ANALOG DEVICES

UG-556
Software User Guide

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

Transmitter Library API Specifications for HDMI/DVI/Analog Interfaces

Analog Devices Inc.

PRODUCT OVERVIEW

The Transmitter library API is used as a programmer's reference for using and utilizing various aspects of the Analog Devices, Inc., HDMI/DVI/analog products.

PURPOSE OF THIS MANUAL

The purpose of the *Transmitter Library API Specifications* document is to provide a reference for using various aspects of the Transmitter library API , which is a module of the Advantiv® software stack. It provides a detailed list of Application Program Interface (APIs) and describes their associated data structures, macros and preprocessor definitions.

This document is intended to be used in conjunction with the Advantiv® software architecture specifications document.

SCOPE OF THIS MANUAL

The *Transmitter LibraryAPI Specifications* document only describes the API specification and any parts of the Transmitter library that are particular to the TX module. For a complete description of the Advantiv® software stack, refer to the Advantiv® software architecture specifications document.

INTENDED AUDIENCE

The primary audience for this manual is software developers who are building the interface between the hardware registers and their application.

WHAT'S NEW IN THIS MANUAL

Refer to the Revision History.

MANUAL CONTENTS

The manual consists of:

- Chapter 1, Software Architecture on Page 7.
  Provides information about the Transmitter library and its folder structure.
- Chapter 2, Configuration of Transmitter Library on Page 9
  Provides information for configuring transmitter module parameters.
- Chapter 3, Tx Library Usage on Page 11
  Provides information about how to use the TX library from the application view point
- Chapter 4, Description of APIs on Page 13
  Provides information about all the APIs used in the TX module.
- Chapter 5, Notification Events on Page 57

TECHNICAL OR CUSTOMER SUPPORT

You can reach Analog Devices, Inc. Customer Support in the following ways:

- Visit the Audio/Video Products Web site at http://www.analog.com/en/audiovideo-products/analoghdmidvi-interfaces/products/index.html
- Post questions at http://ez.analog.com/community/video
- Phone questions to **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:
  Analog Devices, Inc.
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA

Advantiv®
Advanced Television Solutions
by Analog Devices

TABLE OF CONTENTS

Product Overview ... 1
Purpose of This Manual ... 1
Scope of This Manual ... 1
Intended Audience ... 1
What's New in This Manual ... 1
Manual Contents ... 1
Technical or Customer Support ... 1
Revision History ... 3
Product Information ... 5
Analog Devices Web Site ... 5
EngineerZone ... 5
Related Documents ... 5
Copyright Information ... 6
Disclaimer ... 6
Trademark and Service Mark Notice ... 6
1 Software Architecture ... 7
1.1 Introduction to Transmitter Library ... 7
1.2 Folder Structure ... 8
1.2.1 Folders Descriptions ... 8
2 Configuration of Transmitter Library ... 9
2.1 Compilation Switches ... 9
2.2 User-Supplied Configuration Files ... 9
2.3 User-Supplied Initialization Data ... 10
3 Tx Library Usage ... 11
3.1 Transmitter Library Compilation ... 11
3.2 Sample Application ... 11
3.3 Multiple Device Support ... 11
4 Description of APIs ... 13
4.1 ADIAPI_TxSetDeviceIndex ... 13
4.2 ADIAPI_TxGetDeviceIndex ... 13
4.3 ADIAPI_TxInit ... 14
4.4 ADIAPI_TxShutdown ... 14
4.5 ADIAPI_TxGetChipRevision ... 15
4.6 ADIAPI_TxSetEnabledEvents ... 15
4.7 ADIAPI_TxIsr ... 16
4.8 ADIAPI_TxIntPending ... 17
4.9 ADIAPI_TxSetConfig ... 17
4.10 ADIAPI_TxOverrideHpdPD ... 18
4.11 ADIAPI_TxEnableTMDS ... 18
4.12 ADIAPI_TxSetInputPixelFormat ... 19
4.13 ADIAPI_TxSetInputVideoClock ... 20
4.14 ADIAPI_TxSetOutputPixelFormat ... 20
4.15 ADIAPI_TxSetManualPixelRepeat ... 21
4.16 ADIAPI_TxSetAutoPixelRepeat ... 21
4.17 ADIAPI_TxSetOutputColorDepth ... 22
4.18 ADIAPI_TxSetCSC ... 23
4.19 ADIAPI_TxSetAudioInterface ... 24
4.20 ADIAPI_TxSetAudChanMapping ... 26
4.21 ADIAPI_TxSetAudNValue ... 27
4.22 ADIAPI_TxSetAudCTS ... 27
4.23 ADIAPI_TxSetAudMCLK ... 28
4.24 ADIAPI_TxSetAudClkPolarity ... 28
4.25 ADIAPI_TxSetAudChStatSampFreq ... 29
4.26 ADIAPI_TxSetAudChanStatus ... 30
4.27 ADIAPI_TxAudInputEnable ... 31
4.28 ADIAPI_TxSetI2sInput ... 32
4.29 ADIAPI_TxSetOutputMode ... 32
4.30 ADIAPI_TxHdcpEnable ... 33
4.31 ADIAPI_TxGetBksvList ... 33
4.32 ADIAPI_TxGetBstatus ... 34
4.33 ADIAPI_TxGetHdcpState ... 34
4.34 ADIAPI_TxGetLastHdcpError ... 35
4.35 ADIAPI_TxGetEdidSegment ... 35
4.36 ADIAPI_TxGetHpdMsenState ... 36
4.37 ADIAPI_TxGetEdidControllerState ... 36
4.38 ADIAPI_TxOutputModeHdmi ... 37
4.39 ADIAPI_TxOutputEncrypted ... 37
4.40 ADIAPI_TxPllLocked ... 37
4.41 ADIAPI_TxGetStatus ... 38
4.42 ADIAPI_TxMuteAudio ... 40
4.43 ADIAPI_TxMuteVideo ... 40
4.44 ADIAPI_TxSetAvmute ... 40
4.45 ADIAPI_TxGetAvmute ... 41
4.46 ADIAPI_TxEnablePackets ... 41
4.47 ADIAPI_TxGetEnabledPackets ... 42
4.48 ADIAPI_TxSendAVInfoframe ... 43
4.49 ADIAPI_TxSendAudioInfoframe ... 44
4.50 ADIAPI_TxSendACPPacket ... 44
4.51 ADIAPI_TxSendSPDPacket ... 45
4.52 ADIAPI_TxSendISRC1Packet ... 45

4.53 ADIAPI_TxSendISRC2Packet....46
4.54 ADIAPI_TxSendGMDPacket....46
4.55 ADIAPI_TxSendMpegPacket....47
4.56 ADIAPI_TxSendSpare1Packet....47
4.57 ADIAPI_TxSendSpare2Packet....48
4.58 ADIAPI_TxArcSetMode....48
4.59 ADIAPI_TxCecSetActiveDevice....49
4.60 ADIAPI_TxCecEnable....49
4.61 ADIAPI_TxCecReset....49
4.62 ADIAPI_TxCecSendMessage....50
4.63 ADIAPI_TxCecResendLastMessage....50
4.64 ADIAPI_TxCecReadMessage....51
4.65 ADIAPI_TxCecSetLogicalAddr....51
4.66 ADIAPI_TxCecAllocateLogAddr....52
4.67 ADIAPI_TxCecGetStatus....52
4.68 ADIAPI_TxSetVideoClkDelay....53
4.69 ADIAPI_TxHDCPEnabled....54
4.70 ADIAPI_TxCecSendMessageOut....54
4.71 ADIAPI_MonitorActiveTx....54
4.72 ADIAPI_TxNormalOp....55
4.73 ADIAPI_TxAdjustFreqRange(UINT16 FreqMhz)....55
4.74 ADIAPI_TxSetAutoTmdsPdnMode....55
4.75 ADIAPI_TxDisableAutoPjCheck....56
5 Notification Events....57

REVISION HISTORY

5/13—Revision 0: Initial Analog Devices Published Version

Revision	Date	Remarks
1.1	August 14 2008	Initial revision
1.2	October 28 2008	Changes to reflect ATV new software structure
1.3	November 14 2008	Updates to reflect implementation
1.4	December 10 2008	Added multi device support
1.5	January 19 2009	Added API to get device Index
1.6	September 14 2009	Added one parameter to TX init API to select init state Added ADIAPI_TxSendMpegPacket Added ADIAPI_TxSendSpare1Packet
1.7	February 16, 2010	Added ADIAPI_TxArcSetMode
1.8	March 31 2010	Added the following API functions: ADIAPI_TxIntPending ADIAPI_TxSetI2sInput ADIAPI_TxGetEdidControllerState ADIAPI_TxOutputEncrypted ADIAPI_TxPllLocked ADIAPI_GetAvmute ADIAPI_SendSpare2Packet ADIAPI_TxSetVideoClkDelay Updated return code for ADIAPI_TxSendxxPacket API functions
1.9	February 18 2011	Added the following API functions: ADIAPI_TxHDCPEnabled ADIAPI_TxCECSendMessageOut ADIAPI_MoitorActiveTx ADIAPI_InitTxDevsN ADIAPI_CntTxDevsNEDID ADIAPI_InformTXNVideoStatus ADIAPI_InformTXNOperMode ADIAPI_InformTXNMuteStatus
1.10	March 21 2011	Added remarks for ADV7850 support

Revision	Date	Remarks
2.0	February 12 2012	Updated for Release 1.60 Added the following APIs: ADIAPI_TxSetAutoTmdsPdnMode ADIAPI_TxAdjustFreqRange ADIAPI_TxNormalOp
2.1	February 22, 2013	Removed the following APIs as they are no longer used in software: ADIAPI_InitTxDevsN ADIAPI_CntTxDevsNEDID ADIAPI_InformTXNVideoStatus ADIAPI_InformTXNOperMode ADIAPI_InformTXNMuteStatus Added the below API: ADIAPI_TxDisableAutoPjCheck
2.2	April 4, 2013	Updated the folder structure in section 1.2, typos corrected.

PRODUCT INFORMATION

Product information can be obtained from the Analog Devices Web site and other Web sources.

Analog Devices Web Site

The Analog Devices Web site, www.analog.com, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to http://www.analog.com/processors/technical_library. The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to Analog Devices technical support engineers. You can search FAQs and technical information to get quick answers to your questions about Analog Devices video products.

RELATED DOCUMENTS

Title	Description
Receiver Library API Specifications	Describes the Receiver library specifications
Advantiv® *Software Architecture Specifications*	Describes the Advantiv® software stack

COPYRIGHT INFORMATION

DISCLAIMER

Analog Devices, Inc. (ADI) reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties, which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

The information contained in this document is proprietary of ADI. This document must not be made available to anybody other than the intended recipient without the written permission of ADI.

The content of this document is believed to be correct. If any errors are found within this document or if clarification is needed, contact the video community on EZ forum (http://ez.analog.com/community/video).

TRADEMARK AND SERVICE MARK NOTICE

The Analog Devices logo is a registered trademark of Analog Devices, Inc. The Advantiv® is a registered trademark of Analog Devices Inc. All other brand and product names are trademarks or service marks of their respective owners.

All other brand and product names are trademarks or service marks of their respective owners.

Analog Devices' Trademarks and Service Marks may not be used without the express written consent of Analog Devices, such consent only to be provided in a separate written agreement signed by Analog Devices. Subject to the foregoing, such Trademarks and Service Marks must be used according to Analog Devices' Trademark Usage guidelines. Any licensee wishing to use Analog Devices' Trademarks and Service Marks must obtain and follow these guidelines for the specific marks at issue.

1 SOFTWARE ARCHITECTURE

This chapter provides information about the Transmitter library and its folder structure.

The following topics are covered:

- Introduction to Transmitter Library on Page 7
- Folder Structure on Page 8

1.1 INTRODUCTION TO TRANSMITTER LIBRARY

The Transmitter library is a collection of APIs that provide a consistent interface to a variety of HDMI TX hardware modules. The APIs are designed so that they can be used on different, closely-related, HDMI TX chip families, thus enhancing application portability.

The library is a software layer that sits between the application and the TX hardware. The library is intended to serve the following purposes:

- Provide the application with a set of APIs that can be used to configure HDMI TX hardware without the need for low-level register access. This makes the application portable across different revisions of the hardware and even across different hardware modules.
- Provide basic services to aid the application in controlling the TX module, such as interrupt service routine, HDCP high-level control and status information.

The library does not, in any shape or form, alter the configuration or state of the HDMI TX module on its own. It is the responsibility of the application to poll for status, prepare and send InfoFrames, parse EDID, and configure the HDMI TX module accordingly. The library acts only as an abstraction layer between the application and the hardware.

As an example, the application is responsible for the following:

- Parsing EDID
- Allocating CEC logical address
- Parsing CEC messages
- Responding to CEC messages
- Configuring the input/output color space
- Configuring audio interface
- Configuring video interface
- Handling of pixel repeat modes
- Muting audio and/or video
- Preparing and sending HDMI packets and InfoFrames

The application should access the TX module only through the TX library's exported APIs. This enhances application portability and reduces dependencies on any particular TX hardware. If the application chooses to directly access TX module hardware (for example, using I^2C or TX HAL macros,) this should be done in a very limited scope, when it is absolutely necessary, and it should be understood that this practice will effectively reduce application portability.

