Application Report
SPRAC10–February 2016

Tools and Techniques for Audio Debugging

Misael Lopez Cruz

ABSTRACT

Debugging audio issues can be a challenging task due to the dynamic nature of the audio systems, in terms of runtime routing flexibility, gain control, and so forth. Investigating the root cause of system level issues is a difficult task without the appropriate tools and techniques.

This document discusses some of the available tools for debugging audio issues. These tools span from TI-specific to generic Linux® or Android™ ones. The scope of this document is ALSA sound cards, although some tools are not directly tied to ALSA or could be adapted for non-ALSA sound cards.

Contents

List of Figures

Android is a trademark of Google.
Linux is a registered trademark of Linus Torvalds.
All other trademarks are the property of their respective owners.

Copyright © 2016, Texas Instruments Incorporated

1 Introduction

This document provides different tools and techniques useful for debugging audio issues encountered in Linux-based systems. The document is mostly focused on the Android operating system, but some of the techniques are also applicable to Linux systems.

The focus of the techniques presented in this document is the multichannel audio serial port (McASP), which is the audio peripheral in the DRA7xxx Infotainment Application Processor family. Most tools and techniques assume that McASP is accessed through the Linux ALSA framework.

2 ALSA procfs

procfs is a filesystem in Linux that provides information about the internal state of the kernel. It is mounted in `/proc`.

The ALSA procfs entries can be found at `/proc/asound/`. The most relevant entries for the DRA7xx audio are:

- `/proc/asound/cards`. Lists all cards registered in the system. This entry can be useful for:
  - Verifying if a sound card has been properly registered and is present in the system.
  - Finding the index of a given card. Card indexes are dynamically assigned as cards get registered in the system. Often times, user-space software assumes cards have certain indexes which could be different depending on runtime dynamics, such as when card dependencies are not met. For example, the index 0 could be assigned to card B instead of card A, if card A's registration is deferred due to a missing dependency (such as with a i2c codec that has not been probed yet).
- `/proc/asound/pcm`. Lists all PCM devices of all sound cards present in the system. This entry is useful to see the indexes and names of all PCM devices.
- `/proc/asound/cardX/pcmY[p,c]/*`. Each PCM device in all cards in the system has a procfs directory like this, where X is the card index number (such as from `/proc/asound/cards`) and Y in the PCM device index (such as from `/proc/asound/pcm`).

  This directory contains PCM device-related information and status:
  - *info*: General information about the PCM device, such as the card index, device index, stream direction, and so forth.
  - *hw_params*: When the PCM device is open, this procfs entry shows the channel count, sample rate, format, period size, and buffer size.

    The requested period and buffer sizes may differ from the ones finally granted. The `hw_params` entry can be used to query the actual parameters taken by the hardware:

```
shell@jacinto6evm:/ # tinyplay /data/media/0/Music/audio.wav -p1023 &
Playing sample: 2 ch, 44100 hz, 16 bit

shell@jacinto6evm:/ # cat /proc/asound/card0/pcm0p/sub0/hw_params
access: RW_INTERLEAVED
format: S16_LE
subformat: STD
channels: 2
rate: 44100 (44100/1)
period_size: 1024     <- Mismatch between requested (1023) and granted (1024)
buffer_size: 4096        period sizes due to ALSA hw_rules
```

  - *sw_params*: When the PCM device is open, this entry shows the start and stop thresholds, silence threshold, and so forth.

Copyright © 2016, Texas Instruments Incorporated

– *status*: When the PCM device is open, this entry shows the substream state (such as running, xrun, and so forth), number of available frames, hardware, and application pointers. The `hw_ptr` and `app_ptr` can identify cases where the underlying hardware is not able to transmit or receive any audio data. For example, if McASP is configured in slave mode and the master does not provide the BCLK/FSYNC clocks properly, the McASP transfers will stall. In that case, the `hw_ptr` and `appl_ptr` will have the same values when reading the status entry multiple times:

```
shell@jacinto6evm:/ # cat /proc/asound/card2/pcm0p/sub0/status
state: RUNNING
owner_pid   : 3587
trigger_time: 946693842.816186923
tstamp      : 946693842.816626936
delay       : 4128
avail       : 0
avail_max   : 0
-----
hw_ptr      : 64   <- hw_ptr reached AFIFO size, but didn't increase further
appl_ptr    : 4192 <- appl_ptr reached the ALSA buffer size, but didn't
                      increase further
```

3 McASP Register Settings

The McASP registers can be dumped or described with the omapconf tool. The omapconf tool is a Linux user-space stand-alone application designed to provide a quick and easy way to diagnose TI processors, including OMAP and DRA7xx. The source code and more information about omapconf can be found at https://github.com/omapconf.

The omapconf tool has support for McASP, which includes dumping all McASP registers of a given instance and high-level analysis of the McASP state.

```
shell@jacinto6evm:/ # audio-tool tone sine 1000 -r44100 &

shell@jacinto6evm:/ # omapconf show mcasp3

shell@jacinto6evm:/ # omapconf dump mcasp3
```

The high-level analysis and description of the McASP state given by omapconf is based on the McASP register values, and is grouped in sections similar to those in the McASP peripheral.

Copyright © 2016, Texas Instruments Incorporated

3.1 Data Ports and Buffers

This section groups the McASP register information about the port used to transfer data to and from McASP. When the DATA port is used, the AFIFOs can be used to store up to 64 audio words.

Data Ports and Buffers	
Port	CFG bus
Transmit DMA	
DMA request	Enabled
Status	No error
Receive DMA	
DMA request	Enabled
Status	No error
Transmit Buffer (XBUF)	
Status	No error
Receive Buffer (RBUF)	
Status	No error
Write FIFO (WFIFO)	
State	Disabled
Threshold	16 samples
Level	0 samples in FIFO
Read FIFO (RFIFO)	
State	Disabled
Threshold	16 samples
Level	0 samples in FIFO

