NXP Semiconductors
User's Guide

Document Number: AUG
Rev. O8.1.0_2.0.0-beta, 01/2019

Android™ User's Guide

Contents

1 Overview

This document describes how to build Android Oreo 8.1 platform for the i.MX 8 series devices. It provides instructions for:

- Configuring a Linux® OS build machine.
- Downloading, patching, and building the software components that create the Android™ system image.
- Building from sources and using pre-built images.
- Copying the images to boot media.
- Hardware/software configurations for programming the boot media and running the images.

For more information about building the Android platform, see source.android.com/source/building.html.

2 Preparation

The minimum recommended system requirements are as follows:

- 16 GB RAM
- 300 GB hard disk

For any problems on the building process related to the jack server, see the Android website source.android.com/source/jack.html.

2.1 Setting up your computer

To build the Android source files, use a computer running the Linux OS. The Ubuntu 16.04 64bit version and openjdk-8-jdk is the most tested environment for the Android Oreo 8.1 build.

After installing the computer running Linux OS, check whether all the necessary packages are installed for an Android build. See "Setting up your machine" on the Android website source.android.com/source/initializing.html.

In addition to the packages requested on the Android website, the following packages are also needed:

```
$ sudo apt-get install uuid uuid-dev
$ sudo apt-get install zlib1g-dev liblz-dev
$ sudo apt-get install liblzo2-2 liblzo2-dev
$ sudo apt-get install lzop
$ sudo apt-get install git-core curl
$ sudo apt-get install u-boot-tools
$ sudo apt-get install mtd-utils
$ sudo apt-get install android-tools-fsutils
$ sudo apt-get install openjdk-8-jdk
$ sudo apt-get install device-tree-compiler
$ sudo apt-get install gdisk
```

NOTE

If you have trouble installing the JDK in Ubuntu, see How to install misc JDK in Ubuntu for Android build.

Configure git before use. Set the name and email as follows:

- git config --global user.name "First Last"
- git config --global user.email "first.last@company.com"

2.2 Unpacking the Android release package

After you have set up a computer running Linux OS, unpack the Android release package by using the following commands:

```
# cd ~ (or any other directory where the imx-o8.1.0_2.0.0-beta.tar.gz file is placed)

$ tar xzvf imx-o8.1.0_2.0.0-beta.tar.gz
```

3 Building the Android platform for i.MX

3.1 Getting i.MX Android release source code

The i.MX Android release source code consists of three parts:

- NXP i.MX public source code, which is maintained in the CodeAurora Forum repository.
- AOSP Android public source code, which is maintained in android.googlesource.com.
- NXP i.MX Android proprietary source code package, which is maintained in www.NXP.com

Assume you had i.MX Android proprietary source code package imx-o8.1.0_2.0.0-beta.tar.gz under ~/. directory. To generate the i.MX Android release source code build environment, execute the following commands:

```
$ mkdir ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo  > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=${PATH}:~/bin
$ source ~/imx-o8.1.0_2.0.0-beta/imx_android_setup.sh
# By default, the imx_android_setup.sh script will create the source code build environemnt
in the folder ~/android_build
# ${MY_ANDROID} will be refered as the i.MX Android source code root directory in all i.MX
Andorid release documentation.
$ export MY_ANDROID=~/android_build
```

3.2 Building Android images

Building the Android image is performed when the source code has been downloaded (Section 3.1) and patched (Section 3.2).

Commands **lunch <buildName-buildType>** to set up the build configuration and **make** to start the build process are executed.

The build configuration command **lunch** can be issued with an argument <Build name>-<Build type> string, such as **lunch mek_8q-userdebug**, or can be issued without the argument presenting a menu of selection.

The Build Name is the Android device name found in the directory ${MY_ANDROID}/device/fsl/. The following table lists the i.MX build names.

Table 1. Build names

Build name	Description
mek_8q-userdebug	i.MX 8QuadMax/8QuadXPlus MEK Board

The build type is used to specify what debug options are provided in the final image. The following table lists the build types.

