Developing Audio Products with Cortex-M3/NuttX/C++11

Masayuki.Ishikawa@sony.com
Senior Software Engineer
Sony Video & Sound Products Inc.
Agenda

- Product outline
- Typical software development
- Porting NuttX to MCU
- Power management and fast ELF loading
- C++11 and standard library
- Debugging with apps
- adb support and testing with adb
- Demo videos
Product Outline

ICD-UX560
- microSDHC and microSDXC support
- Focus and wide mic mode
- Digital Pitch Control

ICD-SX2000
- LPCM recording (up to 96k/24bit)
- FLAC/LPCM playback (up to 192k/24bit)
- Wireless control with REC Remote

NW-WS410
- Water proof (salt water)
- Ambient sound mode
- Up to 12h of battery life
## Hardware Comparison

<table>
<thead>
<tr>
<th>Model name</th>
<th>ICD-UX560</th>
<th>ICD-SX2000</th>
<th>NW-WS410</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public release</td>
<td>2015/10</td>
<td>2016/01</td>
<td>2016/02</td>
</tr>
<tr>
<td>RTM*</td>
<td>2015/09</td>
<td>2015/12</td>
<td>2015/11</td>
</tr>
<tr>
<td>CPU package</td>
<td>TQFP</td>
<td>WLP</td>
<td>WLP</td>
</tr>
<tr>
<td>eMMC</td>
<td>4GB, 8GB</td>
<td>16GB</td>
<td>4GB, 8GB</td>
</tr>
<tr>
<td>SD card</td>
<td>microSDHC</td>
<td>microSDHC</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>microSDXC</td>
<td>microSDXC</td>
<td>-</td>
</tr>
<tr>
<td>Audio CODEC</td>
<td>DA7213</td>
<td>DA7211x2 + CXD3774GF</td>
<td>CS47L01</td>
</tr>
<tr>
<td>Display</td>
<td>OLED 128x128</td>
<td>STN LCD 128x128</td>
<td>-</td>
</tr>
<tr>
<td>Serial Flash</td>
<td>-</td>
<td>Winbond 2MB</td>
<td>-</td>
</tr>
<tr>
<td>NFC</td>
<td>-</td>
<td>CXD2249GG (HCI)</td>
<td>-</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>-</td>
<td>CSR8811 (HCI)</td>
<td>-</td>
</tr>
<tr>
<td>FM Tuner</td>
<td>Si4708</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*RTM = Release To Manufacturing
## Typical Software Development

<table>
<thead>
<tr>
<th>Product models</th>
<th>Android based</th>
<th>Linux based</th>
<th>RTOS based</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>ARM Cortex-A series w/ MMU</td>
<td>ARM Cortex-A series w/ MMU</td>
<td>ARM Cortex-M series w/o MMU</td>
</tr>
<tr>
<td>Clock</td>
<td>1GHz -</td>
<td>500MHz -</td>
<td>100MHz –</td>
</tr>
<tr>
<td>Memory</td>
<td>mDDR2 512MB -</td>
<td>mDDR 64MB -</td>
<td>SRAM 128KB –</td>
</tr>
<tr>
<td>SPI Flash</td>
<td>Not used</td>
<td>Not used</td>
<td>Normally used</td>
</tr>
<tr>
<td>Toolchain</td>
<td>arm gcc (Google provides)</td>
<td>arm gcc (SoC vendor provides)</td>
<td>Provided by MCU vendor</td>
</tr>
<tr>
<td>BSP (Board Support Package)</td>
<td>Provided by SoC vendor</td>
<td>Provided by SoC vendor</td>
<td>Provided by MCU vendor</td>
</tr>
<tr>
<td>Programming Language</td>
<td>Java + native (C/C++)</td>
<td>C/C++</td>
<td>C (C++)</td>
</tr>
<tr>
<td>Debug commands</td>
<td>Can load dynamically</td>
<td>Can load dynamically</td>
<td>Need to link statically</td>
</tr>
<tr>
<td>Debug tools</td>
<td>adb, gdb + gdb server</td>
<td>gdb + gdb server</td>
<td>Commercial ICE</td>
</tr>
</tbody>
</table>
Why we chose NuttX

