Android for Non-Mobile Device

Ms Kalpana Saini.
shelly_saini18@yahoo.com

Mr. Noor Mohammed.
S.noor.mohammed@gmail.com

Mr. Swapnesh Taterh
swapnesh_t@hotmail.com

Shekhawati Engineering College, Dundlod, Rajasthan, India

Abstract**—**This paper is an effort to represent Android as a powerful software stack that can be used in non-mobile platforms. Android is a software platform on mobile devices by Open Handset Alliance (Google). It includes an operating system based over Linux, middleware and key applications such as email client, calendar, maps, browser, and contacts. Besides mobile phones, we are surrounded by number of digital gadgets like set-top boxes e-book readers, net books, digital photo frames etc. that require internet for content transfer like traffic, photos, map, music, videos, news etc. As many of these devices are portable, low power consumption, small form factor, intuitive UI, larger displays and time-to-market become major differentiators. Simplicity and user-friendliness are the keywords for these gadgets. Android presents a compelling value proposition in bringing internet connectivity and a broad range of applications to consumer devices. It helps device manufacturers to build innovative products faster. Since Android was designed for low CPU usage and memory constrained mobile phones, it is well suited for consumer devices. The Android application framework and SDK now extend beyond the handset assumptions for which it was initially developed. This paper evaluates one of the key feature requirement in of high end consumer devices i.e. display capabilities of android. A proof of concept is developed to support high resolutions on devices having larger screens than mobile phones.

Keywords: **Linux Kernel**, **Android, open-source, Dalvik, OMAP3EVM**

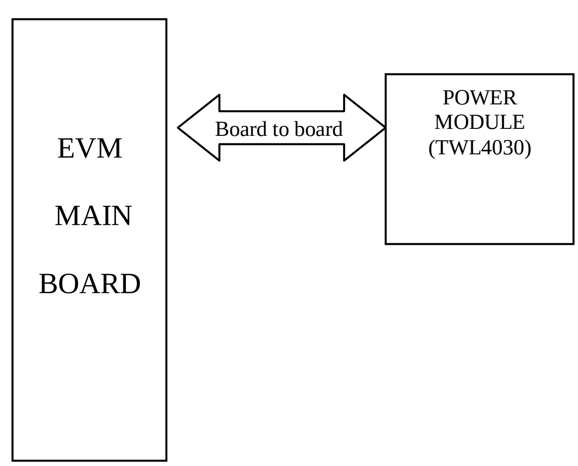
1. Hardware and Software Specifications and Android Porting

This topic covers hardware and software setup to evaluate android for non-mobile devices. It also discuss about hardware booting and android porting steps on the OMAP3EVM hardware.

1.1 Hardware Details

The hardware board chosen to display android capabilities is Texas Instrument's OMAP3EVM. This platform is based on the ARM® Cortex™-A8 core.

The OMAP™ 3 architecture is designed to provide best-in-class video, image, and graphics processing sufficient to support the following:



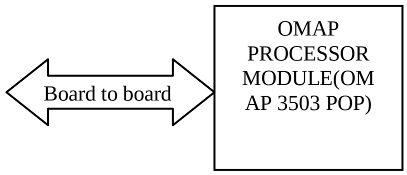


Figure 1.1 System Block Diagram OMAP35x EVM

* Streaming video
*2D/3D mobile gaming
* Video conferencing
* High-resolution still image Applications:
* Portable Navigation Devices
* Portable Media Player
* Ultra Mobile Devices
* Advanced Portable Consumer Electronics
* Gaming

1.2 EVM Hardware Setup

This section explains the process of setting up the EVM hardware for the purpose of running android. The information is the same as in the supplied Setup Guide.

1.3Main Board SW4

The main board's SW4 DIP switch controls the boot mode of OMAP3 processor (Figure6). The default setting shown above will try to boot from UART3. If no response is seen in a short time (< 1 S) the processor will attempt to boot from the attached flash memory.Consult the Hardware User's Guide for details and for other settings. Please ensure to match the numbers in the diagram to the numbers on the DIP switch as the orientation of the switch may not be what will expect.

