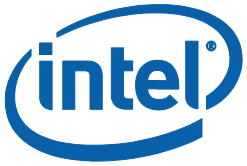


Intel® Smart Sound Technology NHLT Specification

Architecture Guide/Overview

January 2020

Revision 0.8.1

Contents

Figures

Revision History

Document Number	Revision Number	Description	Revision Date
561555	0.6	Initial release.	October 2015
	0.7	Clarifications added	December 2015
~~NA~~	~~0.7~~	~~Re-release as public document~~	~~February 2016~~
595976	0.8	Significant bug fixes	May 2018
	0.8.1	KBL platforms corrections	January 2020

§ §

1 Introduction

1.1 Purpose of this Document

This document describes methods used in Intel® Smart Sound Technology Audio DSP products for non-HD Audio endpoints configuration performed by FW via ACPI. This configuration is applicable to both Windows* and Linux* based operating systems used in Skylake products. The goal is to provide information for BIOS, FW and SW engineers to implement an NHLT audio configuration.

1.2 Document Scope

The Intel® Smart Sound Technology Audio DSP Non-HD Audio ACPI HLD explains the structure of ACPI tables used to configure non-HD Audio endpoints connected to Audio DSP. The endpoints are connected to Audio DSP subsystem via either I2S/TDM (SSP) or PDM (DMIC). The document also describes suggested configuration method for Non-HD Audio Codecs.

1.3 Terminology

Term	Description
A2DP	Advance Audio Distribution Profile
ACPI	Advanced Configuration and Programming Interface version 5.0
ALSA	Advanced Linux* Sound Architecture
ASoC	ALSA System on Chip layer
Blob	Binary information which contain configuration data and other properties for each HW end point of the SKL SoC. The HW endpoints connect to specific hardware. E.g. an entry that contains DMIC coefficients for the DMIC end point.
BT/Bluetooth	Bluetooth is a local connectivity wireless protocol. Bluetooth supports the transport of voice signals (i.e. wireless headsets). Bluetooth basebands typically have a PCM audio interface for the voice data and a UART or USB for control, data, and compressed audio (using Sub-Band Coding)
DFW	Dynamic Firmware also called topology binary. This can be seen as a collection of pipelines connecting the SOC with the endpoints through the ADSP for various use cases. E.g. it has pipeline for connecting playback path to SSP0 port which in turn connects to ADI speaker. It'll also have a playback and record path to SSP1 which connects to Nuvoton codec. The pipeline connection & selection is done through the exposed mixer controls.
DSP	The firmware which is loaded to the DSP IP. This contains all the algorithms and other modules (like wake on voice / speaker protection modules. Copier module for passing data, mixers and switches to route data). For our discussion below we can assume the DSP firmware to be a constant.
HDMI	High Definition Multimedia Interface
LPE	Low Power Engine (Audio)

Term	Description
NHLT	Non HD Audio Link Table. This is an ACPI table containing non-HD Audio endpoints and links configuration.
Sx	Wildcard for S3/S4/S5 – Platform specific ACPI Power Management states (described in ACPI spec)
Sinks	Audio outputs / speakers
Sources	Audio inputs / microphone
HW CODEC	Hardware component which supports audio sinks and sources.
PCM	Pulse Code Modulation. Standard technique of representing an audio stream using x bits sampled uniformly y times a second. Each sample captures the amplitude of the signal at that point in time. PCM samples are sent over serial buses between processors and audio codecs
PnP	Plug and Play
SAS	Software Architecture Specification
SSP	Synchronous Serial Port – the I2S/TDM port
SST	Smart Sound Technology
VoIP	Voice over Internet Protocol

1.4 Reference Documents

Document	Document No./Location
Microphone array support in Windows – describes geometries and microphone type definitions	https://msdn.microsoft.com/en-us/library/windows/hardware/dn613960(v=vs.85).aspx
Information on ACPI header	http://lxr.free-electrons.com/source/include/acpi/actbl.h#L101

§ §

2 Non-HD Audio Endpoint Description Table

Any Non-HD Audio endpoint that the platform needs to use must have a corresponding NHLT entry. If a particular non-HD Audio endpoint has been disabled, then that device or link will not be present in the ACPI configuration table.

The non-HD Audio endpoint information will be defined as an ACPI Data Table consisting of the standard ACPI Table Header followed by an endpoint descriptor for each non-HD Audio endpoint to be supported.

The platform driver retrieves ACPI configuration table by executing the following _DSM method:

```
// Interface ID: A69F886E-6CEB-4594-A41F-7B5DCE24C553
// Function 1: Query Non HD Audio Descriptor Table
// This function is used by SST driver to discover the
// non HD Audio devices supported by the audio DSP.
Arg0 — UUID: A69F886E-6CEB-4594-A41F-7B5DCE24C553 (Buffer)
Arg1 — Revision ID: 0x01 (Integer)
// When the data structure changes this will change.  Not currently implemented
Arg2 — Function Index: 0x01 (Integer)
Arg3 — Unused
Return — ACPI Table describing the non HD Audio links and devices supported by the
audio DSP. (Buffer)
```

The following diagram illustrates the layout of the table:

Figure 1 Table Layout

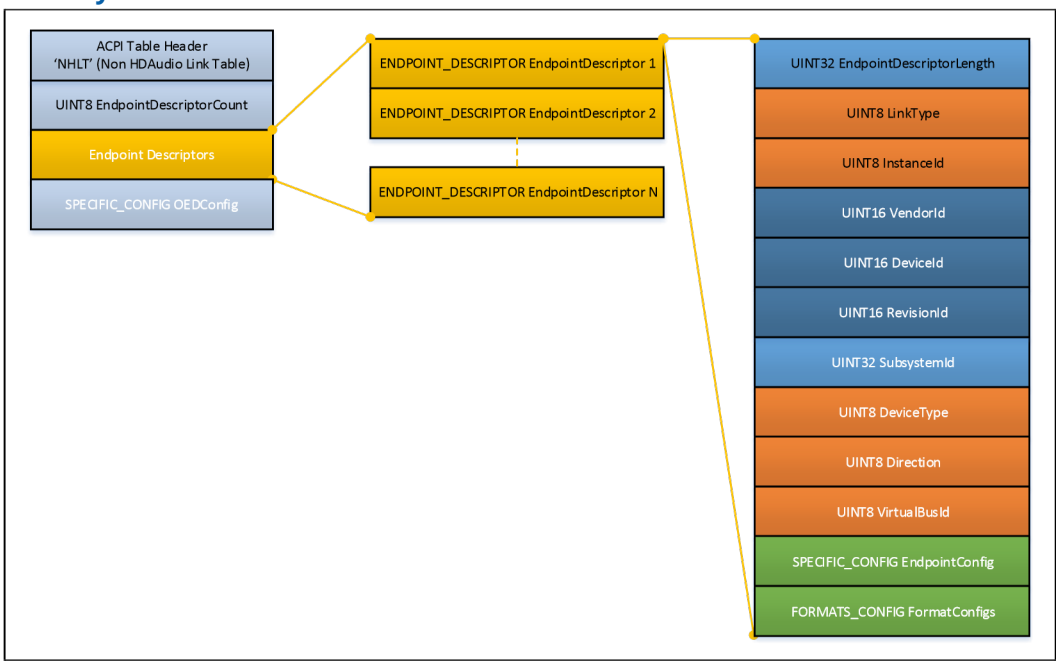


Table 2-1. NHLT Endpoint Descriptor

Size (B)	Name	Description
4	EndpointDescriptorLength	Size of entire endpoint descriptor. This includes the size of the "EndpointDescriptorLength" field.
1 (4 bits only)	LinkType	Type of link: 0 – reserved for HD-Audio 1 – reserved for DSP 2 – PDM 3 – SSP 4 – reserved for Slimbus 5 – reserved for SoundWire 6-7 – reserved for future use
1	Instance ID	Virtual device instance unique to link type (in range 0 - 7) First or only device on link should have InstanceId = 0. All devices with the same LinkType, DeviceType and InstanceId will be exposed by the same CFD. If OEM wants to expose separate CFD for subset of devices with given DeviceType on particular LinkType then another InstanceId should be used for these selected devices. i.e: If there are 2 different InstaceId used for devices with the same LinkType and DeviceType then 2 CFDs shall be exposed accordingly. Intel virtual device drivers support only InstanceID = 0. InstanceID other than 0 require virtual device driver provided

Size (B)	Name	Description
		by OEM.
2	Vendor ID	Virtual device Vendor ID used for building PnP address for matching SW driver to device. For Intel virtual device drivers should be always 0x8086.
2	Device ID	Virtual Device ID used for building PnP address for matching SW driver to device. For Intel virtual device drivers should always match ID of Intel provided virtual device drivers: 0xAE20 – for PDM DMIC 0xAE30 – for BT sideband 0xAE34 – for I2S/TDM codecs
2	Revision ID	Virtual device Revision ID used for building PnP address for matching SW driver to device. OEM platform revision ID should be used.
4	Subsystem ID	Virtual device Subsystem ID used for building PnP address for matching SW driver to device. OEM unique platform subsystem ID should be used.
1	Device Type	Type of device (unique to link type): SSP Link: 0 – BT Sideband 1 – FM 2 – Modem 3 – reserved for future use 4 – SSP Analog Codec 5-7 – reserved for future use PDM Link: 0 – PDM (on cAVS1.8 (CNL+) platforms only) 1 – PDM on cAVS1.5 (KBL) based platforms 2-7 – reserved Slimbus Link: 0-7 – TBD Soundwire Link: 0-7 – TBD
1	Direction	Endpoint direction: 0 – Render 1 – Capture 2 – Render with loopback / feedback 3 – Feedback for render (smartamp)
1	Virtual Bus ID	Virtual Bus Line For SSP Link Type this is SSP port number. For DMIC this is always 0 because there is only one PDM link seen from SW/FW point of view.

2.1 NHLT Element Descriptions

Note that the SPECIFIC_CONFIG and FORMATS_CONFIG structures are defined below.

```
typedef struct _SPECIFIC_CONFIG
{
        UINT32          CapabilitiesSize; //does not include size of this field
        BYTE[]          Capabilities;
} SPECIFIC_CONFIG;
```

SPECIFIC_CONFIG is used to differentiate between TDM and I2S mode for the SSP ports and is primarily used for DMIC specifics.

```
typedef struct _FORMATS_CONFIG
{
        UINT8                           FormatsCount;
        FORMAT_CONFIG[]                 FormatsConfiguration;
} FORMATS_CONFIG;
```

Additionally, the FORMAT_CONFIG structure included in the FORMATS_CONFIG structure is defined as:

```
typedef struct _FORMAT_CONFIG
{
        WAVEFORMATEXTENSIBLE                Format;
        SPECIFIC_CONFIG                 FormatConfiguration;
} FORMAT_CONFIG;
```

Where standard datatypes are defined as follows (simplified format):

```
typedef struct {
        WORD  wFormatTag;
        WORD  nChannels;
        DWORD nSamplesPerSec;
        DWORD nAvgBytesPerSec;
        WORD  nBlockAlign;
        WORD  wBitsPerSample;
        WORD  cbSize;
        WORD  wValidBitsPerSample;
        DWORD dwChannelMask;
        GUID  SubFormat;
} WAVEFORMATEXTENSIBLE;
```

Table 2-2. WaveFormatExtensible

Size (B)	Name	Description
2	wFormatTag	Format type of Waveform-audio, it is always 0xFFFE for

