

TLV320ADC3101 Linux Audio Driver

ABSTRACT

The TLV320ADC3101 audio driver has been developed with an I2C control interface and I2S audio streaming. The code was tested on an Revision C OMAP3530 Evaluation Module running on Linux 2.6.28 kernel. This data sheet discusses the CODEC ASoC driver, including the hardware connections between the ADC3101 EVM and the Mistral OMAP3530 EVM, and the respective installation.

Table of contents

1	Introduction.....	3
2	Connections.....	3
3	Device Driver.....	4
3.1	I2C Driver.....	5
3.2	MCBSP DRIVER.....	5
3.3	DMA DRIVER.....	5
3.4	CODEC DRIVER	5
3.5	MINI DSP.....	6
4	Features Supported	7
5	Driver Sources	8
6	Driver Installation.....	9
6.1	Pre-Requisites	10
6.2	Setup	10
6.3	Record	11
6.4	Mixer Controls	11
7	Driver Build	16
8	Driver Configuration.....	18
9	Testing	18
9.1	Pre-Requisites	18
9.2	Board Setup.....	18
10	Configuration of MINI DSP	19
10.1	Volume Lite Example:.....	19
10.2	TiLoad Utility	19
10.3	Current ADC3101 MINI DSP support	20
11	Testing on other platforms.....	21
12	Known Issues/Caveats in this release.....	21

1 Introduction

The TLV320ADC3101 (also referred to as the ADC3101) is a low-power, stereo audio codec supporting sampling rates from 8 kHz to 96 kHz with an integrated programmable-gain amplifier providing up to 40-dB analog gain or AGC, a programmable miniDSP for custom audio processing, programmable input combination of single-ended or fully differential configurations, fixed, predefined and parameterizable signal processing blocks, integrated PLL. This device is ideal for battery-powered portable equipment.

This data sheet discusses the Linux Audio Driver for the ADC3101 codec that was developed to enable users to quickly set up, run, and use the codec device with the Linux 2.6.28.

The CODEC driver described in this document was developed and tested using the TLV320ADC3101 EVM Rev A and OMAP3530 EVM Rev C.

2 Connections

The ADC3101 device must be wired and connected to a host processor, where the device driver code is ported and executed. For interfacing with the host processor, we will require two buses (or ports) namely, the control bus and the audio data bus. The control bus on the ADC3101 is the I2C serial bus. For the data bus interface, the McBSP from the OMAP35x is utilized.

In developing the ADC3101 drivers for this application, the TI ADC3101 EVM board and the OMAP3530 EVM platform with the OMAP3 processor were used.

The control interface between the OMAP3530 and the ADC3101 EVM is through the I2C Lines [SDA and SCL]. The I2C2 from OMAP3 is being used for interfacing to the ADC3101 EVM. McBSP1 of the OMAP3 is being interfaced to drive the Digital Audio interface.

In this configuration, codec is master. The Maser clock for ADC3101 is 12MHz obtained from sys_clkout2 signal on OMAP3.

On the I2C-controlled ADC3101, the following digital signals that are essential for running the audio driver are:

1. To reset the Audio Codec: RESETB
2. The I2C bus, two wires: SCL, SDA
3. The main audio codec clock: MCLK
4. The data bus, three wires: BCLK, WCLK, DOUT

The wiring diagram describes the wiring details between the OMAP3530 EVM and ADC3101 EVM.

SIGNAL ON OMAP3EVM Main Board	Pin Number on OMAP3EVM Main Board	DIR	SIGNAL ON ADC3101 EVM	PIN No. ON ADC3101 EVM
I2C2_SDA	P7.A30	↔	SDA	J5.20
I2C2_SCL	P7.A31	→	SCL	J5.16
SYS_CLKOUT2	P7.B52	→	MCLK	J5.17
MCBSP1_CLKR	P7.A16	←	BCLK	J5.3
MCBSP1_FSR	P7.A17	←	WCLK	P5.7
MCBSP1_DR	P7.A19	←	DOUT	J5.13
GPIO_158	P7.A18	→	nRESET	J4.8
GPIO_10	P7.B53	←	GPIO1	J4.2
GND	P7.B54	→	DGND	J3.5
VBAT	P7.A1	→	+5VA	J3.3
VIO_1v8	P7.A3	→	+1.8VD	J3.7
			+3.3VD	J3.9

Table 1 Connection Diagram between OMAP3530 EVM and ADC3101 EVM

3 Device Driver

The codec driver for ADC3101 is present under "sound/soc/codecs" path in the Linux Kernel.

