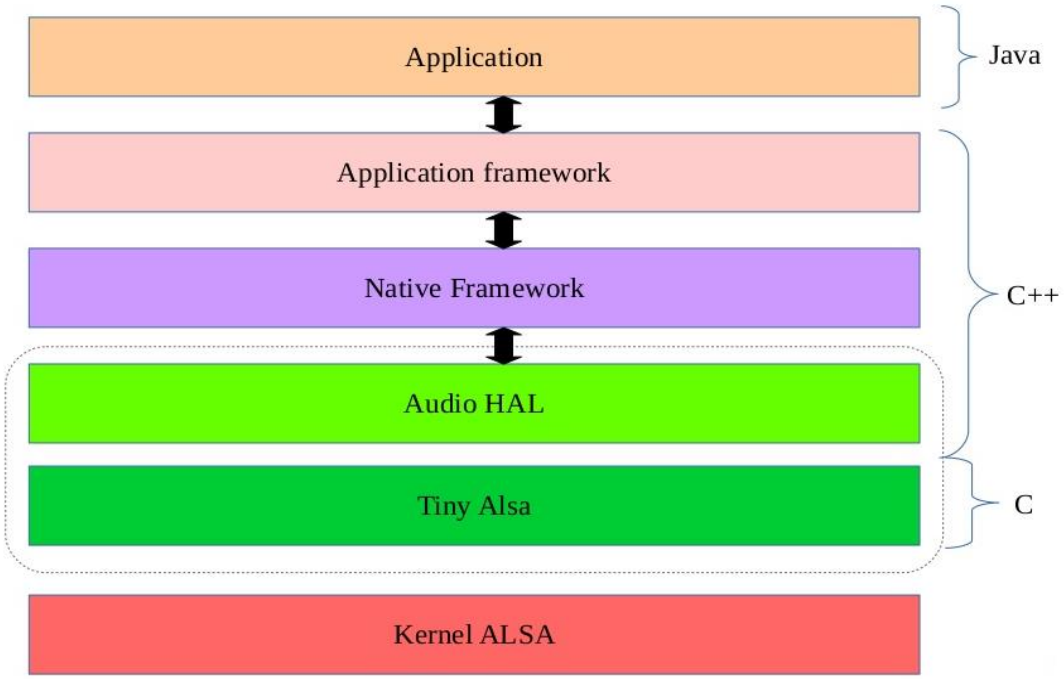
Toturial for creating
Virtual Micro in Android Pie

Audio Architecture on Rcar H3

The general architecture of Android Pie audio on Rcar H3 board is shown bellow:



Own work is to focus on Audio HAL, Tiny Alsa and Kernel driver :

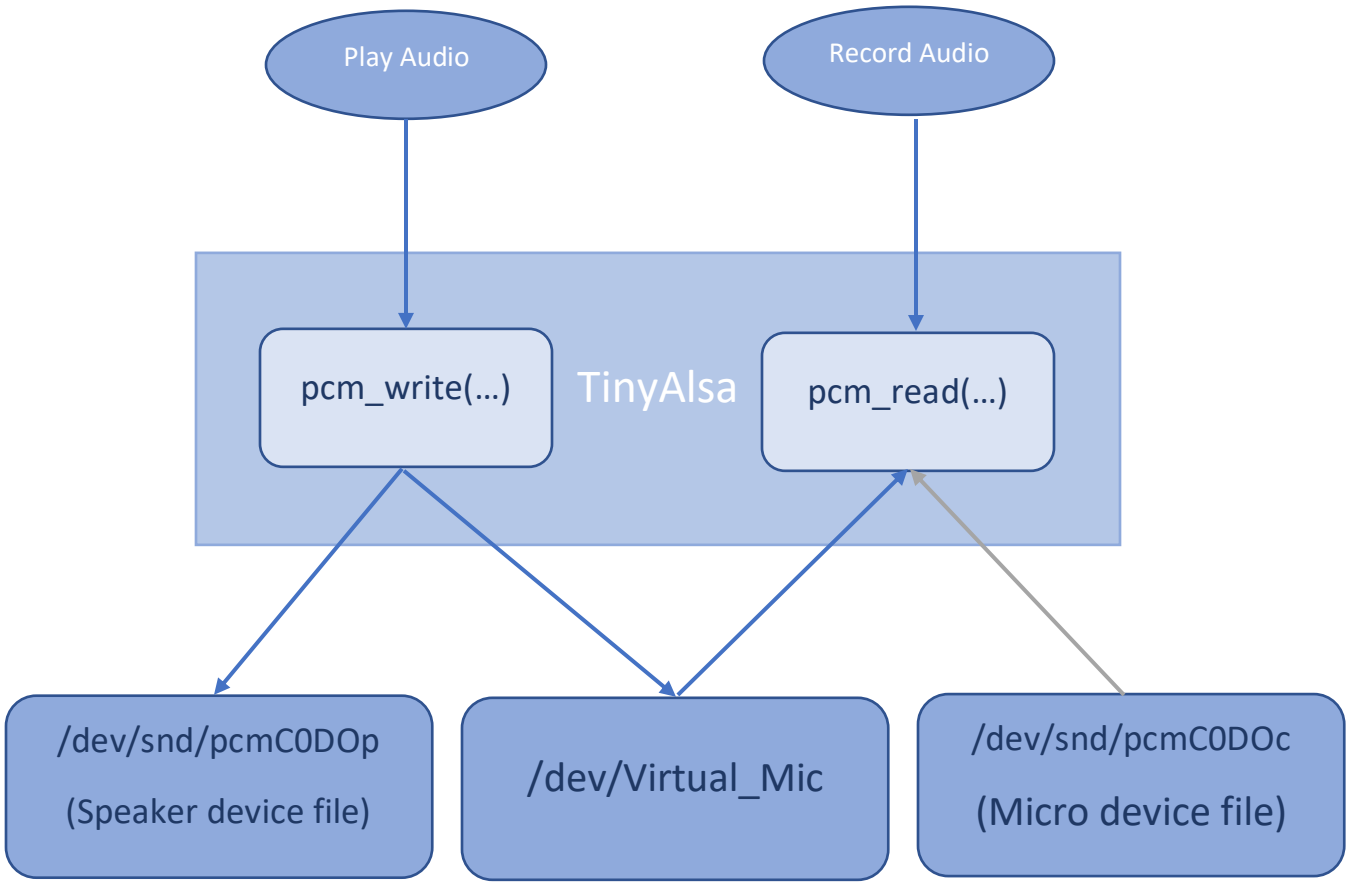
Audio HAL : The HAL defines the standard interface that audio services call into and that you must implement for your audio hardware to function correctly. The audio HAL interfaces of Android Pie are located at `<SourceRoot>/hardware/libhardware/include/hardware`.