<table>
<tr><th>Size (B)</th><th>Name</th><th>Description</th></tr>
<tr><td></td><td></td><td>WAVEFORMATEXTENSIBLE structure.</td></tr>
<tr><td>2</td><td>wChannels</td><td>Number of channels in waveform audio data</td></tr>
<tr><td>4</td><td>dSamplesPerSec</td><td>Waveformat audio data sample rate in samples per second</td></tr>
<tr><td>4</td><td>dAvgBytesPerSec</td><td>Waveformat audio required average data transfer rate measured in bytes per second</td></tr>
<tr><td>2</td><td>wBlockAlign</td><td>Block alignment in bytes, must be equal to value of nChannels and wBitsPerSample divided by 8 (as bits in byte)</td></tr>
<tr><td>2</td><td>wBitsPerSample</td><td>Number of bits per each sample of waveformat audio data, it may be any integer multiple of 8. This represent the container size and may be larger or equal compared to sample size.</td></tr>
<tr><td>2</td><td>cbSize</td><td>Size in bytes of extra format information appended after cbSize field, this field is set always to 22 bytes.</td></tr>
<tr><td>2</td><td>wValidBitsPerSample</td><td>Number of bits of precision in audio signal, may be equal to size of container as indicated in wBitsPerSample or smaller.</td></tr>
<tr><td>4</td><td>dwChannelMask</td><td>Bitmask specifying assignment of channels in the stream to speaker positions in the following manner:
<table>
<tr><th>Speaker Position</th><th>Flag Bit</th></tr>
<tr><td>SPEAKER_FRONT_LEFT</td><td>0x1</td></tr>
<tr><td>SPEAKER_FRONT_RIGHT</td><td>0x2</td></tr>
<tr><td>SPEAKER_FRONT_CENTER</td><td>0x4</td></tr>
<tr><td>SPEAKER_LOW_FREQUENCY</td><td>0x8</td></tr>
<tr><td>SPEAKER_BACK_LEFT</td><td>0x10</td></tr>
<tr><td>SPEAKER_BACK_RIGHT</td><td>0x20</td></tr>
<tr><td>SPEAKER_FRONT_LEFT_OF_CENTER</td><td>0x40</td></tr>
<tr><td>SPEAKER_FRONT_RIGHT_OF_CENTER</td><td>0x80</td></tr>
<tr><td>SPEAKER_BACK_CENTER</td><td>0x100</td></tr>
<tr><td>SPEAKER_SIDE_LEFT</td><td>0x200</td></tr>
<tr><td>SPEAKER_SIDE_RIGHT</td><td>0x400</td></tr>
<tr><td>SPEAKER_TOP_CENTER</td><td>0x800</td></tr>
<tr><td>SPEAKER_TOP_FRONT_LEFT</td><td>0x1000</td></tr>
<tr><td>SPEAKER_TOP_FRONT_CENTER</td><td>0x2000</td></tr>
<tr><td>SPEAKER_TOP_FRONT_RIGHT</td><td>0x4000</td></tr>
<tr><td>SPEAKER_TOP_BACK_LEFT</td><td>0x8000</td></tr>
<tr><td>SPEAKER_TOP_BACK_CENTER</td><td>0x10000</td></tr>
</table></td></tr>
</table>

Size (B)	Name	Description
		SPEAKER_TOP_BACK_RIGHT 0x20000 **NOTE:** This field is applicable only for playback. For capture endpoints defined as microphone arrays it should be always 0 (DIRECTOUT), and geometry would be provided in the microphone array data structures.
16	gSubFormat	Subformat of the data indicated by GUID, it may be either PCM format {0x00000001, 0x0000, 0x0010, {0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71} Or Vendor defined subformat GUID for proprietary data subformat.

Note: All the mentioned structures are 1-byte packed.

Format of UEFI FW ACPI Description header that describes NHLT:

```
typedef struct {
        UINT32  Signature;
        UINT32  Length;
        UINT8   Revision;
        UINT8   Checksum;
        UINT8   OemId[6];
        UINT64  OemTableId;
        UINT32  OemRevision;
        UINT32  CreatorId;
        UINT32  CreatorRevision;
} EFI_ACPI_DESCRIPTION_HEADER;
```

Note: For more information on ACPI header you can refer to Linux source at following location:

http://lxr.free-electrons.com/source/include/acpi/actbl.h#L101

Refer to ACPI specification, search for ACPI_TABLE_HEADER.

The following subsections describe low-level definitions used in ACPI Non-HD Audio Endpoint Description table for description of various endpoints connected to DSP subsystem for Intel® SST.

FORMAT_CONFIG structure follows by SPECIFIC_CONFIG structure and structure of SPECIFIC_CONFIG depends on the type DeviceType. In the subsequent section the SPECIFIC_CONFIG structure is explained for different Devices e.g. Microphones, Speakers etc.

2.2 Device Capabilities Structure

```
typedef struct _DEVICE_SPECIFIC_CONFIG
{
        BYTE            VirtualSlot; // timeslot for multichannel transmission
        BYTE            ConfigType; // 0 – basic, 1 – microphone array
} DEVICE_SPECIFIC_CONFIG;

enum eIntcConfigType
{
        eIntcConfigTypeGeneric  = 0,
        eIntcConfigTypeMicArray = 1,


        eIntcConfigTypeRenderWithLoopback = 2, //not supported in Windows

        eIntcConfigTypeRenderFeedback = 3,      //in case of endpoint capture direction means
feedback for render

}
```

Device specific configuration begins with the header _DEVICE_SPECIFIC_CONFIG as defined above. This is generic header which is present in all types of device specific configuration (SPECIFIC_CONFIG).

2.3 Multi-channel Transmission

Streams on multichannel buses (i.e.: TDM) are addressed by SW using virtual slots. FW maps link channels into virtual slots based on configuration passed by driver by reading the NHLT table.

Examples of multichannel mapping:

Capture Example.