Figure 1. Data Ports and Buffers

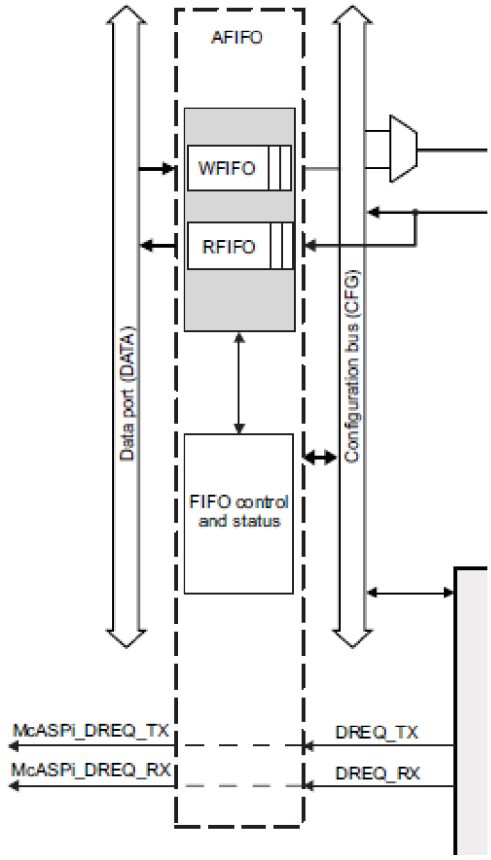

Figure 2. McASP Data Ports in omapconf

Copyright © 2016, Texas Instruments Incorporated

3.2 Control

This section groups the McASP register information about the enabled state machines and the enabled and active slots. For instance, Figure 3 shows that McASP was used for stereo (two enabled slots) audio playback (only TX FSM is enabled).

```
|--------------------------------------|
| Control                              |
|--------------------------------------|
| Transmit State-Machine |             |
|   State                | Active      |
| Transmit Sequencer     |             |
|   Enabled Slots        | 2           |
|   Active Slots         | 2           |
|   Active Slots Mask    | 0x00000003  |
|   Current Slot         | 0           |
| Receive State-Machine  |             |
|   State                | Held in reset |
| Receive Sequencer      |             |
|   Enabled Slots        | 2           |
|   Active Slots         | 2           |
|   Active Slots Mask    | 0x00000003  |
|   Current Slot         | Inactive    |
|--------------------------------------|
```

Figure 3. Control Dump

3.3 Clocks (BCLK)

This section describes the bit clock generation including the clock source selection, dividers, and polarity. Figure 4 shows that the bit clock is generated from AHCLK and that this clock is divided by 8.

```
|-------------------------------------------------|
| Clocks                                          |
|-------------------------------------------------|
| Transmit Bit Clock        |                     |
|   State                   | Running             |
|   Divider                 | Divide-by 8         |
|   Source                  | Internal            |
|   Polarity                | Driven on rising edge |
| Transmit High-Speed Clock |                     |
|   State                   | Running             |
|   Divider                 | Divide-by 1         |
|   Source                  | External (AHCLKX pin) |
|   Polarity                | Non-inverted        |
| Receive Bit Clock         |                     |
|   State                   | Held in reset       |
|   Divider                 | Divide-by 8         |
|   Source                  | Internal            |
|   Polarity                | Samples on falling edge |
| Sync Mode                 | Synchronous to TX   |
| Idle Mode                 | No-idle             |
|-------------------------------------------------|
```

Figure 4. Clocks

Copyright © 2016, Texas Instruments Incorporated

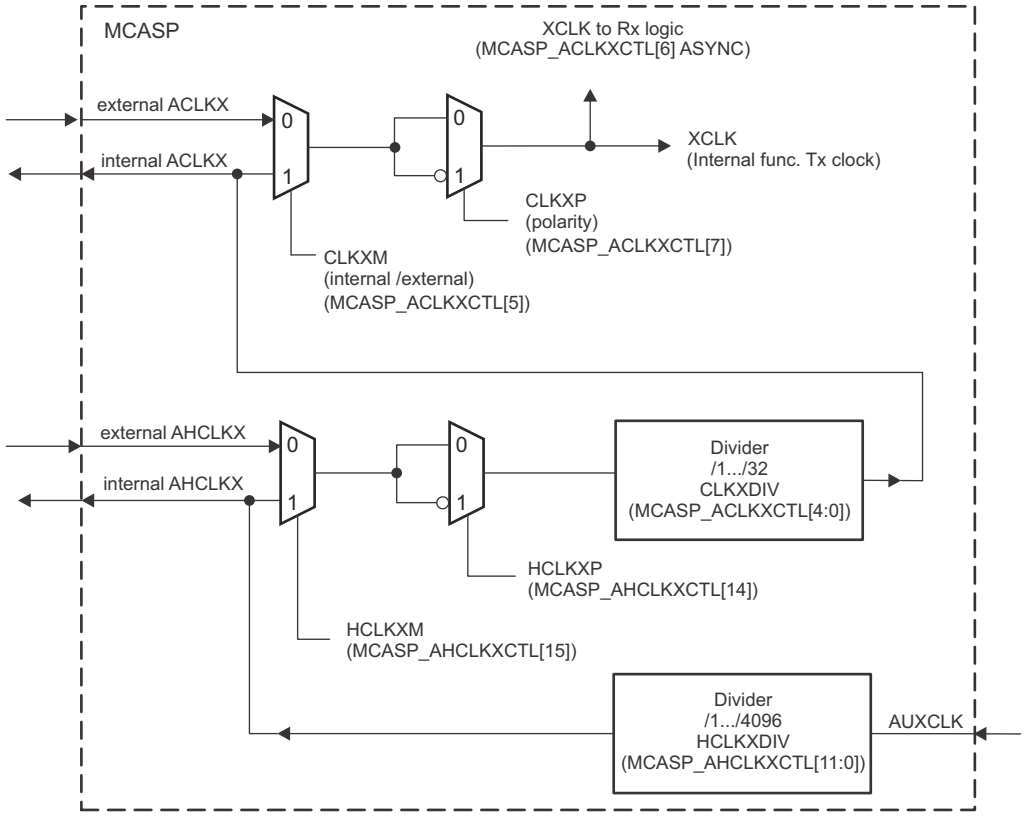

Figure 5. McASP Clocks in omapconf

3.4 Frame Sync Generator (FSYNC)

Information about the frame sync duration, polarity, delay, and clock source can be found in this section.

Figure 7 shows that the McASP frame sync generator is configured for two slots, no delay with respect to the start of the frame, 1-bit duration, and that the frame starts on the rising edge. This configuration is consistent with the DSP_B mode set by the driver.

