# Android Stack Integration in Embedded Systems

Soumya Kanti Datta
Mobile Communication Department
EURECOM
Sophia Antipolis, France
Soumya-Kanti.Datta@eurecom.fr

***Abstract*—Smartphone usage has increased manifold upon introduction of Google's Android. Since its introduction, Android has evolved at an outstanding pace in terms of application development, commercialization and market share of Android powered devices. Although originally developed for smartphones, now the embedded system industry has realized the capabilities of Android. Due to open source nature, rich user interface, wide range of connectivity, secure communication, data encryption and multitasking, Android is being integrated and ported to various embedded systems. These include set-top boxes, IPTV, Google TV, In-Vehicle Infotainment systems. One major advantage of such integration is that an Android app can address the functionalities all these devices powered by Android and developers need not to write several applications for different embedded systems. These systems will also benefit from Android power management capabilities. This paper makes an attempt to promote Android software stack as a suitable operating system for embedded systems. The Linux kernel modifications introduced in Android are described in details. The procedure to integrate Android stack in an embedded system are outlined and Android porting is also briefed. The power management benefits are also pointed out. Finally the paper concludes with several important advantages of such embedded system with android stack.**

***Keywords- embedded system; Android stack integration; Android power management; Linux kernel;***

## I. Introduction

Tracing the history reveals that Android Inc. was bought by Google Inc. who pioneered Android platform. Later in 2007, Google along with 78 international firms announced the formation of Open Handset Alliance (OHA). The founding members include T-Mobile, eBay, Google, Broadcom Corporation, Intel Corporation, Nvidia Corporation, Qualcomm, Texas Instrument, LG, Samsung, Sony, Motorola etc. OHA has contributed to the research & development of Android platform which is mostly covered by BSD and Apache licenses. Since the initial release of Android software stack, it has come a long way with new versions releasing every few months. The most current version is Android Ice Cream Sandwich. Until very recent times, Android was targeted for smartphones and tablets only. The entire software stack, application development tools were designed and developed for those devices only. But the sophisticated features of Android (e.g. user interface and connectivity, secure communication, network stack) have made it lucrative for other embedded devices also. Telecom, medical, automotive and home application devices are potential candidates for Android integration and porting.

Being an open source system is the first advantage of Android integration into embedded systems. Also Inter Process Communication (IPC), multitasking, rich User Interface (UI), OpenSSL for secure communication, SQLite for database and 2G/3G/Wi-Fi/Bluetooth connectivity are making the platform attractive for any kind of embedded systems. Such systems which already have a legacy system running on top of Linux, can integrate Android into their systems. The integration process modifies the underlying Linux kernel to support Android specific features and merges Android stack to the legacy system. As a result, both work simultaneously but independent of each other. This is very important aspect and should be carefully done so that inclusion of Android does not affect the functioning of the present legacy system. Initially, Android supported only ARM embedded architecture. But due to its mentioned advantages, the platform got successfully ported to other architectures also. Porting of Android to different embedded architecture is also briefed in this paper. Due to wider adaptation, now-a-days, Android has found its presence in devices including set top boxes, cars & In Vehicle Infotainment [10], IPTV, Google TV, tablet computers, e-readers and other embedded devices. The features of Android which benefit the embedded systems used in telecom, automobile, medical, home application are listed below.

- For telecom devices, it provides a complete solution for network stack, connectivity, secure communication using SSL, apps to monitor the traffic and many more.
- For futuristic smart cars, Android is the natural choice as it includes GPS, motion sensors, apps giving map of places. Location based applications could inform about the nearest hotel, car parking, gas station and more. Android also supports audio and video which could be a part of in vehicle infotainment.
- With the advent of telemedicine, the medical embedded systems require easily operating user interface (UI) and reliable connectivity. Android provides a rich UI and also several connectivity options like 2G, 3G and Wi-Fi.

This work is partially sponsored by French research project "Smart 4G Tablet" Pole SCS.

