Audio Training Exercises

Session Exercises

Clone the git repo for exercises.
$ git clone https://github.com/embitude/training/
PS. Just git pull, if repo has already been cloned

Session 1 Exercises

Exercise 1: Listing the sound card and devices

1. On the target board, execute the following command:
   $ modprobe snd-dummy
   This would load the dummy sound driver, which registers the sound card and corresponding devices
2. List the playback & capture devices
   $ aplay -l (This should list the playback devices)
   $ arecord -l (This would list the capture devices)

Steps for compiling the ALSA applications

The ALSA application compilation needs the cross alsa lib for compilation. This doesn't comes as a part of toolchain. So, the sdk generated which has the support for alsa lib is used to compile the applicatoin

1. Get into the Builds directory & untar the sdk
   $ cd training/Builds
   $ tar -xf arm-buildroot-linux-gnueabihf_sdk-buildroot.tar.gz
2. Relocate the sdk
   $ cd arm-buildroot-linux-gnueabihf_sdk-buildroot
   $ ./relocate-sdk.sh
3. Next step is to source the sdk environment in shell which is used for application compilation & compile the application
   $ source training/Builds/arm-buildroot-linux-gnueabihf_sdk-buildroot/environment-setup
   $ cd training/Audio/Apps
   $ make

Exercise 2: Set Parameters

The objective here is to set the parameters & verify if it has been updated

1. Navigate to Audio/Apps directory
   $ cd training/Audio/Apps
2. Compile the application
   $ make
   This would generate the executable for all the applications. Let's test the set_params application
3. Transfer the application on the board and make sure that the snd-dummy is loaded
   $ scp set_params root@192.168.7.2:
   $ modprobe snd-dummy
4. Execute the application
   $ ./set_params hw:0,0

Exercise 3: Minimal Playback application

The objective here is to test minimal playback application on target board

1. Navigate to Audio/Apps directory
   $ cd training/Audio/Apps
2. Compile the application
   $ make
   This would generate the executable for all the applications. Let's test the playback_min application
3. Transfer the application on the board and make sure that the snd-dummy is loaded
   $ scp playback_min root@192.168.7.2:
   $ modprobe snd-dummy
4. Execute the application
   $ ./playback_min hw:0,0
   This should playback the audio to the dummy sound card

Exercise 4: Minimal Capture application

The objective here is to test minimal capture application on target board

1. Navigate to Audio/Apps directory
   $ cd training/Audio/Apps
2. Compile the application
   $ make
   This would generate the executable for all the applications. Let's test the cap_min application
3. Transfer the application on the board and make sure that the snd-dummy is loaded
   $ scp cap_min root@192.168.7.2:
   $ modprobe snd-dummy
4. Execute the application
   $ ./cap_min hw:0,0
   This should capture the audio from dummy sound card

Exercise 5: Playback Application with various transfer methods

The objective here is to demonstrate the various mechanisms to transfer the data to the driver

1. Navigate to Audio/Apps directory
   $ cd training/Audio/Apps
2. Compile the application
   $ make
   This would generate the executable for all the applications. Let's test the playback application
3. Transfer the application on the board and make sure that the snd-dummy is loaded
   $ scp playback root@192.168.7.2:
   $ modprobe snd-dummy
4. Execute the application
   $ ./playback < <audio file>
   This should playback the audio with selected transfer method

Session 2 Exercises (ALSA Drivers)

Exercise 1: Register the Platform Driver & Platform Device

The idea over here is to make sure that the platform driver and device is getting registered & the corresponding probe is getting invoked.

1. Navigate to Audio/Drivers/Alsa directory
   $ cd training/Audio/Driver/Alsa
2. Complete the todos 1.1 to 1.5 in dummy.c & compile the driver
   $ make
   This would generate the kernel module with name dummy.ko.
3. Transfer dummy.ko to the board & load the same
   $ insmod dummy.ko
   Verify that the probe gets invoked on insmod and remove gets invoked on rmmod.

