NXP Semiconductors

Application Note

Document Number: AN12195

Rev. 0, 06/2018

Implement Low-Power Audio on i.MX8M

1. Introduction

This document discusses about low power audio application design on i.MX8M. In Low power audio, the audio file is decoded on A53 core and played on M4 core, which is intend to an low power application on i.MX8M platform.

2. Chip overview

This section provides main features of i.MX 8M chips.

2.1. i.MX8M chip overview

The i.MX 8M family is a set of NXP products focused on delivering the latest and greatest video and audio experience combining state-of-the-art media-specific features with high-performance processing while optimized for lowest power consumption.

- Built in TSMC 28HPC to achieve both high-performance and low-power consumption.
- Relies on a powerful fully coherent core based on a quad Cortex-A53 cluster. Graphics processing handled by the GC7000Lite GPU from Vivante supporting the latest graphic APIs.
- Advanced security modules for secure boot, cipher acceleration and DRM support.

Contents

- General purpose Cortex-M4 processor for low- power processing.
- A wide range of audio interfaces including I2S, AC97, TDM and S/PDIF.
- Large set of commonly used peripherals in consumer/industrial markets including USB 3.0, PCIe and Ethernet.

3. Low Power Audio Application Overview

In this application, A53 core will send request to CM4 core and then sleep. CM4 core play or record audio independently, and wakeup A53 core when the operation is done.

3.1. Components

Five software components are created to implement this feature:

- Linux play & record application
- Linux RPMSG character device driver
- Linux Remote core share memory driver
- FreeRTOS Audio Server

Figure 1 shows how a low power application works:

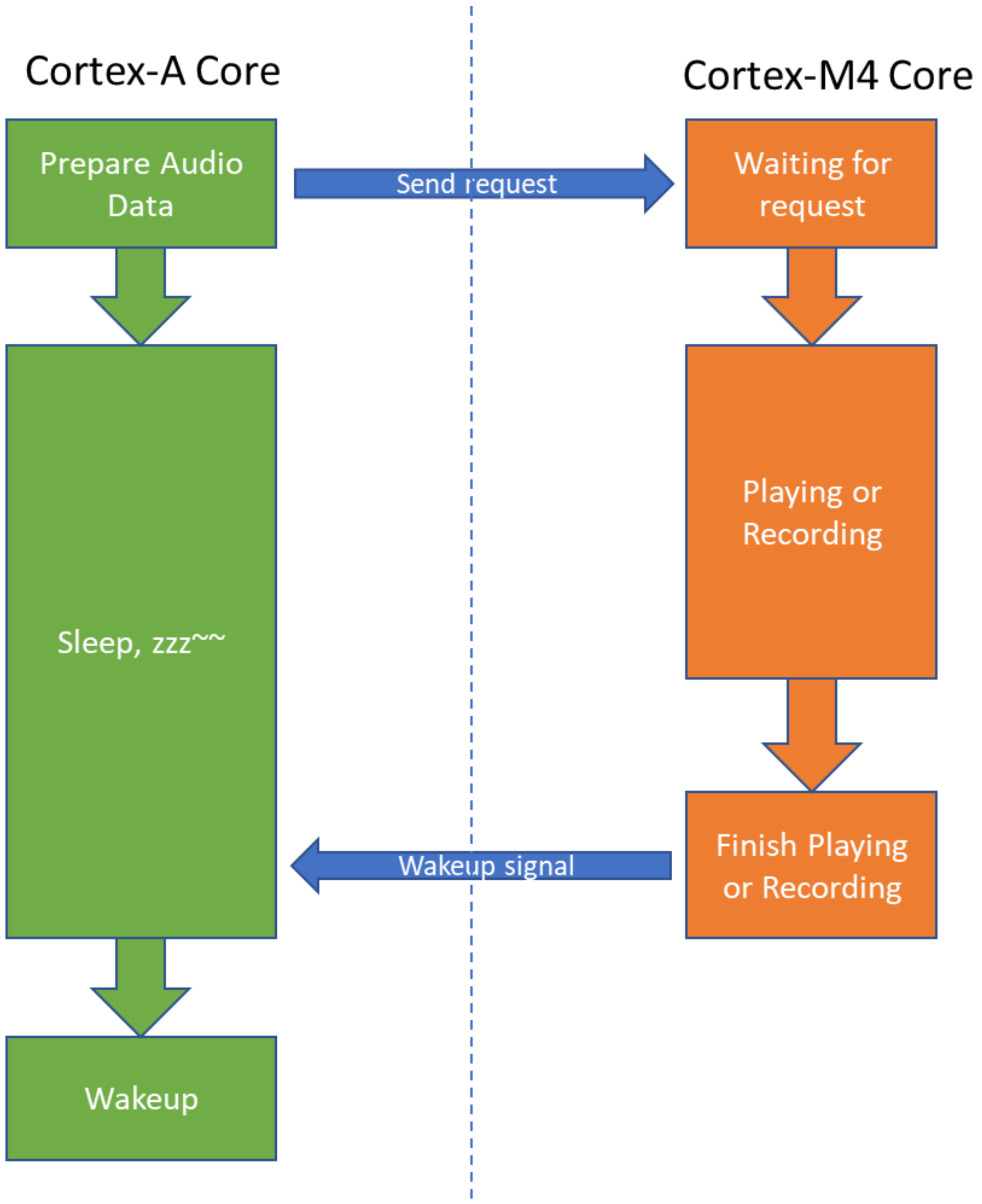


Figure 1. Low Power Audio Procedure

4. Software design

4.1. RPMSG character device driver

On i.MX8, cores use RPMSG mechanism to communicate messages. RPMSG is implemented based on MU module.For more details about RPMSG, please refer to below URL:

https://github.com/OpenAMP/open-amp/wiki/OpenAMP-RPMsg-Virtio-Implementation

We implement a RPMSG character device driver based on below considerations:

1. Currently, Linux kernel in i.MX specifically supports I2S RPMSG, TTY RPMSG, PMC RPMSG drivers. There's no common driver to transfer messages between cores.
2. To handle various user cases on M4 core and reduce effort to develop specific RPMSG driver.
3. Normally, there is a message protocol between kernel and M4 core application. **With this common driver, we can implement the protocol in user space, instead of kernel driver.**

Driver descriptions:

- When probed, create a /dev/rpmsg_ctrl0 device.
- User can open /dev/rpmsg_ctrl0 and use ioctl command RPMSG_CREATE_EPT_IOCTL to create an RPMSG End Point device, e.g. /dev/rpmsg0.
- The end-point device can accept open, read, write, poll and close operations.
- User can use ioctl command RPMSG_DESTROY_EPT_IOCTL to destroy an end-point device.

4.2. Remote core share memory driver

There're two considerations for a shared memory space between cores:

1. **Contiguous memory space.**
2. **Memory physical address.**

In Linux,

- Malloced memory are remapped to process's address space. So, we can only get a virtual address. But M4 core can't recognize a virtual address.
- DDR memory are organized by pages, which is buffer-able, cache-able and not contiguous. But M4 core don't have such mechanism to handle this memory space.

Thus, in low-power audio, Linux need to create a non-buffer-able, non-cache-able and contiguous memory to store decoded audio data. Only such memory space can be handled by M4 core.