1.2 FOLDER STRUCTURE

The collective files of the Transmitter library reside in a sub-folder under the Advantiv® software main folder. The library is supplied in source format. All source files are in standard ANSI C to simplify porting to any platform. The Transmitter library folder is structured as shown in Figure 1.

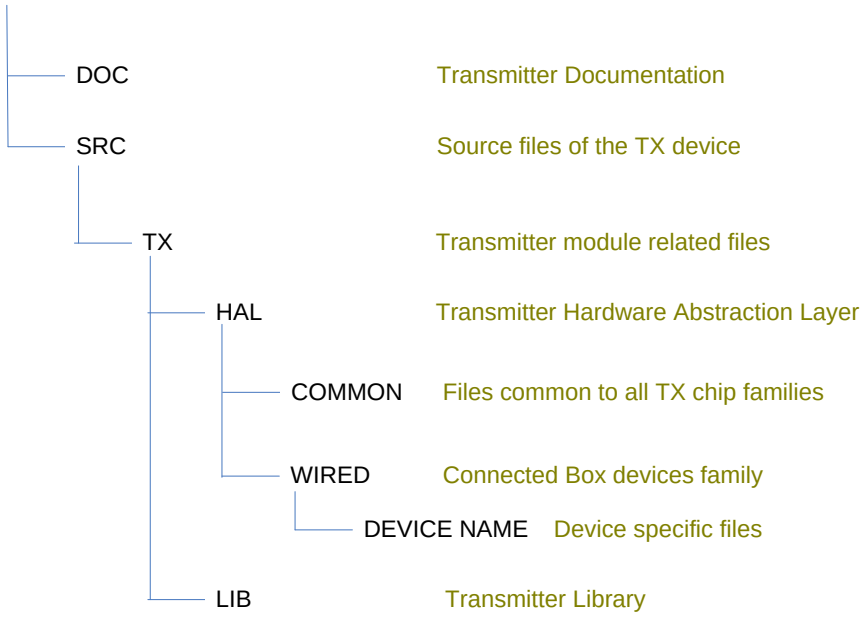

Figure 1. Folder Structure

1.2.1 *Folders Descriptions*

/ DOC

This folder contains all TX module documentations.

/SRC

This folder contains the source file of the TX device.

/SRC/ TX

The TX module root folder contains all the header files needed by upper level layers (the application) to access the Transmitter library. This is the only folder in the TX folders tree that should be 'seen' by the application. Only one file, *tx_lib.h*, exists in this folder. This file contains all necessary prototypes, macros and defines needed to access the TX software library.

/TX / HAL

This folder contains the TX Hardware Abstraction Layer files and sub-folders. The transmitter HAL APIs and macros should not be referenced directly by the application.

/TX / HAL / COMMON

This folder contains the TX HAL files that are common to all TX chip families.

/TX / HAL / WIRED

This folder contains the generic TX HAL files for wired (Connected Box) TX devices.

/TX / HAL / WIRED / DEVICE NAME

This folder contains the TX module low-level access macros and functions that are particular to the TX device chip family. The device name can be either ADV7510 or ADV7511 or ADV7513.

/TX / LIB

This folder contains the TX module library exported APIs and interrupt service routine.

2 CONFIGURATION OF TRANSMITTER LIBRARY

This chapter provides information for configuring transmitter module parameters.

Some of the transmitter module parameters can be configured by the application at compile time. Parameters such as I²C device addresses, inclusion of CEC support, maximum supported downstream device count, and so on, can be specified by the application, if required.

A combination of compile-time switches and user-supplied configuration files are used to configure the transmitter library as described in the following sections.

The following topics are covered:

- Compilation Switches on Page 9
- User-Supplied Configuration Files on Page 9
- User-Supplied Initialization Data on Page 10

2.1 COMPILATION SWITCHES

Some parameters of the Transmitter library are configured through compiler switches (symbols). The compiler switches required to compile the Transmitter library are described here.

TX_DEVICE = 7510

This is the target TX device. Example value is 7510 for ADV7510.

TX_USER_CONFIG = 1

This switch is used to enable the application to configure the transmitter library at compile time using a configuration file. If set to 1, the transmitter library will include an application-supplied header file named *tx_config.h* that contains all the configuration parameters needed by the library. This file must reside outside the transmitter folders tree (typically in the application folder) and the path for this file must be set in the compiler include path. If this switch is set to 0, the application does not need to provide the *tx_config.h* file and a default value for all configuration parameters will be used. Section 2.2 on Page 9 describes the structure of the transmitter configuration file and the default values for all configuration parameters.

TX_USER_INIT = 1

This switch is used to enable the application to initialize the transmitter hardware with certain values during runtime. If this switch is set to 1, in addition to the default initialization sequence, the library will initialize the transmitter hardware using a set of application-supplied tables containing register values that need to be set during initialization. The structure of this table is defined in Section 2.3 on Page 10. This initialization data must reside outside the transmitter folders tree (typically in the application folder). If this switch is set to 0, the application does not need to provide the initialization tables and only the default initialization values will be used. It should be noted that the use of application-supplied initialization data is highly discouraged as it binds the application to a specific TX hardware module. This feature is provided as a last resort to initialize the TX hardware when all else fails.

Note that in Analog Devices example applications these values are set in a header file called *atv_preprocessor.h*. When integrating Analog Devices middleware or example applications the compilation switch may be adjusted in this file.

2.2 USER-SUPPLIED CONFIGURATION FILES

If the compilation switch TX_USER_CONFIG is set to 1 (refer to Section 2.1 on Page 9), the application must provide a header file named *tx_config.h* that contains TX module compile-time configuration settings that need to be adjusted according to the target application or platform.

The configuration parameters that need to be defined in the *tx_config.h* file are described below along with the default values that will be used if the TX_USER_CONFIG switch is set to 0.

Note that in systems with multiple TX devices (and depending on the type of TX device in use,) the sub-devices map addresses should be defined to values other than default, and no other device in the system should use the default map values. The reason is that some TX devices when powered down (on HPD high-to-low transition) will revert to the default values for the sub-device addresses, and some registers in those maps will still be available on the I2C bus, which will lead to conflict with any other device (be it another TX or any other I2C device) that is assigned to one of the default values.

```
#define TX_I2C_MAIN_MAP_ADDR              0x72
#define TX_I2C_PKT_MEM_MAP_ADDR           0x70
```

```
#define TX_I2C_CEC_MAP_ADDR              0x78
#define TX_I2C_EDID_MAP_ADDR             0x7E
```

The above macros define the I2C device addresses of all HDMI TX sub-device maps. All sub-devices I2C base addresses are configurable except the TX_MAIN_MAP_ADDR which is hardware fixed and cannot be changed (only two selections are available through the hardware setting). The above addresses can be changed if it conflicts with another I2C device in the system.

```
#define TX2_I2C_MAIN_MAP_ADDR            0x7A
#define TX2_I2C_PKT_MEM_MAP_ADDR         0x76
#define TX2_I2C_CEC_MAP_ADDR             0x82
#define TX2_I2C_EDID_MAP_ADDR            0x86
```

The above macros define the I2C device addresses of secondary HDMI TX sub-device maps. All sub-devices I2C base addresses are configurable except the TX2_MAIN_MAP_ADDR which is hardware fixed and cannot be changed. The above addresses can be changed if it conflicts with another I2C device in the system. If the system contains only one HDMI TX device, the above set of addresses is irrelevant.

```
#define TX_NUM_OF_DEVICES                1
```

This macro defines the number of HDMI TX devices in the system. Currently only one HDMI TX device is supported.

```
#define TX_INCLUDE_CEC                       1
```

Set to 1 to include CEC support in the library. If CEC is not required, set this macro to 0 to remove CEC support. Removing CEC support will reduce overall code size.

```
#define TX_SUPPORTED_DS_DEVICE_COUNT    10
```

This macro defines the maximum number of HDCP downstream devices supported by the library. The number of downstream BKSV list reported by the library to the upper level layers will never exceed this value even if the number of BKSVs reported by the downstream device (including DS repeater) exceeds this value.

2.3 USER-SUPPLIED INITIALIZATION DATA

The TX library provides the ADIAPI_TxInit API to initialize the TX hardware module. Some of the TX module registers are initialized by this API to fixed values that cannot be controlled directly from the application. However, the application can specify additional settings to any of the TX module registers by setting the TX_USER_INT compilation switch to 1 (refer to Section 2.1 on Page 9 for details)

When TX_USER_INIT is set to 1, the application must define two tables containing TX registers initialization sequences. Those tables will be used by the ADIAPI_TxInit API to perform additional TX hardware initialization after completing the default initialization sequence.

The structure of the two tables is as follows:

UCHAR UserTxRegInitTable [] = {device, register, value, …, 0, 0, 0};

UCHAR UserTxFieldInitTable[] = {device, register, mask, value, …, 0, 0, 0, 0};

The first table is used to initialize a set of TX registers. The table consists of a concatenated list of 3-byte entries, where each entry specifies the required setting for one register. Each entry starts with the I2C device address, followed by the register address and the value to be written to that register. The next 3 bytes represent the initialization value for the next register and so on. The end of the table is marked by a three 0 bytes.

The second table is used to initialize a set of TX registers fields. The table consists of a concatenated list of 4-byte entries, where each entry specifies the required setting for a register field. Each entry starts with the I2C device address, followed by the register address, followed by the filed mask then the field value to be written to that register. The next 4 bytes represent the initialization value for the next field and so on. The end of the table is marked by a four 0 bytes. An example is defined below:

```
UCHAR UserTxFieldInitTable[] = {
0x72, 0x14, 0xF0, 0x90,    /* Set device 0x72 register 0x14 bits 4-7 = 0x09 */
0x72, 0x55, 0x60, 0x20,    /* Set device 0x72 register 0x55 bits 5-6 = 0x01 */
0x00, 0x00, 0x00, 0x00           /* Table end */
};
```

It should be noted that the use of application-supplied initialization data is highly discouraged as it binds the application to a specific TX hardware module. This feature is provided only as a last resort to initialize the TX hardware when all else fails.

3 TX LIBRARY USAGE

This chapter provides information about how to use the TX library from the application view point.

The following topics are covered:

- Transmitter Library Compilation on Page 11
- Sample Application on Page 11
- Multiple Device Support on Page 11

3.1 TRANSMITTER LIBRARY COMPILATION

The Transmitter library is designed to be a self-contained module. The library, however, relies on the platform hardware abstraction layer (the HAL, see the Advantiv® software architecture specifications document) to gain access to the underlying hardware.

It is the responsibility of the library's user to provide platform HAL APIs, either in source format or as a binary module to be linked with the transmitter library. All the data types used by the library must also be provided as specified in the Advantiv® software architecture specifications document.

The following steps are required to compile the transmitter library.

1. Provide platform HAL APIs and data types
2. Set the compilation switches as described in Section 2.1 on Page 9.
3. Create the notification events handling function as described in Section 5 on Page 57.
4. If required, create the user-configuration file and user-initialization tables as described in Section 2.2 on Page 9 and Section 2.3 on Page 10.

3.2 SAMPLE APPLICATION

The following steps are required to be taken by the application at run time.

1. Initialize the Transmitter hardware and software stack by calling the ADIAPI_TxInit API.
2. Enable the events that the application wishes to be notified about, by calling the ADIAPI_TxSetEnabledEvents API.
3. Configure the Transmitter module using the library APIs. This can include, for example, setting the input/output color space, setting the audio and video interface, enabling/disabling HDCP, and so on.
4. If a transmitter interrupt is detected, the application should call the Transmitter ISR ADIAPI_TxIsr API to process the interrupt. Since the transmitter state usually changes following an interrupt, the application should either enquire about transmitter state following interrupt processing or use notification events handling function to take the proper action on certain events. Refer to Section 5 on Page 57 and ADIAPI_TxSetEnabledEvents API for more details.

3.3 MULTIPLE DEVICE SUPPORT

The API library supports multiple TX devices. This means all the APIs provided by the library can be used to configure one or more TX devices at any given time. For a system with multiple devices, the library provides three options to do just that:

- The APIs can be applied to a single, pre-selected, TX device. The pre-selected TX device can be set by calling the ADIAPI_TxSetDeviceIndex API and will remain selected for all subsequent API calls until changed by another call to ADIAPI_TxSetDeviceIndex. This method has the advantage of not having to specify a device for each API call.
- The pre-selected device can be overridden for any API call by specifying the device index for which the API is intended in the first (additional) parameter to the API. The prototype of this type of API is identical to the single-device equivalent with two exceptions: The API name is suffixed with the letter 'N', and the first (additional) parameter of the API is a UCHAR specifying the target device Index. The pre-selected device (as defined above) remains unchanged for all subsequent API calls.
- This option is similar to option 2 above except that the API can be applied to all devices in the system by selecting a device index of TX_DEV_ALL. As in option 2, the API name is suffixed with 'N' and the device index is specified in the first (additional) parameter to the API. All other parameters remain the same as for single device APIs. The pre-selected device remains unchanged for all subsequent API calls.