The OMAP3 ASoC Platform driver is present under "sound/soc/omap" path in the Linux Kernel.

This CODEC driver has dependency on I2C driver. Linux 2.6.28 Kernel has the adapter driver for OMAP3 I2C controller. The codec driver registers the client driver with the I2C core for communication with ADC3101.

Platform Driver has dependency on McBSP driver. Linux 2.6.28 has driver support for McBSP.

From a hardware standpoint, the ADC3101 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 ADC3101 audio control registers; the McBSP interface transfers audio data between the host and the ADC3101. Additionally, the ADC3101 MCLK pin should receive an external clock for ADC and to operate.

On the host side, the McBSP1 pins are connected to the ADC3101's WCLK, BCLK and DOUT pins. The sys_clkout2 pin is connected to MCLK, which is programmed to generate a 12MHz clock.

3.1 I2C Driver

I2C client driver for ADC3101 is written for performing write to ADC3101. In the codec driver will call "adc3101_probe", this function will probe for ADC3101 and if present will register the i2c client through "i2c_add_driver (&adc3101_i2c_driver);" will add i2c client driver.

3.2 MCBSP DRIVER

The McBSP driver for omap3 is present under "linux/arch/arm/plat-omap/" path in the Linux tree. For testing on omap3, ADC3101 EVM is wired with McBSP1 interface. McBSP is configure as slave and is programmed to operate in I2S mode.

3.3 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.

3.4 CODEC DRIVER

ADC3101 ASoC Codec driver consists of two files tlv320adc3101.c and tlv320adc3101.h available under "linux/sound/soc/codecs" path in the Linux tree. tlv320adc3101.c file contains the driver code for initializing the codec, controls for changing the sampling frequency, volume, format etc. tlv320adc3101.h file contains the macros and data structure definitions.

```
struct snd_soc_dai adc3101_dai = {  
  
    .name = "adc3101",  
    .capture = {  
  
        .stream_name = "Capture",  
        .channels_min = 1,  
        .channels_max = 2,  
        .rates = ADC3101_RATES,  
        .formats = ADC3101_FORMATS, },  
  
    .ops = {  
  
        .hw_params = adc3101_hw_params,  
  
    },  
  
    .dai_ops = {
```

```
.set_sysclk = adc3101_set_dai_sysclk,  
.set_fmt = adc3101_set_dai_fmt,  
  
}  
};
```

The above data structure informs the capabilities of ADC3101 to ASoC core layer. *channels_min* and *channels_max* specify the number of channels supported. ADC3101_RATES macro indicates the sampling frequencies supported by the driver. ADC3101_FORMATS macro indicates the data format supported by the driver. ops contains callback functions for setting the sampling frequency and data formats.

3.5 MINI DSP

ADC3101 CODEC contains miniDSP core, coupled to ADC. The software development is supported through TI's PurePath Studio Development Environment, whose output is a configuration file. The configuration file instructs which register to program with what value. This information will be with respect to initialization of codec, instructions for miniDSP and co-efficient values for the program/algorithm etc.

The configuration file comes in two variants.

- Standard .cfg format. This format is to be used ADC3101 Control Software program.
- C header format.

The Linux driver supports the configuration file in the first & second format. During the driver build process, the driver will include the configuration file. At run time the driver will program the codec registers according to the information in configuration file.

Users of the Linux driver need to do modification in tlv320adc3101_mini-dsp.h, so as to include their configuration file and exercise the codec to their needs.

Note:

The miniDSP can be exercised by enabling or disabling the macro "CONFIG_MINI_DSP" in tlv320adc3101.h.

If this flag is disabled, the driver audio processing will be restricted to different processing block (PRB) comprising of different filters.

If this flag is enabled, the driver audio processing will be performed by the miniDSP and hence the processing can be customized.

4 Features Supported

This section provides the list of the features supported by the ADC3101 Audio Driver.