Also, Linux need to get the physical address of the memory and pass it to M4 core.

Normally, in Linux, we have two ways to create such memory space.

1. Use DMA coherent memory space, which is using CMA (Contiguous Memory Allocator).
2. Predefined a reserved memory space in DTS file.

Here we recommend using DMA memory space. It is more flexible. We don't need to allocate a reserved memory space and we can free the allocated space. Also, we can get the physical address.

Function *dma_alloc_coherent()* is an ideal function for such case.

Based on above, we create a remote core share memory driver, which can help user to allocate, read and write a DMA coherent memory space.

Driver descriptions:

- Create a /dev/rmtcore_shm character device.
- Accept open, read, write, seek, ioctl and close operations.
- By default, the driver will create a 1MB contiguous memory space. Use can use ioctl command RMTCORE_SHM_CHG_BUF_SIZE to get a new buffer with new size.
- User can use ioctl command RMTCORE_SHM_GET_BUF_ADDR_PHY to get buffer physical address.
- Two additional ioctl commands are RMTCORE_SHM_GET_BUF_ADDR_VIRT and RMTCORE_SHM_GET_BUF_SIZE, which can be used to get virtual buffer address and buffer size.

4.3. AMP Optimization

By default, when M4 is detected, kernel enables all modules' clock gates. So SOC current will be very HIGH. On i.mx8M, the current is around 500 mA.

So in kernel, we need to enable clocks required byM4 program and gate all other modules' clocks.

4.4. Linux low power audio application design

The main diagram shows relationship between low power audio modules.

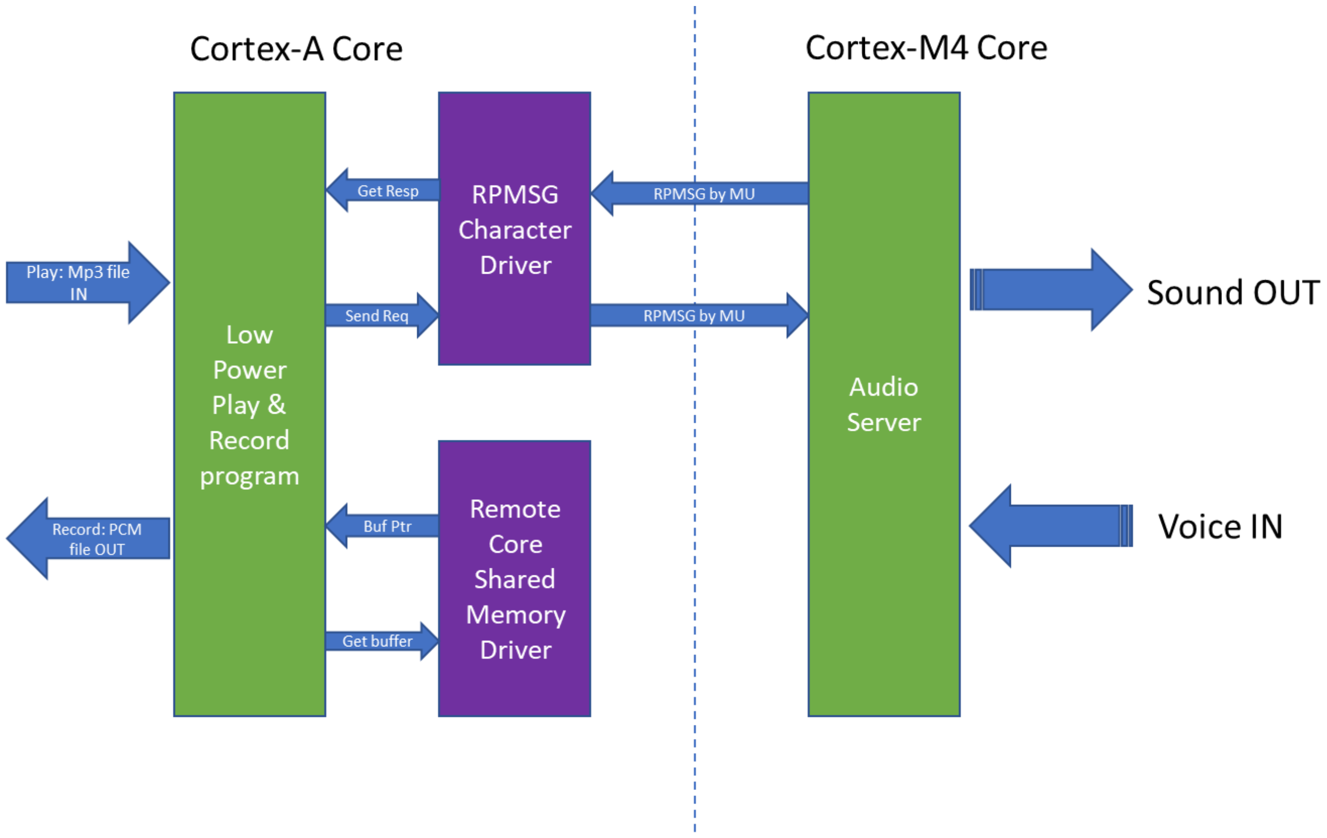


Figure 2. Data Exchange Diagram

4.4.1. Low-Power Play

The low-power play program will decode an mp3 file and play the file on M4 core.

In the program, we will:

1. Decode a mp3 audio file.
2. Copy decoded data to shared memory space.
3. Send physical address of allocated buffer to M4 core.
4. Get to sleep and wait for wakeup signal.
5. When playing is finished, wakeup from MU signal.

4.4.2. Low-Power Record

The low-power record program will receive the voice on M4 core and save to PCM raw data on ‘A’ core.

The program will:

1. Allocate enough shared memory space.
2. Send physical address of allocated buffer to M4 core.
3. Get to sleep and wait for wakeup signal.
4. When record is finished, wakeup from MU signal.
5. Get audio data from shared memory space and save to file.

4.4.3. Decode a mp3 audio file

In this program, we use gstreamer to decode a mp3 file to PCM raw data.

The gstreamer command to decode an mp3 file is:

gst-launch-1.0 filesrc location=xxxx.mp3 ! mpegaudioparse ! beepdec ! filesink location=xxxx.pcm

User can then write code based on the gstreamer command.

NOTE

This is just an example of decoding audio file to raw data, for other file types, user can choose any ways to get the PCM raw data.

In the example code, we suppose that the sample rate, channel information and sample format are known. In a real application, user may need to use other ways, e.g. libmad to decode the mp3 header to get these parameters.

4.4.4. Copy decoded data to shared memory space

Here we’ll use remote core shared memory driver to copy PCM raw data to a contiguous memory space.

For code example, please refer to function *pcm_write_to_cma_buffer()* in lp_play.c.

NOTE

As on i.MX8, kernel CMA area is very large, about 1G size. In this example, we allocate memory space for whole PCM raw data. In user programs, if memory is limited, user can use other mechanism, like circle buffer, ping-pong, etc.

4.4.5. Send physical address of allocated buffer to M4 core.

Use RPMSG character device to send physical address of buffer to M4 core.

Here we emphasize that **normally, there should be a protocol between A core and M4 core.** With RPMSG character device, we can implement the protocol in user space.

For example, in low-power play program, to access an audio device on remote M4 core, we create an audio request and response protocol.

