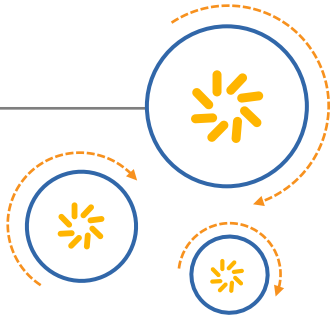
Qualcomm Technologies, Inc.



DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor

Little Kernel Boot Loader Overview

July 2016

Questions or comments: https://www.96boards.org/DragonBoard410c/forum

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

LM80-P0436-1 Rev D

Revision history

Revision	Date	Description
D	July 13, 2016	Updated to “E” part.
C	June 11, 2015	Miscellaneous update.
B	May 26, 2015	Updated Revision history and © date.
A	April 21, 2015	Initial release.

Contents

1 Introduction

1.1 Purpose

This document is intended for engineers using DragonBoard 410c and supporting Little Kernel (LK). Little Kernel is the boot loader that performs the basic tasks of hardware initialization, reading the Linux kernel and ramdisk from storage and loading it up to RAM, setting up initial registers and command line arguments for the Linux kernel, and jumps to the kernel. LK is based on the open source project on www.kernel.org.

1.2 Scope

Engineers should have a basic understanding of device trees. A device tree is data structure for describing hardware. It has a tree of nodes, and each node can contain properties and other nodes.

The scope is limited to the Android platform.

1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

1.4 Acronyms

Acronym	Definition
APQ	Application Processor Qualcomm
ARM	Advanced RISC Machines
CAF	Code Aurora Foundation codeaurora.org
CDP	Code Development Platform
CPU	Central Processing Unit
DTS	Digital Test Sequence
eMMC	Embedded Multimedia Card
HLOS	High Level Operating System
ID	Identification
LK	Little Kernel
MMC	Multimedia Card
MMU	Memory Management Unit
MSM	Mobile Station Modem
MTP	Modem Test Platform

Acronym	Definition
PMIC	Power Management Integrated Circuit
RAM	Random Access Memory
SBC	Single Board Computer
SD	Secure Digital
SDC	Secure Digital Controller
SDHCI	Secure Digital Host Controller Interface
SMEM	System Memory
SOC	System on a Chip
SPMI	System Power Management Interface
USB	Universal Serial Bus

1.5 Additional information

For additional information, go to https://www.96boards.org/DragonBoard410c/docs.

2 Android Boot Loader (Little Kernel)

2.1 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.

NOTE: LK runs in 32-bit mode even on a 64-bit architecture. The jump from LK 32-bit to 64-bit kernel goes through secure mode.

2.2 Code download/compilation

- Downloading code
  - Cloning the LK tip: git clone git://codeaurora.org/kernel/lk.git
  - Updating the tip: git pull origin, or git fetch origin
  - Checking out a specific branch: git checkout –b '<the branch name we want to give>' <commit id from caf>

    ```
    :  git checkout -b mylk remotes/origin/master
    ```
- Compiling the LK
  - The AndroidBoot.mk file has the following instruction:

    ```
    export PATH=$PATH:<Path to arm-eabi-*> binaries
    export TOOLCHAIN_PREFIX=arm-eabi-
    make msm8916 EMMC_BOOT=1
    ```

    creates a build-msm8916 directory inside lk

    Where:

    ```
    <target name> – Found inside /lk/target
    ```

NOTE: The compiler path in any Android build is /prebuilt/gcc/linux-x86/arm/arm-toolchain/arm-eabi-4.7/bin/arm-eabi-.

- build-<target> contains: emmc_appsboot.mbn (image file) and LK, which contains the symbols.
- make aboot (if the whole build is synced).