Feature	Description	Remarks
Sampling Rates	Recording: 8000Hz, 11025Hz, 16000Hz, 22050Hz, 32000Hz, 44100Hz, 48000Hz, 96000Hz	N/A.
Audio Formats	Recording: Mono- 8 Bit, Mono- 16 Bit, Stereo- 8 Bit and Stereo- 16 Bit	Linux Supports 16-bit only.
File formats supported for Recording	Waveform Audio (.wav)	N/A
Volume Control	Controlled using ADC Volume control ,PGA Gain control , Volume Lite control	amixer command is used to exercise this feature.
AGC Control	Audio Gain Control	amixer command is used to exercise this feature.
Input Source Selection	6 single ended inputs, 6 differential inputs, On Board and Digital Mic input	amixer command is used to exercise this feature.
Input attenuation	0db or -6 db Input attenuation	amixer command is used to exercise this feature.
miniDSP Enable or Disable	miniDSP can be enabled or disabled for sampling frequency 44100Hz	The miniDSP can be exercised by enabling or disabling the macro "CONFIG_MINI_DSP" in tlv320adc3101.h
Loading custom .cfg files	TILoad application can be used to parse and write to Codec register	This feature is exercised using TILoad Application.

5 Driver Sources

The tlv320adc3101 ASOC driver is found in the release source package **"adc3101_asoc_driver_1.0.tar.gz"** The directory structure of the source package is shown below.

adc3101_asoc_driver_1.0

```
|
|-- codec_driver
|   |-- main_Rate44_pps_driver.h
|   |-- tlv320adc3101.c
|   |-- tlv320adc3101.h
|   |-- tlv320adc3101_mini-dsp.c
|   `-- tlv320adc3101_mini-dsp.h
|-- kernel_patch
|   `-- linux-omap-2.6-adc3101-support.patch
|-- machine_driver
|   `-- omap3evm.c
|-- omap3_baseport
|   |-- mach-omap2
|   |   |-- board-omap3evm.c
|   |   |-- mcbsp.c
|   |   `-- mux.c
|   `-- plat-omap
|       |-- mcbsp.c
|       |-- mcbsp.h
|       `-- mux.h
|-- platform_driver
|   |-- omap-mcbsp.c
|   |-- omap-mcbsp.h
|   |-- omap-pcm.c
|   `-- omap-pcm.h
```


6 Driver Installation

This section explains the installation of the ADC3101 Driver package on Linux platforms. The Linux 2.6.28 kernel for OMAP3 platform is taken from the open source GIT tree.

1. Getting the Linux 2.6.28 kernel

- Clone open source linux-omap tree from kernel.org:

```
# git clone git://git.kernel.org/pub/scm/linux/kernel/git/tmlind/linux-omap-2.6.git
# cd linux-omap-2.6
```

- Create a development branch based on 2.6.28-omap1 tag
#git checkout -b adc3101 v2.6.28-omap1

2. Adding TLV320ADC3101 ASOC driver support

ADC3101 ASOC driver support can be added to Linux 2.6.28 kernel either by applying a unified kernel patch or by manually modifying the Linux kernel tree.

- **TLV320ADC3101 ASOC driver using unified kernel patch:**

```
# cat ../linux-omap-2.6-adc3101-support.patch | patch -p1 --dry-run
# cat ../linux-omap-2.6-adc3101-support.patch | patch -p1
```

- **Adding TLV320ADC3101 ASOC driver Manually**

- Extract the release source package "**adc3101_asoc_driver_1.0.tar.gz**" into an installation directory \$(DIR).
- Manually copy all the files from \$(DIR)/codec_driver/ to the "sound/soc/codecs" path in Linux tree.
- Copy all files from \$(DIR)/machine_driver/ folder to "sound/soc/omap" path in Linux tree.
- Copy all files from \$(DIR)/platform_driver/ folder to "sound/soc/omap" path in Linux tree
- Copy all files from \$(DIR)/omap3_baseport/mach-omap2/ folder to "arch/arm/mach-omap2/"
- Copy the file \$(DIR)/omap3_baseport/plat-omap/mcbbsp.c to "arch/arm/plat-omap/mcbbsp.c"
- Copy the file \$(DIR)/omap3_baseport/plat-omap/mcbbsp.h to "arch/arm/plat-omap/include/mach/mcbbsp.h"
- Copy the file \$(DIR)/omap3_baseport/plat-omap/mux.h to "arch/arm/plat-omap/include/mach/mux.h"
- Add the below source to the end of "sound/soc/codecs/Kconfig" file

```
config SND_SOC_TLV320ADC3101
tristate
depends on I2C
```