Code snippet:

```
/* Open audio device */
    memset(&audio_msg_req, 0, sizeof(audio_rpmsg_request_t));
    audio_msg_req.header.category = SRTM_AUDIO_CATEGORY;
    audio_msg_req.header.majorVersion = (__u8)((SRTM_AUDIO_VERSION & 0xFF00U) >> 8U);;
    audio_msg_req.header.minorVersion = (__u8)(SRTM_AUDIO_VERSION & 0xFFU);;
    audio_msg_req.header.type = SRTM_MESSAGE_TYPE_REQUEST;
    audio_msg_req.header.command = SRTM_AUDIO_SERV_REQUEST_CMD_TX_OPEN;
    audio_msg_req.header.priority = 0;
    rtn_bytes = write(rpmsg_ep_fd, &audio_msg_req, sizeof(audio_rpmsg_request_t));
    if (rtn_bytes != sizeof(audio_rpmsg_request_t)) {
        fprintf(stderr, "Not all request msg data transmitted or send failed: %ld\n",
rtn_bytes);
        return -1;
}


    /* Read response */
    memset(&audio_msg_resp, 0, sizeof(audio_rpmsg_response_t));
    rtn_bytes = read(rpmsg_ep_fd, &audio_msg_resp, sizeof(audio_rpmsg_response_t));
    if (rtn_bytes != sizeof(audio_rpmsg_response_t)) {
        fprintf(stderr, "Not all resp msg data received or read failed: %ld\n", rtn_bytes);
        return -1;
    }
    if (SRTM_AUDIO_SERV_REQUEST_CMD_TX_OPEN == audio_msg_resp.header.command) {
        if (audio_msg_resp.param.result[1])
            fprintf(stderr, "Got TX_OPEN response msg! Failed!\n");
        else
            fprintf(stderr, "Got TX_OPEN response msg! PASS!!\n");
    }
```

For details, refer to *pcm_send_to_remote()* function in lp_play.c or lp_record.c.

4.4.6. Linux kernel get to sleep

The command to enter suspend mode in Linux is:

echo mem > /sys/power/state

We still need to use the command in code.

There’re two ways in user space to bring kernel to suspend mode (or standby mode).

1. Use *system()* function. E.g.

   system("echo mem > /sys/power/state");

2. Use *popen()* function.

   Please refer to *pcm_sleep_and_wakeup()* function in lp_play.c or lp_record.c.

4.4.7. Linux kernel wakeup

As in the Arm Trusted Firmware (ATF) patch, we have added MU interrupt as wakeup source. When kernel get MU interrupt from M4 core, it will wake up.

4.5. Audio Server on Cortex-M4 core

An audio server example is created on the Cortex-M4 core. The audio server follows the protocols defined between A core low-power applications to do playing or recording.

NOTE

Notes for M4 core applications:

- M4 core application also uses virtual queue mechanism. The virtual queue used in M4 core application is created by kernel virtual queue driver, So M4 core application need to be launched before kernel. Only by this, can a RPMSG device be created.
- Modules used in M4 core should be disabled in kernel DTS file. Refer to fsl-imx8mq-evk-lpaud.dts for reference.
- To work with the ATF patch, M4 core application need to enable clock of SIM, SIM_MAIN, SIM_S, SIM_WAKEUP, DEBUG, DRAM and SEC_DEBUG.
- Modules using these clocks are shared between M4 and A53. So M4 application need to enable it in its domain. Or, these clocks will be gated when A53 enters suspend mode.
- See BOARD_BootClockRUN() in clock_config.c of audio_server example for details.

5. Run low-power audio example

This chapter describe the procedure to run the low power audio example.

We suppose that the user has already followed YOCTO user guide to build an SD image and created an SD card for booting.

5.1. Apply low-power patch for ATF

The ATF has implemented **Power State Coordination Interface** (**PSCI**) and the low-power feature is in it.

In Linux kernel of default BSP release, it will bring both Cortex-A53 core and Cortex-M4 core to suspend mode. So a low-power patch is needed in ATF.

The ATF patch will prevent DDR from entering retention and don’t disable PLLs.

Follow below instructions to rebuild a flash.bin and program it to SD card (if you’re using SD boot).

5.1.1. Clone relevant repositories

Relevant repositories and files are:

Name	Clone address	Branch
arm-trusted-firmware	https://source.codeaurora.org/external/imx/imx-atf	imx_4.9.51_imx8m_ga
imx-mkimage	https://source.codeaurora.org/external/imx/imx-mkimage	imx_4.9.51_imx8m_ga
linux-firmware-imx	https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-7.2.bin	
uboot-imx	https://source.codeaurora.org/external/imx/uboot-imx	imx_v2017.03_4.9.51_imx8m_ga
Linux-imx	https://bitbucket.sw.nxp.com/scm/imx/linux-imx.git	imx_4.9.51_imx8m_ga

5.1.2. Setup aarch64 toolchain

Follow the yocto user guide in BSP release package to build a aarch64 toolchain.

URL: https://www.nxp.com/webapp/Download?colCode=L4.9.51_8MQ_BETA_LINUX_DOCS

5.1.3. Build uboot-imx and copy binaries

Command:

```
make ARCH=arm imx8mq_evk_defconfig
make ARCH=arm -j8
```

After building finished, copy mkimage, u-boot-spl.bin, u-boot-nodtb.bin, fsl-imx8mq-evk.dtb to imx-mkimage/iMX8M/ and rename to mkimage_uboot.

Command:

```
cp ${SRC_UBOOT_DIR}/tools/mkimage ${TARGET_MKIMG_DIR}/mkimage_uboot
cp ${SRC_UBOOT_DIR}/spl/u-boot-spl.bin ${TARGET_MKIMG_DIR}/
```

cp ${SRC_UBOOT_DIR}/u-boot-nodtb.bin ${TARGET_MKIMG_DIR}/

cp ${SRC_UBOOT_DIR}/arch/arm/dts/fsl-imx8mq-evk.dtb ${TARGET_MKIMG_DIR}/

5.1.4. Apply low-power patch and build bl31.bin

Command:

git apply ./0001-Patch-for-low-power-application.patch

make PLAT=imx8mq bl31

After build, copy bl32.bin to imx-mkimage.

Command:

cp ${SRC_ATF_DIR}/build/imx8mq/release/bl31.bin ${TARGET_MKIMG_DIR}/

5.1.5. Unpack firmware binary

Command:

1. *wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-7.2.bin*
2. *chmod +x firmware-imx-7.2.bin*
3. *./firmware-imx-7.2.bin*
4. *cd firmware-imx-7.2/firmware/ddr/synopsys*
5. *cp ${SRC_FIRMWARE_DIR}/firmware/hdmi/cadence/signed_hdmi_imx8m.bin ${TARGET_MKIMG_DIR}/*
6. *cp ${SRC_FIRMWARE_DIR}/ddr/lpddr4_pmu_train_1d_dmem.bin ${TARGET_MKIMG_DIR}/*
7. *cp ${SRC_FIRMWARE_DIR}/ddr/lpddr4_pmu_train_1d_imem.bin ${TARGET_MKIMG_DIR}/*
8. *cp ${SRC_FIRMWARE_DIR}/ddr/lpddr4_pmu_train_2d_dmem.bin ${TARGET_MKIMG_DIR}/*
9. *cp ${SRC_FIRMWARE_DIR}/ddr/lpddr4_pmu_train_2d_imem.bin ${TARGET_MKIMG_DIR}/*

5.1.6. Build final flash.bin

After all above operations, enter imx-mkimage and build flash.bin.

