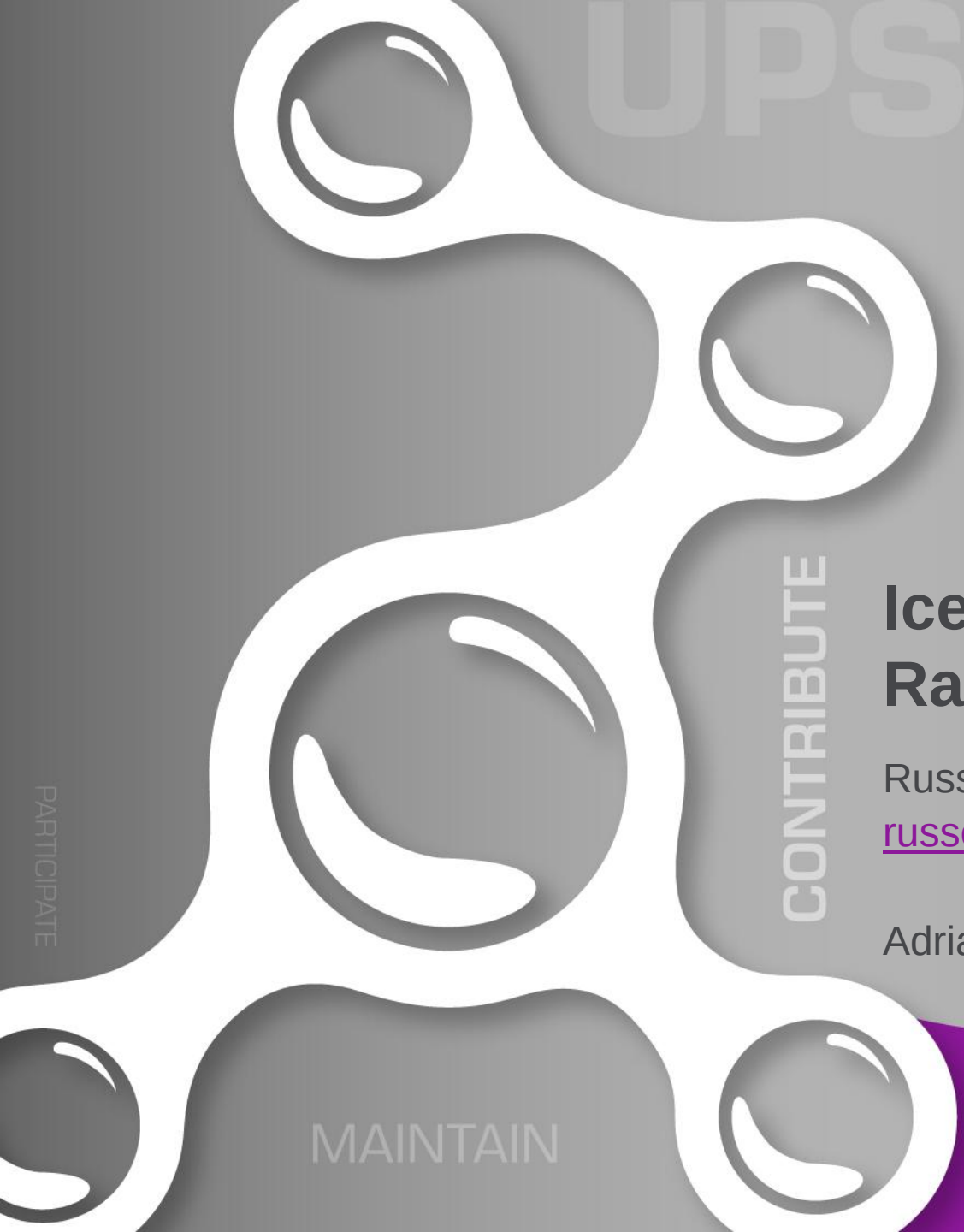


Ice Cream Sandwich Rapid Bring Up

Russell Webb – Software Engineer
russell.webb@intel.com

Adrian Negreanu – Software Engineer

INTEL OPEN SOURCE TECHNOLOGY CENTER

Introduction

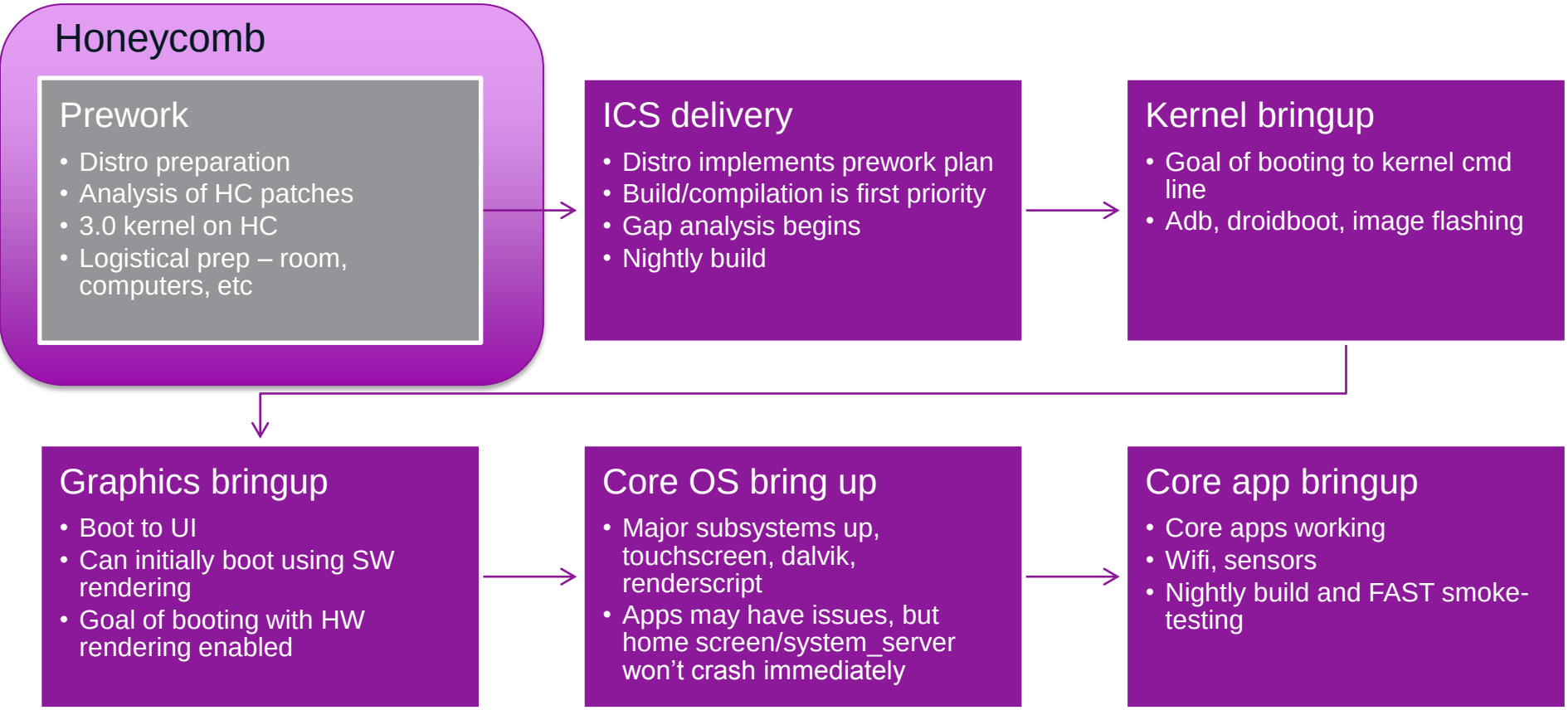
- Who am I?
  - Medfield Tablet Team Lead with Intel's Open Source Technology Center.
  - Started working on Android* with 0.9 release for T-Mobile Garminfone.
- A bit about my team and Intel's Android* Board Support Package (BSP)
  - We have a globally distributed engineering team developing the BSP.
  - We have a distribution team that handles Infrastructure and SCM, also geographically distributed.
  - We are responsible for delivering the Android reference OS for the Intel Medfield Tablet reference design.
  - We use a provisioning OS called droidboot that implements the fastboot protocol for the target.
- Our goal for ICS
  - Challenged to enable ICS on Medfield tablet in days, not weeks.
  - Get majority of developers off of HC and working on new OS as quickly as possible.
  - Get to market quickly on ICS.