- Home gateway devices that deal with digital audio and video could use Android as it provides several audio and video libraries and applications to play them.
- With the advent of Android Open Accessory Development Kit (ADK), unveiled in Google I/O 2011, developers can build custom hardware to be controlled by Android [1].
- Android comes with its own power management extension which is suitable for embedded devices running with low power and increase battery life.
- An Android app (created to perform some specific tasks) can function on various embedded devices running Android even if they have different embedded architecture. The different devices (which might have different architectures) can communicate to each other through one Android app. This could be achieved by making minimal changes made in the app for different device specific architectures. This simplifies maintenance of application itself, cost and other overheads.

Thus, the embedded industry has been gradually shifting its focus on Android for immense benefits. Rest of the paper is organized as below. Section II highlights some important underlying features of Android stack that further motivate Android integration to embedded devices. The Linux kernel additions are described in detail in section III. Section IV discusses in details the Android integration steps, challenges and their solutions while section V briefly touches the Android porting issue. Section VI describes the advantages of Android integration specially benefits of power management.

## II. ANDROID STACK OVERVIEW

The Android stack is composed of Linux kernel, native libraries, android runtime, application framework and application layer as depicted in Fig. 1 [2].

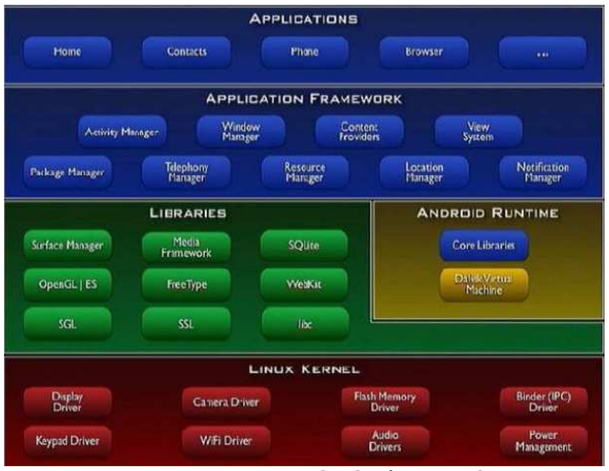


Figure 1. Android software stack.

Android versions till Honeycomb rely on Linux kernel 2.6 while Android Ice Cream Sandwich is based on kernel 3.0. Linux kernel is chosen as it provides several device drivers, memory & process management, network stack, security and other core services. Google has modified the Linux kernel 2.6.33 to address efficient power, memory and runtime managements for mobile devices and by extension other embedded devices.

The native libraries (e.g. libc, libm) are written in C/C++. These libraries are designed and developed to run on devices with limited power and main memory. Main libraries include surface manager, 3D libraries (OpenGL ES), Media libraries (Mpeg4, Mp3, JPG etc.), Libwebcore (for web browser), SQLite.

Android runtime is composed of Dalvik virtual machine (VM) and core libraries written in Java [11]. Again the constraints of mobile systems like limited power, memory have played pivotal role behind the birth of Dalvik VM. Android SDK incorporates a tool ‘dx’ that converts java byte codes from .jar form to .dex which runs on the Dalvik VM. Introduction of the Dalvik VM is unique to Android and the former is capable of executing any java based application of the embedded system quite efficiently.

The capabilities of native libraries are exposed to the developers through applications framework. The components (e.g. activity manager, telephony manager, content providers) are written in Java. The topmost layer in the Android architecture contains all the applications used by the end-users.

## III. ANDROID LINUX KERNEL

As mentioned, the Linux kernel 2.6.33 is modified as per the special needs of smartphones. But these additions are also beneficial other embedded systems. The Android kernel introduces changes in memory management, adds new features (e.g. logger, alarm) and runtime power management driver (wake locks). Following gives the Android kernel specific features [3].