Table 2. Build types

Build type	Description
user	Production ready image, no debug
userdebug	Provides image with root access and debug, similar to "user"
eng	Development image with debug tools

Android build steps are as follows:

1. Change to the top level build directory.

   ```
   $ cd ${MY_ANDROID}
   ```
2. Set up the environment for building. This only configures the current terminal.

   ```
   $ source build/envsetup.sh
   ```
3. Execute the Android **lunch** command. In this example, the setup is for the production image of i.MX 8QuadXPlus MEK Board/Platform device with user type.

   ```
   $ lunch mek_8q-userdebug
   ```
4. Execute the **make** command to generate the image.

   ```
   $ make -j4 2>&1 | tee build-log.txt
   ```

When the **make** command is complete, the build-log.txt file contains the execution output. Check for any errors.

For BUILD_ID & BUILD_NUMBER changing, update build_id.mk in your ${MY_ANDROID} directory. For details, see the *Android™ Frequently Asked Questions* (AFAQ).

The following outputs are generated by default in ${MY_ANDROID}/out/target/product/mek_8q:

- root/: root file system (including init, init.rc). Mounted at /.
- system/: Android system binary/libraries. Mounted at /system.
- data/: Android data area. Mounted at /data.
- recovery/: root file system when booting in "recovery" mode. Not used directly.
- dtbo-imx8qm.img: Board's device tree binary. It is used to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display for i.MX 8QuadMax MEK.
- dtbo-imx8qm-hdmi.img: Board's device tree binary. It is used to support physical HDMI display for i.MX 8QuadMax MEK.
- dtbo-imx8qm-mipi-panel.img: Board's device tree binary. It is used to support MIPI panel display for i.MX 8QuadMax MEK.
- dtbo-imx8qxp.img: Board's device tree binary. It is used to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with dual-camera support for i.MX 8QuadXPlus MEK.
- dtbo-imx8qxp-ov5640mipi.img: Board's device tree binary. It is used to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI displays with single MIPI camera support for i.MX 8QuadXPlus MEK.
- vbmeta-imx8qm.img: Android Verify boot metadata image for dtbo-imx8qm.img. It is used to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI displays for i.MX 8QuadMax MEK.
- vbmeta-imx8qm-hdmi.img: Android Verify boot metadata image for dtbo-imx8qm-hdmi.img. It's used to support physical HDMI display for i.MX 8QuadMax MEK.
- vbmeta-imx8qm-mipi-panel.img: Android Verify boot metadata image for dtbo-imx8qm-mipi-panel.img. It is used to support MIPI panel display for i.MX 8QuadMax MEK.
- vbmeta-imx8qxp.img: Android Verify boot metadata image for dtbo-imx8qxp.img. It is used to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI displays with dual camera support for i.MX 8QuadXPlus MEK.
- vbmeta-imx8qxp-ov5640mipi.img: Android Verify boot metadata image for dtbo-imx8qxp-ov5640mipi.img. It is used to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI displays with single camera support for i.MX 8QuadXPlus MEK.
- ramdisk.img: Ramdisk image generated from "root/". Not directly used.
- system.img: EXT4 image generated from "system/". Can be programmed to "SYSTEM" partition on SD/eMMC card with "dd".
- partition-table.img: GPT partition table image. Used for 16 GB SD card.
- partition-table-7GB.img: GPT partition table image. Used for 8 GB SD card.
- partition-table-28GB.img: GPT partition table image. Used for 32 GB SD card.
- u-boot-imx8qm.imx: U-Boot image with no padding for i.MX 8QuadMax MEK.
- u-boot-imx8qxp.imx: U-Boot image without padding for i.MX 8QuadXPlus MEK.
- u-boot-imx8qm-mek-uuu.imx: U-Boot image used by UUU for i.MX 8QuadMax MEK. It is not flashed to MMC.
- u-boot-imx8qxp-mek-uuu.imx: U-Boot image used by UUU for i.MX 8QuadXPlus MEK. It is not flashed to MMC.
- vendor.img: vendor image, which holds platform binaries. Mounted at **/vendor**.
- boot.img: a composite image that includes the kernel Image, ramdisk, and boot parameters.

