Real-time Audio on Embedded Linux

April, 2011

Remi Lorriaux
Adeneo Embedded

Embedded Linux Conference 2011

Agenda

- Introduction to audio latency
- Choosing a board for our project
- Audio using a vanilla kernel
- Audio using a real-time kernel
- JACK on our embedded device
- Conclusion

Who am I?

- Software engineer at Adeneo Embedded (Bellevue, WA)
  - Linux, Android
  - Main activities:
    - Firmware development (BSP adaptation, driver development, system integration)
    - Training delivery
- Guitar player (at home)

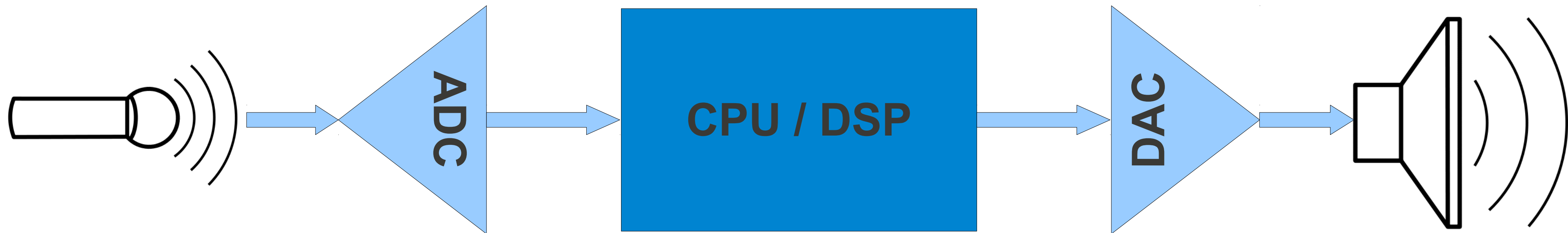
Context and objectives

- Case study
  - Using a generic embedded device, see how we can create a Linux system
  - Does not necessarily apply to all types of devices
  - Not an in-depth study of real-time systems

- Focus on software integration
  - Not an exhaustive review of all software solutions
  - We want to use open source components