2.3 Kernel authentication

- Generation of signed boot.img.
  a. The Android build system supports generation of the signed boot image using the user's private key.
  b. 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.
  c. 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.
  d. On the Android Lollipop release, Verified boot (Google's defined mechanism) is used for the authentication of kernel and recovery images.
- LK (Android boot) authenticates the kernel (boot.img).
  a. If the TARGET_BOOTIMG_SIGNED= true flag is set in the target's BoardConfig.mk file, LK verifies boot.img before booting up into Linux kernel.
  b. During bootup, LK strips out the raw boot.img and signed hash attached at the end of the image. LK calculates the SHA256 hash of the complete raw boot.img and compares it with the hash provided in the boot.img. If both hashes match, kernel image is verified successfully.

     Call flow in LK for verifying signed kernel image – verify_signed_bootimg()→ image_verify().
  c. On successful verification of the kernel image, LK passes "androidboot.authorized_kernel=true" to the kernel in the kernel command line.
  d. Users must add their own certificate with public key in the bootable/bootloader/lk/platform/msm_shared/certificate.cfile. LK uses this certificate for decryption of the signed hash value present in the end of the boot.img.

2.4 Device tree identification

2.4.1 Device trees

- Device tree is a data structure for describing the hardware.
- Device tree source (dts):
  a. A simple tree structure of nodes and properties.
  b. Properties are key-value pairs and node may contain both properties and child nodes.
  c. Format of the .dts file is C-like, supports C and C++ style comments.
  d. For ARM architecture, the device tree source can be found in the kernel/arch/arm/boot/dts/qcom folder.

- Device tree blob (dtb):
  a. Device tree compiler (.dtc) compiles the .dts into a binary object (.dtb) understandable by the Linux kernel.
  b. This blob is appended to the kernel image as shown below during compilation.

Boot img header
Kernel
RAM disk
Device tree table

- Device tree table header

```
struct dt_table{
   uint32_t magic;
   uint32_t version;
   uint32_t num_entries;
   };
```

- Device tree entry

```
struct dt_entry{
   uint32_t platform_id; → Platform ID/Chipset ID
   uint32_t variant_id; → Hardware variants (MTP, CDP, etc.)
   uint32_t board_hw_subtype; → Distinguishes between subtypes like
   pmicvariants, fusion/standalone etc.
   uint32_t soc_rev;→ SOC revision
   uint32_t pmic_rev[4];→ PMIC revision
   uint32_t offset;
   uint32_t size;
   };
```

- Each DTS per device will add a qcom,msm-id / qcom,board-id / qcom,pmic-id entry.
  a. qcom,msm-id entry specifies the MSM™ chipset, hardware revision, and optional manufactured foundry.
  b. qcom,board-id entry specifies the hardware variant and subtype revision.
  c. qcom,pmic-id entry specifies the PMIC chips used on a given MSM platform.
- LK uses this information at boot-up to decide which device tree to use and passes this device tree to the kernel.
  - qcom,msm-id = <x z>;
  - qcom,board-id = <y y'>;
  - qcom,pmic-id = <pmic1 pmic2 pmic3 pmic4>;
- x = Platform ID (chipset ID and optional foundry ID).
  - Bits 0-15 = Unique MSM chipset ID

- Bits 16-23 = Optional foundry ID. If the boot loader does not find a device tree that exactly matches the foundry-id with the hardware, it chooses the device tree with foundry-id = 0.
- Bits 24-31 = Reserved

- z = ID for SoCrevision (hardware revision).
- y = ID for SBC (hardware variants), etc.
- y' = ID for platform subtype (assumed zero if absent).
  - "pmic#" cell is a 32-bit integer, which is defined as follows:
    - bits 31-24 = unused
    - bits 23-16 = PMIC major version
    - bits 15-8 = PMIC minor version
    - bits 7-0 = PMIC model number
- The entry can optionally be an array:
  - qcom,msm-id = <x1 z1>, <x2 z2>, ...;
  - qcom,board-id = <y1 y1'>, ...;
  - qcom,pmic-id = <pmic1 pmic2 pmic3 pmic4>, <pmic11 pmic21 pmic31 pmic41>, …… ;
- For example, for DragonBoard 410c (referred to as APQ8016 SBC in the code), the following common property is added to kernel/arch/arm/boot/dts/qcom/apq8016-sbc.dts:

```
qcom,msm-id = <206 0>, // platform id, soc_rev
<247 0>;
qcom,board-id = <24 0>; // platform hardware, platform subtypes
```