Command:

make SOC=iMX8M flash_hdmi_spl_uboot

5.1.7. Program flash.bin to SD card

Program flash.bin to SD card (if SD card is the boot device).

Command:

dd if=iMX8QM/flash.bin of=/dev/<your device> bs=1k seek=33

5.2. Rebuild kernel with new drivers

In kernel, apply RPMSG character device driver and remote core shared memory driver patches.

Command:

git apply ./0001-Add-character-device-driver-for-rpmsg.patch

git apply ./0002-Add-remote-core-share-memory-driver.patch

Rebuild kernel

make ARCH=arm64 defconfig

make ARCH=arm64 -j8

When kernel image is built out, copy it to SD card.

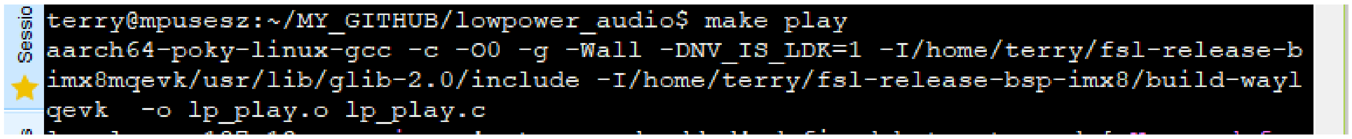
5.3. Build low-power audio applications

In low-power audio directory, use below command to make it.

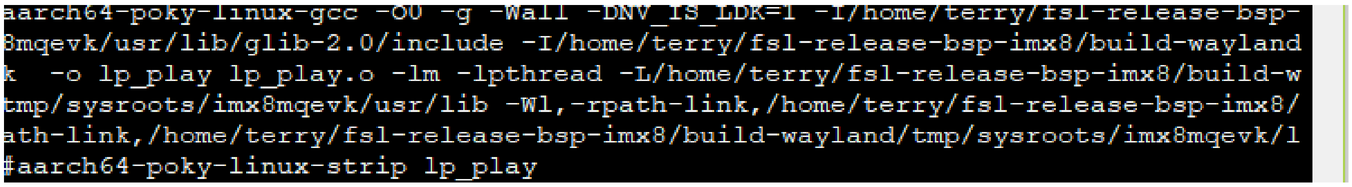
NOTE

The cross-compiler toolchain need to ready.

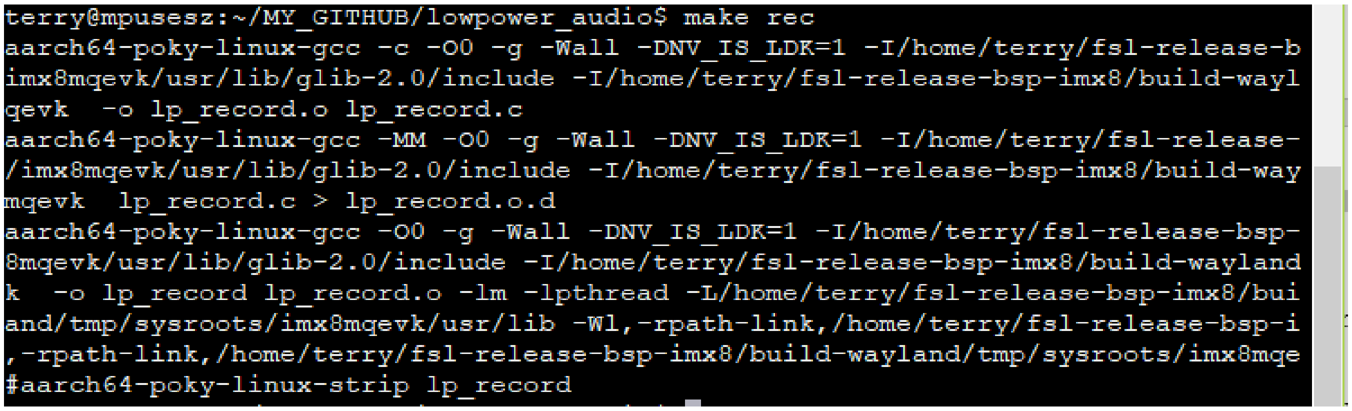
Use “make play” and “make rec” to build lp_play and lp_record applications.



```
terry@mpusesz:~/MY_GITHUB/lowpower_audio$ make play
aarch64-poky-linux-gcc -c -O0 -g -Wall -DNV_IS_LDK=1 -I/home/terry/fsl-release-b
imx8mqevk/usr/lib/glib-2.0/include -I/home/terry/fsl-release-bsp-imx8/build-wayl
qevk  -o lp_play.o lp_play.c
```



```
aarch64-poky-linux-gcc -O0 -g -Wall -DNV_IS_LDK=1 -I/home/terry/fsl-release-bsp-
8mqevk/usr/lib/glib-2.0/include -I/home/terry/fsl-release-bsp-imx8/build-wayland
k  -o lp_play lp_play.o -lm -lpthread -L/home/terry/fsl-release-bsp-imx8/build-w
tmp/sysroots/imx8mqevk/usr/lib -Wl,-rpath-link,/home/terry/fsl-release-bsp-imx8/
ath-link,/home/terry/fsl-release-bsp-imx8/build-wayland/tmp/sysroots/imx8mqevk/l
#aarch64-poky-linux-strip lp_play
```