Link Channels	0	1	2	3	4	5	6	7
Virtual Slot Assignments	3	3	0	1	2	2	2	2
Stream Channel Assignments	0	1	0	0	3	0	2	1
Stream Description	Render (HP) Loopback		Voice	Speech	Raw Capture (4-ch)			

Maximum number of channels that can be transmitted or received is restricted to 8. This is defined as “link channels” in the above table. Link channels can be grouped together to form a “stream”. This group of link channels are assigned a “virtual slot”.

Consider the capture example given above. Let's say there is a codec connected onto I2S Port, which transmits four different streams.

1. Loopback stream.
2. Voice stream
3. Speech stream
4. Raw capture stream.

Loopback stream is transmitted in Link Slot 0 and Link Slot 1. In this example this has been given a virtual slot ID of 3. So virtual slot ID can also be looked as "Stream ID". The "Stream ID" 3 is transmitted by I2S device in Link Slot 0 and Link Slot 1.

Maximum there can be 8 virtual slot ID, considering 8 MONO channels. In that case each virtual slot will correspond to one Link slot.

The example capture mapping is translated into following virtual slots.

Virtual Slot	Description	StreamType	Link Channels (As Ordered In Stream)
0	Voice	Voice	2
1	Speech	Speech	3
2	Raw Capture		5,7,6,4
3	Render (HP) Loopback	Loopback	0,1

Render example:

Link Channels	0	1	2	3	4	5	6	7
Virtual Slot Assignments	0	1	0	1	1	1	1	1
Stream Channel Assignment	1	1	0	0	3	2	4	5
Stream Description	Render (HP)	Render(Spk)	Render(HP)	Render (Speakers)				

Likewise consider Render case. In the above example there are two Virtual Slots or "Stream ID". Stream ID 0/Virtual Slot 0 corresponds to Link channel 0 and Link channel 2. So considering I2S link as example link, Device would receive Headphone stream in slot 0 and 2. This is bit weird but it's possible. That's why taken as example. Second Stream, Stream ID 1/Virtual Slot 1 is routed to Speakers and in this example there are 6 speakers assumed on the codec side which receives data via Link channels 1, 3, 4, 5, 6 and 7.

The example render mapping is translated into following virtual slots.

Virtual Slot	Description	Stream Type	Link Channels (As Ordered in Stream)
0	HP	-	2,0
1	Speakers	-	3,1,5,4,6,7

Note: One endpoint is using only one virtual bus (link line).

2.4 Microphone Array Device Capabilities Structure

This section defines the SPECIFIC_CONFIG structure for microphone arrays.

```
typedef struct _MIC_ARRAY_DEVICE_SPECIFIC_CONFIG
{
        DEVICE_SPECIFIC_CONFIG DeviceConfig;
        BYTE                   ArrayTypeEx;
} MIC_ARRAY_DEVICE_SPECIFIC_CONFIG;
```

where ArrayTypeEx is defined as follows:

Bits [7:4]	Bits [3:0]
Array Definition Extension	Array Type

Following values in bits[3:0] are used for array types:

Table 2-3. ArrayType Description

ArrayType	Description
0xA	Linear 2-element, Small
0xB	Linear 2-element, Big
0xC	Linear 4-element, 1st geometry
0xD	Planar L-shaped 4-element
0xE	Linear 4-element, 2nd geometry
0xF	vendor defined

Following values in bits[7:4] are used for array definition extension definition:

Bits[7:4] value	Array Definition Extension
0x0	No extension
0x1	Microphone SNR and Sensitivity extension
0x2-0xF	*Reserved for future use*

In many cases platform vendor would like to have its own microphone placement and in that case they can use the following structure.

```
typedef struct _VENDOR_MIC_ARRAY_DEVICE_SPECIFIC_CONFIG
{
        MIC_ARRAY_DEVICE_SPECIFIC_CONFIG MicArrayDeviceConfig;
        BYTE                    NumberOfMicrophones;
        VENDOR_MIC_CONFIG           MicConfig[NumberOfMicrophones];
} VENDOR_MIC_ARRAY_DEVICE_SPECIFIC_CONFIG;
```

```
typedef struct _VENDOR_MIC_CONFIG
{
        BYTE            Type;
        BYTE            Panel;
        WORD            SpeakerPositionDistance; // mm
        WORD            HorizontalOffset; // mm
        WORD            VerticalOffset; // mm
        BYTE            FrequencyLowBand; // 5*Hz
        BYTE            FrequencyHighBand; // 500*Hz
        SHORT           DirectionAngle; // -180 - + 180
        SHORT           ElevationAngle; // -180 - + 180
        SHORT           WorkVerticalAngleBegin; // -180 - + 180   with 2 deg step
        SHORT           WorkVerticalAngleEnd; // -180 - + 180         with 2 deg step
        SHORT           WorkHorizontalAngleBegin; // -180 - + 180      with 2 deg step
        SHORT           WorkHorizontalAngleEnd; // -180 - + 180       with 2 deg step
} VENDOR_MIC_CONFIG;
```

Table 2-4. Microphone Type Description

Value	Type of Microphone
0	KSMICARRAY_MICTYPE_OMNIDIRECTIONAL
1	KSMICARRAY_MICTYPE_SUBCARDIOID
2	KSMICARRAY_MICTYPE_CARDIOID
3	KSMICARRAY_MICTYPE_SUPERCARDIOID
4	KSMICARRAY_MICTYPE_HYPERCARDIOID
5	KSMICARRAY_MICTYPE_8SHAPED
6	Reserved
7	KSMICARRAY_MICTYPE_VENDORDEFINED

Table 2-5. Panel (Location of the Microphone)

Value	Panel (Location)
0	Top

Value	Panel (Location)
1	Bottom
2	Left
3	Right
4	Front (default)
5	Rear

In case array definition extension is set to non-zero value microphone array structure (VENDOR_MIC_ARRAY_DEVICE_SPECIFIC_CONFIG for vendor defined array type (0xF) or MIC_ARRAY_DEVICE_SPECIFIC_CONFIG for other array types) is followed by one of below microphone array extensions.