NOTE

- To build the U-Boot image separately, see Building U-Boot images.
- To build the kernel uImage separately, see Building a kernel image.
- To build boot.img, see Building boot.img.
- To build dtbo.img, see Building dtbo.img.

3.2.1 Configuration examples of building i.MX devices

The following table shows examples of using the `lunch` command to set up different i.MX devices. After the desired i.MX device is set up, the `make` command is used to start the build.

Table 3. i.MX device lunch examples

Build name	Description
i.MX 8QuadMax/8QuadXPlus MEK Board	$ lunch mek_8q-userdebug

After the `lunch` command is executed, the **make** command is issued:

```
$ make 2>&1 | tee build-log.txt
```

3.2.2 Build mode selection

There are three types of build mode to select: eng, user, and userdebug.

The userdebug build behaves the same as the user build, with the ability to enable additional debugging that normally violates the security model of the platform. This makes the userdebug build with greater diagnosis capabilities for user test.

The eng build prioritizes engineering productivity for engineers who work on the platform. The eng build turns off various optimizations used to provide a good user experience. Otherwise, the eng build behaves similar to the user and userdebug builds, so that device developers can see how the code behaves in those environments.

In a module definition, the module can specify tags with LOCAL_MODULE_TAGS, which can be one or more values of optional (default), debug, eng.

If a module does not specify a tag (by LOCAL_MODULE_TAGS), its tag defaults to optional. An optional module is installed only if it is required by product configuration with PRODUCT_PACKAGES.

The main differences among the three modes are listed as follows:

- eng: development configuration with additional debugging tools
    - Installs modules tagged with: eng and/or debug.
    - Installs modules according to the product definition files, in addition to tagged modules.
    - ro.secure=0
    - ro.debuggable=1
    - ro.kernel.android.checkjni=1
    - adb is enabled by default.
- user: limited access; suited for production
    - Installs modules tagged with user.
    - Installs modules according to the product definition files, in addition to tagged modules.
    - ro.secure=1
    - ro.debuggable=0
    - adb is disabled by default.
- userdebug: like user but with root access and debuggability; preferred for debugging
    - Installs modules tagged with debug.
    - ro.debuggable=1
    - adb is enabled by default.

There are two methods for the build of Android image.

Method 1: Set the environment first and then issue the `make` command:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh   #set env
$ make -j4 PRODUCT-XXX userdebug 2>&1 | tee build-log.txt  #XXX depends on different boards.
See the table below.
```

Table 4. Android system image production build method 1

i.MX development tool	Description	Image build command
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK	$ make -j4 PRODUCT-mek_8q-userdebug

Method 2: Set the environment and then use `lunch` command to configure argument. See table below. An example for the i.MX 8QuadMax/8QuadXPlus MEK board is as follows:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q-userdebug
$ make -j4
```

Table 5. Android system image production build method 2

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK	mek_8q-userdebug

To create Android over-the-air, OTA, and package, the following make target is specified:

```
$ make otapackage -j4
```

For more Android platform building information, see source.android.com/source/building.html.

3.3 Building U-Boot images

You can use this command to generate u-boot.imx under the Android environment:

```
# U-Boot image for i.MX 8QuadMax/8QuadXPlus MEK board:
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q-userdebug
$ make bootloader -j4
```

3.4 Building a kernel image

Kernel image is automatically built when building the Android root file system.

