ANALOG DEVICES

ADV7183
DEVICE DRIVER

DATE: 31 JAN 2006

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
2006/01/31	Initial release
2006/05/15	Updated to new device access interface Added register access examples

Table 1 – Revision History

1. Overview

The ADV7183 driver provides the user application control over the ADV7183 video decoder.
The decoder's registers are configured using TWI and the video dataflow through PPI port. The application program can configure ADV7183 registers and specific return codes are sent in result of success or any failure. The driver uses Device Access Service (adi_device_access) to access internal registers of the encoder.

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/ppi/adi_ppi.h> This file contains all definitions, function prototypes etc. specific to PPI device
- <drivers/deviceaccess/adi_device_access.h> This file contains all definitions, function prototypes etc. for TWI/SPI device access service
- <drivers/decoder/adi_ad7183.h> This file contains all definitions, function prototypes etc. specific to ADV7183 device

2.2. Source Files

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

- adi_ad7183.c

3. Lower Level Drivers

ADV7183 driver is layered on TWI and PPI drivers

3.1. TWI

The TWI device driver is used by the ADV7183 driver to configure decoder registers.

3.2. PPI

The PPI device driver is used by the ADV7183 to read in the video data from the decoder.
The PPI device to be used shall be specified by the user; by default the ADV7183 driver sets PPI device 0 to be used for its video dataflow.

Application can directly communicate with the PPI device allocated for ADV7183 video dataflow by calling adi_dev_Control( ) function with PDDHandle specific to ADV7183 driver, command specific to the PPI driver and value specific to the command.

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

The ADV7183 driver uses one PPI port and DMA control and one TWI port, this can be either a hardware TWI if the Blackfin device being used has a hardware port (BF537), or pseudo TWI if no TWI hardware exists(BF533 or BF561). In this case the TWI uses one timer and 2 general purpose flags.

4.1. Interrupts

This driver does not use any interrupts directly, please see PPI and TWI documentation for resources required by these drivers.

4.2. DMA

The driver doesn't support DMA directly, but uses a DMA driven PPI for its video dataflow. ADV7183 supports only inbound dataflow and memory should be allocated for one DMA channel.

4.3. Timers

This driver does not use any timers directly, however check the PPI and TWI documentation for timer resources required by these drivers.

4.4. Real-Time Clock

This driver does not require the real-time clock.

4.5. Programmable Flags

No programmable flags are directly used by this driver. If the client intends to use a pseudo TWI to control ADV7183, a TWI configuration table must be passed with flag settings that will be used for pseudo TWI operation.

4.6. Pins

This driver does not use any external pins.

5. Supported Features of the Device Driver

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

5.1. Directionality

The driver supports the dataflow directions listed in the table below.

ADI_DEV_DIRECTION	Description
ADI_DEV_DIRECTION_INBOUND	Supports the reception of data in 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 loopback 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 – ignored
- ADI_DEV_2D_BUFFER
  - Two-dimensional buffer
  - pAdditionalInfo – ignored

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_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_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_REGISTER_READ
  - Reads a single device register
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)
- ADI_DEV_CMD_REGISTER_FIELD_READ
  - Reads a specific field location in a single device register
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- ADI_DEV_CMD_REGISTER_TABLE_READ
  - Reads a table of selective device registers
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)
- ADI_DEV_CMD_REGISTER_FIELD_TABLE_READ
  - Reads a table of selective device register fields
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- ADI_DEV_CMD_REGISTER_BLOCK_READ
  - Reads a block of consecutive device registers
  - Value – ADI_DEV_ACCESS_REGISTER_BLOCK * (register specifics)
- ADI_DEV_CMD_REGISTER_WRITE
  - Writes to a single device register
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)
- ADI_DEV_CMD_REGISTER_FIELD_WRITE
  - Writes to a specific field location in a single device register
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- ADI_DEV_CMD_REGISTER_TABLE_WRITE
  - Writes to a table of selective device registers
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)

- ADI_DEV_CMD_REGISTER_FIELD_TABLE_WRITE
  - Writes to a table of selective device register fields
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- ADI_DEV_CMD_REGISTER_BLOCK_WRITE
  - Writes to a block of consecutive device registers
  - Value – ADI_DEV_ACCESS_REGISTER_BLOCK * (register specifics)

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. The driver also supports commands specific to PPI driver. Please refer to PPI driver documentation for further information.

- ADI_AD7183_CMD_SET_TWI_DEVICE_NUMBER
  - Sets the TWI device number to use
  - Value – u32 (device number)
- ADI_AD7183_CMD_SET_TWI_DEVICE_ADDRESS
  - Sets the device TWI address
  - Value – u32 (device address)
- ADI_AD7183_CMD_SET_TWI_CONFIG_TABLE
  - Sets the configuration options for TWI device
  - Value - ADI_DEV_CMD_VALUE_PAIR * (extra TWI configuration controls)
- ADI_AD7183_CMD_OPEN_PPI
  - Sets PPI device number to be used for ADV7183 video dataflow.
  - Value – u32 (device number)
- ADI_AD7183_CMD_SET_VIDEO_FORMAT
  - Sets the PPI device control register and Lines Per Frame register to PAL or NTSC mode.
  - Value – u32 (0(NTSC) or 1(PAL))
- ADI_AD7183_CMD_SET_ACTIVE_VIDEO
  - Sets PPI control register to receive active video field only.
  - Value – NULL
- ADI_AD7183_CMD_SET_VERTICAL_BLANKING
  - Sets PPI control register to receive vertical blanking only.
  - Value – NULL

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 callback function of the client should be prepared to process each of the events in these 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 this 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.

This driver does not have any specific events.

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.