The following examples illustrate how to use the APIs for all of the above methods:

```
UCHAR Rev0, Rev1;
ADIAPI_TxSetDeviceIndex(0);                    /* Pre-select device number 0    */
ADIAPI_TxInitN(TX_DEV_ALL);                    /* Initialize all devices        */
ADIAPI_TxEnableTmds(TRUE, TRUE);         /* Enable TMDS for device 0            */
ADIAPI_TxGetChipRevisionN(1, &Rev1);     /* Get device 1 revision         */
                                               /* Device 0 is still pre-selected*/
ADIAPI_TxGetChipRevision(&Rev0);         /* Get device 0 revision         */
ADIAPI_TxEnableTmdsN(1, TRUE, TRUE);     /* Enable TMDS for device 1            */
```

Notes:

- APIs that are used to retrieve device information, such as ADIAPI_TxGetChipRevision, cannot use the TX_DEV_ALL as a device index.
- CEC APIs do not have multiple devices support (only the single device version is available) since only one CEC engine is allowed to be active at any given time.

4 DESCRIPTION OF APIs

This chapter provides information about all the APIs used in the TX module.

The Transmitter library provides a comprehensive set of APIs to control, configure and provide status on all aspects of the HDMI TX module. All library APIs are available for the application software to use at any time.

For a description of the Advantiv® APIs structure, data types and return values, refer to the Advantiv® software architecture specification document.

All APIs are listed in the format used for a single TX device. For multiple device support, refer to Section 3.3 on Page 11.

4.1 ADIAPI_TXSETDEVICEINDEX

Description

The TX library supports multiple TX devices as defined in Section 2.2 on Page 9. When the application calls any of the library APIs, the called API apply to the current (selected) device only. The selected device is an index of the TX device in the current system. For systems with only one TX device, the index is 0 and cannot be changed. For systems with multiple TX devices, the index can range from 0 to (TX_NUM_OF_DEVICES – 1).

This API is used to select which TX device the API call will affect. The selected device remains in effect until it is changed by another call to this API. Refer to Section 3.3 on Page 11 for more information.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetDeviceIndex (*UCHAR* DevIdx)

Parameters

DevIdx

Index of the device to be used by subsequent API calls. The range is 0 to (TX_NUM_OF_DEVICES – 1)

Return Value

ATVERR_OK	Operation completed successfully
ATVERR_INV_PARM	If the device index is out of range

Remarks

None.

4.2 ADIAPI_TXGETDEVICEINDEX

Description

For systems will multiple TX devices, this API can be used to return the current TX device the library is processing at any given time. This is usually the device index set by the ADIAPI_TxSetDeviceIndex API. However, during notification call-back (refer to Section 5 on Page 57) the device being processed may differ from the one selected by the application. This API can be used to identify which TX device the notification call-back is coming from. Refer to Section 3.3 on Page 11 for more information.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetDeviceIndex (*UCHAR* *DevIdx)

Parameters

DevIdx

This pointer will receive the index of the device currently being processed by the library. The range is 0 to (TX_NUM_OF_DEVICES – 1).

Return Value

ATVERR_OK

Operation completed successfully

Remarks

None.

4.3 ADIAPI_TXINIT

Description

Power-up and initialize HDMI TX hardware and software module. This function will perform a complete TX module reset and brings the chip into a known state. The default behavior of the HDMI TX chip is to automatically power down if HPD is low. To change this default behavior, the ADIAPI_TxOverrideHpdPD API can be used.

Additional application-defined initialization can be done by this API as described in Section 2.3 on Page 10.

The TMDS clock and data lines will be disabled following a call to this API (or whenever the system is initialized) unless the configuration flag TX_ENABLE_TMDS_ON_INIT is set as described in the ADIAPI_TxSetConfig API.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxInit (*BOOL* FullInit)

ATV_ERR ADIAPI_TxInitN (*UCHAR* DeviceIndex, *BOOL* FullInit)

Parameters

FullInit

Select if it is required to perform full initialization (all h/w modules will be reset) or partial initialization (CEC module will not be affected)

Set to TRUE to perform full initialization. This should be used when doing a cold start.

Set to FALSE to perform partial initialization. This should be used for warm start (for example coming out of standby) to preserve the state of the CEC engine.

Return Value

ATVERR_OK	Operation completed successfully
ATVERR_FAILED	If the chip is powered down

Remarks

See also ADIAPI_TxShutdown.

See also ADIAPI_TxSetConfig for TMDS clock and data line handling following a reset.

4.4 ADIAPI_TXSHUTDOWN

Description

Power down HDMI TX hardware. The chip power-down will be set to high and all TMDS lines will be disabled. This API can also be used to enter stand-by mode.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxShutdown (*TX_PD_MODE* PdMode)

ATV_ERR ADIAPI_TxShutdownN (*UCHAR* DeviceIndex**, ...**)

Parameters

PdMode

Select the power-down mode. Three power-down modes are available:

TX_PD_MODE1

Entire chip is powered down except HPD and Rx Sense interrupts and CEC engine.

TX_PD_MODE2

Everything is powered down except CEC engine

TX_PD_MODE3

Everything is powered down

Return Value

ATVERR_OK

Remarks

See also ADIAPI_TxInit.

4.5 ADIAPI_TXGETCHIPREVISION

Description

Gets HDMI TX chip revision.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetChipRevision (***UCHAR*** *TxRev)

ATV_ERR ADIAPI_TxGetChipRevisionN (***UCHAR*** DeviceIndex**, ...**)

Parameters

TxRev

Pointer to receive HDMI TX chip revision.

Return Value

ATVERR_OK

Remarks

None.

4.6 ADIAPI_TXSETENABLEDEVENTS

Description

Enables or disables user notification on certain events as described in Section 5 on Page 57.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetEnabledEvents (***TX_EVENT*** Events, ***BOOL*** Enable)

ATV_ERR ADIAPI_TxSetEnabledEventsN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Events

The events that needs to be enabled or disabled ORed together. For a list of valid events, refer to Section 5 on Page 57. Only the events supplied in this parameter will be affected. All other events' state (Enabled/Disabled) will remain unchanged.

The TX_EVENT enum also offers 3 additional values that are used as an event groups: The values are:

TX_EVENT_ALL_EVENTS

This value defines all supported events

TX_EVENT_HDMI_EVENTS

This value groups all HDMI events. HDMI events are enabled by default and constitute the following events:

TX_EVENT_HPD_CHG

TX_EVENT_MSEN_CHG

TX_EVENT_EDID_READY

TX_EVENT_BKSV_READY

TX_EVENT_HDCP_AUTHENTICATED

TX_EVENT_HDCP_ERROR

TX_EVENT_CEC_EVENTS

This value defines all CEC events. CEC events are:

TX_EVENT_CEC_RX_MSG

TX_EVENT_CEC_TX_DONE

TX_EVENT_CEC_TX_TIMEOUT

TX_EVENT_CEC_TX_ARB_LOST

TX_EVENT_CEC_TX_LOG_ADDR_ALLOC

Enable

TRUE to enable notification on the supplied events

FALSE to disable notification on the supplied events

Return Value

ATVERR_OK

Remarks

None.

4.7 ADIAPI_TXISR

Description

Process the TX device interrupts. This function should be called by the application as soon as a TX device interrupt is detected. If the application uses polling and the interrupt line from the HDMI TX to the MCU is not connected, this function can be called periodically to poll and process any outstanding interrupts.

It should be noted that some of the TX interrupts can take relatively long time to process. For example, an EDID interrupt will consume at least the amount of time needed to read all 256 bytes of EDID via I2C. It is thus advisable for interrupt-driven real-time applications to disable the transmitter interrupt to the MCU before calling this function and re-enable it after this function returns.

The application will be notified on any change of the operating conditions if the notification events are enabled using the ADIAPI_TxSetEnabledEvents API.

The various interrupts, the action taken in the ISR and the associated notification event are listed in the table below. The details of the notification events can be found in Section 5 on Page 57.

Table 1. ISR and its Notification Event

Interrupt	Default Action	Notification Event
HPD Low	None	TX_EVENT_HPD_CHG
HPD High	Hardware reset is performed (ADIAPI_TxInit called)	TX_EVENT_HPD_CHG
Rx Sense Low/High	None	TX_EVENT_MSEN_CHG
EDID Ready	Read the next EDID segment up to TX_SUPPORTED_EDID_SEGMENTS to internal buffer	TX_EVENT_EDID_READY when a new segment is fully read
BKSV ready	Read and concatenate downstream BKSV into internal buffer	TX_EVENT_BKSV_READY when ALL downstream BKSVs are received
HDCP Authenticated	None	TX_EVENT_HDCP_AUTHENTICATED
HDCP Error	None	TX_EVENT_HDCP_ERROR
Vsync Edge	None	TX_EVENT_VSYNC_EDGE
Audio FIFO Full	None	TX_EVENT_AUDIO_FIFO_FULL
Embedded Sync Polarity Error	None	TX_EVENT_EMB_SYNC_ERROR
CEC TX Ready	None	TX_EVENT_CEC_TX_READY
CEC RX Ready	None	TX_EVENT_CEC_RX_READY
CEC TX Retry timeout	None	TX_EVENT_CEC_ERR_TIMEOUT
CEC TX Arbitration Lost	None	TX_EVENT_CEC_ERR_ARB_LOST

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxIsr (*void*)

ATV_ERR ADIAPI_TxIsrN (*UCHAR* DeviceIndex)

Parameters

Return Value

ATVERR_OK

The ISR completed execution and no other interrupts are pending. All pending interrupts are cleared.

ATVERR_FAILED

No pending interrupts detected.

Remarks

None.

4.8 ADIAPI_TXINTPENDING

Description

Check if TX interrupt is pending.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxIntPending (void)

Parameters

Return Value

ATVERR_TRUE	Pending interrupt detected
ATVERR_FALSE	No pending interrupt detected

4.9 ADIAPI_TXSETCONFIG

Description

Configure how TX ISR responds to various events and set other generals operational parameters.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetConfig (*TX_CONFIG* TxConfg, *BOOL* Set)

ATV_ERR ADIAPI_TxSetConfigN (*UCHAR* DeviceIndex**, ...**)

Parameters

TxConfig

Any of the TX_CONFIG values ORed together. Possible values are:

TX_INIT_ON_HPD_HIGH

This flag causes the TX ISR to perform h/w initialization when the HPD signal changes from low to high. The initialization is done by calling the ADIAPI_TxInit API.

TX_INIT_ON_HPD_LOW

This flag causes the TX ISR to perform h/w initialization when the HPD signal changes from high to low. The initialization is done by calling the ADIAPI_TxInit API.

TX_INIT_ON_EDID_ERROR

This flag causes the TX ISR to perform h/w initialization if an EDID segment is received that was not expected or requested. The initialization is done by calling the ADIAPI_TxInit API.

TX_HDCP_DISABLE_ON_ERROR

This flag causes the TX ISR to disable HDCP engine on any HDCP errors. Note that while this flag instructs the HDCP h/w to disable HDCP, the HDCP engine will not be actually disabled until it reaches authenticated state.

TX_ENABLE_TMDS_ON_INIT

This flag causes the any call to the ADIAPI_TxInit API to power-up TMDS clock and data lines. For more information, refer to the ADIAPI_TxInit API.

TX_ENABLE_DBG

This flag enables the debug messages from the TX module to be sent to the platform's console output channel. This flag only directs the debug messages to the output channel. To be able to see the messages on the console, the console must be enabled using the compilation switch "UART_DEBUG" as described in the ATV software architecture document.

Set

Set to TRUE to set the flags supplied in TxConfig

Set to FALSE to reset the flags supplied in TxConfig

Return Value

ATVERR_OK

Remarks

None.

4.10 ADIAPI_TXOVERRIDEHPDPD

Description

Maintain the current power state regardless of the state of the sink HPD signal. By default, HDMI TX chip will automatically power-down if the sink's HPD signal changes state from HIGH to LOW. This API can be used to change this default behavior so that the chip will remain powered-up regardless of the state of the sink HPD.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxOverrideHpdPD (***BOOL*** Override)

ATV_ERR ADIAPI_TxOverrideHpdPDN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Override

TRUE to disable automatic power-down when sink HPD goes low

FALSE to enable automatic power-down when sink HPD goes low.

Return Value

ATVERR_OK

Remarks

None.

4.11 ADIAPI_TXENABLETMDS

Description

Enable or disable TMDS output clock and data lines.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxEnableTMDS (***BOOL*** Enable, ***BOOL*** SoftOn)

ATV_ERR ADIAPI_TxEnableTMDSN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Enable

TRUE to enable TMDS clock and data lines

FALSE to disable TMDS clock and data lines

SoftOn

TRUE to enable soft TMDS clock turn on. This avoids glitches in the TMDS clock when it is turned on.

FALSE to disable the soft turn-on feature

Return Value

ATVERR_OK

Remarks

None.

4.12 ADIAPI_TXSETINPUTPIXELFORMAT

Description

Sets HDMI TX input pixel data format. The HDMI TX can accept video data from 8 to 36 pins, with various configurations to accommodate 4:4:4 or 4:2:2 format, embedded or separate sync, single or double data rate, repeated pixels and different pin assignments to interface with video data sources.

For most applications, the input pixel format needs to be set only once, unless the video source can change its output pixel format on the fly, in which case the HDMI TX input format must also be changed to match.

