

TLV320AIC31xx Audio Driver Datasheet

ABSTRACT

The TLV320AIC31xx Linux Audio driver has been developed with an I2C control interface and I2S audio streaming. The code was tested on an OMAP4 EVM, running on Linux 2.6.35 kernel. This document discusses the CODEC ASoC driver, including the hardware connection between the OMAP4 EVM and TLV320AIC3110 EVM, and the respective installations.

Table of contents

1 Introduction.....	3
2 Connections.....	3
3 Driver features	5
4 Device Driver	6
5 Installation	8
5.1 Pre-Requisites	9
5.2 Setup	9
5.3 Driver Build.....	9
5.4 Driver Build Flags	10
6 Testing.....	12
6.1 General Driver Testing	12
6.1.1 Playback related controls	12
6.1.2 Recording function	13
6.1.3 Program Codec Register.....	14
6.1.4 Display Mixer control	15
7 Testing on other platforms.....	17
8 Tested Features	19
9 Additional Information	19
10 Known Caveats and limitations	20

1 Introduction

The TLV320AIC3110 is a low-power, highly integrated, high-performance codec which supports stereo audio DAC, and mono audio ADC.

The TLV320AIC3110 features a high-performance audio codec with 24-bit stereo playback and monaural record functionality. The device integrates several analog features, such as a microphone interface, headphone drivers, and speaker drivers.

The digital audio data format is programmable to work with popular audio standard protocols (I2S, left/right-justified) in master, slave, DSP, and TDM modes. Bass boost, treble, or EQ can be supported by the programmable digital-signal processing blocks (PRB). An on chip PLL provides the high-speed clock needed by the digital-signal processing block

This application report discusses the driver for the AIC31xx, codec that was developed to enable users to quickly set up, run, and use the codec device with the Linux OS. The AIC31xx driver was coded to support the latest ASoC architecture to make to port into different host processors.

The CODEC driver described in this document was run and tested on TLV320AIC3110 EVM and a customer board containing the OMAP4430 ES 2.0

2 Connections

The AIC3110 device must be wired and connected to a host processor, where the device driver code is ported and executed. The two buses/ports for AIC3110 operation are control bus and audio data bus. The control bus on the AIC3110 is an I2C bus. The audio data streams through the data bus in I2S/DSP/Left Justified/Right Justified formats on the AIC3110.

In developing the AIC3110 driver for this application, the custom board based on OMAP4430 is used.

- On the I2C-controlled AIC3110, the seven digital signals that are essential for running the audio driver are:
- The I2C bus, two wires: SCL and SDA
- The main audio codec clock: MCLK
- The data bus, three wires: BCLK, WCLK, DIN, DOUT
- The GPIO interrupt wire.

The wiring diagram in Figure 1 describes the wiring details between the OMAP4430 and AIC3110.