2.4.2 Identifying the right device tree

- LK scans through the device tree table to look for a matching entry. The search order is:
  a. Exact match − All of these MUST match.
     - Platform id − Contains msm ID and foundry ID.
     - MSM ID
     - Foundry ID − Look for exact match; if not found choose device tree with foundry-id(0x0).
       - Platform subtype − Subtype for the board
       - Platform type, hardware ID
       - PMIC model
- Best match IDs − The priority for lookup starting from the highest priority. The highest information is equal to or lower than the runtime detected SoCrev (read from SMEM).
  a. SoCversion
  b. PMIC0 major+minor
  c. PMIC1 major+minor

d. PMIC2 major+minor

e. PMIC3 major+minor

2.4.3 Updating the device tree

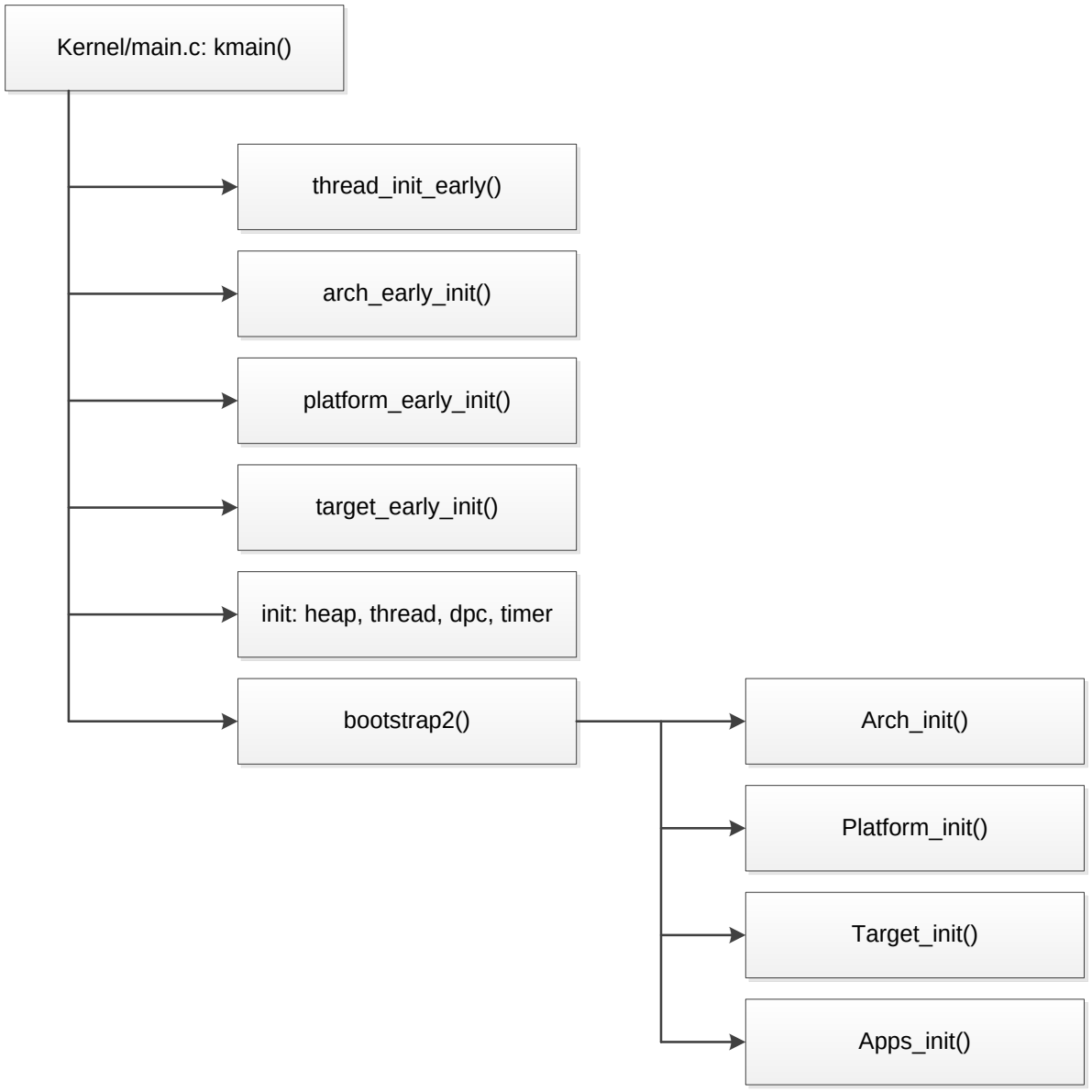
- LK populates the memory node with the memory regions' start address and size.
- LK also modifies the chosen node to add the boot arguments and RAM disk properties.
  a. The device tree flag (DEVICE_TREE) is set in /project/$(PROJECT).mk.
  b. The device tree is defined in /arch/arm/boot/dts/qcom/apq8016-sbc.dtsi.