For detailed information regarding HDMI TX input video pin assignments, refer to the ADV7510 programming guide.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetInputPixelFormat (***UCHAR*** BitsPerColor, ***TX_IN_FORMAT*** Format,

UCHAR Style, ***TX_CHAN_ALIGN*** Alignment, ***BOOL*** RisingEdge)

ATV_ERR ADIAPI_TxSetInputPixelFormatN (***UCHAR*** DeviceIndex**, ...**)

Parameters

BitsPerColor

Specify the number of bits per color component. This can be 8, 10 or 12

Format

Video input format. This value indicates if the input video is SDR or DDR, with embedded or separate sync and if the input pixel clock is twice the pixel rate. This can be one of the following:

SDR_444_SEP_SYNC

SDR_422_SEP_SYNC

SDR_422_EMP_SYNC

SDR_422_SEP_SYNC_2X_CLK

SDR_422_EMB_SYNC_2X_CLK

DDR_444_SEP_SYNC

DDR_422_SEP_SYNC

DDR_422_EMB_SYNC

For more information about input pin assignment for each case, refer to the ADV7510 programming guide.

Style

Three input styles are available: 1, 2 or 3. For more information about input pin assignment for each style, refer to the ADV7510 programming guide.

Alignment

This value specifies the bit alignment of each channel in 4:2:2 modes. Three alignment types are available:

ALIGN_LEFT

ALIGN_RIGHT

ALIGN_EVEN

For more information about input pin assignment for each type, refer to the ADV7510 programming guide.

RisingEdge

This value specifies the clocking edge for DDR modes.

Set to TRUE to select rising edge. Set to FALSE to select falling edge.

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	One or more of the input parameters are invalid or the selected combination of the input parameters is invalid

Remarks

See also ADIAPI_TxSetOutputPixelFormat.

Refer to the ADV7510/1 programming guide for detailed description on pin assignment.

This API is not supported by transceiver type devices such as the ADV7623/2 and ADV7850.

4.13 ADIAPI_TXSETINPUTVIDEOCLOCK

Description

Divide HDMI TX input video clock to generate the correct pixel clock. The input video clock to HDMI TX must be equal to the pixel clock. In cases where the input clock is NOT equal to the pixel clock, this function should be used to divide the input clock to generate the correct pixel clock. This function is designed to be used for compatibility with some older source devices.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetInputVideoClock (***UCHAR*** ClkDivide)

ATV_ERR ADIAPI_TxSetInputVideoClockN (***UCHAR*** DeviceIndex**, ...**)

Parameters

ClkDivide

Set to 1, 2 or 4 to divide HDMI TX input clock by 1, 2 or 4 to generate the pixel clock

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API is not supported by transceiver type devices such as the ADV7623/2 and ADV7850.

4.14 ADIAPI_TXSETOUTPUTPIXELFORMAT

Description

Sets HDMI TX output pixel format. This API defines the up-conversion from 4:2:2 to 4:4:4 encoding or the down-conversion from 4:4:4 to 4:2:2 encoding along with the method to be used for up-conversion.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetOutputPixelFormat (***TX_OUT_ENCODING*** OutFormat, ***BOOL*** Interpolate)

ATV_ERR ADIAPI_TxSetOutputPixelFormatN (***UCHAR*** DeviceIndex**, ...**)

Parameters

OutFormat

This defines the required output pixel encoding format. This can be one of the following:

OUT_ENC_RGB_444

OUT_ENC_YUV_444

OUT_ENC_YUV_422

This value should match the Y1Y0 value sent in the AV info-frame.

Interpolate

This parameter is used only when up-converting the input from 4:2:2 to 4:4:4 encoding.

Set to TRUE to up-convert using linear interpolation.

Set to FALSE to up-convert using line duplication.

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

See also ADIAPI_TxSetInputPixelFormat.

4.15 ADIAPI_TXSETMANUALPIXELREPEAT

Description

Manually sets HDMI TX output pixel repetition rate and parameters.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetManualPixelRepeat (***UCHAR*** Vic, ***UCHAR*** Factor, ***UCHAR*** PrValue)

ATV_ERR ADIAPI_TxSetManualPixelRepeatN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Vic

This value defines the Video Identification Code that should be sent in the AV info-frame. See ADIAPI_TxSendAVInfoframe for more details

Factor

This value defines the required multiplication factor of the input pixel clock. Possible values are 1, 2, 3 and 4.

PrValue

This value defined the PR (Pixel Repeat) value that should be sent in the AV info-frame. See ADIAPI_TxSendAVInfoframe for more details

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

See also ADIAPI_TxSetAutoPixelRepeat.

4.16 ADIAPI_TXSETAUTOPIXELREPEAT

Description

Sets HDMI TX to automatically calculate and output the correct pixel repetition rate based on detected video format and audio sampling frequency. The audio sampling frequency is either extracted from the stream or defined by the user. See ADIAPI_TxSetAudChStatSampFreq for more details.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAutoPixelRepeat (*TX_AUTO_PR_MODE* Mode, *UCHAR* InVic,
TX_REFR_RATE RefreshRate, *UCHAR* AspectRatio)

ATV_ERR ADIAPI_TxSetAutoPixelRepeatN (*UCHAR* DeviceIndex**, ...**)

Parameters

Mode

PR_NORMAL

Normal automatic pixel repetition. In this mode, HDMI TX will automatically calculate the required pixel repetition based on audio sampling rate and detected VIC. The resulting video identification code (VIC) and Pixel Repeat value (PR) will be automatically inserted in the AV info-frame. See ADIAPI_TxSendAVInfoframe for more details.

PR_MAX

Maximum automatic pixel repetition. This mode is similar to normal automatic mode, except the required pixel repetition will always be set to the highest possible value the HDMI TX is capable of. This makes video timing independent of audio timing.

InVic

This value defines the video identification code (VIC) of the input video format. If the VIC of the input video is not known, this value must be set to 0xff and the 'RefreshRate' and 'Aspect' parameters must be used.

RefreshRate

This value defines the refresh rate for video modes with low refresh rate or with 2x or 4x the normal refresh rate, when the 'Vic' parameter is undefined (set to 0xff) Possible values are:

REFRESH_NORMAL

REFRESH_LOW

REFRESH_2X

REFRESH_4x

AspectRatio

This value defines the aspect ratio of the input video, when the 'Vic' parameter is undefined (set to 0xff). Possible values are:

4*3 (=12 for 4x3 aspect)

16*9 (=144 for 16x9 aspect)

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

See also ADIAPI_TxSetManualPixelRepeat.

4.17 ADIAPI_TXSETOUTPUTCOLORDEPTH

Description

Sets output color depth and the method used to handle deep color down-conversion. When the input color depth to the HDMI TX is less than the color depth of the output, the remaining least-significant bits of the output will be filled with 0s. When the input color depth is larger than the output, truncation or active dithering can be used to reduce the color depth.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetOutputColorDepth (*UCHAR* Depth, *TX_DC_METHOD* DcMethod)

ATV_ERR ADIAPI_TxSetOutputColorDepthN (*UCHAR* DeviceIndex**, ...**)

Parameters

Depth

Required color depth of the output. This value can be 24, 30 or 36 and will correctly set the general control packet color depth field. Any other value will be written un-modified to the General Control Packet CD (Color Depth) field.

DcMethod

This value specifies the down-conversion method that will be used if the input color depth is larger than the output color depth. Possible values are:

TX_DC_TRUNCATE

TX_DC_ACTIVE_DITHER

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

None.

4.18 ADIAPI_TXSETCSC

Description

Sets color space conversion for HDMI TX chip.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetCSC (*TX_CS_MODE* InColorSpace, *TX_CS_MODE* OutColorSpace)

ATV_ERR ADIAPI_TxSetCSCN (*UCHAR* DeviceIndex**, ...**)

Parameters

InColorSpace

Color space input to TX device. This can be one of the values in Table 2.

Table 2. Tx Input Color Space Modes

TX_CS_MODE	Meaning	Range
TX_CS_RGB	RGB	0-255
TX_CS_YUV_601	YCrCb 601 (SDTV)	16-235
TX_CS_YUV_709	YCrCb 709 (HDTV)	16-235
TX_CS_YCC_601	xvYCC 601 (Extended gamut SDTV)	0-255
TX_CS_YCC_709	xvYCC 709 (Extended gamut HDTV)	0-255
TX_CS_AUTO	Disable CSC (out CS = In CS)	

The TX_CS_AUTO setting will disable Color Space Conversion.

OutColorSpace

Color space output from TX device. This can be one of the values in Table 3.

Table 3. Tx Output Color Space Modes

TX_CS_MODE	Meaning	Range
TX_CS_RGB	RGB	0-255
TX_CS_YUV_601	YCrCb 601 (SDTV)	16-235
TX_CS_YUV_709	YCrCb 709 (HDTV)	16-235
TX_CS_YCC_601	xvYCC 601 (Extended gamut SDTV)	0-255
TX_CS_YCC_709	xvYCC 709 (Extended gamut HDTV)	0-255
TX_CS_AUTO	Disable CSC (out CS = In CS)	

The TX_CS_AUTO setting will disable Color Space Conversion.

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

None.

4.19 ADIAPI_TXSETAUDIOINTERFACE

Description

Sets the input audio interface and output audio packet type.

The TX device has three physical audio interfaces:

- I2S Inputs (8 Channels)
- SPDIF
- DSD

The I2S interface can accept data in the following formats:

- Standard I2S
- Right justified I2S
- Left justified I2S
- AES3 (IEC 60958-3)

Two different output packet formats can be selected when the input is I2S: Audio sample packet or HBR packet.

When the input interface is I2S, for all formats except AES3, the channel status data to be sent to the receiver (in the ASP/HBR packet) must be explicitly set in the TX registers, since I2S contains pure audio samples. For AES3 format, the TX device can extract the channel status from the data stream or can use user-defined values from registers.

The mapping between I2S input and the data sent in the audio sample packet is configurable (For example, I2S3 input left channel can be sent in the audio sample packet sub-frame 0 instead of the default sub-frame 6) See ADIAPI_TxSetAudChanMapping for details.

The SPDIF interface can accept 2 channel L-PCM audio or AES3 (IEC 60958-3) audio at sampling rates of up to 192 KHz. The sampling frequency extracted from the stream will be sent in the Audio Sample Packet channel status bits. The sampling frequency used for pixel repeat can be either the one extracted from the stream or a user-defined value.

As with I2S, the TX device can output either Audio Sample Packet or High Bit Rate packet when the input is SPDIF.

The DSD interface can be used to input DSD or DST audio. The output audio packets will be either one-bit audio for DSD or DST audio packet for DST.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudioInterface (***TX_AUD_FORMAT*** InputFormat,

TX_AUD_PKT_TYPE OutType, ***UCHAR*** HbrStrmCount)

ATV_ERR ADIAPI_TxSetAudioInterfaceN (***UCHAR*** DeviceIndex**, ...**)

Parameters

InputFormat

This value defines the audio interface and format to be used for inputting audio to the HDMI TX. This can be one of the values defined in Table 4. Note that some input formats cannot be used with certain output packet types.

Table 4. Tx Input Audio Format

TX_AUD_FORMAT	Input Interface	Input Format	Valid Output Packet Type (TX_AUD_PKT_TYPE)
TX_I2S_STD	I2S	Standard I2S	AUD_SAMP_PKT HBR_STRM_PKT
TX_I2S_RJUST	I2S	Right justified I2S	AUD_SAMP_PKT HBR_STRM_PKT
TX_I2S_LJUST	I2S	Left justified I2S	AUD_SAMP_PKT HBR_STRM_PKT
TX_I2S_AES3	I2S	AES3 direct	AUD_SAMP_PKT HBR_STRM_PKT
TX_I2S_SPDIF	I2S	IEC61937 Bi-phase Mark	HBR_STRM_PKT
TX_SPDIF	SPDIF	IEC61937 Bi-phase Mark	AUD_SAMP_PKT
TX_DSD_NORM	DSD	DSD normal	ONE_BIT_ASP
TX_DSD_SDIF3	DSD	SDIF-3	ONE_BIT_ASP
TX_DSD_DST	DSD	DST normal	DST_AUD_PKT
TX_DSD_DST_SDR	DSD	DST 2X	DST_AUD_PKT
TX_DSD_DST_DDR	DSD	DSD 1X (DDR)	DST_AUD_PKT

OutType

This value defines the audio type (packet type) that will be output by the HDMI TX as illustrated in Table 5. Note that some packet types can be used only with certain input formats.

Table 5. Tx Audio Output Packet Type

Output Packet Type (TX_AUD_PKT_TYPE)	Definition	Valid Input Formats
AUD_SAMP_PKT	Audio Sample Packet	I2S_STD, I2S_RJUST, I2S_LJUST, I2S_AES3
HBR_STRM_PKT	High Bit Rate Audio Stream Packet	I2S_STD, I2S_RJUST, I2S_LJUST, I2S_AES3, I2S_SPDIF
ONE_BIT_ASP	One Bit Audio Sample Packet	DSD_NORM, DSD_SDIF3
DST_AUD_PKT	DST Audio Packet	DSD_DST, DSD_DST_SDR, DSD_DST_DDR

HbrStrmCount

This parameter is used only when the output audio packet type is HBR_STRM_PKT. It specifies the number of HBR streams encoding. This value can only be 1 or 4.