Return codes specific to TWI/SPI Device access service

- ADI_DEV_RESULT_TWI_LOCKED
  - Indicates the present TWI device is locked in other operation
- ADI_DEV_RESULT_REQUIRES_TWI_CONFIG_TABLE
  - Client need to supply a configuration table for the TWI driver
- ADI_DEV_RESULT_CMD_NOT_SUPPORTED
  - Command not supported by the Device Access Service
- ADI_DEV_RESULT_INVALID_REG_ADDRESS
  - The client attempting to access an invalid register address
- ADI_DEV_RESULT_INVALID_REG_FIELD
  - The client attempting to access an invalid register field location
- ADI_DEV_RESULT_INVALID_REG_FIELD_DATA
  - The client attempting to write an invalid data to selected register field location

- ADI_DEV_RESULT_ATTEMPT_TO_WRITE_READONLY_REG
  - The client attempting to write to a read-only location
- ADI_DEV_RESULT_ATTEMPT_TO_ACCESS_RESERVE_AREA
  - The client attempting to access a reserved location
- ADI_DEV_RESULT_ACCESS_TYPE_NOT_SUPPORTED
  - Device Access Service does not support the access type provided by the driver

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_AD7183_RESULT_CMD_NOT_SUPPORTED
  - Command supplied by the client is not supported.
- ADI_AD7183_RESULT_PPI_NOT_OPENED
  - Results when client tries to set DataFlow or DataFlow-Method before PPI is opened.
- ADI_AD7183_RESULT_BAD_VIDEO_FORMAT
  - Results when the client provides an invalid video format.

6. 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.

- ADIAD7183EntryPoint

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
PPI device	0	0,1	ADI_AD7183_CMD_OPEN_PPI
Video Format	0(NTSC)	0(NTSC), 1(PAL)	ADI_AD7183_CMD_SET_VIDEO_FORMAT
TWI device	0	0,1	ADI_AD7183_CMD_SET_TWI_DEVICE
TWI address	ADV7183_TWI_ADDR0	ADV7183_TWI_ADDR0, ADV7183_TWI_ADDR1	ADI_AD7183_CMD_SET_TWI_ADDRESS

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
TWI Configuration Table	Pointer to TWI configuration table of type ADI_DEV_CMD_VALUE_PAIR	ADI_AD7183_CMD_SET_TWI_CONFIG_TABLE
PPI device	0 (for BF533, BF537) 0, 1 (for BF561)	ADI_AD7183_CMD_OPEN_PPI
Video format	0(NTSC), 1(PAL)	ADI_AD7183_CMD_SET_VIDEO_FORMAT
Dataflow method	See section 5.2	ADI_DEV_CMD_SET_DATAFLOW_METHOD

Table 5 – Additional Required Settings

7. Hardware Considerations

The TWI slave address of ADV7183 can be set by issuing the command 'ADI_ADV7183_CMD_SET_TWI_DEVICE_ADDRESS'. If the client intends to use pseudo TWI to access ADV7183 registers, specific port pins should be set in Blackfin to generate TWI SCL and SDA.

The following table is a list of registers that can be accessed on the ADV7183. Please refer to the ADV7183 device manual for a full description of registers and chip functionality.

Common Registers

Register	Address	Value	r/w	Description
ADV7183_INPUT_CTR	0x00	0x00	rw	Input Control
ADV7183_VIDEO_SELECTION	0x01	0xC8	rw	Video Selection
ADV7183_OUTPUT_CTR	0x03	0x0C	rw	Output Control
ADV7183_EXTENDED_OUTPUT_CTR	0x04	0x45	rw	Extended Output Control
ADV7183_AUTODETECT_ENABLE	0x07	0x7F	rw	Autodetect Enable
ADV7183_CONTRAST	0x08	0x80	rw	Contrast
ADV7183_BRIGHTNESS	0x0A	0x00	rw	Brightness
ADV7183_HUE	0x0B	0x00	rw	Hue
ADV7183_DEF_VALUE_Y	0x0C	0x36	rw	Default Value Y
ADV7183_DEF_VALUE_C	0x0D	0x7C	rw	Default Value C
ADV7183_ADI_CTR	0X0E	0x00	rw	ADI Control
ADV7183_POWER_MGM	0x0F	0x00	rw	Power Management
ADV7183_STATUS1_RO	0x10	0xxx	r	Status 1
ADV7183_IDENT_RO	0x11	0xxx	r	Ident
ADV7183_STATUS2_RO	0x12	0xxx	r	Status 2
ADV7183_STATUS3_RO	0x13	0xxx	r	Status 3
ADV7183_ANALOG_CLAMP_CTR	0x14	0x12	rw	Analog Clamp Control
ADV7183_DIGITAL_CLAMP_CTR1	0x15	0x40	rw	Digital Clamp Control 1
ADV7183_SHAPING_FILTER_CTR	0x17	0x01	rw	Shaping Filter Control
ADV7183_SHAPING_FILTER_CTR2	0x18	0x93	rw	Shaping Filter Control 2
ADV7183_COMB_FILTER_CTR	0x19	0xF1	rw	Comb Filter Control
ADV7183_ADI_CTR2	0x1D	0x00	rw	ADI Control 2
ADV7183_PIXEL_DELAY_CTR	0x27	0x58	rw	Pixel Delay Control
ADV7183_MISC_GAIN_CTR	0x2B	0xE1	rw	Misc Gain Control
ADV7183_AGC_MODE_CTR	0x2C	0xAE	rw	AGC Mode Control
ADV7183_CHROMA_GAIN_CTR1	0x2D	0xF4	rw	Chroma Gain Control 1
ADV7183_CHROMA_GAIN_CTR2	0x2E	0x00	rw	Chroma Gain Control 2
ADV7183_LUMA_GAIN_CTR1	0x2F	0xFx	rw	Luma Gain Control 1
ADV7183_LUMA_GAIN_CTR2	0x30	0xxx	rw	Luma Gain Control 2
ADV7183_VSYNC_FIELD_CTR1	0x31	0x12	rw	VSync Field Control 1
ADV7183_VSYNC_FIELD_CTR2	0x32	0x41	rw	Vsync Field Control 2
ADV7183_VSYNC_FIELD_CTR3	0x33	0x84	rw	Vsync Field Control 3
ADV7183_HSYNC_POS_CTR1	0x34	0x00	rw	Hsync Position Control 1
ADV7183_HSYNC_POS_CTR2	0x35	0x02	rw	Hsync Position Control 2
ADV7183_HSYNC_POS_CTR3	0x36	0x00	rw	Hsync Position Control 3
ADV7183_POLARITY	0x37	0x01	rw	Polarity
ADV7183_NTSC_COMB_CTR	0x38	0x80	rw	NTSC Comb Control
ADV7183_PAL_COMB_CTR	0x39	0xC0	rw	PAL Comb Control
ADV7183_ADC_CTR	0x3A	0x10	rw	ADC Control
ADV7183_MANUAL_WINDOW_CTR	0x3D	0x43	rw	Manual Window Control

Table 6 – ADV7183 Common Registers

List of register fields.

