**Porting :::**

The advantage of making Android available to

multiple device platforms would mean that an application developed for one device could easily be made

available for another platform with minimal porting needs (consider for example, a TiVo recording client

that is made available both on your set top box and one your phone – same application, same code but

available on two devices to suit your mobility needs)

>> Android is a Dalvik Virtual Machine based software platform that runs on a Linux based kernel. Therefore,

to port an Android platform, one needs to port the underlying Linux OS and then the Android platform

SDK as well

>> The minimum recommended requirements for porting Android on reference platforms are:

128 MB RAM

256 MB Flash Memory

Source: http://source.android.com/release-features

..........................................................................................................................

>> **Porting Android on ARM based target platforms can be divided in two stages:**

**Stage 1: Porting Linux**

Following are the main steps for porting Linux:

o Download the patches of Linux Kernel for supporting PXA310 processor.

o Prepare the ARM based tool chain for compilation of Linux source code.

o Choose and Configure boot loader as per target board.

o Compile the boot loader.

o Configure Linux kernel as per target platform (this may require developing some

additional device drivers or customization of the existing device drivers for the reference

design board) .

o Compile the Linux kernel source

o Burn compressed Linux image on the target platform

**Stage 2: Porting Android**

Following are the main steps for porting Android:

o Download the patches for Android

o Update RIL (Radio Interface Library) for target platform

o Update Board specific components such as Codec, Camera, Audio, Wifi, Power

Management, Bluetooth etc.

o Compile Android source code

o Burn system image on Target platform

......................................................................................................

>> **<PL\_Image>-<Version>-<Chipset>**

Where:

PL\_Image – LNX.LA.Branch for Linux Android

Version – Variable number of digits used to represent the build ID version.

Chipset – 8x16 for MSM8916/APQ8016E

**LNX.LA.3.7.3-00810-8939.1 (Build is from the 3.7.3.branch-version number-chipset)**

.........................................................................................

>> make –j8 to build everything

make –j8 aboot to build LK bootloader emmc\_appsboot.mbn

make –j8 bootimage to build kernel boot.img

make –j8 clean cleans the out directory completely

................................................................................................ss

>> Output is generated in $BUILDROOT/out/target/product/msm8916/. The list of files is as

follows:

emmc\_appsboot.mbn

boot.img

cache.img

userdata.img

system.img

persist.img

recovery.img

........................................................................................

>> Flashing methods

Android USB Driver (android\_winusb.inf)

Android adb Interface

Android Boot Loader Interface (fastboot)

....................................................................................

>> **Android device tree structure**

build/ – Build environment setup and makefiles

bionic/ – Android C library

dalvik/ – Android JVM

kernel/ – Linux kernel framework/

– Android platform layer (system libraries and Java components)

system/ – Android system (utilities and libraries, fastboot, logcat, liblog)

external/ – Non-Android-specific Open Source projects required for Android

prebuilt/ – Precompiled binaries for building Android, e.g., cross-compilers

packages/ – Standard Android Java applications and components

development/ – Android reference applications and tools for developers

hardware/ – HAL (audio, sensors) and Qualcomm specific hardware wrappers

vendor/qcom/ – Qualcomm target definitions, e.g., msm8916

vendor/qcom-proprietary/ – Qualcomm-proprietary components

out/ – Built files created by user

out/host/ – Host executables created by the Android build

out/target/product/<product> – Target files appsboot\*.mbn – Applications boot loader

boot.img – Android boot image (Linux kernel + root FS)

system.img – Android components (/system)

userdata.img – Android development applications and database

root/ – Root FS directory, which compiles into ramdisk.img and merged into boot.img

system/ – System FS directory, which compiles into system.img

obj/ – Intermediate object files

include/ – Compiled include files from components lib/ STATIC\_LIBRARIES/

SHARED\_LIBRARIES / EXECUTABLES / APPS /

symbols/ – Symbols for all target binaries

............................................................................

>> **Android source tree structure**

The Android source tree structure is laid out as follows:

/ – Root directory (ramdisk.img, read-only)

init.rc – Initialization configuration files (device config, service startups) init.qcom.rc

dev/ – Device nodes

proc/ – Process information

sys/ – System/kernel configuration

sbin/ – System startup binaries (adb daemon; read-only)

system/ – From system.img (read-write) – bin/ – Android system binaries

– lib/ – Android system libraries

– xbin/ – Nonessential binaries

– framework/ – Android framework components (Java)

– app/ – Android applications (Java)

– etc/ – Android configuration files

sdcard/ – Mount point for SD card

data/ – From userdata.img (read-write)

– app/ – User installed Android applications

– tombstones/ – Android crash logs

.......................................................................................................................

**>> Device tree**

A device tree is a tree structure used to describe the physical hardware in a system

>> With device trees, the kernel itself no longer needs specific code for each version of hardware. Instead, the code is located in a separate binary: the device tree blob. This enables us to target different hardware with the same kernel image by simply changing the much simpler, and much smaller, device tree binary.

>> The device tree can be passed to the kernel either through appending it to the kernel image or through the bootloader. The machine type is now defined in the device tree itself.

>> The device tree is a tree structure with nodes that describe the physical devices in the system that cannot be dynamically detected by software. The nodes are organized in a hierarchical parent/child relationship.

>> Many times, flash partitions are described in the device tree

>> .dts files are board level definitions. The .dts extension denotes “device tree source”.

>> dtsi files are files included by the .dts files and generally contain SoC-level definitions. Device tree files do not have to be monolithic; instead, they can be split into several files, including each other. By convention, .dtsi files are included files, containing definitions of SoC-level information, while .dts files are final device trees containing board-level information. The .dtsi extension denotes “device tree source include”.

