Installation Guide

Broadcom Proprietary and Confidential

Release Note: This document can be shared with customers.

Introduction

This document describes Broadcom Android software integration. It contains information on how to build, load, and boot Android onto our reference devices given an Android TV source tree populated.

This is an "Android TV ready" release based on Google's Android operating system, intended to support any Android TV solutions based on Broadcom's BCM7251S (Banff), BCM7252S (Avko), BCM72604A0 (Elfin), BCM72604B0 (Grouse), BCM7268 (Dawson), BCM7271 (Cypress), and BCM7278 (Fundy) application processors.

Build Instructions

The build environment should be hosted on a Linux PC desktop. Familiarity with traditional Linux bash shell and command-line utilities is assumed.

Your Compile Environment

The operating system of your build machine is recommended to be Ubuntu 14.04.5.

1. Install Ubuntu 64-bits OS from your Ubuntu installation disk. Type the following commands to verify the version and bit-depth.

```
uname -a
```

Output from the command above should have "x86_64 GNU/Linux" at the end indicating the system is using 64-bit core.

```
cat /etc/issue
```

The command above will return the Ubuntu version your system has installed.

1. Update Ubuntu packages.
   - System → Administration → Update Manager → Install Updates
   - Reboot
   1. Prepare Android build environment as described here: http://source.android.com/source/initializing.html.
   - Note that Java 8 along with various packages are required.
   - **Please ensure that those additional Ubuntu tools and packages as prescribed in the link above are installed in your Ubuntu machine as our build depends on them.**

NOTE: Our build requires that gawk be installed, which should be covered in the official Android installation prerequisite above, but in case not, please install gawk explicitly:

sudo apt-get install gawk

sudo update-alternatives --config awk

Please also install mtools and libssl-dev explicitly:

sudo apt-get install mtools libssl-dev

1. Obtain the “repo” tool by typing the following commands. The “repo” tool is used for fetching the AOSP source code from Google’s server.

curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > repo

chmod a+x repo

sudo mv repo /usr/local/bin/

Compiling Target Binaries

Prepare your workspace

For illustration purpose, this document assumes that the Android source tree is located under /home/brcm/android.

NOTE 1: By default, the reference build is configured for “ZD” (i.e. development) part, and only “ZD” secure binaries get included. If your device requires a different set of binaries, such as the case with “ZB” (i.e. production) part, you need to update your specific device configuration to pull in the the correct set of secure binaries. The location to update the settings is in <device.mk> for your device.

Three environment variables to consider:

a) export SAGE_BINARY_EXT

b) export SAGE_BL_BINARY_PATH

c) export SAGE_APP_BINARY_PATH

a) SAGE_BINARY_EXT defaults to "_dev", which is the naming convention extension used for development SAGE binaries. For the corresponding production binaries, SAGE_BINARY_EXT should be set this to <empty>. The variable applies to both the SAGE bootloader and all SAGE TA's.

b) SAGE_BL_BINARY_PATH is the location of the bootloader binary. It defaults to "$(BSEAV_TOP)/lib/security/sage/bin/$(BCHP_CHIP)$(BCHP_VER)/dev", which points to the development binary folder. To use the production binary set of the same device, re-assign the variable without the trailing "/dev". This variable can also be re-assigned to any path in the system where the SAGE binaries are located.

c) SAGE_APP_BINARY_PATH is the location of the TA's binaries, defaults to SAGE_BL_BINARY_PATH. Override this variable only if the SAGE bootloader and SAGE TA’s are not located in the same directory.

For example, the overrides to production SAGE binaries for 7271 B0 look like this:

export SAGE_BINARY_EXT :=

export SAGE_BL_BINARY_PATH := vendor/broadcom/refsw/BSEAV/lib/security/sage/bin/7271B0

NOTE 2: URSR 3PIP add-on packages, such as 3PIP (Dolby) package, is extracted under vendor/broadcom/refsw.

Compile target image

Now you have fetched the source code, you have a workspace that you can compile an image for your Broadcom reference platform as follows:

```
cd /home/brcm/android

source ./build/envsetup.sh

lunch <target name>-[userdebug|user]

make -j<N> > log 2>&1
```

Where:

- <target name> could be one of the following target names, depending on what platform you have:

Please see the Build Target guide for more information.

- <N> in "make -j<N>" should be replaced by the number of concurrent jobs that your build machine can handle (e.g. make -j8).

NOTE: For build server with low (8GB or lower) memory, it is necessary to ensure the jack server virtual machine is being allocated sufficient memory upfront. Otherwise, the build would fail due to "out of memory" condition on the jack server. The example below shows how jack-server can be reconfigured with an appropriate memory setting.

```
export JACK_SERVER_VM_ARGUMENTS="-Dfile.encoding=UTF-8 -XX:+TieredCompilation -Xmx4g"

./prebuilts/sdk/tools/jack-admin kill-server

./prebuilts/sdk/tools/jack-admin start-server
```

The resulting compiled image will be located in:

./out/target/product/<target name>

Determine What Your Platform Supports

Does your platform have eMMC on-board?

A HW platform that can handle the proper way of booting Android MUST have an eMMC on-board as the primary storage, however many earlier versions of Broadcom reference designs do NOT have eMMC designed in. The following steps help you to determine if your platform is designed and equipped with what is needed to boot from eMMC.

