TI81XX PSP AUDIO Driver User Guide

DM81xx AM38xx Audio Driver User Guide

Linux PSP

IMPORTANT
TI81XX refers to TI816X, TI814X and TI813X.

Introduction

The AIC31 audio module contains audio analog inputs and outputs. It is connected to the main TI81xx processor through the TDM/I2S interface (audio interface) and used to transmit and receive audio data. The AIC31 audio codec is connected via Multi-Channel Audio Serial Port (McASP) interface, a communication peripheral, to the main processor.

McASP provides a full-duplex direct serial interface between the host device (DM1xx) and other audio devices in the system such as the AIC31 codec. It provides a direct interface to industry standard codecs, analog interface chips (AICs) and other serially connected A/D and D/A devices:

- Inter-IC Sound (I2S) compliant devices
- Pulse Code Modulation (PCM) devices
- Time Division Multiplexed (TDM) bus devices.

The AIC31 audio module is controlled by internal registers that can be accessed by the high speed I2C control interface.

This user manual defines and describes the usage of user level and platform level interfaces of the ALSA SoC Audio driver.

References

1. ALSA SoC Project Homepage [http://www.alsa-project.org/main/index.php/ASoC [1]]
2. ALSA Project Homepage [http://www.alsa-project.org/main/index.php/Main_Page [2]]
3. ALSA User Space Library [http://www.alsa-project.org/alsa-doc/alsa-lib/ [3]]
4. Using ALSA Audio API [http://www.equalarea.com/paul/alsa-audio.html [4]] Author: Paul Davis
5. TLV320AIC31 - Low-Power Stereo CODEC with HP Amplifier [http:/ / focus. ti. com/ lit/ ds/ symlink/ tlv320aic31.pdf [5]]

Acronyms & Definitions

Audio Driver: Acronyms

Acronym	Definition
ALSA	Advanced Linux Sound Architecture
ALSA SoC	ALSA System on Chip
DMA	Direct Memory Access
I2C	Inter-Integrated Circuit
McASP	Multi-channel Audio Serial Port
PCM	Pulse Code Modulation
TDM	Time Division Multiplexing
OSS	Open Sound System
MMAP	Memory Mapped
I2S	Inter-IC Sound

Features

This section describes the features supported by ALSA SoC Audio driver.

- Supports AIC31 audio codec on DM81xx in ALSA SoC framework.
- Multiple sample rates support (8KHz, 11.025KHz, 16KHz, 22.05KHz, 32KHz, 44.1KHz, 48KHz, 64KHz, 88.2KHz and 96KHz) for both capture and playback.
- Supports audio in stereo mode.
- Supports simultaneous playback and record (full-duplex mode).
- Start, stop, pause and resume feature.
- Supports mixer interface for audio codecs.
- Supports MMAP mode for both playback and capture.
- McASP is configured as slave and AIC31 Codec is configured as master.

ALSA SoC Architecture

Introduction

The overall project goal of the ALSA System on Chip (ASoC) layer is to provide better ALSA support for embedded system on chip processors and portable audio codecs. Currently there is some support in the kernel for SoC audio, however it has some limitations:

- Currently, codec drivers are often tightly coupled to the underlying SoC CPU. This is not really ideal and leads to code duplication.
- There is no standard method to signal user initiated audio events e.g. Headphone/Mic insertion, Headphone/Mic detection after an insertion event.
- Current drivers tend to power up the entire codec when playing (or recording) audio. This is fine for a PC, but tends to waste a lot of power on portable devices. There is also no support for saving power via changing codec oversampling rates, bias currents, etc.

Design

The ASoC layer is designed to address these issues and provide the following features:

- **Codec independence**: Allows reuse of codec drivers on other platforms and machines.
- **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.
- **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.
- **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 splits an embedded audio system into three components:

- **Codec driver**: The codec driver is platform independent and contains audio controls, audio DAPM 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 DAPM at start of playback.