- Add the below source to the end of "sound/soc/codecs/Makefile" file

```
snd-soc-tlv320ADC3101-objs := tlv320adc3101.o tlv320adc3101_mini-  
dsp.o  
obj-$(CONFIG_SND_SOC_TLV320ADC3101) +=snd-soc-tlv320adc3101.o
```

- Add the below source to the end of "sound/soc/omap/Kconfig" file

```
config SND_OMAP_SOC_OMAP3EVM  
    tristate "SoC Audio support for OMAP3EVM (TLV320ADC3101  
codec)"  
    depends on SND_OMAP_SOC && MACH_OMAP3EVM  
    select SND_OMAP_SOC_MCBSP  
    select SND_SOC_TLV320ADC3101  
    help  
    Say Y if you want to add support for tlv320adc3101 Soc  
    Audio on omap3evm.
```

- Add the below source to the end of "sound/soc/omap/Makefile" file

```
snd-soc-omap3evm-objs := omap3evm.o  
obj-$(CONFIG_SND_OMAP_SOC_OMAP3EVM) += snd-soc-omap3evm.o
```

6.1 Pre-Requisites

The driver is tested using amixer applications. The driver is tested only CODEC as master, with I2S protocol and the audio data type as stereo, 16 bits data format

- Arm Linux tool chain version 4.2.0 available from http://www.codesourcery.com/gnu_toolchains/arm/portal/releases306. From there choose IA32 GNU/Linux Tar ball
- OMAP3530 Revision C EVM
- TLV320ADC3101 Revision A EVM

6.2 Setup

- The connections between OMAP3530EVM and TLV320ADC3101EVM are to be made according to the table above.
- Build Linux kernel image
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
omap3_evm_adc3101_defconfig
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm uImage
- Boot the kernel image arch/arm/boot/uImage on OMAP3530EVM interfaced to ADC3101-EVM hardware setup.

6.3 Record

- Connect one end of "audio loop-back" cable to host system and other end to LINE IN JACK (J9), which is on ADC3101EVM
- Play audio on host side (through media player)
- Run the following arecord command

```
[root@OMAP3530EVM ~]# arecord -c2 -f S16_LE -r 44100 -d 10
rec_44100.wav
Recording WAVE 'rec_44100.wav' : Signed 16 bit Little Endian, Rate 44100
Hz, Stereo
```

```
[root@OMAP3530EVM ~]#
```

Usage:

```
arecord -c<number of channels> -f <format> -r <sample_rate> -d
<Duration> <file_name>
```

Where	
Number of channels	2 stereo
file_type	type of the file format (wav)
Format	sample format (case insensitive). The recognized sample formats are S8 U8 S16_LE S16_BE U16_LE U16_BE S24_LE S24_BE U24_LE U24_BE S32_LE S32_BE U32_LE U32_BE FLOAT_LE FLOAT_BE FLOAT64_LE FLOAT64_BE IEC958_SUBFRAME_LE IEC958_SUBFRAME_BE MU_LAW A_LAW IMA_ADPCM MPEG GSM SPECIAL S24_3LE S24_3BE U24_3LE U24_3BE S20_3LE S20_3BE U20_3LE U20_3BE S18_3LE S18_3BE U18_3LE
sample_rate	sampling rate (8000 Hz, 11025 Hz, 22050 Hz, 32000 Hz, 44100 Hz, 48000 Hz, 96000 Hz)
Duration	Duration in seconds
File Name	File name given to recorded audio

Note: MiniDSP supports only 44100Hz sampling frequency.