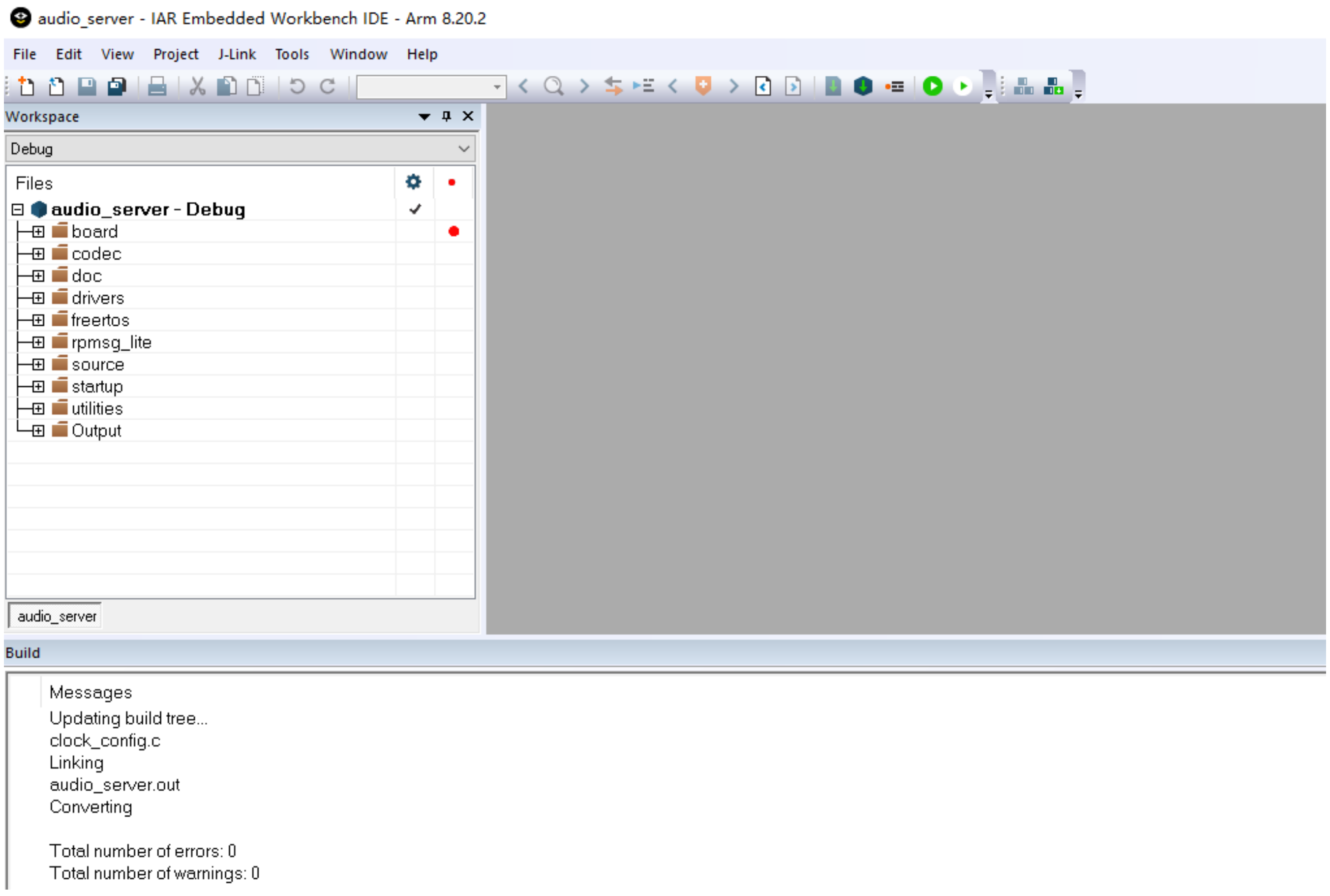


```
terry@mpusesz:~/MY_GITHUB/lowpower_audio$ make rec
aarch64-poky-linux-gcc -c -O0 -g -Wall -DNV_IS_LDK=1 -I/home/terry/fsl-release-b
imx8mqevk/usr/lib/glib-2.0/include -I/home/terry/fsl-release-bsp-imx8/build-wayl
qevk  -o lp_record.o lp_record.c
aarch64-poky-linux-gcc -MM -O0 -g -Wall -DNV_IS_LDK=1 -I/home/terry/fsl-release-
/imx8mqevk/usr/lib/glib-2.0/include -I/home/terry/fsl-release-bsp-imx8/build-way
mqevk  lp_record.c > lp_record.o.d
aarch64-poky-linux-gcc -O0 -g -Wall -DNV_IS_LDK=1 -I/home/terry/fsl-release-bsp-
8mqevk/usr/lib/glib-2.0/include -I/home/terry/fsl-release-bsp-imx8/build-wayland
k  -o lp_record lp_record.o -lm -lpthread -L/home/terry/fsl-release-bsp-imx8/bui
and/tmp/sysroots/imx8mqevk/usr/lib -Wl,-rpath-link,/home/terry/fsl-release-bsp-i
,-rpath-link,/home/terry/fsl-release-bsp-imx8/build-wayland/tmp/sysroots/imx8mqe
#aarch64-poky-linux-strip lp_record
```

5.4. Build Cortex-M4 core application

The Cortex-M4 application is developed by IAR.

So, install IAR and open the audio server example. Built it.

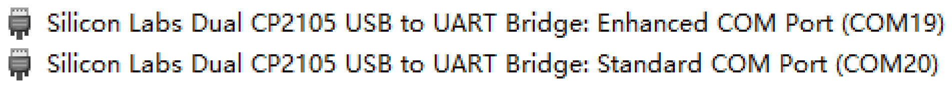


```
Messages
Updating build tree...
clock_config.c
Linking
audio_server.out
Converting

Total number of errors: 0
Total number of warnings: 0
```

After build, copy audio_server.bin to SD card.

5.5. Setup windows environment

When the USB cable is inserted to iMX8MQ EVK board and PC, there should be two uart devices discovered.


Silicon Labs Dual CP2105 USB to UART Bridge: Enhanced COM Port (COM19)
Silicon Labs Dual CP2105 USB to UART Bridge: Standard COM Port (COM20)

The Enhanced COM port is for A core and the standard COM port is for M4 core.

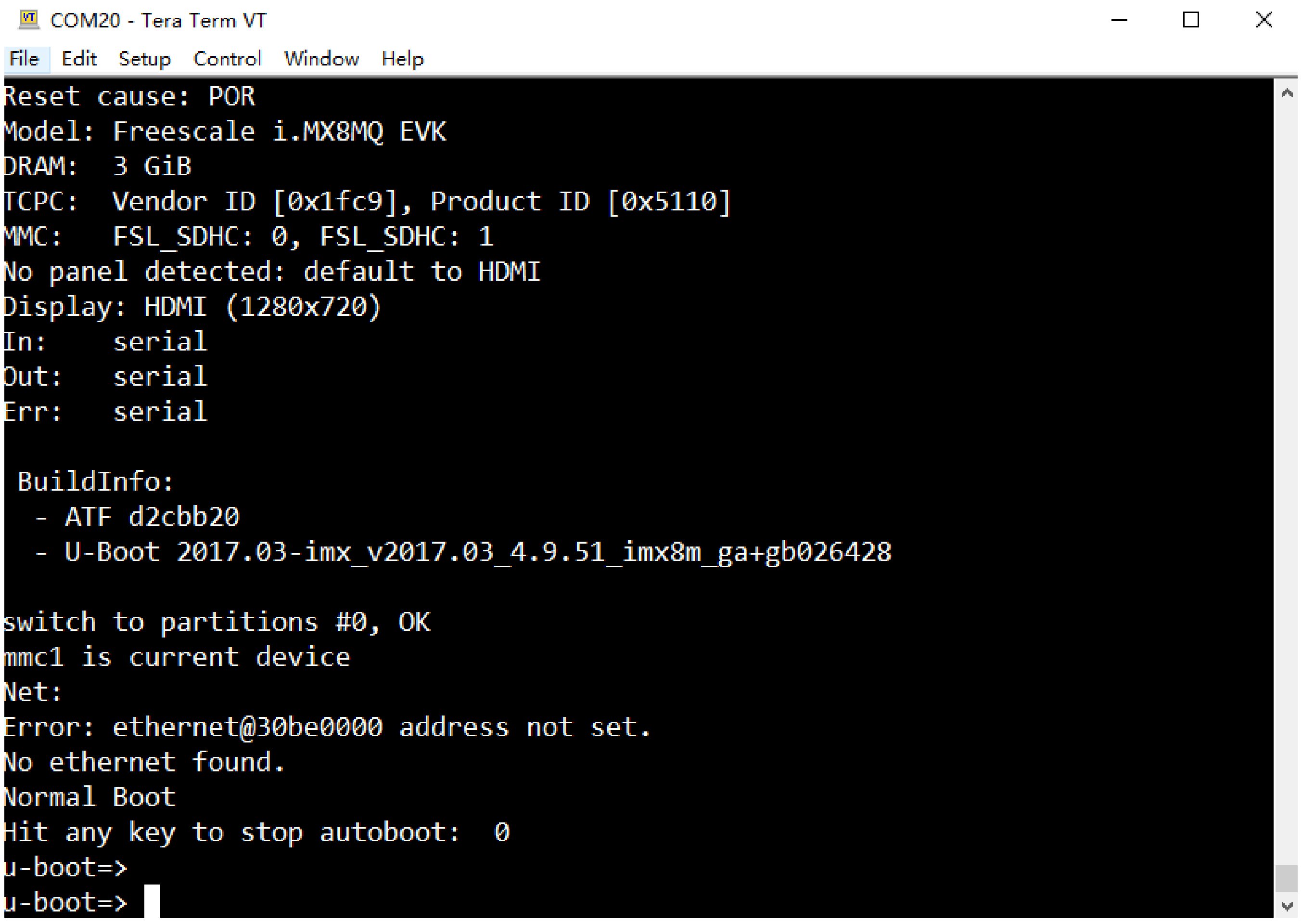
Open two consoles for these two COM port.