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid input parameter value

Remarks

For High Bit Rate audio (if the OutType parameter is set to HBR_STRM_PKT), this API will set some operational parameters according to Table 6.

Table 6. Operational Parameters for HBR Audio

Parameter	Setting for HBR Audio	APIs Used to Change Parameter
Audio Info-frame CA field	0x1F	ADIAPI_TxSendAudioInfoframe
Audio Info-frame CC field	0x07	ADIAPI_TxSendAudioInfoframe
Channel Status Sampling Frequency Source	User defined value	ADIAPI_TxSetAudChStatSampFreq ADIAPI_TxSetAudChanStatus
Channel Status Sampling Frequency	0x09	ADIAPI_TxSetAudChStatSampFreq ADIAPI_TxSetAudChanStatus
I2S inputs	All inputs will be enabled if audio is not muted All inputs will be enabled after audio is un-muted	ADIAPI_TxAudInputEnable
MCLOCK ratio	128xFS	ADIAPI_TxSetAudMCLK

Calling any of the APIs defined in the third column above will change the setting for the corresponding parameter according to user input.

4.20 ADIAPI_TXSETAUDCHANMAPPING

Description

Sets the mapping between I2S input channels and the output audio samples. The default setting is one-to-one according to Table 7. The default setting can be changed using this API.

Table 7. Input Channels and Output Audio Samples Mapping

Input Channel	Output Sample
I2S0 Left Channel	Audio Sample 0 Left Channel
I2S0 Right Channel	Audio Sample 0 Right Channel
I2S1 Left Channel	Audio Sample 1 Left Channel
I2S1 Right Channel	Audio Sample 1 Right Channel
I2S2 Left Channel	Audio Sample 2 Left Channel
I2S2 Right Channel	Audio Sample 2 Right Channel
I2S3 Left Channel	Audio Sample 3 Left Channel
I2S3 Right Channel	Audio Sample 3 Right Channel

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudChanMapping (*TX_AUD_CHAN* InChan, *TX_AUD_CHAN* OutSample)

ATV_ERR ADIAPI_TxSetAudChanMappingN (*UCHAR* DeviceIndex**, ...**)

Parameters

InChan

Input channel ID. This can be one of the following:

CH0_LEFT
CH0_RIGHT
CH1_LEFT
CH1_RIGHT
CH2_LEFT
CH2_RIGHT
CH3_LEFT
CH3_RIGHT

OutSample

Output sample position. This can be one of the following:

CH0_LEFT
CH0_RIGHT
CH1_LEFT
CH1_RIGHT
CH2_LEFT
CH2_RIGHT
CH3_LEFT
CH3_RIGHT

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid input parameter value

Remarks

None.

4.21 ADIAPI_TXSETAUDNVALUE

Description

Sets 'N' value that will be used by the HDMI TX to calculate the audio sampling frequency. CTS (Cycle Time Stamp) can be set using the ADIAPI_TxSetAudCTS API.

HDMI TX uses both N and CTS to calculate the audio sampling frequency according to the formula:

$128\ f_S = f_{TMDS\_CLK}\ N / CTS$

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudNValue (*UINT32* NValue)

ATV_ERR ADIAPI_TxSetAudNValueN (*UCHAR* DeviceIndex**, ...**)

Parameters

NValue

Specify the 20-bit 'N' value that HDMI TX will use to calculate the audio sampling frequency.

If this value is set to 0, the N value will be calculated using the sampling frequency obtained from the current audio info-frame/audio channel status. If new audio info-frame or channel status is received, this function must be called to update the N value.

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid N value

Remarks

None.

4.22 ADIAPI_TXSETAUDCTS

Description

Sets CTS (Cycle Time Stamp) value that will be used by the HDMI TX to calculate the audio sampling frequency. The 'N' value can be set using the ADIAPI_TxSetAudNValue API.

HDMI TX uses both N and CTS to calculate the audio sampling frequency according to the formula

$128\ f_S = f_{TMDS\_CLK}\ N / CTS$

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudCTS (*UINT32* CTS)

ATV_ERR ADIAPI_TxSetAudCTSN (*UCHAR* DeviceIndex**, ...**)

Parameters

CTS

Specifies the 20-bit 'CTS' value that HDMI TX will use to calculate the audio sampling frequency.

If this value is set to 0, the CTS value will be automatically calculated by the chip using the SCLK.

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid CTS value

Remarks

None.

4.23 ADIAPI_TXSETAUDMCLK

Description

Sets HDMI TX input audio master clock (MCLK) frequency. The MCLK can be externally supplied or internally generated using SCLK.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudMCLK (*TX_MCLK_FREQ* MClk)

ATV_ERR ADIAPI_TxSetAudMCLKN (*UCHAR* DeviceIndex**, ...**)

Parameters

MClk

Defines the HDMI TX audio master clock frequency. This can be one of the following values:

```
TX_MCLK_128FS     /* Set MCLK = 128 * Sampling frequency */
TX_MCLK_256FS     /* Set MCLK = 256 * Sampling frequency */
TX_MCLK_384FS     /* Set MCLK = 384 * Sampling frequency */
TX_MCLK_512FS     /* Set MCLK = 512 * Sampling frequency */
TX_MCLK_HBR       /* Set MCLK for High Bit Rate audio    */
TX_MCLK_AUTO      /* Generate MCLK internally using SCLK */
```

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid MCLK value

Remarks

For High Bit Rate audio, MCLK must be set to TX_MCLK_HBR. The ADIAPI_TxSetAudioInterface API automatically adjusts MCLK for HBR audio. When the user changes the audio type to anything other than High Bit Rate (using ADIAPI_TxSetAudioInterface), the MCLK setting will be automatically changed to reflect the value defined in the last call to ADIAPI_TxSetAudMCLK.

See also ADIAPI_TxSetAudNValue and ADIAPI_TxSetAudCTS.

4.24 ADIAPI_TXSETAUDCLKPOLARITY

Description

Sets the input clock polarity for MCLK, SCLK and DSD clock.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudClkPolarity (*BOOL* RisingEdge)

ATV_ERR ADIAPI_TxSetAudClkPolarityN (*UCHAR* DeviceIndex**, ...**)

Parameters

RisingEdge

Clock polarity for MCLK, (if externally supplied) SCLK and DSD clock

Set to TRUE to latch input data on rising edge

Set to FALSE to latch input data on falling edge

Return Value

ATVERR_OK

Remarks

None.

4.25 ADIAPI_TXSETAUDCHSTATSAMPFREQ

Description

Sets the sampling frequency to be sent in the Audio Sample Packet's channel status bits. The source of the sampling frequency (Extracted from input stream or defined by user) can be set independently from the rest of the channel status fields defined in the ADIAPI_TxSetAudChanStatus API.

The TX device can use the sampling frequency extracted from the input stream or the sampling frequency defined by user (in the channel status bits) for automatic pixel repeat calculation. See ADIAPI_TxSetAutoPixelRepeat for more details.

The source of channel status bits and/or sampling frequency sent in the sample packet can be user-defined, extracted from input stream, or both, depending on the input audio format as described in Table 8.

Table 8. Channel Status and Sampling Frequency for Audio Formats

Input Audio	Channel Status Source	Sampling Frequency Source	Sampling frequency source for Pixel Repeat
I2S	User defined	User defined	User defined
I2S–AES3	User defined / From stream	User defined / From stream	User defined / From Stream
SPDIF	From stream	From Stream	User defined / From Stream

This API only defines the source and value of the channel status sampling frequency. Incorrect settings of the sampling frequency source will be ignored by the TX device (e.g., if input audio is I2S and the user selects the sampling frequency source to be from stream, the TX device will ignore the setting and use the latest programmed sampling frequency. The default sampling frequency is 44.1 KHz)

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAudChStatSampFreq (*TX_AUD_FS* SampFreq)

ATV_ERR ADIAPI_TxSetAudChStatSampFreqN (*UCHAR* DeviceIndex**, ...**)

Parameters

SampFreq

Specify the channel status sampling frequency that will be used for pixel repeat calculation and will be sent in the audio sample packet. This can be one of the following:

```
TX_FS_32KHZ        /* 32 KHz    */
TX_FS_44KHZ        /* 44.1 KHz */
TX_FS_48KHZ  /* 48 KHz    */
TX_FS_88KHZ  /* 88.2 KHz */
TX_FS_96KHZ  /* 96 KHz    */
TX_FS_176KHZ       /* 176.4 KHz*/
TX_FS_192KHZ       /* 192 KHz   */
TX_FS_HBR          /* Setting for HBR audio (768 KHz) */
TX_FS_FROM_STRM    /* Use sampling freq extracted from audio stream */
```

Note that for HBR (High Bit Rate) audio, the sampling frequency must be set to TX_FS_HBR. This is done implicitly by the ADIAPI_TxSetAudioInterface API. Setting the sampling frequency to TX_FS_FROM_STRM will only change the source of the sampling frequency; the sampling frequency value programmed into the chip will not change.

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid SampFreq value

Remarks

For HBR audio, the sampling frequency must be set to TX_FS_HBR. The ADIAPI_TxSetAudioInterface API automatically adjusts the channel status sampling frequency for HBR audio.

See also ADIAPI_TxSetAutoPixelRepeat.

See also ADIAPI_TxSetAudChanStatus.

4.26 ADIAPI_TXSETAUDCHANSTATUS

Description

Sets HDMI TX audio channel status parameters. This function should be called when new audio channel status is received. This function will automatically set the size of the right-justified I2S word size, either from audio info-frame (if the audio info-frame sample size is not 0) or from the CsWordLen field of the supplied channel status.

Depending on the input audio format, this API also defines the source of the channel status bits that will be sent in the audio sample packet (either user-defined or extracted from the input stream) as defined in Table 9. The sampling frequency field of the channel status can also be independently set by the ADIAPI_TxSetAudChStatSampFreq API.

Table 9. Source of Channel Status for Input Audio Format

Input Audio	Channel Status Source	Sampling Frequency
I2S	User defined	Refer to ADIAPI_TxSetAudChStatSampFreq
I2S – AES3	User defined / From stream	Refer to ADIAPI_TxSetAudChStatSampFreq
SPDIF	From stream	Refer to ADIAPI_TxSetAudChStatSampFreq

Synopsis

```
#include "adi_apis.h"
typedef struct
{
    UCHAR   CsConsumer;          /* Channel status data. Refer to Appendix A */
    UCHAR   CsSampType;    /* Channel status data. Refer to Appendix A */
    UCHAR   CsCopyright;   /* Channel status data. Refer to Appendix A */
    UCHAR   CsEmphasis;    /* Channel status data. Refer to Appendix A */
    UCHAR   CsCatCode;           /* Channel status data. Refer to Appendix A */
    UCHAR   CsSrcNum;            /* Channel status data. Refer to Appendix A */
    UCHAR   CsChanNum;           /* Channel status data. Refer to Appendix A */
    UCHAR   CsSampFreq;    /* Set to 0xFF to ignore */
    UCHAR   CsClkAccur;    /* Channel status data. Refer to Appendix A */
    UCHAR   CsWordLen;           /* Channel status data. Refer to Appendix A */
}TX_CHAN_STATUS;
```

ATV_ERR ADIAPI_Tx SetAudChanStatus (*BOOL* FromStream, *TX_CHAN_STATUS* *ChanStat);

ATV_ERR ADIAPI_Tx SetAudChanStatusN (*UCHAR* DeviceIndex**, ...**)

Parameters

FromStream

This parameter indicates if the channel status data (sent in the audio sample packet) is to be extracted from the incoming audio stream or taken from the user defined values (pointed to by the ChanStat parameter) This option is only valid for audio modes where channel status data is present in the incoming audio stream. For all other modes, the channel status data will be taken from the user-defined values. The sampling frequency field of the channel status can also be set independently using the ADIAPI_TxSetAudChStatSampFreq API.

Set to TRUE to use channel status from audio stream.

Set to FALSE to use channel status supplied by the ChanStat parameter

ChanStat

Pointer to CHAN_STATUS structure containing the required audio channel status to be used. This parameter will be ignored if the FormStream parameter is set to TRUE.

If FromStream is set to FALSE, the sampling frequency defined in the CsSampFreq field will overwrite any sampling frequency set by the ADIAPI_TxSetAudChStatSampFreq API and will also set the source of the sampling frequency to

match the channel status source settings. If this is not wanted, the sampling frequency setting can be ignored by setting the CsSampFreq field to 0xFF.

Return Value

ATVERR_OK

Remarks

If FromStream parameter is set to FALSE, the sampling frequency defined in the CsSampFreq field will overwrite any sampling frequency set by the ADIAPI_TxSetAudChStatSampFreq API and will also set the source of the sampling frequency to match the channel status source settings. If this is not wanted, the sampling frequency setting can be ignored by setting the CsSampFreq field to 0xFF.

For HBR (High Bit Rate) audio, the sampling frequency must be set to 768KHz. This is automatically done by the ADIAPI_TxSetAudioInterface API. However, any call to

ADIAPI_TxSetAudChanStatus will overwrite the CA and CC fields as provided by the caller.

