatically reconfigures programmable flag (PF) pins to enable EPPI Clock, Frame Sync and EPPI data pins for desired data width (8/10/12/14/16/18/24-bits). The user must use caution to insure that the EPPI does not use any PF pins used by any other general purpose I/O device, and vice-versa.

8. Using EPPI Driver in Applications

This section explains how to use EPPI 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 # EPPI devices used + 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 a secondary interrupt memory of size **ADI_INT_SECONDARY_MEMORY** for each EPPI DMA channel Data interrupt handler and for each DMA channel error interrupt handler. Also, additional secondary interrupt memory must be provided in case the user decides to enable EPPI error reporting.

8.3. Typical usage of EPPI device driver

a. EPPI (driver) initialization

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

Step 2: Enable EPPI error reporting (if required)

b. Initializing and controlling EPPI (hardware)

Step 3: Configure EPPI device registers to specific operating mode (refer section 9 for configuration examples)

c. Submitting buffers

Step 4: Queue buffers to EPPI DMA using adi_dev_Read( ) or adi_dev_Write( ), depending on data direction

Step 5: Respond to callbacks

c. Terminating EPPI driver

Step 6: Disable EPPI error reporting (if already enabled)

Step 7: Terminate EPPI driver with adi_dev_Terminate( )

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

9. EPPI Device Configuration table examples for ADSP-BF548 Ez-Kit Lite

9.1. IUT–R 656 NTSC Interlaced video out

```
ADI_DEV_CMD_VALUE_PAIR      Eppi_ITUR656_NTSCi_VideoOut[ ] =
{
        { ADI_EPPI_CMD_SET_PORT_DIRECTION,              (void *)1       },  /* EPPI in transmit mode                    */
        { ADI_EPPI_CMD_SET_TRANSFER_TYPE,               (void *)3       },  /* GP Transfer mode                         */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG,           (void *)0       },  /* 0 FS mode. Frame Syncs not driven        */
        { ADI_EPPI_CMD_SET_ITU_TYPE,                    (void *)0       },  /* ITU Type - Interlaced                    */
        { ADI_EPPI_CMD_ENABLE_BLANKGEN,                 (void *)FALSE   },  /* Disable BLANKGEN                         */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN,       (void *)FALSE   },  /* Externally generated Clock               */
        { ADI_EPPI_CMD_SET_CLOCK_POLARITY,              (void *)2       },  /* Drive data on falling edge               */
        { ADI_EPPI_CMD_SET_DATA_LENGTH,                 (void *)0       },  /* 8 bit out                                */
        { ADI_EPPI_CMD_SET_SKIP_ENABLE,                 (void *)FALSE   },  /* Disable skipping                         */
        { ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE,          (void *)TRUE    },  /* DMA unpacking enabled                    */
        { ADI_EPPI_CMD_SET_SWAP_ENABLE,                 (void *)FALSE   },  /* Swapping disabled                        */
        { ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD,              (void *)FALSE   },  /* Splitting disabled                       */
        { ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK,      (void *)1       },  /* Regular watermark                        */
        { ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK,       (void *)3       },  /* Urgent watermark                         */
        { ADI_EPPI_CMD_SET_SAMPLES_PER_LINE,            (void *)1716    },  /* Samples per Line                         */
        { ADI_EPPI_CMD_SET_LINES_PER_FRAME,             (void *)525     },  /* Lines per Frame                          */
        { ADI_DEV_CMD_END,                              NULL            }   /* Terminate this configuration table       */
};
```