2.4.1 Microphone SNR and Sensitivity extension

Microphone SNR and Sensitivity extension is defined as follows:

```
typedef struct _MIC_SNR_SENSITIVITY_EXTENSION
{
        LONG    SNR;
        LONG    Sensitivity;
} MIC_SNR_SENSITIVITY_EXTENSION;
```

Where:

SNR

SNR specifies the microphone signal to noise ratio (SNR) measured in dB units. The value uses fixed point decimal representation. The data is stored as a 16.16 fixed point value. The upper 16 bits are used for the whole number of the value and the lower 16 bits are used for the fractional portion of the value.

Sensitivity

Sensitivity specifies the microphone sensitivity in decibels relative to full scale (dBFS) units. The value uses fixed point decimal representation. The data is stored as a 16.16 fixed point value. The upper 16 bits are used for the whole number of the value and the lower 16 bits are used for the fractional portion of the value.

Complete microphone array definition structures will look like:
For vendor defined microphone arrays:

```
typedef struct _VENDOR_MIC_ARRAY_DEVICE_SPECIFIC_CONFIG_SNR_EXT
{
        VENDOR_MIC_ARRAY_DEVICE_SPECIFIC_CONFIG VendorMicArrayDeviceConfig;
        MIC_SNR_SENSITIVITY_EXTENSION           SnrSensitivityExt;
} VENDOR_MIC_ARRAY_DEVICE_SPECIFIC_CONFIG_SNR_EXT;
```

For other microphone arrays:

```
typedef struct _MIC_ARRAY_DEVICE_SPECIFIC_CONFIG_SNR_EXT
{
        MIC_ARRAY_DEVICE_SPECIFIC_CONFIG      MicArrayDeviceConfig;
        MIC_SNR_SENSITIVITY_EXTENSION         SnrSensitivityExt;
} MIC_ARRAY_DEVICE_SPECIFIC_CONFIG_SNR_EXT;
```

2.5 Microphone Array Coordinates

This section explains microphone positions.

Note that all positions and angles are related to “speaker position” as defined in MSFT specifications.

2.5.1 Front (Main) Microphone Array Coordinates

Below pictures show coordinate system of front (or main) microphone array in case of laptop and tablet.

Figure 2 Coordinate System of Front (Main) Laptop Microphone Array

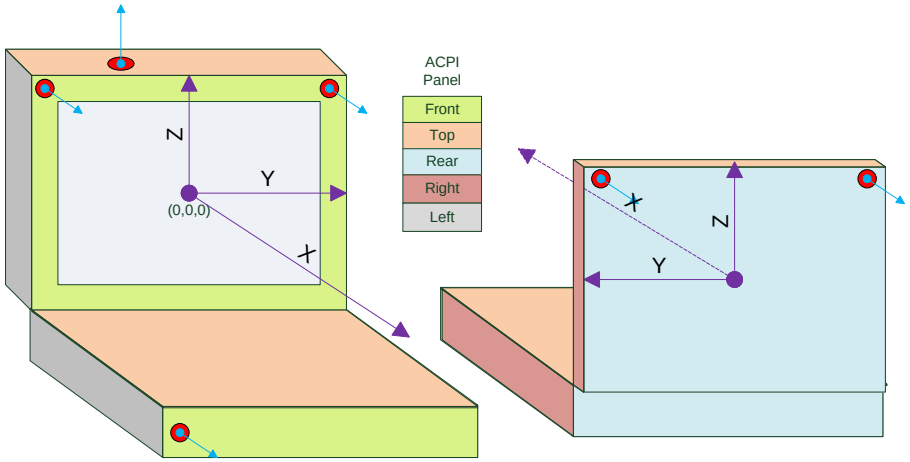
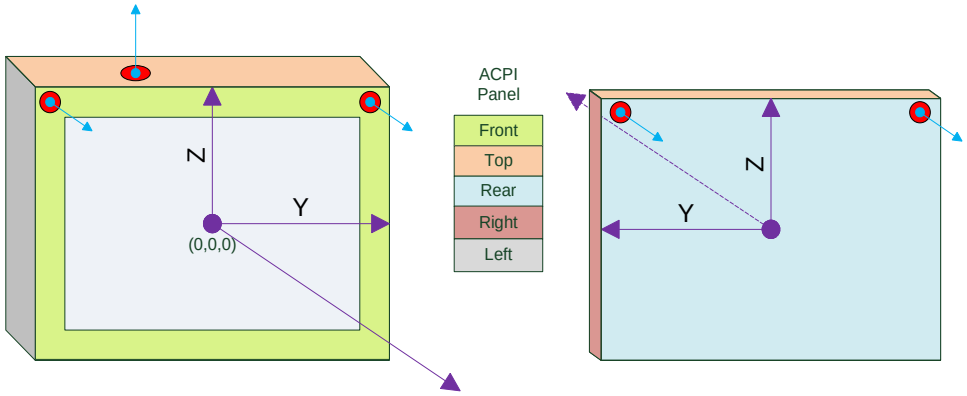


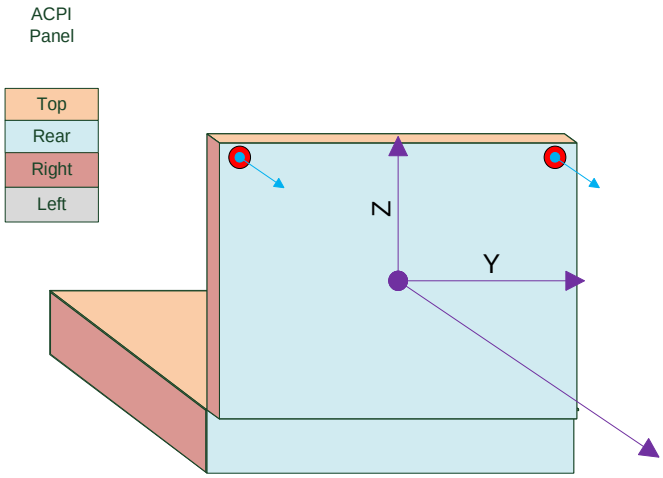
Figure 3 Coordinate System of Front (Main) Tablet Microphone Array



2.5.2 Rear (Additional) Microphone Array Coordinates

Below pictures show coordinate system of rear (additional to front) microphone array in case of laptop and tablet. Rear microphone array works with speaker in front of the rear panel.