Following architecture diagram shows all the components and the interactions among them:

Configuration

To enable/disable audio support, start the *Linux Kernel Configuration* tool:

```
$ make menuconfig
```

Select *Device Drivers* from the main menu.

```
...
    ...
    Power management options --->
    [ ] Networking support --->
    Device Drivers --->
    File systems --->
    Kernel hacking --->
    ...
    ...
```

Select *Sound card support* as shown here:

```
...
    ...
    Multimedia devices --->
    Graphics support --->
<*> Sound card support --->
[*] HID Devices --->
[*] USB support --->
    ...
    ...
```

Select *Advanced Linux Sound Architecture* as shown here:

```
--- Sound card support
<*> Advanced Linux Sound Architecture --->
< > Open Sound System (DEPRECATED) --->
```

Select *ALSA for SoC audio support* as shown here:

```
...
    ...
[*] ARM sound devices --->
[*] SPI sound devices --->
<*> ALSA for SoC audio support --->
```

Select *SoC Audio for TI81XX chip* as shown here:

```
--- ALSA for SoC audio support
<*> SoC Audio for the TI81XX chip
<*> SoC Audio support for TI81XX EVM
< > Build all ASoC CODEC drivers (NEW)
```

Note: *Soc Audio support for TI81XX EVM* option appears if *SoC Audio for the TI81XX chips* is selected

Module Build

Module build for the audio driver is supported. To do this, at all the places mentioned in the section above select module build (short-cut key **M**).

After the modules are built, the following sound related kernel modules (*.ko) will be generated

```
sound/soundcore.ko
sound/core/snd.ko
sound/core/snd-hwdep.ko
sound/core/snd-timer.ko
sound/core/snd-page-alloc.ko
sound/core/snd-pcm.ko
sound/core/snd-rawmidi.ko
sound/core/snd-usb-audio.ko
sound/core/snd-usb-lib.ko
sound/soc/snd-soc-core.ko
sound/soc/codecs/snd-soc-tlv320aic3x.ko
sound/soc/davinci/snd-soc-davinci.ko
sound/soc/davinci/snd-soc-davinci-mcasp.ko
sound/soc/davinci/snd-soc-evm.ko
```

The kernel modules need to be inserted in the order listed above.

Amixer commands for AIC3x audio codec

To unmute/mute the microphone input

```
$ amixer sset 'Right PGA Mixer Mic3L' on/off
$ amixer sset 'Right PGA Mixer Mic3R' on/off
$ amixer sset 'Left PGA Mixer Mic3L' on/off
$ amixer sset 'Left PGA Mixer Mic3R' on/off
```

To unmute/mute the line input

```
$ amixer sset 'Right PGA Mixer Line1R' on/off
$ amixer sset 'Right PGA Mixer Line1L' on/off
$ amixer sset 'Left PGA Mixer Line1R' on/off
$ amixer sset 'Left PGA Mixer Line1L' on/off
```

To select differential configuration for the line input

```
$ amixer sset 'Left Line1R Mux' differential
$ amixer sset 'Right Line1L Mux' differential
```

Note: differential configuration is used to minimizes electronic crosstalk and electromagnetic interference over the audio input cable.

To increase the volume for capture and playback

```
$ amixer cset name='PCM Playback Volume' x%,x% (x between 0-100)
$ amixer cset name='PGA Capture Volume'  x%,x%
```

Note: When using one input port for audio capture (microphone or line) please ensure that the other port is muted. If both the input ports are left enabled or configured incorrectly the captured audio stream may be noisy.