NOTE

Different computer will have a different COM port number.

5.6. Boot u-boot

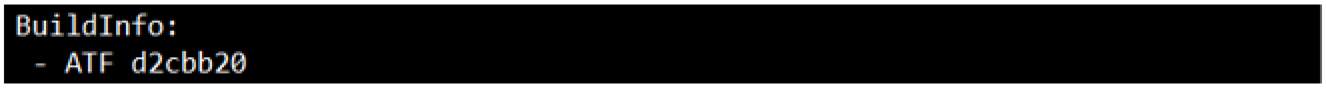
Insert SD card to board Switch SW701 to ON. The u-boot log appears on standard COM port. Press any key to stop the boot.



```
Reset cause: POR
Model: Freescale i.MX8MQ EVK
DRAM:  3 GiB
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110]
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
No panel detected: default to HDMI
Display: HDMI (1280x720)
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF d2cbb20
  - U-Boot 2017.03-imx_v2017.03_4.9.51_imx8m_ga+gb026428

switch to partitions #0, OK
mmc1 is current device
Net:
Error: ethernet@30be0000 address not set.
No ethernet found.
Normal Boot
Hit any key to stop autoboot:  0
u-boot=>
u-boot=>
```

NOTE

When u-boot starts, you can also check BuildInfo to confirm if the ATF patch is really loaded.



```
BuildInfo:
 - ATF d2cbb20
```

5.7. Run M4 core audio server first

Here we need to run M4 core audio server first.

Create macros in u-boot to run applications:

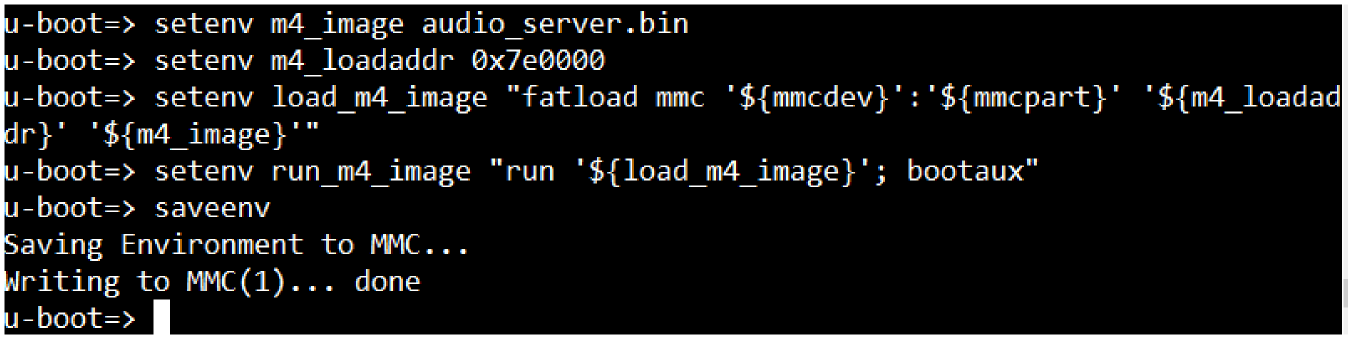
u-boot=> setenv m4_image audio_server.bin

u-boot=> setenv m4_loadaddr 0x7e0000

u-boot=> setenv load_m4_image "fatload mmc '${mmcdev}':'${mmcpart}' '${m4_loadaddr}' '${m4_image}'"

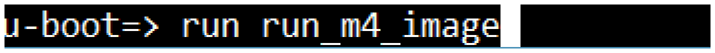
u-boot=> setenv run_m4_image "run load_m4_image; bootaux '${m4_loadaddr}'"

u-boot=> saveenv



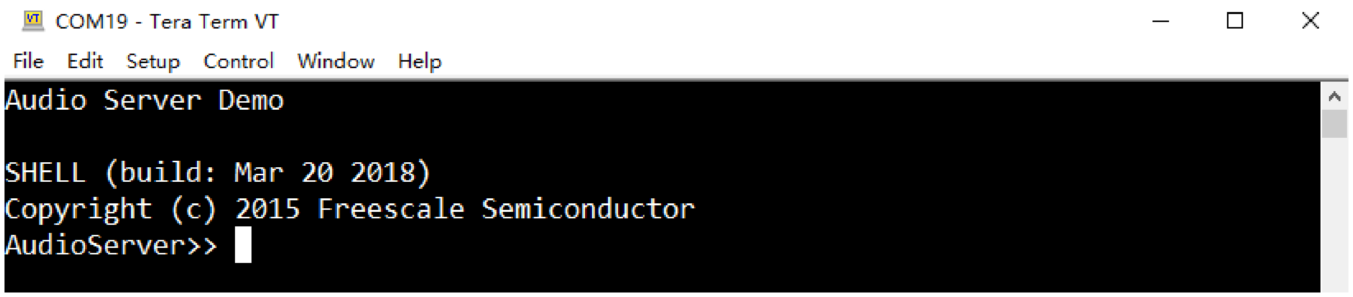
```
u-boot=> setenv m4_image audio_server.bin
u-boot=> setenv m4_loadaddr 0x7e0000
u-boot=> setenv load_m4_image "fatload mmc '${mmcdev}':'${mmcpart}' '${m4_loadad
dr}' '${m4_image}'"
u-boot=> setenv run_m4_image "run '${load_m4_image}'; bootaux"
u-boot=> saveenv
Saving Environment to MMC...
Writing to MMC(1)... done
u-boot=>
```

Run m4 image:



```
u-boot=> run run_m4_image
```

You should be able to see a shell on Enhanced COM port.