- POSIX and libc are supported
  - Can reuse existing software
  - Can reduce training costs
- ELF* is supported
  - Can divide into small apps
- Driver framework is supported
  - Helps us implement drivers
- Has Linux-like configuration system
  - Helps us develop multiple products
- Many MCUs and boards are supported
  - Helps us port NuttX to new MCU
- BSD license is available

* ELF = Executable and Linking Format

From http://www.nuttx.org/
Technical Challenges

- Porting NuttX to MCU
- How to use open tools such as openocd
- Need to consider small RAM size
- How to reuse existing software
- How to apply modern software development
Software Stack and tools

GUI apps

Non GUI apps

UI toolkit

Application Manager

Services

BT stack

NFC stack

NuttX 7.5 + peripheral drivers

MCU (LC823450) *

QEMU 1.4.0

*MCU is not a part of software stack.
tools: gcc-arm-none-eabi-4_8-2014q1, openocd-0.9.0-dev
LC823450 Features

- ARM Cortex-M3 dual core
- 32bit fixed point, dual-MAC original DSP
- Internal SRAM (1656KB) for ARM and DSP
- I2S I/F with 16/24/32bit, MAX 192kHz (2chx2)
- Hard wired audio functions
  - MP3 encoder and decoder, EQ (6-band equalizer), etc.
- Integrated analog functions
  - Low-power Class D HP amplifier, system PLL
  - Dedicated audio PLL, ADC
- Various interfaces
  - USB2.0 HS device / host (not OTG), eMMC, SD card, SPI, I2C, etc.
- ARM and DSP clock max frequency
  - 160MHz at 1.2V
  - 100MHz at 1.0V

From http://www.onsemi.com/PowerSolutions/product.do?id=LC823450
Porting NuttX to MCU

- Started with LC823450 FPGA
  - FPGA code was provided by ON Semiconductor
  - Ported NuttX-7.4 first, then merged 7.5
  - Cortex-M3 (20MHz), NVIC, Timer, UART, GPIO
  - eMMC, SD, DMA, SPI, LCD
  - I2C, I2S, Audio Buffer, Audio CODEC
  - RTC, ADC, USB
  - SPI-Flash, Bluetooth, DSP

- After LC823450 ES arrived
  - Test MAX CPU clock with PLL
  - Test eMMC boot
  - Implement power management
  - Implement suspend & resume
OpenOCD

- The very first step
  - Need to prepare before porting NuttX
- Version 0.9.0-dev
  - SWD (Serial Wire Debug) supported
  - With FTDI FT232H board
- Prepare startup scripts
  - Cortex-M sysreset
  - Be careful with adaptor clock
- Load the program to SRAM
- Load to SPI-Flash
  - Implement SPI-Flash driver
eMMC/SD driver

- Implement as a block device
- Call ROM APIs
  - idevicecard, readsector, writesector, etc.
  - instead of using eMMC driver in NuttX
- Use fixed partitions
  - due to ROM code restrictions
- Use DMA
  - to reduce CPU load
- Work with hotplug driver
  - i.e. SD card detection
  - newly introduced

![Diagram showing the eMMC/SD driver implementation with blocks and connections to ROM API and block device API.](image)
File Systems

- Using NuttX file systems
  - procfs for debugging, wake_lock, etc.
  - vfat for program files, properties, database
- Add eVFAT
  - Provided by ON Semiconductor
  - FAT32, exFAT supported
  - IC recorder specific APIs supported
  - Cache control supported
- Others
  - Add read only option
  - Add remount option
eVFAT

- Implement using NuttX VFS* APIs
- Call ROM APIs
  - mount, open, read, write, lseek, …, etc
- Add new IOCTLs
  - divide, …
- Add UTF-8 from/to UTF-16 conversion
- Use a dedicated stack like IRQ
  - Because some APIs need more stacks

* VFS = Virtual File System
Audio Support

- Features
  - H/W MP3 encoder and decoder
  - H/W Audio Buffer (64KB)
  - Beep generator
  - Mute & volume control
  - In DSP
    - Decoders (WMA, AAC, FLAC, etc)
    - Audio signal processing

