Getting started with HDMI TX Driver

Version 0.32

Table of content

Table of figures

1. Introduction

1.1 Purpose and Scope

This document aims at describing how to use the HDMI Tx driver. The intended audience is anyone who wants to make use of its application programming interface in order to drive a TDA998x HDMI transmitter.

1.2 How to integrate HDMI Tx driver in your application in a nutshell

1. Unpack the delivered zip file, mandatory files in order to build up your own HDMI Tx application are the following (Refer to 0) :

Shared Include files
Directory HdmiTx/sde2/inc/

HIGH LEVEL DRIVER	
TDA9984 & TDA9989/88	**TDA9983 & TDA9981**
Directory HdmiTx/sde2/comps/tmdlHdmiTx/inc	Directory HdmiTx/sde2/comps/tmdlTDA9983/inc
Directory HdmiTx/sde2/comps/tmdlHdmiTx/src	Directory HdmiTx/sde2/comps/tmdlTDA9983/src
Directory HdmiTx/sde2/comps/tmdlHdmiTx/cfg	Directory HdmiTx/sde2/comps/tmdlTDA9983/cfg

LOW LEVEL DRIVER		
Directory HdmiTx/sde2/comps/tmbslHdmiTx/inc		
TDA9984	**TDA9989/88**	**TDA9983 & TDA9981**
HdmiTx/sde2/comps/tmbslTDA9984\inc	HdmiTx/sde2/comps/tmbslTDA9989\inc	HdmiTx/sde2/comps/tmbslTDA9983\inc
HdmiTx/sde2/comps/tmbslTDA9984\src	HdmiTx/sde2/comps/tmbslTDA9989\src	HdmiTx/sde2/comps/tmbslTDA9983\src

2. Write your OS / No OS wrapper implementation (Refer to 3.1.1)
3. Write your I^2C wrapper implementation (Refer to 3.1.3)
4. Write your HDMI Tx configuration file (Refer to 3.11)
5. Determine whether you will use the driver in ISR or polling mode (Refer to 3.1.2)
6. Write your own HDMI TX application by looking at source code example in provided test application (Refer to 3.14.4)
7. For complete explanations regarding APIs parameters refer to the HTML API reference in the delivered zip.

2. Overall Description

2.1 System architecture

Source HDMI transmitters may be used in various consumers electronic applications such as digital set-top boxes, DVD players/recorders, camcorders in order to output uncompressed digital audio/video streams at high bitrates to cope with today's HD requirements.

Figure 1 depicts the architecture of HDMI systems:

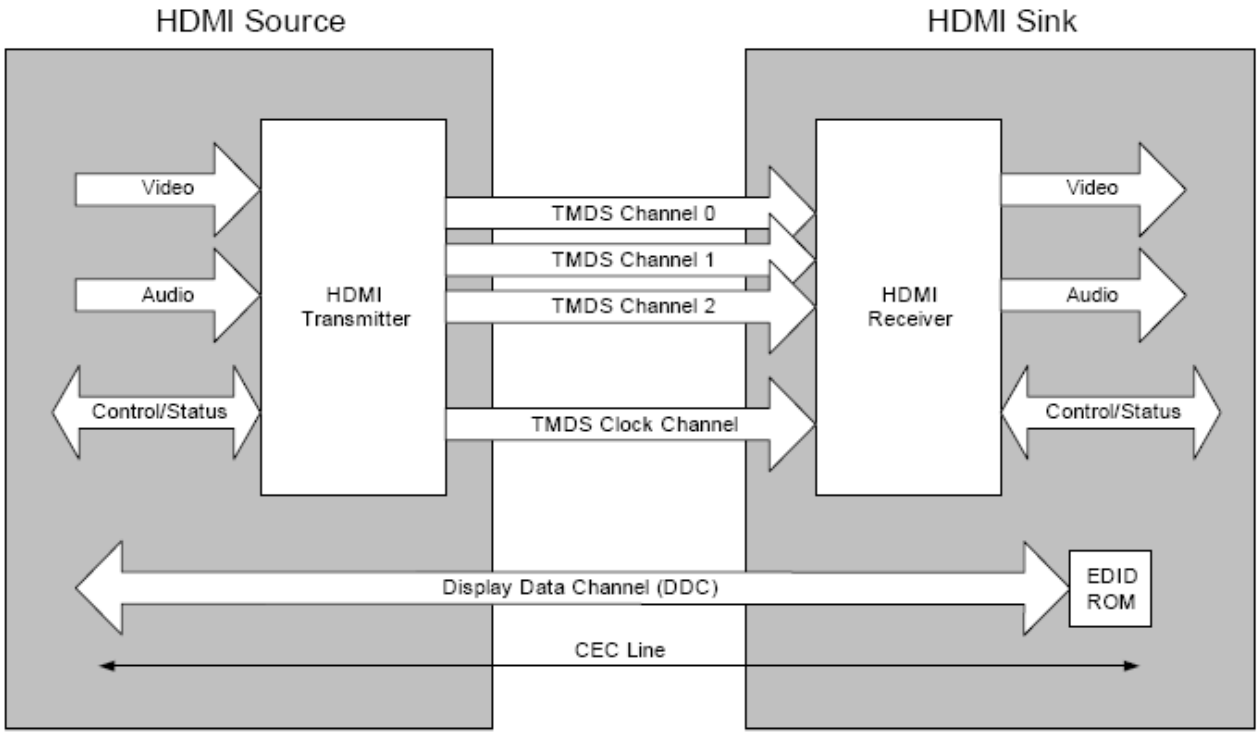


Figure 1 HDMI System Architecture

HDMI system architecture is defined to consist of Sources and Sinks. A given device may have one or more HDMI inputs and one or more HDMI outputs. Each HDMI input on these devices shall follow all of the rules for an HDMI Sink and each HDMI output shall follow all of the rules for an HDMI Source.

As shown in Figure 1, the HDMI cable and connectors carry four differential pairs that make up the TMDS data and clock channels. These channels are used to carry video, audio and auxiliary data. In addition, HDMI carries a DDC channel.

The DDC is used by the Source to read the Sink's Enhanced Extended Display Identification Data (E-EDID) in order to discover the Sink's configuration and/or capabilities. Besides this communication bus is also used for HDCP authentication.

The optional CEC protocol provides high-level control functions between all of the various audiovisual products in a user's environment.

2.2 Software architecture

Software architecture is depicted in Figure 2:

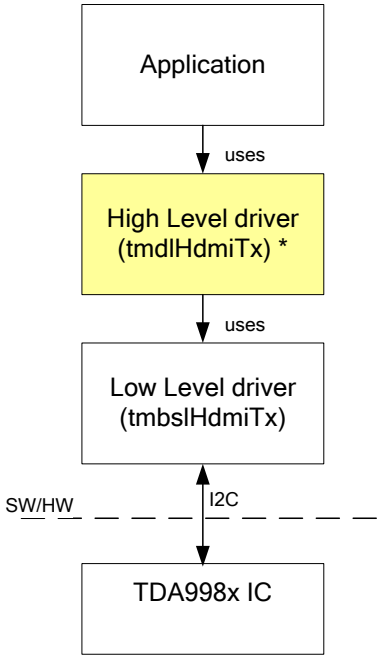


Figure 2 HDMI TX software architecture

Drivers are split in two parts: the low level driver (tmbslHdmiTx) that provides a first level of abstraction of the underlying hardware and the high level driver (tmdlHdmiTx or tmdlTDA9983) that provides a high level of functionality.

Low level driver offers a wide API allowing control of most IC parameters. It can be directly used by customers that want full control of the device or that want to optimize their memory usage. High level driver offers a restricted API for customers that are always using the device in a “standard” configuration and want to optimize their software development costs.

2.3 Hardware architecture

Hardware architecture is depicted in Figure 3:

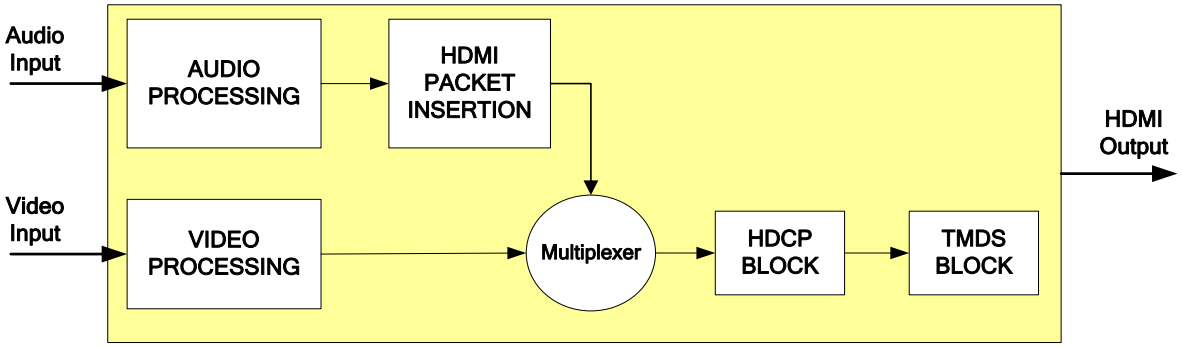


Figure 3 HDMI TX hardware architecture

The audio/video processing blocks provide some facilities to transform incoming audio/video data to cope with Sink configuration and/or capabilities. Video pixel, packet and control data are multiplexed together before being encrypted and send through the TMDS channels (HDMI Output).