6.4 Mixer Controls

Display Mixer control

Usage: amixer controls

Example:

```
root@arago:~# amixer controls
```

```
numid=38,iface=MIXER,name='Line In Jack'
numid=9,iface=MIXER,name='Mic Bias 1 Voltage'
numid=10,iface=MIXER,name='Mic Bias 2 Voltage'
numid=18,iface=MIXER,name='ADC Volume Control (0=-12dB, 64=+20dB)'
numid=11,iface=MIXER,name='ADC soft stepping'
numid=5,iface=MIXER,name='AGC Attack Time control'
```

```
numid=6,iface=MIXER,name='AGC Decay Time control'
numid=4,iface=MIXER,name='AGC Maximum PGA Control'
numid=7,iface=MIXER,name='AGC Noice bounce control'
numid=8,iface=MIXER,name='AGC Signal bounce control'
numid=3,iface=MIXER,name='AGC Target Level Control'
numid=2,iface=MIXER,name='Audio Gain Control (AGC)'
numid=40,iface=MIXER,name='Digital MIC In'
numid=19,iface=MIXER,name='Left ADC Fine Volume (0=-0.4dB, 4=0dB)'
numid=22,iface=MIXER,name='Left DMic Input Left ADC switch'
numid=33,iface=MIXER,name='Left Input Selection DIF1_L swi'
numid=34,iface=MIXER,name='Left Input Selection DIF2_L swi'
numid=35,iface=MIXER,name='Left Input Selection DIF3_L swi'
numid=30,iface=MIXER,name='Left Input Selection IN1_L swit'
numid=36,iface=MIXER,name='Left Input Selection IN1_R swit'
numid=31,iface=MIXER,name='Left Input Selection IN2_L swit'
numid=32,iface=MIXER,name='Left Input Selection IN3_L swit'
numid=12,iface=MIXER,name='Left Linein1 input attenuation'
numid=13,iface=MIXER,name='Left Linein2 input attenuation'
numid=14,iface=MIXER,name='Left Linein3 input attenuation'
numid=39,iface=MIXER,name='MIC In Jack'
numid=1,iface=MIXER,name='PGA Gain Volume Control (0=0dB, 80=40dB)'
numid=20,iface=MIXER,name='Right ADC Fine Volume (0=-0.4dB, 4=0dB)'
numid=21,iface=MIXER,name='Right DMic Input Right ADC swit'
numid=26,iface=MIXER,name='Right Input Selection DIF1_R sw'
numid=27,iface=MIXER,name='Right Input Selection DIF2_R sw'
numid=28,iface=MIXER,name='Right Input Selection DIF3_R sw'
numid=29,iface=MIXER,name='Right Input Selection IN1_L swi'
numid=23,iface=MIXER,name='Right Input Selection IN1_R swi'
numid=24,iface=MIXER,name='Right Input Selection IN2_R swi'
numid=25,iface=MIXER,name='Right Input Selection IN3_R swi'
numid=15,iface=MIXER,name='Right Linein1 input attenuation'
numid=16,iface=MIXER,name='Right Linein2 input attenuation'
numid=17,iface=MIXER,name='Right Linein3 input attenuation'
numid=37,iface=MIXER,name='Volume_1 (0=-110dB, 116=+6dB)'
```

PGA Gain Volume Control

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "PGA Gain Volume Control"

x = 0 to 80

Example:

```
amixer cset numid=1 25, 25
```

ADC Volume Control

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "ADC Volume Control"
x = 0 to 64

Example:

```
amixer cset numid=18 40,40
```

Volume Lite control

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "Volume_1 (0=-110dB, 116=+6dB)"
x = 0 to 116

Example:

```
amixer cset numid=37 105
```

Mic Bias 1 Voltage and Mic Bias 2 Voltage

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "Mic Bias 1 Voltage" or "Mic Bias 2 Voltage"
x = off, 2V, 2.5V or AVDD

Example:

```
amixer cset numid=9 off
```

ADC soft stepping

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "ADC soft stepping"
x = 1, 2 or off

Example:

```
amixer cset numid=11 1
```

Audio Gain Control (AGC)

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "Audio Gain Control (AGC)"
x = off or on

Example:
`amixer cset numid=2 off,off`

Line In Jack

Usage:
`amixer cset numid=<id1> x`

id1 is the numid of control with name "Line In Jack"
x = off or on

Example:
`amixer cset numid=38 On`

MIC In Jack

Usage:
`amixer cset numid=<id1> x`

id1 is the numid of control with name "MIC In Jack"
x = off or on

Example:
`amixer cset numid=39 On`

Digital MIC In

Usage:
`amixer cset numid=<id1> x`

id1 is the numid of control with name "Digital MIC In"
x = off or on

Example:
`amixer cset numid=40 off`

Left ADC Fine Volume

Usage:
`amixer cset numid=<id1> x`

id1 is the numid of control with name "Left ADC Fine Volume"
x = 0 to 4

Example:
`amixer cset numid=19 2`

Right ADC Fine Volume

Usage:
`amixer cset numid=<id1> x`

id1 is the numid of control with name "Right ADC Fine Volume"
x = 0 to 4

Example:

```
amixer cset numid=20 2
```

Input attenuation

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "Right Linein1 input attenuation or Left Linein1 input attenuation or Right Linein2 input attenuation or Left Linein2 input attenuation or Right Linein3 input attenuation or Left Linein3 input attenuation"