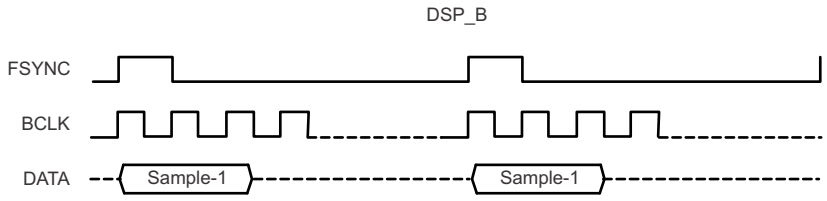

Figure 6. The DSP_B Format

Copyright © 2016, Texas Instruments Incorporated

```
|-------------------------------------------------|
| Frame Sync Generator                            |
|-------------------------------------------------|
| Transmit Frame Sync |                           |
|   Generator State   | Active                    |
|   Source            | Internal                  |
|   Polarity          | Frame starts on rising edge |
|   Pulse Width       | Single bit                |
|   Slot Count        | 2 (TDM)                   |
|   Data Delay        | 0-bit                     |
|   Status            | No error                  |
| Receive Frame Sync  |                           |
|   Generator State   | Held in reset             |
|   Source            | Internal                  |
|   Polarity          | Frame starts on rising edge |
|   Pulse Width       | Single bit                |
|   Slot Count        | 2 (TDM)                   |
|   Data Delay        | 0-bit                     |
|   Status            | No error                  |
| Sync Mode           | Synchronous to TX         |
|-------------------------------------------------|
```

Figure 7. Frame Sync Generator

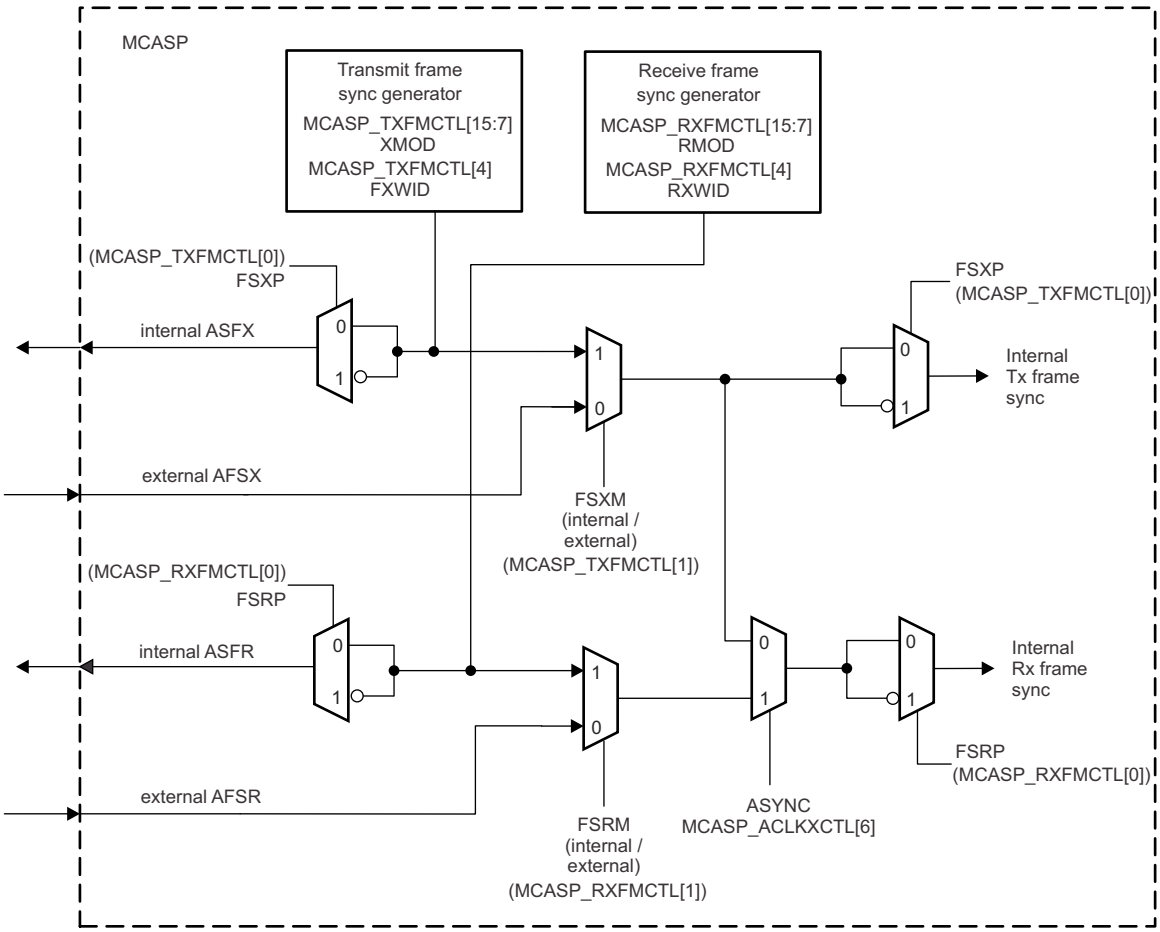

Figure 8. McASP Frame Sync Generator in omapconf

Copyright © 2016, Texas Instruments Incorporated

3.5 Format Units

The McASP format unit provides a flexible mechanism to remap and reformat the bits in the audio words. Figure 9 shows the settings applicable for a 16-bits/sample audio configuration.

Format Units	
Transmit Format Unit	
Slot Size	16 bits
Bit Mask	0x0000FFFF
Padding	Pad with 0
Right-Rotation	16 bit positions
Bitstream Order	MSB first
Receive Format Unit	
Slot Size	16 bits
Bit Mask	0x0000FFFF
Padding	Pad with 0
Right-Rotation	0 bit positions
Bitstream Order	MSB first

Figure 9. Format Units

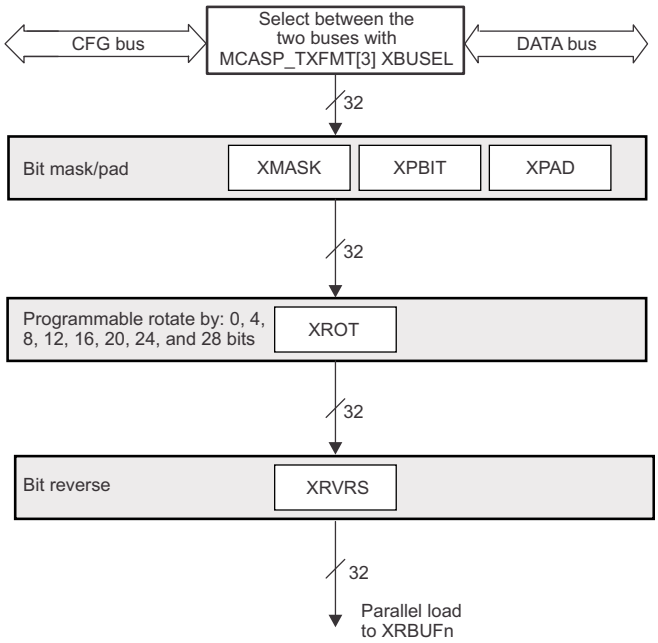

Figure 10. McASP Format Units in omapconf

3.6 Serializer

This section shows the mode (TX, RX, or inactive) and the inactive state (logic level in inactive slots) of each serializer. The McASP instance in Figure 11 uses two serializers: AXR0 for transmit and AXR1 for receive.