1. If your platform came with BOLT pre-installed, the easiest way to find out whether your platform has eMMC support is by issuing the ”show device” command at BOLT’s command prompt:

```
BOLT> show devices

Device Name        Description

------------------- ---------------------------------------------------------

          uart0  16550 DUART at 0xf040a900 channel 0

           mem0  Memory

         flash0  EMMC flash Data : 0x000000000-0x370000000 (14080MB)

   flash0.macadr  EMMC flash Data : 0x000004400-0x000004600 (512B)

    flash0.nvram  EMMC flash Data : 0x000004600-0x000014600 (64KB)

      flash0.bsu  EMMC flash Data : 0x000014600-0x000054600 (256KB)

     flash0.misc  EMMC flash Data : 0x000100000-0x000200000 (1024KB)

    flash0.hwcfg  EMMC flash Data : 0x000200000-0x000300000 (1024KB)

     flash0.boot  EMMC flash Data : 0x000300000-0x002300000 (32MB)

  flash0.recovery  EMMC flash Data : 0x002300000-0x004300000 (32MB)

    flash0.cache  EMMC flash Data : 0x004300000-0x014300000 (256MB)

   flash0.system  EMMC flash Data : 0x014300000-0x062B00000 (1256MB)

  flash0.userdata  EMMC flash Data : 0x062B00000-0x1D1FFBE00 (5877MB)

         flash1  EMMC flash Boot1: 0x000000000-0x000400000 (4MB)

         flash2  EMMC flash Boot2: 0x000000000-0x000400000 (4MB)

         flash3  EMMC flash RPMB : 0x000000000-0x000020000 (128KB)

           eth0  GENET Internal Ethernet at 0xf0b00000

          mdio0  GENET MDIO at 0xf0b00800

          sata0  SATA3 AHCI Device

          sata1  SATA3 AHCI Device

        usbdev0  USB BDC at 0xf0472000

*** command status = 0
```

The example above shows that this platform has an eMMC on-board and the eMMC flash Boot1 partition is "flash1".

For platforms that do not have eMMC on board, it would look something like the following:

```
BOLT> show devices

Device Name        Description

------------------- ---------------------------------------------------------

          uart0  16550 DUART at 0xf040a900 channel 0

           mem0  Memory

     flash0.bolt  SPI flash @ CS0: 0x00000000-0x00100000 (1024KB)

   flash0.macadr  SPI flash @ CS0: 0x00100000-0x00110000 (64KB)

    flash0.nvram  SPI flash @ CS0: 0x00110000-0x00120000 (64KB)

   flash0.kernel  SPI flash @ CS0: 0x00120000-0x01F70000 (31MB)

  flash0.devtree  SPI flash @ CS0: 0x01F70000-0x01F80000 (64KB)

   flash0.splash  SPI flash @ CS0: 0x01F80000-0x02000000 (512KB)

         flash0  SPI flash @ CS0: 0x00000000-0x02000000 (32MB)

  flash1.rootfs0  NAND flash @ CS1: 0x00000000-0x100000000 (4096MB)

  flash1.rootfs1  NAND flash @ CS1: 0x100000000-0x200000000 (4096MB)

         flash1  NAND flash @ CS1: 0x00000000-0x200000000 (8192MB)

           eth0  GENET Internal Ethernet at 0xf0b00000

          mdio0  GENET MDIO at 0xf0b00800

          sata0  SATA3 AHCI Device

          sata1  SATA3 AHCI Device

    tcpfastboot0  TCP Fastboot (port 1234)

     tcpconsole0  TCP Console (port 23)

    tcpfastboot1  TCP Fastboot (port 1234)

     tcpconsole1  TCP Console (port 23)
```

NOTE: If your platform does NOT have eMMC on-board, please contact your FAE to source a reference platform that is on the "Supported Reference Platform" list in the release note.

1. Once you have confirmed that your platform has eMMC on-board, the next step is to find out whether your platform is boot-strapped to boot BOLT from eMMC. Use the “info” command and look for “Boot Device: XXX” as follows:

```
BOLT> info

========================================================================

  CPU speed: 1719MHz

DDR0 Frequency: 1067MHz, 4Gx16 phy:32   40000000 @ 00000000

DDR1 Frequency: 1067MHz, 4Gx16 phy:32   40000000 @ 80000000

 Total memory: 2048MB

  Boot Device: EMMC
```

NOTE: If your platform is boot-strapped to boot BOLT from eMMC as illustrated above (e.g. look for “Boot Device: EMMC” in the output), then you can proceed to the next section, otherwise please contact your FAE to source a reference platform that is on the “Supported Reference Platforms” list in the release notes.

Does your platform support USB device mode?

All Broadcom supported reference platforms as listed in the release notes should have USB device port designed in, which allows features such as fastboot-over-USB and adb-over-USB to work. In the cases where your platforms do not have native USB device port support, one can use Fastboot-over-TCP and adb-over-TCP instead.

