ANALOG DEVICES

ADI_PIXC DEVICE DRIVER

DATE: JULY 20, 2011

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
May 15, 2007	Initial release
July 20, 2011	Corrected the code snippet for input buffer table submission

Table 1 – Revision History

1. Overview

This document describes use of the Pixel Compositor (PIXC) device driver.

The Pixel Compositor provides data overlay, transparent color, and color space conversion support for active (TFT) flat-panel digital color/monochrome LCD displays or analog NTSC/PAL video output. The color space conversion and text/graphic overlay capabilities, along with visual effect controls, such as transparency control, shorten the processing time on an image data stream, reduce power consumption and save system board space by removing the need for external glue logic

PIXC requires 3 DMA channels: one for the (Input) image data, one for the overlay data and the third for storing the results back to L3, L2 or L1 memory. Overlay DMA channel is used only when the client enables the Pixc overlay option. DMA channel handling and buffer submission techniques are explained later in this document.

The PIXC driver contains inbuilt lookup tables for all hardware supported color conversion modes. Application has to pass a PIXC driver specific command with pointer to a data structure holding all frame information. Additionally, PIXC driver can configure the color conversion registers to support color conversion of ITU-R 656 type YUV422 frame to/from RGB888 frame. Refer to section 10.2 (page 25) for examples.

The PIXC 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/pixc/adi_pixc.h>
  - This file contains all definitions, function prototypes etc. specific to the PIXC 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/pixc/adi_pixc.c>
  - This file contains all the source code for the PIXC device driver. All source code is written in 'C'. There are no assembly level functions in this driver.

3. Lower Level Drivers

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

PIXC device requires an additional (driver) memory of size ADI_DEV_DEVICE_MEMORY

4.1. Interrupts

PIXC 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 ADI_INT_SECONDARY_MEMORY size must be provided when the client decides to enable PIXC Status reporting

4.2. DMA

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

By default, PIXC driver requires two DMA channels – one for Image input and one for data output. Overlay DMA channel usage depends on PIXC operating mode.

Each DMA channel requires an additional memory of size ADI_DMA_CHANNEL_MEMORY

4.3. Timers

The PIXC device driver does not use timer service.

4.4. Real-Time Clock

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

4.5. Programmable Flags

The PIXC device driver does not use flag services.

4.6. Pins

The PIXC device driver does not use any pins.

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_BIDIRECTIONAL	PIXC only supports bi-directional dataflow.

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
- ADI_DEV_BUFFER_PAIR
  - Buffer pair
  - 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_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)
- ADI_DEV_CMD_SET_INBOUND_DMA_CHAIN_CHANNEL_ID
    - Sets DMA channel IDs for inbound DMA chain
    - Value – ADI_DEV_DMA_ACCESS * (DMA chain data access structure)
- ADI_DEV_CMD_GET_INBOUND_DMA_CHAIN_CHANNEL_ID
    - Gets DMA channel IDs of inbound DMA chain
    - Value – ADI_DEV_DMA_ACCESS * (DMA chain data access structure)
- ADI_DEV_CMD_SET_OUTBOUND_DMA_CHAIN_CHANNEL_ID
    - Device manager closes a DMA channel used by a peripheral
    - Value – ADI_DEV_DMA_ACCESS * (DMA chain data access structure)
- ADI_DEV_CMD_GET_OUTBOUND_DMA_CHAIN_CHANNEL_ID
    - Device manager closes a DMA channel used by a peripheral
    - Value – ADI_DEV_DMA_ACCESS * (DMA chain data access structure)
- ADI_DEV_CMD_GET_INBOUND_DMA_CHAIN_CURRENT_ADDRESS
    - Device manager closes a DMA channel used by a peripheral
    - Value – ADI_DEV_DMA_ACCESS * (DMA chain data access structure)
- ADI_DEV_CMD_GET_OUTBOUND_DMA_CHAIN_CURRENT_ADDRESS
    - Device manager closes a DMA channel used by a peripheral
    - Value – ADI_DEV_DMA_ACCESS * (DMA chain data access structure)