Amixer commands for AIC26 audio codec

Not applicable for DM8168/DM8148 EVMs

To enable audio Capture:

```
$ amixer cset numid=9 on
$ amixer cset numid=10 7
$ amixer cset numid=19 2
$ amixer cset numid=11 15
$ amixer cset numid=12 0
$ amixer cset numid=15 100
$ amixer cset numid=13 0
$ amixer cset numid=14 0
$ amixer cset numid=16 0
```

To enable audio Playback:

```
$ amixer cset numid=2 on
$ amixer cset numid=1 63
$ amixer cset numid=18 on
```

Using mic input on the EVM

Make sure that the following steps have been completed with the help of appropriate amixer commands

1. Mic input unmuted
2. Line input muted
3. Capture volume set to a reasonable level of around 70%

Note: MIC IN capture gain setting must not be more than 70%. Audio clipping is observed on the output beyond the 70% capture gain.

Using line-in input on the EVM

Make sure that the following steps have been completed with the help of appropriate amixer commands

1. Line-in input unmuted
2. Mic input muted
3. Use differential configurations for LINE IN to suppress the noise

Note: Please adjust volume level of the audio source used to ensure the capture quality. If the source signal level increased beyond a limit, some part of the input signals will get clipped and the noise will be amplified along with the input signal and will result in noisy output.

A source volume level of 10% is advisable while using PC as the Audio source to the Line In for better Capture quality.

Application Interface

This section provides the details of the Application Interface for the ALSA Audio driver.

Application developer uses ALSA-lib, a user space library, rather than the kernel API. The library offers 100% of the functionality of the kernel API, but adds major improvements in usability, making the application code simpler and better looking.

The online-documentation for the same is available at:

http://www.alsa-project.org/alsa-doc/alsa-lib/ [3]

Device Interface

The operational interface in `/dev/` contains three main types of devices:

*   PCM devices for recording or playing digitized sound samples,
*   CTL devices that allow manipulating the internal mixer and routing of the card, and,
*   MIDI devices to control the MIDI port of the card, if any.

Device Interface

Name	Description
/dev/snd/controlC0	Control devices (i.e. mixer, etc)
/dev/snd/pcmC0D0c	PCM Card 0 Device 0 Capture device
/dev/snd/pcmC0D0p	PCM Card 0 Device 0 Playback device

Proc Interface

The `/proc/asound` kernel debug interface which allows the users to know the status and configuration interface.
A lot of useful information about the sound system can be found in the `/proc/asound` subdirectory.

See the table below for different proc entries in `/proc/asound`:

Proc Interface

Name	Description
cards	List of registered cards
version	Version and date the driver was built on
devices	List of registered ALSA devices
pcm	The list of allocated PCM streams
cardX/ (X = 0-7)	The card specific directory
cardX/pcm0p	The directory of the given PCM playback stream
cardX/pcm0c	The directory of the given PCM capture stream

Commonly Used APIs

Some of the commonly used APIs to write an ALSA based application are:

Commonly Used APIs

Name	Description
snd_pcm_open	Opens a PCM stream
snd_pcm_close	Closes a previously opened PCM stream
snd_pcm_hw_params_any	Fill params with a full configuration space for a PCM
snd_pcm_hw_params_test_ <<parameter>>	Test the availability of important parameters like number of channels, sample rate etc. For e.g. snd_pcm_hw_params_test_format, snd_pcm_hw_params_test_rate etc.
snd_pcm_hw_params_set_ <<parameter>>	Set the different configuration parameters. For e.g. snd_pcm_hw_params_set_format, snd_pcm_hw_params_set_rate etc.
snd_pcm_hw_params	Install one PCM hardware configuration chosen from a configuration space
snd_pcm_writei	Write interleaved frames to a PCM
snd_pcm_readi	Read interleaved frames from a PCM
snd_pcm_prepare	Prepare PCM for use
snd_pcm_drop	Stop a PCM dropping pending frames
snd_pcm_drain	Stop a PCM preserving pending frames