In order to determine if your platform has USB device mode support, issue the "show devices" command from BOLT and look for "usbdev0" device. If your platform has USB device mode support, then the line should look like the following.

```
BOLT> show devices

Device Name        Description

-------------------  -------------------------------------------------------

          ...  ...

      usbdev0  USB BDC at 0xf0472000
```

If your platform does not return something that looks like the above example, then your platform does NOT have USB device mode support and you must use TCP for your fastboot and adb operations instead.

NOTE: Instruction in this installation guide assume that your platform have USB device mode support. If your platform does NOT have USB device mode support, please do the following instead:

- In the BOLT context, add the "-transport=tcp" option in all references to the "android fastboot" commands in this document (e.g. BOLT> android fastboot -transport=tcp) so that fastboot-over-tcp is used.
- In the BOLT context, set environment variable FB_TRANSPORT_MODE (i.e., setenv -p FB_TRANSPORT_MODE tcp).
- In the host server context, insert "-s tcp:<ip>" parameter to the fastboot flash command.

Host machine usb driver setup

Please follow this instruction to ensure that a usb driver is properly installed and setup for fastboot and adb (over usb) to work. On Ubuntu, the vendor id to use in /etc/udev/rules.d/51-android.rules is "18d1":

```
SUBSYSTEM=="usb", ATTRS{idVendor} =="18d1", MODE="0666"
```

First Boot Preparation

If your HW platform has never been loaded with a valid Android image, you MUST follow the steps below to ensure that your system's bootloader, partition table, Android BSU, and all HW specific configuration parameters are updated and properly provisioned. Once your bootloader and the rest of the boot related settings are provisioned, you will not need to repeat the steps in this section.

Update BOLT

NOTE: The following procedure assumes that your Linux host machine already has TFTP server set up. The easiest way to flash your compiled bootloader to your target platform is via TFTP from your Linux machine.

1. In ./out/target/product/<target>/ of your workspace, you should find some bootloader images called "bolt-<X>.bin", where <X> indicates the different hardware/chip variants. Depending on your target variant, you will need to pick the corresponding image when upgrading bootloader on your platform. To determine which binary you should use, check the label on your enclosure or on the PCB. It should read something like the following. If you are unsure about your variant, please consult your FAE.

BCM7252SZDKFS**BB**1G

The highlighted characters indicate which variant your platform is. If yours has “BB” on it as highlighted above, you should use the -bb binary. If yours has “BA” instead, use the -ba binary.

1. Once you have determined the right bootloader image to use, copy it to your TFTP server.
2. In BOLT's command prompt, issue the following command to flash your bootloader image to your target platform. Note that the following command assumes that your platform is connected to some network via Ethernet that can communicate with your TFTP server.

```
BOLT> ifconfig eth0 -auto

BOLT> flash <tftp-server-ip>:<path-to-bolt-xx.bin> flash1
```

1. Reboot your target platform after the bootloader is successfully upgraded.

NOTE: Each HW variant supports different box modes and only certain RTS box modes are specifically configured for Android's use cases and requirements. Historically, the RTS box mode is configured in BOLT. Starting in Android O / URSR 17.4, the RTS box mode is now set in the Linux environment, during init phrase. RTS box mode set (or default) in BOLT will not overridden and would NOT be honored. Please refer to the Build Targets guide for more information on RTS box mode supported for each target/platform.

Update Android's Partition Table in your eMMC

1. Locate your Android BSU file in out/target/product/bcm_platform/android_bsu.elf of your workspace and copy it over to your TFTP server.
2. From BOLT command prompt, issue the following command to load your compiled Android BSU file from your TFTP server.

```
BOLT> ifconfig eth0 -auto

BOLT> boot -elf -noclose -bsu <tftp-server-ip>:<android_bsu.elf>
```

1. Once your BSU is successfully loaded, it should return to the BOLT command prompt. Note the IP address assigned to your platform.
2. Verify that BSU is loaded and that the 'android boot' command is available as shown below:

```
BOLT> help android boot
 SUMMARY
   Load an Android boot image into memory and boot it
 USAGE
   android boot [options] [arg]

   This command loads and boots and Android image. The
   optional arg can be used to override any builtin
   kernel command line arguments.

   The file or partition to use for booting can be
   specified with environment variable or explicitly
   in the command with -i option.
   Expected format:
     dev (e.g. flash1.partition)

   The following environment variables can be used:
     ANDROID_BOOT_IMG - for normal boot image
     ANDROID_RECOVERY_IMG - for recovery boot image
 OPTIONS
   -rawfs ........... Load the file from an unformatted file system
*** command status = 0
```

1. Once you have confirmed that your Android BSU is properly loaded, the next step is to use "android fastboot" to update the partition table of your platform. BUT before you proceed to updating the eMMC partition table, you must be aware of the following:
   - The partition table as defined in this Android release assumes the onboard eMMC is 8GB. **Specifically, the partition table size defined for makegpt (see LOCAL_DEVICE_GPT in the device makefile) MUST be equal or smaller than the size as shown in flash0 (via "show devices") in bolt.** If your platform has an eMMC flash size smaller than 8GB, then you have to tailor the partition table image (i.e. gpt.bin) accordingly with the "makegpt" tool that comes with the AOSP source tree. This document does not cover how to change the partition table; please refer to the help menu from the "makegpt" tool for instructions. If your platform has more than 8GB, then you can still use the default gpt.bin although only 8GB would be provisioned.
   1. Put your target platform into fastboot mode:

```
BOLT> android fastboot -device=flash0
```

1. Use fastboot to flash "gpt.bin" to your target platform. In your workspace, run the following command from your Linux build machine:

```
./out/host/linux-x86/bin/fastboot flash gpt ./out/target/product/<target>/gpt.bin
```

NOTE 1: If you have a platform that has never been used for Android before, it is possible that there is not a valid GPT pre-loaded to your eMMC and you would see a log message complaining that "GPT Header signature is wrong". This is harmless as long as you only update the 'gpt' partition as per the instructions in this section.

