Application Note DT-AN-2195-1

HDR and Dolby Atmos® Support for DTA-2195

1. Introduction

The DTA-2195 *12G-SDI input and output with HDMI 2.0 output* is a PCI-Express adapter for interfacing 4K-UHD signals with a PC. Starting from the November 2017 release of the DekTec SDK (DTAPI), the DTA-2195 can generate a 4K HDMI stream that includes HDR signalling and Dolby Atmos® audio. This application note explains how to configure DekTec Matrix API® 2.0 to generate such streams.

Basic- and reference-information about DekTec Matrix API® 2.0 is available on the DekTec website at www.dektec.com/products/SDK/DTAPI/MatrixApi/. This application note contains a number of code snippets; the complete example is part of the Matrix-API example code available from the MatrxApi page on the DekTec webpage.

2. References

CTA-861 ANSI/CTA-861-G, A DTV Profile for Uncompressed High Speed Digital Interfaces, November 2016, Consumer Technology Association.

ST 372 SMPTE ST 372:2017, Dual Link 1.5 Gb/s Digital Interface for 1920 × 1080 and 2048 × 1080 Picture Formats.

ST 2084 SMPTE ST 2084:2014, High Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays.

BT.2100 ITU-R BT.2100, Image parameter values for high dynamic range television for use in production and international programme exchanges.

3. HDR – High Dynamic Range

3.1. HDMI InfoFrames

HDR information is transferred to an HDMI-capable display using the Dynamic Range and Mastering InfoFrame (CTA-861 section 6.9). If the HDMI display attached to the DTA-2195 supports HDR, the DTA-2195 will send this InfoFrame automatically.

The user application supplies the data for the Dynamic Range and Mastering InfoFrame in the SDI frame, in the form of ancillary data packets in the luminance stream of the VANC.

The Colorimetry and Extended Colorimetry values used in the Auxiliary Video Information InfoFrame (CTA-861 section 6.4) can be set by the user application with a DTAPI function call.

3.2. Ancillary Data Packets

3.2.1. Ancillary Data Packet Type

Two types of ancillary data packets are used for transferring HDR information in an SDI frame:

1. Payload Identification (DID 0x41, SDID 0x01, SMPTE ST 372), to indicate the transfer characteristics. These are used in the Dynamic Range and Mastering InfoFrame as the EOTF (Electro-Optical Transfer Function).

   The transfer characteristics are defined in the Payload Identification as follows:

Byte	Bits	Value	Characteristics	Standard
2	5..4	0x00	SDR-TV	SMPTE ST 274
		0x01	HLG HDR	ITU-R BT.2100
		0x02	PQ HDR	SMPTE ST 2084

2. Extended HDR/WCG Metadata (DID 0x41, SDID 0x0c, SMPTE ST 2108:201x) to indicate the HDR metadata. The payload of the ancillary data packet is defined in the SMPTE ST 2108. At the time of this writing, the SMPTE ST 2108 specification is not yet released.

   The payload of the ancillary data packet is as follows:

Byte	Byte Description
0	Metadata frame type
1	Metadata frame length (n)
2	Metadata data byte 1
…	…
n+1	Metadata data byte n

   Only static metadata is currently supported by the DTA-2195.

   Please refer to SMPTE ST 2108 and CTA-861 for the Metadata Data Byte content details.