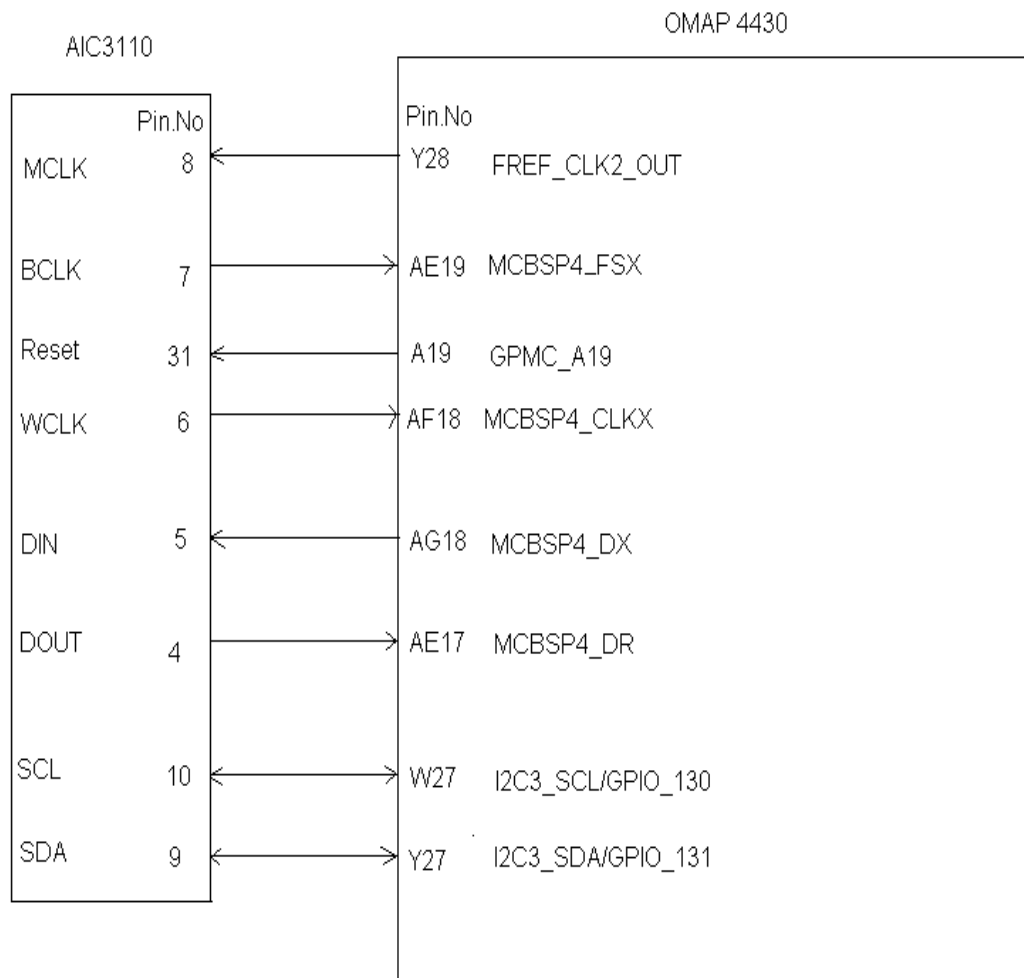


Figure 1: TLV320AIC3110 connection with OMAP4430

As shown in the diagram above, the Reset signal was driven through a custom Signal coming from the OMAP4430 and the I2C3 Controller Signals were used to interface with the AIC3110 Audio Codec Chipset.

The Master Clock Input was driven from the FREF_CLK2_OUT signal from the OMAP4430 and was configured to provide a 19.2 Mhz Clock Input to the Audio Codec Chipset.

3 Driver features

This section provides the list of the features supported by the AIC31xx Linux Audio Driver. The driver code-base supports AIC3100 and AIC3110 Audio Codec chipsets.

Feature	Description	Remarks
Sampling Rates	Recording: 8000Hz, 11025Hz, 16000Hz, 22050Hz, 32000Hz, 44100Hz, 48000Hz, 96000Hz and 192000Hz	N/A.
Audio Formats for Playback	Recording: Mono- 16 Bit Playback: Stereo-16 bit	Linux Supports 16-bit only.
File formats supported for Recording	Waveform Audio (.wav)	N/A
Playback Volume Control	Controlled using DAC Volume control	amixer command is used to exercise this feature.
Record Volume Control	Controlled using ADC Volume Control	amixer command is used to exercise this feature
AGC Control	Audio Gain Control for Delay, Noise, Enable/Disable, Attack Time, Hysteresis supported for both left and Right Channels	amixer command is used to exercise this feature.
Input Source Selection	Only MIC1LM is used	Only MIC1LM is used/configured on the custom board
MICBIAS Control	MICBIAS Enable/Disable Control and Generation supported	amixer command is used to exercise this feature.
Master/Slave Modes	Support for building the Codec as Master on the I2S Interface	
Dynamic Range Compression	Support for enable/disable, Hysteresis, Threshold, Attack, Delay, Hold Time configurations	amixer command is used to exercise this feature.
Output Routing	Support for routing DAC outputs to either Headphones or Speakers	
MUTE Control	Supports MUTE/UNMUTE Options for both DAC and ADCs	amixer command is used to exercise this feature.
Process block selection	Supports selection of DAC process block and ADC process block.	amixer command is used to exercise this feature.
Beep Generation	Supports configuration of the BEEP parameters such as Cos, Length, Sin, Master Volume Control	amixer command is used to exercise this feature.

4 Device Driver

The Advanced Linux Sound Architecture provides audio and MIDI functionality to the Linux Operating System. ALSA only defined the API's in the ASoC code layer but left the design of platform and codec driver to the platform developers. Some of the limitations were.

- Codec drivers are tightly coupled to the SoC. This lead to code duplication.
- The drivers tend to power up the entire codec when playing (or recording audio). This may be expensive on embedded platforms.

ASoC was developed to provide a better ALSA support for embedded system on chip processors and portable audio codec. The following are the advantages of ASoC architecture.