2.4 Data flow

Data flow is depicted in Figure 4:

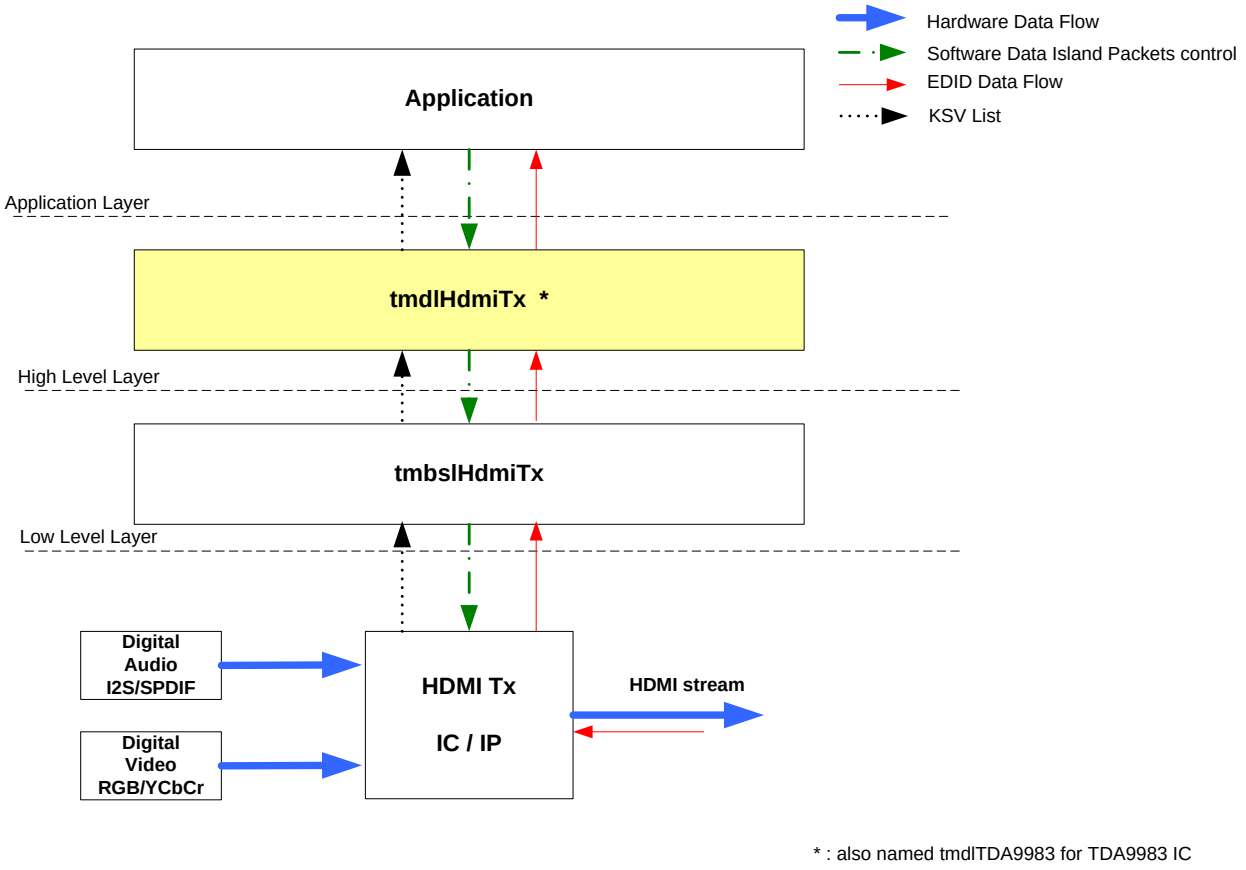


Figure 4 tmdlHdmiTx data flow

Data is retrieved from digital audio/video sources and converted into an HDMI [HDMI_SPEC] compliant stream thanks to a hardware data path.

The application controls data island packets insertion by enabling (or disabling) packets insertion and providing associated packets payload.

EDID data may be retrieved by the Application in order to choose video / audio setting that match display possibilities.

Receiver KSV list is readable in order to manage HDCP revocation list.

3. Software usage

3.1 Software context

3.1.1 OS / No OS Mode

HDMI TX driver may be used in an application that makes use of an operating system or not. An OS wrapper has to be written in file *tmdlHdmiTx_IW.c* to implement prototypes defined in *tmdlHdmiTx_IW.h*.

3.1.1.1 OS Mode

When an OS is used the following rules shall be followed:

- tmdlHdmiTx component APIs shall be called from a task context.
- Only API *tmdlHdmiTxHandleInterrupt* may be called from ISR context, since no low level driver calls are allowed in interrupt context, the internal processing of this call is deported under a task context.
- Per instance the OS mode requires: 2 tasks, 1 semaphore, and 1 command queue.

3.1.1.2 No OS Mode

In this context, tmdlHdmiTx is not sharing CPU resources with other drivers and no tasks or other OS objects are used.

When there is no OS the following rules shall be followed:

- Polling mode shall be used. (see chapter below)
- Application will have to call periodically function *tmdlHdmiTxCheck* (we recommend each 40 msec) in order to verify that the sink is still HDCP-capable. For this purpose this API provide a *timeSinceLastCall* parameter which provide a time base used to know when a check shall be done.

The compilation flag **TMFL_NO_RTOS** shall be defined in order to use the HDMI TX drivers in this mode. (Refer to 3.12)

3.1.1 Userland / Linux kernel Mode

HDMI Tx diver can be used in userland context or in Linux kernel context.

- Using HDMI driver from the user space is easier for non-Linux-expert, but needs to have I2C functions available from userland.
- The compilation flag **TMFL_LINUX_OS_KERNEL_DRIVER** shall be defined to use HDMI driver in Linux kernel mode.

3.1.2 Polling / interrupt Mode

HDMI Tx interruptions may be used or not. When there is no interrupt line connected to the host processor or when there is no OS a polling mode shall be used. That means that function *tmdlHdmiTxHandleInterrupt* shall be called by the Application periodically (each 40ms).

3.1.3 I2C wrapper

In order to read/write HDMI Tx IC register an I^2C driver has to be provided by the application, besides an I^2C wrapper has to be written in file *tmdlHdmiTx_cfg.c* to implement prototypes defined in *tmbslHdmiTx_types.h*.

3.1.4 Multi low level driver support

tmdlHdmiTx API is common to all HDMI Tx family. When an API is called and that the underlying hardware does not support it a TMDL_ERR_DLHDMITX_NOT_SUPPORTED error code is returned. Please refer to the provided datasheet for a list of supported features.

3.2 Software state diagram

Figure 5 shows the internal state machine of the tmdlHdmiTx (and tmdlTDA9983) component

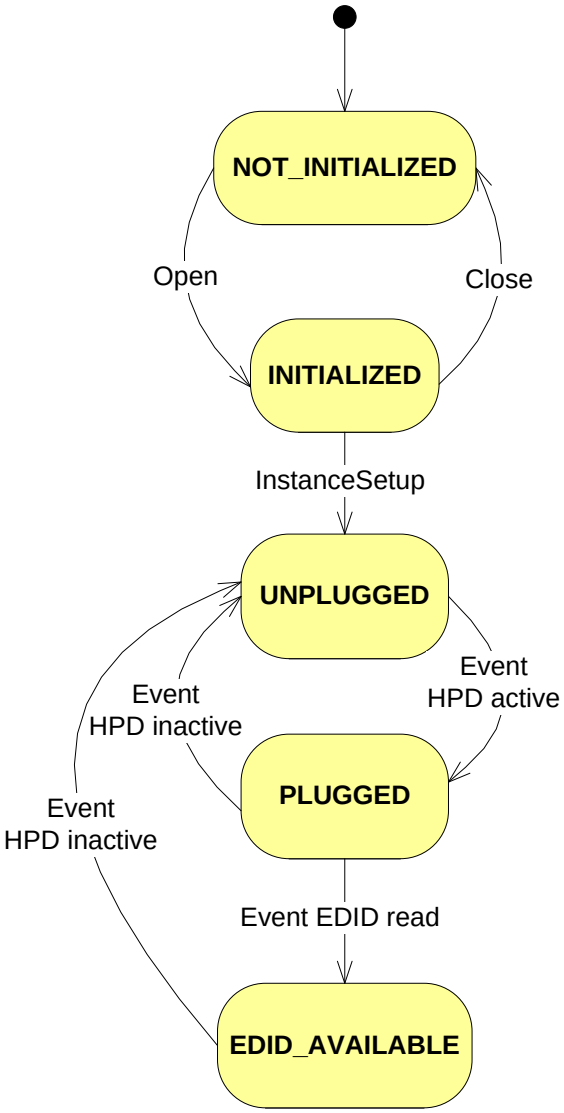


Figure 5 tmdlHdmiTx state diagram

Presented states have the following signification:

State name	State description
NOT_INITIALIZED	This is the default state of the driver before initialization.
INITIALIZED	The driver has been instantiated but not configured yet.
UNPLUGGED	The driver is configured. Receiver has not asserted a high voltage level yet.
PLUGGED	Transition from Low to High detected on the HPD input.
EDID_AVAILABLE	Receiver EDID has been received.