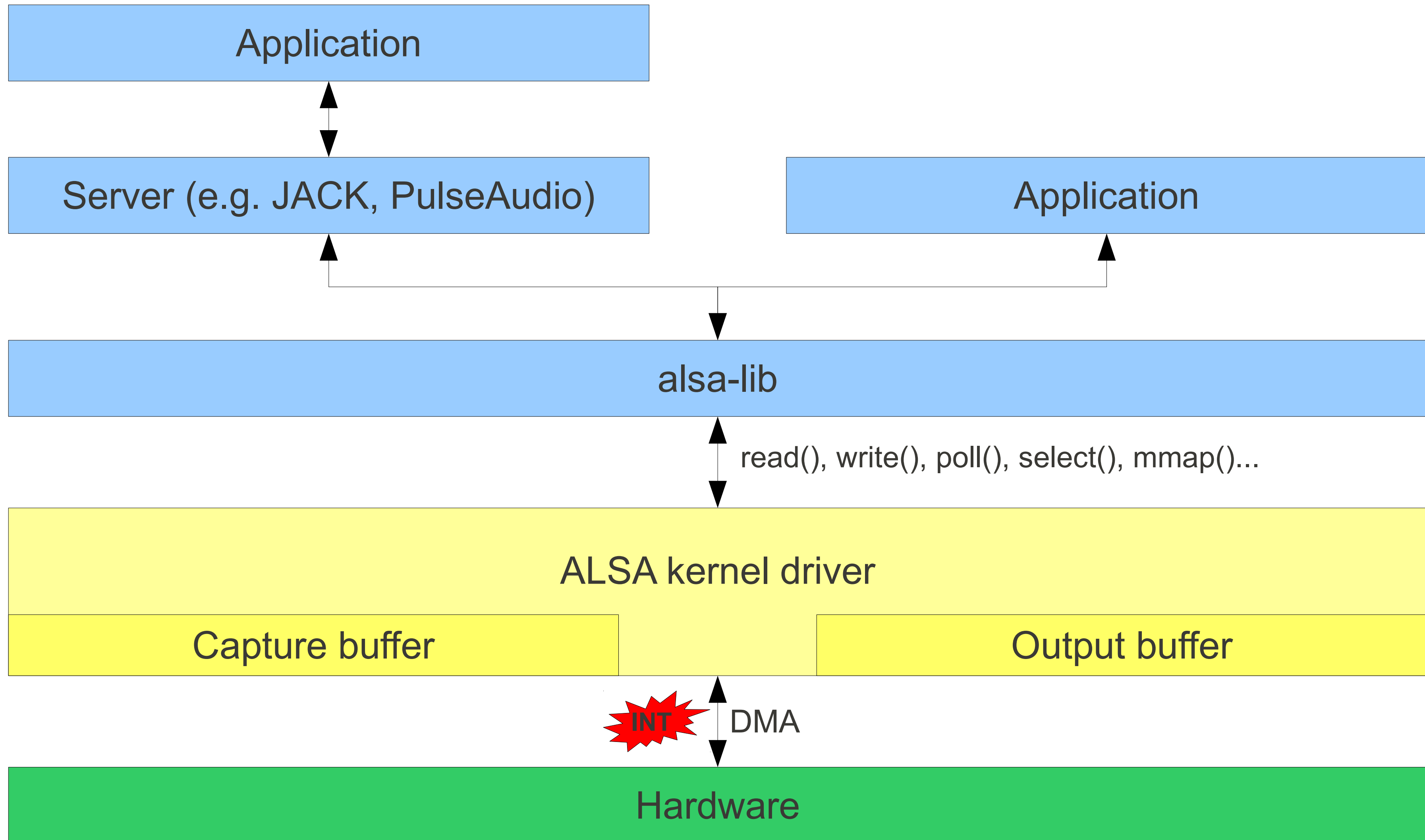
Adeneo Embedded

A typical audio chain



- Digital processing made with a DSP or a general-purpose CPU
- CPUs used in embedded devices generally have sufficient power to do the audio processing in software

Typical audio chain on Linux



Audio Latency

- Delay between an action and its effect, e.g.
  - Pressing a key → Hearing a note
  - Record input → Processed output
- Causes
  - Physical: 3ft away from loudspeakers ~ 3ms latency
  - Hardware: conversion can cause delays (magnitude: 40/Fs)
  - Software: various levels of buffering

Adeneo Embedded

Audio Latency

- Consequences on applications:
  - Music: critical issue
  - Communications: larger latencies can be tolerated but still have to be limited (Android specifies continuous output latency < 45 ms, continuous input latency < 50 ms)
  - Audio/Video playback: larger latencies can be tolerated as long as synchronization is maintained

Adeneo Embedded

Measuring audio latency

- Measuring total latency (including hardware):
  - Use an oscilloscope or a soundcard
  - Can measure the difference between 2 measurements to assess software changes
- Measuring software latency
  - ALSA: use *latency* test in alsa-lib (can automatically figure out the optimal latency through trial and error)
  - JACK: *jdelay, qjacklam*