3.2.2. Using Ancillary Data in the Matrix API® 2.0

Apart from general configuration (see Matrix-API documentation), the following specific steps must be executed for creating ancillary data with the Matrix API:

1. Enable and configure AUX data.

```
DtMxRowConfig  RowConfig;

RowConfig.m_AuxDataEnable = true;
RowConfig.m_AuxData.m_DataType = DT_AUXDATA_SDI;
RowConfig.m_AuxData.m_Sdi.m_AncPackets.m_OutputMode = DT_OUTPUT_MODE_ADD;
```

2. Disable VPID Processing. If VPID processing would be enabled, the DTA-2195 will overwrite the VPID ancillary data packets delivered by the user application.

```
RowConfig.m_AuxData.m_Sdi.m_DisableVpidProcessing = true;
```

3. Register a callback function that is called when ancillary data must be generated for a new frame.

```
DtMxProcess  m_TheMatrix;

m_TheMatrix.AddMatrixCbFunc(OnNewFrameGenAux, &Context);
```

4. Implement the callback function.

```
void OnNewFrameGenAux(DtMxData* pData, void* pContext)
{
  DtMxRowData&  OurRow = pData->m_Rows[0];

  // Generate the ancillary VPID data according to SMPTE ST 372-2017
  void*  pVpId = GenerateVpidAncData();

  // Insert the ancillary data into the luminance stream of the vanc
  OurRow.m_CurFrame->AncAddPacket(pVpId, DTAPI_SDI_VANC, DTAPI_SDI_LUM, 0);

  // Generate the ancillary HDR data according to SMPTE ST 2108
  void*  pHdrAncData = GenerateHdrAncData();

  // Insert the ancillary data into the luminance stream of the vanc
  OurRow.m_CurFrame->AncAddPacket(pHdrAncData, DTAPI_SDI_VANC, DTAPI_SDI_LUM, 0);
}
```

3.2.3. Matrix API® 2.0 callback example code

The following code snipped is an example of a callback function used to deliver ancillary data packets (HDR metadata and VPID data) to the Matrix API. The `DtMxHdrVpid` and `DtMxHdrMetaData` types are helper classes used in the example code to build the ancillary data packets.

The HDR metadata (`cFrameData`) in this example is fake data and the VPID data (`cFrameDataVpid`) is not complete.

See the source of `DtMxHdmiHdrDemo` for a complete example.

```
void  MxHdmiHdrDemo::OnGenerateAnc(DtMxData* pData)
{
  DTAPI_RESULT  dr;

  DtMxRowData&  OurRow = pData->m_Rows[OUT_ROW];

  // Must have a valid frame
  if (OurRow.m_CurFrame->m_Status != DT_FRMSTATUS_OK)
     return;    // Frame is not valid the frame buffers are unusable

  // Refer to SMPTE ST-452-1:201x, 4.1.6 "Payload Identifier"
  // Refer to SMPTE ST-372:201x, see section 7 "Payload Identification"
  // Transfer characteristics (bits 5 - 4 of Byte 2) is set to HLG (1h)
  // Other values set do not conform to any spec!
  unsigned char  cFrameDataVpid[] = { 0xAA, 0xD0, 0x00, 0x00 };

  std::vector<unsigned char>  FrameDataVpid(cFrameDataVpid,
               &cFrameDataVpid[sizeof(cFrameDataVpid)/sizeof(cFrameDataVpid[0])]);

  // Create a HDR VPID packet
  DtMxHdrVpid  Vpid;
  Vpid.SetVpid(FrameDataVpid);

  // Add VPID packet to frame. Insert in luminance stream of the VANC
  dr = OurRow.m_CurFrame->AncAddPacket(Vpid, DTAPI_SDI_VANC, DTAPI_SDI_LUM, 0);
  MX_ASSERT(dr == DTAPI_OK);

  // Create frame data for static HDR metadata frame
  const unsigned char  cFrameData[] =
  {
     0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
     0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10,
     0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18,
  };
  std::vector<unsigned char>  FrameData(cFrameData,
                              cFrameData[sizeof(cFrameData)/sizeof(cFrameData[0])]);

  // Create a HDR Metadata packet
  DtMxHdrMetadata  Hdr;
  Hdr.SetMetadata(DtMxHdrMetadata::FRAME_TYPE_STATIC_TYPE1, FrameData);

  // Add to frame. Insert in luminance stream of the VANC
  dr = OurRow.m_CurFrame->AncAddPacket(Hdr, DTAPI_SDI_VANC, _SDI_LUM, 0);
  MX_ASSERT(dr == DTAPI_OK);
}
```

3.3. Colorimetry and Extended Colorimetry values

The colorimetry (C0 and C1) and extended colorimetry (EC0, EC1 and EC2) as described in the CTA 861 specification for the Auxiliary Video Information (AVI) InfoFrame can be set by the user application. If not set, the attached display will use the default colorimetry as defined in the CTA-861 specification.