3.3 Software use cases

tmdlHdmiTx (and tmdlTDA9983) use cases are depicted in Figure 6 :

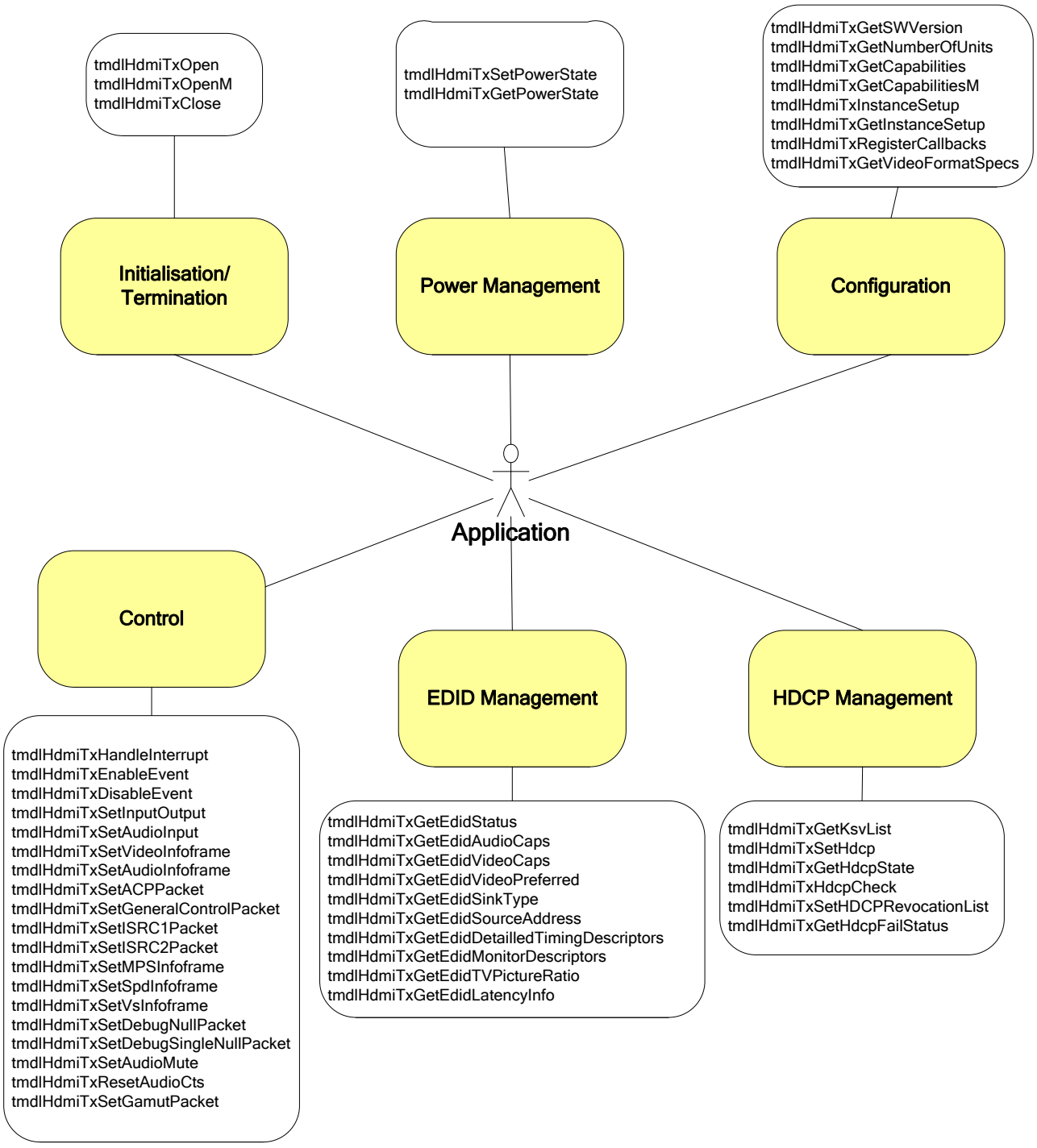


Figure 6 tmdlHdmiTx use cases

- **Initialization / Termination**

tmdlHdmiTxOpen and *tmdlHdmiTxClose* shall be called in order to respectively instantiate, terminate a given tmdlHdmiTx SW instance.

Note that *tmdlHdmiTxOpen* shall be called prior to any other APIs for a given instance (except *tmdlHdmiTxGetSWVersion*).

- **Configuration**

- Then, HDMI Tx instance shall be configured in order to define its behavior thanks to a *tmdlHdmiTxInstanceSetup* call.

  In this function application specifies whether

  - the instance has to cope with [SIMPLAY_HD] specification
  - the instance is part of a repeater device, which change its behavior regarding HDCP management.

  Moreover it is up to the application to provide a pointer to the memory space allocated for the EDID data and the size of this space in bytes. This size shall be a multiple of 128 since Application has to provide space for reading an entire number of EDIDs blocks.

- Application is informed of underlying HDMI Tx capabilities with a call to *tmdlHdmiTxGetCapabiliites*

Capability name	Capability Description
deviceVersion	HDMI Tx IC chip family
fastI2C	Boolean indicating fast I^2C handling
hdmiVersion	Supported HDMI specifications
audioPacket	Booleans indicating whether the chip supports or not: - HBR - DST - oneBitAudio
hdcp	Boolean indicating whether the chip supports HDCP or not.
scaler	Boolean indicating whether the chip has a scaler or not.

- Application has to register a callback function (*tmdlHdmiTxRegisterCallbacks*) in order to be informed of the followings events:

Event name	Event Description
HDCP_ACTIVE	The transmitter enables HDCP encryption.
HDCP_INACTIVE	An HDCP error has been detected or HDCP handling has been disabled. In case of a HDCP failure, user application can call function *tmdlHdmiTxGetHdcpFailStatus* in order to know at which HDCP step it occurs.
HPD_ACTIVE	Hot Plug detect signal is asserted. Upon detection of this signal the driver will internally trigger an EDID read procedure.
HPD_INACTIVE	Hot Plug detect signal is de-asserted.
RX_KEYS_RECEIVED	Receiver's KSV (Bksv) received
RX_DEVICE_ACTIVE [1]	Receiver connected to the transmitter and powered up.

[1] Note available for TDA 9983 IC

RX_DEVICE_INACTIVE [1]	Receiver disconnected or no more powered up.
EDID_RECEIVED	An EDID block has been received and read.

⇨ *tmdlHdmiTxRegisterCallbacks* shall be called before *tmdlHdmiTxInstanceSetup.*

Some events processing can be found in function `eventCallbackTx` in the provided example application.

- **Power Management**

API *tmdlHdmiTxSetPowerState* may be use to set the chip in standby and then to reduce the power consumption. In this mode the IC is not able to output HDMI stream but is still able to report HDMI connection status via HPD_ACTIVE / HPD_INACTIVE and RX_DEVICE_ACTIVE / RX_DEVICE_INACTIVE [1] events.

- **Control**

- Application shall notify the driver of interrupts via the API *tmdlHdmiTxHandleInterrupt*, the cause of the interrupt is parsed by the tmdlHdmiTx and is reported to the application in the form of an event. In the case where the host processor is not connected to the HDMI Tx IC interrupt line it is expected that the application triggers periodically (we recommend each 40 msec) the driver thanks to this API.

- Application defines the audio/video input characteristics and the chosen video output thanks to *tmdlHdmiTxSetInputOuput*. There is not check of consistency when this API is call; it's up to the application to set a video output that is supported by the connected display. Moreover thanks to this API it is possible to force the type of the sink (HDMI or DVI).

- When only audio characteristics have to be changed, API *tmdlHdmiTxSetAudioInput* can be called.

- Application controls data island packets insertion by enabling (or disabling) packets insertion and providing associated packets payload. This is done thanks to the following APIs:

API name	**Usage**
tmdlHdmiTxSetVideoInfoframe	Enable Auxiliary Video InfoFrame generation with provided data, or disable it. (*)
tmdlHdmiTxSetAudioInfoframe	Enable Audio InfoFrame generation with provided data, or disable it. (*)
tmdlHdmiTxSetACPPacket	Enable Audio Content Protection Packets generation with provided data, or disable it.
tmdlHdmiTxSetGeneralControlPacket	Enable General Control Packet generation with provided data, or disable it.
tmdlHdmiTxSetISRC1Packet	Enable International Standard Recording Code packet generation with provided data, or disable it.
tmdlHdmiTxSetISRC2Packet	Enable International Standard Recording Code packet generation with provided data, or disable it.
tmdlHdmiTxSetMPSInfoframe	Enable MPEG Source InfoFrame packet generation with provided data, or disable it.
tmdlHdmiTxSetSpdInfoframe	Enable Source Product Description InfoFrame packet generation with provided data, or disable it.
tmdlHdmiTxSetVsInfoframe	Enable Vendor Specific InfoFrame packet generation with

[1] Not avalable for TDA9983 IC

	provided data, or disable it.
tmdlHdmiTxDebugSetNullPacket	Enable Null Packet generation or disable it. For debug purpose only.
tmdlHdmiTxDebugSetSingleNullPacket	Send a single null packet. For debug purpose only.
tmdlHdmiTxSetGamutPacket	Enable Gamut Metadata packet generation with provided data, or disable it.
tmdlHdmiTxSetExtendedColorimetry	Enable or Disable extended colorimetry and its YCC quantization range settings in AVI infoFrame according to HDMI1.4. If the extended colerimetry needs to send Gamut Metadata packet, the above API *tmdlHdmiTxSetGamutPacket* is called. The latter only handles the extended colorimetries xvYCC601 and xvYCC709 without YCC quantization range indication but this API handles all the extended colorimetries xvYCC601, xvYCC709, sYCC601, AdobeYCC601, AdobeRGB with YCC quantization range indication. When an extended colorimetry is transmitted, the Tx has to be set to bypass mode, that is the output video mode is set to the same as the input video mode. This will quanrantee that no expected color space coversion is made internally by the Tx.