### A. *Efficient Memory Management*

The main goal behind such changes is to ameliorate the memory usage as the amount of RAM available in smartphones and embedded systems is limited. Android introduces two features e.g. ASHMEM and PMEM which are two different ways of allocating memory to kernel. Also the standard Out of Memory (OOM) feature of mainline kernel is modified and low memory killer is added to Android kernel.

- **ASHMEM** – The Anonymous Shared Memory (ASHMEM) is used to provide shared memory by allocating a named memory block that can be shared across multiple processes. The advantage of ASHMEM is that it can be freed by kernel. To use

ASHMEM, a process opens "/dev/ashmem" and performs mmap() on it. mmap() is a system call that maps files or devices in memory.

- **PMEM** – The Physical Memory (PMEM) on the other hand allows allocating contiguous memory to drivers and libraries.
- **Low memory killer** – It mainly informs running processes to save their state in case a critical low memory situation occurs. When worsened, it starts to terminate processes with low importance.

### B. *Runtime Power Management*

Although Android inherits the power management (PM) of Linux kernel but the former has put forward its own PM system [7], [9]. Again the motivation behind such advanced PM is that Android will run on devices having limited battery life and the power saving features is different than personal computer. A power driver has been added to the Linux kernel and the driver allows controlling the peripherals: screen display & backlight, keyboard backlight and button backlight. The power for peripherals is controlled by "Wake Locks" which are requested by applications through a power management API present in applications framework layer. Wake locks are means through which applications keep the screen on, the CPU stays awake to react quickly to interrupts.

### C. *Other Additions and Modifications*

The other additions and modifications to the kernel are described as below.

- **IPC Binder:** Although standard Linux kernel has Inter Process Communication (IPC) mechanism, Android adds its own IPC. The Android IPC implementation is based on OpenBinder and has the advantage of being light weight. The binder driver uses shared memory to pass the messages between threads and processes [8].
- **Logger:** Android extends the logging capabilities by adding four logging classes e.g. main, system, event and radio.
- **Alarm:** A driver which provides timers that can wake the device up from sleep and a monotonic time base that runs while the device is asleep.
- **RAM_CONSOLE:** Gives ability to save console output to a reserved ram area for diagnostics on a subsequent boot.
- **Timed output/gpio:** It allows chaining a gpio pin and restores it automatically after a given timeout and exposes a user space interface used by vibrator code.
- USB gadget driver for Android Debug Bridge (ADB).

## IV. ANDROID STACK INTEGRATION

This section describes the Android stack integration process for an embedded system that has a legacy system based on Linux. The main outcomes of such Android integration are primarily manifold as mentioned below.

- Integration of majority of the Android stack in the embedded system.
- Execution of Android applications on the embedded systems.
- Android running in parallel to the legacy system without affecting its execution.

The entire integration process can be sub-divided into following phases:

- Embedded device requirements
- Preparing Linux kernel to support Android
- Resolving the dependency on display hardware
- Configuring & building Android sources for an embedded system

### A. *Embedded device requirements*

The minimum requirements for an embedded device to run Android stack are listed in Table 1.

TABLE I. EMBEDDED DEVICE REQUIREMENTS

| **Feature** | **Requirement** |
|---|---|
| Chipset | ARM-based |
| Memory | 128MB RAM, 256MB external Flash |
| Display | TFT LCD, 16-bit color |
| Navigation keys | 5-way navigation with 5 application keys, power, camera and volume controls |
| USB | Standard USB interface |

Android integration requires that the device should have ARM chipset. Many embedded devices do not have a display and navigation keys and Android integration in that case poses a problem. But that can be addressed by including virtual display, keypad & power drivers and it is discussed later.

### B. *Preparing Linux kernel to support Android*

The unique Android kernel features must be added separately to Linux kernel 2.6.33. In this work it is assumed that the embedded system has kernel 2.6.33. The addition could be done using a patch containing all the Android specific features. Most of these drivers are available as open source. The added files are listed in table 2.