See also ADIAPI_TxSetAudChStatSampFreq.

4.27 ADIAPI_TXAUDINPUTENABLE

Description

Enable/disable audio input signal to HDMI TX.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxAudInputEnable (***TX_AUD_INTERFACE*** Interface, ***BOOL*** Enable)

ATV_ERR ADIAPI_TxAudInputEnableN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Interface

Specify the audio interface to enable/disable. This can be one of the following values:

```
TX_AUD_IN_I2S0          /* I2S channel 0 */
TX_AUD_IN_I2S1          /* I2S channel 1 */
TX_AUD_IN_I2S2          /* I2S channel 2 */
TX_AUD_IN_I2S3          /* I2S channel 3 */
TX_AUD_IN_I2S           /* All I2S channels (0-3) */
TX_AUD_IN_SPDIF        /* SPDIF          */
TX_AUD_IN_DSD0          /* DSD channel 0 */
TX_AUD_IN_DSD1          /* DSD channel 1 */
TX_AUD_IN_DSD2          /* DSD channel 2 */
TX_AUD_IN_DSD3          /* DSD channel 3 */
TX_AUD_IN_DSD4          /* DSD channel 4 */
TX_AUD_IN_DSD5          /* DSD channel 5 */
TX_AUD_IN_DSD6          /* DSD channel 6 */
TX_AUD_IN_DSD7          /* DSD channel 7 */
TX_AUD_IN_DSD           /* All DSD channels (0-7) */
TX_AUD_IN_ALL           /* All audio inputs */
```

Enable

Set to TRUE to enable the audio interface specified in the "Interface" parameter

Set to FALSE to disable the audio interface specified in the "Interface" parameter

Return Value

ATVERR_OK	Function completed successfully
ATVERR_INV_PARM	Invalid Interface value

Remarks

None.

4.28 ADIAPI_TXSETI2SINPUT

Description

Enable I2S audio input 0-3 based on audio info-frame channel allocation and if the input stream is HBR.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetI2sInput (UCHAR ChanCount, UCHAR ChanAlloc,
TX_AUD_PKT_TYPE AudType)

ATV_ERR ADIAPI_TxSetI2sInputN (*UCHAR* DeviceIndex**, ...**)

Parameters

ChanCount

Number of audio channels (0-7)

ChanAlloc

Channel allocation field from Audio InfoFrame

AudType

Input audio packet type as defined in Table 10.

Table 10. Input Audio Packet Type

Audio Packet Type Packet Type (TX_AUD_PKT_TYPE)	Definition
AUD_SAMP_PKT	Audio Sample Packet
HBR_STRM_PKT	High Bit Rate Audio Stream Packet
ONE_BIT_ASP	One Bit Audio Sample Packet
DST_AUD_PKT	DST Audio Packet

Return Value

ATVERR_OK Function completed successfully

Remarks

None.

4.29 ADIAPI_TXSETOUTPUTMODE

Description

Sets output video mode of HDMI TX to HDMI or DVI.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetOutputMode (*TX_OUTPUT_MODE* OutMode)

ATV_ERR ADIAPI_TxSetOutputModeN (*UCHAR* DeviceIndex**, ...**)

Parameters

OutMode

Required HDMI TX output mode.

Can be TX_OUT_MODE_HDMI or TX_OUT_MODE_DVI.

Return Value

ATVERR_OK Function completed successfully

ATVERR_INV_PARM Invalid OutMode value

Remarks

None.

4.30 ADIAPI_TXHDCPENABLE

Description

Enable or disable HDCP on HDMI TX output.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxHdcpEnable (*BOOL* EncEnable, *BOOL* FrameEncEnable)

ATV_ERR ADIAPI_TxHdcpEnableN (*UCHAR* DeviceIndex**, ...**)

Parameters

EncEnable

TRUE to enable HDCP

FALSE to disable HDCP

FrameEncEnable

TRUE to enable encryption of the current frame

FALSE to disable encryption of the current frame while maintaining HDCP synchronization

Return Value

ATVERR_OK

Remarks

None.

4.31 ADIAPI_TXGETBKSVLIST

Description

Read BKSVs list from HDMI TX once all BKSVs are read.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetBksvList (*UCHAR* *BksvList, *UCHAR* *BksvCount)

ATV_ERR ADIAPI_TxGetBksvListN (*UCHAR* DeviceIndex**, ...**)

Parameters

BksvList

Pointer to a buffer to receive the downstream BKSV list as read by the HDMI TX. This list is available only after HDMI TX successfully read all downstream BKSVs. This list will not be available if HDCP is disabled or if any HDCP errors are detected. The size of the buffer must be large enough to accommodate the number of BKSVs specified in the TX_SUPPORTED_DS_DEVICE_COUNT configuration parameter (i.e., Minimum buffer size will be TX_SUPPORTED_DS_DEVICE_COUNT * 5)

This parameter can be set to NULL to only return the number of available BKSVs in the BksvCount parameter.

BksvCount

This is a pointer to receive the number of BKSVs reported by the downstream device. This number will also include the downstream repeater BKSV if the downstream device is a repeater. This value normally specify the number of BKSVs returned in the BksvList buffer, unless the BKSV count reported by the downstream device exceeds TX_SUPPORTED_DS_DEVICE_COUNT, in which case the BksvList buffer will hold the first TX_SUPPORTED_DS_DEVICE_COUNT BKSVs.

Return Value

ATVERR_OK Function completed successfully

ATVERR_FAILED HDCP is disabled or authentication is not complete or HDCP errors encountered

Remarks

None.

4.32 ADIAPI_TXGETBSTATUS

Description

Read downstream device Bstatus and Bcaps registers. Bstatus and Bcaps are available only if the state of the HDCP engine is HDCP_BSTATUS_READY, HDCP_BKSV_LIST_READY or HDCP_AUTHENTICATED. The HDCP state can be obtained by calling the ADIAPI_TxGetHdcpState API.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetBstatus (***UINT16*** *Bstatus, ***UCHAR*** *Bcaps)

ATV_ERR ADIAPI_TxGetBstatusN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Bstatus

Pointer to receive the downstream device HDCP Bstatus register.

Bcaps

Pointer to receive the downstream device HDCP BCAPS register

Return Value

ATVERR_OK Function completed successfully

ATVERR_FAILED HDCP is disabled or downstream device is not available

Remarks

Bstatus and Bcaps are available only if the state of the HDCP engine is HDCP_BSTATUS_READY, HDCP_BKSV_LIST_READY or HDCP_AUTHENTICATED. The HDCP state can be obtained by calling the ADIAPI_TxGetHdcpState API.

4.33 ADIAPI_TXGETHDCPSTATE

Description

Read the current status of HDCP engine.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetHdcpState (***TX_HDCP_STATE*** *HdcpState)

ATV_ERR ADIAPI_TxGetHdcpStateN (***UCHAR*** DeviceIndex**, ...**)

Parameters

HdcpState

This is a pointer to receive the current status of HDCP engine. Possible states are:

TX_HDCP_NO_DS_DEVICE

TX_HDCP_DISABLED

TX_HDCP_BSTATUS_READY

TX_HDCP_BKSV_LIST_READY

TX_HDCP_AUTHENTICATED

Return Value

ATVERR_OK Function completed successfully

Remarks

None.

4.34 ADIAPI_TXGETLASTHDCPERROR

Description

Return the last error status of HDCP engine. This API returns the last encountered HDCP error(s) since the previous read using this API. All returned error bits will be cleared following a call to this API.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetLastHdcpError (*TX_HDCP_ERR* *Error)

ATV_ERR ADIAPI_TxGetLastHdcpErrorN (*UCHAR* DeviceIndex**, ...**)

Parameters

Status

This is a pointer to receive HDCP errors that occurred since the last call to this API. Returned errors are ORed together. Possible error bits are:

TX_HDCP_ERR_BAD_RECV_BKSV

TX_HDCP_ERR_RI_MISMATCH

TX_HDCP_ERR_PJ_MISMATCH

TX_HDCP_ERR_I2C_ERROR

TX_HDCP_ERR_REP_DONE_TIMEOUT

TX_HDCP_ERR_MAX_CASCADE_EXCEEDED

TX_HDCP_ERR_V_DASH_CHECK_FAILED

TX_HDCP_ERR_MAX_DEVICE_EXCEEDED

All error bits will be cleared upon calling this function. HDMI TX automatically restarts the authentication process on any HDCP error.

Return Value

ATVERR_OK

Remarks

None.

4.35 ADIAPI_TXGETEDIDSEGMENT

Description

Read a 256-byte EDID segment received by HDMI TX.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetEdidSegment (*UCHAR* SegNum, *UCHAR* *SegBuf)

ATV_ERR ADIAPI_TxGetEdidSegmentN (*UCHAR* DeviceIndex**, ...**)

Parameters

SegNum

EDID segment number to read (Starting from 0)

SegBuf

This is a pointer to a 256-byte buffer to receive the requested EDID segment. This buffer will contain valid data only if the return value is ATVERR_OK

Return Value

ATVERR_OK	EdidBuf will contain the requested EDID segment
ATVERR_FAILED	Requested EDID segment is not available

Remarks

None.

4.36 ADIAPI_TXGETHPDMSENSTATE

Description

Return the Hot Plug Detect and Monitor Sense state of HDMI TX.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetHpdMsenState (*BOOL* *Hpd, *BOOL* *Msen);

ATV_ERR ADIAPI_TxGetHpdMsenStateN (*UCHAR* DeviceIndex**, ...**)

Parameters

Hpd

This is a pointer to receive the sink device Hot Plug Detect state. This parameter can be set to NULL if the HPD state is not required. If not NULL, on return, it will be set to TRUE if HPD is high and FALSE if HPD is low.

Msen

This is a pointer to receive the sink device monitor sense state. This parameter can be set to NULL if the monitor sense state is not required. If not NULL, on return, it will be TRUE if monitor sense is high and FALSE if monitor sense is low.

Return Value

ATVERR_OK

Remarks

None.

4.37 ADIAPI_TXGETEDIDCONTROLLERSTATE

Description

Gets EDID/HDCP controller state.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetEdidControllerState (TX_EDID_CTRL_STATE *State)

ATV_ERR ADIAPI_TxGetEdidControllerStateN (*UCHAR* DeviceIndex**, ...**)

Parameters

State

This is a pointer to the current status of EDID/HDCP engine. Possible states are:

TX_HDCP_NO_DS_DEVICE

TX_HDCP_DISABLED

TX_HDCP_BSTATUS_READY

TX_HDCP_BKSV_LIST_READY

TX_HDCP_AUTHENTICATED

Return Value

ATVERR_OK Function completed successfully

Remarks

None.

4.38 ADIAPI_TXOUTPUTMODEHDMI

Description

Gets output mode: HDMI or DVI.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxOutputModeHdmi (BOOL *IsHdmi)

ATV_ERR ADIAPI_TxOutputModeHdmiN (*UCHAR* DeviceIndex**, ...**)

Parameters

IsHdmi

This is a pointer to the output mode.

TRUE if the output mode is HDMI

FALSE if the output mode is DVI

Return Value

ATVERR_TRUE	Output mode is HDMI
ATVERR_FALSE	Output mode is DVI

Remarks

None.

4.39 ADIAPI_TXOUTPUTENCRYPTED

Description

Check if the output is encrypted.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxOutputEncrypted (BOOL * Encrypted)

ATV_ERR ADIAPI_TxOutputEncryptedN (*UCHAR* DeviceIndex**, ...**)

Parameters

Encrypted

This is a pointer to encryption state.

TRUE if the output is encrypted

FALSE if the output is not encrypted

Return Value

ATVERR_TRUE	Output is encrypted
ATVERR_FALSE	Output is not encrypted

4.40 ADIAPI_TXPLLLOCKED

Description

Check if the PLL is locked.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxPllLocked (BOOL * Locked)

ATV_ERR ADIAPI_TxPllLockedN (*UCHAR* DeviceIndex**, ...**)

Parameters

Locked

This is a pointer to PLL lock state.

TRUE if the PLL is locked

FALSE if the PLL is not locked

Return Value

ATVERR_TRUE	PLL is locked
ATVERR_FALSE	PLL is not locked

4.41 ADIAPI_TXGETSTATUS

Description

This API provides the status of HDMI TX module. It can be used by the application to get some information regarding HDMI TX current state.