To set these values, use the `SetHdmiColorimetry` function from the `DtAvOutput` class. The class must be attached to the HDMI output port (port 4) of the DTA-2195 before using this function.

```
DTAPI_RESULT  DtAvOutput::SetHdmiColorimetry(
  [in] int  Colorimetry,
  [in] int  ExtendedColorimetry
);
```

The table below describes the parameter values:

Parameter	Bit position	CEA-861 naming
Colorimetry	0	C0
	1	C1
ExtendedColorimetry	0	EC0
	1	EC1
	2	EC2

See the CTA-861 specification for the definition of these bits.

Example of setting the colorimetry to ITU-R BT.2020 R'G'B' or Y'C'BC'R:

```
DtAvOutput  Dta2195HdmiPort;
Dta2195HdmiPort.AttachToPort(&The2195Card, 4);
Dta2195HdmiPort.SetHdmiColorimetry(3, 6);
```

4. Dolby Atmos®

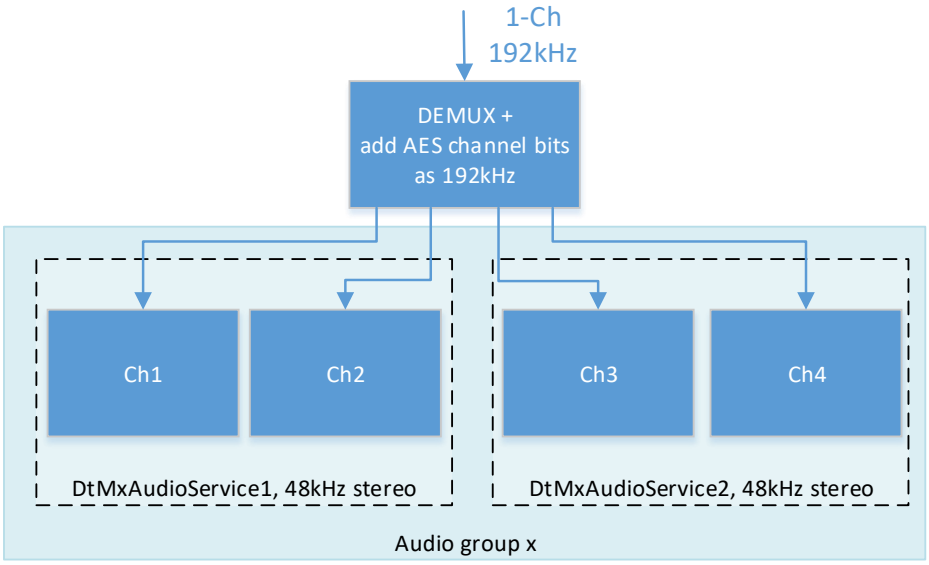
4.1. Basics

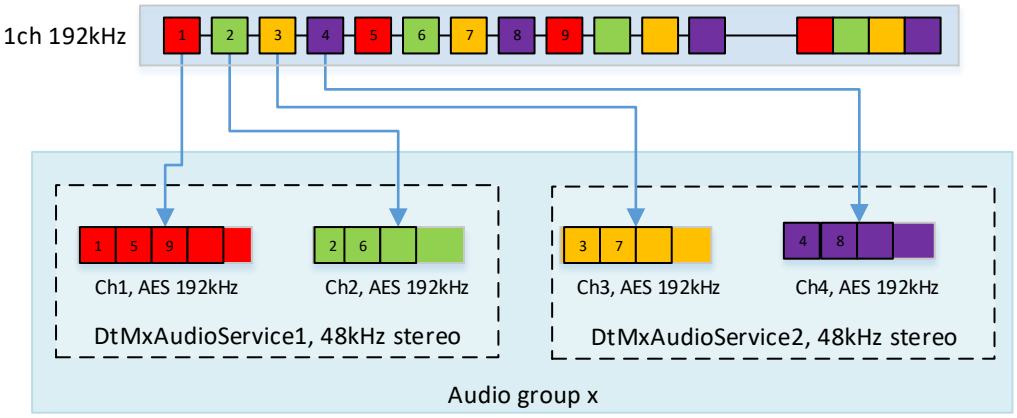
To output Dolby Atmos with the DTA-2195, 192-kHz audio data must be packetized in SDI packets. The DTA-2195 then transcodes the audio data from SDI to the HDMI output.

SMPTE does not specify a data format for packetizing 192-kHz audio data in a SDI packet. To support Dolby Atmos on the DTA-2195 HDMI output, a DekTec-defined data format is used to transfer the 192-kHz audio data in a SDI frame: The 192-kHz audio channel is demultiplexed into 4 48-kHz channels, while the AES-signaling bits are set to "192 kHz". It is still possible to combine standard 48-kHz audio channels besides the 192-kHz audio channels in one SDI frame so that SDI compatible devices can select the standard 48-kHz audio channels.

In the Matrix API each 48-kHz stereo channel is described in one audio service (class `DtMxAudioService`). For one 192-kHz channel two audio services are required in one audio group. The audio services for the 192-kHz channel must be initialized as if it was a standard 48 kHz channel.

The demultiplexing process is explained in the figures below.