NOTE 2: If you have Android platform tool setup on your host machine, you can use the standard fastboot tool instead of the version generated from build.

1. Use fastboot to reboot your target platform from the host machine:

```
./out/host/linux-x86/bin/fastboot reboot
```

1. Your partition table should have been updated by now. Run the "show devices" command in BOLT to confirm the newly created partitions show up properly as illustrated below:

```
BOLT> show devices

Device Name        Description

------------------- -------------------------------------------------------

          uart0  16550 DUART at 0xf040a900 channel 0

           mem0  Memory

         flash0  EMMC flash Data : 0x000000000-0x370000000 (14080MB)

    flash0.macadr  EMMC flash Data : 0x000004400-0x000004600 (512B)

     flash0.nvram  EMMC flash Data : 0x000004600-0x000014600 (64KB)

       flash0.bsu  EMMC flash Data : 0x000014600-0x000054600 (256KB)

      flash0.misc  EMMC flash Data : 0x000100000-0x000200000 (1024KB)

     flash0.hwcfg  EMMC flash Data : 0x000200000-0x000300000 (1024KB)

      flash0.boot  EMMC flash Data : 0x000300000-0x002300000 (32MB)

  flash0.recovery  EMMC flash Data : 0x002300000-0x004300000 (32MB)

     flash0.cache  EMMC flash Data : 0x004300000-0x014300000 (256MB)

  flash0.metadata  EMMC flash Data : 0x014300000-0x016300000 (32MB)

    flash0.system  EMMC flash Data : 0x016300000-0x062B00000 (1224MB)

  flash0.userdata  EMMC flash Data : 0x062B00000-0x1D1FFBE00 (5877MB)

         flash1  EMMC flash Boot1: 0x000000000-0x000400000 (4MB)

         flash2  EMMC flash Boot2: 0x000000000-0x000400000 (4MB)

         flash3  EMMC flash RPMB : 0x000000000-0x000020000 (128KB)

           eth0  GENET Internal Ethernet at 0xf0b00000

          mdio0  GENET MDIO at 0xf0b00800

          sata0  SATA3 AHCI Device

        usbdev0  USB BDC at 0xf0472000

*** command status = 0
```

1. Set your environment variables:

```
BOLT> setenv -p FB_DEVICE_TYPE flash0
BOLT> setenv -p ANDROID_BOOT_IMG flash0.boot
```

1. Lastly, check if your BOLT environment variables still carry a valid MAC address. If you don't see a MAC address listed from the "printenv" command in BOLT, then program your board with the MAC address you recorded in earlier step or get BOLT to generate one for you automatically based on your board serial number as illustrated below:

```
BOLT> macprog <board-type> <serial-num> <board-rev>
```

If you are not sure how to use "macprog" from BOLT, type the following command and it will give you an example:

```
BOLT> help macprog
```

NOTE: You MUST assign a valid & unique MAC address to your platform. Do not copy the MAC address from one STB to another. Also, the first 3 bytes of the MAC address is the OUI (Organizationally Unique Identifier) field should “00:10:18”, indicating the vendor is Broadcom (e.g. 00:10:18:xx:yy:zz).

Flash Android BSU

With the partition table properly created in your eMMC, you can now flash your Android BSU to the "bsu" partition from your BOLT command prompt as follows. This allows you to boot from the bsu partition instead of loading the Android BSU from your TFTP server everytime after reboot.

```
BOLT> ifconfig eth0 -auto
BOLT> flash <tftp-server-ip>:<path-to-android_bsu.elf> flash0.bsu
```

Hardware Configuration (hwcfg) Partition

The hwcfg partition in your eMMC device is specifically used for storing hardware specific content such as DRM binaries (e.g. drm.bin and drm_hdcp1x.bin) and Wi-Fi configuration file (e.g. nvm.txt) that are unique to each platform. This section describes all the steps for preparing and generating the hwcfg.img image that is unique to your HW platform. For devices with bp3-enabled part, the hwcfg is expected to be vfat formatted. For more information on converting a given hwcfg to vfat format, please refer to document "Android and BP3|PAK".

Create hwcfg.img

1. First, you need to provide the following files and information (contact your FAE if you have questions about how to generate your own drm.bin files):
   - drm.bin
   - drm_hdcp1x.bin
   - An unique WiFi MAC address assigned to your platform

NOTE 1: WiFi will not work without this step.

NOTE 2: If your platform does not have the required DRM binaries files (drm.bin and drm_hdcp1x.bin) in the hwcfg partition, all DRM protected content would NOT be played.