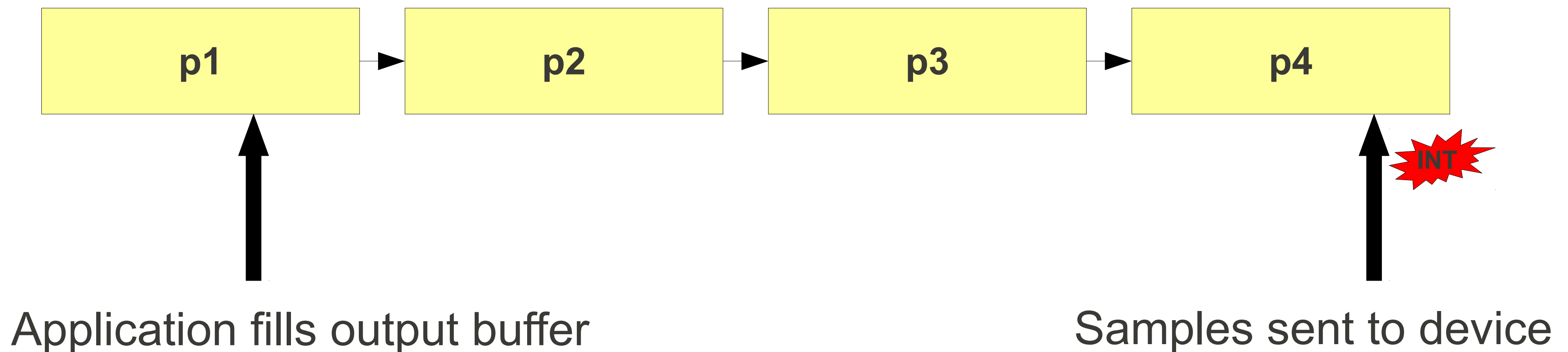
Adeneo Embedded

Buffering

- ALSA uses periods: blocks of audio samples
- Size and number is configurable on runtime by applications
- At the end of each period, an interrupt is generated.

  → Shorter periods: more interrupts

  → Shorter periods + less periods: lower latency



Adeneo Embedded

Reducing audio latency

- Buffering creates latency

  → Solution: *Reduce buffering*

- Reducing buffer size

  → Interrupts are generated more frequently

  → The system has less time to process the buffers: risk of over/underrun

  → The system must be designed to schedule the different tasks on time

Tuning the latency

- alsa-lib provides “*latency*”
  - Test tool used for measuring latency between capture and playback
  - Audio latency is measured from driver (difference when playback and capture was started)
  - Can choose a real-time priority (we changed the code to make it configurable at runtime)

Adeneo Embedded

Tuning the latency

- Trial and error:
  - Start with a low latency value
  - Load the system (CPU, IO stress, your application...)
  - If XRUN or bad sound quality → Increase the latency (periods number and size)
- "*latency*" can do it for you automatically:

```
> latency -m [minimum latency in frames] -M [maximum latency in frames] -r [sample rate] -S [priority] -p
```


Adeneo Embedded

Loading the system

- Ideally:
  - Realistic load (i.e. your end applications)
  - Worst-case scenario
- Our experiment:
  - Stress tests from the Linux Test Projet (LTP)
    - CPU stress
    - Memory allocation
  - Generate IO interrupts copying from block devices
  - *cyclictest*

Adeneo Embedded

Choosing a board for our project

- Deciding factors:
  - Pick a board that is supported in the latest mainline kernel with real-time patches (2.6.33)
  - Tested real-time features (Open Source Automation Development Lab)
  - Pick a board that was available at the time
  - Power and features did not matter (except for audio)

- Other boards could have been chosen (e.g. BeagleBoard)

Our project: Hardware

- Zoom™ AM3517 EVM Development Kit (*AM3517 EVM*)
- ARM™ Cortex™-A8 600 MHz
  → *Powerful, but real-time issues still have to be taken into account*
- Ethernet, USB, SD, NAND, Wireless, CAN, LCD, Video capture...
  → *Many ways to generate software interrupts*
- Audio in/out (obviously!)

Adeneo Embedded

Our project: Software

- Bootloader
  - U-Boot (Built by Arago)
- Kernel
  - 2.6.33
  - Patched to 2.6.33.7
  - Patched to 2.6.33.7.2-rt30 (RT version)
- Root filesystem
  - Built by Arago
  - Customized recipe (*arago-console-image.bb* + JACK)
  - Test applications: alsa-lib tests, LTP, rt-tests

Our project: Software challenges

- Support for the AM3517 EVM in mainline 2.6.33 is somewhat minimal (this has changed a lot)
- Ported some of the board code/drivers from TI's PSP (2.6.32): audio, SD/MMC
- Still, not many drivers are supported on our test system (not exactly a real-life example)

Experiment #1: Vanilla kernel

- Vanilla kernel: no RT patches
- Kernel configuration (*see files/kernel/am3517_std_defconfig*)

```
CONFIG_PREEMPT_DESKTOP=y
CONFIG_PREEMPT=y
```

(other preemption schemes not investigated)

Adeneo Embedded

Experiment #1: First observation

- The priority of latency has to be set to real-time otherwise, it over/underruns for low latency values (under 1024 samples @ 44.1 kHz)
- Even using a non-RT kernel, once the application priority is increased, it runs fine when competing with other “stress” processes (can be different in the real world)
- *cyclictest* also exhibits this behavior