- Implementation
  - NuttX has an audio subsystem
  - Technically possible to use existing features
  - But we decided to develop new APIs like alsa-lib
  - Non-blocking API
Audio Playback Example (AAC, …)

- **Cortex-M3**
  - Set up audio routing and buffers
  - Set up external audio CODEC
  - Load DSP code and boot
  - Read a file on eMMC/SD
  - Parse audio frame
  - Write the frame to the Audio Buffer

- **DSP**
  - Decode the frame
  - Do post process
  - Write PCM data to the Audio Buffer
Audio Recording example (MP3)

- **Cortex-M3**
  - Set up audio routing and buffers
  - Set up external audio CODEC
  - Load DSP code and boot
  - Wait for the buffer to be filled
  - Write the audio frame to a file

- **DSP**
  - Wait for the audio buffer from I2S
  - Perform preprocessing of the frame
  - Write to the Audio Buffer

- **MP3 Encoder**
  - Wait for the audio buffer from DSP
  - Encode the PCM data
  - Write to the Audio Buffer
Power Management

- Clock gating
  - Disable clocks for unused blocks
- Power gating
  - Disable power for unused blocks
  - ISOLATED-A : Audio
  - ISOLATED-B/C/D : SRAM
  - ISOLATED-E : USB Host
  - ISOLATED-G : SPI-Flash cache
- DVFS
- Suspend & Resume

From LC823450-D.PDF
DVFS* (1/2)

- Voltage control
  - 1.2V at 160MHz, 1.0V at 100MHz
- Clock control
  - CPU/DSP clock, AHB clock
- Clock table example
  - Active mode: 160M/80M/40M/24M
  - Idle mode: 24M/12M/6M/3M
- Autonomous control
  - Calculates the idle ratio
  - Controls divider and selector
- Boost the clock
  - when the keys are pressed
  - while loading applications

* DVFS = Dynamic Voltage and Frequency Scaling
DVFS (2/2)

- NuttX has CPU load monitoring
  - To monitor each task load
  - But the load in IRQ handler is not considered
- Need more accurate idle time
  - With simple calculation
  - Accumulate sleep time in usec during WFI
  - Calculate the idle ratio
- Use an internal H/W timer for tick
  - Instead of SYSTICK in Cortex-M3
  - As the timer is not affected by clock change
  - Results in simple calculation

\[
\text{idle\_time} = \sum \text{sleep\_time}
\]

\[
\text{idle\_ratio}_n = \frac{\text{idle\_time}_n - \text{idle\_time}_{n-1}}{\text{interval}}
\]
Suspend & Resume

- Introduce `wake_lock`
  - Provides APIs similar to those of Android kernel
  - If the power state is set to “mem” and no `wake_lock` exists, enter to SLEEPDEEP mode

- Implementation
  - Use a kernel thread
  - Power down unused blocks
    - e.g. Audio, SD, etc…
  - USB Suspend must be considered
  - Set SLEEPDEEP flag in NVIC and issue WFI
  - Woken up by interrupt when the following are received
    - i.e. GPIO, RTC alarm, USB resume
  - Power on the blocks if needed
  - Synchronize the kernel time with RTC
ELF* support

- **Motivation**
  - To overcome limited memory
  - More flexible than overlay

- **Divide into small applications**
  - e.g: Home, Settings, Play, Rec, …etc.

- **Can use separate debug commands**
  - without linking them to the applications.
  - e.g: ps, free, …

* ELF = Executable and Linking Format

ps & free on ICD-SX2000
Fast ELF loading

1. Section data cache
   - Allocate a big heap to hold tables to reduce eMMC access.
   - Use unused SRAM areas, if possible
     - e.g. DSP program & work area

2. Symbol name replacement
   - Shorten symbols by hashing their names
   - Sort A-Z and do binary-search in find-by-name
   - Need to modify the build system
Developing with QEMU*

**Motivation**
- To port the Bluetooth stack
- To port in-house GUI toolkit
- To develop applications

**Implementation**
- Start with 1.4.0
- Use TI Stellaris for QEMU hardware
- Use lm3s6965-ek for NuttX
- Increase SRAM size to 4MB
- Fix SD driver
- Fix NVIC issue