NOTE 3: If you have not been given your platform specific DRM binaries or assigned an unique WiFi MAC address, you are still required to flash the hwcfg.img that gets created as part of the standard build procedure (even though the hwcfw.img file is empty or does not contain all the needed content), otherwise Android would not be able to mount the hwcfg partition and Android would abort the mount process and some partitions would not be mounted.

NOTE 4: On cypress*/b71t devices, which comes with on-chip WiFi, the WiFi MAC address is provisioned from the on-board Ethernet MAC address +1. However, the nvm.txt file provisioned here is still used.

1. Create a "hwcfg" directory at the top of your workspace, and put your platform specific drm.bin and drm_hdcp1x.bin into it. Please make sure both DRM binaries have the necessary read permissions; chmod 744 should be sufficient.

```
mkdir /home/brcm/android/hwcfg
```

```
cp drm.bin /home/brcm/android/hwcfg/drm.bin
```

```
cp drm_hdcp1x.bin /home/brcm/android/hwcfg/drm_hdcp1x.bin
```

NOTE: If you do not have your platform specific drm.bin and/or drm_hdcp1x.bin yet, or do not wish to use one, you can omit these files from the hwcfg directory and continue to the next step, but you won't be able to play any DRM contents as noted earlier.

1. Create wifimac.txt inside the hwcfg directory and put your unique WiFi mac address into it.

```
echo 00:11:22:aa:bb:cc > /home/brcm/hwcfg/wifimac.txt
```

NOTE: You could omit wifimac.txt, but your device will then use a default WiFi MAC address which may conflict with other platforms that happen to use the same default WiFi MAC address and will experience connectivity issues on the WiFi AP due to collisions.

1. From the top of your workspace, run the following command. Note that this command is run automatically as part of a full build.

```
make makehwcfg
```

A file called "hwcfg.img" will be created in the out/target/product/<target>/ directory of your workspace.

Flash hwcfg.img

1. Load Android BSU from your previously flashed location and put your device in fastboot mode from BOLT command prompt:

```
BOLT> boot -elf -noclose -bsu flash0.bsu
```

```
BOLT> android fastboot -device=flash0
```

Once your BSU is successfully loaded, it should return to the BOLT command prompt.

1. Back to your Linux host build machine, use the fastboot tool in out/host/linux-x86/bin/ of your workspace to flash the hwcfg.img that was just created. For example:

```
sudo ./out/host/linux-x86/bin/fastboot flash hwcfg ./out/target/product/<target>/hwcfg.img
```

Image Flashing

This section assumes that your HW platform has already been seeded with a bootloader, Android BSU, proper partition table and all platform specific parameters as instructed in Section 4: First Boot Preparation.

The following instructions outlines the steps you should use to flash your compiled images to the rest of your system partitions.

1. Load Android BSU from your previously flashed location:

```
BOLT> boot -elf -noclose -bsu flash0.bsu
```

Once your BSU is successfully loaded, it should return to the BOLT command prompt.

1. Put your target platform in fastboot mode:

```
BOLT> android fastboot -device=flash0
```

1. From your host machine, "fastboot flash" all the images as illustrated below:

```
<path-to-fastboot> flash bootloader ./out/target/product/<target>/bootloader.img

<path-to-fastboot> flash boot ./out/target/product/<target>/boot.img

<path-to-fastboot> flash system ./out/target/product/<target>/system.img

<path-to-fastboot> flash vendor ./out/target/product/<target>/vendor.img

<path-to-fastboot> flash userdata ./out/target/product/<target>/userdata.img

<path-to-fastboot> flash cache ./out/target/product/<target>/cache.img

<path-to-fastboot> flash recovery ./out/target/product/<target>/recovery.img [non-A/B device only]
```

NOTE 1:

NOTE 2:

If the target platform is one of 972604, 97268 or 97271, it supports the A/B system update method by default instead of the traditional ota update method. On those platforms, there would not be a recovery partition (as recovery is built as a part of boot.img). The primary boot and system partitions would be named as boot_i and system_i respectively. Please refer to Section 10 for more information regarding A/B system update.

bootloader.img contains both bolt and android_bsu.elf, and both would be updated after flash. Additionally, android_bsu.elf has gpt.bin embedded, and the current gpt would be updated (if different) as android_bsu is loaded.

1. Once you have flashed all the images, reboot the target from your host machine:

```
<path-to-fastboot> reboot
```

Booting up your Image

Now you have everything in place and ready to boot up Android on your device as follows:

1. From BOLT command prompt, issue the following command to load Android BSU that was previously flashed to your system:

```
BOLT> boot -elf -noclose -bsu flash0.bsu
```

Once your BSU is successfully loaded, it should return to the BOLT command prompt.

NOTE: Unlike BOLT, the Android BSU needs to be reloaded each time you reboot the device.

1. Verify that BSU is loaded and that the 'android boot' command is available (optional)

```
BOLT> help android boot
 SUMMARY
   Load an Android boot image into memory and boot it
 USAGE
   android boot [options] [arg]

   This command loads and boots and Android image. The
   optional arg can be used to override any builtin
   kernel command line arguments.

   The file or partition to use for booting can be
   specified with environment variable or explicitly
   in the command with -i option.
   Expected format:
     dev (e.g. flash1.partition)

   The following environment variables can be used:
     ANDROID_BOOT_IMG - for normal boot image
     ANDROID_RECOVERY_IMG - for recovery boot image
 OPTIONS
   -rawfs ........... Load the file from an unformatted file system
*** command status = 0
```

1. Boot up Android!!!

```
BOLT> android boot -rawfs
```

NOTE 1: By default, the system boots up with SELinux=enforcing.