User Space Interactions

This section depicts the sequence of operations for a simple playback and capture application.
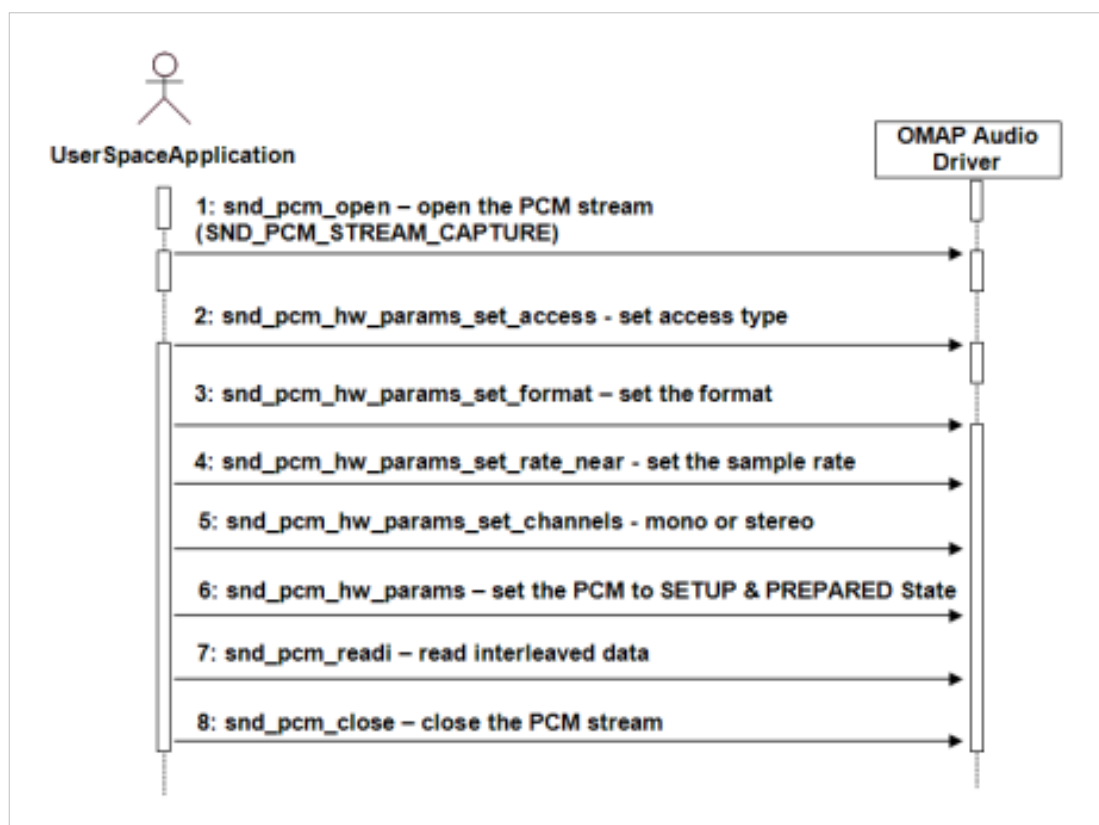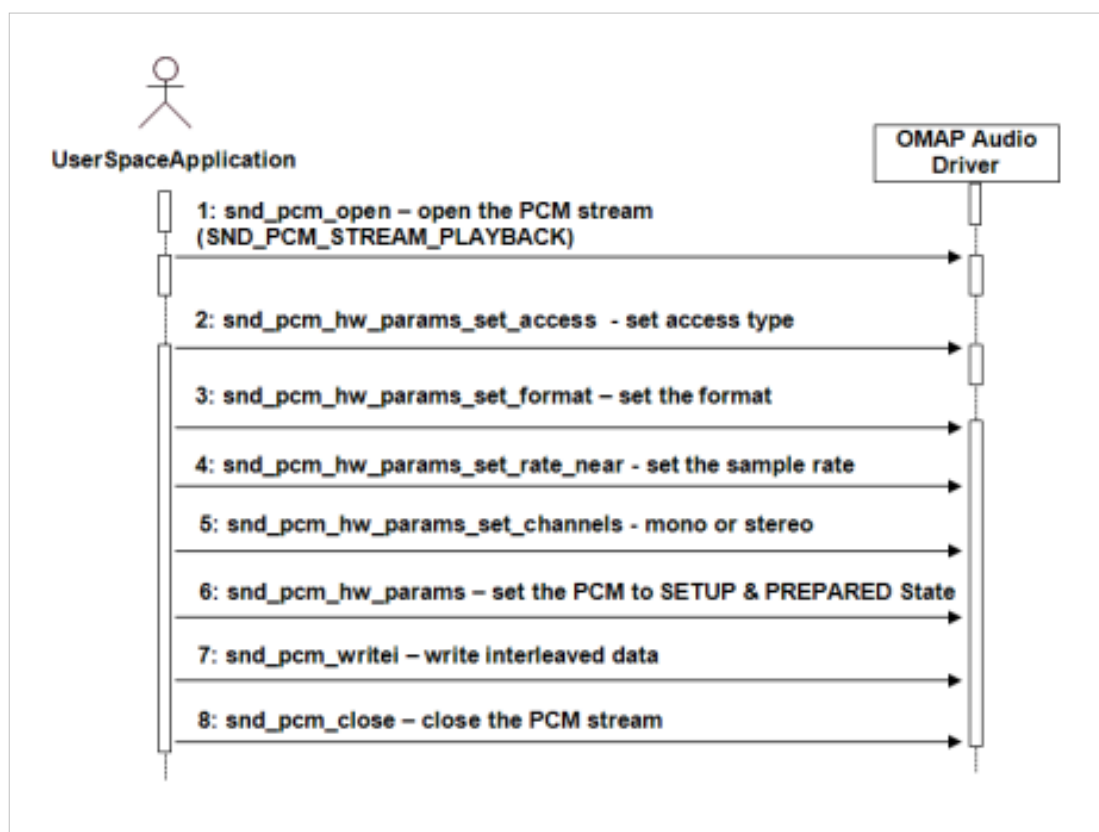
Sample Applications