>> The Device Tree Compiler (DTC) is the tool that is used to compile the source into a binary form. Source code for the DTC is located in scripts/dtc.

The output of the device tree compiler is a device tree blob (DTB), which is a binary form that gets loaded by the boot loader and parsed by the Linux kernel at boot.

On ARM ® and ARM ® 64-bit architectures, DTBs to be generated at build time are listed in arch/../boot/dts/Makefile, although they can be manually compiled by the DTC at any time.

...........................................................................................

**>> U - Boot**

U-Boot updates the flattened device tree (FDT) with platform-specific information, such as the information derived from the reset configuration word (RCW), environment variables, and hardware configuration. The most common areas that U-Boot touches are related to frequency, MAC addresses, LIODNs (Peripheral MMU settings), and memory size — although the actual fix-ups are board specific and are not documented in any place other than the U-Boot code. Within U-Boot, the main function where this all occurs is ft\_board\_setup().

............................................................................

**GIT Commands :**

**>> git log branchB..branchA** ( show the commits on branchA that are not on branchB)

**git log --follow [file]** ( show the commits that changed file, even across renames)

**git diff branchB...branchA** ( show the diff of what is in branchA that is not in branchB)

.......................................................................................

**Gerrit :**

>> Gerrit allows to review commits before they are integrated into a target branch.

Gerrit is built on top of Git it manages standard Git repositories and controls access and updates to them

>> Four eyes catch more bugs

Catching bugs early can save hours of debugging later

>> Coding standards Keep overall readability and code quality high

>> Gerrit allows to review commits before they are integrated into the target branch, but code review is optional

**git push origin HEAD:refs/for/master**

................................................................................................................................................

**SKATE-212**

The new SKATE-212, is a member from Kemsys Technologies. SKATE BOARD family of Single Board Computers and is powered by the Qualcomm® Snapdragon™ 212 series application processor (APQ8009). SKATE-212 development kit features quad core Arm Cortex A7 class computing with easy access to industry standard I/O’s which creates the perfect environment for a variety of Android/Linux based applications including digital signage, industrial automation and video conferencing. Unique features include on-board support for WiFi/BLE, GPS, HDMI displays, dual-MIPI-CSI cameras, MIPI-DSI, and RJ45 interfaces.

........................................................................................................................

**REPO :**

>> The repo tool is a source code configuration management tool used by the Android project. The repo tool is a front end to git written in Python. It uses a manifest file to help download code organized as a set of projects that are stored in different git repositories.

>> Create a ~/bin directory in your home directory

>> . Download the repo script. curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

>> Set the repo script attributes to executable. chmod a+x ~/bin/repo

Include the installed directory location for repo in your PATH. export PATH=~/bin:$PATH

>> Enter the following command to configure the build environment shell settings: source build/envsetup.sh

NOTE: You must use the source command so the environment settings are defined in the current shell

>> Enter the lunch command to select the build configuration, or enter it with no parameters to see an interactive menu for making selections.

>> Sometimes for certain changes which may not take effect in kernel, it may be useful to manually remove kernel obj files:

rm -rf out/target/product/msm8909/obj/KERNEL\_OBJ/

rm out/target/product/msm8909/boot.img

>> Once the device is detected, flash the binaries to the target. Run the following commands to flash all the android apps images:

EX : fastboot flash boot boot.img

>> Reboot the board

NOTE: When re-flashing Android Images, the initial boot will take up to 3 minutes as Android initializes user space databases and files. This boot time will reduce to less than a minute on subsequent boots. Or First time boot will take more time, as OS need to setup application and other internals.

>> At some time during development, if the user flashed wrong binary to any of the android partitions (like boot, system, persist, userdata), the device will not boot. At that time, the user can recover the device through recovery mode.

>> 1. To configure adb, modify the USB driver by navigating to the following directory:

cd /etc/udev/rules.d/

1. Plug the USB cable into the target.

2. Navigate to the following directory: cd /out/host/linux-x86/bin

3. Enter the command below to register a device: sudo ./adb devices

4. Push the files as follows: ./adb push AppName.apk /system/app/. or ./adb install AppName.apk NOTE: In general, the syntax is: adb push <filename> <loction on target>

.....................................................................................................................................

**To Add New module in Android Source tree:**

Android makefiles (Android.mk) are similar to regular GNU makefiles; some differences are:

>> Predefined variables to assign for source files, include paths, compiler flags, libraryλ includes, etc.

EX : LOCAL\_SRC\_FILES – List of all source files to include

LOCAL\_MODULE – Module name (used for “m”)

LOCAL\_CFLAGS – C compiler flags override

LOCAL\_SHARED\_LIBRARIES – Shared libraries to include

.......................................................................................................................................

**LK overview Android boot loader is the LK boot loader**

**LK performs:**

Hardware initialization: setting up vector table, MMU, cache, initialize peripherals, storage, USB, crypto, etc.

* Loads boot.img from storage.
* Supports flashing and recovery
* The Android build system supports generation of the signed boot image using the user’s private key.
* The build system calculates the SHA256 hash of the raw boot.img and signs the hash with the user’s private key (specified by $(PRODUCT\_PRIVATE\_KEY) flag defined in device/qcom/common/common.mk. It then concatenates this signed hash value at the end of raw boot.img to generate signed boot.img.
* Users must set the PRODUCT\_PRIVATE\_KEY flag with their private key file. Currently, it is set to device/qcom/common/qcom.key, which is a test private key and open sourced on CAF