(*) Application may not call *tmdlHdmiTxSetVideoInfoframe* and *tmdlHdmiTxSetAudioInfoframe* to describe active audio and video since this is internally handled by an API call to *tmdlHdmiTxSetInputOutput* where the driver provide minimal required information. For video, video format (see definition in [CEA-861-D]) and video output (RGB or YUV422 or YUV444) are filled in.

For audio, an audio info frame packet structure is filled as such:

```
pktAif.ChannelCount       = aifChannelCountCode;  /* number of audio channels */
pktAif.CodingType         = 0; /* refer to stream header as specified in [HDMI_SPEC] */
pktAif.SampleSize         = 0; /* refer to stream header as specified in [HDMI_SPEC]*/
pktAif.SampleFreq         = refer to [HDMI_SPEC] Table Audio Infoframe Packet contents
pktAif.ChannelAlloc       = 0; /* ch 1 & 2 to speakers Front Left & Front Right */
pktAif.LevelShift         = 0; /* 0dB level shift */
pktAif.DownMixInhibit     = 0; /* down-mix stereo permitted */
```

- Application is able to mute audio On or Off with API *tmdlHdmiTxSetAudioMute*.
- Application is able to reset CTS value with *tmdlHdmiTxResetAudioCts* for instance when the audio sample rate change.
- Application is able to retrieve HPD and RXsense status from driver thanks to APIs *tmdlHdmiTxGetHPDStatus* and *tmdlHdmiTxGetRXSenseStatus*.

- **EDID Management**

Application is able to manage EDID related data with the following APIs:

API name	API usage
tmdlHdmiTxGetEdidStatus	Use this API to know EDID reading status: EDID_READ : All EDID blocks are available EDID_READ_INCOMPLETE : Some EDID blocks read but no sufficient memory provided by application to store all EDID blocks. EDID_ERROR_CHK_BLOCK_0 : Block 0 checksum error EDID_ERROR_CHK : Block 0 read but checksum errors in other blocks EDID_NOT_READ : EDID not read. EDID_STATUS_INVALID: Low level driver has returned and invalid status code.
tmdlHdmiTxGetEdidAudioCaps	Use this API to know receiver display audio capabilities where : - Encountered Audio Data Block are stored in an array - An audio flag integer is filled as such: b7 basic audio supported b6 AI supported (ACP supported) b5-b0 0
tmdlHdmiTxGetEdidVideoCaps	Use this API to know receiver display video capabilities where : - Encountered Video Data Block are stored in an array - A video flags integer is filled as such: b7 underscan supported[1] b6 YCbCr 4:4:4 supported b5 YCbCr 4:2:2 supported b4 undefined b3 undefined b2 undefined b1 xvYCC709 supported b0 xvYCC601 supported
tmdlHdmiTxGetEdidVideoPreferred	Get preferred video format from previously read EDID.
tmdlHdmiTxGetEdidSinkType	This API will return the sink type (DVI or HDMI). If the EDID read has failed or has not been done the returned type will be DVI.
tmdlHdmiTxGetEdidSourceAddress	Get HDMI Tx own physical address. Refer to [HDMI_SPEC] § Physical Address
tmdlHdmiTxGetEdidDetailledTimingDescriptors	This API will return a number of Detailed Timing Descriptors regarding what is available from previously read EDID.

[1] The Underscan mode displays the full video frame, which reveals content on the edges that is recorded but not shown.

tmdlHdmiTxGetEdidMonitorDescriptors	This API will return the Monitor Name Descriptor and the Monitor Range Limit descriptor, plus other descriptors if available.
tmdlHdmiTxGetEdidTVPictureRatio	This API will return the sink device's aspect ratio.
tmdlHdmiTxGetEdidLatencyInfo	This API will return the sink device's latency information. Refer to [HDMI_SPEC] §8.9.1

- **HDCP Management**

- The application has the ability to retrieve the receiver key list in order to check with a revocation key list with API *tmdlHdmiTxGetKsvList*.
- *tmdlHdmiTxCheck* API is proposed when this driver is used without any OS to check that the sink is still HDCP-capable. When this driver is used within an OS, this is handled internally thanks to the HDCP task.
- *tmdlHdmiTxGetHdcpState* provides information regarding the internal HDCP protocol state, where the following stages have been defined :

HDCP state	State meaning
CHECK_NOT_STARTED	HDCP algorithm not STARTED.
CHECK_IN_PROGRESS	HDCP algorithm STARTED.
CHECK_PASS	HDCP Encryption enabled and running on.
CHECK_FAIL_DRIVER_STATE	HDCP Encryption disabled. Refer to state A0 description in [HDCP].
CHECK_FAIL_DEVICE_T0	A problem has occurred during HDCP authentication.
CHECK_FAIL_DEVICE_RI	Comparison Ri = R'i failed
CHECK_FAIL_DEVICE_FSM	Comparison Pj = P'j failed

- *tmdlHdmiTxSetHdcp* API shall be used in order to enable/disable HDCP encryption. In order to enable HDCP encryption it shall be called after *tmdlHdmiTxSetInputOuput*.
- *tmdlHdmiSetHDCPRevocationList* API shall be used to set a list of KSV to revoke. This list is then checked on the fly or during HDCP authentication.
    - *on the fly:* the list is compared to the current ksv list, if a sink is in the list the API will return TMDL_DLHDMITX_HDCP_NOT_SECURE ; it is then up to the application to stop HDCP. If no sinks are found in the list TMDL_DLHDMITX_HDCP_SECURE is returned.
    - *during HDCP authentication*: if the KSV of specified sink is in the revocation list, HDCP authentication will fail for this sink. This API must be called prior to enabling the HDCP encryption.

  Note: **the KSV revocation list must be located in static memory since no copy of the list is made within the driver.**

- **3D Management**

- From sink to source, 3D capabilities of sink (like TV) can be read using the *tmdlHdmiTxGetEdidExtraVsdbData*

  API. Then refer to HDMI standard to parse the tmdhHdmiTxEdidExtraVsdbData structure and get the vendor specific data block with 3D capabilites.
- From the source to the sink, when 3D is used, the 3D video mode of HDMI output can be claimed in the vendor specific info frame as described in HDMI standard using the

tmdlHdmiTxSetVsinfoFrame API or using the *tmdlHdmiTxSetInputOutput* API that automatically set the vendor specific info frame.

3.4 Software sequence diagrams

3.5 Output audio/video

Hereafter we present a typical API sequence that shall be executed in order to output audio/video content. Thi sequence is used in the provided example application in function `HdmiTx_Init`