This chapter describes the audio sample applications provided along with the package. The source for these sample applications are available in the Examples directory of the Release Package folder.

Introduction

Writing an audio application involves the following steps:

- Opening the audio device
- Set the parameters of the device
- Receive audio data from the device or deliver audio data to the device
- Close the device

These steps are explained in detail in this section.

Note

User space ALSA libraries can be downloaded from this link http://www.alsa-project.org/main/index.php/ Download [6]. User needs to build and install them before he starts using the ALSA based applications. Follow the INSTALL guide in the example code package to build the sample apps

A minimal playback application

This program opens an audio interface for playback, configures it for stereo, 16 bit, 44.1kHz, interleaved conventional read/write access. Then its delivers a chunk of random data to it, and exits. It represents about the simplest possible use of the ALSA Audio API, and isn't meant to be a real program.

Opening the audio device

To write a simple PCM application for ALSA, we first need a handle for the PCM device. Then we have to specify the direction of the PCM stream, which can be either playback or capture. We also have to provide some information about the configuration we would like to use, like buffer size, sample rate, pcm data format. So, first we declare:

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

#define BUFF_SIZE 4096

int main (int argc, char *argv[])
{
  int err;
  short buf[BUFF_SIZE];
  int rate = 44100; /* Sample rate */
  unsigned int exact_rate; /* Sample rate returned by */
  /* Handle for the PCM device */
  snd_pcm_t *playback_handle;
  /* Playback stream */
  snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
  /* This structure contains information about the hardware and can be used to specify the configuration to be used for */
  /* the PCM stream. */
  snd_pcm_hw_params_t *hw_params;
```

The most important ALSA interfaces to the PCM devices are the "plughw" and the "hw" interface. If you use the "plughw" interface, you need not care much about the sound hardware. If your sound card does not support the sample rate or sample format you specify, your data will be automatically converted. This also applies to the access type and the number of channels. With the "hw" interface, you have to check whether your hardware supports the configuration you would like to use. Otherwise, user can use the default interface for playback by:

```
/* Name of the PCM device, like plughw:0,0 */

  /* The first number is the number of the soundcard, the second number is the number of the device. */


  static char *device = "default"; /* playback device */
```

Now we can open the PCM device:

```
/* Open PCM. The last parameter of this function is the mode. */
  if ((err = snd_pcm_open (&playback_handle, device, stream, 0)) < 0) {
    fprintf (stderr, "cannot open audio device (%s)\n", snd_strerror (err));
    exit (1);
  }
```

Setting the parameters of the device

Now we initialize the variables and allocate the `hwparams` structure:

```
/* Allocate the snd_pcm_hw_params_t structure on the stack. */
  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameters (%s)\n", snd_strerror (err));
    exit (1);
  }
```

Before we can write PCM data to the soundcard, we have to specify access type, sample format, sample rate, number of channels, number of periods and period size. First, we initialize the hwparams structure with the full configuration space of the soundcard:

```
/* Init hwparams with full configuration space */
  if ((err = snd_pcm_hw_params_any (playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize hardware parameter structure (%s)\n", snd_strerror (err));
    exit (1);
  }
```

Now configure the desired parameters. For this example, we assume that the soundcard can be configured for stereo playback of 16 Bit Little Endian data, sampled at 44100 Hz. Therefore, we restrict the configuration space to match this configuration only. The access type specifies the way in which multi-channel data is stored in the buffer. For INTERLEAVED access, each frame in the buffer contains the consecutive sample data for the channels. For 16 Bit stereo data, this means that the buffer contains alternating words of sample data for the left and right channel.

```
/* Set access type. */
  if ((err = snd_pcm_hw_params_set_access (playback_handle, hw_params, SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type (%s)\n", snd_strerror (err));
    exit (1);
  }


  /* Set sample format */
```

```
    if ((err = snd_pcm_hw_params_set_format (playback_handle, hw_params, SND_PCM_FORMAT_S16_LE)) < 0) {

      fprintf (stderr, "cannot set sample format (%s)\n", snd_strerror (err));

      exit (1);

    }



  /* Set sample rate. If the exact rate is not supported by the hardware, use nearest possible rate. */

  exact_rate = rate;

  if ((err = snd_pcm_hw_params_set_rate_near (playback_handle, hw_params, &exact_rate, 0)) < 0) {

    fprintf (stderr, "cannot set sample rate (%s)\n", snd_strerror (err));

    exit (1);

  }



  if (rate != exact_rate) {

    fprintf(stderr, "The rate %d Hz is not supported by your hardware.\n ==> Using %d Hz instead.\n", rate, exact_rate);

  }



  /* Set number of channels */

  if ((err = snd_pcm_hw_params_set_channels (playback_handle, hw_params, 2)) < 0) {

    fprintf (stderr, "cannot set channel count (%s)\n", snd_strerror (err));

    exit (1);

  }
```

Now we apply the configuration to the PCM device pointed to by `pcm_handle` and prepare the PCM device.

```
  /* Apply HW parameter settings to PCM device and prepare device. */
  if ((err = snd_pcm_hw_params (playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n", snd_strerror (err));
    exit (1);
  }


  snd_pcm_hw_params_free (hw_params);


  if ((err = snd_pcm_prepare (playback_handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n", snd_strerror (err));
    exit (1);
  }
```

Writing data to the device

After the PCM device is configured, we can start writing PCM data to it. The first write access will start the PCM playback. For interleaved write access, we use the function:

```
/* Write some junk data to produce sound. */
  if ((err = snd_pcm_writei (playback_handle, buf, BUFF_SIZE/2)) != BUFF_SIZE/2) {
    fprintf (stderr, "write to audio interface failed (%s)\n", snd_strerror (err));
    exit (1);
  } else {
    fprintf (stdout, "snd_pcm_writei successful\n");
  }
```

After the PCM playback is started, we have to make sure that our application sends enough data to the soundcard buffer. Otherwise, a buffer under-run will occur. After such an under-run has occurred, `snd_pcm_prepare` should be called.

Closing the device

After the data has been transferred, the device needs to be closed by calling:

```
  snd_pcm_close (playback_handle);


  exit (0);
}
```

A minimal record application

This program opens an audio interface for capture, configures it for stereo, 16 bit, 44.1kHz, interleaved conventional read/write access. Then its reads a chunk of random data from it, and exits. It isn't meant to be a real program.

Note that it is not possible to use one pcm handle for both playback and capture. So you have to configure two handles if you want to access the PCM device in both directions.

```c
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>


#define BUFF_SIZE 4096


int main (int argc, char *argv[])
{
  int err;
  short buf[BUFF_SIZE];
  int rate = 44100; /* Sample rate */
  int exact_rate; /* Sample rate returned by */
  snd_pcm_t *capture_handle;

  /* This structure contains information about the hardware and can be used to specify the configuration to be used for */
  /* the PCM stream. */
  snd_pcm_hw_params_t *hw_params;

  /* Name of the PCM device, like hw:0,0 */
  /* The first number is the number of the soundcard, the second number is the number of the device. */
  static char *device = "hw:0,0"; /* capture device */

  /* Open PCM. The last parameter of this function is the mode. */
  if ((err = snd_pcm_open (&capture_handle, device, SND_PCM_STREAM_CAPTURE, 0)) < 0) {
    fprintf (stderr, "cannot open audio device (%s)\n", snd_strerror (err));
    exit (1);
  }


  memset(buf,0,BUFF_SIZE);
```

```c
/* Allocate the snd_pcm_hw_params_t structure on the stack. */

if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {

  fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n", snd_strerror (err));

  exit (1);

}


/* Init hwparams with full configuration space */

if ((err = snd_pcm_hw_params_any (capture_handle, hw_params)) < 0) {

  fprintf (stderr, "cannot initialize hardware parameter structure (%s)\n", snd_strerror (err));

  exit (1);

}


/* Set access type. */

if ((err = snd_pcm_hw_params_set_access (capture_handle, hw_params, SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {

  fprintf (stderr, "cannot set access type (%s)\n", snd_strerror (err));

  exit (1);

}


/* Set sample format */

if ((err = snd_pcm_hw_params_set_format (capture_handle, hw_params, SND_PCM_FORMAT_S16_LE)) < 0) {

  fprintf (stderr, "cannot set sample format (%s)\n", snd_strerror (err));

  exit (1);

}


/* Set sample rate. If the exact rate is not supported by the hardware, use nearest possible rate. */

exact_rate = rate;

if ((err = snd_pcm_hw_params_set_rate_near (capture_handle, hw_params, &exact_rate, 0)) < 0) {

  fprintf (stderr, "cannot set sample rate (%s)\n", snd_strerror (err));

  exit (1);

}


if (rate != exact_rate) {

  fprintf(stderr, "The rate %d Hz is not supported by your hardware.\n ==> Using %d Hz instead.\n", rate, exact_rate);

}


/* Set number of channels */

if ((err = snd_pcm_hw_params_set_channels(capture_handle, hw_params, 2)) < 0) {

  fprintf (stderr, "cannot set channel count (%s)\n", snd_strerror (err));

  exit (1);

}


/* Apply HW parameter settings to PCM device and prepare device. */

if ((err = snd_pcm_hw_params (capture_handle, hw_params)) < 0) {

  fprintf (stderr, "cannot set parameters (%s)\n", snd_strerror (err));

  exit (1);

}
```

```
snd_pcm_hw_params_free (hw_params);


if ((err = snd_pcm_prepare (capture_handle)) < 0) {

  fprintf (stderr, "cannot prepare audio interface for use (%s)\n", snd_strerror (err));

  exit (1);