Figure 4 Coordinate System of Rear (Additional) Microphone Array



2.6 Render device with feedback

In case render endpoint have HW feedback (for Smartamp), capture virtual slot is provided.

```
typedef struct _RENDER_FEEDBACK_DEVICE_SPECIFIC_CONFIG
{
        DEVICE_SPECIFIC_CONFIG DeviceConfig;
        BYTE            FeedbackVirtualSlot;    //render slot in case of capture
        WORD            FeedbackChannels;       //informative only
        WORD            FeedbackValidBitsPerSample;
} RENDER_FEEDBACK_DEVICE_SPECIFIC_CONFIG;
```

Where:

FeedbackVirtualSlot

In case of render direction FeedbackVirtualSlot specifies virtual slot for stream to transmit render feedback data. In case of capture direction FeedbackVirtualSlot specifies virtual slot for render stream the feedback is provided for.

FeedbackChannels

In case of render direction FeedbackChannels specifies number of channels of stream to transmit render feedback data. In case of capture direction FeedbackChannels specifies number of channels of render stream the feedback is provided for.

FeedbackValidBitsPerSample

In case of render direction FeedbackValidBitsPerSample specifies valid bits per sample of stream to transmit render feedback data. In case of capture direction FeedbackValidBitsPerSample specifies valid bits per sample of render stream the feedback is provided for.

§ §

3 Examples

3.1 DMIC Connected via PDM Interface

This example provides sample values for a platform with 2 PDM DMICs that needs to be used either in 16 KHz, 2 channels, 16 bit mode or 48 KHz, 2 channels, 24bit mode.

In this case there would be one ENDPOINT DESCRIPTOR as shown in the table below.

Table 3-1-1. ENDPOINT_DESCRIPTOR [0]

Fields	Value	Description
EndpointDescriptorLength	X	Length of the endpoint descriptor
LinkType	2	PDM
InstanceId	0	Always 0 for Intel drivers.
VendorId	0x8086	Intel Vendor ID
DeviceId	0xAE20	Intel Virtual PDM Device ID
RevisionId	X	OEM Platform Revision ID
SubsystemId	X	OEM Platform Subsystem ID
DeviceType	1 for KBL 0 for CNL+	Type of the device connected on this link. For PDM this is always set to 1 for cAVS1.5 (KBL) based platforms or 0 for cAVS1.8 (CNL+) platforms.
Direction	1	Capture
VirtualBusId	0	Always 0 for PDM

ENDPOINT_DESCRIPTOR is followed by SPECIFIC_CONFIG, in this case it is

MIC_ARRAY_DEVICE_SPECIFIC_CONFIG as shown in the table below.

Table 3-1-2. MIC_ARRAY_DEVICE_SPECIFIC_CONFIG (SPECIFIC_CONFIG)

Fields	Value	Description
CapabilitiesSize	3	Size of the capabilities in bytes
VirtualSlot	0	Virtual slot ID as explained in section 2.3. Always 0 for PDM.
ConfigType	1	Microphone Array
ArrayType	0xA	Microphone Array type is Linear 2-element, Small

These microphone devices can be configured either in 16 KHz, 2 ch, 16bit or 48 KHz, 2ch, 24bit mode, so it requires two FORMAT_CONFIG which are defined in the following table.

Table 3-1-3. FORMAT_CONFIG

Fields	Value	Description

Table

FormatConfigCount	2	Two configurations, one for 16KHz, 2 ch and 16bit and second for 48Khz, 2 ch and 24 bit mode.
FORMAT_CONFIG[0]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	WAVEFORMATEXTENSIBLE for 16KHz, 2 channels, 16bit This also contains SPECIFIC_CONFIG which contains firmware specific data structure and register settings.
FORMAT_CONFIG[1]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	WAVEFORMATEXTENSIBLE for 48KHz, 2 channels, 24bit This also contains SPECIFIC_CONFIG which contains firmware specific data structure and register settings.

3.2 SSP (TDM Mode) Device ACPI Configuration

This example provides sample configuration for an I2S based codec connected to SoC via I2S interface, communicating in multi slot TDM Mode.

Between codec and SoC there are basically two render streams, one for headset and one for speakers. In addition to that there are two capture streams. The first capture stream is receiving data from DMIC0 and DMIC1, and second capture stream is receiving data from DMIC2 and DMIC3.

Headset stream is getting transmitted on VirtualSlot0 and Speaker stream is getting transmitted on VirtualSlot1. This information is represented by ENDPOINT_CONFIGURATION[0] and [1] respectively.

Similarly DMIC(0,1) stream is getting received on VirtualSlot0 and DMIC(2,3) stream is getting received on VirtualSlot1. This information is represented by ENDPOINT_CONFIGURATION[2] and [3] respectively.