  The skeleton device tree is defined in /arch/arm/boot/dts/skeleton.dtsi:

```
{ #address-cells = <1>;
#size-cells = <1>;
chosen { };
aliases { };
memory { device_type = "memory"; reg = <0 0>; };
};
```

2.5 LK call flow

- The sequence starts with arch/arm/crt0.S: _start.
  a. Set up CPU.
  b. Call __cpu_early_init() if necessary (platform-specific initialization sequence) platform/msm8916/(arch_init.S): __cpu_early_init.
  c. Relocate if necessary.
  d. Set up stack.
  e. Call kmain().
- The function calls from kmain() are shown below:



- Calls made from bootstrap2():
  a. arch/arm/arch.c –arch_init()
     Stub
  b. platform/<platform>/(platform.c) –platform_init()
     Stub
  c. target/<target>/(init.c) –target_init()
     Init SPMI
     Init keypad
     Set drive strength and pull configs for SDC pins (we have transitioned to SDHCI)
     Init the SD host controller/MMC card; identify the MMC card; set the clock, etc.

mmc_init()

Read the partition table from the eMMC card

partition_read_table()

d. app/init.c –apps_init()

Init apps that are defined using APP_START and APP_END macros; aboot_init()is called

Run the app in a separate thread if it has .entry section

e. app/aboot/aboot.c –aboot_init()

Performs any one of the following operations based on settings/circumstances:

- Regular boot
- Fastboot mode to accept images
- Recovery mode to jump to recovery firmware

2.6 LK regular boot

- Recovery flag or fastboot keys not set.
- Pulls out boot.img from the MMC and loads it into the scratch region (base address = 0x80000000) specified in target/msm8916/rules.mk.
- Loads kernel from the scratch region into KERNEL_ADDR (retrieved from boot image header).
- Loads RAM disk from the scratch region into RAMDISK_ADDR (retrieved from boot image header).
- Finds the right device tree (for the appropriate SoC) from the device tree table and loads it at TAGS_ADDR (retrieved from boot image header).
- Updates the device tree by:
  - Getting the offset for the '/memory' node and '/chosen' node.
  - Adding HLOS memory regions (start address and size) as "reg" properties to '/memory' node.
  - Adding the cmd line as "bootargs" to the '/chosen' node.
  - Adding the RAM disk properties as "linux, initrd-start" and "linux, initrd-end" to the '/chosen' node.
  - Disable cache, interrupts, jump to kernel.
- This boot flow is illustrated through code snippets in the next section.