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 PIXC registers

- ADI_PIXC_CMD_SET_COLOR_CONVERSION_MODE
    - Sets PIXC registers in selected color conversion mode. Refer Table 7 – page 24 for list of supported color conversion modes. Refer section 10.2 – page 25 for example code.
    - Value – ADI_PIXC_CONVERSION_MODE * (Address of PIXC color conversion mode structure)
- ADI_PIXC_CMD_ENABLE_ITUR656_SUPPORT
    - Sets PIXC registers to support color conversion of ITU-R 656 (UYVY) type YUV422 frame to/from RGB888 frame (Refer section 10.3 – page 25 for more information)
    - Value – TRUE/FALSE, Default=FALSE
- ADI_PIXC_CMD_SET_CONTROL_REG

  - Sets the PIXC control register
  - Value – u16
- ADI_PIXC_CMD_SET_PIXELS_PER_LINE
  - Sets Pixels Per Line register
  - Value – u16 (range:0xFFFF - 0x0001)
- ADI_PIXC_CMD_SET_LINES_PER_FRAME
  - Sets Lines per Frame register
  - Value – u16 (range:0xFFFF - 0x0001)
- ADI_PIXC_CMD_SET_OVERLAY_A_HSTART
  - Sets Overlay A Horizontal Start register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_A_HEND
  - Sets Overlay A Horizontal End register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_A_VSTART
  - Sets Overlay A Vertical Start register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_A_VEND
  - Sets Overlay A Vertical End register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_A_TRANSPARENCY
  - Sets Overlay A Transparency value register
  - Value – u16 (range:0xF - 0x0)
- ADI_PIXC_CMD_SET_OVERLAY_B_HSTART
  - Sets Overlay B Horizontal Start register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_B_HEND
  - Sets Overlay B Horizontal End register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_B_VSTART
  - Sets Overlay B Vertical Start register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_B_VEND
  - Sets Overlay B Vertical End register
  - Value – u16 (range:0xFFF - 0x000)
- ADI_PIXC_CMD_SET_OVERLAY_B_TRANSPARENCY
  - Sets Overlay B Transparency value register
  - Value – u16 (range:0xF - 0x0)
- ADI_PIXC_CMD_SET_RY_CONVERSION_COEFFICIENT
  - Sets R/Y coefficients for color space conversion matrix
  - Value – s32
- ADI_PIXC_CMD_SET_GU_CONVERSION_COEFFICIENT
  - Sets G/U coefficients for color space conversion matrix
  - Value – s32
- ADI_PIXC_CMD_SET_BV_CONVERSION_COEFFICIENT
  - Sets B/V coefficients for color space conversion matrix
  - Value – s32
- ADI_PIXC_CMD_SET_COLOR_CONVERSION_BIAS
  - Sets color conversion bias for color conversion matrix
  - Value – s32
- ADI_PIXC_CMD_SET_TRANSPARENCY_COLOR
  - Sets Transparency color value register
  - Value – u32

Commands to configure individual bits/fields of Pixel compositor control register

- ADI_PIXC_CMD_SET_OVERLAY_A_ENABLE
  - Enable/Disable Overlay A
  - Value = TRUE/FALSE (Default=FALSE)
- ADI_PIXC_CMD_SET_OVERLAY_B_ENABLE
  - Enable/Disable Overlay B
  - Value = TRUE/FALSE (Default=FALSE)
- ADI_PIXC_CMD_SET_TRANSPARENCY_COLOR_ENABLE
  - Enable/Disable Transparency color
  - Value = TRUE/FALSE (Default=FALSE)
- ADI_PIXC_CMD_SET_IMAGE_DATA_YUV
  - Sets Image Data format to YUV
  - Value – NULL
- ADI_PIXC_CMD_SET_OVERLAY_DATA_YUV
  - Sets Overlay Data format to YUV
  - Value – NULL
- ADI_PIXC_CMD_SET_OUTPUT_DATA_YUV
  - Sets Output Data format to YUV
  - Value – NULL
- ADI_PIXC_CMD_SET_RESAMPLE_MODE_DUPLICATE
  - Sets re-sampling mode to duplicate for up and down-sampling
  - Value – NULL
- ADI_PIXC_CMD_SET_IMAGE_DATA_RGB
  - Sets Image Data format to RGB
  - Value – NULL
- ADI_PIXC_CMD_SET_OVERLAY_DATA_RGB
  - Sets Overlay Data format to RGB
  - Value – NULL
- ADI_PIXC_CMD_SET_OUTPUT_DATA_RGB
  - Sets Output Data format to RGB
  - Value – NULL
- ADI_PIXC_CMD_SET_RESAMPLE_MODE_AVERAGE
  - Sets re-sampling mode to average for up and down-sampling
  - Value – u16 (range:0xFFF - 0x000)