TABLE II. FILES TO BE ADDED TO LINUX KERNEL AND THEIR PURPOSE

| Name of the file added | Purpose |
|---|---|
| drivers/staging/android/binder.c | Introduces Android IPC Binder subsystem |
| drivers/staging/android/binder.h | Header file for binder.c |
| drivers/staging/android/logger.c | Introduces the logging system for Android |
| drivers/staging/android/logger.h | Header file for logger.c |
| drivers/staging/android/lowmemorykiller.c | Adds low memory killer driver |
| drivers/staging/android/Kconfig | Contains Android Configurations |
| drivers/staging/android/Makefile | Makefile to build the sources |
| drivers/staging/android/ram_console.c | Ability to save console output to a reserved ram area for diagnostics on a subsequent boot. |
| drivers/staging/android/timed_gpio.h | Header file for timed_gpio.c |
| drivers/staging/android/timed_gpio.c | This exposes a user space interface for timed GPIOs. It is used in the vibrator code. |
| drivers/staging/android/timed_output.c | Used to calculate timed output |
| drivers/staging/android/timed_output.h | Header for timed_output.c |
| include/linux/ashmem.h | Serves as header for ashmem.c |
| mm/ashmem.c | Adds ASHMEM driver |
| drivers/misc/pmem.c | Implementation of process memory allocator |
| include/linux/android_pmem.h | Header file for pmem.c |
| kernel/power/wakelock.c | Used for power management files. |
| drivers/usb/gadget/android.c | USB gadget driver for ADB |
| include/linux/wakelock.h | Header file for wakelock.c |
| drivers/rtc/alarm.c | To support Android alarm manager |
| include/linux/android_alarm.h | Header file for alarm.c |

Then existing kernel configuration file of the legacy system of the embedded system has to be modified to take into account the new features [4]. The file should contain the following:

```
CONFIG_ANDROID_KERNEL_CORE=y

#
#Android
#
#CONFIG_ANDROID_RAM_CONSOLE is not set
CONFIG_ANDROID_POWER=y
CONFIG_ANDROID_BINDER_IPC=y
CONFIG_ANDROID_LOGGER=y
CONFIG_ASHMEM=y
#CONFIG_ANDROID_RAM_CONSOLE is not set
#CONFIG_ANDROID_TIMED_GPIO is not set
```

It is to be noted that the above Android kernel configuration varies depending upon the embedded system and its usage.

### C. *Resolving the dependency on display hardware*

Android stack has a strong **dependency on hardware** of a smartphone. It is evident that the largest dependency is on the touch screen as user interacts via the touch screen. A majority of the embedded systems still does not contain any display. This challenge can be overcome by either adding a USB based touch screen along with its driver or including virtual display drivers [12]. In either case, the driver(s) must be added to the Android Linux kernel using a patch. It is possible to remove the hardware dependencies, but then there will not remain anything useful in the Android stack to integrate in the embedded system. It is easier to add virtual drivers for display along with virtual keypad and power drivers to the kernel. Adding virtual drivers is essential for integrating Android embedded systems that does not include a touch screen. These drivers could be developed as per the embedded system.

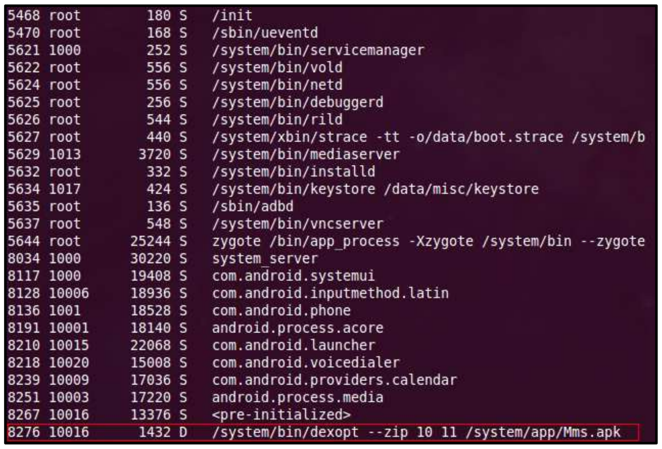
### D. *Configuring & building Android sources for an embedded system*

The next step is to download the android sources and configure them to run on the specific embedded system. The necessary tools required to download and compile Android sources are explained in [5] and [6]. This paper gives a complete overview of the source code configuration [13].