2.7 Code snippet

2.7.1 boot_linux_from_mmc() {

```
structboot_img_hdr*hdr=(void*)buf; // boot image header
```

```
/* Read boot image header from emmcpartition into buf*/
if(mmc_read(ptn+offset,(unsignedint*)buf,page_size)){… }

/* Read image without signature to the scratch address */
if (mmc_read(ptn+ offset, (void *)image_addr, imagesize_actual)) { …. }

/* Read signature to the scratch address */
if(mmc_read(ptn+ offset, (void *)(image_addr+ offset), page_size))

/* Kernel image authentication */
verify_signed_bootimg(image_addr, imagesize_actual);

/* Move kernel, ramdiskand device tree to correct address */
memmove((void*) hdr->kernel_addr, (char *)(image_addr+ page_size), hdr-
>kernel_size);
memmove((void*) hdr->ramdisk_addr, (char *)(image_addr+ page_size+
kernel_actual), hdr->ramdisk_size);

/* Find the DT table address */
dt_table_offset = ((uint32_t)image_addr + page_size + kernel_actual +
ramdisk_actual + second_actual);
table = (struct dt_table*) dt_table_offset;

/* Calculate the index of device tree within device tree table */
if(dev_tree_get_entry_info(table, &dt_entry) != 0){ }

/* boot_linux : update device tree and jump to kernel */
boot_linux((void *)hdr->kernel_addr, (unsigned *) hdr->tags_addr, (const
char *)cmdline, board_machtype(),
(void *)hdr->ramdisk_addr, hdr->ramdisk_size);
}
```

2.7.2 void boot_linux() { …

```
update_device_tree((void *)tags, final_cmdline, ramdisk, ramdisk_size); /*
shown on the next slide */
…..
if (IS_ARM64(kptr))
scm_elexec_call((paddr_t)kernel, tags_phys); /* Jump to a 64bit kernel */
else
entry(0, machtype, (unsigned*)tags_phys); /* Jump to a 32 bitkernel */
}
```

2.7.3 Code snippet – updating the device tree

```
Int update_device_tree(constvoid*fdt,char*cmdline,
void*ramdisk,unsignedramdisk_size)
{ ……
uint32_t*memory_reg;

/*Checkthedevicetreeheader*/
offset=fdt_check_header(fdt);
…….
/*Getoffsetofthe”memory”node*/
offset=fdt_path_offset(fdt,”/memory”);

/* Update “memory” node* /
ret=target_dev_tree_mem(fdt,offset);

/*Getoffsetofthe”chosen”node*/
ret=fdt_path_offset(fdt,”/chosen”);

/*Addingthecmdlinetothe”chosen”node*/
ret=fdt_setprop_string(fdt,offset,(constchar*)”bootargs”,(constvoid*)cmdlin
e);

/*Addingtheinitrd-start tothechosennode*/
ret=fdt_setprop_u32(fdt,offset,”linux,initrd-start”,(uint32_t)ramdisk);

/*Addingtheinitrd-end tothechosennode*/
ret=fdt_setprop_u32(fdt,offset,”linux,initrd-
end”,((uint32_t)ramdisk+ramdisk_size));
…. }
```

2.8 LK fastboot mode

- aboot_init checks if:
  - boot.img not present, or
  - volume down key is pressed
- Checks reason for reboot – check_reboot_mode.
- Registers handlers for fastbootcommands:

  ```
  fastboot_register(cmd_list[i].name,cmd_list[i].cb);
  ```
- Initializes fastboot

  ```
  fastboot_init(void *base, unsigned size)
  ```

  Creates a thread associated with fastboot_handler()

  Thread waits for USB event

- Sets up USB

```
udc_start()
```

2.9 Fastboot commands

- Fastbootcommands are currently disabled by default on user/production builds due to security considerations.