9.2. IUT–R 656 PAL Interlaced video out

```
ADI_DEV_CMD_VALUE_PAIR      Eppi_ITUR656_PALi_VideoOut[ ] =
{
        { ADI_EPPI_CMD_SET_PORT_DIRECTION,          (void *)1       },  /* EPPI in transmit mode                    */
        { ADI_EPPI_CMD_SET_TRANSFER_TYPE,           (void *)3       },  /* GP Transfer mode                         */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG,       (void *)0       },  /* 0 FS mode. Frame Syncs not driven        */
        { ADI_EPPI_CMD_SET_ITU_TYPE,                (void *)0       },  /* ITU Type - Interlaced                    */
        { ADI_EPPI_CMD_ENABLE_BLANKGEN,             (void *)FALSE   },  /* Disable BLANKGEN                         */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN,   (void *)FALSE   },  /* Externally generated Clock               */
        { ADI_EPPI_CMD_SET_CLOCK_POLARITY,          (void *)2       },  /* Drive data on falling edge               */
        { ADI_EPPI_CMD_SET_DATA_LENGTH,             (void *)0       },  /* 8 bit out                                */
        { ADI_EPPI_CMD_SET_SKIP_ENABLE,             (void *)FALSE   },  /* Disable skipping                         */
        { ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE,      (void *)TRUE    },  /* DMA unpacking enabled                    */
        { ADI_EPPI_CMD_SET_SWAP_ENABLE,             (void *)FALSE   },  /* Swapping disabled                        */
        { ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD,          (void *)FALSE   },  /* Splitting disabled                       */
        { ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK,  (void *)1       },  /* Regular watermark                        */
        { ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK,   (void *)3       },  /* Urgent watermark                         */
        { ADI_EPPI_CMD_SET_SAMPLES_PER_LINE,        (void *)1728    },  /* Samples per Line                         */
        { ADI_EPPI_CMD_SET_LINES_PER_FRAME,         (void *)625     },  /* Lines per Frame                          */
        { ADI_DEV_CMD_END,                          NULL            }   /* Terminate this configuration table       */
};
```

9.3. IUT–R 656 NTSC Interlaced – Active video out with Internal Blank generation

```
ADI_DEV_CMD_VALUE_PAIR      Eppi_ITUR656_NTSCi_ActiveVideoOut[ ] =
{
        { ADI_EPPI_CMD_SET_PORT_DIRECTION,                  (void *)1       },  /* EPPI in transmit mode                                          */
        { ADI_EPPI_CMD_SET_TRANSFER_TYPE,                   (void *)3       },  /* GP Transfer mode                                               */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG,               (void *)0       },  /* 0 FS mode. Frame Syncs not driven                              */
        { ADI_EPPI_CMD_SET_ITU_TYPE,                        (void *)0       },  /* ITU Type - Interlaced                                          */
        { ADI_EPPI_CMD_ENABLE_BLANKGEN,                     (void *)TRUE    },  /* Enable BLANKGEN                                                */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN,           (void *)FALSE   },  /* Externally generated Clock                                     */
        { ADI_EPPI_CMD_SET_CLOCK_POLARITY,                  (void *)2       },  /* Drive data on falling edge                                     */
        { ADI_EPPI_CMD_SET_DATA_LENGTH,                     (void *)0       },  /* 8 bit out                                                      */
        { ADI_EPPI_CMD_SET_SKIP_ENABLE,                     (void *)FALSE   },  /* Disable skipping                                               */
        { ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE,              (void *)TRUE    },  /* DMA unpacking enabled                                          */
        { ADI_EPPI_CMD_SET_SWAP_ENABLE,                     (void *)FALSE   },  /* Swapping disabled                                              */
        { ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD,                  (void *)FALSE   },  /* Splitting disabled                                             */
        { ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK,          (void *)1       },  /* Regular watermark                                              */
        { ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK,           (void *)3       },  /* Urgent watermark                                               */
        { ADI_EPPI_CMD_SET_SAMPLES_PER_LINE,                (void *)1716    },  /* Samples per Line                                               */
        { ADI_EPPI_CMD_SET_LINES_PER_FRAME,                 (void *)525     },  /* Lines per Frame                                                */
        { ADI_EPPI_CMD_SET_FS1_WIDTH,                       (void *)268     },  /* Horizontal blanking samples per line                           */
        { ADI_EPPI_CMD_SET_FIELD1_PRE_ACTIVE_DATA_VBLANK,   (void *)17      },  /* Vertical blank before start of Field 1 Active Data             */
        { ADI_EPPI_CMD_SET_FIELD1_POST_ACTIVE_DATA_VBLANK,  (void *)2       },  /* Vertical blank after the end of Field 1 Active Data            */
        { ADI_EPPI_CMD_SET_FIELD2_PRE_ACTIVE_DATA_VBLANK,   (void *)17      },  /* Vertical blank before start of Field 2 Active Data             */
        { ADI_EPPI_CMD_SET_FIELD2_POST_ACTIVE_DATA_VBLANK,  (void *)3       },  /* Vertical blank after the end of Field 2 Active Data            */
        { ADI_EPPI_CMD_SET_FS1_PERIOD,                      (void *)1440    },  /* Active Video samples per line or Vertical blanking samples per line */
        { ADI_EPPI_CMD_SET_FIELD1_ACTIVE_DATA_LINES,        (void *)243     },  /* # of Active data lines in Field 1                              */
        { ADI_EPPI_CMD_SET_FIELD2_ACTIVE_DATA_LINES,        (void *)243     },  /* # of Active data lines in Field 2                              */
        { ADI_DEV_CMD_END,                                  NULL            }   /* Terminate this configuration table                             */
};
```