1.4 Setup Terminal Program

A serial port terminal program should be used to communicate with the EVM's serial port console. For Windows users HyperTerminal or TeraTerm are recommended. For Linux users Minicom is recommended. In any case, the serial modem settings are the same:

Booting Introduction
Data bits: 8
Parity: None
Stop bits: 1
Flow control: none

The OMAP processor follows a 2 stage boot process. The first stage is loaded into the internal static ram by the ROM code. Because the internal static ram is very small (64k), the first stage loader is needed to initialize memory and enough of the peripheral devices to access and load the second stage loader into main memory. It is the job of the second stage loader to initialize the remaining hardware and prepare the system for kernel boot

2 SD Card Boot

Assuming there was no answer from the host during serial boot, the ROM looks for an SD Card on the first MMC controller. If a card is found, the ROM then looks for the first FAT32 partition within the partition table. Once the partition is found, the root directory is scanned for a special signed file called "MLO". Assuming all is well with the file, it is transferred into the internal sram and control is passed to it. The SD Card x-loader looks for a FAT32 partition on the first MMC controller and scans the top level directory for a file named "u-boot.bin". It then transfers the file into main memory and transfers control to it. Since putting a Linux file system on a FAT32 partition is problematic, it is recommended to create 2 partitions. The first partition is boot partition between 64-128 Megabytes and the second partition is a Linux partition consuming the rest of the card.

Fdisk drive and print partition information fdisk /dev/sdc
Command (m for help): p
Disk /dev/sdc: 1018 MB, 1018691584 bytes<more>...
Look for the size in bytes of the device and calculate the number of cylinders, dropping factions, if we have 255 heads and 63 sectors. new_cylinders = Size / 8225280 (for this example we will have 993001472 / 8225280 which equals 120.725 or 120 cylinders)

Since we are changing the underlying geometry of the disk, we must clear the partition table before doing it. So delete all partitions using the fdisk 'd' command - yes, all data on the card are lost. Once that is done, we can set the new geometry in expert mode

We will set the # of heads to 255, # of sectors to 63, and # of cylinders to new cylinders.

To delete partition use fdisk /dev/sdb1
Command (m for help):d

Even the gparted tool can be used to delete the existing patition on the SD-card.

Command (m for help): x
Expert command (m for help): h
Number of heads (1-256, default 30): 255
Expert command (m for help): s
Number of sectors (1-63, default 29): 63
Warning: setting sector offset for DOS compatibility
Expert command (m for help): c
Number of cylinders (1-1048576, default 2286): 120
Now we return to the main menu and create our 2 partitions as needed - 1 boot partition of 64Meg and the rest a linux partition.

Expert command (m for help): r
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p Partition number (1-4): 1
First cylinder (1-123, default 1):

Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-123, default 123): +64M

Command (m for help): n
Command action
e extended
p primary partition (1-4)
p Partition number (1-4): 2

First cylinder (10-123, default 10):
Using default value 10
Last cylinder or +size or +sizeM or +sizeK (10-123, default 123):
Using default value 123
Set the partition type of the first partition to FAT32 and make it active.
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))
Command (m for help): t

Partition number (1-4): 2
Hex code (type L to list codes): 83
Changed system type of partition 2 to 83 +LINUX
Command (m for help): a
Partition number (1-4): 1

You have to format 1st partition with vfat32 file system. You have to format 2nd partition with EXT3 LINUX file system.
The partition table should look something like the following. Notice the heads, sectors, and cylinders. Make sure partition 1 is active and FAT32. If it looks good - write the new partition information out.

Command (m for help): pDisk /dev/sdc: 993 MB, 993001472 bytes 255 heads, 63 sectors/track, 120 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x00000000

Device Boot Start End Blocks Id System
/dev/sdc1 * 1 9 72261 c W95 FAT32 (LBA)
/dev/sdc2 10 120 891607+ 83 Linux
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x partitions, please see the fdisk manual page for additional information.

Syncing disks.
Device Boot Start End Blocks Id System
/dev/sdc1 * 1 9 72261 c W95 FAT32 (LBA)
/dev/sdc2 10 120 891607+ 83 Linux
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x partitions, please see the fdisk manual page for additional information.

Syncing disks. Formating the partitions
Format the filesystems on the partitions mkfs.vfat -F 32 -n boot /dev/sdc1 mkfs.ext3 /dev/sdc2
use tune2fs -c 100 /dev/sdc2
/* to increase the maximum mount count */
Downloading the the OMAP Android Source :
This repository contains the following:
Android Source
Omap kernel source for the android Omap bootloaders source.(i.e, x-loader and u-boot source).