Register Address: ADV7183_INPUT_CTR
Register Fields:
- ADV7183_VID_SEL
- ADV7183_INSEL

Register Address: ADV7183_VIDEO_SELECTION
Register Fields:
- ADV7183_ENHSPLL
- ADV7183_BETACAM
- ADV7183_ENVSPROC

Register Address: ADV7183_OUTPUT_CTR
Register Fields:
- ADV7183_VBI_EN
- ADV7183_TOD
- ADV7183_OF_SEL
- ADV7183_SD_DUP_AV

Register Address: ADV7183_EXTENDED_OUTPUT_CTR
Register Fields:
- ADV7183_BT656_4
- ADV7183_TIM_OE
- ADV7183_BL_C_VBI
- ADV7183_EN_SFL_PI
- ADV7183_RANGE

Register Address: ADV7183_AUTODETECT_ENABLE
Register Fields:
- ADV7183_AD_SEC525_EN
- ADV7183_AD_SECAM_EN
- ADV7183_AD_N443_EN
- ADV7183_ADP60_EN
- ADV7183_AD_PALN_EN
- ADV7183_AD_PALM_EN
- ADV7183_AD_NTSC_EN
- ADV7183_AD_PAL_EN

Register Address: ADV7183_CONTRAST
Register Fields:
- ADV7183_CON

Register Address: ADV7183_BRIGHTNESS
Register Fields:
- ADV7183_BRI

Register Address: ADV7183_HUE
Register Fields:
- ADV7183_HUE0

Register Address: ADV7183_DEF_VALUE_Y
Register Fields:
- ADV7183_DEF_Y

- ADV7183_DEF_VAL_EN

Register Address: ADV7183_DEF_VALUE_C
Register Fields:
- ADV7183_DEF_C

Register Address: ADV7183_ADI_CTR
Register Fields:
- ADV7183_SUB_USR_EN

Register Address: ADV7183_POWER_MGM
Register Fields:
- ADV7183_RES
- ADV7183_PWRDN
- ADV7183_PDBP

Register Address: ADV7183_STATUS1_RO
Register Fields:
- ADV7183_COL_KILL
- ADV7183_AD_RESULT2
- ADV7183_AD_RESULT1
- ADV7183_AD_RESULT0
- ADV7183_FOLLOW_PW
- ADV7183_FSC_LOCK
- ADV7183_LOST_LOCK
- ADV7183_IN_LOCK

Register Address: ADV7183_IDENT_RO
Register Fields:
- ADV7183_IDENT

Register Address: ADV7183_STATUS2_RO
Register Fields:
- ADV7183_FSCNSTD
- ADV7183_LLNSTD
- ADV7183_MVAGCDET
- ADV7183_MVPSDET
- ADV7183_MVCST3
- ADV7183_MVCSDET

Register Address: ADV7183_STATUS3_RO
Register Fields:
- ADV7183_PALSWLOCK
- ADV7183_INTERLACE
- ADV7183_STDFLDLEN
- ADV7183_FREE_RUN_ACT
- ADV7183_SD_OP_50HZ
- ADV7183_GEMD
- ADV7183_INST_HLOCK

Register Address: ADV7183_ANALOG_CLAMP_CTR
Register Fields:
- ADV7183_CCLEN

Register Address: ADV7183_DIGITAL_CLAMP_CTR1
Register Fields:
- ADV7183_DCT

Register Address: ADV7183_SHAPING_FILTER_CTR
Register Fields:
- ADV7183_CSFM
- ADV7183_YSFM

Register Address: ADV7183_SHAPING_FILTER_CTR2
Register Fields:
- ADV7183_WYSFMOVR
- ADV7183_WYSFM

Register Address: ADV7183_COMB_FILTER_CTR
Register Fields:
- ADV7183_NSFSEL
- ADV7183_PSFSEL

Register Address: ADV7183_ADI_CTR2
Register Fields:
- ADV7183_TRI_LLC
- ADV7183_EN28XTAL
- ADV7183_VS_JIT_COMP_EN

Register Address: ADV7183_PIXEL_DELAY_CTR
Register Fields:
- ADV7183_SWPC
- ADV7183_AUTO_PDC_EN
- ADV7183_CTA
- ADV7183_LTA

Register Address: ADV7183_MISC_GAIN_CTR
Register Fields:
- ADV7183_CKE
- ADV7183_PW_UPD

Register Address: ADV7183_AGC_MODE_CTR
Register Fields:
- ADV7183_LAGC
- ADV7183_CAGC

Register Address: ADV7183_CHROMA_GAIN_CTR1
Register Fields:
- ADV7183_CAGT
- ADV7183_CMG8

Register Address: ADV7183_CHROMA_GAIN_CTR2
Register Fields:
- ADV7183_CMG0

Register Address: ADV7183_LUMA_GAIN_CTR1
Register Fields:
- ADV7183_LAGT
- ADV7183_LMG8

Register Address: ADV7183_LUMA_GAIN_CTR2
Register Fields:
- ADV7183_LMG0

Register Address: ADV7183_VSYNC_FIELD_CTR1
Register Fields:
- ADV7183_NEWAVMODE
- ADV7183_HVSTIM

Register Address: ADV7183_VSYNC_FIELD_CTR2
Register Fields:
- ADV7183_VSBHO
- ADV7183_VSBHE

Register Address: ADV7183_VSYNC_FIELD_CTR3
Register Fields:
- ADV7183_VSEHO
- ADV7183_VSEHE

