Porting Android ICS to a custom board
A war story

Matthias Brugger
Outline

- Android beyond smartphones?
- Why a war story
- Our custom board
- Android building system
- Lessons learned
  - Add device
  - Bootloader/Kernel integration
  - Powersupply
  - Button
  - Touchscreen calibration
  - Wi-Fi
  - Sound
  - HW acceleration
Android beyond smartphones?

- Easy to implement applications (SDK, API, Java)
- Human-Machine-Interface
  - well defined
  - tested in the wild
- Reliability of the system
  - good for HMI centred embedded systems

- Integrate native application
- Porting to a new board can be a war story
Why is it a war story

- Little documentation
- Small community
- Vendor specific communities
- Developing process of Android
- A huge jungle of programming languages
Why is it a war story

- Android is Linux, but... Android is not Linux!
  - Patched Kernel adds new features
  - Userspace varies widely
    - Own libc (bionic)
    - Lots of basic building blocks are not integrated
      - XWindows
      - Busybox
  - Core system is executed on dalvikVM
  - IPC implementation (binder) varies from SystemV
  - Building blocks are compiled to dynamic libraries which are loaded by the core system in different ways and layers
Why is it a war story

- Java Applications
  - Java Framework and Services
    - Java Native Interface
      - Native Daemons and Libraries
        - Kernel
Our custom board
Our custom board

- One button only
- No phone
- No battery monitor
- Ethernet
- Wifi
- Touchscreen
- Sound
- HW acceleration from OMAP3
Android building system

- Important folders
  - build
  - frameworks/base
  - external
  - hardware
  - device
  - out/target/product/<product_name>
Lesson learned I: Add device to our build

- Devices are found under `device/<manufacturer>/<board>` folder
- `<device_name>.mk`
  - Define which basic packages to install for the board
  - Inherit from default product (in `build/target/product`)
  - Add the proper device:
- `device.mk`
  - Define files that have to be copied to the rootfs
  - Define where to find the overlay
- `BoardConfig.mk`
  - Define build flags
- `vendorsetup.sh`
  - Add lunch menu option
Lessons Learned II: Bootloader/Kernel integration

- Problems
  - In AOSP precompiled Kernel image
  - Specific compiler needed?

- Solution
  - We add a “wrapper” in the Makefile
  - After building the android “userspace” build
    - Bootloader
    - Kernel
    - Copy kernel modules to `out/target/product`

- Extra boot parameter:
  - `androidboot.console=ttyO2`
  - `init=/init`
Lessons Learned III: Powersupply

- Boot hangs on splash screen
- Have a look with logcat what's going on
Lessons Learned III: Powersupply

E/BatteryService(1008): Could not open /sys/class/power_supply
(…)
I/SystemServer(1008): Battery Service
W/dalvikvm(1008): No implementation found for native Lcom/android/server/BatteryService;.native_update ()V
W/dalvikvm(1008): threadid=11: thread exiting with uncaught exception (group=0x409e11f8)
I/Process (1008): Sending signal. PID: 1008 SIG: 9
E/AndroidRuntime(1008): *** FATAL EXCEPTION IN SYSTEM PROCESS: android.server.ServerThread
E/AndroidRuntime(1008): java.lang.UnsatisfiedLinkError: native_update
E/AndroidRuntime(1008): at com.android.server.BatteryService.native_update(Native Method)
E/AndroidRuntime(1008): at com.android.server.BatteryService.update(BatteryService.java:233)
E/AndroidRuntime(1008): at com.android.server.BatteryService.<init>(BatteryService.java:148)
E/AndroidRuntime(1008): at com.android.server.ServerThread.run(SystemServer.java:196)
I/Zygote (967): Exit zygote because system server (1008) has terminated
E/installId (913): eof
E/installId (913): failed to read size
I/installId (913): closing connection
I/installId (913): new connection
I/ServiceManager (903): service 'gfxinfo' died
I/ServiceManager (903): service 'activity' died
I/ServiceManager (903): service 'cpuinfo' died
I/ServiceManager (903): service 'sensorservice' died
I/ServiceManager (903): service 'meminfo' died
I/ServiceManager (903): service 'account' died
I/ServiceManager (903): service 'usagestats' died
I/ServiceManager (903): service 'permission' died
I/ServiceManager (903): service 'hardware' died
I/ServiceManager (903): service 'content' died
(…)
Lessons Learned III: Powersupply

SystemServer
  run()
  create

BatteryService
  BatteryService()
  update()

JNI_OnLoad(...)
  register
  read
  update_native()

/sys/class/power_supply

Battery Service JNI

Kernel
Lessons Learned III: Powersupply

- What has happened?
  - When JNI is loaded, it registers the low level services (e.g. BatteryService)
    - read values from sysfs
    - BatteryService JNI registers function native_update at BatteryService.java
  