Basically configuration files related to the embedded system are to be added to the downloaded Android sources. The added files are:

- A product specific makefile: it includes the product name and product device where the product is the embedded system.
- AndroidProducts.mk: this file points to individual product makefile.
- system.prop: it is used in case the developer wants to modify any system properties.
- product_config.mk: contains product specific definitions and without this file the Android build system will simply fail.
- Android.mk: it is the make file for Android build for the new embedded system.

After that the Android sources should be built and if the steps till now have been followed correctly, the Android root file system will be created successfully. The Android root file system should be merged with the root file system of the legacy system running on the embedded system. If include path of the libraries of Android and legacy system overlap, the resulting system will not work. Efforts must be given to identify such issue although normally the include paths of libraries are different for both the systems. After the system boot up when the shell is available, “ps” command can be used to check Android daemons (e.g. logd, adbd, rild, vold etc.), applications (e.g. servicemanager, mediaserver), system_server and Java applications (e.g. com.android.phone) that are running. Figure 2 portrays such a scenario.



```
5468 root        180 S   /init
5470 root        168 S   /sbin/ueventd
5621 1000        252 S   /system/bin/servicemanager
5622 root        556 S   /system/bin/vold
5624 root        556 S   /system/bin/netd
5625 root        256 S   /system/bin/debuggerd
5626 root        544 S   /system/bin/rild
5627 root        440 S   /system/xbin/strace -tt -o/data/boot.strace /system/b
5629 1013       3720 S   /system/bin/mediaserver
5632 root        332 S   /system/bin/installd
5634 1017        424 S   /system/bin/keystore /data/misc/keystore
5635 root        136 S   /sbin/adbd
5637 root        548 S   /system/bin/vncserver
5644 root      25244 S   zygote /bin/app_process -Xzygote /system/bin --zygote
8034 1000      30220 S   system_server
8117 1000      19408 S   com.android.systemui
8128 10006     18936 S   com.android.inputmethod.latin
8136 1001      18528 S   com.android.phone
8191 10001     18140 S   android.process.acore
8210 10015     22068 S   com.android.launcher
8218 10020     15008 S   com.android.voicedialer
8239 10009     17036 S   com.android.providers.calendar
8251 10003     17220 S   android.process.media
8267 10016     13376 S   <pre-initialized>
8276 10016      1432 D   /system/bin/dexopt --zip 10 11 /system/app/Mms.apk
```

Figure 2. Android components running on an embedded system.

It is worthy to note that on first booting of Android, the process “dexopt” optimizes all the java byte codes and stores them in the system. Next time when the device is booted, the optimized java codes are used and the system boots up and works faster.

## V. ANDROID PORTING

This section briefs Android porting to a Linux system with a previously unseen CPU architecture [14]. In this case also, attention should be paid to the following issues:

- Firstly, the target Linux kernel needs to be prepared i.e. patched with Android specific kernel features.
- The Dalvik Virtual Machine (VM) needs to be accurately ported to the new CPU. The Dalvik VM runtime is written in portable C, but Java Native Interface (JNI) Call Bridge of runtime is non-portable. This could be worked around using the open source Foreign Function Interface (FFI). Dalvik VM has dependencies on Android core libraries including OpenSSL, zlib and ICU. These libraries also need to be ported.
- The native libraries must be optimized to suite the new CPU and then ported. Also additional support might be necessary for applications framework APIs.