Serializers	
Transmit Serializers	Active
Receive Serializers	Cleared
Serializer 0	
Mode	Transmit
Inactive State	Logic Low
Serializer 1	
Mode	Receive
Inactive State	Hi-Z
Serializer 2	
Mode	Inactive
Inactive State	Hi-Z
Serializer 3	
Mode	Inactive
Inactive State	Hi-Z

Figure 11. Serializers

Copyright © 2016, Texas Instruments Incorporated

3.7 Pin Control

The McASP pins can be configured as McASP pins or GPIO pins. This pin control is internal to McASP and is not the same as the device level pin control.

Not all McASP instances have the same number of pins. McASP1 and 2 have 16 serializers, while the rest have 4 serializers.

Pin Control	
AFSR	
Functionality	Receive Frame Sync
Direction	Output
ACLKR	
Functionality	Receive Bit Clock
Direction	Output
AFSX	
Functionality	Transmit Frame Sync
Direction	Output
ACLKX	
Functionality	Transmit Bit Clock
Direction	Output
AHCLKX	
Functionality	Transmit High-Freq Clock
Direction	Input
AXR0	
Functionality	TX/RX Data Channel 0
Direction	Output
AXR1	
Functionality	TX/RX Data Channel 1
Direction	Input
AXR2	
Functionality	TX/RX Data Channel 2
Direction	Input
AXR3	
Functionality	TX/RX Data Channel 3
Direction	Input

Figure 12. Pin Control

Copyright © 2016, Texas Instruments Incorporated

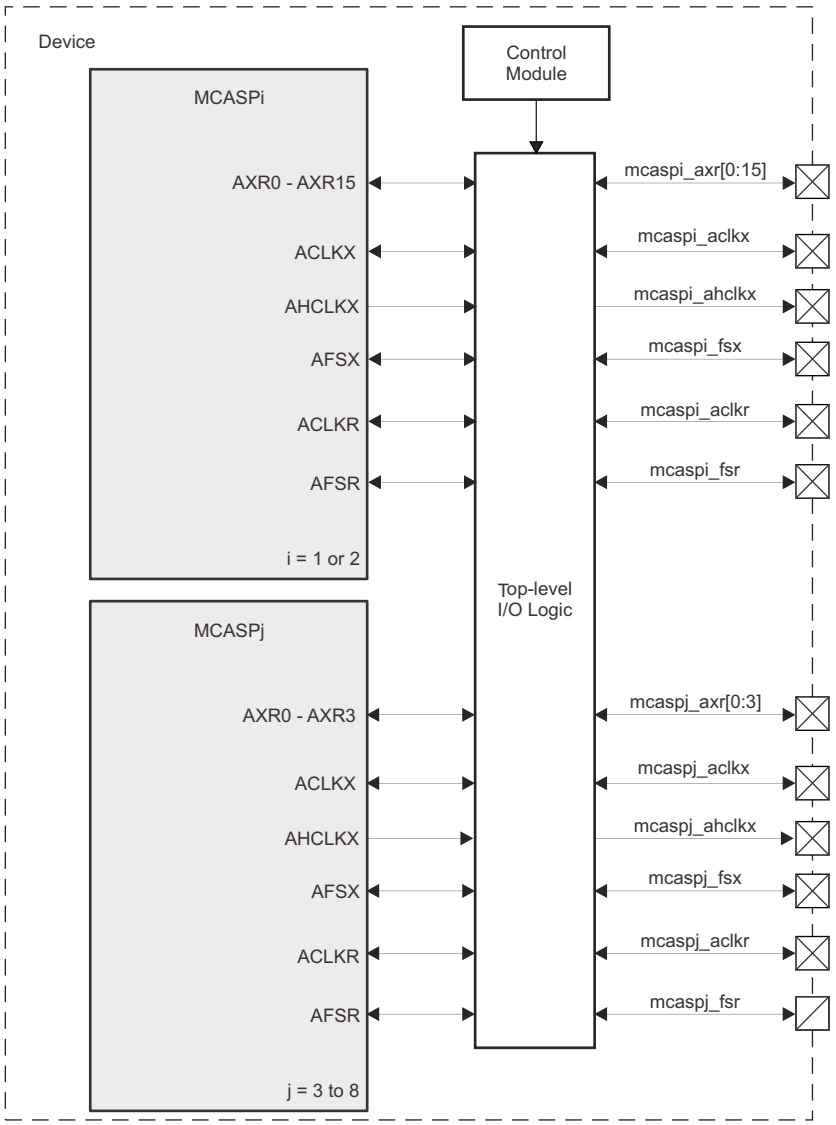

Figure 13. McASP Pin Control in omapconf

4 Overruns and Underruns (XRUNs)

XRUNs can occur due to different reasons; the most common ones are covered in this section.

4.1 Media Storage I/O Latency

Media storage I/O latency is probably the most common cause of xruns in single-threaded, non-optimized audio applications. Typically, the single thread in that kind of application sequentially accesses the media storage (to retrieve new samples for playback or store recorded audio) and the ALSA PCM device (to render samples for playback and get new samples in capture). Due to the sequential nature of that process, the ALSA buffer is more likely to underflow (playback) or overflow (capture) when the buffer is small or when the media I/O latency is high. Most test applications, including aplay, arecord, tinyplay, tinycap, and so forth, are single-threaded.

There are different ways to root cause this problem, from optimizing the audio application for media I/O latency to storing the audio sample files in a RAM disk.

Copyright © 2016, Texas Instruments Incorporated

4.1.1 Optimize the Audio Application

Having separate threads for accessing the media storage and reading and writing to the ALSA PCM device can help prevent xruns. An additional circular buffer in a faster memory (such as RAM) is required for this approach.

One thread could access the audio hardware:

- Playback: Read data from the circular buffer and write it to the ALSA.
- Capture: Read new samples from ALSA and write it to the circular buffer.

The other thread could operate in larger chunks of data (which is typically better in terms of media storage throughput):

- Playback: Read more audio samples from the media storage and write them to the circular buffer.
- Capture: Read new audio samples from the circular buffer and write them to the media storage.