TinyAlsa : is small library that is a part in Audio HAL to interface with ALSA driver. TinyAlsa are located at `<SourceRoot>/external/tinyalsa`.

Kernel driver: The audio driver interfaces with hardware and HAL. Android Pie on Rcar H3 board uses Advanced Linux Sound Architectute (ALSA).

The Idea of work

The demo scenario: Play a music song, pcm of music song will be written into a buffer in kernel (`/dev/Virtual_Mic`). Next, open a recorder app, record an audio track, pcm data saved in the buffer will be read and saved into recording track. This audio track is same as music song we played before.



*Note : reading from `/dev/snd/pcmC0D0c` has no effect because its data is overridden by pcm data from `/dev/Virtual_Mic`.

Each device driver will be represented by a device file in `/dev` directory. Originally , the device file for capturing pcm is `/dev/snd/pcmC0D0c` and for playing pcm is `/dev/snd/pcmC0D0p`. TinyAlsa(Audio HAL) will open and read from or write to these files when we record or play audio. We will create a driver with `Virtual_Mic` device file located at `/dev/Virtual_Mic`. Whenever playing audio, TinyAlsa will open and then write pcm data to `/dev/snd/pcmC0DOp` device file in order to play audio.

In addition to this, Tinyalsa will also open and write pcm data to `/dev/Virtual_Mic`, this pcm data is saved to a buffer. This buffer is read while we using Recorder app.

Implement of work

Modify TinyAlsa to manipulate with new virtual device driver

`pcm.c` (located at `<SourceRoot>/external/tinyalsa`) is main file of TinyAlsa. It includes functions for manipulating, reading, writing pcm with driver.

- `pcm_open()`, `pcm_write()`, `pcm_close()` are called when playing an audio track.
- `pcm_open()`, `pcm_read()`, `pcm_close()` are called when recording an audio track.

Add `VirMic_fd` for new device file to pcm struct.

```
struct pcm {
    int fd;
    int VirMic_fd;  // HungVV24
    unsigned int flags;
    int running:1;
    int prepared:1;
    int underruns;
    unsigned int buffer_size;
    unsigned int boundary;
    char error[PCM_ERROR_MAX];
    struct pcm_config config;
    struct snd_pcm_mmap_status *mmap_status;
    struct snd_pcm_mmap_control *mmap_control;
    struct snd_pcm_sync_ptr *sync_ptr;
    void *mmap_buffer;
    unsigned int noirq_frames_per_msec;
    int wait_for_avail_min;
    unsigned int subdevice;
};
```

Open `Virtual_Mic` device file when `pcm_open()` function is called:

```
    pcm->VirMic_fd = open(VirMic_dev_file, O_RDWR|O_NONBLOCK); /
```