1. Codec independence. Allows reuse of codec drivers on other platforms and machines.
2. Easy I2S/PCM audio interface setup between codec and SoC. Each SoC interface and codec registers it's audio interface capabilities with the core and are subsequently matched and configured when the application hw params are known.
3. Dynamic Audio Power Management (DAPM). DAPM automatically sets the codec to it's minimum power state at all times. This includes powering up/down internal power blocks depending on the internal codec audio routing and any active streams
4. Pop and click reduction. Pops and clicks can be reduced by powering the codec up/down in the correct sequence (including using digital mute). ASoC signals the codec when to change power states

To achieve all this, ASoC basically splits an embedded audio system into 3 components

- Codec driver: The codec driver is platform independent and contains audio controls, audio interface capabilities, codec dapm definition and codec IO functions.
- Platform driver: The platform driver contains the audio dma engine and audio interface drivers (e.g. I2S, AC97, PCM) for that platform.
- Machine driver: The machine driver handles any machine specific controls and audio events. i.e. turning on an amp at start of playback

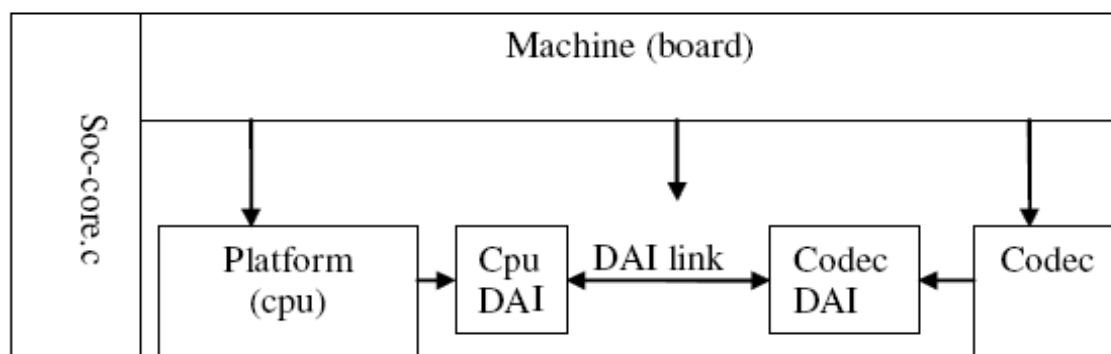


Figure 2: Linux ASoc Framework

The codec driver for AIC31xx is present under "linux-2.6.35/sound/soc/codecs" path in the Linux Kernel.

The OMAP4 ASoC Platform driver is present under "linux-2.6.35/sound/soc/omap" This CODEC driver has dependency on I2C driver. Linux 2.6.35 Kernel has the adapter driver for OMAP4 I2C controller. The codec driver registers the client driver with the I2C core for communication with AIC3110. Platform Driver has dependency on McBSP driver. Linux 2.6.35 has driver support for McBSP.

From a hardware standpoint, the AIC31xx audio driver must have both I2C and data buses (for audio control and audio data streaming, respectively). The I2C bus controls the audio codec operation by writing to the AIC31xx audio control registers; the McBSP interface transfers audio data between the host and the AIC3110. Additionally, the AIC3110 MCLK pin should receive an external clock for ADC and DAC to operate.

On the host side, the McBSP4 pins are connected to the AIC3110's WCLK, BCLK, DIN and DOUT pins. The FREF_CLK2_OUT pin is connected to MCLK, which is programmed to generate a 19.2 MHz clock.

I2C Driver

I2C client driver for AIC3110 is written for performing write to AIC3110. In the codec driver, "platform_driver_register (&tlv320aic31xx_i2c_driver);" will in turn call "AIC31xx_codec_probe". This function will probe for AIC31xx and if present will register the i2c client through "i2c_attach_client" function call.

MCBSP DRIVER

The McBSP driver for omap4 is present under "linux/sound/soc/omap/" path in the Linux tree. For testing on omap4, AIC3110 is wired with McBSP4 interface. In this codec driver McBSP 4 is configured as slave and is programmed to operate in I2S mode.

DMA DRIVER

McBSP driver utilizes DMA interface for transferring the data from the data buffer to McBSP interface. DMA driver is available under "linux/arch/arm/plat-omap/" path in the Linux tree. McBSP uses two logical channels (which ever are available) and utilizes chaining feature for transferring continuous transfers.

CODEC DRIVER

AIC31xx ASoC Codec driver consists of two files tlv320aic31xx.c and tlv320aic31xx.h available under "linux/sound/soc/codecs" path in the Linux tree. The tlv320aic31xx.c file contains the driver code for initializing the codec, controls for changing the sampling frequency, playback volume; mute/unmute etc. tlv320aic31xx.h file contains the macros and data structure definitions.

```
static struct snd_soc_dai_ops tlv320aic31xx_dai_ops = {
    .hw_params = aic31xx_hw_params,
    .digital_mute = aic31xx_mute,
    .set_sysclk = aic31xx_set_dai_sysclk,
    .set_fmt = aic31xx_set_dai_fmt,
};

struct snd_soc_dai_driver tlv320aic31xx_dai[] = {
{
    .name = "tlv320aic3110",
    .playback = {
        .stream_name = "Playback",
        .channels_min = 1,
        .channels_max = 2,
        .rates = AIC31XX_RATES,
        .formats = AIC31XX_FORMATS,
    },
    .capture = {
        .stream_name = "Capture",
        .channels_min = 1,
        .channels_max = 2,
        .rates = AIC31XX_RATES,
        .formats = AIC31XX_FORMATS,
    },
    .ops = &tlv320aic31xx_dai_ops,
},
};
```