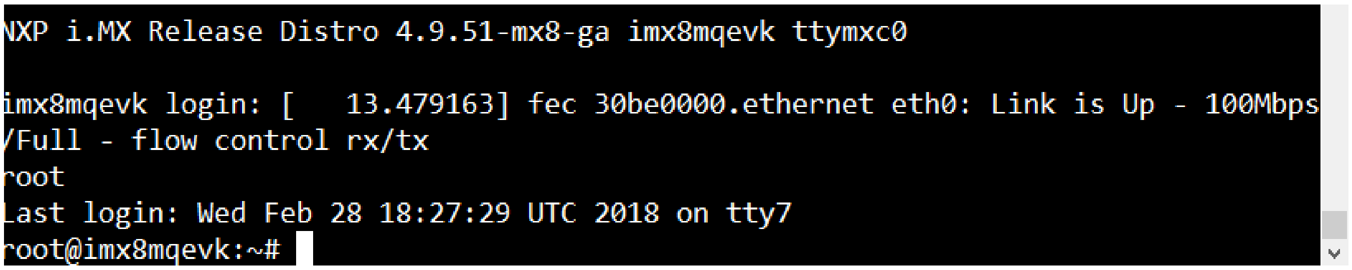


```
Audio Server Demo

SHELL (build: Mar 20 2018)
Copyright (c) 2015 Freescale Semiconductor
AudioServer>>
```

5.8. Run low-power audio applications

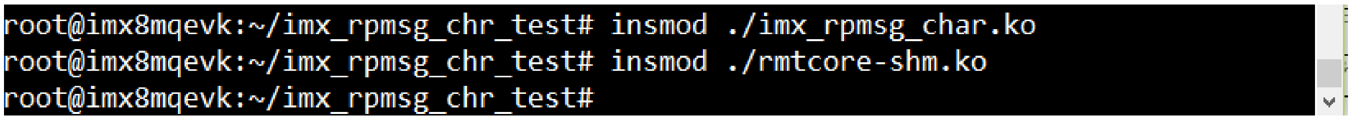
1. When audio server is running on M4 core, boot kernel in u-boot.

2. Use root to login



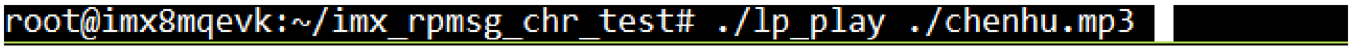
```
NXP i.MX Release Distro 4.9.51-mx8-ga imx8mqevk ttymxc0

imx8mqevk login: [   13.479163] fec 30be0000.ethernet eth0: Link is Up - 100Mbps
/Full - flow control rx/tx
root
Last login: Wed Feb 28 18:27:29 UTC 2018 on tty7
root@imx8mqevk:~#
```

3. Install RPMSG character device and Remote core shared memory drivers



```
root@imx8mqevk:~/imx_rpmsg_chr_test# insmod ./imx_rpmsg_char.ko
root@imx8mqevk:~/imx_rpmsg_chr_test# insmod ./rmtcore-shm.ko
root@imx8mqevk:~/imx_rpmsg_chr_test#
```

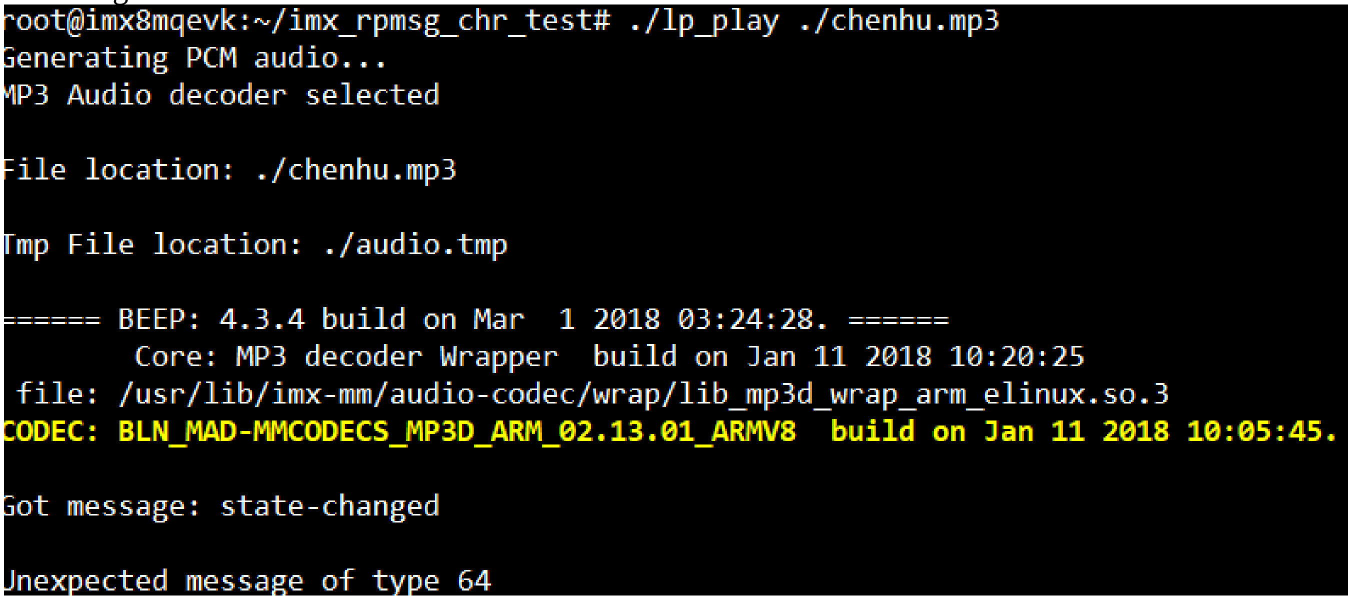
4. Run ./lp_play ./xxxx.mp3 to launch low power play.



```
root@imx8mqevk:~/imx_rpmsg_chr_test# ./lp_play ./chenhu.mp3
```

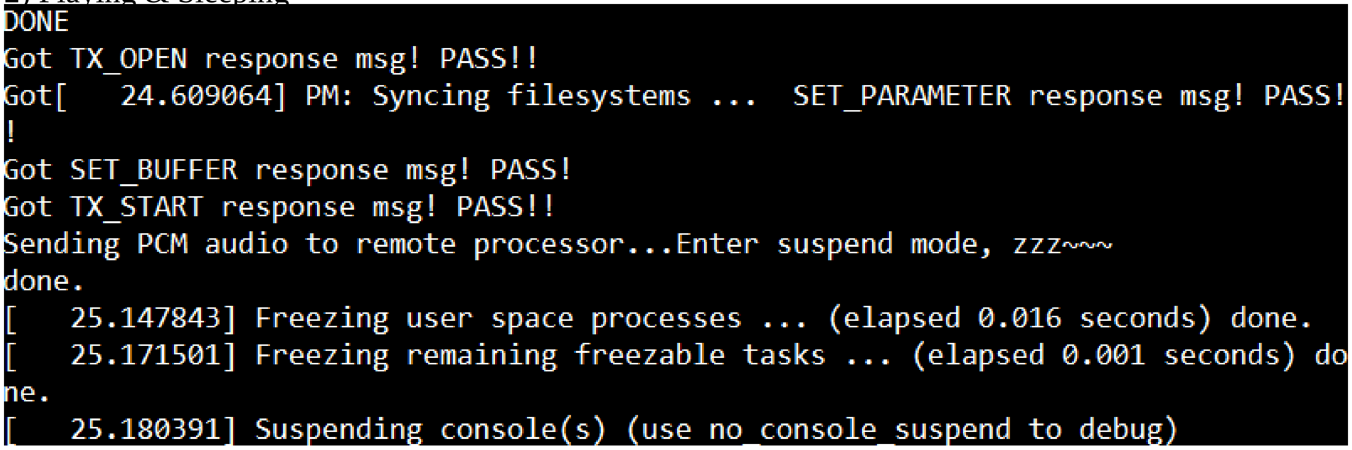
Message log:

1) Decoding:



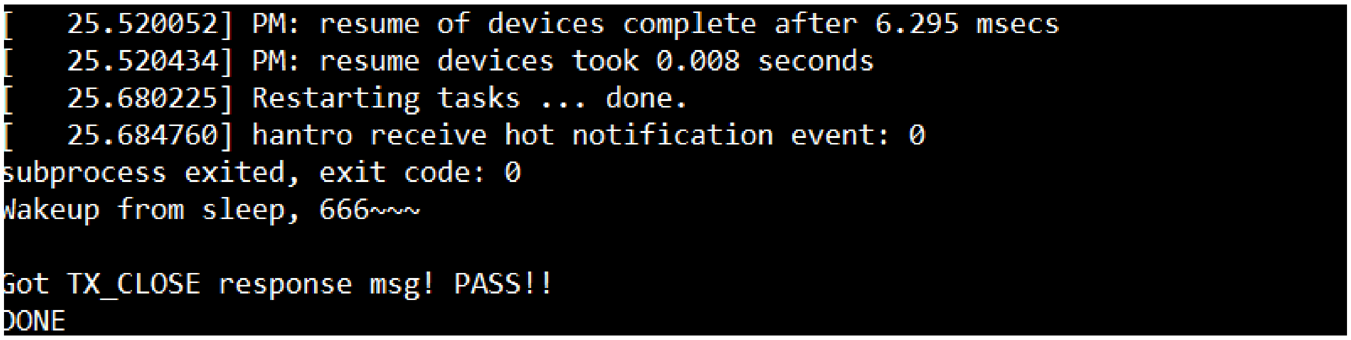
```
root@imx8mqevk:~/imx_rpmsg_chr_test# ./lp_play ./chenhu.mp3
Generating PCM audio...
MP3 Audio decoder selected

File location: ./chenhu.mp3

Tmp File location: ./audio.tmp

====== BEEP: 4.3.4 build on Mar  1 2018 03:24:28. ======
        Core: MP3 decoder Wrapper  build on Jan 11 2018 10:20:25
 file: /usr/lib/imx-mm/audio-codec/wrap/lib_mp3d_wrap_arm_elinux.so.3
CODEC: BLN_MAD-MMCODECS_MP3D_ARM_02.13.01_ARMV8  build on Jan 11 2018 10:05:45.

Got message: state-changed

Unexpected message of type 64
```

2) Playing & Sleeping



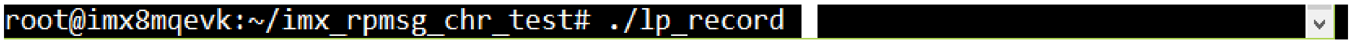
```
DONE
Got TX_OPEN response msg! PASS!!
Got[   24.609064] PM: Syncing filesystems ...  SET_PARAMETER response msg! PASS!
!
Got SET_BUFFER response msg! PASS!
Got TX_START response msg! PASS!!
Sending PCM audio to remote processor...Enter suspend mode, zzz~~~
done.
[   25.147843] Freezing user space processes ... (elapsed 0.016 seconds) done.
[   25.171501] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) do
ne.
[   25.180391] Suspending console(s) (use no_console_suspend to debug)
```

3) Wakeup



```
[   25.520052] PM: resume of devices complete after 6.295 msecs
[   25.520434] PM: resume devices took 0.008 seconds
[   25.680225] Restarting tasks ... done.
[   25.684760] hantro receive hot notification event: 0
subprocess exited, exit code: 0
Wakeup from sleep, 666~~~

Got TX_CLOSE response msg! PASS!!
DONE
```

5. Run ./lp_record to launch low power record. The recorded data will be saveed as audio_rec.tmp.



```
root@imx8mqevk:~/imx_rpmsg_chr_test# ./lp_record
```

6. Time consumption

As in low-power audio solution, the data will be decoded on A53 core and play on M4 core. There is some loss of time in protocol communication between cores. We'll try to calculate the loss of time in this solution here.

In this example, we'll play a 48 kbps, 22050 Hz, stero mp3 file in two ways, A53 core play directly and M4 core play via RPMSG. The duration from open the file using gstreamer to playback will be compared. Only playback is tested and user can get an evaluation on recording based on playback case. Here's the result:

Table 1. Time Consumption

Play method	Procedure	Duration
A53 Play Directly	GST Decode -> A53 Playback	116.410 ms

M4 Play using RPMSG	GST Decode to Shared Buffer -> Send Msg to M4 -> M4 Playback	467.560ms

As low-power audio playback will do additional operations, like decoding mp3 to file, allocating CMA buffer, coping audio data from file to buffer and then from buffer to CMA buffer, some additional time is needed, about 350 ms.

There can be some optimizations for low-power play additional operations, like using appsync in GST decoding, which will decode the mp3 data to a buffer directly, in this way, the additional time can be halved.

7. Power Consumption Test

This chapter compares the power consumption with using or without using low power audio. Only playback is tested and user can get an evaluation on recording based on playback case.

Normal playback:

Table 2. Normal playback

i.MX8MQ GA EVK dtb, no display, eth down, SDCard boot, idle						
	ARM	SOC	GPU	VPU	DRAM	NVC DRAM
Voltage(V)	0.896	0.877	0	0	0.992	1.096
Playback Current (mA)	65	205	0	0	135-446.4	70
Power (mW)	58.24	179.785	0	0	148.8 (Suppose average current is 150 mA)	71.24
Total Power (mW)	458.065					

Table 3. Low-power audio playback

i.MX8MQ GA EVK low power audio dtb, no display, eth down, SDCard boot, idle						
	ARM	SOC	GPU	VPU	DRAM	NVC DRAM
Voltage(V)	0.896	0.877	0	0	0.992	1.096

Playback Current (mA)	5	230	0	0	150 (Support DDR running at a low frequency point)	65
Power (mW)	4.48	201.74	0	0	148.8	71.24
Total Power (mW)	426.26					

NOTE

1. As kernel enters suspend mode when playing, we can see a current reduction from 65 mA to 5 mA on ARM power rail.
2. Compared with kernel play, M4 play will add about 25 mA to SOC. Note that M4 is running at 25 MHz.
3. A DDR optimization is needed so that DDR can continuerunning at a lower frequency when kernel enter suspend mode. Otherwise, the DDR current will be 450mA. Here we suppose that DDR is running at a low frequency when kernel enters Suspend mode.

8. Conclusion

This document mainly describes how to implement a low-power audio application on i.MX8M. This solution takes advantage of multicore architecture on i.MX8MQ and aims to provide an audio solution with less-power consumption. This solution can also be extended to other i.MX8 platforms.

9. References

- i.MX8MQ Reference Mannual
- RPMSG: https://github.com/OpenAMP/open-amp/wiki/OpenAMP-RPMsg-Virtio-Implementation

10. Revision history

Revision number	Date	Substantive changes
0	6/2018	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12195
Rev. 0
06/2018