4.1.2 Store Samples in a RAM Disk

Placing the audio samples in a RAM disk is a quick way to identify xruns caused by media storage I/O latency. While this is not a solution for the xruns, it helps to narrow down the root cause.

```
shell@jacinto6evm:/ # mkdir /data/audio_test

shell@jacinto6evm:/ # mount -t tmpfs -o size=20m tmpfs /data/audio_test

shell@jacinto6evm:/ # df
Filesystem               Size     Used     Free   Blksize
/dev                   498.5M    24.0K   498.5M   4096
/sys/fs/cgroup         498.5M    12.0K   498.5M   4096
/mnt/asec              498.5M     0.0K   498.5M   4096
/mnt/obb               498.5M     0.0K   498.5M   4096
/system                743.9M   321.4M   422.5M   4096
/factory                11.7M    40.0K    11.7M   4096
/cache                 248.0M   148.0K   247.8M   4096
/data                    6.0G   254.8M     5.8G   4096
/mnt/shell/emulated      6.0G   254.8M     5.8G   4096
/data/audio_test        20.0M     0.0K    20.0M   4096

shell@jacinto6evm:/ # cp /data/media/0/Music/audio.wav /data/audio_test

shell@jacinto6evm:/ # tinyplay /data/audio_test/audio.wav
Playing sample: 2 ch, 44100 hz, 16 bit

shell@jacinto6evm:/ # tinycap /data/audio_test/record.wav -r44100 -b16 -c2
Capturing sample: 2 ch, 44100 hz, 16 bit
```

4.1.3 Using Special Device Files

Similarly, xruns caused by the media storage latency could be identified or discarded by playing or capturing from special devices files, such as playing from `/dev/zero` (or `/dev/urandom`), recording to `/dev/null`.

```
# aplay /dev/urandom -f cd -d 60

# arecord /dev/null -f cd -d 60
```

These special device files do not introduce the latency that media storage does, such as recording `/dev/null` just discards all data written to it. This approach is best when the content of the recorded or played data is not the problem being debugged.

Copyright © 2016, Texas Instruments Incorporated

Android's tinyplay can not render from `/dev/urandom` or `/dev/zero`, because the utility only supports RIFF/WAVE files. This is not a limitation when recording to `/dev/null` using tinycap.

4.1.4 Rendering Patterns

Rendering predefined patterns is not precisely meant to debug xrun issues, but due to the fact that the application audio buffer is prefilled in advance with the pattern or filling the buffer is relatively fast, they typically do not introduce high latency.

Speaker-test and audio-tool are examples of this kind of applications.

```
shell@jacinto6evm:/ # audio-tool tone sine 1000 0 -r44100 -b16 -c2 -t60
```

```
root@dra7xx-evm:~# speaker-test -r 44100 -c 2 -t pink

speaker-test 1.0.27.2

Playback device is default
Stream parameters are 44100Hz, S16_LE, 2 channels
Using 16 octaves of pink noise
Rate set to 44100Hz (requested 44100Hz)
```

More information about the audio-tool can be found at https://github.com/audio-tool/audio-tool.

4.2 Interconnect Latency

The latency involved in transferring data to and from an audio peripheral (such as McASP) to the ALSA buffer can also cause overflows and underflows, which may or may not be reported as ALSA xruns.

McASP has one transmit buffer (TXBUF) and one receive buffer (RXBUF) for each serializer, and these buffers are one audio word deep. The service time for these buffers is very small, as the buffer must be written or read before one word is shifted in or out by the McASP serializer. Thus, the higher the sample rate or the number of channels or slots, the more stringent the latency requirement is. For more details on the McASP event service time, see the *DRA75x, DRA74x Technical Reference Manual* (SPRUHI2).

McASP also has an internal AFIFO that can be used to store up to 64 audio words. This additional FIFO can help relax the latency requirements. The McASP AFIFO must be accessed through eDMA for McASP1-3, and either sDMA or eDMA for McASP4-8. TI recommends enabling the McASP AFIFO, regardless of the sample rate or channel count, to prevent overflows and underflows that may occur when the system is highly loaded.

4.2.1 Measuring L3 Throughput

The throughput on the DRA7xx L3 interconnect can be measured with the omapconf tool. This measurement is useful to verify if the L3 bandwidth utilized by audio (such as sDMA) is within the expected limits, and to compare it against overall bandwidth utilization at the system level.

The L3 bandwidth measurement is done with respect to the EMIFs (external memory interfaces) in the DRA7xx processor. The relevant initiator for audio playback is sdma_rd, as the transfers are reads from the EMIF standpoint. sdma_wr is the initiator required to be monitored for audio capture.

The example below shows the L3 throughput measured in the Jacinto6 EVM with 10" display, during a use-case with 6-channels audio playback at 44.1 kHz, 16-bits. The expected audio throughput is (0.5292 MB/s).

The total throughput for the audio use-case is the one combined from the two EMIFs in the system. Thus, for the playback example, counter0 + counter1 = 0.53 MB/s is the measured L3 throughput.

Copyright © 2016, Texas Instruments Incorporated

This example also shows the L3 traffic by all other initiators (alldmm). In this case, most of the L3 traffic comes from the display panel 1280×800, 60 Hz (245.76 MB/s).

```
shell@jacinto6evm:/ # audio-tool tone sine 1000 -D3 -r44100 -b16 -c6 &
shell@jacinto6evm:/ # omapconf trace bw \
> --tr0 r+w --m0 sdma_rd \
> --tr1 r+w --m1 sdma_rd \
> --tr2 r+w --m2 sdma_wr \
> --tr3 r+w --m3 sdma_wr \
> --tr4 r+w --m4 alldmm \
> --tr5 r+w --m5 alldmm
...
Counter: 0  Master:         sdma_rd Transaction: r+w Probe: emif1
Counter: 1  Master:         sdma_rd Transaction: r+w Probe: emif2
Counter: 2  Master:         sdma_wr Transaction: r+w Probe: emif1
Counter: 3  Master:         sdma_wr Transaction: r+w Probe: emif2
Counter: 4  Master:          alldmm Transaction: r+w Probe: emif1
Counter: 5  Master:          alldmm Transaction: r+w Probe: emif2
delay in us: 1000000
overflow delay in us: 1000000 (iterations=1)
accumulation type: 2
iterations (0=infinite): 0
Overflow counter index: DISABLED (overflow delay used)
Overflow threshold: DISABLED (overflow delay used)
        Time Stamp(32KHz ticks) ->   Throughput(MB/s)
time:    End     Start    Delta -> Counter0 Counter1 Counter2 Counter3 Counter4 Counter5
time: 87347347  87314554  32793 ->   0.27     0.26     0.00     0.00     55.91   194.90
time: 87380140  87347347  32793 ->   0.27     0.26     0.00     0.00     55.90   194.90
time: 87412932  87380140  32792 ->   0.27     0.26     0.00     0.00     55.91   194.90
time: 87445725  87412932  32793 ->   0.27     0.26     0.00     0.00     55.90   194.89
```