NOTE 2: All the boot parameters are implicitly passed to the kernel with this booting method. To understand what boot parameters are defined for each supported platform, please refer to the corresponding <target>.mk found in device/broadcom/dboards/<target> in your workspace.

Boot using Function Key

You may assign the boot sequence to a function key such as "F2" so that you can load the Android BSU and boot up Android by simply pressing the F2 function key on your keyboard from the BOLT command prompt. For example:

```
BOLT> setenv -p F2 'boot -elf -noclose -bsu flash0.bsu; android boot -rawfs'
```

Automatic Boot Up

BOLT support automatic boot up by assigning the boot up sequence to the "STARTUP" environment variable. For example:

```
BOLT> setenv -p STARTUP 'boot -elf -noclose -bsu flash0.bsu; android boot -rawfs'
```

This will make BOLT to execute the commands assigned to "STARTUP" on boot every time.

NOTE 1: To break out from the automatic startup sequence, press "Ctrl-c" right as BOLT comes up.

NOTE 2: To disable the automatic startup feature, use the "unsetenv" command in BOLT command prompt to reset the STARTUP variable (i.e. unsetenv STARTUP).

ADB Connection

For "userdebug" variant images, we enabled both ADB over USB (via the USB device port) and ADB over TCP simultaneously by default. ADB over TCP is default to listen on port 4321.

For "user" variant images, the ADB connection is handled over USB via the USB device port of your platform (if your platform supports one).

NOTE: ADB debug report on user images **turned off** by default. To enable ADB debug, please go to "Development Mode" by selecting "About" 5 times in Settings.

To enable (for "user" images) or switch the ADB port used for ADB over TCP:

```
adb tcpip <port>
```

Recovery Mode

1. In order to get into the recovery mode, you must ensure that the recovery.img is flashed to the recovery partition (for devices on a/b update, that is built-in as part of the boot image). If you had followed the instructions in this guide, your recovery partition should have already been flashed with a proper recovery.img binary, if not, please refer to "Section 4: First Boot Preparation".
2. Set the BOLT environment variable 'ANDROID_RECOVERY_IMG' to point to the recovery partition.

```
BOLT> setenv -p ANDROID_RECOVERY_IMG flash0.recovery
```

1. Now you can initiate the recovery process from Android shell as follows:

```
reboot recovery
```

The following illustrates what output messages you would expect to see from the console port when recovery is initiated from console.

```
shell@b52ssffdr4:/ $ reboot recovery

[  618.372893] SysRq : Emergency Remount R/O

[  618.406359] EXT4-fs (mmcblk0p11): re-mounted. Opts: (null)

[  618.435243] EXT4-fs (mmcblk0p8): re-mounted. Opts: (null)

[  618.440842] Emergency Remount complete

[  618.679877] dhd_prot_ioctl : bus is down. we have nothing to do

[  618.685838] CFGP2P-ERROR) wl_cfgp2p_bss_isup : 'cfg bss -C 0' failed: -1

[  618.692594] CFGP2P-ERROR) wl_cfgp2p_bss_isup : NOTE: this ioctl error is normal when the BSS has not been created yet.

[  618.703345] dhd_prot_ioctl : bus is down. we have nothing to do
[  618.709292] CFG80211-ERROR) wl_cfg80211_clear_security : wsec clear failed

[  618.716298] dhd_prot_ioctl : bus is down. we have nothing to do

[  618.722248] CFG80211-ERROR) wl_cfg80211_clear_security : auth clear failed

[  618.729260] dhd_prot_ioctl : bus is down. we have nothing to do

[  618.735213] CFG80211-ERROR) wl_cfg80211_clear_security : wpa_auth clear failed

[  618.742589] CFG80211-ERROR) wl_notifier_change_state : wl_notifier_change_state : busstate is
DHD_BUS_DOWN!

[  618.752420] CFG80211-ERROR) wl_event_handler : was terminated

[  618.758356] CFGP2P-ERROR) wl_cfgp2p_disable_discovery :  do nothing, not initialized

[  618.766203] CFGP2P-ERROR) wl_cfgp2p_deinit_priv : In

[  618.771243] dhd_prot_ioctl : bus is down. we have nothing to do

[  618.777230] CFG80211-ERROR) wl_dongle_down : WLC_DOWN error (-1)

[  618.789399] CFG80211-ERROR) wl_notifier_change_state : wl_notifier_change_state : busstate is
DHD_BUS_DOWN!

[  621.911944] reboot: Restarting system with command 'recovery'

[  621.917761] brcmstb_reboot: cmd='recovery', val=114
```