Register Address: ADV7183_HSYNC_POS_CTR1
Register Fields:
- ADV7183_HSB8
- ADV7183_HSE8

Register Address: ADV7183_HSYNC_POS_CTR2
Register Fields:
- ADV7183_HSB0

Register Address: ADV7183_HSYNC_POS_CTR3
Register Fields:
- ADV7183_HSE0

Register Address: ADV7183_POLARITY
Register Fields:
- ADV7183_PHS
- ADV7183_PVS
- ADV7183_PF
- ADV7183_PCLK

Register Address: ADV7183_NTSC_COMB_CTR
Register Fields:
- ADV7183_CTAPSN
- ADV7183_CCMN
- ADV7183_YCMN

Register Address: ADV7183_PAL_COMB_CTR
Register Fields:
- ADV7183_CTAPSP
- ADV7183_CCMP
- ADV7183_YCMP

Register Address: ADV7183_ADC_CTR
Register Fields:
- ADV7183_PWRDN_ADC0
- ADV7183_PWRDN_ADC1
- ADV7183_PWRDN_ADC2

Register Address: ADV7183_MANUAL_WINDOW_CTR
Register Fields:
- ADV7183_CKILLTHR

Normal (page1) registers

Register	Address	Value	r/w	Description
ADV7183_RESAMPLE_CTR	0x41	0x41	rw	Resample Control
ADV7183_GEMSTAR_CTR1	0x48	0x00	rw	Gemstar Control 1
ADV7183_GEMSTAR_CTR2	0x49	0x00	rw	Gemstar Control 2
ADV7183_GEMSTAR_CTR3	0x4A	0x00	rw	Gemstar Control 3
ADV7183_GEMSTAR_CTR4	0x4B	0x00	rw	Gemstar Control 4
ADV7183_GEMSTAR_CTR5	0x4C	0xx0	rw	Gemstar Control 5
ADV7183_CTI_DNR_CTR1	0x4D	0xEF	rw	CTI DNR Control 1
ADV7183_CTI_DNR_CTR2	0x4E	0x08	rw	CTI DNR Control 2
ADV7183_CTI_DNR_CTR4	0x50	0x08	rw	CTI DNR Control 4
ADV7183_LOCK_COUNT	0x51	0x24	rw	Lock Count
ADV7183_FREERUN_LINE_LENGTH1	0x8F	0x00	w	Freerun Line Length 1
ADV7183_VBI_INFO_RO	0x90	0xxx	r	VBI Info
ADV7183_WSS1_RO	0x91	0xxx	r	WSS 1
ADV7183_WSS2_RO	0x92	0xxx	r	WSS 2
ADV7183_EDTV1_RO	0x93	0xxx	r	EDTV 1
ADV7183_EDTV2_RO	0x94	0xxx	r	EDTV 2
ADV7183_EDTV3_RO	0x95	0xxx	r	EDTV 3
ADV7183_CGMS1_RO	0x96	0xxx	r	CGMS 1
ADV7183_CGMS2_RO	0x97	0xxx	r	CGMS 2
ADV7183_CGMS3_RO	0x98	0xxx	r	CGMS 3
ADV7183_CCAP1_RO	0x99	0xxx	r	CCAP 1
ADV7183_CCAP2_RO	0x9A	0xxx	r	CCAP 2
ADV7183_LETTERBOX1_RO	0x9B	0xxx	r	Letter Box 1
ADV7183_LETTERBOX2_RO	0x9C	0xxx	r	Letter Box 2
ADV7183_LETTERBOX3_RO	0x9D	0xxx	r	Letter Box 3
ADV7183_CRC_ENABLE	0xB2	0x1C	w	CRC enable
ADV7183_ADC_SWITCH1	0xC3	0xxx	rw	ADC Switch 1
ADV7183_ADC_SWITCH2	0xC4	0xxx	rw	ADC Switch 2
ADV7183_LETTERBOX_CTR1	0xDC	0xAC	rw	Letterbox Control1
ADV7183_LETTERBOX_CTR2	0xDD	0x4C	rw	Letterbox Control2
ADV7183_SD_OFFSET_CB	0xE1	0x80	rw	SD Offset Cb
ADV7183_SD_OFFSET_CR	0xE2	0x80	rw	SD Offset Cr
ADV7183_SD_SATURATION_CB	0xE3	0x80	rw	SD Saturation Cb
ADV7183_SD_SATURATION_CR	0xE4	0x80	rw	SD Saturation Cr
ADV7183_NTSC_VBIT_BEGIN	0xE5	0x25	rw	NTSC V Bit Begin
ADV7183_NTSC_VBIT_END	0xE6	0x04	rw	NTSC V Bit End
ADV7183_NTSC_FBIT_TOGGLE	0xE7	0x63	rw	NTSC F Bit Toggle
ADV7183_PAL_VBIT_BEGIN	0xE8	0x65	rw	PAL V Bit Begin
ADV7183_PAL_VBIT_END	0xE9	0x14	rw	PAL V Bit End
ADV7183_PAL_FBIT_TOGGLE	0xEA	0x63	rw	PAL F Bit Toggle
ADV7183_DRIVE_STRENGTH	0xF4	0x15	rw	Drive Strength
ADV7183_IF_COMP_CTR	0xF8	0x00	rw	IF Comp Control
ADV7183_VS_MODE_CTR	0xF9	0x00	rw	VS Mode Control

Table 7 – ADV7183 Normal (Page 1) Registers

List of register fields.