When only one EMIF is present, it is not required to have two counters per initiator. A simplified omapconf command can be used instead:

```
shell@jacinto6evm:/ # audio-tool tone sine 1000 -D3 -r44100 -b16 -c6 &
shell@jacinto6evm:/ # omapconf trace bw \
> --tr0 r+w --m0 sdma_rd \
> --tr1 r+w --m1 sdma_wr \
> --tr2 r+w --m2 alldmm
...
Counter: 0  Master:         sdma_rd Transaction: r+w Probe: emif1
Counter: 1  Master:         sdma_wr Transaction: r+w Probe: emif1
Counter: 2  Master:          alldmm Transaction: r+w Probe: emif1
delay in us: 1000000
overflow delay in us: 1000000 (iterations=1)
accumulation type: 2
iterations (0=infinite): 0
Overflow counter index: DISABLED (overflow delay used)
Overflow threshold: DISABLED (overflow delay used)
             Time Stamp(32KHz ticks) ->   Throughput(MB/s)
time:           End       Start   Delta ->   Counter0   Counter1   Counter2
time:       1355848     1323055   32793 ->       0.53       0.00     250.80
time:       1388641     1355848   32793 ->       0.58       0.00     250.86
time:       1421434     1388641   32793 ->       0.53       0.00     250.79
time:       1454226     1421434   32792 ->       0.53       0.00     250.81
```


Copyright © 2016, Texas Instruments Incorporated

When eDMA is used instead of sDMA, the targets to monitor are different. The corresponding omapconf command is:

```
shell@jacinto6evm:/ # omapconf trace bw \
--tr0 r+w --m0 edma_tc1_wr \
--tr1 r+w --m1 edma_tc1_wr \
--tr2 r+w --m2 edma_tc1_rd \
--tr3 r+w --m3 edma_tc1_rd \
--tr4 r+w --m4 alldmm \
--tr5 r+w --m5 alldmm
```

5 Systrace

Android's Systrace is a helpful tool to capture and display the execution times of application and system processes, as well as Linux and Android kernel data such as the CPU scheduler, disk activity, threads, and so forth. The tool generates an HTML report that shows an overall picture of the system processes of the Android device for a given period of time. More details about the systrace tool can be found at Android's website: http://developer.android.com/tools/help/systrace.html.

The systrace tool has support for audio events, but the existing audio events are more useful for debugging at user-space level (such as Audio Flinger). The events can be extended to include ALSA events that could help monitor the amount of audio frames available in the ALSA buffer. This information is useful when investigating causes for ALSA xruns in a system level perspective.

Typically, the ALSA buffer stays near full level to prevent underruns in audio playback. Similarly, the ALSA buffers stays near empty level to prevent overruns in audio record.

5.1 Requirements

5.1.1 Host Tools

The Systrace scripts are part of the Android SDK tools, which can be downloaded from the Android Developers Website (https://developer.android.com/sdk/index.html#Other).

Depending on the host OS, uncompress the zip/tarball file or run the installer. For example, in a Linux host:

```
# tar zxf android-sdk_r24.3.3-linux.tgz
# cd android-sdk-linux
```

The Android SDK platform-tools do not come by default in the tarball, so they must be downloaded using the Android SDK manager, which can be found in the SDK tools.

```
# ./tools/android
```


Copyright © 2016, Texas Instruments Incorporated

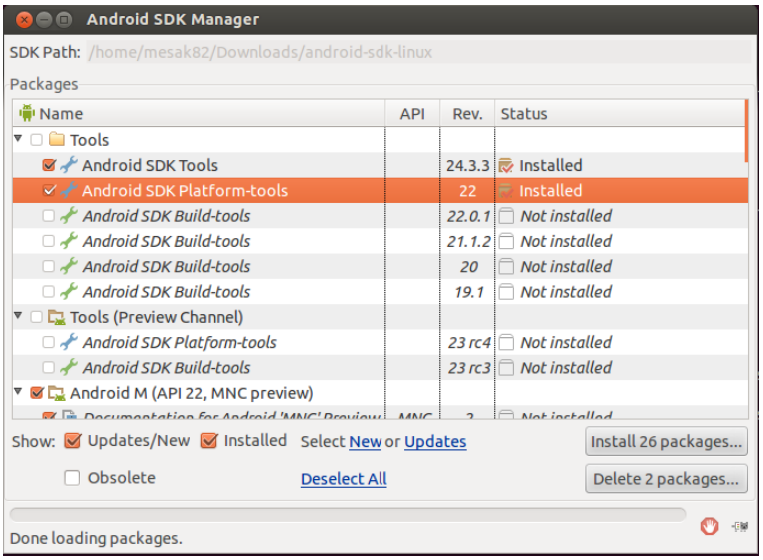

Figure 14. Install the Android SDK Platform-Tools

5.1.2 Kernel Requirements

Systrace requires kernel tracing support and default tracers to be enabled. Enabling the CONFIG_ENABLE_DEFAULT_TRACERS config option is required. This config option is listed in Android kernel recommended configs.

The amount of available frames in the ALSA buffer can be displayed using counter events in Systrace. These events require a specific trace event format.

The available frames can be probed at different places in the ALSA framework. These trace events do not exist in the kernel, so they must be explicitly added. The following patch should be applied on the kernel:

```
# cd $KERNEL_DIR
# git fetch http://review.omapzoom.org/kernel/omap refs/changes/96/36696/1 && git cherry-
pick FETCH_HEAD
```

Recompile the kernel and flash it to the target device.

5.1.3 Android Source Requirements

Systrace uses the atrace under the hood, which is an Android native utility. The new “alsa” tag must be added to the list of valid tags in atrace.

```
# cd frameworks/native
# git fetch http://review.omapzoom.org/platform/frameworks/native
refs/changes/94/36694/1 && git cherry-pick FETCH_HEAD
```


Copyright © 2016, Texas Instruments Incorporated

If using the asynchronous mode in atrace described in *Capturing Events Asynchronously*, an additional patch must be applied:

```
# git fetch http://review.omapzoom.org/platform/frameworks/native
refs/changes/94/36694/1 && git cherry-pick FETCH_HEAD
```

The atrace utility must be recompiled and pushed to the device.

```
# cd -
# make –j4
# adb push out/target/product/jacinto6evm/system/bin/atrace /system/bin/
```

5.2 Capturing Events Through ADB

The following is required to be run on the target device:

- Systrace can capture traces from kernel events but not from Android user-space unless `/sys/kernel/debug` permissions are changed on the target device:

```
# chmod –R 777 /sys/kernel/debug
# stop
# start
```

On the host machine, the following steps are required:

1. Restart the ADB daemon with root permissions.
2. Run systrace script with the relevant events for the use-case. Specify the capture duration accordingly:

```
# platform-tools/systrace/systrace.py -o/tmp/alsa_events.html -t5 input am
hal sched freq idle disk load worq audio alsa
```

3. The events are captured on the target and downloaded to the host through ADB.
4. The HTML trace report is created in the location specified in the argument `-o`, or in the current directory if the output file is not specified.