Android sources provide internal documentation which elaborates such porting to unseen CPU architectures.

## VI. ADVANTAGES OF ANDROID INTEGRATION

Embedded systems that have Android integrated into them benefit from several advantages.

- For devices without a display, a USB based touch screen can be added. This would open new vistas of application for the device. The device could be configured at real time using an Android app. The configuration could be done by interacting through the touch screen or by remotely sending commands. The status of the device could be displayed. An Android application could be written to control some features of the device. For example, if the device is being used as router, the app can monitor the packets passing through and generate some statistics.
- Embedded systems could execute standalone java modules in Dalvik VM.
- Devices that are deployed in rural areas could use the Android connectivity and secure communication features to report collected data to a command center.
- During I/O 2011, Google announced that they are extending Android for home automation. Android 4.0 is already deployed for Google TV which promises to revolutionize the TV experience of users. It would be also possible to customize the hardware capabilities and use the Android platform to control home appliances.
- Another important advantage is increasing power efficiency of embedded systems. Android has its own PM features to control the CPU resources and is superior to the Linux PM. There are numerous power widgets available that increase power efficiency by controlling several features of the embedded device. The usage of Wi-Fi, 2G/3G, brightness of display (if any) could be intelligently controlled by an Android app since these consume high power. Specific app could be produced to monitor the power consumption pattern of embedded systems and automatically control mentioned features to conserve the battery life. Thus the battery life will be prolonged and impact on environment will be reduced.

## VII. CONCLUSION

In a nutshell, this paper brings into attention the benefits of integrating Android software stack into different embedded systems. The different features of Android that can benefit the telecom, residential, automobile and medical devices are presented. Developers can build Android applications that will address specific functionalities of these systems. Android specific kernel additions are described in detail and kernel configuration file modification is shown with an example. Developers can modify the configuration file according to the kernel requirement. The files that should be added to the Linux kernel 2.6.33 are listed along with their purpose. One important challenge during Android integration is to work around the dependency of Android stack on display hardware (i.e. touch screen). The solution is to add virtual display drivers (to the Linux kernel) that emulate the real hardware drivers. The Android source configuration for the device and Android porting to unseen CPU architecture is briefed. Such devices will also benefit from the intelligent power management applications. Other notable advantages of such integration are highlighted.

In future work, a power management Android application will be developed that monitors the power consumption. The power consumption pattern could be studied and the power dissipated in various hardware components (display, Wi-Fi etc.) could be modeled. An app could be developed that controls power dissipation using the power models. Using IPC mechanism, the application will be able to communicate with legacy system also. Different approaches for power saving and their impact on performance will be evaluated.

## Acknowledgment

The author thanks Mr. Antoine Boiteau & Mr. Yannick Vignon of Mindspeed Technologies and Prof. Yves Roudier, EURECOM for their valuable guidance during an internship on Android integration. The author also expresses his gratitude to Prof. Christian Bonnet, EURECOM for his valuable suggestions on Android power management.

## References

[1] http://betanews.com/2011/05/10/google-extends-android-into-embedded-hardware-home-automation/

[2] http://developer.android.com/guide/basics/what-is-android.html

[3] http://elinux.org/Android_Kernel_Features

[4] http://www.kandroid.org/online-pdk/guide/bring_up.html

[5] http://source.android.com/source/initializing.html

[6] http://source.android.com/source/downloading.html

[7] http://developer.android.com/reference/android/os/PowerManager.html

[8] http://en.wikipedia.org/wiki/OpenBinder

[9] www.elinux.org/Android_Power_Management

[10] http://www.autoblog.com/2011/07/15/harman-to-bring-android-integration-to-cars-finally

[11] http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf

[12] http://www.netmite.com/android/mydroid/development/pdk/docs/display_drivers.html

[13] http://www.netmite.com/android/mydroid/development/pdk/docs/build_new_device.html

[14] http://www.kandroid.org/online-pdk/guide/dalvik.html