Exercise 2: Register the Sound card

The objective here is to register the sound card with the alsa core

1. Navigate to Audio/Driver/Alsa directory
   $ cd training/Audio/Driver/Alsa
2. Complete the todos 2.1 to 2.4 in dummy.c & compile the driver
   $ make
   This would generate the kernel module with name dummy.ko.
3. Transfer dummy.ko to the board & load the same
   $ insmod dummy.ko
4. Verify that the sound card is listed
   $ cat /proc/asound/MySoundCard/id (Should display the id string for the sound card)
   $ cat /proc/asound/card0/id

Exercise 3: Register the PCM device and operations

The objective here is to register the pcm device for the card

1. Navigate to Audio/Driver directory
   $ cd training/Audio/Driver/Alsa
2. Complete the todos 3.1 to 3.4 in dummy.c & compile the driver
   $ make
   This would generate the kernel module with name dummy.ko.
3. Transfer dummy.ko to the board & load the same
   $ insmod dummy.ko
4. Verify that the sound card is listed
   $ cat /proc/asound/MySoundCard/id (Should display the id string for the sound card)
   $ cat /proc/asound/card0/id
5. Verify that the capture & playback devices are listed
   $ aplay -l (Should list the playback devices)
   $ arecord -l (Should list the capture devices)
6. Next verify if the device files are created
   $ ls /dev/snd/pcmC0D0[p/c]
   These are the devices files for the capture & playback
7. Next, try playback and observe the behaviour
   $ aplay <audio_file>

Exercise 4: Playback ops

The objective here is to observe the pcm ops getting invoked as a part of application invoking the playback

1. Navigate to Audio/Driver/Alsa directory
   $ cd training/Audio/Driver/Alsa
2. Complete the todos 3.5 to 3.9 in dummy.c & compile the driver
   $ make
   This would generate the kernel module with name dummy.ko.
3. Transfer dummy.ko to the board & load the same
   $ insmod dummy.ko
4. Verify that the capture & playback devices are listed
   $ aplay -l (Should list the playback devices)
5. Verify the playback
   aplay <audio file>
   Observe the calls such as open, close, hw_params, prepare, trigger and hw_free getting invoked

Exercise 5: Update the buffer positions

The objective here is to get the playback working & update the pointers accordingly

1. Navigate to Audio/Driver directory
   $ cd training/Audio/Driver/Alsa
2. Complete the todos 4.1 to 4.18 in dummy.c & compile the driver
   $ make
   This would generate the kernel module with name dummy.ko.
3. Transfer dummy.ko to the board & load the same
   $ insmod dummy.ko
4. Verify that the capture & playback devices are listed
   $ aplay -l (Should list the playback devices)
5. Verify the playback
   aplay <audio file>
   The playback should work without any issues and buffer positions should be updated accordingly

Exercise 6: Capture the dummy data

The objective here is to fill capture buffer with the pre-defined data

1. Navigate to Audio/Driver/Alsa directory
   $ cd training/Audio/Driver/Alsa
2. Complete the todos 5.1 to 5.3 in dummy.c & compile the driver
   $ make
   This would generate the kernel module with name dummy.ko.
3. Transfer dummy.ko to the board & load the same
   $ insmod dummy.ko
4. Verify that the capture & playback devices are listed
   $ aplay -l (Should list the playback devices)
5. Verify the capture
   arecord <audio file>

Transfer the audio file to the PC and observe the waveform

Session-3 Exercises (ASoC Drivers)
Kernel Configuration & Device Tree Changes

1. Kernel Changes
   The audio controller is not enabled in the default kernel configuration. So, enable the same by selecting the following menu options:
   + Advanced Linux Sound Architecture > ALSA for SoC audio support -> SoC Audio for Texas Instruments chips using eDMA
   + Advanced Linux Sound Architecture > ALSA for SoC audio support -> Multichannel Audio Serial Port (McASP) support
2. Device Tree changes
   Update the device tree with the one available at Trainings/Audio/Dtb
3. Compile the Kernel & dtb & transfer it to the board
   $ make zImage
   $ make dtbs
   $ scp arch/arm/boot/zImage root@192.168.7.2:/Data/
   $ scp arch/arm/boot/dts/am335x-boneblack.dtb root@192.168.7.2:/Data/
   Make sure to reboot the board

Exercise 1: Register a platform driver for the machine class driver

The idea over here is to make sure that the probe for the machine driver is invoked. The corresponding platform device is registered in dtb.

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos 1.1 to 1.4 in test_machine.c & compile the driver
   $ make
   This would generate the kernel module with name test_machine.ko.
3. Transfer test_machine.ko to the board & load the same
   $ insmod test_machine.ko
   This should invoke the probe for the driver

Exercise 2: Registering the ASoC sound card

The objective here is to populate the sound card data structure & register it with the ASoC framework

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos 2.1 to 2.8 in test_machine.c & compile the driver
   $ make
   This would generate the kernel module with name test_machine.ko.
3. Transfer test_machine.ko to the board & load the same
   $ insmod test_machine.ko
   Observe the behaviour, if the sound card is getting registered

Exercise 3: Writing a dummy codec driver

Having a codec class driver is mandatory for ASoC framework. So, the objective here is to write a basic dummy codec to be used with the machine driver

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos 1.1 to 1.5 in dummy_codec.c & compile the driver
   $ make
   This would generate the kernel module with name dummy_codec.ko.