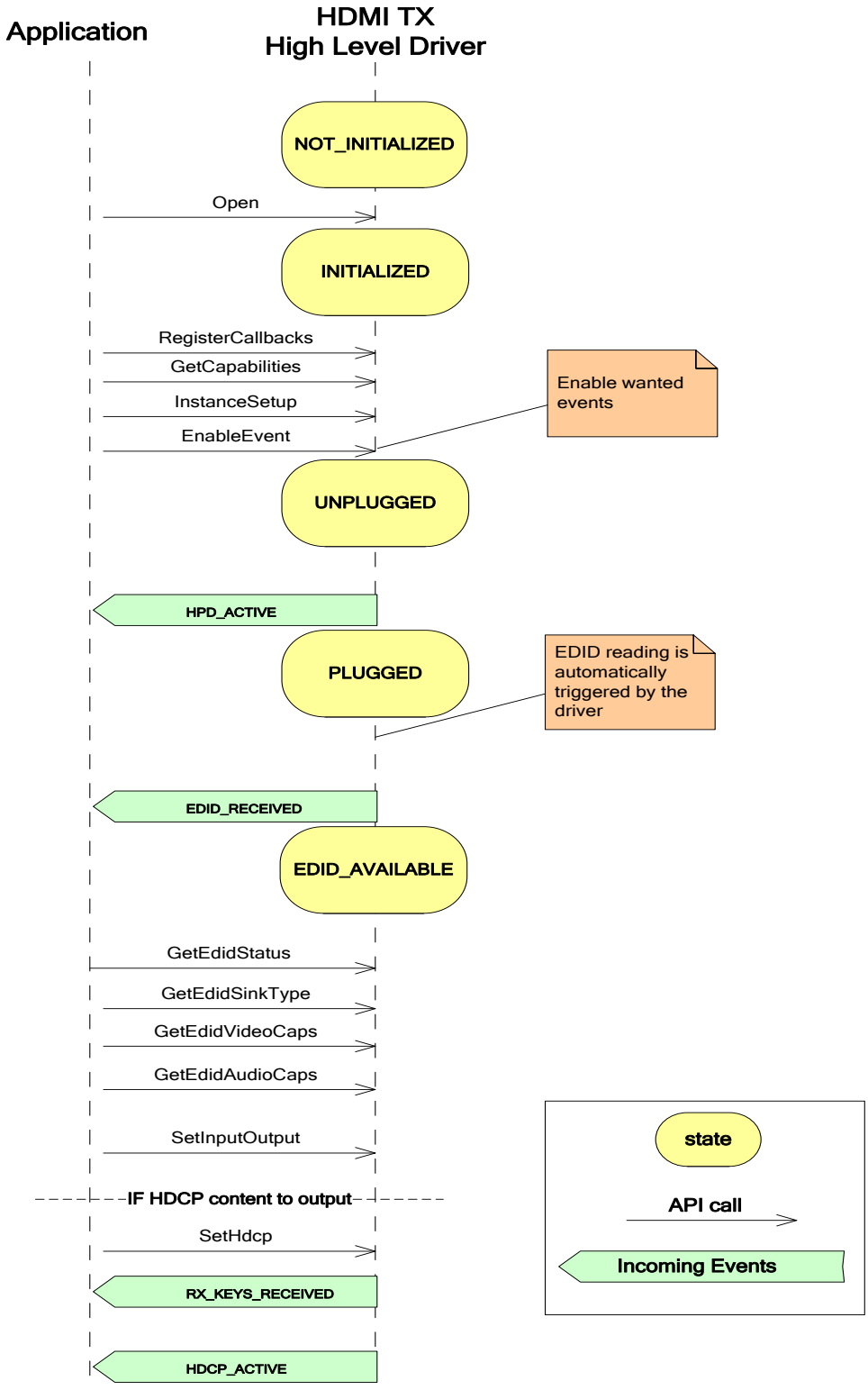


Figure 7 Enable HDMI Output sequence diagram

At first application has to initialize an HDMI Tx instance thanks to a call to *tmdlHdmiTxOpen*.

Then it shall register a callback function in order to be notified of HDMI driver events (*tmdlHdmiTxRegisterCallbacks*). It has also to retrieve the capabilities of the underlying HDMI Tx IC in order to know which functionalities are supported. Then wanted notification events are enabled with several calls to *tmdlHdmiTxEnableEvent.*

After that API *tmdlHdmiTxInstanceSetup* is called to configure the driver. Depending on *tmdlHdmiTxHandleInterrupt* call mode (via host processor interrupts or periodically from a task context) events are reported to the application regarding the receiver activity and current HDMI Tx status.

When EDID has been read at driver level, Application checks the result of the EDID reading process via a call to *tmdlHdmiTxGetEdidAudioStatus*. Then, Application is able to analyze receiver capabilities (*tmdlHdmiTxGetEdidSinkType*, *tmdlHdmiTxGetEdidAudioCaps, and tmdlHdmiTxGetEdidVideoCaps*) and to configure HDMI Tx input and output in order to start streaming the audio/video content (*tmdlHdmiTxSetInputOuput*). If the EDID can not be read, it is up to the Application to choose a video output mode that should supported by the sink (this is dependent of the targeted consumer electronic application).

If this content shall be encrypted a call to *tmdlHdmiTxSetHdcp* is required.

3.6 Change audio and/or video format

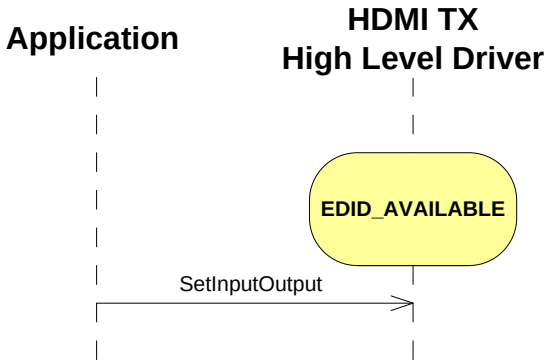


Figure 8 Change audio/Video format sequence diagram

In order to change audio and/or video format, application has to call API *tmdlHdmiTxSetInputOuput.*

3.7 Change audio format only

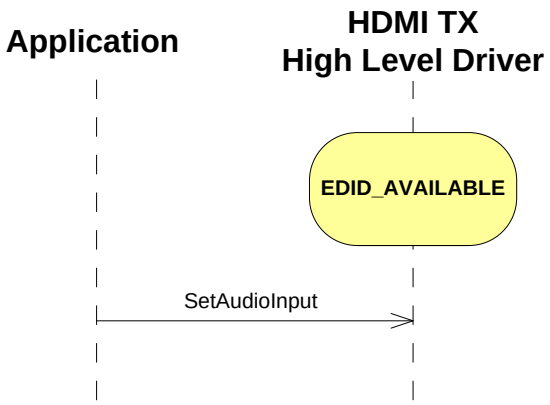


Figure 9 Change Audio format only sequence diagram

In order to change audio format, application has to call API *tmdlHdmiTxSetAudioInput.*

3.8 Standby usage

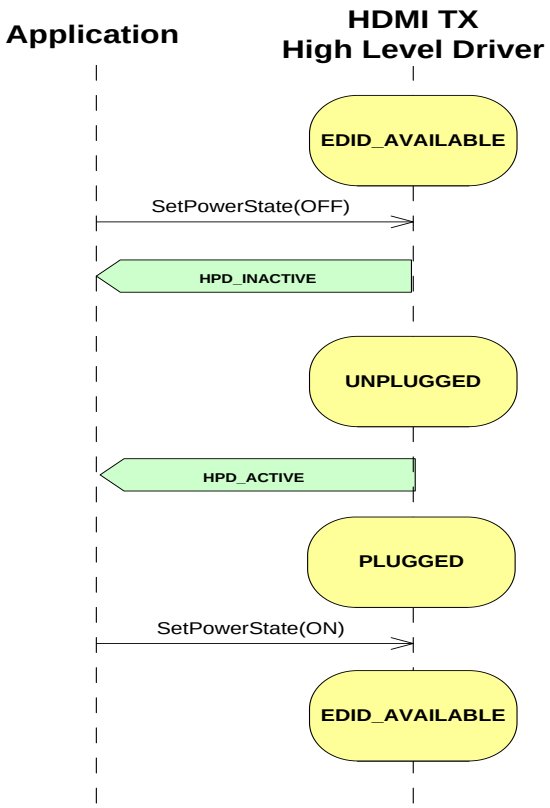


Figure 10 Standby OFF-ON sequence diagram

Application can switch the HDMI Tx in power OFF mode with a call to *tmdlHdmiTxSetPowerState*. In this mode HPD events are still monitored, therefore when the power is switched ON again an EDID read will be internally triggered when required.

In order to save power some Applications may want to cut 5V when HDMI is not used. This operation has to be done carefully regarding HDMI driver state. Basically, here is what has to be checked :

- 5V shall be there before doing *tmdlHdmiTxOpen*
- After setting the IC in Standby it is possible to cut 5V
- Before doing *tmdlHdmiTxSetPowerState(ON)*, 5V shall be present.

Those principles are depicted in picture below:

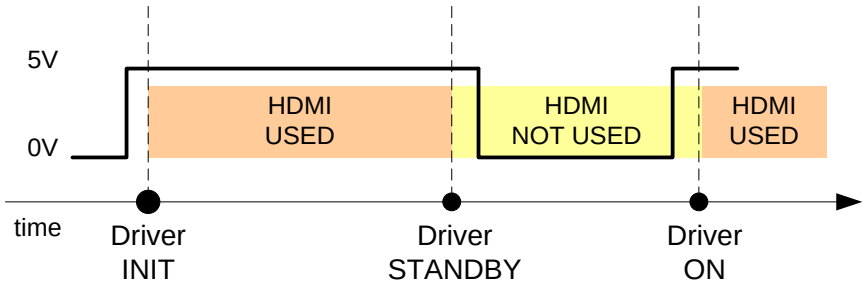


Figure 11 HDMI 5V manipulation regarding driver usage

3.9 Stop HDMI streaming

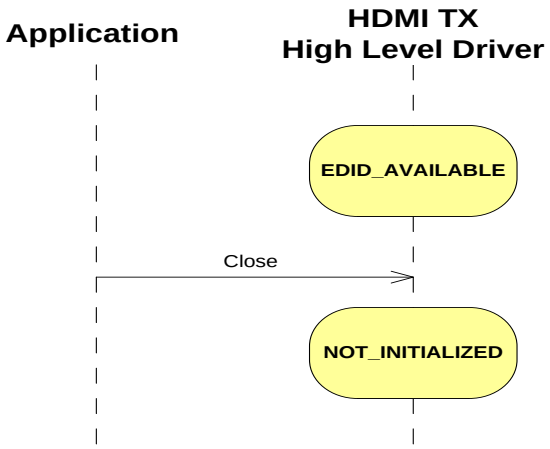
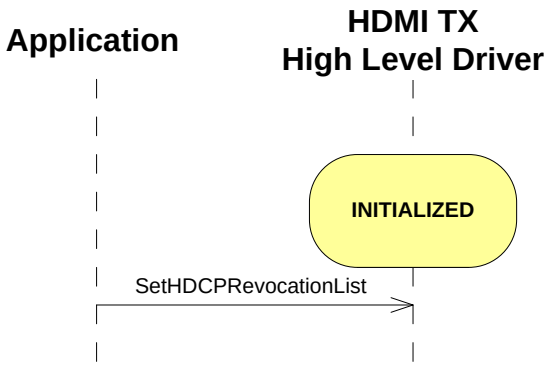


Figure 12 Stop HDMI streaming

In order to stop the streaming Application has to call the API *tmdlHdmiTxClose.*

3.10 Set HDCP revocation list



In order to set a list of sinks to revoke, the application must call the API *tmdlHdmiTxSetHDCPRevocationList*.