The above data structure informs the capabilities of AIC3110 to ASoC core layer. The members .channels_min and .channels_max specify the number of channels supported. AIC31xx_RATES macro indicates the sampling frequencies supported by the driver. AIC31xx_FORMATS macro indicates the data format supported by the driver. dai_ops contains callback functions for mute/unmute, setting the sampling frequency and data formats.

5 Installation

This section presents the installation steps for running the AIC31xx Linux Audio Driver on Linux.

5.1 Pre-Requisites

- Please follow the setup provided in L27.INC1.12.1_OMAP4_GingerBread_ES2_Release Notes
- Custom board (Which contains both OMAP4 and TLV320AIC3110)

The link to the above release is given below:

http://omappedia.org/wiki/L27.INC1.12.1_OMAP4_GingerBread_ES2_Release_Notes

Also the user is advised to install the ARM cross-compiler as specified in the above link. This is used for building the updated Linux OS Image.

5.2 Setup

- Untar the "TLV320AIC31xx_Driver_Release_1_0.tar" provided along with this release into the local development PC.
- Copy the contents of the Codec_driver Folder into the sound/soc/codecs folder of the Linux Kernel Tree.
- Copy the contents of the Machine_driver folder into the sound/soc/omap folder of the Linux Kernel Tree.
- Modify the makefile and kconfig files situated under the sound/soc/codecs folder of the Linux Kernel Tree.
- Modify the makefile and the kconfig files situated under the sound/soc/omap folder of the Linux Kernel Tree.
- Run command "make kc1_defconfig"
- Build Linux image by giving command "make kernel"
- Boot the newly build kernel and file system on KC1 board using SD-Card.

IMPORTANT NOTE: The command "make kc1_defconfig" and "make kernel" are specific to the custom board. For generic OMAP4 based build procedures, the user is advised to refer to the L27.INC.1.12.1 OMAP4 Release Notes.

5.3 Driver Build

- Modify the makefile and kconfig files situated under the sound/soc/codecs folder of the Linux Kernel Tree.
- Modify the makefile and the kconfig files situated under the sound/soc/omap folder of the Linux Kernel Tree.
- Run command "make kc1_defconfig"
- Build Linux image by giving command "make kernel"
- Boot the newly build kernel and file system on custom board using SD-Card.

5.4 Driver Build Flags

This section describes the build related flags and macros provided in the AIC31xx Audio Driver sources. Before doing the compilation the build flags need to be verified in-order to ensure the inclusion/ exclusion of various functionalities while building.

The following build flags are present under the tlv320aic31xx.c source file.

These are local compile flags used to enable/disable ADC and DAC specific portions of the code-base. In the default operating condition, we need both of these below flags to be defined.

The below flag controls the powering up and down of the ADC section from the Codec Driver.

- `#define ADC_EN`
`#undef ADC_EN`

The below flag controls the powering up and down of the DAC Section from the Codec Driver.

- `#define DAC_EN`
`#undef DAC_EN`

The following flags are used to enable/disable the DAPM support for the driver. This can be undefined as it is not needed in this release.

- `#define DRIVER_DAPM_SUPPORT`
`#undef DRIVER_DAPM_SUPPORT`

The following build flags are present under the tlv320aic31xx.h header file.

The below mentioned flag is to enable the debug flags listed immediate next to this build flag.

- `#define AIC31x_CODEC_DEBUG 1`

The below mentioned flags will replace the DBG() statements with printk(). These flags are used for debug purpose.

- `#ifdef AIC31x_CODEC_DEBUG`
`#define DBG(x...) printk(KERN_INFO x)`
`#else`
`#define DBG(x...)`
`#endif`

This below macro for GPIO is specific to the Custom Board. This will replace the macro with the value (PIN No) 49 wherever referred.

- `#define OMAP4_HEADSET_MIC_DETECT_GPIO 49`

The following flags will replace the AUDIO_NAME macro with aic3110 and AIC3110_VERSION macro with 0.1 wherever it has been referred.