Register Address: ADV7183_RESAMPLE_CTR
Register Fields:

- ADV7183_SFL_INV

Register Address: ADV7183_GEMSTAR_CTR1
Register Fields:
- ADV7183_CDECEL8

Register Address: ADV7183_GEMSTAR_CTR2
Register Fields:
- ADV7183_CDECEL0

Register Address: ADV7183_GEMSTAR_CTR3
Register Fields:
- ADV7183_CDECOL8

Register Address: ADV7183_GEMSTAR_CTR4
Register Fields:
- ADV7183_CDECOL0

Register Address: ADV7183_GEMSTAR_CTR5
Register Fields:
- ADV7183_CDECAD

Register Address: ADV7183_CTIDNR_CTR1
Register Fields:
- ADV7183_DNR_EN
- ADV7183_CTI_AB
- ADV7183_CTI_AB_EN
- ADV7183_CTI_EN

Register Address: ADV7183_CTIDNR_CTR2
Register Fields:
- ADV7183_CTI_TH

Register Address: ADV7183_CTIDNR_CTR4
Register Fields:
- ADV7183_DNR_TH

Register Address: ADV7183_LOCK_COUNT
Register Fields:
- ADV7183_FSCLE
- ADV7183_SRLS
- ADV7183_COL
- ADV7183_CIL

Register Address: ADV7183_FREERUN_LINE_LENGTH1
Register Fields:
- ADV7183_LLC_PAD_SEL

Register Address: ADV7183_VBI_INFO_RO
Register Fields:
- ADV7183_CGMSD
- ADV7183_EDTVD
- ADV7183_CCAPD
- ADV7183_WSSD

Register Address: ADV7183_WSS1_RO

Register Fields:
- ADV7183_WSS1

Register Address: ADV7183_WSS2_RO
Register Fields:
- ADV7183_WSS2

Register Address: ADV7183_EDTV1_RO
Register Fields:
- ADV7183_EDTV1

Register Address: ADV7183_EDTV2_RO
Register Fields:
- ADV7183_EDTV2

Register Address: ADV7183_EDTV3_RO
Register Fields:
- ADV7183_EDTV3

Register Address: ADV7183_CGMS1_RO
Register Fields:
- ADV7183_CGMS1

Register Address: ADV7183_CGMS2_RO
Register Fields:
- ADV7183_CGMS2

Register Address: ADV7183_CGMS3_RO
Register Fields:
- ADV7183_CGMS3

Register Address: ADV7183_CCAP1_RO
Register Fields:
- ADV7183_CCAP1

Register Address: ADV7183_CCAP2_RO
Register Fields:
- ADV7183_CCAP2

Register Address: ADV7183_LETTERBOX1_RO
Register Fields:
- ADV7183_LB_LCT

Register Address: ADV7183_LETTERBOX2_RO
Register Fields:
- ADV7183_LB_LCM

Register Address: ADV7183_LETTERBOX3_RO
Register Fields:
- ADV7183_LB_LCB

Register Address: ADV7183_CRC_ENABLE_WR
Register Fields:
- ADV7183_CRC_ENABLE

Register Address: ADV7183_ADC_SWITCH1
Register Fields:

- ADV7183_ADC1_SW
- ADV7183_ADCO_SW

Register Address: ADV7183_ADC_SWITCH2
Register Fields:
- ADV7183_ADC_SW_MAN
- ADV7183_ADC2_SW

Register Address: ADV7183_LETTERBOX_CTR1
Register Fields:
- ADV7183_LB_TH

Register Address: ADV7183_LETTERBOX_CTR2
Register Fields:
- ADV7183_LB_SL
- ADV7183_LB_EL

Register Address: ADV7183_SD_OFFSET_CB
Register Fields:
- ADV7183_SD_OFF_CB

Register Address: ADV7183_SD_OFFSET_CR
Register Fields:
- ADV7183_SD_OFF_CR

Register Address: ADV7183_SD_SATURATION_CB
Register Fields:
- ADV7183_SD_SAT_CB

Register Address: ADV7183_SD_SATURATION_CR
Register Fields:
- ADV7183_SD_SAT_CR

Register Address: ADV7183_NTSC_VBIT_BEGIN
Register Fields:
- ADV7183_NVBEGDELO
- ADV7183_NVBEGDELE
- ADV7183_NVBEGSIGN
- ADV7183_NVBEG

Register Address: ADV7183_NTSC_VBIT_END
Register Fields:
- ADV7183_NVENDDELO
- ADV7183_NVENDDELE
- ADV7183_NVENDSIGN
- ADV7183_NVEND

Register Address: ADV7183_NTSC_FBIT_TOGGLE
Register Fields:
- ADV7183_NFTOGDELO
- ADV7183_NFTOGDELE
- ADV7183_NFTOGSIGN
- ADV7183_NFTOG

Register Address: ADV7183_PAL_VBIT_BEGIN
Register Fields:
- ADV7183_PVBEGDELO

- ADV7183_PVBEGDELE
- ADV7183_PVBEGSIGN
- ADV7183_PVBEG

Register Address: ADV7183_PAL_VBIT_END
Register Fields:
- ADV7183_PVENDDELO
- ADV7183_PVENDDELE
- ADV7183_PVENDSIGN
- ADV7183_PVEND

Register Address: ADV7183_PAL_FBIT_TOGGLE
Register Fields:
- ADV7183_PFTOGDELO
- ADV7183_PFTOGDELE
- ADV7183_PFTOGSIGN
- ADV7183_PFTOG

Register Address: ADV7183_DRIVE_STRENGTH
Register Fields:
- ADV7183_DR_STR
- ADV7183_DR_STR_C
- ADV7183_DR_STR_S

Register Address: ADV7183_IF_COMP_CTR
Register Fields:
- ADV7183_IFFILTSEL

Register Address: ADV7183_VS_MODE_CTR
Register Fields:
- ADV7183_VS_COAST_MODE
- ADV7183_EXTEND_VS_MIN_FREQ
- ADV7183_EXTEND_VS_MAX_FREQ

Interrupt (page2) registers

Register	Address	Value	r/w	Description
ADV7183_INT_CONFIG0	0x40	0x10	rw	Interrupt Config 0
ADV7183_INT_STATUS1	0x42		r	Interrupt Status 1
ADV7183_INT_CLEAR1	0x43	0x00	w	Interrupt Clear 1
ADV7183_INT_MASK1	0x44	0x00	rw	Interrupt Maskb 1
ADV7183_INT_STATUS2	0x46		r	Interrupt Status 2
ADV7183_INT_CLEAR2	0x47	0xx0	w	Interrupt Clear 2
ADV7183_INT_MASK2	0x48	0xx0	rw	Interrupt Maskb 2
ADV7183_RAW_STATUS3	0x49		r	Raw Status 3
ADV7183_CTI_DNR_CTR4	0x4A		r	Interrupt Status 3
ADV7183_INT_CLEAR3	0x4B	0x00	w	Interrupt Clear 3
ADV7183_INT_MASK3	0x4C	0x00	rw	Interrupt Maskb 3

Table 8 – ADV7183 Interrupt (Page 2) Registers

List of register fields.