Overview

- Phases of the bring up: how we set milestones
  - Prework, Code landing, first build, first command line boot, first UI boot, peripheral systems bring up
- Key places to look when first bringing up Android
- Cleaning up
- Lessons learned
- Questions

Phases of work

- **Six distinct phases:**
  - Each phase has clear entry and exit criteria



Prework Phase

- Because ICS was on a 3.0 kernel, we knew it would be key to have a working 3.0 kernel prior to ICS source availability.
- Honeycomb OS was used as the preparation vehicle. First goal before ICS release was to have honeycomb running well on a 3.0 kernel.
- All of our patches for AOSP projects were analyzed and put into groups: don’t port, maybe port, port.
  - Emphasis was to minimize the number of patches we carry forward. We did NOT want to simply pull everything forward without knowing for sure it was necessary.

Prework Phase - Infrastructure

- Infrastructure set up with ics branch initially pointed to honeycomb. Allowed us later to easily repoint once ics was released and mirrored in.
  - Created ics branch in manifest project
  - Created ics/master branches for all HC projects

- We developed scripts to easily add new mirrored branches – for every mirrored project that has honeycomb-mr2-release, create new mirror for ics-mr0. This allowed us to rapidly add new mirrors when source was made available.

ICS released!

- Infrastructure pre-work plan put into effect – mirrors were up within one hour.
- First place to look before the mirroring was done – manifest project. Check for new/removed projects and modify our mirrors accordingly.
- Code distribution – save .repo/ directory to a usb stick and distribute that way. Having 10 people run multithreaded repo syncs will take a while.
  - Copy .repo/, then "**repo sync --local**"
- We created a temp branch at this point and used that for the early bring up. Having this branch plus lots of hacks and workarounds made it easier to clean up later.

Early tasks – fixing the build

- First order of business once code is distributed is to successfully build. How to parallelize this task?
- We used "**make -k**" and a whiteboard to collect the build errors. As one was encountered, someone was assigned to fix it and the rest continued on. Communication was critical in this phase.
- The engineer assigned to resolve the build issue had two options: work around the error or fix it properly.
- When a workaround was uploaded, the patch was tracked as a bug so that a proper fix would be uploaded later.

Fixing the build – cont'd

- Some components were simply excluded (HeightMapProfiler, for instance – non-essential sample app). Others were quickly fixed and pushed to Gerrit.
- This is where we also ported any necessary build project patches – mostly related to our in-tree kernel build.
- We still used Gerrit for patches. Code review is valuable even when going very fast. Catching problems early saves debugging time later.
- Build was successful within a few hours of source code availability.

Getting through the boot

- Our prework paid off and kernel was booting with minimal changes to init scripts.

- Android init – parses init.<bootmedia>.rc, init.rc, and init.<boardname>.rc
  - Action triggers: early-init, init, early-fs, fs, post-fs, post-fs-data, early-boot, boot, property changes,
  - Services: classes of services (core, main), onrestart, oneshot
  - This is a key place to disable zygote to test the kernel independent of the Android userspace. Add "**disabled**" to service zygote.

Runtime errors

- SystemServer failed to start, due to unmet service dependencies. In many cases, the problematic Android services can be excluded by not starting them in the first place (harder to do with core services like ActivityManager).
- Located in
  - frameworks/base/services/java/com/android/server/SystemServer.java
- All the core Android services are started in the system server. Non-critical ones that were failing (in our case, Nfc) can be temporarily commented out so that system_server will start.

Runtime errors

- In Android 4.0, Tinyalsa was introduced as the audio HAL (we had previously carried forward alsa_sound that was deprecated in earlier versions of Honeycomb).