*QEMU is open source CPU emulator*
C++11

- **Motivation**
  - Improve productivity
  - Performance benefits

- **Features**
  - `auto` keyword
    - the compiler determines the type
  - Lambda expression to define function objects
  - New smart pointer
    - to avoid memory leaks
    - introduced `std::unique_ptr<>` and `std::shared_ptr<>`
  - Move semantics to optimize copying
    - introduced move constructor and assignment
    - introduced `std::move()`
  - `override`, `final`, `nullptr`, `constexpr`

---

### C++ Standard library

<table>
<thead>
<tr>
<th></th>
<th>libc++</th>
<th>libstdcxx</th>
<th>libstdc++</th>
<th>STLPort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintained by</td>
<td>APACHE</td>
<td>GNU</td>
<td>STLPort</td>
<td></td>
</tr>
<tr>
<td>C++11 support</td>
<td>Fully supported</td>
<td>Not supported</td>
<td>Fully supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>License</td>
<td>MIT and UIUC (BSD-like)</td>
<td>Apache</td>
<td>GPLv3 (mainline) GPLv2 (ver 4.2)</td>
<td>Boris Fomitchev</td>
</tr>
<tr>
<td>Others</td>
<td>LLVM and Clang supported Newer codebase and easier to port</td>
<td>4.2.1 released in 2008/05</td>
<td>Tightly integrated with g++</td>
<td>5.2.1 released in 2008/10</td>
</tr>
</tbody>
</table>

From http://libcxx.llvm.org/
Code size reduction

- **Example**
  - kernel and static libraries

- **Approaches**
  1. Started with ‘-O2’
  2. Plus Compile with ‘-Os’
  3. Plus GC of unused sections at link
  4. Plus Symbol name replacement
Debugging with apps (1/2)

- **openocd supports** some OSes
  - Linux, FreeRTOS, ChibiOS, …
  - The feature is very useful to debug deadlocks
  - Unfortunately NuttX is not supported

- **Implementation**
  - Similar to other RTOSes (e.g. ChibiOS)
  - Prepare symbol list to look up
    - i.e. g_readytorun, g_tasklisttable, …
  - Implement update_threads callback
  - Fix memory corruption in rtos.c

*The code is now available on https://github.com/sony/openocd-nuttx*
Debugging with apps (2/2)

- Typical scenario
  - Crash occurs when testing
  - Crash logs are saved in RAM
  - Reboot by WDT
  - Save the logs to a file when booting
  - Pull the log with adb
  - Analyze the log with debug symbols
adb* support

- **Motivation**
  - To test the system without proprietary tools
  - To retrieve internal logs

- **Features**
  - push, pull and shell with a remote execution
  - The feature is disabled at the factory before shipping

- **Implementation**
  - Start with the NuttX USB serial driver
    - composite version
  - Change the USB descriptors
  - Implement the protocols from scratch

* adb = Android Debug Bridge
Integration & testing with adb

- **For development**
  1. Push codes and create a pull request
  2. Build the code
  3. Deploy the software to each product
  4. Test the products with adb
  5. Store the test results with Jenkins

- **At factory**
  - PCB* tests are done with adb
  - After all tests pass, adb is disabled

*PCB = Printed Circuit Board
Automated Unit-testing with googletest

- **Google Test**
  - Google’s C++ testing framework
  - Port to NuttX and libc++ environment

- **Motivation**
  - To find bugs early
  - To clarify interfaces between modules
  - To refactor code safely
  - To make sure code works correctly on new target boards

- **Executing Test**
  - Transfer and execute test cases with adb
  - Faster-cycle of developing and testing
DSP software development

- Procedure
  - Develop code on the simulator
  - Run the sample app on Cortex-M3 and wait for loading DSP code
  - Load the DSP code via DSP-ICE then start the DSP
  - Continue the app on Cortex-M3
Demo videos

- Video #1: adb, fast ELF loading, DVFS
- Video #2: stress testing tool like Android monkey
Thank you