x = 0 , 1

Example:

```
amixer cset numid=15 1
```

Input Selection

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "Left Input Selection IN1_L switch or Right Input Selection IN1_R switch or Left Input Selection IN2_L switch or Right Input Selection IN2_R switch or Left Input Selection IN3_L switch or Right Input Selection IN3_R switch etc"

x = on or off

Example:

```
amixer cset numid=27 1
```

Digital Mic Input to Left/Right ADC control

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "Right DMic Input Right ADC switch or Left DMic Input Left ADC switch"

x = on or off

Example:

```
amixer cset numid=21 1
```

7 Driver Build

The following steps are useful for testing ADC3101 driver on other platform and/or different kernel version. The steps can be used is useful for testing the TLV320ADC3101 on any Hardware platform having ASoC platform, machine support and I2C driver.

Also this procedure can be used on any Linux kernel version having ASoC support.

- Copy the source files (tlv320adc3101.c, tlv320adc3101.h, tlv320adc3101_mini-dsp.c, tlv320adc3101_mini-dsp.h, main_Rate44_pps_driver.h) under "sound/soc/codecs".
- Add the below source to the end of "sound/soc/codecs/Kconfig" file

config SND_SOC_TLV320ADC3101
tristate
depends on I2C

- Add the below source to the end of "linux/sound/soc/codecs/Makefile" file

snd-soc-tlv320ADC3101-objs := tlv320adc3101.o tlv320adc3101_mini-dsp.o
obj-\$(CONFIG_SND_SOC_TLV320ADC3101) +=snd-soc-tlv320adc3101.o

- Modify the Machine driver to select TLC320ADC3101 codec. In the machine driver initialize "codec_dev" member of "struct snd_soc_device" to &soc_codec_dev_adc3101 and "codec_dai" member of "struct snd_soc_dai_link" to &adc3101_dai.
- Update Kconfig and Makefile for the platform driver to use ADC3101 codec.

NOTE:

The interface between the platform driver and codec driver is through *struct snd_soc_dai*. The platform driver calls the codec functions thorough function pointers hw_params, set_sysclk, set_fmt are all function pointers

```
struct snd_soc_dai adc3101_dai = {  
  
    .name = "adc3101",  
  
    .capture = {  
  
        .stream_name = "Capture",  
        .channels_min = 1,  
        .channels_max = 2,  
        .rates = ADC3101_RATES,  
        .formats = ADC3101_FORMATS, },  
  
    .ops = {  
  
        .hw_params = adc3101_hw_params,
```



```
    },  
  
    .dai_ops = {  
        .set_sysclk = adc3101_set_dai_sysclk,  
        .set_fmt = adc3101_set_dai_fmt,  
    }  
};
```

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

Example: struct snd_soc_dai adc3101_dai exported by ADC3101 codec 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 OMAP3530 using ADC3101 codec driver

```
static struct snd_soc_dai_link omap3evm_dai = {  
  
    .name = "TLV320ADC3101",  
    .stream_name = "ADC3101",  
    .cpu_dai = &omap_mcbasp_dai[0],  
    .codec_dai = &adc3101_dai,  
    .init = omap_adc3101_init,  
    .ops = &omap3evm_ops,  
};  
  
static struct snd_soc_machine snd_soc_machine_omap3evm = {  
    .name = "omap3evm",  
    .dai_link = &omap3evm_dai,  
    .num_links = 1,  
};  
  
static struct snd_soc_device omap3evm_snd_devdata = {  
  
    .machine = &snd_soc_machine_omap3evm,  
    .platform = &omap_soc_platform,  
    .codec_dev = &soc_codec_dev_adc3101,  
};
```

8 Driver Configuration

The driver has support for following MCLK frequencies 12MHz. On OMAP35x platform, the testing was conducted with MCLK frequency configured with 12MHz.