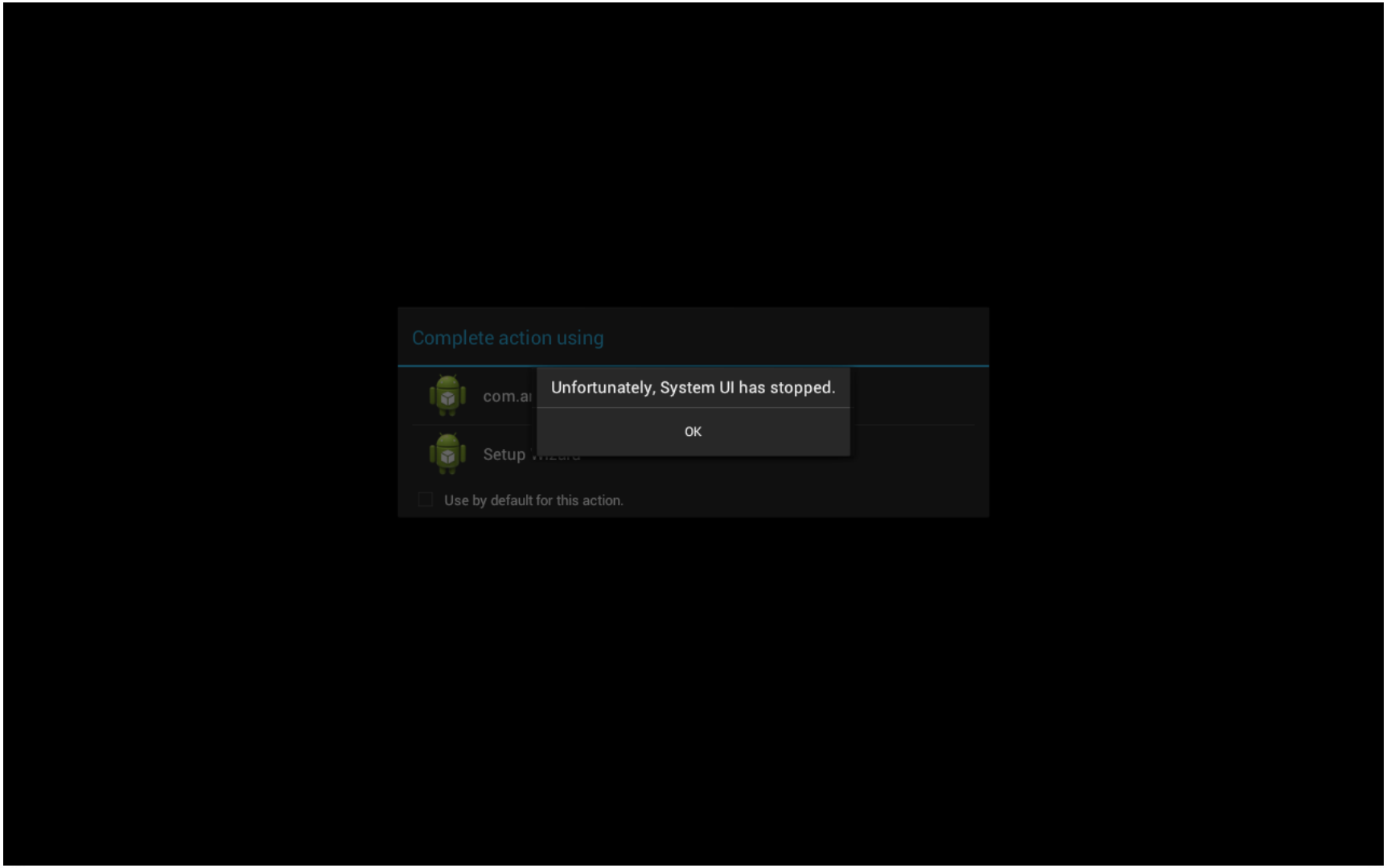
- A work-around was used, using the generic Android HAL and a dummy kernel sound card.
  - BOARD_USES_GENERIC_AUDIO:=true # BoardConfig.mk
  - CONFIG_SND_DUMMY=y

- Major changes were added in the camera HAL, so we also used the stub:
  - USE_CAMERA_STUB:=true # BoardConfig.mk

Booting to UI

- The Graphics were not usable at first, so a temporary solution was needed to unblock the UI.
- In this case, PixelFlinger was used. This was forced by modifying egl.cfg (**0 0 android**) and removing all PowerVR libraries from $(PRODUCT_PACKAGES).
- We also modified gralloc framebuffer to hardcode screen dimensions, because kernel mode-setting led Android to assume double buffering was available.
- Having a functional, albeit slow, SW-rendering UI brought to our attention that the touchscreen protocol changed and unblocked other UI dependent tasks.

Progress!!