}


/* Read data into the buffer. */

if ((err = snd_pcm_readi (capture_handle, buf, 128)) != 128) {

  fprintf (stderr, "read from audio interface failed (%s)\n", snd_strerror (err));

  exit (1);

} else {

  fprintf (stdout, "snd_pcm_readi successful\n");

}


snd_pcm_close (capture_handle);


exit (0);

}
```

FAQ

How to add Multi-Channel Audio support ?

The Linux SoC audio driver is implemented under the ALSA framework. Please refer to attached PPT

The current version of the TI81XX audio driver only support stereo mode of operations due the codec limitations used in the EVM. To add the multi-channel audio capture support using an external capture setup (FPGA) connected to the McASP using single serializer follow the steps give bellow.

Assumptions

• Audio captured data is provided to the McASP in Time Division Multiplex(TDM) format connected through a single serializer.

• The is no or little of configurations required for the codec or the FPGA are done outside similar to the HDMI codec used as the reference.

Add the support in ALSA SoC Driver

1. Add Dummy codec configurations. Please refer the sample dummy codec added for the HDMI driver and add similar one. (PFA for the patch).

2. Modify the Machine Driver (Glue layer) Refer to the WebEx Presentations for how to configure the glue layer to connect between the codec and the McASP driver

Ex: Use the following code snippet use case and modify the data in the sound/soc/davinci/davinci-evm.c . This will create a new capture device in ALSA

static struct snd_soc_dai_link ti81xx_evm_dai[] = {

```
        {
                    [.....]      Remove the configurations for the AIC codec in the EVEM
        },
        {
                    .name = "DUMMY_SOC_LINK",
```

```
                        .stream_name = "dummy",

                        .cpu_dai_name = " davinci-mcasp.2",

                        .platform_name = "davinci-pcm-audio",

                        .codec_dai_name = " DUMMY-DAI-CODEC",      /* DAI name */

                        .codec_name = "dummy-dummy-codec", /*strings from dummy-codec driver*/

              },
```

}

3. Use the ALSA user space applications to capture data

Refer the sample capture application specified in the Audio User guide. http://processors.wiki.ti.com/index.php/ TI81XX_Audio_Driver_User_Guide#A_minimal_record_application Modify the number of channel specified in the code to enable multi-channel capture bases on input given.

```
/* Set number of channels */
if ((err = snd_pcm_hw_params_set_channels(capture_handle, hw_params, <Number of channels>)) < 0) {
  fprintf (stderr, "cannot set channel count (%s)\n", snd_strerror (err));
  exit (1);
}
```

The captured output buffer will have the multi-channel PCM data.

Note: For testing purpose use the ALSA arecrod utility to capture the data to a WAV file and verify the data using the audio analyzer tool.

```
TI8166_EVM#     arecord –f cd  –t wav  –c <num of channels to capture> <Audio file.wav>
```

The above command will capture the data from save it in the WAV file format

References

[1] http://www.alsa-project.org/main/index.php/ASoC

[2] http://www.alsa-project.org/main/index.php/Main_Page

[3] http://www.alsa-project.org/alsa-doc/alsa-lib/

[4] http://www.equalarea.com/paul/alsa-audio.html

[5] http://focus.ti.com/lit/ds/symlink/tlv320aic31.pdf

[6] http://www.alsa-project.org/main/index.php/Download

Article Sources and Contributors

TI81XX PSP AUDIO Driver User Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=121765 *Contributors*: Deepu.raj, Parth.saxena, RK

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

Image:Alsa-usecase1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Alsa-usecase1.png *License*: unknown *Contributors*: SriramAG

Image:Alsa-usecase2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Alsa-usecase2.png *License*: unknown *Contributors*: SriramAG

License

b.   Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a.   Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b.   Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c.   If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d.   No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e.   This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f.   The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.