Register Address: ADV7183_INT_CONFIG0
Register Fields:
- ADV7183_INTRQ_DUR_SEL
- ADV7183_MV_INTRQ_SEL
- ADV7183_MPU_STIM_INTRQ
- ADV7183_INTRO_OP_SEL

Register Address: ADV7183_INT_STATUS1
Register Fields:
- ADV7183_MV_PS_CS_Q
- ADV7183_SD_FR_CHNG_Q
- ADV7183_SD_UNLOCK_Q
- ADV7183_SD_LOCK_Q

Register Address: ADV7183_INT_CLEAR1
Register Fields:
- ADV7183_MV_PS_CS_CLR
- ADV7183_SD_FR_CHNG_CLR
- ADV7183_SD_UNLOCK_CLR
- ADV7183_SD_LOCK_CLR

Register Address: ADV7183_INT_MASK1
Register Fields:
- ADV7183_MV_PS_CS_MSKB
- ADV7183_SD_FR_CHNG_MSKB
- ADV7183_SD_UNLOCK_MSKB
- ADV7183_SD_LOCK_MSKB

Register Address: ADV7183_INT_STATUS2
Register Fields:
- ADV7183_WSS_CHNGD_Q
- ADV7183_CGMS_CHNGD_Q
- ADV7183_CEMD_Q
- ADV7183_CCAPD_Q

Register Address: ADV7183_INT_CLEAR2
Register Fields:
- ADV7183_MPU_STIM_INTRQ_CLR
- ADV7183_WSS_CHNGD_CLR
- ADV7183_CGMS_CHNGD_CLR
- ADV7183_CEMD_CLR
- ADV7183_CCAPD_CLR

Register Address: ADV7183_INT_MASK2
Register Fields:
- ADV7183_MPU_STIM_INTRQ_MSKB
- ADV7183_WSS_CHNGD_MSKB
- ADV7183_CGMS_CHNGD_MSKB
- ADV7183_CEMD_MSKB
- ADV7183_CCAPD_MSKB

Register Address: ADV7183_RAW_STATUS3
Register Fields:
- ADV7183_SCM_LOCK
- ADV7183_SD_H_LOCK
- ADV7183_SD_V_LOCK

- ADV7183_SD_OP_50HZ_RS

Register Address: ADV7183_INT_STATUS3
Register Fields:
- ADV7183_PAL_SW_LK_CHNG_Q
- ADV7183_SCM_LOCK_CHNG_Q
- ADV7183_SD_AD_CHNG_Q
- ADV7183_SD_H_LOCK_CHNG_Q
- ADV7183_SD_V_LOCK_CHNG_Q
- ADV7183_SD_OP_CHNG_Q

Register Address: ADV7183_INT_CLEAR3
Register Fields:
- ADV7183_PAL_SW_LK_CHNG_CLR
- ADV7183_SCM_LOCK_CHNG_CLR
- ADV7183_SD_AD_CHNG_CLR
- ADV7183_SD_H_LOCK_CHNG_CLR
- ADV7183_SD_V_LOCK_CHNG_CLR
- ADV7183_SD_OP_CHNG_CLR

Register Address: ADV7183_INT_MASK3
Register Fields:
- ADV7183_PAL_SW_LK_CHNG_MSKB
- ADV7183_SCM_LOCK_CHNG_MSKB
- ADV7183_SD_AD_CHNG_MSKB
- ADV7183_SD_H_LOCK_CHNG_MSKB
- ADV7183_SD_V_LOCK_CHNG_MSKB
- ADV7183_SD_OP_CHNG_MSKB

8. Appendix

8.1. Using ADV7183 Device Driver in Applications

This section explains how to use ADV7183 device driver with an application.

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 one TWI device + memory for one PPI device + memory for ADV7183 device + memory for other devices used by the application

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 1 DMA channel for PPI device + memory for DMA channels used for devices included in the application

Initialize Ez-Kit, Interrupt manager, Deferred Callback Manager, DMA Manager, Device Manager (all application dependent)

a. ADV7183 (driver) initialization

Step 1: Open ADV7183 Device driver with device specific entry point (refer section 6.1 for valid entry points)

Step 2: Set TWI device number

Step 3: Pass TWI Configuration table (refer section 8.2 for TWI configuration table examples)

Step 4: Set PPI device number to be used for ADV7183 video data flow
Example:
```
// Set ADV7183 to use PPI 0 for video dataflow
adi_dev_Control (ADV7183DriverHandle, ADI_ADV7183_CMD_OPEN_PPI, (void *) 0);
```

Step 5: Command PPI to operate in NTSC or PAL mode
Example:
```
// Set PPI to operate in PAL mode
adi_dev_Control (ADV7183DriverHandle, ADI_AD7183_CMD_SET_VIDEO_FORMAT, (void *) 1);
```

b. ADV7183 (hardware) initialization

Step 6: Set ADV7183 TWI device address
Example:
```
// set ADV7183 TWI device address
adi_dev_Control(ADV7183DriverHandle ADI_AD7183_CMD_SET_TWI_ADDRESS,
                    (void *) ADV7183_TWI_ADDR0);
```

Step 7: Configure ADV7183 device to specific mode using device access commands
(refer section 8.3.2 for examples)

c. Video Dataflow configuration

Step 8: Set video dataflow method

Step 9: Load ADV7183 video buffers

Step10: Enable ADV7183 video dataflow

d. Terminating ADV7183 driver

Step11: Terminate ADV7183 driver with adi_dev_Terminate( )