- `#define AUDIO_NAME "aic3110"`
`#define AIC3110_VERSION "0.1"`

The following build flags are defined under omap4_panda_aic31xx.c file

The following flag is used to enable/disable the DAPM support for the driver

- `#define DRIVER_DAPM_SUPPORT`
`#undef DRIVER_DAPM_SUPPORT`

6 Testing

The Linux Audio Driver is tested using `alsa_play` and `alsa_mixer` applications. The driver is tested only with CODEC as master, with I2S protocol and the audio data type as stereo, 16 bits data format.

6.1 General Driver Testing

The Linux audio driver testing is done through playback and record testing. Following are the procedures to test the playback and record on the AIC31xx Codec Driver

Playback

- Plugin the headphone to HEADSET jack on KC1 EVM board
- Run command "`alsa_play xyz.wav`"

6.1.1 Playback related controls

In order to get the audio at output the following controls should be enabled.

- Playback Volume

This playback volume needs to be at reasonable value in order to hear the audio output coming out through the dedicated output paths.

```
$ alsa_mixer cset numid=1 100
numid=1,iface=MIXER,name='DAC Playback Volume'
; type=INTEGER,access=rw-----,values=2,min=0,max=175,step=0
: values=100,100
```

- Headphone controls

If the user wants the playback through headphones,

```
$ alsa_mixer cset numid=12 5
numid=12,iface=MIXER,name='HP driver Volume(0 = 0 dB, 9 = 9 dB)'
; type=INTEGER,access=rw-----,values=2,min=0,max=9,step=0
: values=5,5

$ alsa_mixer cset numid=58 100
numid=58,iface=MIXER,name='HP Analog Gain Volume(0 = 0 dB, 127 = -78.3)'
; type=INTEGER,access=rw-----,values=2,min=0,max=127,step=0
: values=100,100
```

Make sure that the Headphone driver is un-muted

```
$ alsa_mixer cset numid=7 1
numid=7,iface=MIXER,name='HP driver mute'
; type=BOOLEAN,access=rw-----,values=2
```

```
: values=on,on
```

The following amixer control gives the flexibility to the user to enable or disable the headset detection.

```
$ alsactl cset numid=40 1
numid=40,iface=MIXER,name='Headset detection Enable / Disable'
; type=ENUMERATED,access=rw-----,values=1,items=2
; Item #0 'Disabled'
; Item #1 'Enabled'
: values=1
```

▪ Speaker controls

If the user wants the playback through speakers then the following controls are preferred.

Make sure that the speaker driver is in un-muted state.

```
$ alsactl cset numid=8 0
numid=8,iface=MIXER,name='SP driver mute'
; type=BOOLEAN,access=rw-----,values=2
: values=off,off
```

The speaker driver volume should be set to value which gives the audible output.

```
$ alsactl cset numid=57 3
numid=57,iface=MIXER,name='SP Class - D driver Volume(0 = 6 dB, 4 = 24)'
; type=INTEGER,access=rw-----,values=2,min=0,max=4,step=0
: values=3,3
```

```
$ alsactl cset numid=59 100
numid=59,iface=MIXER,name='SP Analog Gain Volume(0 = 0 dB, 127 = -78.3)'
; type=INTEGER,access=rw-----,values=2,min=0,max=127,step=0
: values=100,100
```

6.1.2 Recording function

The recording to work, the following controls are to be used.

First make sure that the ADC is not muted.

```
$ alsactl cset numid=56
numid=56,iface=MIXER,name='ADC Mute'
; type=ENUMERATED,access=rw-----,values=1,items=2
; Item #0 'Unmute'
; Item #1 'Mute'
: values=0
```

The volume level controls of ADC are provided through the following two controls. Set the gain values properly before doing the record.

```
$ alsa_amixer cset numid=9 4
numid=9,iface=MIXER,name='ADC FINE GAIN'
; type=INTEGER,access=rw-----,values=1,min=0,max=4,step=0
: values=4

$ alsa_amixer cset numid=10 100
numid=10,iface=MIXER,name='ADC COARSE GAIN'
; type=INTEGER,access=rw-----,values=1,min=0,max=64,step=0
: values=64
```

Keep the MICPGA gain to a desired value so that the input signal will be gained as expected

```
$ alsa_amixer cset numid=11 89
numid=11,iface=MIXER,name='ADC MIC_PGA GAIN'
; type=INTEGER,access=rw-----,values=1,min=0,max=119,step=0
: values=89
```

The following control is to set the MIC PGA gain as 0db/controlled by corresponding register values. Before doing record make sure that this control is set with the value 0.

```
$ alsa_amixer cset numid=52 0
numid=52,iface=MIXER,name='MIC PGA Setting'
; type=ENUMERATED,access=rw-----,values=1,items=2
; Item #0 'Gain controlled by D0 - D6'
; Item #1 '0 db Gain'
: values=0
```

Record

- Plug in the headphone to HEADSET jack on AIC3110 EVM board.
- Speak on the mic to record data.
- Run the following arecord command.