9 Testing

This section provides more information on the testing of the ADC3101 Audio Driver.

9.1 Pre-Requisites

- Linux 2.6.28 Kernel Image integrated with the ADC3101 Audio Driver.
- OMAP35x EVM interfaced with the ADC3101 EVM for enabling the Image download and testing.

9.2 Board Setup

This section explains the board setup and instructions to be followed before testing the ADC3101 Codec Driver.

- The connections between OMAP3530EVM and TLV320ADC3101 EVM are to be made according as per Section 2 of this document.
- Download the newly built Linux uImage into the OMAP3530 EVM wired with ADC3101EVM.
- With this user will be able to use LINE IN JACK (J9) or on board MIC for recording functionality.

10 Configuration of MINI DSP

ADC3101 CODEC contains miniDSP core, coupled to ADC. The miniDSP can be configured by enabling or disabling the macro "CONFIG_MINI_DSP" in tlv320adc3101.h.

If this flag is enabled, the driver audio processing will be performed by the miniDSP and hence the processing can be customized. The Linux driver support "Volume Lite" amixer control to exercise the mini-DSP,

10.1 Volume Lite Example:

Usage:

```
amixer cset numid=<id1> x
```

id1 is the numid of control with name "=Volume_1 (0=-110dB, 116=+6dB)"
x = 0 to 116

Example:

```
amixer cset numid=35 105
```

Using the amixer control we can test the static mini-DSP configuration and run time the driver will program the codec registers according to the information in configuration file (.cfg file) using TiLoad Utility.

10.2 TiLoad Utility

The TiLoad utility facilitates dynamic configuration of MINI DSP using the script (.cfg) files.

Usage:

```
#mknod /dev/minidsp c <major> <minor>  
#./TiLoad -f <config file name>.cfg
```

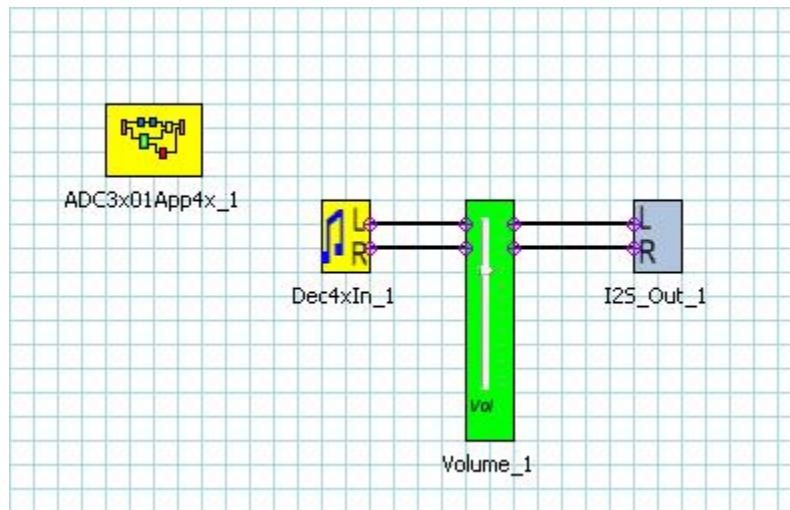
Example:

```
#mknod /dev/minidsp c 254 0  
#./TiLoad -f Pop_Rate44.cfg
```

10.3 Current ADC3101 MINI DSP support

The release 1.0 is tested and released with "Volume Lite" component in MINI DSP. The user can load different MINI DSP program through TiLoad utility at run-time.

The configuration files used in release 1.0 were generated using TI Pure Path Studio (PPS) project and having the characteristics as shown below diagram.



11 Testing on other platforms

User has to make hardware modifications as supported by their platform and has to test recording functionality.

12 Known Issues/Caveats in this release

- The adc3101 ASOC driver is tested for Line input as well as Mic Input using single-ended LINEIN1L and LINEINR input channels. Other inputs are supported through mixer controls but not tested because of the hardware limitations.
- Differential Line inputs are supported through mixer controls but not tested because of the hardware limitations.
- Digital Microphone support is implemented in the driver, but not tested because of unavailability of Digital Mic hardware.