3. Transfer dummy_codec.ko to the board & load the same
   $ insmod dummy_codec.ko
   This would register the codec driver. Next step is to load the machine driver and check if the sound card is created
   $ insmod test_machine.ko
   This should register the sound card and corresponding capture & playback devices should be available
4. Playback the Audio
   $ aplay <audio file>

Exercise 4: Settting up the clocks & the format

The objective here is to set the proper clocks in the machine driver to get the playback going

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos 3.1 to 3.3 in test_machine.c & compile the driver
   $ make
   This would generate the kernel module with name test_machine.ko.
3. Transfer test_machine.ko to the board & load the same
   $ insmod test_machine.ko
   $ insmod dummy_codec.ko
4. Next step is to perform the loopback test by shorting pin no. 28 & 30 of P9 header
   $ aplay <audio file>
   $ arecore a.wav
   Transfer the file a.wav to the PC and verify if the loopback works

Session-4 Exercises

Exercise 1: I2C based codec driver

The objective over here is to develop the i2c based codec driver

1. First step is to update the dtb to add the support for I2c codec. For this, refer training/Audio/Dtb/am335x-boneblack_i2c.dts. Add the node for i2c1, i2c1_pins and modify the ti,audio-codec property as below:
   ti,audio-codec = <&i2c_codec>;
2. Once done, compile the dtb, transfer it to the board and reboot
   $ make dtbs
   $ scp arch/arm/boot/dts/am335x-boneblack.dtb root@192.168.7.2:/Data
   $ reboot
3. Next step is to write the i2c based codec driver. Form this, navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
4. Complete the todos 1.1 & 1.2 in i2c_codec.c, compile & transter it to the board along with machine driver
   $ make
   This would generate i2c_codec.ko module
   $ scp i2c_codec.ko test_machine.ko root@192.168.7.2:
5. Load the codec driver and the machine driver
   $ insmod i2c_codec.ko
   $ insmod test_machine.ko
   This would invoke the probe for the codec driver. However sound card registration would fail as codec driver is not registered with the ASoC framework.

Exercise 2: Registering I2C Codec driver with ASoC framework

The objective here is to register the codec with ASoC framework

1. Codec Connection Details:

PIN (P9 Header BBB)	Description	PIN (Codec)
DC_3.3V (3, 4)	3.3V	VCC (1 & 2)
GND (1,2)	Ground	Ground (3 & 4)
I2C1_SDA (18)	I2C Data (SDA)	SDA (5)
I2C1_SCL (17)	I2C Clock (SCL)	SCL (7)
SPI1_SCLK (31)	I2S Bitclock	CLK (9 & 10)
SPI1_D0 (29)	I2S Frame Sync	WS (11 & 12)
SPI1_CS0 (28)	I2S Data input to Codec	RXSDA (13)
SPI1_D1 (30)	I2S Data out from Codec	TXSDA
Refer Page 86 of BBB_SRM.pdf for bbb pin names		

2. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
3. Complete the todos 2.1 to 2.3 in i2c_codec.c, compile & transter it to the board along with machine driver
   $ make
   This would generate i2c_codec.ko module. Transfer it to the board & load the same on the board
   $ scp i2c_codec.ko root@192.168.7.2:
   $ insmod i2c_codec.ko

Exercise 3: Configuring the codec

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos 3.1 in i2c_codec.c, compile & transter it to the board along with machine driver
   $ make
   This would generate i2c_codec.ko module. Transfer it to the board & load the same on the board
   $ scp i2c_codec.ko root@192.168.7.2:
   $ insmod i2c_codec.ko
3. Plaback & record the audio
   $ aplay <file>
   This should playback the audio on the codec
   $ arecord -r 48000 -c 2 -f S16_LE a.wav
   This should record the audio from the headset

Exercise 4: Simple Audio Card

The objective here is to replace the machine driver with generic simple audio card, thereby obliviating the need for machine driver code

1. Update the dtb by adding the ‘sound’ node’ with simple audio card from training/Audio/Dtb/am335x-boneblack_simple_audio.dts.