Ex:

```
# arecord -c 1 -t wav -f S16_LE -r 8000 -d 20 rec.wav
```

6.1.3 Program Codec Register

The program register control usage is different than other controls.

Here the value passed after numid will be converted into hexadecimal, and that hexadecimal value will be parsed as two parts. The upper byte will represent the register address and the lower byte will represent the value to be written on the

register. You can verify the written value on the particular register by using cget command as usual.

```
$ alsa_amixer cset numid=60 0
reg = 0 val = 0
numid=60,iface=MIXER,name='Program Registers'
; type=INTEGER,access=rw-----,values=1,min=0,max=65535,step=0
: values=0
```

6.1.4 Display Mixer control

Usage: amixer controls

```
$ alsa_amixer controls
numid=24,iface=MIXER,name='Mic Bias Voltage'
numid=10,iface=MIXER,name='ADC COARSE GAIN'
numid=9,iface=MIXER,name='ADC FINE GAIN'
numid=11,iface=MIXER,name='ADC MIC_PGA GAIN'
numid=56,iface=MIXER,name='ADC Mute'
numid=26,iface=MIXER,name='ADC Processing Block Selection(0 <->25)'
numid=31,iface=MIXER,name='AGC Attack Time control'
numid=33,iface=MIXER,name='AGC Decay Time control'
numid=30,iface=MIXER,name='AGC Maximum PGA Control'
numid=37,iface=MIXER,name='AGC Noice bounce control'
numid=38,iface=MIXER,name='AGC Signal bounce control'
numid=29,iface=MIXER,name='AGC Target Level Control'
numid=32,iface=MIXER,name='AGC_ATC_TIME_MULTIPLIER'
numid=34,iface=MIXER,name='AGC_DECAY_TIME_MULTIPLIER'
numid=35,iface=MIXER,name='AGC_HYSTERISIS'
numid=36,iface=MIXER,name='AGC_NOISE_THRESHOLD'
numid=28,iface=MIXER,name='Audio Gain Control(AGC)'
numid=23,iface=MIXER,name='Beep Cos(x) LSB'
numid=22,iface=MIXER,name='Beep Cos(x) MSB'
numid=19,iface=MIXER,name='Beep Length LSB'
numid=18,iface=MIXER,name='Beep Length MID'
numid=17,iface=MIXER,name='Beep Length MSB'
numid=21,iface=MIXER,name='Beep Sin(x) LSB'
numid=20,iface=MIXER,name='Beep Sin(x) MSB'
numid=15,iface=MIXER,name='Beep generator Enable / Disable'
```

numid=16,iface=MIXER,name='Beep generator Volume Control(0 = -61 dbdB,'
numid=50,iface=MIXER,name='CM selection for ADC IP M - terminal'
numid=1,iface=MIXER,name='DAC Playback Volume'
numid=25,iface=MIXER,name='DAC Processing Block Selection(0 <->25)'
numid=6,iface=MIXER,name='DAC Volume soft stepping'
numid=5,iface=MIXER,name='DAC volume Control register/pin'
numid=4,iface=MIXER,name='DAC volume Extra control'
numid=41,iface=MIXER,name='DRC Enable / Disable'
numid=43,iface=MIXER,name='DRC Hysteresis value(0 = 0 db, 3 = 3 db)'
numid=42,iface=MIXER,name='DRC Threshold value(0 = -3 db, 7 = -24 db)'
numid=45,iface=MIXER,name='DRC attack rate'
numid=46,iface=MIXER,name='DRC decay rate'
numid=44,iface=MIXER,name='DRC hold time'
numid=58,iface=MIXER,name='HP Analog Gain Volume(0 = 0 dB, 127 = -78.3'
numid=39,iface=MIXER,name='HP Output common - mode voltage control'
numid=12,iface=MIXER,name='HP driver Volume(0 = 0 dB, 9 = 9 dB)'
numid=7,iface=MIXER,name='HP driver mute'
numid=40,iface=MIXER,name='Headset detection Enable / Disable'
numid=2,iface=MIXER,name='Left DAC Mute'
numid=13,iface=MIXER,name='Left DAC input selection'
numid=52,iface=MIXER,name='MIC PGA Setting'
numid=55,iface=MIXER,name='MIC1LM CM Setting'
numid=51,iface=MIXER,name='MIC1LM selection for ADC I/P M - terminal'
numid=49,iface=MIXER,name='MIC1LM selection for ADC I/P P - terminal'
numid=53,iface=MIXER,name='MIC1LP CM Setting'
numid=47,iface=MIXER,name='MIC1LP selection for ADC I/P P - terminal'
numid=54,iface=MIXER,name='MIC1RP CM Setting'
numid=48,iface=MIXER,name='MIC1RP selection for ADC I/P P - terminal'
numid=60,iface=MIXER,name='Program Registers'
numid=3,iface=MIXER,name='Right DAC Mute'
numid=14,iface=MIXER,name='Right DAC input selection'
numid=59,iface=MIXER,name='SP Analog Gain Volume(0 = 0 dB, 127 = -78.3'
numid=57,iface=MIXER,name='SP Class - D driver Volume(0 = 6 dB, 4 = 24'
numid=8,iface=MIXER,name='SP driver mute'
numid=27,iface=MIXER,name='Throughput of 7 - bit vol ADC for pin'