The following are the default Android build commands to build the kernel image:

```
$ cd ${MY_ANDROID}/vendor/nxp-opensource/kernel_imx
$ echo $ARCH && echo $CROSS_COMPILE
```

Make sure that you have those two environment variables set. If the two variables are not set, set them as follows:

```
$ export ARCH=arm64
$ export CROSS_COMPILE=${MY_ANDROID}/prebuilts/gcc/linux-x86/aarch64/aarch64-linux-
android-4.9/bin/aarch64-linux-android-

# Generate ".config" according to default config file under arch/arm64/configs/
```

```
android_defconfig.
# to build the kernel zImage for i.MX 8QuadMax/8QuadXPlus
$ make android_defconfig
$ make KCFLAGS=-mno-android

# to build the zImage which is used in MfgTOOL
# zImage is under mfgtools\Profiles\Linux\OS Firmware\firmware\
$ make defconfig
$ make KCFLAGS=-mno-android -j4
```

The kernel images are found in ${MY_ANDROID}/out/target/product/mek_8q/obj/KERNEL_OBJ/arch/arm64/boot/Image.

3.5 Building boot.img

boot.img and boota are default booting commands.

As outlined in Running the Android Platform with a Prebuilt Image, we use boot.img and boota as default commands to boot instead of the uramdisk and zImage we used before.

Use this command to generate boot.img under Android environment:

```
# Boot image for i.MX 8QuadMax/8QuadXPlus MEK board
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q-userdebug
$ make bootimage -j4
```

3.6 Building dtbo.img

Dtbo image holds the board's device tree binary.

Use the following commands to generate dtbo.img under the Android environment:

```
# dtbo image for the i.MX 8QuadMax/8QuadXPlus MEK board
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q-userdebug
$ make dtboimage -j4
```

4 Running the Android Platform with a Prebuilt Image

To test the Android platform before building any code, use the prebuilt images from the following packages and go to "Download Images" and "Boot".

Table 6. Image packages

Image package	Description
android_o8.1.0_2.0.0-beta_image_8qmek.tar.gz	Prebuilt-image and UUU script files for the i.MX 8QuadMax/8QuadXPlus MEK board, which includes NXP extended features.

The following tables list the detailed contents of android_o8.1.0_2.0.0-beta_image_8qmek.tar.gz image package.

The table below shows the prebuilt images to support the system boot from SD/eMMC on i.MX 8QuadMax/8QuadXPlus MEK boards.

Table 7. Images for i.MX 8QuadMax/8QuadXPlus MEK

i.MX 8QuadMax/8QuadXPlus MEK image	Description
/u-boot-imx8qm.imx	Bootloader (with padding) for i.MX 8QuadMax MEK board
/u-boot-imx8qxp.imx	Bootloader (with padding) for i.MX 8QuadXPlus MEK board
/u-boot-imx8qm-mek-uuu.imx	Bootloader used by UUU for i.MX 8QuadMax MEK board. It is not flashed to MMC.
/u-boot-imx8qxp-mek-uuu.imx	Bootloader used by UUU for i.MX 8QuadXPlus MEK board. It is not flashed to MMC.
/boot.img	Boot image for i.MX 8QuadMax/8QuadXPlus MEK board.
/system.img	System image for i.MX 8QuadMax/8QuadXPlus MEK board.
/vendor.img	Vendor image for i.MX 8QuadMax/8QuadXPlus MEK board.
/partition-table.img	GPT table image for 16 GB SD card
/partition-table-7GB.img	GPT table image for 8 GB SD card
/partition-table-28GB.img	GPT table image for 32 GB SD card
/vbmeta-imx8qm.img	Android Verify Boot metadata Image for i.MX 8QuadMax MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display.
/vbmeta-imx8qm-hdmi.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support physical HDMI display.
/vbmeta-imx8qm-mipi-panel.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support MIPI panel display.
/vbmeta-imx8qxp.img	Android Verify Boot metadata image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with dual camera support.
/vbmeta-imx8qxp-ov5640mipi.img	Android Verify Boot metadata image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with single MIPI camera support.
/dtbo-imx8qm.img	Device Tree image for i.MX 8QuadMax MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display.
/dtbo-imx8qm-hdmi.img	Device Tree image for i.MX 8QuadMax MEK board to support physical HDMI display.
/dtbo-imx8qm-mipi-panel.img	Device Tree image for i.MX 8QuadMax MEK board to support MIPI panel display.
/dtbo-imx8qxp.img	Device Tree image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with dual camera support.
/dtbo-imx8qxp-ov5640mipi.img	Device Tree image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with single MIPI camera support.