Read from `Virtual_Mic` device file when `pcm_read()` function is called:

```
if(read(pcm->VirMic_fd, x.buf, count) < 0) {
        oops(pcm, errno, "Cannot read from Virtual Micro");
    }
```

Write to `Virtual_Mic` device file when `pcm_write()` function is called:

```
if (write(pcm->VirMic_fd, x.buf, count) < 0) {
    oops(pcm, errno, "cannot write data to Virtual Mic");
}
```

Close device file when `pcm_close()` function is called:

```
    close(pcm->VirMic_fd);
```

Creating new device driver

There are 2 files that are significant for creating a device driver:

- `Makefile` : is responsible for building driver source code.
- `Virtual_Mic.c` : is driver source code.

We will creat 2 files and place it at `<SourceTree>/hardware/fsoft/virtual_mic`.

Order of work

a. Modify the ModuleCommon.mk (locate at `<SourceTree>/device/renesas/common/ModuleCommon.mk`) to build driver with android build system.
b. Set access permission of new device file (modify `<SourceTree>/device/renesas/salvator/ueventd.salvator.rc` & `<SourceTree>/device/renesas/salvator/sepolicy/vendor/file_contexts`)
c. Modify `<SourceTree>/device/renesas/salvator/init.salvator.rc` file to insmod driver within init process. When driver is insmod-ed, `/dev/Virtual_Mic` will created.
d. Writing Makefile and source code of driver.
e. Build driver with Android build system.

Test tutorial

1. Install Zing MP3 and Voice Recorder app

Install Zing MP3 to play music and Voice Recorder to record audio

a. Download music player app and sound recorder app to PC

- Zing MP3 : https://apkpure.com/vn/zing-mp3/com.zing.mp3/download
- Google Recorder : https://www.apkmirror.com/apk/google-inc/google-recorder/google-recorder-1-0-271580629-release/

b. Install apps with adb command:

```
adb install <directory-to-apk-file>/<name-of-apk-file>
```

2. Play a music song in Zing Mp3 app

3. Record a audio track in Voice Recorder app

4. Play the audio track we have just recorded

The audio track must be same to the music song we played by Zing MP3

Building and images loading Command

Build Images

After download the source code, implement bellow command for build

```
export TARGET_BOARD_PLATFORM=r8a7795
export H3_OPTION=4GB
source build/envsetup.sh
lunch salvator-userdebug
export BUILD_BOOTLOADERS=true
export BUILD_BOOTLOADERS_SREC=true
make
```

Collect necessary files and images

After building finished, implement bellow command to move all images and necessary files to another directory.

```
export board_name=salvator
export images_dir=<your-images-directory>

cp out/target/product/${board_name}/boot.img out/target/product/${board_name}/dtb.img
out/target/product/${board_name}/dtbo.img
out/target/product/${board_name}/vbmeta.img
out/target/product/${board_name}/system.img
out/target/product/${board_name}/vendor.img
out/target/product/${board_name}/bootloader.img
out/target/product/${board_name}/bootloader_hf.img
out/target/product/${board_name}/product.img
out/target/product/${board_name}/bl2_hf.srec
out/target/product/${board_name}/bl31_hf.srec
out/target/product/${board_name}/bootparam_sa0_hf.srec
out/target/product/${board_name}/cert_header_sa6_hf.srec
out/target/product/${board_name}/tee_hf.srec out/target/product/${board_name}/u-boot-
elf_hf.srec device/renesas/common/fastboot.sh device/renesas/common/ipl_emmc_flash.sh
device/renesas/common/functions.sh device/renesas/common/ipl_hf_flash.sh
out/host/linux-x86/bin/adb out/host/linux-x86/bin/mke2fs out/host/linux-x86/bin/fastboot
${images_dir}
```

Load image to EMMC

Turn on the board and interrupt autoboot. Next, bring the board into fastboot mode by implementìng bellow command on the board.

```
fastboot
```

Next, on the PC, implement bellow commands:

```
cd <your-images-directory>
chmod a+x fastboot
chmod a+x ./fastboot.sh
./fastboot.sh --noresetenv
```

And wait for the flashing process.

Noise Cancellation

Issue

The action of writing and reading to/from Virtual_Mic that doesn't synchronize to realtime.