3.11 Software configuration

The application specifies per tmdlHdmiTx instance HDMI TX parameters by setting values in file *tmdlHdmiTx_cfg.c*:

COMMAND_TASK_PRIORITY_0	This parameter set the priority of the command task that performs the deported processing of the ISRs.
COMMAND_TASK_STACKSIZE_0	This parameter set the stack size of the command task.[1]
COMMAND_TASK_QUEUESIZE_0	This parameter set the size of the queue which is used to deport ISRs commands.[2]
HDCP_CHECK_TASK_PRIORITY_0	This parameter set the priority of the hdcp task. This task is used to trigger low level driver third part of HDCP authentication protocol, where verification is made at a minimum rate of once every two seconds to insure that the video receiver is still able to correctly decrypt the information. It is recommended to set the same priority has the one set to the command task.
HDCP_CHECK_TASK_STACKSIZE_0	This parameter set the stack size of the hdcp task.
UNIT_I2C_ADDRESS_0	Define the I^2C address of the HDMI Tx IC.
TxI2cReadFunction	Function pointer of the infrastructure dependent I^2C read function.
TxI2cWriteFunction	Function pointer of the infrastructure dependent I^2C write function.
KEY_SEED	16 bits seed for keys decryption during the loading into the HDCP memory. Contact your NXP field application engineer to get this key.
TMDL_HDMITX_PATTERN_BLUE	Used to manage video output when the Sink is not HDCP-capable. Refer to [SIMPLAY_HD]
dataEnableSignalAvailable	When using external synchronization mode : set this parameter to 1 if DE signal is available, 0 otherwise.

By respectively filling videoPortMapping_YUV444, videoPortMapping_RGB444, videoPortMapping_YUV422, videoPortMapping_CCIR656 arrays the Application defines how the video input signals are connected to the HDMI TX IC.

For instance, let's assume that your video input signal is coming as such :

[1] This value is OS dependent.

[2] The value provided with sample configuration file can be used.

Video Input	RGB444	YUV444	YUV422sp	CCIR656
VPA[7..0]	B	U	-	CbYCrCb
VPB[7..0]	G	Y	Y	-
VPC[7..0]	R	V	CbCr	-

you have then to set your configuration file like this:

```
const tmdlHdmiTxCfgVideoSignal444 videoPortMapping_YUV444[MAX_UNITS][6] = {
    {
        TMDL_HDMITX_VID444_BU_0_TO_3,   /* Signals connected to VPA[0..3] */
        TMDL_HDMITX_VID444_BU_4_TO_7,   /* Signals connected to VPA[4..7] */
        TMDL_HDMITX_VID444_GY_0_TO_3,   /* Signals connected to VPB[0..3] */
        TMDL_HDMITX_VID444_GY_4_TO_7,   /* Signals connected to VPB[4..7] */
        TMDL_HDMITX_VID444_VR_0_TO_3,   /* Signals connected to VPC[0..3] */
        TMDL_HDMITX_VID444_VR_4_TO_7    /* Signals connected to VPC[4..7] */
    }
};

const tmdlHdmiTxCfgVideoSignal444 videoPortMapping_RGB444[MAX_UNITS][6] = {
    {
        TMDL_HDMITX_VID444_BU_0_TO_3,   /* Signals connected to VPA[0..3] */
        TMDL_HDMITX_VID444_BU_4_TO_7,   /* Signals connected to VPA[4..7] */
        TMDL_HDMITX_VID444_GY_0_TO_3,   /* Signals connected to VPB[0..3] */
        TMDL_HDMITX_VID444_GY_4_TO_7,   /* Signals connected to VPB[4..7] */
        TMDL_HDMITX_VID444_VR_0_TO_3,   /* Signals connected to VPC[0..3] */
        TMDL_HDMITX_VID444_VR_4_TO_7    /* Signals connected to VPC[4..7] */
    }
};

const tmdlHdmiTxCfgVideoSignal422 videoPortMapping_YUV422[MAX_UNITS][6] = {
    {
        TMDL_HDMITX_VID422_NOT_CONNECTED,   /* Signals connected to VPA[0..3] */
        TMDL_HDMITX_VID422_NOT_CONNECTED,   /* Signals connected to VPA[4..7] */
        TMDL_HDMITX_VID422_Y_4_TO_7,        /* Signals connected to VPB[0..3] */
        TMDL_HDMITX_VID422_Y_8_TO_11,       /* Signals connected to VPB[4..7] */
        TMDL_HDMITX_VID422_UV_4_TO_7,       /* Signals connected to VPC[0..3] */
        TMDL_HDMITX_VID422_UV_8_TO_11       /* Signals connected to VPC[4..7] */
    }
};

const tmdlHdmiTxCfgVideoSignalCCIR656 videoPortMapping_CCIR656[MAX_UNITS][6] = {
    {
        TMDL_HDMITX_VIDCCIR_4_TO_7,         /* Signals connected to VPA[0..3] */
        TMDL_HDMITX_VIDCCIR_8_TO_11,        /* Signals connected to VPA[4..7] */
        TMDL_HDMITX_VIDCCIR_NOT_CONNECTED,  /* Signals connected to VPB[0..3] */
        TMDL_HDMITX_VIDCCIR_NOT_CONNECTED,  /* Signals connected to VPB[4..7] */
        TMDL_HDMITX_VIDCCIR_NOT_CONNECTED,  /* Signals connected to VPC[0..3] */
        TMDL_HDMITX_VIDCCIR_NOT_CONNECTED   /* Signals connected to VPC[4..7] */
    }
};
```

For YCbCr 422 semi-planar and YCbCr 422 compliant with ITU656 video inputs the HDMI TX IC can handle 12 bits, therefore if only 8 bits are coming in we only map the MSB bits.

By respectively filling enableAudioPort**xxx**, enableAudioClockPort**xxx** the Application defines how the audio input signals are connected to the HDMI TX IC. Variables groundAudioPort**xxx**, groundAudioClockPort**xxx** are provided to connect to ground unused ports. For instance, the PCB layout below entails the following setting for I2S and SPDIF:

```
UInt8 enableAudioPortSPDIF[MAX_UNITS]      = {0x40};
UInt8 enableAudioClockPortSPDIF[MAX_UNITS] = {DISABLE_AUDIO_CLOCK_PORT};
UInt8 groundAudioPortSPDIF[MAX_UNITS]      = {0xbf};
UInt8 groundAudioClockPortSPDIF[MAX_UNITS] = {ENABLE_AUDIO_CLOCK_PORT_PULLDOWN};

UInt8 enableAudioPortI2S[MAX_UNITS]        = {0x03};
UInt8 enableAudioPortI2S8C[MAX_UNITS]      = {0x1f};
UInt8 enableAudioClockPortI2S[MAX_UNITS]   = {ENABLE_AUDIO_CLOCK_PORT};
UInt8 groundAudioPortI2S[MAX_UNITS]        = {0xfc};
UInt8 groundAudioPortI2S8C[MAX_UNITS]      = {0xe0};
UInt8 groundAudioClockPortI2S[MAX_UNITS]   = {DISABLE_AUDIO_CLOCK_PORT_PULLDOWN};
```

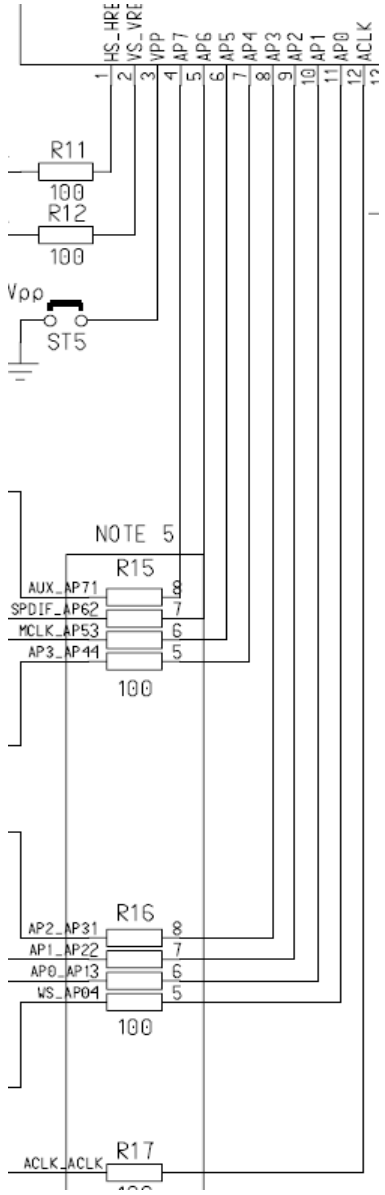


Figure 13: PCB Audio configuration example

During its initialization, the tmdlHdmiTx driver retrieves the values set by the application.

3.12 Hardware diversity

TDA9981, TDA9983, TDA9984, TDA9989 require a video pixel clock always coming in when the IC is not in power off.
This is not the case for TDA19989/88.