The table below describes UUU scripts in android_o8.1.0_2.0.0-beta_image_8qmek.tar.gz. They are used with the UUU binary file to download the images above into the board.

Table 8. UUU scripts

UUU script name	Funcfion
uuu-android-mx8qm-mek-emmc.lst	Used for i.MX 8QuadMax MEK board with UUU binary file to download image files into eMMC
uuu-android-mx8qm-mek-sd.lst	Used for i.MX 8QuadMax MEK board with UUU binary file to download image files into SD card
uuu-android-mx8qxp-mek-emmc.lst	Used for i.MX 8QuadXPlus MEK board with UUU binary file to download image files into eMMC
uuu-android-mx8qxp-mek-sd.lst	Used for i.MX 8QuadXPlus MEK board with UUU binary file to download image files into SD card

NOTE

boot.img is an Android image that stores kernel Image and ramdisk together. It also stores other information such as the kernel boot command line and machine name. This information can be configured in android.mk. It can avoid touching the boot loader code to change any default boot arguments.

5 Programming Images

The images from the prebuilt release package or created from source code contain the U-Boot boot loader, system image, GPT image, vendor image, and vbmeta image. At a minium, the storage devices on the development system (MMC/SD or NAND) must be programmed with the U-Boot boot loader. The i.MX 8 series boot process determines what storage device to access based on the switch settings. When the boot loader is loaded and begins execution, the U-Boot environment space is then read to determine how to proceed with the boot process. For U-Boot environment settings, see Section Booting.

The following download methods can be used to write the Android System Image:

- UUU and UUU script file to download all images to the eMMC/SD card.
- fsl-sdcard-partition.sh to download all images to the SD card.
- fastboot_imx_flashall script to download all images to the eMMC/SD storage.

5.1 System on eMMC/SD

The images needed to create an Android system on eMMC/SD can either be obtained from the release package or be built from source.

The images needed to create an Android system on eMMC/SD are listed below:

- U-Boot image: u-boot.imx
- GPT table image: partition-table.img
- Android dtbo image: dtbo.img
- Android boot image: boot.img
- Android system image: system.img
- Android verify boot metadata image: vbmeta.img
- Android vendor image: vendor.img

5.1.1 Storage partitions

The layout of the eMMC card for Android system is shown below:

- [Partition type/index] which is defined in the GPT.
- [Start Offset] shows where partition is started, unit in MB.

The system partition is used to put the built-out Android system image. The userdata parition is used to put the unpacked codes/data of the applications, system configuration database, etc. In normal boot mode, the root file system is mounted from the system partition. In recovery mode, the root file system is mounted from the boot partition.

Table 9. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader	0 KB (i.MX 8QuadMax eMMC) or 32 KB (i.MX 8QuadXPlus, i.MX 8QuadMax SD card) or 33 KB (i.MX 8M Quad, i.MX 8M Mini)	4 MB	N/A	bootloader
1	dtbo_a	8 MB	4 MB	N/A	dtbo.img
2	dtbo_b	Follow dtbo_a	4 MB	N/A	dtbo.img
3	boot_a	Follow dtbo_b	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
4	boot_b	Follow boot_a	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
5	system_a	Follow boot_b	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
6	system_b	Follow system_a	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
7	misc	Follow system_b	4 MB	N/A	For recovery store bootloader message, reserve
8	metadata	Follow datafootor	2 MB	N/A	For system slide show
9	presistdata	Follow metadata	1 MB	N/A	Option to operate unlock \unlock
10	vendor_a	Follow persistdata	256 MB	EXT4. Mount at / vendor	vendor.img
11	vendor_b	Follow vendor_a	256 MB	EXT4. Mount at / vendor	vendor.img
12	userdata	Follow vendor_b	Remained space	EXT4. Mount at /data	Application data storage for system application, and for internal media partition, in /mnt/sdcard/ dir.