[...]

```
BOLT> boot -elf -noclose -bsu flash0.bsu; android boot -rawfs

Loader:elf Filesys:raw Dev:flash0.bsu File:(null) Options:(null)

Starting program at 0x1800000 (DTB @ 0x7618000)

Adding Android commands to BOLT

Done loading Android BSU

Autogen PRODUCTNAME = b52ssffdr4

Adding Android img loader to BOLT

boot_reason = 114

boot reason = recovery

Setting WKTMR to 1420070400 sec (UTC)
```

boot_cmd=boot -loader=img -rawfs flash0.recovery

```
Loader:img Filesys:raw Dev:flash0.recovery File:(null) Options:(null)

    magic: ANDROID!

 kernel_size: 5818208

 kernel_addr: 0x10000

 ramdisk_size: 18247642

 ramdisk_addr: 0x2008000

 second_size: 0

 second_addr: 0xf08000

  tags_addr: 0x8100

  page_size: 4096

     name: ''

   cmdline: 'mem=1024m@0m mem=1024m@2048m bmem=336m@672m bmem=240m@2048m
brcm_cma=784m@2288m ramoops.mem_address=0x3F800000 ramoops.mem_size=0x800000
ramoops.console_size=0x400000 pmem=8m@1016m'

      id: 153088176

kernel_offset: 0x1000, 1421 pages

ramdisk_offset: 0x58e000, 4455 pages

Loading ramdisk image to address 0x02008000

Adding initrd info to DT

DTB @ 0x7618000

DTB @ 0x7618000

Adding kernel command line to DT

DTB @ 0x7618000

Seeking to start of the kernel image

Reading 5818208 bytes from zImage.

...........
```

```
Starting program at 0x8000 (DTB @ 0x7618000)

32 bit boot...

Uncompressing Linux... done, booting the kernel.

[    0.000000] Booting Linux on physical CPU 0x0

[    0.000000] Initializing cgroup subsys cpu

[    0.000000] Initializing cgroup subsys cpuacct
```

[...]

You can also boot in recovery mode by holding one of the front-panel buttons (BT pair, or WPS) during boot. Once you see the Android recovery screen, you may control the on-screen menu either from keyboard or using the front-panel buttons.

OTA Mode

Building OTA Package

In order to use OTA, you need to first verify that you can boot into recovery move using the "reboot recovery" command from the console (see Section 8: Recovery Mode).

From the top of your workspace, the OTA package can be built with first generating the target files:

```
source ./build/envsetup.sh

lunch <your_specific_lunch_combo>

make clean # to make sure all timestamps are updated

make otatools

mkdir <dist_output>

make dist -j<N> DIST_DIR=<dist_output>
```

After the build completes, a zipped up copy of the target files should be generated at the <dist_output>/<target>-target-files.zip. You can now generate a full OTA via the release tool script:

```
./build/tools/releasetools/ota_from_target_files <dist_output>/<target>-target-files.zip ota_full.zip
```

An incremental OTA package can be built similarly with two target files (i.e. the delta) with the same release tool script, for example:

```
./build/tools/releasetools/ota_from_target_files -i ORG-target-files.zip <dist_output>/<target>-target-files.zip
ota_incremental.zip
```

NOTE 1: Android system build timestamp will not change, unless it's built again from a clean workspace. So it's better to use "make clean" before you build the OTA package for validation purpose.

NOTE 2: In the current system configuration, cache partition is not large enough to hold the OTA zip package for a GMS-incorporated image, so it's better to use an AOSP OTA package to validate the OTA process if you intend to use the cache partition for OTA validation.

Before Applying the OTA Package

Before uploading and applying the OTA package to your platform, you should make note of the following system properties. These information would allow you to confirm at a later time whether the OTA process was successful or not:

- Kernel Build time (for Android system kernel and Android recovery kernel)
- Android System Build time

The following commands can be used to obtain the kernel build time and android system build time:

```
$ adb shell getprop | grep build.date

[ro.build.date.utc]: [1434711100]

[ro.build.date]: [Fri Jun 19 03:51:40 PDT 2015]
```

```
$ adb shell cat /proc/version

Linux version 3.14.28-1.6pre-02390-g276cb88 (xyz@server) (gcc version 4.8.4 20141110 (prerelease)
(Broadcom stbgcc-4.8-1.2) ) #1 SMP Fri Jun 19 04:02:57 PDT 2015
```

Applying the OTA Package

There are a few options to apply this generated OTA package. This section captures the different options you have.

OTA update via cache partition

Push the generated OTA zip package file to the cache partition, and trigger the update through console as follows.

NOTE 1: Only use this option if your OTA package size fits into the cache partition.

NOTE 2: This option requires the “userdebug” image as console access is required.

On the host computer:

```
adb push b52ssffdr4-ota-eng.svcricjenkins.zip /cache/update.zip
```

On the device:

```
130|shell@b52ssffdr4:/ # mkdir -pv /cache/recovery

130|shell@b52ssffdr4:/ # echo --update_package=/cache/update.zip > /cache/recovery/command

130|shell@b52ssffdr4:/ # reboot recovery
```

The "reboot recovery" command will trigger a reboot and the system will get into recovery mode and apply the uploaded OTA package file.

OTA update via “/sdcard” device

If your cache partition is NOT large enough to hold the OTA package (which is likely the case for GMS-incorporated OTA package), you can use “/sdcard” instead of the cache partition.