  → **Use *sched_setscheduler()* or *chrt* and set your audio thread priority to SCHED_RR or SCHED_FIFO**

Adeneo Embedded

Experiment #1: CPU usage vs latency

- Measured with no load:
    - > ./latency -m 64 -M 64 -r 44100 -p -S 60

→ **Low-latencies can generate a large CPU load!**

Latency (samples)	Latency (@ 44.1kHz)	CPU Load (%)
64	1.5 ms	65%
128	2.90 ms	3%
1024	23 ms	< 1%
8192	185 ms	< 1%

Experiment #1: Loading the system

- Creating CPU load:
  - LTP CPU stress test

    → Competition between processes (preemptible)
- Creating IRQ pressure:
  - Transfers from SD Card

- **Result: Works fine with 1.5ms latencies**

  → Vanilla + SCHED_FIFO did the trick

Why use a real-time kernel?

- Tuning the application priority and audio buffering requirements can be sufficient for “soft real-time” audio systems – where occasional errors do not cause unacceptable audio quality consequences

  BUT:

- There is still a lot of non-preemptible code in the vanilla kernel
  - Interrupt handlers, tasklets...
  - Regions protected by spinlocks (depending on the number of processors on your system)

CONFIG PREEMPT RT Patch

- Using the CONFIG PREEMPT RT Patch gives you:
  - Almost full-preemption
  - Threaded interrupts: ability to prioritize interrupts like processes → your application can have a higher priority than hardware interrupts

    → Lower risk of over/underruns

Making the system real-time

- Apply the CONFIG PREEMPT RT Patch on vanilla kernels.
  - Latest available version: patch-2.6.33.7.2-rt30
  - Also set the configuration
- Difficult task on non-mainline/recent kernels
  - Making non-mainline boards compatible can be difficult (check locks and interrupt management in drivers and platform code, make sure that the timers are appropriate)
  - Cannot apply the patch on proprietary drivers

Checking the real-time behavior

- Check that the patch has been correctly applied:
  - Running "*ps*" on the device will show that IRQs are threaded
  - Use *cyclictest* to check the scheduling latency under load

    → The non-RT kernel fails right away under high IRQ pressure

    → Latencies remain bounded with RT
  - Use ftrace to see advanced information (see kernel documentation for usage) – changes the timing behavior of the system!

Tuning the real-time system

- Set the priority of IRQs and processes (decreasing priority):
  - (High-resolution) Timers (especially since we are using a tickless system!)
  - Audio (e.g. DMA on our platform)
  - Your audio process
  - Other interrupts
  - Other processes

  - More information (FFADO project)

Tuning audio parameters (RT edition)

- Same process as the non-RT experiment: trial and error
- Use the results provided by *cyclictest* to adjust the latency, e.g.
- Does not solve the problem of using shared hardware resources (e.g. DMA, busses...)

  → Requires careful platform and driver design

```
root@am3517-evm:~# cyclictest -t5 -n -p 60
policy: fifo: loadavg: 10.89 9.99 6.83 12/76 1342
T: 0 ( 1338) P:60 I:1000 C:  42699 Min:     21 Act:   36 Avg:   37 Max:      96
T: 1 ( 1339) P:59 I:1500 C:  28463 Min:     25 Act:   41 Avg:   37 Max:     135
T: 2 ( 1340) P:58 I:2000 C:  21344 Min:     25 Act:   37 Avg:   39 Max:     111
T: 3 ( 1341) 41 Avg:   39 Max:     111     22 Act:   54 Avg:   38 Max:      91
T: 3 ( 1341) P:57 I:2500 C:  17140 Min:     22 Act:   41 Avg:   38 Max:      91
T: 4 ( 1342) P:56 I:3000 C:  14283 Min:     25 Act:   27 Avg:   38 Max:      86
```


Adeneo Embedded

Experiment #2: RT Audio

- Dirty trick: Added *udelay(1000)* in the USB interrupt handler!

  → Simulate IRQ pressure since the kernel for our board did not support ethernet, display...

  → Not a perfect example

- **Result: Works flawlessly with 1ms latency**

Improving the experiment

- Run typical applications (UI, gstreamer, LADSPA...)
- More stress on the interrupts: Network, Display, Video Capture...