  File: Top level makefile

```
ifeq($(TARGET_BUILD_VARIANT),user)
CFLAGS += -DDISABLE_FASTBOOT_CMDS=1
endif
```

- To selectively enable any fastbootcommand on user/production build, add the command as shown below:

  File: bootable/bootloader/lk/app/aboot/aboot.c

  Function: void aboot_fastboot_register_commands(void)

```
structfastboot_cmd_desccmd_list[] = {
   /* By default the enabled list is empty. */
   {"", NULL},
   /* move commands enclosed within the below ifndefto here
   * if they need to be enabled in user build.
   */
#ifndefDISABLE_FASTBOOT_CMDS
   /* Register the following commands only for non-user builds */
   {"flash:", cmd_flash},
   {"erase:", cmd_erase},
   {"boot", cmd_boot},
   {"continue", cmd_continue},
   {"reboot", cmd_reboot},
   {"reboot-bootloader", cmd_reboot_bootloader},
   {"oemunlock", cmd_oem_unlock},
   {"oemlock", cmd_oem_lock},
   {"oemverified", cmd_oem_verified},
   {"oemdevice-info", cmd_oem_devinfo},
   {"oemenable-charger-screen", cmd_oem_enable_charger_screen},
   {"oemdisable-charger-screen", cmd_oem_disable_charger_screen},
   {"oem-select-display-panel", cmd_oem_select_display_panel},
#endif
};
```

- cmd_flash("flash")
  - Writes a file to emmc/flash partition.
  - Usage – fastboot flash <partition> [<filename>]

NOTE: A new partition table can be flashed using this command.

Fastboot flash partition <gpt_both0.bin>.

For example: if there are 6 partition tables, we need to specify number 0-5 in the fastboot flash command as follows:

- Fastboot flash partition:0 <gpt_both0.bin>
- Fastboot flash partition:1 <gpt_both1.bin>

- cmd_erase("erase")
  - Erases an individual emmc/flash partition.
  - Usage – fastboot erase <partition>

NOTE: Partition table cannot be erased with this command.

- cmd_boot("boot")
  - Allows us to download a kernel image (and optional ramdisk) and boot the phone with those, instead of using the kernel and ramdisk in the boot partition in emmc.
  - Usage – fastboot boot <kernel> [<ramdisk>]

NOTE: If the device is not unlocked and target_use_signed_kernel() returns 1, this command verifies the authenticity of the kernel image provided.

In verified boot case, if device is not unlocked, this command fails. Users need to do "fastboot oemunlock" to be able to use this command.

- cmd_continue("continue")
  - Allows the system to continue with boot to kernel/HLOS.
  - Usage – fastboot continue
- cmd_reboot("reboot")
  - Reboots the device normally.
  - Usage – fastboot reboot
- cmd_reboot_bootloader(""reboot-bootloader")
  - Reboots the device into fastbootmode.
  - Usage – fastboot reboot-bootloader
- cmd_oem_unlock("oemunlock")
  - Unlocks the device:
    - device.is_unlocked= 1

        - device.is_verified= 0
    - Usage – fastboot oemunlock

NOTE: Users need to make sure to wipe the user data.

- cmd_oem_lock("oemlock")
    - Locks the device.
        - device.is_unlocked= 0
        - device.is_verified= 0
    - Usage – fastboot oemlock

NOTE: Users need to make sure to wipe the user data.

- cmd_oem_verified ("oem verified")
    - Verifies the device.
        - device.is_unlocked = 0
        - device.is_verified = 1
    - Usage − fastboot oemverified

NOTE: Users need to make sure to wipe the user data.

- cmd_oem_devinfo ("oem device-info")
    - Prints following device info:
        - If device is tampered.
        - If device is unlocked.
        - If charger screen is enabled.
    - Usage − fastboot oem device-info
- cmd_oem_enable_charger_screen ("oem enable-charger-screen")
    - Enables the charger screen in Android.
    - Usage − fastboot oem enable-charger-screen
- cmd_oem_disable_charger_screen ("oem disable-charger-screen")
    - Disables the charger screen in Android.
    - Usage − fastboot oem disable-charger-screen
- cmd_oem_select_display_panel ("oem-select-display-panel")
    - Allows to select display panel.
    - Usage − fastboot oem-select-display-panel

2.10 LK recovery mode

- aboot_init checks if KEY_HOME or VOLUME UP is pressed.
- Checks reason for reboot – check_reboot_mode().