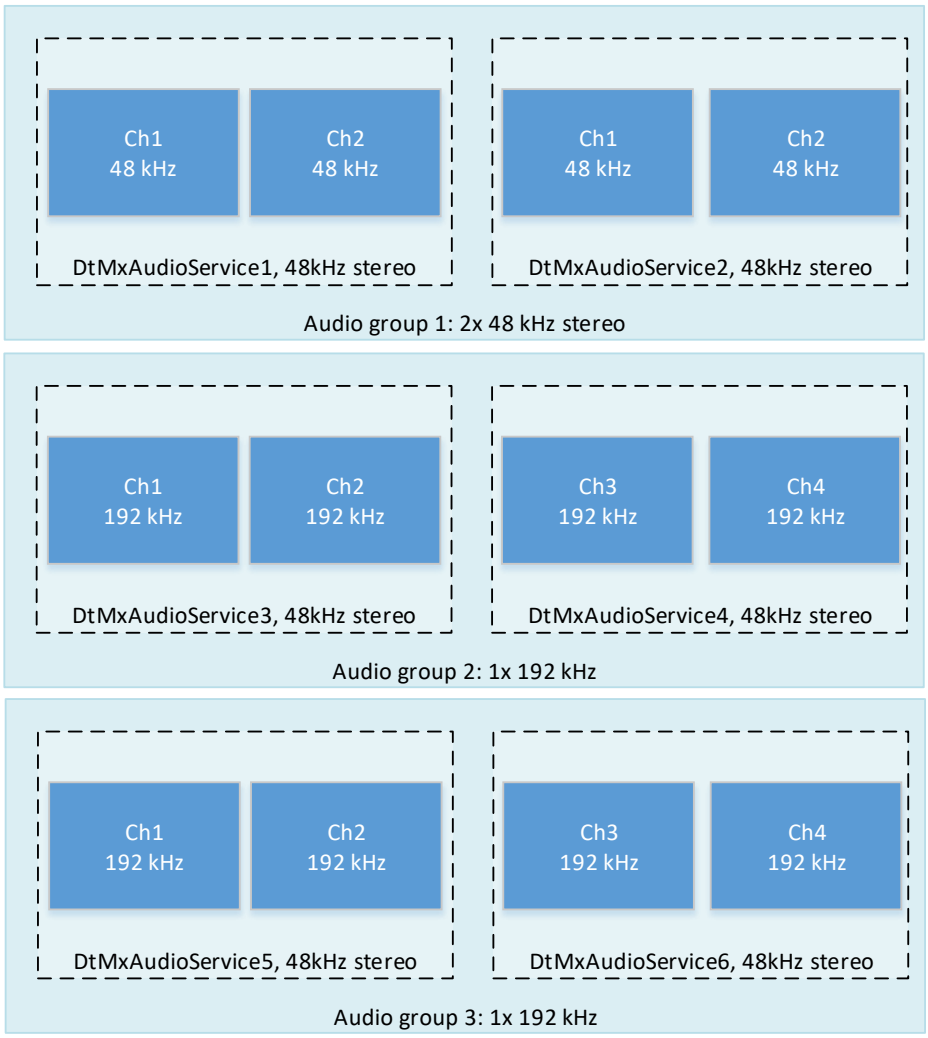




Use the DtAvOutput::SetHdmiAudioChannel(int AudioCh1, int AudioCh2) function to select the correct audio channel for the HDMI output.
Parameter AudioCh1 must be the first channel in the audio group of the first 192 KHz audio data. AudioCh2 must be the first channel in the audio group of the second 192 KHz audio data.
E.g. DtAvOutput::SetHdmiChannel(1, 5);
In this example audio group one contains the first 192 KHz output channel and audio group 2 contains the second 192 KHz channel.
See the DtMx4KPlayerDemo application for a complete example for using 192 KHz audio.

Combining 48 kHz and 192-kHz audio channels is as simple as using both separately.

In the following composition two stereo 48-kHz audio channels are combined with two 192-kHz audio channels in one SDI frame.



4.2. AES channel status bits

To distinguish the 48-kHz audio channels with the 192-kHz audio channels DekTec uses the AES channel status bits to indicate the (encoded) sampling frequency. For the 192-kHz audio channels, the sampling frequency must be set to 192 kHz.

The following bits in the channel status indicate the sampling frequency for the 'Professional applications' digital audio interface:

Byte 0 bit 6...7 must be set to: 0 0