Commands to configure RY conversion coefficient register fields

- ADI_PIXC_CMD_SET_RY_CONVERSION_ELEMENT1
  - Sets RY conversion coefficient element 1 (A11)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_RY_CONVERSION_ELEMENT2
  - Sets RY conversion coefficient element 2 (A12)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_RY_CONVERSION_ELEMENT3
  - Sets RY conversion coefficient element 3 (A13)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_RY_MULTIPLY4_ENABLE
  - Sets RY multiply by 4 factor enable
  - Value = TRUE/FALSE (Default=FALSE)

Commands to configure GU conversion coefficient register fields

- ADI_PIXC_CMD_SET_GU_CONVERSION_ELEMENT1
  - Sets GU conversion coefficient element 1 (A21)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_GU _CONVERSION_ELEMENT2
  - Sets GU conversion coefficient element 2 (A22)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_ GU_CONVERSION_ELEMENT3
  - Sets GU conversion coefficient element 3 (A23)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_GU_MULTIPLY4_ENABLE
  - Sets GU multiply by 4 factor enable
  - Value = TRUE/FALSE (Default=FALSE)

Commands to configure BV conversion coefficient register fields

- ADI_PIXC_CMD_SET_BV_CONVERSION_ELEMENT1
  - Sets BV conversion coefficient element 1 (A31)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_BV _CONVERSION_ELEMENT2
  - Sets BV conversion coefficient element 2 (A32)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_ BV_CONVERSION_ELEMENT3
  - Sets BV conversion coefficient element 3 (A33)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_BV_MULTIPLY4_ENABLE
  - Sets BV multiply by 4 factor enable
  - Value = TRUE/FALSE (Default=FALSE)

Commands to configure color conversion bias register

- ADI_PIXC_CMD_SET_CONVERSION_BIAS_VECTOR1
  - Sets color conversion bias vector 1 (A41)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_CONVERSION_BIAS_VECTOR2
  - Sets color conversion bias vector 2 (A42)
  - Value = s16 (range:0x3FF - 0x000)
- ADI_PIXC_CMD_SET_CONVERSION_BIAS_VECTOR3
  - Sets color conversion bias vector 3 (A43)
  - Value = s16 (range:0x3FF - 0x000)

Commands to configure Transparency color value register

- ADI_PIXC_CMD_SET_TRANSPARENT_COLOR_RY
  - Sets Transparent color for RY component
  - Value = u8 (range:0xFF - 0x00)
- ADI_PIXC_CMD_SET_TRANSPARENT_COLOR_GU
  - Sets Transparent color for GU component
  - Value = u8 (range:0xFF - 0x00)
- ADI_PIXC_CMD_SET_TRANSPARENT_COLOR_BV
  - Sets Transparent color for BV component
  - Value = u8 (range:0xFF - 0x00)

Commands to Enable PIXC interrupt status report

- ADI_PIXC_CMD_ENABLE_STATUS_REPORT
    - Enable/Disable PIXC interrupt status report
    - Value = TRUE/FALSE (Default=FALSE)

Commands to get PIXC individual bit/field value of interrupt status register

- ADI_PIXC_CMD_GET_WATERMARK_LEVEL
    - Gets Watermark level of Pixel compositor FIFOs
    - Value = u8*
- ADI_PIXC_CMD_GET_OVERLAY_FIFO_STATUS
    - Gets Overlay FIFO status
    - Value = u8*
- ADI_PIXC_CMD_GET_IMAGE_FIFO_STATUS
    - Gets Image FIFO status
    - Value = u8*

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_PIXC_EVENT_OVERLAY_COMPLETE
    - Indicates that an overlay interrupt has occurred and PIXC has completed processing an overlay frame.
    - Value – NULL
- ADI_PIXC_EVENT_FRAME_COMPLETE
    - Indicates that a Frame interrupt has occurred and PIXC has completed processing a whole frame.
    - 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_BIDIRECTIONAL_DEVICE
  - Requires the device be opened for bidirectional traffic only.
- ADI_DEV_RESULT_DATAFLOW_NOT_ENABLED
  - Results in sync mode when buffers are provided before dataflow is enabled.
- ADI_DEV_RESULT_BAD_IVG
  - Bad IVG detected.
- ADI_DEV_RESULT_SWITCH_BUFFER_PAIR_INVALID
  - Invalid buffer pair provided with Switch/Update switch buffer type.
- ADI_DEV_RESULT_DMA_CHANNEL_UNAVAILABLE
  - No DMA channel is available to process the given command/buffer.
- ADI_DEV_RESULT_ATTEMPTED_BUFFER_TABLE_NESTING
  - Results when client submits buffer table that contains one or more nested buffer tables.