Table continues on the next page...

Table 9. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
13	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
14	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
15	vbmeta_b	Follow vbmeta_a	1 MB	N/A	For storing the verify boot's metadata

To create these partitions, use UUU described in the *Android™ Quick Start Guide* (AQSUG), or use format tools in the prebuilt directory.

The script below can be used to partition an SD Card and download images to them as shown in the partition table above:

```
$ cd ${MY_ANDROID}/
$ sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> /dev/sdX
# <soc_name> can be imx8qm or imx8qxp.
```

NOTE

- The minimum size of the SD card is 8 GB bytes.
- If the SD card is 8 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> -c 7 /dev/sdX` to flash images.
- If the SD card is 16 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> /dev/sdX` to flash images.
- If the SD card is 32 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> -c 28 /dev/sdX` to flash images.
- /dev/sdX, the X is the disk index from 'a' to 'z', which may be different on each Linux PC.
- Unmount all the SD card partitions before running the script.
- Put related bootloader, boot image, system image, and vbmeta image in your current directory.
- This script needs simg2img tool to be installed on your PC. The simg2img is a tool that converts sparse system image to raw system image on the host PC running Linux OS. The android-tools-fsutils package includes the simg2img command for Ubuntu Linux.

5.1.2 Downloading images with UUU

UUU can be used to download all images into a target device. It is a quick and easy tool for downloading images. See the *Android™ Quick Start Guide* (AQSUG) for detailed description of UUU.

5.1.3 Downloading images with fastboot_imx_flashall script

UUU can be used to flash the Android system image into the board, but it needs to make the board enter serial down mode first, and make the board enter boot mode once flashing is finished.

A new fastboot_imx_flashall script is supported to use fastboot to flash the Android system image into the board. It is more flexible. To use the new script, the board must be able to enter fastboot mode and the device must be unlocked. The table below lists the fastboot_imx_flashall scripts.

Table 10. fastboot_imx_flashall script

Name	Host system to execute the script
fastboot_imx_flashall.sh	Linux OS
fastboot_imx_flashall.bat	Windows OS

With the help of fastboot_imx_flashall scripts, you do not need to use fastboot to flash Android images one-by-one manually. These scripts will automatically flash all images with only one command.

The way to use these scripts is follows:

- Linux shell script usage: sudo fastboot_imx_flashall.sh <option>
- Windows batch script usage: fastboot_imx_flashall.bat <option>

```
Options:
      -h                Displays this help message
      -f soc_name       Flashes the Android image file with soc_name
      -a                Only flashes the image to slot_a
      -b                Only flashes the image to slot_b
      -c card_size      Optional setting: 7 / 14 / 28
                        If it is not set, use partition-table.img (default).
                        If it is set to 7, use partition-table-7GB.img for 8 GB SD card.
                        If it is set to 14, use partition-table-14GB.img for 16 GB SD card.
                        If it is set to 28, use partition-table-28GB.img for 32 GB SD card.
                        Make sure that the corresponding file exists on your platform.
      -m               Flashes the Cortex-M4 image.
      -d dev           Flash dtbo, vbmeta, and recovery image file with dev.
                       If it is not set, use default dtbo, vbmeta, and recovery image.
      -e               Erases user data after all image files are flashed.
      -l               Locks the device after all image files are flashed.
      -D directory     Directory of images.
                       If this script is execute in the directory of the images, it does not
need to use this option.
      -s ser_num       Serial number of the board.
                       If only one board connected to computer, it does not need to use this
option
```

NOTE

- -f option is mandatory. SoC name can be imx8qm or imx8qxp.
- Boot the device to U-Boot fastboot mode, and then execute these scripts. The device should be unlocked first.