Terminate DMA Manager, Deferred Callback etc.., (application dependent)

8.2. TWI Configuration tables

This section contains TWI configuration table examples to access ADV7183 internal registers using BF533, BF537 and BF561 Ez-Kits

```
// Select TWI clock frequency & duty cycle (in this case its 100MHz & 50% Duty Cycle)
adi_twi_bit_rate        rate = { 100, 50 };
```

ADSP-BF533 EZ-KIT Lite & ADSP-BF561 EZ-KIT Lite

BF533 and BF561 do not have an inbuilt TWI peripheral. Analog Devices TWI device driver (adi_twi.c) can be configured in pseudo mode to mimic TWI operation with selected port pins and a timer. BF533 and BF561 Ez-Kits are designed to use PF0 and PF1 to generate TWI SCL and SDA signals respectively.

```
// BF533 TWI mimic pins and timer (PF0=SCL, PF1=SDA & General purpose Timer 0 used for pseudo TWI)
// BF561 TWI mimic pins and timer (PF0=SCL, PF1=SDA & General purpose Timer 2 used for pseudo TWI)\
#if defined (__ADSPBF533__)        // for BF533

adi_twi_pseudo_port     pseudo = { ADI_FLAG_PF0, ADI_FLAG_PF1, ADI_TMR_GP_TIMER_0,
                                   (ADI_INT_PERIPHERAL_ID) NULL };

#elif defined (__ADSPBF561__)      // for BF561

adi_twi_pseudo_port     pseudo = { ADI_FLAG_PF0, ADI_FLAG_PF1, ADI_TMR_GP_TIMER_3,
                                   (ADI_INT_PERIPHERAL_ID) NULL };

#endif

// Pseudo TWI configuration table
ADI_DEV_CMD_VALUE_PAIR  TWIConfig [ ] =  {
        { ADI_TWI_CMD_SET_PSEUDO,               (void *)(&pseudo)                   },
        { ADI_DEV_CMD_SET_DATAFLOW_METHOD,      (void *)ADI_DEV_MODE_SEQ_CHAINED    },
        { ADI_TWI_CMD_SET_FIFO,                 (void *)0x0000                      },
        { ADI_TWI_CMD_SET_RATE,                 (void *)(&rate)                     },
        { ADI_TWI_CMD_SET_LOSTARB,              (void *)1                           },
        { ADI_TWI_CMD_SET_ANAK,                 (void *)0                           },
        { ADI_TWI_CMD_SET_DNAK,                 (void *)0                           },
        { ADI_DEV_CMD_SET_DATAFLOW,             (void *)TRUE                        },
        { ADI_DEV_CMD_END,                      NULL                                }
        };
```

ADSP-BF537 EZ-KIT Lite

BF537 have an inbuilt TWI peripheral and the TWI device driver (adi_twi.c) can be configured to use hardware TWI

```
// Hardware TWI configuration table
ADI_DEV_CMD_VALUE_PAIR  TWIConfig [ ] = {
        { ADI_TWI_CMD_SET_HARDWARE,             (void *)ADI_INT_TWI                 },
        { ADI_DEV_CMD_SET_DATAFLOW_METHOD,      (void *)ADI_DEV_MODE_SEQ_CHAINED    },
        { ADI_TWI_CMD_SET_FIFO,                 (void *)0x0000                      },
        { ADI_TWI_CMD_SET_LOSTARB,              (void *)1                           },
        { ADI_TWI_CMD_SET_RATE,                 (void *)(&rate)                     },
        { ADI_TWI_CMD_SET_ANAK,                 (void *)0                           },
        { ADI_TWI_CMD_SET_DNAK,                 (void *)0                           },
        { ADI_DEV_CMD_SET_DATAFLOW,             (void *)TRUE                        },
        { ADI_DEV_CMD_END,                      NULL                                }
        };
```

8.3. Accessing ADV7183 registers

This section explains how to access the ADV7183 internal registers using driver specific commands and device access commands (refer *'deviceaccess'* documentation for more information).

For ADV7183 register map details, refer to ADV7183-decoder datasheet.
- Common register address 0x00 to 0x3F. (refer to Table 6)
- Page 1(Normal) register address 0x40 to 0xF9. (refer to Table 7)
- Page 2(Interrupt) register address 0x40 to 0x4C. (refer to Table 8)

Depending of the bit field ADV7183_SUB_USR_EN value (0 or 1) of the ADV7183_ADI_CTR register, the page 1 or page 2 registers can be accessed by the user application program.

Using register access command with register macro name and field bit, the user does not have to manually set or clear the bit field ADV7183_SUB_USR_EN of the ADV7183_ADI_CTR register to access page 1 or 2 registers.
The device driver sets automatically the bit field ADV7183_SUB_USR_EN of the ADV7183_ADI_CTR register to 0 if page 1 register is accessed or set to 1 if page 2 register is accessed.

8.3.1. Read ADV7183 internal registers

1. Read a single register

```
// define the structure to access a single device register
ADI_DEV_ACCESS_REGISTER Read_Reg;

// Load the register address to be read
Read_Reg.Address = ADV7183_IDENT_RO;

//clear the Data location
Read_Reg.Data = 0;
// Application calls adi_dev_Control( ) function with corresponding command and value
// Register value will be read back to location - Read_Reg.Data
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_READ, (void *) & Read_Reg);
```

2. Read a specific register field

```
// define the structure to access a specific device register field
ADI_DEV_ACCESS_REGISTER_FIELD Read_Field;

// Load the device register address to be accessed
Read_Field.Address = ADV7183_STATUS1_RO;
// Load the device register field location to be read
Read_Field.Address = ADV7183_IN_LOCK;

//clear the Read_Field.Data location
Read_Field.Data = 0;
// Application calls adi_dev_Control( ) function with corresponding command and value
//The register field value will be read back to location - Read_Field.Data
adi_dev_Control (DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_READ, (void *) & Read_Field);
```