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_PIXC_RESULT_FRAME_TYPE_INVALID
  - Occurs when client provides invalid color conversion mode frame type(s)
- ADI_PIXC_RESULT_PIXEL_PER_LINE_INVALID
  - Occurs when client provides invalid Pixel Per Line value
- ADI_PIXC_RESULT_LINES_PER_FRAME_INVALID
  - Occurs when client provides invalid Lines Per Frame 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.

- ADIPIXCEntryPoint

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_PIXC_CMD_SET_CONTROL_REG
Pixels per Frame Register	0	0xFFFF to 1	ADI_PIXC_CMD_SET_PIXELS_PER_LINE
Lines per Frame Register	0	0xFFFF to 1	ADI_PIXC_CMD_SET_LINES_PER_FRAME
Overlay A Horizontal Start register	0	0xFFF to 0	ADI_PIXC_CMD_SET_OVERLAY_A_HSTART
Overlay A Horizontal End register	0	0xFFF to 0	ADI_PIXC_CMD_SET_OVERLAY_A_HEND
Overlay A Vertical Start register	0	0x3FF to 0	ADI_PIXC_CMD_SET_OVERLAY_A_VSTART
Overlay A Vertical End register	0	0x3FF to 0	ADI_PIXC_CMD_SET_OVERLAY_A_VEND
Overlay A Transparency value	0	0xF to 0	ADI_PIXC_CMD_SET_OVERLAY_A_TRANSPARENCY
Overlay B Horizontal Start register	0	0xFFF to 0	ADI_PIXC_CMD_SET_OVERLAY_B_HSTART
Overlay B Horizontal End register	0	0xFFF to 0	ADI_PIXC_CMD_SET_OVERLAY_B_HEND
Overlay B Vertical Start register	0	0x3FF to 0	ADI_PIXC_CMD_SET_OVERLAY_B_VSTART
Overlay B Vertical End register	0	0x3FF to 0	ADI_PIXC_CMD_SET_OVERLAY_B_VEND
Overlay B Transparency value	0	0xF to 0	ADI_PIXC_CMD_SET_OVERLAY_B_TRANSPARENCY
R/Y coefficients	0	Application dependent	ADI_PIXC_CMD_SET_RY_CONVERSION_COEFFICIENT

Item	Default Value	Possible Values	Command ID
G/U coefficients	0	Application dependent	ADI_PIXC_CMD_SET_GU_CONVERSION_COEFFICIENT
B/V coefficients	0	Application dependent	ADI_PIXC_CMD_SET_BV_CONVERSION_COEFFICIENT
Color Conversion bias	0	Application dependent	ADI_PIXC_CMD_SET_COLOR_CONVERSION_BIAS
Transparency color value register	0	0xFF to 0x00 for each fields	ADI_PIXC_CMD_SET_TRANSPARENCY_COLOR

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_PIXC_CMD_SET_CONTROL_REG
Pixels per Frame Register	0xFFFF to 1	ADI_PIXC_CMD_SET_PIXELS_PER_LINE
Lines per Frame Register	0xFFFF to 1	ADI_PIXC_CMD_SET_LINES_PER_FRAME
R/Y coefficients	Application dependent	ADI_PIXC_CMD_SET_RY_CONVERSION_COEFFICIENT
G/U coefficients	Application dependent	ADI_PIXC_CMD_SET_GU_CONVERSION_COEFFICIENT
B/V coefficients	Application dependent	ADI_PIXC_CMD_SET_BV_CONVERSION_COEFFICIENT
Color Conversion bias	Application dependent	ADI_PIXC_CMD_SET_COLOR_CONVERSION_BIAS

Table 5 – Additional Required Settings

7. Hardware Considerations

None.

8. Using PIXC Driver in Applications

This section explains how to use PIXC 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 PIXC device + memory for other devices used by the application

8.2. Interrupt Manager Data memory allocation

This section explains Interrupt manager memory allocation requirements for applications using this driver. The application should allocate a secondary interrupt memory of size **ADI_INT_SECONDARY_MEMORY** for each PIXC 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 PIXC status reporting.