9.4. IUT–R 656 PAL Interlaced – Active video out with Internal Blank generation

```
ADI_DEV_CMD_VALUE_PAIR      Eppi_ITUR656_PALi_ActiveVideoOut[ ] =
{
        { ADI_EPPI_CMD_SET_PORT_DIRECTION,                  (void *)1       },  /* EPPI in transmit mode                                          */
        { ADI_EPPI_CMD_SET_TRANSFER_TYPE,                   (void *)3       },  /* GP Transfer mode                                               */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG,               (void *)0       },  /* 0 FS mode. Frame Syncs not driven                              */
        { ADI_EPPI_CMD_SET_ITU_TYPE,                        (void *)0       },  /* ITU Type - Interlaced                                          */
        { ADI_EPPI_CMD_ENABLE_BLANKGEN,                     (void *)TRUE    },  /* Enable BLANKGEN                                                */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN,           (void *)FALSE   },  /* Externally generated Clock                                     */
        { ADI_EPPI_CMD_SET_CLOCK_POLARITY,                  (void *)2       },  /* Drive data on falling edge                                     */
        { ADI_EPPI_CMD_SET_DATA_LENGTH,                     (void *)0       },  /* 8 bit out                                                      */
        { ADI_EPPI_CMD_SET_SKIP_ENABLE,                     (void *)FALSE   },  /* Disable skipping                                               */
        { ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE,              (void *)TRUE    },  /* DMA unpacking enabled                                          */
        { ADI_EPPI_CMD_SET_SWAP_ENABLE,                     (void *)FALSE   },  /* Swapping disabled                                              */
        { ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD,                  (void *)FALSE   },  /* Splitting disabled                                             */
        { ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK,          (void *)1       },  /* Regular watermark                                              */
        { ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK,           (void *)3       },  /* Urgent watermark                                               */
        { ADI_EPPI_CMD_SET_SAMPLES_PER_LINE,                (void *)1728    },  /* Samples per Line                                               */
        { ADI_EPPI_CMD_SET_LINES_PER_FRAME,                 (void *)625     },  /* Lines per Frame                                                */
        { ADI_EPPI_CMD_SET_FS1_WIDTH,                       (void *)280     },  /* Horizontal blanking samples per line                           */
        { ADI_EPPI_CMD_SET_FIELD1_PRE_ACTIVE_DATA_VBLANK,   (void *)22      },  /* Vertical blank before start of Field 1 Active Data             */
        { ADI_EPPI_CMD_SET_FIELD1_POST_ACTIVE_DATA_VBLANK,  (void *)2       },  /* Vertical blank after the end of Field 1 Active Data            */
        { ADI_EPPI_CMD_SET_FIELD2_PRE_ACTIVE_DATA_VBLANK,   (void *)23      },  /* Vertical blank before start of Field 2 Active Data             */
        { ADI_EPPI_CMD_SET_FIELD2_POST_ACTIVE_DATA_VBLANK,  (void *)2       },  /* Vertical blank after the end of Field 2 Active Data            */
        { ADI_EPPI_CMD_SET_FS1_PERIOD,                      (void *)1440    },  /* Active Video samples per line or Vertical blanking samples per line */
        { ADI_EPPI_CMD_SET_FIELD1_ACTIVE_DATA_LINES,        (void *)288     },  /* # of Active data lines in Field 1                              */
        { ADI_EPPI_CMD_SET_FIELD2_ACTIVE_DATA_LINES,        (void *)288     },  /* # of Active data lines in Field 2                              */
        { ADI_DEV_CMD_END,                                  NULL            }   /* Terminate this configuration table                             */
};
```