3.13 Software diversity

HDMI TX driver uses the following compilation flags:

Diversity flag	Diversity usage	Comments
FORMAT_PC	When this compilation flag is set video PC formats are supported and video TV formats are supported. When not set only video TV formats are supported.	
TMFL_TDA9981_SUPPORT	Use this compilation flag in order to produce tmbslTDA9983 source code for TDA9981 IC support.	TDA9981 only
TMFL_RX_SENSE_ON	When this compilation flag is set the Rx sense interrupt is managed by the driver.	TDA9981 only
TMFL_HBR_SUPPORT	Define this flag in order to support HBR audio on TDA9989.	TDA9989 only
TMFL_CEC_AVAILABLE	Define this flag in order to support CEC protocol on TDA9989.	TDA9989 only
NO_HDCP	Define this flag is HDCP is not required by your application.	
TMFL_NO_RTOS	Define this flag in order to use the HDMI TX drivers in NO OS mode.	TDA9984, TDA9989 only
TMFL_LINUX_OS_KERNEL_DRIVER	Define this compilation flag to build the drivers as Linux Kernel driver.	
SPDIF_ACLK_TO_CLOCK	Define this compilation flag, if on hardware the audio generator can provide to the TDA9984 an I2S clock in the same time as the SPDIF (see application note “Guidelines to implement TDA9984A ” chapter “SPDIF stability improvement”).	TDA9984 only
TMFL_HDCP_SUPPORT	Define this compilation flag to use HDCP feature.	TDA19989 only
TMFL_TDA19989	Define this compilation flag when using TDA19989 IC.	TDA19989 only
TMFL_TDA19988	Define this compilation flag when using TDA19988 IC.	TDA19988 only

Moreover some compilation keywords have been defined in order to handle several compiler targets. These keywords are : **`FUNC_PTR , CONST_DAT`** `and` **`RAM_DAT`**

They need to be set in order to compile in your environment. For instance under ARM7 we use the following settings : `FUNC_PTR=" " CONST_DAT="const " RAM_DAT=" "`

3.13.1 TDA19988 specificities

- **TDA19988 interoperability**

Same API and source than TDA19989.

- **3D frame packing**

The TDA19988 supports up to 720p@24/25/30/50/60 fps and 1080p@24/25/30 in frame packing.

- **Optimized power**

Power consumption has been optimized by frozing useless clocks related to HDCP (SPDIF, downsampler and color space conversion clocks). This feature is enabled by the **TMFL_TDA19988** flag. See flag **TMFL_HDCP_OPTIMIZED_POWER** in the soure code for more details about power management.

3.13.2 TDA19989 / TDA9989 specificities

- **Power management**

This HDMI TX IC supports the following power modes: (API *tmdlHdmiTxsetPowerState*):

`tmPowerOn`	The chip is fully active
`tmPowerSuspend`	In this mode the IC is not able to output HDMI stream but is still able to report HDMI connection status via HPD_ACTIVE / HPD_INACTIVE and RX_DEVICE_ACTIVE / RX_DEVICE_INACTIVE events.
`tmPowerStandby`	Output activity detection is disabled
`tmPowerOff`	**NOT SUPPORTED**

TDA9989 **shall be powered ON** before being able to output any Audio/Video stream.

- **TDA9989 interoperability**

It is possible to increase TDA9989 level of interoperability (EDID reading issues with some LCD TVs) by defining the following compilation flag : **TMFL_TDA9989_PIXEL_CLOCK_ON_DDC**.

However using this flag introduces some system limitations :

- In order to perform properly the EDID reading, the system shall always provide a stable pixel clock coming in.

- This pixel clock shall not change in between HPD_ACTIVE and EDID_READ event.

In other terms the application is no more free to change video coming in whenever it wants but shall take care of HDMI driver state.

3.13.3 TDA9984 specificities

When the IC is put in STANDBY mode, there is no Rx sense management. In order to implement properly some power saving in HDMI TX user application, once need to do the following :

- upon TMDL_HDMITX_HPD_ACTIVE do tmdlHdmiTxSetPowerState(tmPowerOn)
- upon TMDL_HDMITX_HPD_INACTIVE do tmdlHdmiTxSetPowerState(tmPowerStandby)
- upon TMDL_HDMITX_RX_DEVICE_INACTIVE dotmdlHdmiTxSetPowerState(tmPowerStandby)
- upon TMDL_HDMITX_RX_DEVICE_ACTIVE do nothing.

3.13.4 TDA9981/83 specificities

- **TDA9983 only**: HPD interrupt is not always raised, consequently the interrupt mode can not be used with this IC.
- EDID reading

The EDID reading I²C accesses have to be done by an external microprocessor through the HDMI TX IC's I²C-bus gate. **For that purpose, it is up to the application to implement an external function** (let's call it EdidBlockRead).

EdidBlockRead is registered via a pointer to function in file *tmdlHdmiTx_cfg.c* and will be called by the driver during its EDID reading phase. The expected pointer to function prototype is the following:

```
typedef struct _tmbslHdmiTxSysArgsEdid_t
{
    UInt8 segPtrAddr;       /* 60h (8bits) */
    UInt8 segPtr;           /* 0 or 1 for EDID 4 blocks reading */
    UInt8 dataRegAddr;      /* A0h (8bits) */
    UInt8 wordOffset;       /* 00h , 80h for second block in a given segment */
    UInt8 lenData;          /* length of data to read : 80h (128 bytes) */
    UInt8 *pData;           /* buffer to receive lenData bytes */
} tmbslHdmiTxSysArgsEdid_t;
typedef tmErrorCode_t (FUNC_PTR * ptmbslHdmiTxSysFuncEdid_t)
                                (tmbslHdmiTxSysArgsEdid_t *pSysArgs);
```

In order to ease parameters comprehension, picture below depicts the layout of the E-DDC structure:

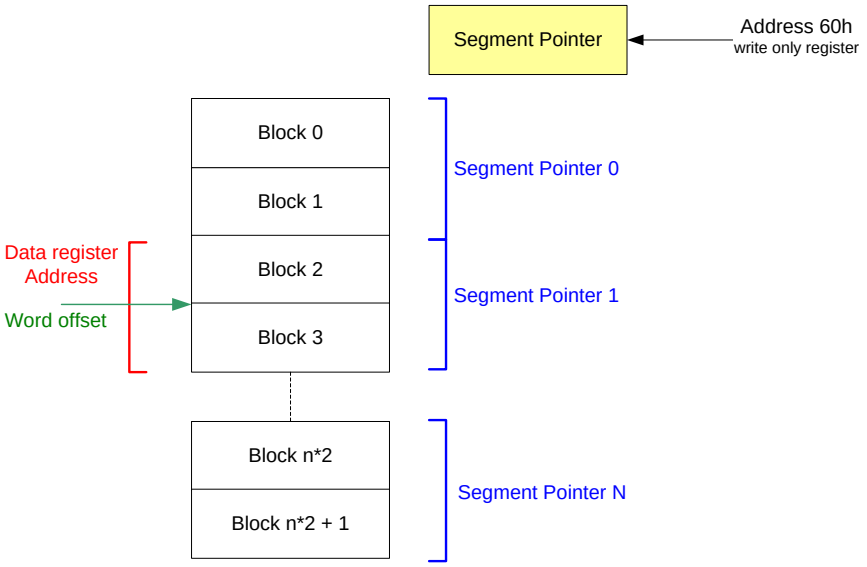


Figure 14 E-DDC segment pointer an block layout

Basically in order to read an EDID block the I²C sequence looks like this:

S	segPtr Addr	**A**	seg Ptr	**A**	**RS**	dataReg Addr	**A**	word Offset	**A**	**RS**	dataReg Addr+1	**A**	pData	**A / NA**	**P**

(S = Start; A = Acknowledge; RS = Repeated Start; pData is read lenData times; NA = No Acknowledge on last byte; P = Stop)

A four block EDID reading I²C sequence performed by the driver will look like this:

```
1. EdidBlockRead(0x00, 0x00, 0xA0, 0x00, 0x80, pData)
2. EdidBlockRead(0x00, 0x00, 0xA0, 0x80, 0x80, pData)
3. EdidBlockRead(0x60, 0x01, 0xA0, 0x00, 0x80, pData)
4. EdidBlockRead(0x60, 0x01, 0xA0, 0x80, 0x80, pData)
```

On the I²C bus one can see the following I2C transactions:

	tus	Address	Dir	Length	Data
0	OK LS S	8 0xA0	TX	1	00
	OK LS SP		RX	128	00 FF 47 98 51 D0 00 00 3D 08
1	OK LS S	8 0xA0	TX	1	80
	OK LS SP		RX	128	F0 02 00 00 00 00 00 00 00 00
2	OK LS S	8 0x60	TX	1	01
	OK LS S	8 0xA0	TX	1	00
	OK LS SP		RX	128	02 03 00 00 40 55 80 D0 8E 21
3	OK LS S	8 0x60	TX	1	01
	OK LS S	8 0xA0	TX	1	80
	OK LS SP		RX	128	02 03 20 0C

Figure 15 Four blocks EDID reading I²C sequence

Note1: the segment pointer positioning for Segment Pointer 0 is optional.

Note2: there is only one stop bit at the end of one EDID block read transaction.

3.14 Zip delivery file structure

3.14.1 High level driver

- **The cfg** directory of the high level driver contains
  - ***tmdlHdmiTx_cfg.h*** a header file that defines the configurable items
  - ***tmdlHdmiTx_cfg.c*** configuration example file.
    ⇨ This file shall be customized by the client application.
  - ***tmdlHdmiTx_IW.h*** a header file that defines the required OS interface
  - ***tmdlHdmiTx_IW.c*** an OS implementation example file.
    ⇨ This file shall be customized by the client application.

3.14.2 Low level driver

The tmbslHdmiTx driver is physically organized in two parts; one which defines a generic API whatever the underlying HDMI Tx hardware, and the other which defines a given HDMI Tx IC driver implementation.