8.3. Typical usage of PIXC device driver

a. PIXC (driver) initialization

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

Step 2: Enable PIXC status reporting (if required)

b. Initializing and controlling PIXC (hardware)

Step 3: Configure PIXC color conversion coefficient registers to specific color conversion mode (refer section 10.2 (page 25) for example)

Step 4: Configure other PIXC device registers to specific operating mode

c. Submitting buffers

Step 5: Queue buffers to PIXC Image/overlay DMA using adi_dev_Read( ) and PIXC output DMA using adi_dev_Write( )

Step 6: Enable PIXC Dataflow

Step 7: Respond to callbacks

c. Terminating PIXC driver

Step 8: Disable PIXC dataflow

Step 9: Disable PIXC status reporting (if already enabled)

Step 10: Terminate PIXC driver with adi_dev_Close( )

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

9. Data structures specific to PIXC driver

9.1. Data structure to set PIXC color conversion mode

```
/* List of frame type(s) supported by PIXC driver*/
typedef enum
{
        ADI_PIXC_FRAME_YUV422,      /* YUV422 formatted frame                                                */
        ADI_PIXC_FRAME_RGB888,      /* RGB888 formatted frame                                                */
        ADI_PIXC_FRAME_YUV444,      /* YUV444 formatted frame (for special usage cases)                      */
        ADI_PIXC_FRAME_DISABLE,     /* Disable this section of frame (applicable only for Overlay Frames)    */
} ADI_PIXC_FRAME_TYPE;

/* Data structure to hold PIXC Frame (color) conversion information */
typedef struct
{
        ADI_PIXC_FRAME_TYPE         InputFrame;         /* Input Image frame type   */
        ADI_PIXC_FRAME_TYPE         OverlayAFrame;      /* Overlay A frame type     */
        ADI_PIXC_FRAME_TYPE         OverlayBFrame;      /* Overlay B frame type     */
        ADI_PIXC_FRAME_TYPE         OutputFrame;        /* Output Image frame type*/
} ADI_PIXC_CONVERSION_MODE;
```

10. Configuring PIXC to perform color conversion/overlay

10.1. List of Color conversion/Overlay modes supported by PIXC driver

Frame type	Implies
ADI_PIXC_FRAME_RGB888	This frame is in RGB888 data format
ADI_PIXC_FRAME_YUV422	This frame is in YUV422 data format
ADI_PIXC_FRAME_YUV444	This frame is in YUV444 data format
ADI_PIXC_FRAME_DISABLE	This frame is in disabled. this frame type is applicable only for Overlay frames

Table 6 – PIXC Frame types

##	Image Frame	Overlay A Frame	Overlay B Frame	Output Frame
1	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888
2	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422
3	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888
4	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888
5	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888
6	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422
7	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422
8	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422
9	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888
10	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888
11	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888
12	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422
13	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422
14	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422
15	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422
16	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888
17	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888
18	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888
19	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888
20	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422
21	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422
22	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_RGB888	ADI_PIXC_FRAME_YUV422
Special usage cases				
23	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422
24	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_DISABLE	ADI_PIXC_FRAME_YUV422
25	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV422
26	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV444
27	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV444
28	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV422	ADI_PIXC_FRAME_YUV444
29	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV444	ADI_PIXC_FRAME_YUV444

Table 7 – List of color conversion/overlay modes supported by PIXC

10.2. Example to set PIXC color conversion registers

Example code to set Pixel compositor conversion mode # 3 (refer Table 7 for supported color conversion modes)

```
/* structure to hold Pixel compositor conversion mode */
ADI_PIXC_CONVERSION_MODE    PixcMode;

/* configure PIXC color conversion mode structure */
PixcMode.InputFrame          = ADI_PIXC_FRAME_YUV422;     /* Input image is in YUV422 format      */
PixcMode.OverlayAFrame       = ADI_PIXC_FRAME_YUV422;     /* Overlay A is in YUV422 format        */
PixcMode.OverlayBFrame       = ADI_PIXC_FRAME_DISABLE;    /* Overlay B is not required            */
PixcMode.OutputFrame         = ADI_PIXC_FRAME_RGB888;     /* Output must be in RGB888 format      */

/* Pass the color conversion mode structure to PIXC driver */
adi_dev_Control(PixcHandle,ADI_PIXC_CMD_SET_COLOR_CONVERSION_MODE, (void *) &PixcMode);
```