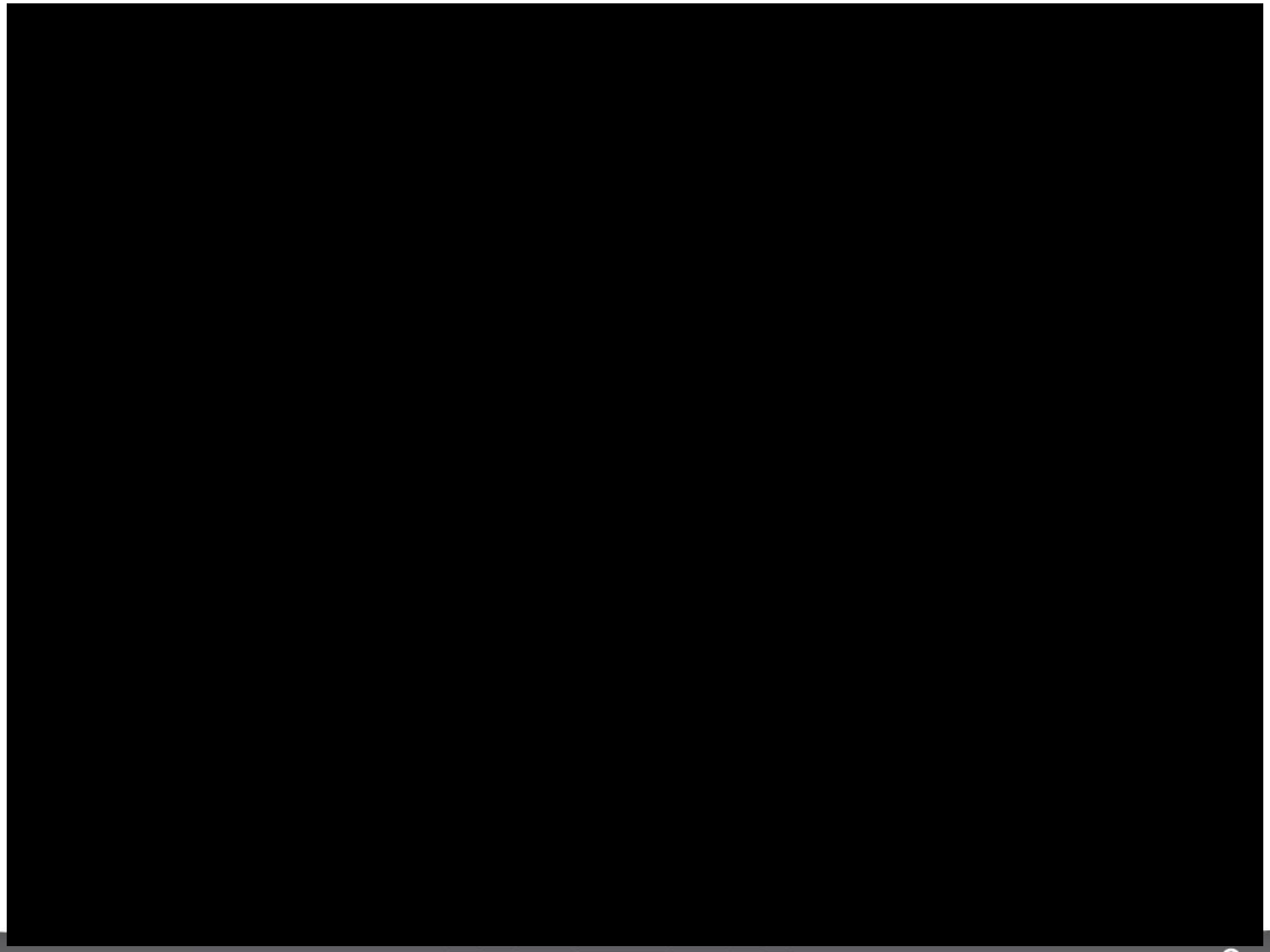
Getting through the boot, continued

- Hardware vs. Software UI Rendering.
  - Since Android 1.0, the window compositing was HW accelerated, but not the actual content drawing. As the resolution used increased, the software rendering became a bottleneck. So, starting with Android 3.0, Hardware Accelerated UI rendering was added, but apps would “opt-in” to HW acceleration. It was not enabled by default.
  - With ICS, Hardware UI Rendering is enabled by default.
  - This can be changed from your BoardConfig.mk:
    USE_OPENGL_RENDERER := true | false

- The HW UI Rendering needs special support from EGL, so this was a key first step for us in booting to the UI. We were able to boot using the SW rendering path first, while our graphics team sorted out the HW path.

- HW rendering was functional 3 hours after the SW path was functional. During this critical period, we were also able to fix the touchscreen driver and resolve several other issues only possible with a functioning UI.

Success!!