Synopsis

#include "adi_apis.h"

```
typedef struct
{
    BOOL    ChipPd;             /* TRUE/FALSE = Power down is On/Off    */
    BOOL    TmdsPd;             /* TRUE/FALSE = TMDS Power is Off/On    */
    BOOL    Hpd;                /* TRUE/FALSE = HPD is On/Off           */
    BOOL    MonSen;             /* TRUE/FALSE = Monitor sense is On/Off */
    BOOL    OutputHdmi;         /* TRUE/FALSE = Output is HDMI/DVI      */
    BOOL    PllLocked;          /* TRUE/FALSE = PLL locked/unlocked     */
    BOOL    VideoMuted;         /* TRUE/FALSE = Video is muted/un-muted */
    BOOL    ClearAVMute;        /* TRUE/FALSE = Clear AVMUTE On/Off     */
    BOOL    SetAVMute;          /* TRUE/FALSE = Set AVMUTE On/Off       */
    BOOL    AudioRep;           /* TRUE/FALSE = Audio repeat En/Dis     */
    BOOL    SpdifEnable;        /* TRUE/FALSE = SPDIF enabled/Disable   */
    UCHAR   I2SEnable;          /* Bits 0-3 = I2S 0-3 enable status     */
    UCHAR   DetectedVic;        /* Detected VIC                         */
    UINT16  HdcpErr;            /* Last HDCP errors                     */
}TX_STATUS;
```

ATV_ERR ADIAPI_TxGetStatus (*TX_STATUS* *TxStat)

ATV_ERR ADIAPI_TxGetStatusN (*UCHAR* DeviceIndex**, ...**)

Parameters

TxStat

This is a pointer to TX_STATUS structure to receive HDMI TX status information. The members of this structure are described below:

ChipPd

Set to TRUE if HDMI TX chip is powered down

Set to FALSE if HDMI TX chip is in normal operation

TmdsPd

Set to TRUE if any of the TMDS lines (Ch0, Ch1, Ch2 or Clk) are powered down

Set to FALSE if all of the TMDS lines (Ch0, Ch1, Ch2 or Clk) are powered up

Hpd

Set to TRUE if sink Hot Plug Detect is high

Set to FALSE if sink Hot Plug Detect is low

MonSen

Set to TRUE if sink monitor sense is high

Set to FALSE if sink monitor sense is low

OutputHdmi

Set to TRUE if the output is HDMI

Set to FALSE if the output is DVI

PllLocked

Set to TRUE if video PLL is locked

Set to FALSE if video PLL is not locked

VideoMuted

Set to TRUE if video output is blacked out

Set to FALSE if video output is not muted

ClearAVMute

Set to TRUE if Clear AV mute is being sent to sink

Set to FALSE if Clear AV mute is not being sent

SetAVMute

Set to TRUE if set AV mute is being sent to sink

Set to FALSE if set AV mute is not being sent

AudioRep

Set to TRUE if audio output is enabled

Set to FALSE if audio output is disabled (muted)

SpdifEnable

Set to TRUE if SPDIF interface is enabled

Set to FALSE if SPDIF interface is disabled

I2SEnable

Bits 0-3 will be set according to I2S channel 0-3 enable state.

If a channel is enabled, the corresponding bit will be set to 1, otherwise it will be 0.

DetectedVic

Video Identification code detected by HDMI TX

HdcpErr

Set to the last received HDCP error.

Return Value

ATVERR_OK

Remarks

None.

4.42 ADIAPI_TXMUTEAUDIO

Description

This API can be used to Mute or un-mute HDMI TX audio output. The audio will be muted by disabling audio sample packets output from HDMI TX.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxMuteAudio (*BOOL* Mute)

ATV_ERR ADIAPI_TxMuteAudioN (*UCHAR* DeviceIndex**, ...**)

Parameters

Mute

TRUE to mute HDMI TX audio output

FALSE to un-mute HDMI TX audio output.

Return Value

ATVERR_OK

Remarks

Audio input to HDMI TX can also be disabled by the ADIAPI_TxAudInputEnable API.

4.43 ADIAPI_TXMUTEVIDEO

Description

This API can be used to Mute or un-mute HDMI TX video output. The video will be muted by sending black level on all TMDS lines.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxMuteVideo (*BOOL* Mute)

ATV_ERR ADIAPI_TxMuteVideoN (*UCHAR* DeviceIndex**, ...**)

Parameters

Mute

TRUE to mute HDMI TX video output

FALSE to unmute HDMI TX video output

Return Value

ATVERR_OK

Remarks

None.

4.44 ADIAPI_TXSETAVMUTE

Description

Sets or clears general control packet AVMUTE signal.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetAvmute (*TX_AVMUTE* State)

ATV_ERR ADIAPI_TxSetAvmuteN (*UCHAR* DeviceIndex**, ...**)

Parameters

State

The required state of Set AVMUTE and Clear AVMUTE signals in the general control packet. This can be one of the following values:

TX_AVMUTE_ON

Set SET_AVMUTE and clear CLEAR_AVMUTE

TX_AVMUTE_OFF

Clear SET_AVMUTE and set CLEAR_AVMUTE

TX_AVMUTE_NONE

Clear both SET_AVMUTE and CLEAR_AVMUTE

TX_AVMUTE_BOTH

Set both SET_AVMUTE and CLEAR_AVMUTE. Note that this setting is not allowed by HDMI.

Return Value

ATVERR_OK

Remarks

None.

4.45 ADIAPI_TXGETAVMUTE

Description

Gets the Set/Clear state of the general control packet AVMUTE signal.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetAvmute (***TX_AVMUTE*** *State)

ATV_ERR ADIAPI_TxGetAvmuteN (***UCHAR*** DeviceIndex**, ...**)

Parameters

State

Pointer to AVMUTE state. This can be one of the following values:

TX_AVMUTE_ON

Set SET_AVMUTE and clear CLEAR_AVMUTE

TX_AVMUTE_OFF

Clear SET_AVMUTE and set CLEAR_AVMUTE

TX_AVMUTE_NONE

Clear both SET_AVMUTE and CLEAR_AVMUTE

TX_AVMUTE_BOTH

Set both SET_AVMUTE and CLEAR_AVMUTE. Note that this setting is not allowed by HDMI.

Return Value

ATVERR_OK

Remarks

None.

4.46 ADIAPI_TXENABLEPACKETS

Description

Enable or disable HDMI TX sending of selected packets and InfoFrames. Once a packet send is enabled, HDMI TX will continue to send the packet periodically on intervals as specified in HDMI specification.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxEnablePackets (*UINT16* Packets, *BOOL* Enable)

ATV_ERR ADIAPI_TxEnablePacketsN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packets

Packets that need to enabled or disables ORed together. Possible values are:

PKT_AV_INFO_FRAME

PKT_AUDIO_INFO_FRAME

PKT_GC_PACKET

PKT_ACP_PACKET

PKT_SPD_PACKET

PKT_GMD_PACKET

PKT_ISRC1_PACKET

PKT_ISRC2_PACKET

PKT_MPEG_PACKET

PKT_VS_PACKET

PKT_ACR_PACKET

PKT_AUDIO_CHANNEL_STATUS

PKT_AUDIO_SAMPLE_PACKET

PKT_ALL_PACKETS

Enable

Set to TRUE to enable sending of packets specified in the "Packets" parameter

Set to FALSE to disable sending of packets specified in the "Packets" parameter

Return Value

ATVERR_OK

Remarks

None.

Example

```
/* Disable all packets send */
ADIAPI_TxEnablePackets (PKT_ALL_PACKETS, FALSE);
/* Enable sending of AV info frame */
ADIAPI_TxEnablePackets (PKT_AV_INFO_FRAME, TRUE);
```

4.47 ADIAPI_TXGETENABLEDPACKETS

Description

Gets information about which packets are currently enabled.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxGetEnabledPackets (*UINT16* *Packets)

ATV_ERR ADIAPI_TxGetEnabledPacketsN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packets

Pointer to receive information about which packets are currently enabled. Enabled packets are returned as bit values ORed together. Possible values are:

PKT_AV_INFO_FRAME

PKT_AUDIO_INFO_FRAME

PKT_GC_PACKET

PKT_ACP_PACKET

PKT_SPD_PACKET

PKT_GMD_PACKET

PKT_ISRC_PACKET

PKT_ACR_PACKET

PKT_AUDIO_CHANNEL_STATUS

PKT_AUDIO_SAMPLE_PACKET

Return Value

ATVERR_OK

Remarks

None.

4.48 ADIAPI_TXSENDAVINFOFRAME

Description

Send AV info-frame to the sink device. The AV info-frame packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendAVInfoframe (***UCHAR*** *Packet, ***UCHAR*** Size)

ATV_ERR ADIAPI_TxSendAVInfoframeN (***UCHAR*** DeviceIndex**, ...**)

Parameters

Packet

Pointer to the AV InfoFrame HB0 (Header Byte 0) This AV info-frame will be sent as-is to the sink device, except the VIC and PR (pixel repeat) fields. The VIC and PR fields sent to the sink will depend on the pixel repeat mode setting using the APIs ADIAPI_TxSetManualPixelRepeat and ADIAPI_TxSetAutoPixelRepeat.

Size

Byte size of the AV InfoFrame (must be 16)

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the AV info-frame packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the AV info-frame packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.49 ADIAPI_TXSENDAUDIOINFOFRAME

Description

Send Audio info-frame to the sink device. The Audio info-frame packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet. This function will automatically set the size of the right-justified I2S word size if the supplied audio info-frame sample size is not 0.

For HBR audio, the CA and CC fields of the audio info-frame must be set to 0x1F and 0x07 respectively, unless the ADIAPI_TxSetAudioInterface API is called afterward to adjust those two fields.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendAudioInfoframe (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendAudioInfoframeN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the audio info-frame packet HB0 (Header Byte 0)

Size

Byte size of the packet (Must be 13)

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the Audio info-frame packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the Audio info-frame packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

For HBR audio, the CA and CC fields in the audio info-frame must be set t 0x1F and 0x07 respectively. This is automatically done by the ADIAPI_TxSetAudioInterface API. However, any call to ADIAPI_TxSendAudioInfoframe will overwrite the CA and CC fields according to the values supplied by the caller.

4.50 ADIAPI_TXSENDACPPACKET

Description

Send ACP packet to the sink device. The ACP packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendACPPacket (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendACPPacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the packet HB0 (Header Byte 0)

Size

Byte size of the packet

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the ACP packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the ACP packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.51 ADIAPI_TXSENDSPDPACKET

Description

Send SPD packet to the sink device. The SPD packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendSPDPacket (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendSPDPacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the packet HB0 (Header Byte 0)

Size

Byte size of the packet

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the SPD packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the SPD packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.52 ADIAPI_TXSENDISRC1PACKET

Description

Send ISRC1 packet to the sink device. The ISRC1 packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendISRC1Packet (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendISRC1PacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the 31-byte ISRC1 packet HB0 (Header Byte 0)

Size

Byte size of the packet

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the ISRC1 packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the ISRC1 packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.53 ADIAPI_TXSENDISRC2PACKET

Description

Send ISRC2 packet to the sink device. The ISRC2 packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendISRC2Packet (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendISRC2PacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the 31-byte ISRC2 packet HB0 (Header Byte 0)

Size

Byte size of the packet

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the ISRC2 packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the ISRC2 packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.54 ADIAPI_TXSENDGMDPACKET

Description

Send Gamut Metadata packet to the sink device. The GMD packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendGMDPacket (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendGMDPacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the packet HB0 (Header Byte 0)

Size

Byte size of the packet.

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the Gamut Metadata packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the GMD packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.55 ADIAPI_TXSENDMPEGPACKET

Description

Send MPEG packet to the sink device. The MPEG packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendMpegPacket (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendMpegPacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the packet HB0 (Header Byte 0)

Size

Byte size of the packet.

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the MPEG packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the MPEG packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.56 ADIAPI_TXSENDSPARE1PACKET

Description

Send any user-defined packet to the sink device. The TX Device has a general-purpose packet memory that can be filled with any data to be sent to the sink. One use of such packets is to send vendor-specific info-frame. The VS (Vendor-specific) packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendSpare1Packet (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendSpare1PacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the packet HB0 (Header Byte 0)

Size

Byte size of the packet.

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the user-defined spare-1 packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the VS packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.57 ADIAPI_TXSENDSPARE2PACKET

Description

Send any user-defined packet to the sink device. The TX Device has a general-purpose packet memory that can be filled with any data to be sent to the sink. One use of such packets is to send vendor-specific info-frame. The VS (Vendor-specific) packet repeat must be enabled using ADIAPI_TxEnablePackets to be able to send this packet.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSendSpare2Packet (*UCHAR* *Packet, *UCHAR* Size)

ATV_ERR ADIAPI_TxSendSpare2PacketN (*UCHAR* DeviceIndex**, ...**)

Parameters

Packet

Pointer to the packet HB0 (Header Byte 0)

Size

Byte size of the packet.

Return Value

ATVERR_OK

ATVERR_INV_PARM

Remarks

This API will set the user-defined spare-2 packet in the HDMI TX internal memory, to be sent to the sink device. The packet will only be sent if the VS packet repeat is enabled. See ADIAPI_TxEnablePackets for more details.

4.58 ADIAPI_TXARCSETMODE

Description

Enable/Disable Audio Return Channel (ARC) operation. This API is valid only on TX or transceiver-type devices that support ARC feature.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxArcSetMode (*TX_ARC_MODE* Mode)

Parameters

Mode

Select required ARC mode. This can be one of the following:

TX_ARC_SINGLE

For single mode

TX_ARC_COMMON

For common mode

TX_ARC_OFF

To turn ARC feature off

Return Value

ATVERR_OK

ATVERR_NOT_AVAILABLE

Remarks

This API can only be used for TX or transceiver type of devices that support Audio Return Channel functionality. It has no effect for other devices.

4.59 ADIAPI_TXCECSETACTIVEDEVICE

Description

In systems with multiple TX devices, the TX library allow only one TX CEC engine to be active at any given time. This API can be used to select which TX device's CEC engine is to be used.