High-end audio applications: JACK

- System for handling real-time, low latency audio (and MIDI)
- Cross-platform: GNU/Linux, Solaris, FreeBSD, OS X and Windows
- Server/client model
- Connectivity:
  - Different applications can access an audio device
  - Audio applications can share data between each other
  - Support for distributing audio processing across a network, both fast & reliable LANs as well as slower, less reliable WANs.
- Designed for professional audio work

Adeneo Embedded

JACK: latency and real-time

- JACK does not add latency
- An RT kernel is needed only if:
  - You want to run JACK with very low latency settings that require real-time performance that can only be achieved with an RT kernel
  - Your hardware configuration triggers poor latency behaviour which might be improved with an RT kernel
- Most users do not need an RT kernel in order to use JACK, and most will be happy using settings that are effective without an RT kernel

JACK on our platform

- Built using OpenEmbedded (added the recipe to our image)
- Used straight out of the box

- Set priorities:
  - Make *jackd* have SCHED_FIFO priority (like your audio application seen before)
  - More information: FFADO wiki

Experiment #3: JACK on our platform

```
root@am3517-evm:~#jackd -R -P 60 -d alsa -i 2 -r 44100 -o 2 - -p 64 -n 6 &
root@am3517-evm:~# jackd 0.118.0
Copyright 2001-2009 Paul Davis, Stephane Letz, Jack O'Quinn, Torben Hohn and ot.
jackd comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions; see the file COPYING for details

JACK compiled with System V SHM support.
loading driver ..
apparent rate = 44100
creating alsa driver ... hw:0|hw:0|64|3|44100|2|2|nomon|swmeter|-|32bit
control device hw:0
configuring for 44100Hz, period = 64 frames (1.5 ms), buffer = 6 periods
ALSA: final selected sample format for capture: 16bit little-endian
ALSA: use 3 periods for capture
ALSA: final selected sample format for playback: 16bit little-endian
ALSA: use 3 periods for playback

root@am3517-evm:~# jack_simple_client
engine sample rate: 44100
```

Experiment #3: JACK on our platform

- **JACK works fine on our platform with < 10 ms latency**

(Note: simplest possible test, so this is a best-case value)

Adeneo Embedded

Using multi-core systems

- Embedded systems can have multi-core architectures:
  - Several CPUs
  - Mixed CPU/DSP
- The audio processing can be assigned to a particular core
- Example: TI Audio SoC example (Mixed DSP and ARM core)

- Shared hardware resources is still important (bus contention, DMA access...)

Conclusion

- Trade-offs:
  - CPU/power consumption
  - Latency
  - Design complexity
- Tune your priorities/audio parameters first and load your system
  - Procedure similar to desktop environments (well documented)

Adeneo Embedded

Conclusion

- Adding real-time support:
  - Not necessarily trivial or required
  - Depends on the implementation
  - Non-RT kernel can be surprisingly adequate for “soft real-time” audio

Questions?

Appendix: Files

- The files used for this experiment should be attached with the presentation
- Just run or have a look at the different scripts
- Run (in order):
  - *oe_prepare.sh*: installs prerequisites for Arago/OpenEmbedded (some other changes might be needed)
  - *oe_download.sh*: downloads and installs custom recipes for OE
  - *oe_build.sh*: builds OE

Adeneo Embedded

Appendix: Files

- Run (continued):
  - *uboot_build.sh*: builds U-Boot (sources pulled from OE)
  - *kernel_download.sh*: downloads the kernel sources and applies relevant patches for our experiments
  - *kernel_build.sh*: builds the RT and non-RT kernel
  - *apps_build.sh*: builds extra test applications out of the OE tree
  - *sd_flash.sh:* flashes the bootloader + kernel + rootfs on an SD Card

Appendix: References

- Real-Time Linux wiki: Lots of information about the RT patch and testing procedures
- The JACK Audio Connection Kit: General presentation. Also covers audio topics on Linux
- FFADO wiki: How to tune audio parameters
- ALSA wiki: General documentation and ALSA samples
- JACK: Developer documentation, tuning...
- AM3517 EVM: Board specification and tools

Adeneo
Embedded