- But...
  - BatteryService JNI interface doesn't find “/sys/class/power_supply”
  - Returns error, which is ignored by JNI_OnLoad

- SystemServer thread creates BatteryService when executed
  - BatteryService tries to update values invoking native_update
  - No JNI interface registered – SystemServer dies...
Lessons Learned III: Powersupply

- Solution
  - No working battery monitor in our system
    - No power supply class in kernel
  - Adding the power supply class
    - Shows battery empty warning in status bar
  - Add the “test power driver”
    - Shows Battery at 50% charging fixed

- Has influence on the behaviour of the Power Manager
Lessons Learned IV: Button Integration

- Can be found in frameworks/base/services/input:
  - Android scans `/dev/input` folder in a loop
  - Polls for events using epoll
  - Identifies touchscreens, joysticks, mice, keyboards automatically

- Key Layout File (`/system/usr/keylayout`) maps raw input key to internal Android key representation
- Key Char Map File (`/system/usr/keychars`) describes action for internal Android key
Lessons Learned IV: Button Integration

- Our case
  - Just one button, no external keyboard
- Two alternatives:
  - Define own Key Layout/Key Char Map files
  - Configure key code in kernel appropriately
  - Beware Key Layout File uses numeric number of key code
  - Not all values in Key Layout have a define in `include/linux/input.h`
Lessons Learned V: Touchscreen calibration

- Detection and input analogous to button integration
- Driver sends X-Y-coordinates

- Calibrate the touchscreen
  - We have to reach all parts of the screen easily
    - Turn on *show touches* in Settings/Developer options
    - Adjust max and min values of your touchscreen in the platform data
Lessons Learned VI: Wi-Fi

- Example of complexity of Android source code
  - Different libraries and services
  - 5 different folders with userspace code
- Uses wpa_supplicant to connect to a protected WLAN
  - Enhanced with Android specific commands
    - START, STOP, SCAN-ACTIVE, SCAN-PASSIVE, RSSI, LINKSPEED, ...
Lessons Learned VI: Wi-Fi

- External/wpa_supplicant_8
- Hardware/.../wlan
- Kernel/net
- WEXT
- Device Driver
- Wpa_supplicant
- Driver_wext
- Jni net_wifi_Wifi
- Libhardware_legacy
- Wpa_ctrl
- WifiStateMachine
- WifiNative
- Frameworks/base/wifi/java/android/
- Frameworks/base/core/jni
- Hardware/libhardware_legacy
Lessons Learned VI: Wi-Fi

- Integration
  - **Legacy implementation** reads wpa supplicant config file
    - `/data/misc/wifi/wpa_supplicant.conf` → product specific
    - `/system/etc/wifi/wpa_supplicant.conf` → template

- **Kernel and firmware module**
  - In BoardConfig.mk define:
    - `WIFI_DRIVER_MODULE_PATH`
    - `WIFI_DRIVER_MODULE_NAME`
    - `WIFI_FIRMWARE_LOADER`
  - Hardware legacy layer in charge of loading/unloading kernel module and firmware; starts/stops wpa_supplicant

- **Start wpa_supplicant and dhcp for wlan**
  - `service wpa_supplicant /system/bin/wpa_supplicant -Dwext -iwlan0`
  - `service dhcpcd_wlan0 /system/bin/dhcpcd -ABKL`
Lessons Learned VII: Audio

- Audio player is stagefright
- AudioFlinger interfaces hardware abstraction
  - Uses struct audio_hw_device in hardware/libhardware/include
  - audio.primary.<device_name>
  - Encapsulates the hardware - custom implementation

![Diagram showing the relationship between AudioSystem, AudioFlinger, libtinyalsa, custom impl., and audio.primary.<device>]

ELCE 2012 – Porting Android ICS
Lessons Learned VIII: Hardware Acceleration

• Two possibilities
  • Integrate mulitmedia framework supporting HW acceleration
    ✓ Has bindings for HW accelerator
    ✓ Might have more codecs then stagefright

✗ Maintenance might be costly

• Integrate HW acceleration support in existing Android multimedia framework
  ✓ Should be easy to maintain

✗ Need to create bindings for the HW accelerator
Lessons Learned VIII: Hardware Acceleration

libmediaplayerservice:
Lessons Learned VIII: Hardware Acceleration

- Add own multimedia framework
  - Register your framework as MediaPlayerInterface
  - Create player instance in MediaPlayerService
  - Implement new MetadataRetriever in MetadataRetrieverClient

- Implement wrapper
  - Audio sink using AudioTrack
  - Video sink using Surface
LibsSurfaceFlinger:
Lessons Learned VI: Hardware Acceleration

libstagefright:
Porting Android ICS to a custom board
A war story

mbrugger@iseebcn.com
Danke schön

ありがとうございます

Gracias

Merci Beaucoup

감사합니다

Thank you

謝謝

3Q

Moito Brigado

Dank U wel