3. Read table of registers

```
// define the structure to access table of device registers
ADI_DEV_ACCESS_REGISTER Read_Regs [ ] = {
                                {ADV7183_STATUS1_RO,      0},
                                {ADV7183_IDENT_RO,        0},
                                {ADV7183__STATUS2_RO,     0},
   /*MUST include delimiter */  {ADI_DEV_REGEND,   0}      // Register access delimiter
                                };

// Application calls adi_dev_Control( ) function with corresponding command and value
// Present value of registers listed above will be read to corresponding Data location in Read_Regs array
// i.e., value of ADV7183_STATUS1_RO will be read to Read_Regs[0].Data,
// ADV7183_IDENT_RO to Read_Regs[1].Data  and ADV7183_STATUS2_RO to Read_Regs[2].Data
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_READ, (void *) &Read_Regs[0]);
```

4. Read table of register(s) fields

```
// define the structure to access table of device register(s) fields
ADI_DEV_ACCESS_REGISTER_FIELD Read_Fields [ ] = {
                            {ADV7183_STATUS1_RO,      ADV7183_IN_LOCK,          0},
                            {ADV7183_ STATUS3_RO,     ADV7183_INTERLACE,        0},
                            {ADV7183_ STATUS3_RO,     ADV7183_STDFLDLEN,        0},
/*MUST include delimiter */ {ADI_DEV_REGEND,          0,        0}      // Register access delimiter
                            };

// Application calls adi_dev_Control( ) function with corresponding command and value
// Present value of register fields listed above will be read to corresponding Data location in Read_Fields array
// i.e., value of ADV7183_IN_LOCK will be read to Read_Fields[0].Data,
// ADV7183_INTERLACE to Read_Fields [1].Data and ADV7183_STDFLDLEN to Read_Fields [2].Data
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_READ, (void *) & Read_Fields [0]);
```

5. Read block of registers

```
// define the structure to access a block of registers
ADI_DEV_ACCESS_REGISTER_BLOCK Read_Block;

// load the number of registers to be read
Read_Block.Count = 4;
// load the starting address of the register block to be read
Read_Block.Address = ADV7183_STATUS1_RO;
// define a 'Count' sized array to hold register data read from the device
u16 Block_Data[4] = { 0 };
// load the start address of the above array to Read_Block data pointer
Read_Block.pData = & Block_Data [0];

// Application calls adi_dev_Control( ) function with corresponding command and value
// Present value of the registers in the given block will be read to corresponding Block_Data[ ] array
// value of ADV7183_STATUS1_RO will be read to Block_Data [0], ADV7183_IDENT_RO to Block_Data[1],
// ADV7183_STATUS2_RO to Block_Data[2] and ADV7183_STATUS3_RO to Block_Data[3]
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_BLOCK_READ, (void *) & Read_Block);
```

8.3.2. Configure ADV7183 internal registers

1. Configure a single ADV7183 register

```
// define the structure to access a single device register
ADI_DEV_ACCESS_REGISTER Cfg_Reg;

// Load the register address to be configured
Cfg_Reg.Address = ADV7183_EXTENDED_OUTPUT_CTR;

//Load the configuration value to Cfg_Reg.Data location
Cfg_Reg.Data = 0x44;
// Application calls adi_dev_Control( ) function with corresponding command and value
//The device register will be configured with the value in Cfg_Reg.Data
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_WRITE, (void *) & Cfg_Reg);
```

2. Configure a specific register field

```
// define the structure to access a specific device register field
ADI_DEV_ACCESS_REGISTER_FIELD Cfg_Field;

// Load the device register address to be accessed
Cfg_Field.Address = ADV7183_EXTENDED_OUTPUT_CTR;
// Load the device register field location to be configured
Cfg_Field.Address = ADV7183_RANGE;

// load the new field value
Cfg_Field.Data = 0;
// Application calls adi_dev_Control( ) function with corresponding command and value
// Selected register field will be configured with the value in Cfg_Field.Data
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_WRITE, (void *) & Cfg_Field);
```

3. Configure table of registers

```
// define the structure to access table of device registers (register address, register configuration value)
ADI_DEV_ACCESS_REGISTER Cfg_Regs [ ] = {
                                {ADV7183_INPUT_CTR,             0x00},
                                {ADV7183_VIDEO_SELECTION,       0xC8},
                                {ADV7183_OUTPUT_CTR,            0x0C},
   /*MUST include this*/        {ADI_DEV_REGEND,   0    }    };      // Register access delimiter


// Application calls adi_dev_Control( ) function with corresponding command and value
// Registers listed in the table will be configured with corresponding table Data values
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *) & Cfg_Regs[0]);
```

4. Configure a table of register(s) fields

```
// define the structure to access table of device register(s) fields
// register address, register field to configure, field configuration value
ADI_DEV_ACCESS_REGISTER_FIELD Cfg_Fields [ ] = {
                                {ADV7183_INPUT_CTR, ADV7183_VID_SEL,            1},
                                {ADV7183_VIDEO_SELECTION, ADV7183_BETACAM, 1},
                                {ADV7183_OUTPUT_CTR, ADV7183_OF_SEL,            2},
   /*MUST include this*/        {ADI_DEV_REGEND,   0    }    };      // Register access delimiter
```

```
// Application calls adi_dev_Control( ) function with corresponding command and value
// Register fields listed in the above table will be configured with corresponding Data values
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *) & Cfg_Fields [0]);
```

5. Configure a block of registers

```
// define the structure to access a block of registers
ADI_DEV_ACCESS_REGISTER_BLOCK Cfg_Block;

// load the number of registers to be configured
Cfg_Block.Count = 3;
// load the starting address of the register block to be configured
Cfg_Block.Address = ADV7183_INPUT_CTR;

// define a 'Count' sized array to hold register data read from the device
u16 Block_Cfg [3]   = { 0x00, 0xC8, 0x0C };

// load the start address of the above array to Cfg_Block data pointer
Cfg_Block.pData = & Block_Cfg [0];

// Application calls adi_dev_Control( ) function with corresponding command and value
// Registers in the given block will be configured with corresponding values in Block_Cfg[ ] array
adi_dev_Control (DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *) &Cfg_Block);
```