5.3 Capturing Events Asynchronously

There are cases where ADB is not available in the system, thus the traces must be captured on the target, stored on a removable media device, and copied to the host machine to generate the systrace output.

The following steps are required to be run on the target device:

1. Change the `/sys/kernel/debug` permissions.

```
# chmod –R 777 /sys/kernel/debug
# stop
# start
```


Copyright © 2016, Texas Instruments Incorporated

2. Capture the traces using the atrace utility. This is the same utility used by the systrace.py script over ADB.

```
# atrace -b 5000 --async_start input am hal sched freq idle disk load workq audio alsa
```

3. Run the use-case (such as playing music or recording voice).
4. Stop the capture and save the traces. The traces are saved in a compressed format.

```
# atrace -z --async_stop > /data/trace.dat
```

5. Copy the trace file to a removable and transfer it to the host machine.

On the host machine:

- Run the systrace script specifying the trace file captured on the target device.

```
# platform-tools/systrace/systrace.py --from-file=/media/sdcard/trace.dat -
o /tmp/alsa_events.html
```

NOTE: At the time of this writing, the Systrace script is not able to parse all events in TI Android Lollipop release, which is based on kernel-3.14. It works on the previous TI Android releases that are based on kernel-3.8.

The following examples show the systrace report for playback and capture cases where ALSA underrun and overrun events occurred. The ALSA buffer level can be monitored as it gets depleted (playback) or full (capture), until it eventually hits the xrun condition. This event can be correlated with other tasks in the system running simultaneously that might have contributed to hit this condition.