7 Testing on other platforms

The following steps are useful for testing AIC31xx driver on other platform and/or different kernel version. The steps can be used for testing the TLV320AIC31xx on any Hardware platform having ASoC platform, machine support and I2C driver.

- a. Copy the source files (tlv320aic3110.c tlv320aic31xx.h) under "linux/sound/soc/codecs".
- b. Add the below source to the end of "linux/sound/soc/codecs/Kconfig" file
config SND_SOC_TLV320AIC3110
tristate "TI TLV320AIC3110 Codec support"
- c. Add the below source to the end of "linux/sound/soc/codecs/Makefile" file
snd-soc-tlv320aic3110-objs := tlv320aic3110.o
obj-\$(CONFIG_SND_SOC_TLV320AIC3110) += snd-soc-tlv320aic3110.o
- d. Modify the Machine driver to select TLC320AIC3110 codec. In the dai_link structure of the machine driver modify the .name and .codec_dai_name members to "tlv320aic3110". The .codec_name member should be similar to .name member.
- e. Modify the Clock Configuration as per the Master Clock Settings for a different board.
- f. Modify the Headset Detection and Reset related GPIO Pins for a different board inside the machine driver.
- g. Update Kconfig and Makefile for the platform driver to use AIC3110 Codec
- f. Build the kernel

NOTE:

The interface between the platform driver and codec driver is through "struct snd_soc_codec_dai". The platform driver calls the codec functions through function pointers hw_params, digital_mute, set_sysclk, set_fmt.

```
static struct snd_soc_dai_ops tlv320aic31xx_dai_ops = {  
    .hw_params = aic31xx_hw_params,  
    .digital_mute = aic31xx_mute,  
    .set_sysclk = aic31xx_set_dai_sysclk,  
    .set_fmt = aic31xx_set_dai_fmt,  
};
```

```
struct snd_soc_dai_driver tlv320aic31xx_dai[] = {
{
    .name = "tlv320aic3110",
    .playback = {
        .stream_name = "Playback",
        .channels_min = 1,
        .channels_max = 2,
        .rates = AIC31XX_RATES,
        .formats = AIC31XX_FORMATS,
    },
    .capture = {
        .stream_name = "Capture",
        .channels_min = 1,
        .channels_max = 2,
        .rates = AIC31XX_RATES,
        .formats = AIC31XX_FORMATS,
    },
    .ops = &tlv320aic31xx_dai_ops,
},
};
```

The Linux Kernel can have many codec drivers and all the codec drivers export object of type "struct snd_soc_dai_driver"

The programmer should change the machine driver depending on his platform/hardware to choose among many codec drivers. Below is code from machine driver for OMAP4 using AIC31xx codec driver

```
static struct snd_soc_dai_link omap4_dai[] = {
{
    .name = "tlv320aic3110",
    .stream_name = "AIC31XX Audio",
    .cpu_dai_name = "omap-mcbsp-dai.3",
    .codec_dai_name = "tlv320aic3110",
    .platform_name = "omap-pcm-audio",
    .codec_name = "tlv320aic3110-codec",
    .init = omap4_aic31xx_init,
    .ops = &omap4_ops,
},
};

/* Audio machine driver */
static struct snd_soc_card snd_soc_omap4_panda = {
    .name = "OMAP4_KC1",
    .long_name = "OMAP4_KC1_AIC3110",
    .dai_link = omap4_dai,
    .num_links = 1,
};

static struct platform_device *omap4_snd_device;
```

8 Tested Features

The tested features of the driver with respect to AIC31xx:

- 1) Playback is tested.
- 2) Recording is tested.
- 3) Playback via headphone
- 4) Playback via loudspeakers

9 Additional Information

The Linux Audio Driver has support for following MCLK frequency 19.2MHz. On omap4 platform, the MCLK frequency configured was 19.2 MHz and tested.