2. Once done, compile the dtb, transfer it to the board and reboot
   $ make dtbs
   $ scp <Kernel source>/arch/arm/boot/am335x-boneblack.dtb root@192.168.7.2:/Data/
   $ reboot
3. Next transfer the codec driver and load the same:
   $ scp i2c_codec.ko root@192.168.7.2:
   $ insmod i2c_codec.ko
   This would register the sound card. Verify the playback & recording

Exercise 5: Simple Audio Card with Clock-in

The objective here is to use the on-board codec oscillator for generating the bitclock & framesync. The clock is 24.576 MHz

1. Update the dtb by adding the 'sound' node' with simple audio card from training/Audio/Dtb/am335x-boneblack_simple_audio_clkin.dts.
2. Once done, compile the dtb, transfer it to the board and reboot
   $ make dtbs
   $ scp <Kernel source>/arch/arm/boot/am335x-boneblack.dtb root@192.168.7.2:/Data/
   $ reboot
3. Next transfer the codec driver and load the same:
   $ scp i2c_codec.ko root@192.168.7.2:
   $ insmod i2c_codec.ko
   This would register the sound card. Verify the playback & recording

Exercise 6: Simple Audio Card with Clock-in

The objective here is to use the on-board codec oscillator for generating the bitclock & framesync. The clock is 24.576 MHz

1. Update the dtb by adding the 'sound' node' with simple audio card from training/Audio/Dtb/am335x-boneblack_simple_audio.dts.
2. Once done, compile the dtb, transfer it to the board and reboot
   $ make dtbs
   $ scp <Kernel source>/arch/arm/boot/am335x-boneblack.dtb root@192.168.7.2:/Data/
   $ reboot
3. Next transfer the codec driver and load the same:
   $ scp i2c_codec.ko root@192.168.7.2:
   $ insmod i2c_codec.ko
   This would register the sound card. Verify the playback & recording

Exercise 7: Adding kcontrols to the codec driver

The objective here is to add the controls to the codec, so as to allow the volume to be controlled dynamically, mute/unmute etc

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos 4.1 in i2c_codec.c, compile & transter it to the board along with machine driver. Refer apis.txt for the prototype
   $ make
   This would generate i2c_codec.ko module. Transfer it to the board & load the same on the board

$ scp i2c_codec.ko root@192.168.7.2:
$ insmod i2c_codec.ko

3. Next thing is to play around with the controls
   $ alsmixer
   This would open the control interface. For the mute/unmute, select DAC and press 'shift m'. For volume up and down, press up/down errors. For Capture menu select F4 and for unmuting the capture, select the 'Capture' control and press 'space'

Session-5 Exercises

Exercise 1: Enabling the DAPM support in codec and machine driver

The objective over here is to develop the add the dapm widgets and paths to the above codec driver

1. Navigate to Audio/Driver/Asoc directory
   $ cd training/Audio/Driver/Asoc
2. Complete the todos (5.1 to 5.4) in i2c_codec_dapm.c, compile & transter it to the board along with machine driver
   $ make
   This would generate i2c_codec_dapm.ko module. Transfer it to the board & load the same on the board
   $ scp i2c_codec_dapm.ko root@192.168.7.2:
   $ insmod i2c_codec_dapm.ko
3. Next, complete all the todos (4.1 to 4.6) in test_machine_dapm.c
4. Compile the driver and transfer it to the board
   $ make
   $ scp test_machine_dapm.ko root@192.168.7.2:
   $ insmod test_machine_dapm.ko
5. Playback the audio
   $ aplay <Audio file>
6. Verify if the widgets are getting activated during the playbacks
   $ mount -t debugfs none /sys/kernel/debug
   $ cat /sys/kernel/debug/asoc/<card_name>/i2c-codec.1-001a/dapm/HP_L

Exercise 2: Enabling the DAPM support in simple audio card

The objective over here is to develop the add the dapm widgets and paths to the simple audio audio card

1. Update the dtb by adding the 'sound' node' with simple audio card from training/Audio/Dtb/am335x-boneblack_simple_audio_dapm.dts.
2. Once done, compile the dtb, transfer it to the board and reboot
   $ make dtbs
   $ scp <Kernel source>/arch/arm/boot/am335x-boneblack.dtb root@192.168.7.2:/Data/
   $ reboot
3. Next transfer the codec driver and load the same:
   $ scp i2c_codec_dapm.ko root@192.168.7.2:
   $ insmod i2c_codec_dapm.ko
   This would register the sound card and dapm paths as above