Do not confuse the recovery "/sdcard" with the emulated "/sdcard" we see during normal operation: the recovery fstab mounts either an external SD card or a USB disk to "/sdcard". This is determined by the recovery.fstab file which is only mounted in recovery mode. For example, our current defaults are using the USB disk (from device/broadcom/common/recovery/fstab.default/recovery.fstab):

```
/dev/block/sda1      /sdcard vfat defaults defaults

#/dev/block/mmcblk0p1  /sdcard vfat defaults defaults
```

If you are unsure, you can push the file to both physical sdcard and the usb emulated sdcard, so that recovery system would be able to access the update.zip file

On the host:

```
adb push BCM97252SSFFDR4-ota-eng.svcricjenkins.zip /mnt/media_rw/usb0/update.zip

adb push BCM97252SSFFDR4-ota-eng.svcricjenkins.zip /mnt/media_rw/sdcard1/update.zip
```

On the device (again: use "/sdcard" regardless if you are using a USB drive)**:**

NOTE: This option requires the “userdebug” image as console access is required.

```
130|shell@b52ssffdr4:/ # mkdir -pv /cache/recovery

130|shell@b52ssffdr4:/ # echo --update_package=/sdcard/update.zip > /cache/recovery/command

130|shell@b52ssffdr4:/ # reboot recovery
```

On a “user” image, user can use the recovery menu to apply the ota from removable storage.

OTA update via the BcmOtaUpdate Application

You can also use the built-in BcmOtaUpdate application to download the generated OTA zip package to the cache partition.

NOTE: Only use this option if your OTA package size fits into the cache partition.

1. In order to use the application, you need to set up an HTTP Server to host the update file.
2. Navigate to "Settings" -> "Apps" -> "System apps"
3. Look for "OtaUpdater" apk, and open it.
4. In the application, input the URL to the http server and update OTA package zip file (e.g. http://<http_server_address>/BCM97252SSFFDR4-ota-eng.svcricjenkins.zip) and click on Download button.
5. The application will download the package, and when finished, it will prompt for installation confirmation.

OTA update via ADB

This method is also known as sideload, as the package is updated using the “adb sideload” command. Here are the steps to initiate the sideload:

NOTE: Only use this option if your OTA package size fits into the cache partition.

1. Reboot to the Android Recovery menu (see Section 8: Recovery Mode)
2. Select the "Apply update from ADB" menu entry
3. Now send the package you want to apply to the device with "adb sideload <filename>"

OTA Validation

Once the system is in recovery mode, the recovery kernel build time should be the same as the system kernel; the easiest way to confirm that is by inspecting the kernel boot log as the system boots up.

```
[  912.472528] reboot: Restarting system with command 'recovery'

[  912.478301] brcmstb_reboot: cmd='recovery', val=114

CPU 01

BCM74450031

PRID72520031

....

Reading 5667624 bytes from zImage...........

Closing network 'eth0'

Starting program at 0x8000 (DTB @ 0x7822000)

Uncompressing Linux... done, booting the kernel.

[    0.000000] Booting Linux on physical CPU 0x0

[    0.000000] Initializing cgroup subsys cpu

[    0.000000] Initializing cgroup subsys cpuacct

[    0.000000] Linux version 3.14.28-1.6pre-02390-g276cb88 (xyz@server) (gcc version 4.8.4 20141110 (prerelease) (Broadcom stbgcc-4.8-1.2) ) #1 SMP Fri Jun 19 04:02:57 PDT 2015
```

After the OTA update.zip package is successfully applied, check the build timestamp again, make sure that they are indeed changed to a newer build time, using the same instructions as described in 10.2 Before Applying the OTA Package.

A/B System Update

A/B system update is a mechanism to allow ota upgrade while keeping a working boot on eMMC. For more information of the general mechanism of A/B System Update, please refer to Google's Android info portal. For Broadcom's integration note on this topic, please refer to document "A/B Update" for more information.

Currently, A/B system update is only supported on BCM7268, BCM7271, BCM72604 and BCM7278.

In essence, the recovery support is now a part of the boot image (so there is no more recovery partition), and the cache partition is no longer used for storing the OTA files. A significantly reduce sized cache partition still exists for other reasons. The freed up space is then split for storing the primary and secondary boot and system images while the data partition stayed intact.

For Broadcom's A/B system update integration, the primary partition is suffixed with _i (i.e., boot_i, system_i), and the secondary partition is suffixed with _e (i.e., boot_e, system_e). Our bootloader is smart enough to pick the current partition slot, and therefore it is optional specifying the _i / _e suffixes on fastboot flash.

It is expected that the following environmental variables be set in the bootloader (bolt):

- setenv -p FB_DEVICE_TYPE flash0
- unsetenv ANDROID_BOOT_IMG **(optional)**
- unsetenv ANDROID_RECOVERY_IMG **(optional)**

Example fastboot command for flashing to the primary boot partition:

- fastboot -u flash boot_i boot.img

Example fastboot command for flashing to the current system partition slot:

- fastboot -u flash system system.img

NOTE: If Android did not boot successfully, our updater would have marked the current slot as 'failed', and would have moved onto the secondary slot. Normally, one of the partitions would have included a working image, and would have no problem with the switch. However, for development purpose, the secondary slot may not have actually included a valid image. In this case, the boot would fail. To recover, set the device in fastboot mode again, and force the current active slot to the primary slot again:

fastboot set_active a

Licensing Notes