Example:

```
sudo ./fastboot_imx_flashall.sh -f imx8qm -a -e -D /imx_pi9.0/mek_8q_car/
```

Options explanation:

- -f imx8qm: Flashes images for i.MX 8QuadMax MEK Board.
- -a: Only flashes slot a.
- -e: Erases user data after all image files are flashed.
- -D /imx_pi9.0/mek_8q_car/: Images to be flashed are in the directory of /imx_pi9.0/mek_8q_car/.

6 Booting

This chapter describes booting from MMC/SD.

6.1 Booting from eMMC/SD

6.1.1 Booting from SD/eMMC on the i.MX 8QuadMax/8QuadXPlus MEK board

The following tables list the boot switch settings to control the boot storage.

Table 11. Boot device switch settings

i.MX 8QuadMax boot switch	Download mode (UUU mode)	SD boot	eMMC boot
SW2 Boot_Mode (1-6 bit)	001000	001100	001000
i.MX 8QuadXPlus boot switch	Download mode (UUU mode)	SD boot	eMMC boot
SW2 Boot_Mode (1-4 bit)	1000	1100	0100

To boot from eMMC, change the board Boot_Mode switch to 000100 (1-6bit) for i.MX 8QuadMax or 0100 (1-4bit) for i.MX 8QuadXPlus.

To boot from SD, change the board Boot_Mode switch to 001100 (1-6bit) for i.MX 8QuadMax or 1100 (1-4bit) for i.MX 8QuadXPlus.

The default environment in boot.img is booting from eMMC. To use the default environment in boot.img, use the following command:

```
U-Boot > setenv bootargs
```

To clear the bootargs environment, use the following command:

```
U-Boot > setenv bootargs console=ttyLP0,115200 earlycon=lpuart32,0x5a060000,115200 init=/
init androidboot.console=ttyLP0 consoleblank=0 androidboot.hardware=freescale
androidboot.fbTileSupport=enable cma=800M@0x960M-0xe00M androidboot.primary_display=imx-drm
firmware_class.path=/vendor/firmware transparent_hugepage=never swiotlb=49152 [Optional]
U-Boot > saveenv                         [Save the environments]
```

NOTE

bootargs environment is an optional setting for boota. The boot.img includes a default bootargs, which is used if there is no definition about the bootargs environment.

6.2 Boot-up configurations

This section explains some common boot-up configurations such as U-Boot environments, kernel command line, and DM-verity configuartions.

6.2.1 U-Boot environment

- bootcmd: the first variable to run after U-Boot boot.
- bootargs: the kernel command line, which the bootloader passes to the kernel. As described in Kernel command line (bootargs), bootargs environment is optional for booti. boot.img already has bootargs. If you do not define the bootargs environment, it uses the default bootargs inside the image. If you have the environment, it is then used.

  To use the default environment in boot.img, use the following command to clear the bootargs environment.

  ```
  > setenv bootargs
  ```
- boota:

  boota command parses the boot.img header to get the Image and ramdisk. It also passes the bootargs as needed (it only passes bootargs in boot.img when it cannot find "bootargs" var in your U-Boot environment). To boot from mmcX, do the following:

  ```
  > boota mmcX
  ```

  To read the boot partition (the partition store boot.img, in this instance, mmcblk0p1), the X is the eMMC bus number, which is the hardware eMMC bus number, in SABRE-SD boards. eMMC is mmc2 or you can add the partition ID after mmcX.

  Add partition ID after mmcX.

  ```
  > boota mmcX boot    #  boot is default
  > boota mmcX recovery      # boot from the recovery partition
  ```

  If you have read the boot.img into memory, use this command to boot from

  ```
  > boota 0xXXXXXXXX
  ```

6.2.2 Kernel command line (bootargs)

Depending on the different booting/usage scenarios, you may need different kernel boot parameters set for bootargs.