For supporting new MCLK frequencies, `aic31xx_divs[]` array in `tlv320aic31xx.c` needs to be updated. For that we need to calculate the value of K (ie J.D), P and R using the below formula

$$\text{MCLK} = 19.2 \text{ MHz}$$

$$\text{PLL_CLK_IN} = \text{MCLK} = 19.2 \text{ MHZ}$$

$$\text{PLL_CLK} / \text{DAC_Fs} = \text{MDAC} * \text{NDAC} * \text{DOSR};$$

(PLL_CLK is 98304000 for 8 KHz)

For DAC_FS = 8000 Hz,

$$98304000 / 8000 = 12288;$$

$$\text{Therefore MDAC} * \text{NDAC} * \text{DOSR} = 12288;$$

Consider NDAC = 24, MDAC = 2 DOSR = 256;

Similarly NADC = 48, MADC = 2, AOSR = 128 is maintained.

$$\text{PLL_CLK} = K * \text{MCLK}, \text{ where MCLK} = 19.2 \text{ MHz}$$

$$\mathbf{Fs = 8 KHz, \quad MCLK = 19.2 MHZ} \quad (Fs = \text{DAC_Fs})$$

$$\text{So, } K = \text{PLL_CLK} / \text{MCLK}$$

$$K = 98304000 / 19200000;$$

K = 5.12

J = 5, D = 1200

Similarly for all the sampling frequencies given below, the required PLL parameters are calculated.

The following table shows the PLL Clock Configuration Table as used by the Linux Audio Driver sources.

```
mclk, rate, p_val, pll_j, pll_d, dosr, ndac, mdac, aosr, nadc, madc,  
blk_N,  
  
    /* 8k rate */  
    {19200000, 8000, 1, 5, 1200, 256, 24, 2, 128, 48, 2, 8},  
    {19200000, 8000, 1, 5, 1200, 256, 24, 2, 0, 24, 2, 8},  
    /* 11.025k rate */  
    {19200000, 11025, 1, 4, 4100, 256, 15, 2, 128, 30, 2, 8},  
    {19200000, 11025, 1, 4, 4100, 256, 15, 2, 0, 15, 2, 8},  
    /* 12K rate */  
    {19200000, 12000, 1, 4, 8000, 256, 15, 2, 128, 30, 2, 8},  
    {19200000, 12000, 1, 4, 8000, 256, 15, 2, 0, 15, 2, 8},  
    /* 16k rate */  
    {19200000, 16000, 1, 5, 1200, 256, 12, 2, 128, 24, 2, 8},  
    {19200000, 16000, 1, 5, 1200, 256, 12, 2, 0, 12, 2, 8},  
    /* 22.05k rate */  
    {19200000, 22050, 1, 4, 7040, 256, 8, 2, 128, 16, 2, 8},  
    {19200000, 22050, 1, 4, 7040, 256, 8, 2, 0, 8, 2, 8},  
    /* 24k rate */  
    {19200000, 24000, 1, 5, 1200, 256, 8, 2, 128, 16, 2, 8},  
    {19200000, 24000, 1, 5, 1200, 256, 8, 2, 0, 8, 2, 8},  
    /* 32k rate */  
    {19200000, 32000, 1, 5, 1200, 256, 6, 2, 128, 12, 2, 8},  
    {19200000, 32000, 1, 5, 1200, 256, 6, 2, 0, 6, 2, 8},  
    /* 44.1k rate */  
    {19200000, 44100, 1, 4, 7040, 256, 4, 2, 128, 8, 2, 8},  
    {19200000, 44100, 1, 4, 7040, 256, 4, 2, 0, 4, 2, 8},  
    /* 48k rate */  
    {19200000, 48000, 1, 5, 1200, 256, 4, 2, 128, 8, 2, 8},  
    {19200000, 48000, 1, 5, 1200, 256, 4, 2, 0, 4, 2, 8},  
    /* 96k rate */  
    {19200000, 96000, 1, 5, 1200, 256, 2, 2, 128, 4, 2, 8},  
    {19200000, 96000, 1, 5, 1200, 256, 2, 2, 0, 2, 2, 8},  
    /* 192k */  
    {19200000, 192000, 1, 5, 1200, 256, 2, 1, 128, 4, 1, 16},  
    {19200000, 192000, 1, 5, 1200, 256, 2, 1, 0, 2, 1, 16},  
};
```

10 Known Caveats and limitations

- The required mixer controls for Automatic Gain Control are provided, but the AGC impact on the audio output is not tested.
- Similarly for Beep Generation the controls are provided but the beep generation is not yet tested.
- Even though the AIC31xx audio driver supports all the sampling rates (8 KHz to 192 KHz), due to android kernel limitations the frequencies 11025, 22050, 44100 are not supported.