Key places to look

- Touchscreen
  - ICS expected protocol B, whereas HC was using protocol A
  - Input Device Configuration (idc) files define properties of the touchscreen, file goes in /system/usr/idc

  Example (from http://source.android.com/tech/input/input-device-configuration-files.html)

  # This is an internal device, not an external peripheral attached to the USB
  # or Bluetooth bus.
  device.internal = 1
  # The device should behave as a touch screen, which uses the same orientation
  # as the built-in display.
  touch.deviceType = touchScreen
  touch.orientationAware = 1
  # Additional calibration properties...
- Touchscreen vendors should be able to assist in tuning the (many) other parameters in this file. The above is the bare minimum.

Key places to look

- Storage volumes
  - Many devices have large internal storage, as well as external storage capabilities
  - Starting with 3.2, storage_list.xml defines the storage volumes available on the device, FUSE and sdcardd used to expose /data/media as /mnt/sdcard

```
<StorageList xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- removable is not set in nosdcard product -->
    <storage android:mountPoint="/mnt/sdcard"
        android:storageDescription="@string/storage_internal"
        android:primary="true"
        android:emulated="true"
        android:mtpReserve="500"/>
    <storage android:mountPoint="/mnt/sdcard2"
        android:removable="true"
        android:storageDescription="@string/storage_sd_card"
        android:emulated="false" />
    <storage android:mountPoint="/mnt/usbcard"
        android:removable="true"
        android:storageDescription="@string/storage_usb"
        android:emulated="false" />
</StorageList>
```

Key places to look

- Android configuration and device overlay
  - Many Android properties controlled by frameworks/base/core/res/res/values/config.xml
    - Tethering/networking configuration
    - Animations, Icons, UI behavior
  - Changes to config.xml should not go in this file, use the overlay mechanism
    - DEVICE_PACKAGE_OVERLAYS – set to path containing overlay for a device. All values in this overlay will override the values in the original xml file
    - PRODUCT_PACKAGE_OVERLAYS – you may have multiple SKUs for a single device, use this to override values specific only to a product.
  - Overlay can be used to replace/override any frameworks/app resource
    - frameworks/base/packages/SettingsProvider/res/values/defaults.xml
    - frameworks/base/packages/SystemUI/res/values/config.xml
    - packages/apps/Launcher2/res/values/config.xml
  - AOSP example – device/samsung/tuna/overlay/

Key places to look

- Product capabilities
  - PackageManager looks at the capabilities (among other things) to perform market filtering.
  - CTS tests try to verify that these are correct also.
  - frameworks/base/core/data/etc
  - http://developer.android.com/guide/topics/manifest/uses-feature-element.html

- ro.sf.lcd_density
  - Virtual screen density with discrete values. Providing a different value is possible and can lead to some strange results.
  - DENSITY_LOW = 120
  - DENSITY_MEDIUM (default) = 160
  - DENSITY_TV = 213
  - DENSITY_HIGH = 240
  - DENSITY_XHIGH = 320

Key places to look

- Disable power management from init.<board>.rc
  - write /sys/power/wake_lock 1
  - This was helpful to rule out any power management issues
- Trouble with touchscreen? Have host mode working.
  - Keyboard and mouse support can help with early bringup.
- Enable Dalvik portable mode:
  - adb shell "**echo dalvik.vm.execution-mode = int:portable >> /data/local.prop**“
  - Can also do this at build time by setting the property
- Hardware UI Rendering
  - frameworks/base/libs/hwui
- Temporarily disable strictmode – lots of verbose log output can make it hard to discern the real issues.
  - persist.sys.strictmode.disable = true

Cleaning up

- The value of a temp branch is that we can leave all the workarounds we had to do for bring up in the git history for that branch, but clean up when we pull those changes back to our master branch.

- We selectively cherry-picked changes and performed clean up on the work done during bring up as it was pulled into master.

- Lifespan of the temp branch was about 48 hours. After that, it was back to mainline development with the whole team ready to roll.

Lessons learned

- Having the right expertise was critical. My team is very strong on Linux fundamentals. Without their knowledge and help, all of the planning in the world couldn't have gotten it done.
- Co-location and real-time communication was critical.
- No team meetings, but short engineering huddles, to go over issues in a very targeted fashion, were productive.
- Always maintain focus on parallelizing and unblocking people.

Questions?

Russell Webb – Software Engineer
russell.webb@intel.com

Legal Disclaimer

INTEL OPEN SOURCE
TECHNOLOGY CENTER