This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecSetActiveDevice (*UCHAR* DeviceIndex)

Parameters

DeviceIndex

This value defines the TX device index to be used for CEC processing. This value can range from 0 to TX_NUM_OF_DEVICES.

Return Value

ATVERR_OK	Operation completed successfully
ATVERR_INV_PARM	Index out of range

Remarks

This function will disable (power-down) the previously active CEC engine and enable the CEC engine selected by the DeviceIndex parameter. There is no need to call ADIAPI_TxCecEnable afterwards.

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.60 ADIAPI_TXCECENABLE

Description

This API enables or disables the CEC controller engine. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecEnable(*BOOL* Enable)

Parameters

Enable

TRUE to enable the CEC controller

FALSE to disable the CEC controller

Return Value

ATVERR_OK	Operation completed successfully

Remarks

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.61 ADIAPI_TXCECRESET

Description

This API resets the CEC controller engine. It is called as part of the ADIAPI_TxCecEnable API. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecReset(void)

Parameters

None

Return Value

ATVERR_OK Operation completed successfully

Remarks

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.62 ADIAPI_TXCECSENDMESSAGE

Description

Send a CEC message. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecSendMessage(***UCHAR*** *MsgPtr, ***UCHAR*** MsgLen)

Parameters

MsgPtr

Pointer to the CEC message to be sent

MsgLen

CEC message length

Return Value

ATVERR_OK

Message is queued to be sent. The application must poll the ADIAPI_TxCecGetStatus API to determine when send is completed before calling any further APIs.

ATVERR_FAILED

If CEC controller is busy. The message will not be sent.

ATVERR_INV_PARM

If MsgLen is larger than maximum message size (16 bytes)

Remarks

This API return immediately. If the return value is ATVERR_OK, The application must poll the CEC engine using the ADIAPI_TxCecGetStatus API to determine if message send was completed successfully.

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.63 ADIAPI_TXCECRESENDLASTMESSAGE

Description

This API sends the last CEC massage again. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecResendLastMessage (***void***)

Parameters

None

Return Value

ATVERR_OK

Message is queued to be sent. The application must poll ADIAPI_TxCecGetStatus API to determine when send is completed before calling any further APIs.

ATVERR_FAILED

If CEC controller is busy, the message will not be sent.

ATVERR_FAILED

If CEC controller is busy, the message will not be sent.

Remarks

This API return immediately. If the return value is ATVERR_OK, The application must poll the CEC engine using the ADIAPI_TxCecGetStatus API to determine if message send was completed successfully.

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.64 ADIAPI_TXCECREADMESSAGE

Description

Read a CEC message if available. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecReadMessage(***UCHAR*** *MsgPtr, ***UCHAR*** *MsgLen)

Parameters

MsgPtr

Pointer to a buffer to receive CEC message (maximum 16 bytes)

MsgLen

Pointer to receive CEC message length

Return Value

ATVERR_OK	Message read into MsgPtr parameter
ATVERR_FAILED	If no message is available

Remarks

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.65 ADIAPI_TXCECSETLOGICALADDR

Description

This API sets the device logical address. Up to 3 different logical addresses can be set for the device. To inquire about logical addresses available for use (not allocated) the application can use the ADIAPI_TxCecAllocateLogAddr API. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetLogicalAddr(***UCHAR*** LogAddr, ***UCHAR*** DevID, ***BOOL*** Enable)

Parameters

LogAddr

Logical address to be set for the device

DevID

The device to set the logical address to. Up to 3 different devices can be used. This value can be 0, 1 or 2.

Enable

Enable or disable the logical address

Return Value

ATVERR_OK	Logical address set
ATVERR_INV_PARM	If DevID is larger than 2

Remarks

See ADIAPI_TxCecAllocateLogAddr for more information.

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.66 ADIAPI_TXCECALLOCATELOGADDR

Description

This API checks the availability of logical addresses. This API is available only if CEC support is included by setting the configuration switch, TX_INCLUDE_CEC.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecAllocateLogAddr(***UCHAR*** *LogAddrList)

Parameters

LogAddrList

Pointer to a prioritized list of logical addresses that the device will try to obtain, terminated by 0xff.

Return Value

ATVERR_OK

Operation is queued to be processed. The application must poll the ADIAPI_TxCecGetStatus API to determine the logical address that can be used.

ATVERR_FAILED

If CEC controller is busy. The operation will not be completed.

Remarks

This API return immediately. If the return value is ATVERR_OK, the application must poll the CEC engine using the ADIAPI_TxCecGetStatus API to determine if the operation is completed and to obtain the available logical address. It is the responsibility of the application to set the device logical address using the ADIAPI_TxCecSetLogicalAddr API.

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.67 ADIAPI_TXCECGETSTATUS

Description

This API returns the status of the last performed CEC operation. Some CEC APIs return immediately and the application is required to poll CEC state to determine if the operation was successful.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxCecGetStatus (***UCHAR*** *Status)

Parameters

Status

Pointer to receive status on CEC engine last operation. The value returned in the Status parameter depends on the return value of this API.

If the return value is ATVERR_FAILED, the Status parameter will be set to the error code indicating the cause of failure.

If the return value is ATVERR_OK, the Status parameter will be set according to the last requested operation as shown in Table 11.

Table 11. Status Parameter as per last Requested Operation

API	Status Value on Success
ADIAPI_TxCecEnable	0
ADIAPI_TxCecReset	0
ADIAPI_TxCecSendMessage	0
ADIAPI_TxCecResendLastMessage	0
ADIAPI_TxCecAllocateLogAddr	First available (not allocated) logical address in the logical address list supplied to the API. If no logical address is available, this value will be 0xFF

Return Value

ATVERR_OK

Last CEC operation was completed successfully. Result is returned in the "Status" parameter.

ATVERR_NOT_AVAILABLE

Last CEC operation is still in progress

ATVERR_FAILED

Last CEC operation failed. Error code is returned in the "Status" parameter as:

- CEC_ERR_TX_TIMEOUT
- CEC_ERR_ARB_LOST

Remarks

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.68 ADIAPI_TXSETVIDEOCLKDELAY

Description

Sets delay for input video clock.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxSetVideoClkDelay (***TX_VIDEO_CLK_DELAY*** Delay)

Parameters

Delay

Delay format. This can be one of the followings:

TX_VIDEO_CLK_DELAY_M_1200PS

Delay by -1200 psec

TX_VIDEO_CLK_DELAY_M_800PS

Delay by -800 psec

TX_VIDEO_CLK_DELAY_M_400PS

Delay by -400 psec

TX_VIDEO_CLK_DELAY_NULL

No delay

TX_VIDEO_CLK_DELAY_P_400PS

Delay by 400 psec

TX_VIDEO_CLK_DELAY_P_800PS

Delay by 800 psec

TX_VIDEO_CLK_DELAY_P_1200PS

Delay by 1200 psec

TX_VIDEO_CLK_DELAY_P_1600PS

Delay by 1600 psec

Return Value

ATVERR_OK

Remarks

This API is not supported by transceiver type devices such as the ADV7850.

4.69 ADIAPI_TXHDCPENABLED

Description

Determine if HDCP is currently enabled.

Synopsis

#include "adi_apis.h"

ATV_ERR ADIAPI_TxHdcpEnabled (***BOOL*** *HdcpOn)

Parameters

HdcpOn

Pointer to the current state of HDCP

- Function will set this to TRUE when called if HDCP is enabled
- Function will set this to FALSE when called if HDCP is disabled

Return Value

ATVERR_TRUE	HDCP enabled
ATVERR_FALSE	HDCP disabled

Remarks

None.

4.70 ADIAPI_TXCECSENDMESSAGEOUT

Description

Send out CEC message in buffer.

Synopsis

ATV_ERR ADIAPI_TxCecSendMessageOut (***void***)

Parameters

Return Value

ATVERR_OK	Message sent successfully
ATVERR_FAILED	CEC controller busy

Remarks

This API can only be used for devices that support CEC functionality. It has no effect for other devices.

4.71 ADIAPI_MONITORACTIVETX

Description

Scans status of multiple Tx devices. It is usually called periodically every 10ms. It must be called to switch Tx devices.

Synopsis

ATV_ERR ADIAPI_MonitorActiveTX (***void***)

Parameters

Return Value

ATVERR_OK

Remarks

None.

4.72 ADIAPI_TXNORMALOP

Description

Return Tx to Normal/ Default Configuration with Manual HDMI DVI selection.

Synopsis

ATV_ERR ADIAPI_TxNormalOp (void);

Parameters

Return Value

ATVERR_OK

Remarks

Targets ADV8002 and ADV8003 Devices only.

4.73 ADIAPI_TXADJUSTFREQRANGE(UINT16 FREQMHZ)

Description

An API to Adjust the Clock Data Timing relationship based on the given TMDS Frequency.

Synopsis

ATV_ERR ADIAPI_TxAdjustFreqRange (UINT16 FreqMHz);

Parameters

FreqMhz TMDS Frequency of Video being processed.

Return Value

ATVERR_OK

Remarks

Targets ADV8002 and ADV8003 Devices only.

4.74 ADIAPI_TXSETAUTOTMDSPDNMODE

Description

An API to Configure the Tx auto power-down mode for Transceiver type devices.

Synopsis

ATV_ERR ADIAPI_TxSetAutoTmdsPdnMode(BOOL HsyncDet, BOOL RxTmdsDet);

Parameters

HSyncDet	Boolean to indicate HS Detection influences auto power down
RxTmdsDet	Boolean to indicate RxTmds Detection Influences auto power down

Return Value

ATVERR_OK

Remarks

Targets 7850 only.

4.75 ADIAPI_TXDISABLEAUTOPJCHECK

Description

An API to turn off the auto enabling of HDCP 1.1 features, regardless of BCAPs status. This feature is applicable to ADV7511 latest devices.

Synopsis

ATV_ERR ADIAPI_TxDisableAutoPjCheck (BOOL Disable);

Parameters

Disable Boolean to enable and disable Automatic 1.1 Encryption based on BCAPS

Return Value

ATVERR_OK

Remarks

Targets 7511 only.

5 NOTIFICATION EVENTS

The Transmitter library provide the option to notify the application of any changes in operating condition, thus relieving the application from polling the library for status changes. The application must define a function that will be called from the library (from inside ADIAPI_TxIsr) upon any change in operating conditions. The prototype for this function is as follows:

UINT16 App_TxNotificationEvent (TX_EVENT EventID, UINT16 EventSize, void *EventData)

where:
EventID is a unique value that identifies the operating condition that has changed.
EventSize either specifies the size of the data pointed to by the EventData parameter or carries some other information related to the event. This value is event dependent as listed in Table 12.
EventData is a pointer to a buffer that contains event-specific information. If the event does not have an associated data, this value will be undefined. The table below lists all events and their associated data.

The name of the notification function is defined by the application. The configuration macro TX_CALLBACK_FUNCTION should be used to define the name selected by the application for the notification function as in the following example:

```
#define TX_CALLBACK_FUNCTION          App_TxNotificationEvent
```

Depending on the enabled events (refer to the ADIAPI_TxSetEnabledEvents API) the application will be notified via the above function of the events listed in Table 12.

Table 12. Tx Event Details

Interrupt	EventID	EventSize	EventData
HPD Low	TX_EVENT_HPD_CHG	0	Pointer to BOOL value FALSE
HPD High	TX_EVENT_HPD_CHG	0	Pointer to BOOL value TRUE
Rx Sense Low/High	TX_EVENT_MSEN_CHG	0	Pointer to BOOL value: FALSE if monitor sense is LOW TRUE if monitor sense is HIGH
EDID Ready	TX_EVENT_EDID_READY when a new EDID segment is fully read	Index of the segment returned in EventData (0, 1, 2, etc)	Pointer to buffer containing the EDID segment read from sink
BKSV ready	TX_EVENT_BKSV_READY when all BKSVs are received	Number of BKSVs returned in EventData	Pointer to a concatenated list of all downstream BKSVs
HDCP Authenticated	TX_EVENT_HDCP_AUTHENTICATED	0	NULL
HDCP Error	TX_EVENT_HDCP_ERROR	0	Pointer to TX_HDCP_ERR value containing the error code
Vsync Edge	TX_EVENT_VSYNC_EDGE	0	NULL
Audio FIFO Full	TX_EVENT_AUDIO_FIFO_FULL	0	NULL
Embedded Sync Polarity Error	TX_EVENT_EMB_SYNC_ERROR	0	NULL
CEC TX Ready	TX_EVENT_CEC_TX_READY	0	NULL
CEC RX Ready	TX_EVENT_CEC_RX_READY	CEC message size	CEC message
CEC TX Retry timeout	TX_EVENT_CEC_ERR_TIMEOUT	0	NULL
CEC TX Arbitration Lost	TX_EVENT_CEC_ERR_ARB_LOST	0	NULL

The return value of the notification event function is not currently used and should always be set to 0.

For multiple devices, the application should call the ADIAPI_TxGetDeviceIndex API at the start of the notification function to get the device index that generated the notification.

NOTES

NOTES

NOTES

I²C refers to a communications protocol originally developed by Philips Semiconductors (now NXP semiconductors).