  If value at restart reason address is RECOVERY_MODE, sets boot_into_recovery = 1.
- boot_linux_from_mmc checks:

```
if (!boot_into_recovery) {
……
…
else {
    index = partition_get_index("recovery");
    ptn = partition_get_offset(index);
    ……..
    ……
}
```

- Gets image from recovery partition.

```
    gned int target_freq, unsigned int relation);
```

EXHIBIT 1

PLEASE READ THIS LICENSE AGREEMENT ("AGREEMENT") CAREFULLY. THIS AGREEMENT IS A BINDING LEGAL AGREEMENT ENTERED INTO BY AND BETWEEN YOU (OR IF YOU ARE ENTERING INTO THIS AGREEMENT ON BEHALF OF AN ENTITY, THEN THE ENTITY THAT YOU REPRESENT) AND QUALCOMM TECHNOLOGIES, INC. ("QTI" "WE" "OUR" OR "US"). THIS IS THE AGREEMENT THAT APPLIES TO YOUR USE OF THE DESIGNATED AND/OR ATTACHED DOCUMENTATION AND ANY UPDATES OR IMPROVEMENTS THEREOF (COLLECTIVELY, "MATERIALS"). BY USING OR COMPLETING THE INSTALLATION OF THE MATERIALS, YOU ARE ACCEPTING THIS AGREEMENT AND YOU AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS, QTI IS UNWILLING TO AND DOES NOT LICENSE THE MATERIALS TO YOU. IF YOU DO NOT AGREE TO THESE TERMS YOU MUST DISCONTINUE AND YOU MAY NOT USE THE MATERIALS OR RETAIN ANY COPIES OF THE MATERIALS. ANY USE OR POSSESSION OF THE MATERIALS BY YOU IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

1.1 **License.** Subject to the terms and conditions of this Agreement, including, without limitation, the restrictions, conditions, limitations and exclusions set forth in this Agreement, Qualcomm Technologies, Inc. ("QTI") hereby grants to you a nonexclusive, limited license under QTI's copyrights to use the attached Materials; and to reproduce and redistribute a reasonable number of copies of the Materials. You may not use Qualcomm Technologies or its affiliates or subsidiaries name, logo or trademarks; and copyright, trademark, patent and any other notices that appear on the Materials may not be removed or obscured. QTI shall be free to use suggestions, feedback or other information received from You, without obligation of any kind to You. QTI may immediately terminate this Agreement upon your breach. Upon termination of this Agreement, Sections 1.2-4 shall survive.

1.2 **Indemnification.** You agree to indemnify and hold harmless QTI and its officers, directors, employees and successors and assigns against any and all third party claims, demands, causes of action, losses, liabilities, damages, costs and expenses, incurred by QTI (including but not limited to costs of defense, investigation and reasonable attorney's fees) arising out of, resulting from or related to: (i) any breach of this Agreement by You; and (ii) your acts, omissions, products and services. If requested by QTI, You agree to defend QTI in connection with any third party claims, demands, or causes of action resulting from, arising out of or in connection with any of the foregoing.

1.3 **Ownership.** QTI (or its licensors) shall retain title and all ownership rights in and to the Materials and all copies thereof, and nothing herein shall be deemed to grant any right to You under any of QTI's or its affiliates' patents. You shall not subject the Materials to any third party license terms (e.g., open source license terms). You shall not use the Materials for the purpose of identifying or providing evidence to support any potential patent infringement claim against QTI, its affiliates, or any of QTI's or QTI's affiliates' suppliers and/or direct or indirect customers. QTI hereby reserves all rights not expressly granted herein.