Table 3-2-1. ENDPOINT_CONFIGURATION[0] (Headset)

Fields	Value	Description
EndpointDescriptorLength	X	Length of the endpoint descriptor
LinkType	3	I2S
InstanceId	0	Always 0 for Intel virtual device.
VendorId	0x8086	Intel Vendor ID
DeviceId	0xAE34	Intel Virtual SSP Device ID
RevisionId	X	OEM Platform Revision ID

SubsystemId	X	OEM Platform Subsystem ID
DeviceType	4	Type of the device connected on this link. 4 for analog codec device.
Direction	0	Playback/Render
VirtualBusId	1	I2S 1 Port

Table 3-2-2. DEVICE_SPECIFIC_CONFIG (Headset)

Fields	Value	Description
CapabilitiesSize	2	Size of the capabilities
VirtualSlot	0	Virtual slot ID as explained in section 2.3 In this case the codec headset receives data from Virtual slot 0, which corresponds to Link Slot 0 and 1.
ConfigType	0	Basic configuration

Table 3-2-3. FORMAT_CONFIG (Headset)

Fields	Value	Description
FormatConfigCount	1	48Khz, 2 ch and 24 bit mode.
FORMAT_CONFIG[0]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	WAVEFORMATEXTENSIBLE for 48Khz, 2 ch and 24 bit mode. This also contains SPECIFIC_CONFIG which contains firmware specific data structure and register settings. Virtual Slot 0 in the DEVICE_SPECIFIC_CONFIG would have a corresponding entry in the blob (SPECIFIC_CONFIG) of FORMAT_CONFIG which would indicate to transmit it in Slot 0 and Slot 1 of the Link.

Table

3.3 Sample NHLT Example

This section explains sample NHLT configuration by taking a hypothetical SKL reference platform. This section should be taken as example only and may not match with a direct platform.

3.3.1 Platform Connectivity

The figure below shows platform connectivity for the SKL reference platform.

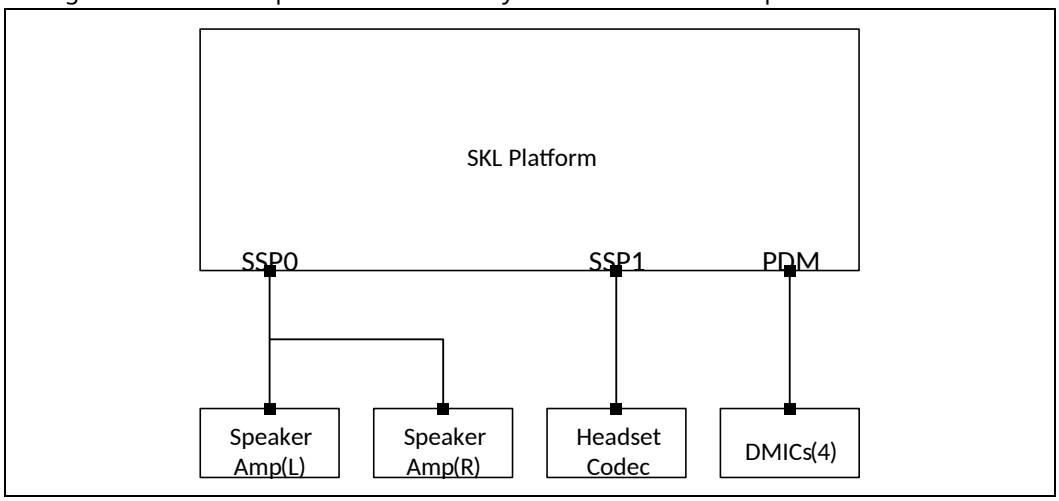


Figure 5 Example platform connectivity

SSP0 is connected to speaker amplifiers and configured in TDM mode, 2 slots

SSP1 is connected to headset and is configured in I2S mode

3.3.2 NHLT Table

There are 4 Endpoint Descriptors are required for based on the above configuration. Endpoint configurations are given in Table 3-27.

Table 3-3-1. NHLT Table

Fields	Value	Description
ACPI Table Header	See description	Please fill the values as per structure EFI_ACPI_DESCRIPTION_HEADER.
EndpointDescriptorCount	4	Number of endpoints described in NHLT
EndpointDescriptor[0]	See below	Endpoint configuration for DMICs. Even though there are four DMICs they are exposed as a single endpoints because DMICs cannot be configured independently with different

		parameters.
EndpointDescriptor[1]	See below	Endpoint configuration for Speaker Amplifiers, even though there are two different speaker amplifiers it is exposed as a single endpoint because they are not expected to be two different endpoints.
EndpointDescriptor[2]	See below	Endpoint configuration for Headset codec. Playback Endpoint configuration.
EndpointDescriptor[3]	See below	Endpoint configuration for Headset codec. Capture Endpoint configuration.
CapabilitiesSize	0	SPECIFIC_CONFIG OEDConfig field present should have size of 0.

Table 3-3-2. ENDPOINT_DESCRIPTOR [0] (DMICs)

Fields	Value	Description
EndpointDescriptorLength	X	Size of entire endpoint descriptor, this includes the size of the “EndpointDescriptorLength” field.
LinkType	2	PDM
InstanceId	0	Always 0.
VendorId	0x8086	Intel Vendor ID used for building PnP address for matching Windows SW driver to device.
DeviceId	0xae20	Intel Virtual device ID used for building PnP address for matching Windows SW driver to device.
RevisionId	X	OEM Platform revision.
SubsystemId	X	OEM Platform Subsystem ID.
DeviceType	1 on KBL 0 on CNL+	PDM device. Note: 1 – on cAVS 1.5 platforms / 0 – on cAVS 1.8 platforms
Direction	1	Capture
VirtualBusId	0	There is only 1 PDM link, even though there are 4 different microphones connected, they are seen as Single endpoint.