Byte 4 bit 35...38 must be set to: 1 1 0 0 (192 kHz sampling frequency)

For details, please refer to IEC-60958-4 section 5.2 for the 'Professional applications' digital audio interface.

4.3. Matrix API® 2.0 Example Code

In the following code snipped, the de-multiplexing of one 192-kHz audio channel into four 48-kHz audio channels is displayed. This code is executed in a new frame callback function. Function `Get192KHzAudioSamples` delivers the 192-kHz audio samples. `pData` points to argument `DtMxData`

of the callback function. The `InitChannelStatus` function initializes the AES channel status bits so that it is indicating 192 kHz.

```
DtMxRowData&  OurRow = pData->m_Rows[0];
DtMxAudioData&  AudioData = OurRow.m_CurFrame->m_Audio;

// Get new 192-kHz audio samples
unsigned int* pAudioSamples = Get192KHzAudioSamples(NumSamples);

// Interleave the 192khz audio samples into the 4 channels
for (int Index=0; Index<4; Index++)
{
   DtMxAudioChannel&  Channel = AudioData.m_Channels[Index];
   for (int k=0; k<NumSamples/4; k++)
      Channel.m_pBuf[k] = pAudioSamples [Index + k*4];
   Channel.m_NumValidSamples = NumSamples / 4;

   // Init the channel status indicating 192 kHz sampling frequency
   InitChannelStatus(Channel, 192000);
}
```

See the source of `DtMxHdmi192kHzAudioDemo` for a complete example.