9.5. RGB888 Video out to Sharp LQ043T1DG01 LCD on ADSP-BF548 Ez-Kit Lite

**** Note: Sharp LQ043T1DG01 LCD driver configures EPPI Clock divide, EPPI windowing and Frame Sync/Blank generation registers with LCD specific values.**

```
ADI_DEV_CMD_VALUE_PAIR      Eppi_RGB888Out_SharpLQ043T1DG01 [ ] =
{
        { ADI_EPPI_CMD_SET_PORT_DIRECTION,              (void *)1       },  /* EPPI in transmit mode                */
        { ADI_EPPI_CMD_SET_TRANSFER_TYPE,               (void *)3       },  /* GP Transfer mode                     */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG,           (void *)2       },  /* GP2 FS mode.                         */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN,       (void *)TRUE    },  /* Internally generated Clock           */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_FS_GEN,          (void *)TRUE    },  /* Internally generated Frame Sync      */
        { ADI_EPPI_CMD_SET_CLOCK_POLARITY,              (void *)1       },  /* Drive data on Raising edge           */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_POLARITY,         (void *)3       },  /* FS1 & FS2 are active low             */
        { ADI_EPPI_CMD_SET_DATA_LENGTH,                 (void *)6       },  /* 24 bit out                           */
        { ADI_EPPI_CMD_SET_SKIP_ENABLE,                 (void *)FALSE   },  /* Disable skipping                     */
        { ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE,          (void *)TRUE    },  /* DMA unpacking enabled                */
        { ADI_EPPI_CMD_SET_SWAP_ENABLE,                 (void *)FALSE   },  /* Swapping disabled                    */
        { ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD,              (void *)FALSE   },  /* Splitting disabled                   */
        { ADI_EPPI_CMD_ENABLE_RGB_FORMATTING,           (void *)FALSE   },  /* Disable RGB formatting               */
        { ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK,      (void *)1       },  /* Regular watermark                    */
        { ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK,       (void *)3       },  /* Urgent watermark                     */
        { ADI_DEV_CMD_END,                              NULL            }   /* Terminate this configuration table   */
};
```

9.6. RGB666 Video out to Sharp LQ043T1DG01 LCD on ADSP-BF548 Ez-Kit Lite

**** Note: Sharp LQ043T1DG01 LCD driver configures EPPI Clock divide, EPPI windowing and Frame Sync/Blank generation registers with LCD specific values.**

```
ADI_DEV_CMD_VALUE_PAIR      Eppi_RGB666Out_SharpLQ043T1DG01 [ ] =
{
        { ADI_EPPI_CMD_SET_PORT_DIRECTION,              (void *)1       },  /* EPPI in transmit mode                */
        { ADI_EPPI_CMD_SET_TRANSFER_TYPE,               (void *)3       },  /* GP Transfer mode                     */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_CONFIG,           (void *)2       },  /* GP2 FS mode.                         */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_CLOCK_GEN,       (void *)TRUE    },  /* Internally generated Clock           */
        { ADI_EPPI_CMD_ENABLE_INTERNAL_FS_GEN,          (void *)TRUE    },  /* Internally generated Frame Sync      */
        { ADI_EPPI_CMD_SET_CLOCK_POLARITY,              (void *)1       },  /* Drive data on Raising edge           */
        { ADI_EPPI_CMD_SET_FRAME_SYNC_POLARITY,         (void *)3       },  /* FS1 & FS2 are active low             */
        { ADI_EPPI_CMD_SET_DATA_LENGTH,                 (void *)5       },  /* 18 bit out                           */
        { ADI_EPPI_CMD_SET_SKIP_ENABLE,                 (void *)FALSE   },  /* Disable skipping                     */
        { ADI_EPPI_CMD_SET_PACK_UNPACK_ENABLE,          (void *)TRUE    },  /* DMA unpacking enabled                */
        { ADI_EPPI_CMD_SET_SWAP_ENABLE,                 (void *)FALSE   },  /* Swapping disabled                    */
        { ADI_EPPI_CMD_SET_SPLIT_EVEN_ODD,              (void *)FALSE   },  /* Splitting disabled                   */
        { ADI_EPPI_CMD_ENABLE_RGB_FORMATTING,           (void *)TRUE    },  /* Enable RGB formatting                */
        { ADI_EPPI_CMD_SET_FIFO_REGULAR_WATERMARK,      (void *)1       },  /* Regular watermark                    */
        { ADI_EPPI_CMD_SET_FIFO_URGENT_WATERMARK,       (void *)3       },  /* Urgent watermark                     */
        { ADI_DEV_CMD_END,                              NULL            }   /* Terminate this configuration table   */
};
```