3.14.3 Shared include files

Some shared headers that contain types definitions files are located under directory HdmiTx\sde2\inc.

3.14.4 Example Application

An HDMI Tx application, used in polling mode, is provided as example for a LPC2148 microcontroller with RTL-RTX as OS. The source files of the example application are located in the following directory:

Example Application
HdmiTx/sde2/comps/tmdlHdmiTx/tst/tmdlHdmiTx_ExampleApplication/src/***tmdlHdmiTx_ExampleApplicationArm7.c***

This application takes a 720x480p60Hz YUV4:2:2sp video as input. Then, depending on display sink capabilities (retrieved from EDID reading) and on HDMI TX IC possibilities it outputs the first video format that is available in those two lists.

This application can be easily customized with the following variables:

Variable name	**Description**
gVideoInConfig	Video input configuration
gVideoOutConfig	Video output configuration
gAudioInConfig	Audio input configuration
gHDCPMode	Enable/Disable the HDCP mode
gSinkType	Specifies the type of sink device
gDlHdmiTxSetupInfo	Specifies setup informations such as EDID buffer size & address, repeater feature, simplay HD feature.

4. Document Management

4.1 Document History

Version	Date	Author	Observations
0.1	18 October 2007	Cyril Bes	Document creation.
0.2	07 November 2007	Cyril Bes	Rework after first review
0.3	15 November 2007	Cyril Bes	Remove multi–instance chapter
0.5	04 December 2007	Cyril Bes	Update document to reflect API changes.
0.6	27 February 2008	Cyril Bes	Update document to reflect API changes and configuration file changes.
0.7	05 March 2008	Cyril Bes	Update HDMI output sequence diagram
0.8	17 March 2008	S. Desramé	Update related to the TDA9983
0.9	21 March 2008	S. Desramé	Update after review
0.10	28 March 2008	Cyril Bes	Add LIPP4200 specificities
0.11	28 March 2008	Cyril Bes	Add TDA9984 specificities
0.12	23 April 2008	Cyril Bes	Refine Video/Audio input ports mapping explanation.
0.13	19 May 2008	Cyril Bes	Make references to deliverd example application.
0.14	22 May 2008	S. Desramé	Add usage of the revocation list
0.15	22 May 2008	S. Desramé	Add constraint regarding the location of the KSV revocation list
0.16	23 May 2008	Cyril Bes	Document all compilation switches. Remove LIPP4200 specificities
0.17	12 June 2008	Cyril Bes	Explain TDA9981/TDA9983 Edid reading specificities.
0.18	13 June 2008	S. Desramé	Update related to PR 1522: remove API tmdlHdmiTxReset
0.19	19 June 2008	Cyril Bes	Remove API tmdlHdmiTxSetAClkRecoveryPacket Add description of new API tmdlHdmiTxGetHdcpFailStatus
0.20	04 July 2008	Cyril Bes	Update tmdlHdmiTxSetHDCPRevocationList behavior and returns code.
0.21	22 Sept 2008	Cyril Bes	Update NO OS documentation. Update TDA9989 SetPowerState usage.
0.22	17 Nov 2008	Cyril Bes	Update TDA9983 specificities. TDA9983 can not be used in interrupt mode (HW issue: HPD interrupt not always generated)
0.23	08 Jan 2009	Cyril Bes	Update tmdlHdmiTxGetEdidVideoCaps VideoFlags description. Add tmdlHdmiTxGetEdidLatencyInfo description.
0.25	28 July 2009	Cyril Bes	Document TDA19989 required compilation flags.
0.26	10 Sept 2009	Cyril Bes	Document TDA9989 sw workaround for EDID Panasonic TH-17LX8 reading issue.
0.27	12 Oct 2009	Cyril Bes	Document new APIs. tmdlHdmiTxGetHPDStatus tmdlHdmiTxGetRXSenseStatus

0.28	28 Oct 2009	Cyril Bes	Explain in which conditions 5V can be cut.
0.29	04 Jan 2009	Cyril Bes	Explain dataEnableSignalAvailable parameter in cfg file.
0.30	05 Mar 2010	Wenzhi Guo	Describe the new API *tmdlHdmiTxSetExtendedColorimetry*
0.31	29 Nov 2010	André Lépine	3D management
0.32	10 May 2011	Vincent Vrignaud	Linux kernel compilation. Minor updates.

4.2 Document References

[TDA9983] Objective Datasheet of TDA9983 rev 01 27 october 2006

[TDA19984] Objective Datasheet of TDA9984 rev. 01 23 july 2007

[TDA19988] Objective Datasheet of TDA9988 rev. 02 28 april 2011

[TDA19989] Objective Datasheet of TDA9989 rev. 01 16 october 2007

[HDMI_SPEC] High-Definition Multimedia Interface Specification Version 1.4a

[CEA-861-D] A Digital Television Profile for Uncompressed High Speed Digital Interfaces

[SIMPLAY_HD] Simplay HD Specification v1.2

[HDCP] High-bandwidth digital content protection system v1.2

4.3 Abbreviations and terminology

CEC	The Consumer Electronics Control protocol provides high-level control functions (such as 'one touch play',) between all of the various audiovisual products in a user's environment.
CTS	Audio data being carried across the HDMI link, which is driven by a TMDS clock running at a rate corresponding to the video pixel rate, does not retain the original audio sample clock. The Cycle Time Stamp (CTS) value is used by the Sink device to recreate this clock.
Data Island	The HDMI link operates in one of three modes: Video Data Period, Data Island period, and Control period. During the Video Data Period, the active pixels of an active video line are transmitted. During the Data Island period, audio and auxiliary data are transmitted using a series of packets. The Control period is used when no video, audio, or auxiliary data needs to be transmitted. A Control Period is required between any two periods that are not Control Periods.
DDC	The Display Data Channel is a digital connection between a monitor and a graphics adapter that allows the display to communicate its specifications to the adapter. The monitor contains a read-only memory chip programmed by the manufacturer with information about the graphics modes that the monitor can display. The data in the monitor's ROM is held in a standard format called extended display identification data (EDID). Moreover, this communication bus is also used for HDCP exchanges.
DST	An audio format which is a lossless compression of Direct Stream Digital (DSD), as used in SuperAudio CD. DST is described in ISO/IEC 14496, part 3, Amendment 6: Lossless coding of oversampled audio.
DVI	The Digital Visual Interface (DVI) is a video interface standard designed to maximize the visual quality of digital display devices such as flat panel LCD computer displays and digital projectors. It is designed for carrying uncompressed digital video data to a display.
EDID	Extended Display Identification Data is a data structure provided by a display to describe its capabilities to a graphics adapter.
HBR	High Bitrate (HBR) Audio Stream Packet (IEC 61937)
HDCP	High-bandwidth Digital Content Protection (HDCP) is a form of Digital Rights Management (DRM) developed by Intel Corporation to control digital audio and video content as it travels across Digital Visual Interface (DVI) or High-Definition Multimedia Interface (HDMI) connections. The specification is proprietary, and creating an implementation of HDCP requires a license.
HDMI	The High-Definition Multimedia Interface (HDMI) is a licensable audio/video connector interface for transmitting uncompressed, encrypted digital streams. HDMI specification defines the protocol and electrical specifications for the signaling, as well as the pin-out, electrical and mechanical requirements of the cable and connectors.
HPD	Hot Plug Detect.
InfoFrame	A data structure defined in [CEA-861-D] that is designed to carry a variety of auxiliary data items regarding the audio or video streams or the source device and is carried from Source to Sink across HDMI.

ISR	An Interrupt Service Routine (ISR), is a callback subroutine in an operating system or device driver whose execution is triggered by the reception of an interrupt.
ISRC	The International Standard Recording Code (ISRC), defined by ISO 3901, is an international standard code for uniquely identifying sound recordings and music video recordings.
KSV	Each HDCP Device contains a set of Device Private Keys. A set of Device Private Keys is associated with a *Key Selection Vector* (*KSV*). Each HDCP Transmitter has assigned to it a unique *KSV* from all other HDCP Transmitters. Also, each HDCP Receiver has assigned to it a unique *KSV* from all other HDCP Receivers.
OBA	1-bit Delta-Sigma modulated signal stream such as that used by Super Audio CD.
Physical Address	In order to allow CEC to be able to address specific physical devices and control switches, all devices shall have a physical address. The physical address discovery process uses only the DDC/EDID mechanism and applies to all HDMI Sinks and Repeaters, not only to CEC-capable devices. A Source or a Repeater reads its physical address from the EDID of the connected Sink.
Preferred Video Format	The video format that a display manufacturer determines provides optimum image.
Repeater	A device with one or more HDMI inputs and one or more HDMI outputs. Repeater devices shall simultaneously behave as both an HDMI Sink and an HDMI Source.
TMDS	Transition Minimized Differential Signaling (TMDS) is a technology for transmitting high-speed serial data and is used by the DVI and HDMI video interfaces. The transmitter incorporates an advanced coding algorithm which has reduced electromagnetic interference over copper cables and enables robust clock recovery at the receiver to achieve high skew tolerance for driving longer cable lengths as well as shorter low cost cables.
Video Format	A video format is sufficiently defined such that when it is received at the monitor, the monitor has enough information to properly display the video to the user. The definition of each format includes a Video Format Timing, the picture aspect ratio, and a colorimetry space.