3 Proof of concept

This concept describes practical activity carried out by me to demonstrate android capabilities to be run on high end consumer devices requiring high resolution display3

3.1 Problem Statement

Android based devices, for the "goldfish" platform, has limits on the size of screens. Out of the box (actually the SDK) the largest screen support is around 800x600.By making few modifications in the android framework, we can emulate screen sizes larger than the800x600.

3.2 High resolution support: Android Native Libraries

Android software stack has native libraries placed over the linux kernel. These are set of C/C++ libraries used by components of the Android system. These libraries are exposed to developers through the Android application framework. Graphics management in android is done by Surface Manager also termed as Surface flinger. Surface flinger provides system-wide surface “composer” coming from different applications and handles all surface rendering to frame buffer device.

Surface flinger has ability to combine 2D and 3D surfaces and surfaces from multiple applications.

3.3 Graphics Management

Surface flinger has a Surface Heap Manager. Every client has a Memory Dealer, as returned by Surface Heap Manager .Every surface of a client also has dealer(s), from client or GPU. A dealer consists of a heap and an allocator. A heap represents a sharable big chunk of memory. And an allocator is an algorithm that returns heap chunks. Small chunks of memory from the heap are returned .So the real processing flow looks like

A client asks for a new surface, create Surface .create Surface calls create Normal Surface Locked A layer is created and set Buffers is called to allocate buffers. Two dealers are created from client, one for front buffers and one for back buffer. Two Layer Bitmaps are created, initialized with the two dealers. Heaps of dealers along with info about the layer are returned.

4. Results

After passing different parameters for different resolution, we found a set of resolutions being supported by android middleware. An analysis results obtained are presented in gives all the consolidated results obtained for different resolution Parameters

5. Conclusions

Definitely, Android stands an excellent chance of succeeding in the consumer electronic device market. Android is enabling a full internet experience on a broad range of consumer devices such as DTVs, set-top boxes, VoIP solutions, mobile internet devices (MIDs), digital picture frames, automotive infotainment, of a mobile devices that requires higher solution support.

6 References

[1] Shih-Hau Fang*" Developing a mobile phone-based fall detection system on Android platform* " Computing, Communications and Applications Conference (ComComAp), 2012, 11-13 Jan. 2012

[2] Amalfitano "*A GUI Crawling-Based Technique for Android Mobile Application Testing* Software" Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on 21-25 March 2011

[3] Erturk, E." *A case study in open source software security and privacy: Android adware "* Internet Security (WorldCIS), 2012 World Congress: 10-12 June 2012

[4] Ahmed Darwish**,** *Image Segmentation for the Purpose Of Object-Based Classification,*, 2003 IEEE,pp 2039-2041

[5] X. Wang, Z. Li, J. Y. Choi, and N. Li. PRECIP: Practical and Retrofittable Confidential Information Protection Against Spyware Surveillance. In *Proceedings of the 16th Network and Distributed System Security Symposium,* NDSS '08, February 2008

[6] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium*, NDSS '11, February 2011

[7] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege Escalation Attacks on Android. In *Proceedings of the 3rd Information Security Conference*, October 2010.

[8] L. Desmet, W. Joosen, F. Massacci, P. Philippaerts, F. Piessens, I. Siahaan, and D. Vanoverberghe. Security-by contract on the .NET platform. *Information Security Technical Report,* 13:25–32, January 2008.

[9] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming Information-Stealing Smartphone Applications (on Android)," in *Proceeding of the 4th International Conference on Trust and Trustworthy Computing*, 2011.

[10] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.

[12] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically Rich Application-Centric Security in Android," in *Proceedings of the 25th Annual Computer Security Applications Conference*.

[13] W. Enck, M. Ongtang, and P. McDaniel, "On Lightweight Mobile Phone Application Certification," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[14] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4Android: A Generic Operating System Framework for Secure Smartphones," in *Proceedings of the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011.

[15] J. Andrus, C. Dall, A. Van't Hof, O. Laadan, and J. Nieh, "Cells: A Virtual Mobile Smartphone Architecture," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011.

[17] A. Porter Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A Survey of Mobile Malware In The Wild," in *Proceedings of the 1st Workshop on Security and Privacy in Smartphone and Mobile Devices*, 2011.

[18] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf, "Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices," in *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011

[19] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the 19th Annual Symposium on Network and Distributed System Security*, 2012.

[20] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "DroidMOSS: Detecting Repackaged Smartphone Applications in Third- Party Android Marketplaces," in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy*, 2012

.

[21] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security," in *Proceedings of the 20th USENIX Security Symposium*, 2011.