1.4 **WARRANTY DISCLAIMER.** YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT THE USE OF THE MATERIALS IS AT YOUR SOLE RISK. THE MATERIALS AND TECHNICAL SUPPORT, IF ANY, ARE PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED. QTI ITS LICENSORS AND AFFILIATES MAKE NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE MATERIALS OR ANY OTHER INFORMATION OR DOCUMENTATION PROVIDED UNDER THIS AGREEMENT, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT, OR ANY EXPRESS OR IMPLIED WARRANTY ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. NOTHING CONTAINED IN THIS AGREEMENT SHALL BE CONSTRUED AS (I) A WARRANTY OR REPRESENTATION BY QTI, ITS LICENSORS OR AFFILIATES AS TO THE VALIDITY OR SCOPE OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT OR (II) A WARRANTY OR REPRESENTATION BY QTI THAT ANY MANUFACTURE OR USE WILL BE FREE FROM INFRINGEMENT OF PATENTS, COPYRIGHTS OR OTHER INTELLECTUAL PROPERTY RIGHTS OF OTHERS, AND IT SHALL BE THE SOLE RESPONSIBILITY OF YOU TO MAKE SUCH DETERMINATION AS IS NECESSARY WITH RESPECT TO THE ACQUISITION OF LICENSES UNDER PATENTS AND OTHER INTELLECTUAL PROPERTY OF THIRD PARTIES.

1.5 **LIMITATION OF LIABILITY.** IN NO EVENT SHALL QTI, QTI'S AFFILIATES OR ITS LICENSORS BE LIABLE TO YOU FOR ANY INCIDENTAL, CONSEQUENTIAL OR SPECIAL DAMAGES, INCLUDING BUT NOT LIMITED TO ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE, OR THE DELIVERY OR FAILURE TO DELIVER, ANY OF THE MATERIALS, OR ANY BREACH OF ANY OBLIGATION UNDER THIS AGREEMENT, EVEN IF QTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE FOREGOING LIMITATION OF LIABILITY SHALL REMAIN IN FULL FORCE AND EFFECT REGARDLESS OF WHETHER YOUR REMEDIES HEREUNDER ARE DETERMINED TO HAVE FAILED OF THEIR ESSENTIAL PURPOSE. THE ENTIRE LIABILITY OF QTI, QTI's AFFILIATES AND ITS LICENSORS, AND THE SOLE AND EXCLUSIVE REMEDY OF YOU, FOR ANY CLAIM OR CAUSE OF ACTION ARISING HEREUNDER (WHETHER IN CONTRACT, TORT, OR OTHERWISE) SHALL NOT EXCEED US$10.

2. **COMPLIANCE WITH LAWS; APPLICABLE LAW.** You agree to comply with all applicable local, international and national laws and regulations and with U.S. Export Administration Regulations, as they apply to the subject matter of this Agreement. This Agreement is governed by the laws of the State of California, excluding California's choice of law rules.

3. **CONTRACTING PARTIES.** If the Materials are downloaded on any computer owned by a corporation or other legal entity, then this Agreement is formed by and between QTI and such entity. The individual accepting the terms of this Agreement represents and warrants to QTI that they have the authority to bind such entity to the terms and conditions of this Agreement.

4. **MISCELLANEOUS PROVISIONS.** This Agreement, together with all exhibits attached hereto, which are incorporated herein by this reference, constitutes the entire agreement between QTI and You and supersedes all prior negotiations, representations and agreements between the parties with respect to the subject matter hereof. No addition or modification of this Agreement shall be effective unless made in writing and signed by the respective representatives of QTI and You. The restrictions, limitations, exclusions and conditions set forth in this Agreement shall apply even if QTI or any of its affiliates becomes aware of or fails to act in a manner to address any violation or failure to comply therewith. You hereby acknowledge and agree that the restrictions, limitations, conditions and exclusions imposed in this Agreement on the rights granted in this Agreement are not a derogation of the benefits of such rights. You further acknowledges that, in the absence of such restrictions, limitations, conditions and exclusions, QTI would not have entered into this Agreement with You. Each party shall be responsible for and shall bear its own expenses in connection with this Agreement. If any of the provisions of this Agreement are determined to be invalid, illegal, or otherwise unenforceable, the remaining provisions shall remain in full force and effect. This Agreement is entered into solely in the English language, and if for any reason any other language version is prepared by any party, it shall be solely for convenience and the English version shall govern and control all aspects. If You are located in the province of Quebec, Canada, the following applies: The Parties hereby confirm they have requested this Agreement and all related documents be prepared in English.