10.3. Enabling/Disabling ITU-R 656 support

PIXC hardware expects/outputs YUV422 data in VYUY format. PIXC driver provides a simple workaround to the user which eliminates the need of any software data handling for applications involving color conversion of ITU-R 656 (UYVY) type YUV422 data. Application can enable/disable ITU-R 656 support by using PIXC driver specific command ‘ADI_PIXC_CMD_ENABLE_ITUR656_SUPPORT‘, with value as TRUE or FALSE, where TRUE enables ITU-R 656 support and FALSE sets PIXC color conversion registers to its original operating mode.

PIXC driver disables ITU-R 656 support by default.

11. Submitting buffers to PIXC

11.1. With both Overlays disabled

```
/* 2D buffer for Input image */
ADI_DEV_2D_BUFFER    InBufferPixc;
/* 2D buffer for Output image */
ADI_DEV_2D_BUFFER    OutBufferPixc;

/* Configure above Input and Output 2D buffers */
……
……

/* Submit PIXC input buffer – type as ADI_DEV_2D */
adi_dev_Write(PixcDriverHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *) &InBufferPixc);

/* Submit PIXC output buffer – type as ADI_DEV_2D */
adi_dev_Read(PixcDriverHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *) &OutBufferPixc);
```

11.2. With one overlay enabled

```
/* 2D buffer for Input image */
ADI_DEV_2D_BUFFER    InBufferPixc;
/* 2D buffer for Overlay image */
ADI_DEV_2D_BUFFER    OvrBufferPixc;
/* 2D buffer for Output image */
ADI_DEV_2D_BUFFER    OutBufferPixc;

/* Configure above Input, Overlay and Output 2D buffers */
……
……

/* create a Table of Buffer pairs to submit PIXC input buffers */
ADI_DEV_BUFFER_PAIR PixcInputBufs [ ] =
{
        { ADI_DEV_2D,              (ADI_DEV_BUFFER *)&InBufferPixc       },  /* PIXC Image Buffer    */
        { ADI_DEV_2D,              (ADI_DEV_BUFFER *)& OvrBufferPixc     },  /* PIXC Overlay Buffer  */
        { ADI_DEV_BUFFER_END,      0                                     }   /* Terminate buffer table*/
};

/* Submit PIXC input buffers – type as ADI_DEV_BUFFER_TABLE */
adi_dev_Write(PixcDriverHandle, ADI_DEV_BUFFER_TABLE, (ADI_DEV_BUFFER *) PixcInputBufs);

/* Submit PIXC output buffer – type as ADI_DEV_2D */
adi_dev_Read(PixcDriverHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *) &OutBufferPixc);
```

11.3. With both Overlays enabled

```
/* 2D buffer for Input image */
ADI_DEV_2D_BUFFER     InBufferPixc;
/* 2D buffer for Overlay A image */
ADI_DEV_2D_BUFFER     OvrABufferPixc;
/* 2D buffer for Overlay B image */
ADI_DEV_2D_BUFFER     OvrBBufferPixc;
/* 2D buffer for Output image */
ADI_DEV_2D_BUFFER     OutBufferPixc;

/* Configure above Input, Overlay A, Overlay B and Output 2D buffers */
……
……

/* Chain Overlay B buffer to Overlay A buffer */
OvrABufferPixc.pNext        = &OvrBBufferPixc;

/* create a Table of Buffer pairs to submit PIXC input buffers */
ADI_DEV_BUFFER_PAIR PixcInputBufs [ ] =
{
        { ADI_DEV_2D,               (ADI_DEV_BUFFER *)&InBufferPixc       },  /* PIXC Image Buffer        */
        { ADI_DEV_2D,               (ADI_DEV_BUFFER *)&OvrABufferPixc     },  /* PIXC Overlay A Buffer    */
        { ADI_DEV_BUFFER_END,       0                                     }   /* Terminate buffer table   */
};

/* Submit PIXC input buffers – type as ADI_DEV_BUFFER_TABLE */
adi_dev_Write(PixcDriverHandle, ADI_DEV_BUFFER_TABLE, (ADI_DEV_BUFFER *) &InBufferPixc);

/* Submit PIXC output buffer – type as ADI_DEV_2D */
adi_dev_Read(PixcDriverHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *) &OutBufferPixc);
```