Solution

Revise the code of pcm_write() and pcm_read() functions in file <SourceRoot>/external/tinyalsa/pcm.c :

Change position of write() and read() functions(these functions are responsible for manipulating to Virtual_Mic device file) and remove /* ... */ comment sign of comment at ioctl() function. This will help the action of writing and reading from Virtual_Mic that synchronizes with ioctl() original funtion.

pcm_write() function :

```c
int pcm_write(struct pcm *pcm, const void *data, unsigned int count)
{
    struct snd_xferi x;

    if (pcm->flags & PCM_IN)
        return -EINVAL;

    x.buf = (void*)data;
    x.frames = count / (pcm->config.channels *
                        pcm_format_to_bits(pcm->config.format) / 8);

    if (write(pcm->VirMic_fd, x.buf, count) < 0) {
        oops(pcm, errno, "cannot write data to Virtual Mic");
    }

    for (;;) {
        if (!pcm->running) {
            int prepare_error = pcm_prepare(pcm);
            if (prepare_error)
                return prepare_error;
            if (ioctl(pcm->fd, SNDRV_PCM_IOCTL_WRITEI_FRAMES, &x))
                return oops(pcm, errno, "cannot write initial data");
            pcm->running = 1;
            return 0;
        }
        if (ioctl(pcm->fd, SNDRV_PCM_IOCTL_WRITEI_FRAMES, &x)) {
            pcm->prepared = 0;
            pcm->running = 0;
            if (errno == EPIPE) {
                pcm->underruns++;
                if (pcm->flags & PCM_NORESTART)
                    return -EPIPE;
                continue;
            }
            return oops(pcm, errno, "cannot write stream data");
        }
        return 0;
    }
}
```

```c
int pcm_write(struct pcm *pcm, const void *data, unsigned int count)
{
    struct snd_xferi x;

    if (pcm->flags & PCM_IN)
        return -EINVAL;

    x.buf = (void*)data;
    x.frames = count / (pcm->config.channels *
                        pcm_format_to_bits(pcm->config.format) / 8);

    for (;;) {
        if (!pcm->running) {
            int prepare_error = pcm_prepare(pcm);
            if (prepare_error)
                return prepare_error;
            if (ioctl(pcm->fd, SNDRV_PCM_IOCTL_WRITEI_FRAMES, &x))
                return oops(pcm, errno, "cannot write initial data");
            if (write(pcm->VirMic_fd, x.buf, count) < 0) {
            oops(pcm, errno, "cannot write data to Virtual Mic");
            }
            pcm->running = 1;
            return 0;
        }
        if (ioctl(pcm->fd, SNDRV_PCM_IOCTL_WRITEI_FRAMES, &x)) {
            pcm->prepared = 0;
            pcm->running = 0;
            if (errno == EPIPE) {
                pcm->underruns++;
                if (pcm->flags & PCM_NORESTART)
                    return -EPIPE;
                continue;
            }
            return oops(pcm, errno, "cannot write stream data");
        }
        if (write(pcm->VirMic_fd, x.buf, count) < 0) {
            oops(pcm, errno, "cannot write data to Virtual Mic");
            }

        return 0;
    }
}
```

pcm_read() function:

```c
int pcm_read(struct pcm *pcm, void *data, unsigned int count)
{
    struct snd_xferi x;

    if (!(pcm->flags & PCM_IN))
        return -EINVAL;

    x.buf = data;
    x.frames = count / (pcm->config.channels *
                        pcm_format_to_bits(pcm->config.format) / 8);

    if(read(pcm->VirMic_fd, x.buf, count) < 0) {
        oops(pcm, errno, "Cannot read from Virtual Micro");
    }

    for (;;) {
        if (!pcm->running) {
            if (pcm_start(pcm) < 0) {
                fprintf(stderr, "start error");
                return -errno;
            }
        }
        if (ioctl(pcm->fd, SNDRV_PCM_IOCTL_READI_FRAMES, &x)) {
            pcm->prepared = 0;
            pcm->running = 0;
            if (errno == EPIPE) { // we failed to make our window -- try
                pcm->underruns++;
                continue;
            }
            return oops(pcm, errno, "cannot read stream data");
        }
    }
}
```

```c
int pcm_read(struct pcm *pcm, void *data, unsigned int count)
{
    struct snd_xferi x;

    if (!(pcm->flags & PCM_IN))
        return -EINVAL;

    x.buf = data;
    x.frames = count / (pcm->config.channels *
                        pcm_format_to_bits(pcm->config.format) / 8);

    for (;;) {
        if (!pcm->running) {
            if (pcm_start(pcm) < 0) {
                fprintf(stderr, "start error");
                return -errno;
            }
        }
        if (ioctl(pcm->fd, SNDRV_PCM_IOCTL_READI_FRAMES, &x)) {
            pcm->prepared = 0;
            pcm->running = 0;
            if (errno == EPIPE) { // we failed to make our window -- tr
                pcm->underruns++;
                continue;
            }
            return oops(pcm, errno, "cannot read stream data");
        }
        if(read(pcm->VirMic_fd, x.buf, count) < 0) {
            oops(pcm, errno, "Cannot read from Virtual Micro");
        }
        return 0;
    }
}
```