Table 3-3-3. DEVICE_SPECIFIC_CONFIG

Fields	Value	Description
CapabilitiesSize	2	Size of the capabilities
VirtualSlot	0	Virtual slot ID as explained in section 2.3 In this case the codec headset receives data from Virtual slot 0, which corresponds to Link Slot 0 and 1.
ConfigType	0	Basic configuration

Table 3-3-4. FORMATS_CONFIG

Fields	Value	Description

Table

FormatConfigCount	2	Two types configurations for DMIC endpoints.
FORMAT_CONFIG[0]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	16Khz, 24bit, 2 channels
FORMAT_CONFIG[1]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	48Khz, 24bit, 4 channels

Note: There are two Format Configuration required based on the SKL reference use cases.

FORMAT_CONFIG[0] – This would be used in the capture use case where firmware would be doing the 24 bit to 16bit conversion. However the blob would configure the PDM blocks into 24bit mode.

FORMAT_CONFIG [1] – This would be used in two scenarios. One for the regular capture use case with 4 channel, 24bit, 4 channel mode and second for alternate capture use case simultaneously. The blob is same for the PDM blocks for 16KHz and 24KHz configuration. In all cases whenever required the 24bit to 16bit conversion is done by firmware and that's why the blob configures the PDM interface always in 24bit mode.

When the LinkType is PDM, Driver ignores the "frequency" field to search the table. The PDM interface always have same blob settings for 16 and 48KHz. This reduces the number of configurations and the size of the table.

Table 3-3-5. ENDPOINT_DESCRIPTOR [1] (Speaker Amplifiers - Playback)

Fields	Value	Description
EndpointDescriptorLength	X	Size of entire endpoint descriptor, this includes the size of the "EndpointDescriptorLength" field.
LinkType	3	SSP Link
InstanceId	0	Always 0.
VendorId	0x8086	Intel Vendor ID used for building PnP address for matching Windows SW driver to device.
DeviceId	0xae34	Intel Virtual device ID used for building PnP address for matching Windows SW driver to device.
RevisionId	X	OEM Platform Revision used for building PnP address for matching SW driver to device.
SubsystemId	X	OEM Platform Subsystem ID used for building PnP address for matching SW driver to device.
DeviceType	4	Device type is Analog Codec
Direction	0	Playback / Render
VirtualBusId	0	SSP0. Codec is connected to SSP0

Table 3-3-6. DEVICE_SPECIFIC_CONFIG (Speaker Amplifiers – Playback)

Fields	Value	Description
CapabilitiesSize	2	2 bytes for the capabilities.
VirtualSlot	0	There is only one virtual slot in the TDM mode. There are two Link slots in the TDM mode but it is seen as a single stream.
ConfigType	0	Basic

Table 3-3-7. FORMATS_CONFIG (Speaker Amplifiers – Playback)

Fields	Value	Description
FormatConfigCount	1	Only one configuration for the Speaker Amplifiers.
FORMAT_CONFIG[0]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	48Khz, 24bit, 2 channels in TDM mode.

Table 3-3-8. ENDPOINT_DESCRIPTOR [2] (Headset - Playback)

Fields	Value	Description
EndpointDescriptorLength	X	Size of entire endpoint descriptor, this includes the size of the “EndpointDescriptorLength” field.
LinkType	3	SSP
InstanceId	0	Always 0.
VendorId	0x8086	Intel Vendor ID used for building PnP address for matching SW driver to device. This is mainly used for loading Windows MINI port drivers.
DeviceId	0xae34	Intel Virtual device Device ID used for building PnP address for matching Windows SW driver to device.
RevisionId	X	OEM Platform Revision used for building PnP address for matching SW driver to device.
SubsystemId	X	OEM Platform Subsystem ID used for building PnP address for matching SW driver to device.
DeviceType	4	Device type is Analog Codec
Direction	0	Playback / Render
VirtualBusId	1	SSP1, Headset Codec is connected to SSP1

Table 3-3-9. SPECIFIC_CONFIG (Headset - Playback)

Fields	Value	Description
CapabilitiesSize	2	Size of the capabilities
VirtualSlot	0	Virtual slot ID as explained in section 2.3 In this case the codec headset receives data from Virtual slot 0, which corresponds to Link Slot 0 and 1.
ConfigType	0	Basic configuration

Table 3-3-10. FORMATS_CONFIG (Headset - Playback)

Table

Fields	Value	Description
FormatConfigCount	1	Only one configuration for the codec.
FORMAT_CONFIG[0]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure	48Khz, 24bit, 2 channels in I2S mode.

Table 3-3-11. ENDPOINT_DESCRIPTOR [3] (Headset - Capture)

Fields	Value	Description
EndpointDescriptorLength	X	Size of entire endpoint descriptor, this includes the size of the “EndpointDescriptorLength” field.
LinkType	3	SSP
InstanceId	0	Always 0.
VendorId	0x8086	Intel Vendor ID used for building PnP address for matching SW driver to device. This is mainly used for loading Windows miniport drivers.
DeviceId	0xae34	Intel Virtual device Device ID used for building PnP address for matching Windows SW driver to device.
RevisionId	X	OEM Platform Revision used for building PnP address for matching SW driver to device.
SubsystemId	X	OEM Platform Subsystem ID used for building PnP address for matching SW driver to device.
DeviceType	4	Device type is Analog Codec
Direction	1	Capture
VirtualBusId	1	SSP1, headset Codec is connected to SSP1

Table 3-3-12. SPECIFIC_CONFIG (Headset - Capture)

Fields	Value	Description
CapabilitiesSize	2	Size of the capabilities
VirtualSlot	0	Virtual slot ID as explained in section 2.3 In this case the codec headset receives data from Virtual slot 0, which corresponds to Link Slot 0 and 1.
ConfigType	0	Basic configuration

Table 3-3-13: FORMATS_CONFIG (Headset - Capture)

Fields	Value	Description
FormatConfigCount	1	Only one configuration for the codec.
FORMAT_CONFIG[0]	Please refer to structure FORMAT_CONFIG and fill the parameters as defined in the structure.	48Khz, 24bit, 2 channels in I2S mode.

§ §