Copyright © 2016, Texas Instruments Incorporated

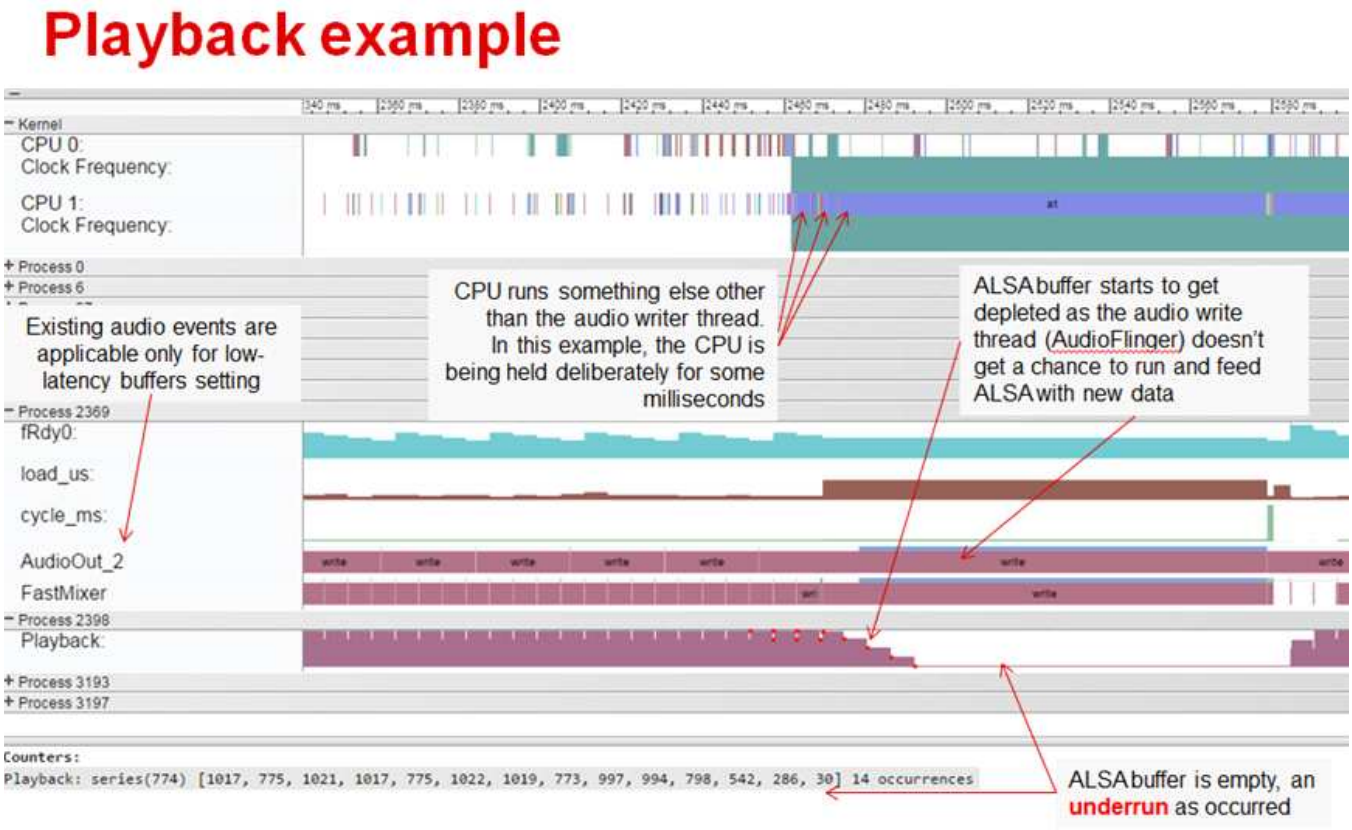

Figure 15. Systrace Report With an ALSA Overrun During Music Playback

Copyright © 2016, Texas Instruments Incorporated

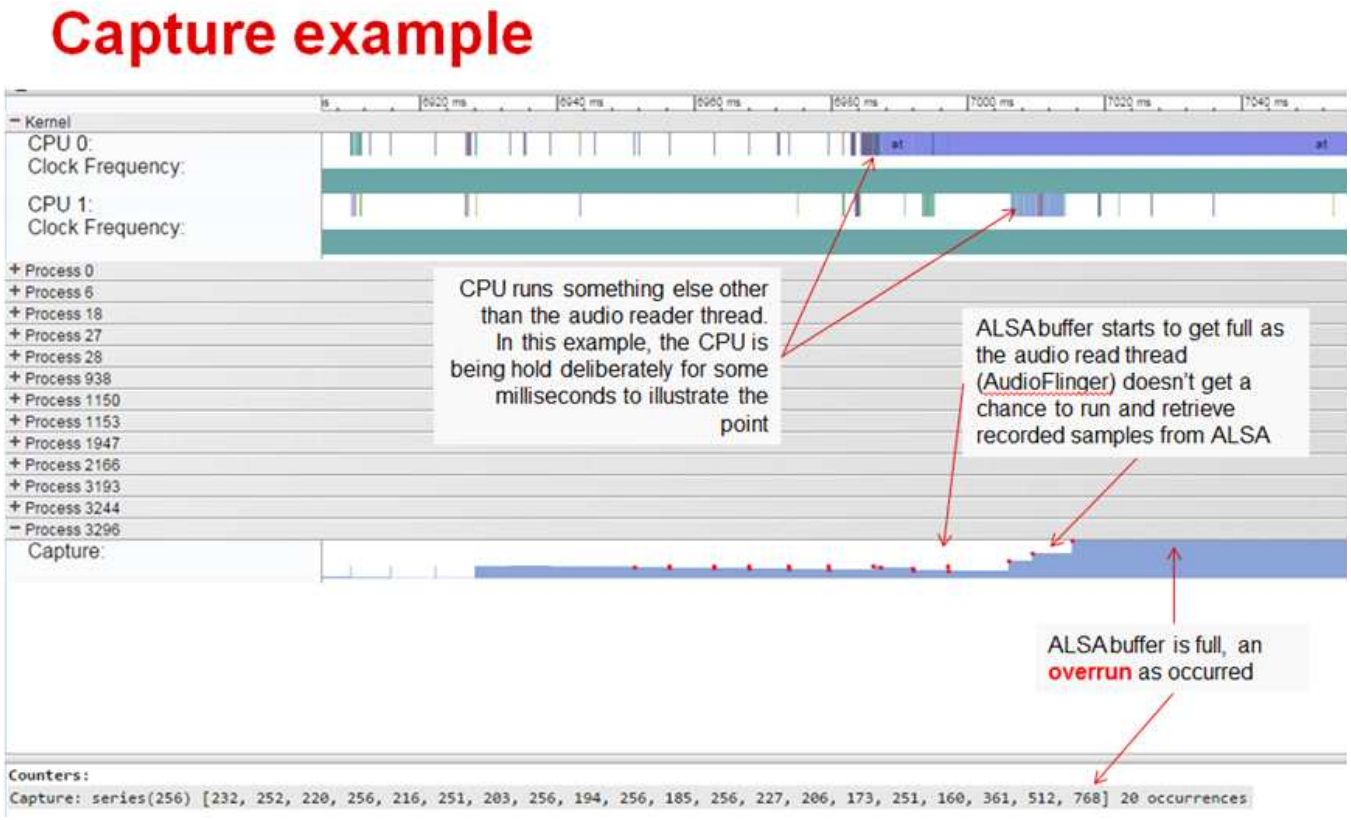

Figure 16. Systrace Report With an ALSA Underrun During Sound Recording

6 Most Common Audio Issues

This section describes the suggested steps required to debug the most commonly encountered audio issues.

6.1 McASP Underflows During Audio Playback

- Check if the McASP WFIFO is enabled. Use the omapconf tool to read the status of the McASP WFIFO (see *Data Ports and Buffers*).

  TI recommends using the McASP AFIFOs to prevent hardware underflows and overflows.
- Ensure the FIFO threshold is optimal. Use the omapconf tool to read the threshold level of the McASP WFIFO (see *Data Ports and Buffers*).

  The FIFO threshold is configured through a devicetree property in the Linux kernel, but can be overwritten by the McASP driver to satisfy hardware constraints.
- Measure the L3 bandwidth to identify the interconnect traffic conditions that may be affecting meeting McASP timing constraints. For the omapconf instructions to measure the L3 bandwidth, see *Measuring L3 Throughput*.

Copyright © 2016, Texas Instruments Incorporated

6.2 ALSA Underruns When Playing a Wave Sample With aplay and tinyplay

- Look for McASP underflows errors in the kernel logs. If they are present, see *McASP Underflows During Audio Playback*.
- Discard media storage I/O latency by trying to render from a RAM disk (see *Store Samples in a RAM Disk*), rendering from special device files (see *Using Special Device Files*) or rendering pre-generated patterns (see *Rendering Patterns*)

  If the ALSA underflow errors disappear, then the problem is the bottleneck in the sequential order of reading data from media storage and then writing it to the ALSA PCM device, as aplay and tinyplay are test applications.
- Use the ALSA XRUN debug infrastructure for further debugging. More information can be found at http://www.alsa-project.org/main/index.php/XRUN_Debug.

6.3 Input/Output Error During Audio Playback

- The input and output error typically happens when the ALSA PCM device is not consuming any data. ALSA times out after 10 secs of inactivity.
- Read the ALSA application (app_ptr) and hardware pointers (hw_ptr) and check if the values are changing. This information is found in the status file of the corresponding PCM device's procfs entry. For more information about ALSA procfs, see *ALSA procfs*.

  When the hw_ptr is not moving, the audio hardware did not consume the audio data.
- If McASP is a slave, make sure the BCLK and FSYNC are provided by the companion analog codec. Use the omapconf tool to determine the McASP BCLK/FSYNC settings (see *Clocks (BCLK)* and *Frame Sync Generator (FSYNC)*).
- Check if the pin mux settings (such as direction, mux mode, and so forth) are set according to the McASP master/slave configuration. for more information about the McASP pin mux settings, see the *DRA75x/74x SoC for Automotive Infotainment Technical Reference Manual* (SPRUHI2) or the *DRA75x, DRA74x Infotainment Applications Processor Data Manual* (SPRS857).

6.4 Audio Playback is Muted

- Run the audio playback application for at least 10 seconds to discard input and output errors.

  If an input or output error occurred, follow the steps in *Input and Output Error During Audio Playback*.
- Check if the pin mux settings for the transmit serializers are set accordingly to the McASP serializer allocation. For more information about the McASP pin mux settings, see the *DRA75x/74x SoC for Automotive Infotainment Technical Reference Manual* (SPRUHI2) or the *DRA75x, DRA74x Infotainment Applications Processor Data Manual* (SPRS857).
- Check if the audio routing and volume settings are correct on the codec side. The codec mixer controls can be checked using amixer or tinymix tools.

6.5 McASP Overflows During Audio Record

When looking for McASP RFIFO status, use the steps listed in *McASP Underflows During Audio Playback*.

6.6 ALSA Overruns When Capturing Audio With arecord and tinycap

When applicable for audio playback or audio record, see the steps listed in *ALSA Underruns When Playing a Wave Sample With aplay and tinyplay*.

6.7 Input and Output Error During Audio Record

For more information, see the steps in *Input and Output Error During Audio Playback*.

7 References

- *DRA75x/74x SoC for Automotive Infotainment Technical Reference Manual* (SPRUHI2)
- *DRA75x, DRA74x Infotainment Applications Processor Data Manual* (SPRS857)

Copyright © 2016, Texas Instruments Incorporated

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2016, Texas Instruments Incorporated