Table 12. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
console	Where to output kernel log by printk.	console=ttymxc0	i.MX 8M Mini use console=ttymxc0.
init	Tells kernel where the init file is located.	init=/init	All use cases. "init" in the Android platform is located in "/" instead of in "/sbin".
androidboot.console	The Android shell console. It should be the same as console=.	androidboot.console=ttymxc0	To use the default shell job control, such as Ctrl+C to terminate a running process, set this for the kernel.
cma	CMA memory size for GPU/VPU physical memory allocation.	cma=800M or cma=1280M or cma=800M@0x960M-0xe00M • For i.MX 8M Mini and i.MX 8QuadMax, it is 800 MB by default.	Start address is 0x96000000 and end address is 0xDFFFFFFFF. The CMA size can be configured to other value, but cannot exceed 1184 MB, because the Cortex-M4 core will also allocate memory

Table continues on the next page...

Table 12. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
		- For i.MX 8M Quad, it is 1280 MB by default. - For i.MX 8QuadXPlus and 8QuadMax, it is 800 MB by default.	from CMA and Cortex-M4 cannot use the memory larger than 0xDFFFFFFFF.
androidboot.selinux	Argument to disable selinux check and enable serial input when connecting a host computer to the target board's USB UART port. For details about selinux, see Security-Enhanced Linux in Android.	androidboot.selinux=permissive	Android Oreo 8.1 CTS requirement: serial input should be disabled by default. Setting this argument enables console serial input, which will violate the CTS requirement. Setting this argument will also bypass all the selinux rules defined in Android system. It is recommended to set this argument for internal developer.
androidboot.primary_display	It is used to chose and fix primary display.	androidboot.primary_display=imx-drm	androidboot.primary_display=mxsfb-drm is only used for MIPI display.
androidboot.lcd_density	It is used to set the display density and over write ro.sf.lcd_density in init.rc for MIPI-to-HDMI display.	androidboot.lcd_density=160	-
androidboot.displaymode	It is used to configure the kernel/driver work mode/fps.	- 4k display should be configured as: androidboot.displaymode =4k. The default fps is 60fps. To configure fps, change this value to 4kp60/4kp50/4kp30. - 1080p display should be configured as: androidboot.displaymode =1080p. The default fps is 60fps. To configure fps, change this value to 1080p60/1080p50/1080p30. - 720p display should be configured as: androidboot.displaymode =720p. The default fps is 60fps. To configure fps, change this value to 720p60/720p50/720p30. - 480p display should be configured as: androidboot.displaymode =480p. The default fps is 60fps. To configure fps,	The system will find out and work at the best display mode, and display mode can be changed through this bootargs.

Table continues on the next page...

Table 12. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
		change this value to 480p60/480p50/480p30.	
androidboot.fbTileSupport	It is used to enable framebuffer super tile output.	androidboot.fbTileSupport=enable	It should not be set when connecting the MIPI-to-HDMI display or MIPI panel display.
firmware_class.path	It is used to set the Wi-Fi firmware path.	firmware_class.path=/vendor/firmware	-
transparent_hugepage	It is used to change the sysfs boot time defaults of Transparent Hugepage support.	transparent_hugepage=never/always/madvise	-
swiotlb	It is used to configure the swiotlb size. The kernel default value is 64 MB.	swiotlb=49152	i.MX 8QuadMax/8QuadXPlus configures it to 96 MB (swiotlb=49152) to fix the swiotlb overflow issue from the Wi-Fi driver.

6.2.3 DM-verity configuration

DM-verity (device-mapper-verity) provides transparent integrity checking of block devices. It can prevent device from running unauthorized images. This feature is enabled by default. Replacing one or more partitions (boot, vendor, system, vbmeta) will make the board unbootable. Disabling DM-verity provides convience for developers, but the device is unprotected.

To disable DM-verity, perform the following steps:

1. Unlock the device.
   a. Boot up the device.
   b. Choose **Settings** -> **Developer Options** -> **OEM Unlocking** to enable OEM unlocking.
   c. Enter Fastboot mode on the device. Execute the following command on the target side:

      ```
      reboot bootloader
      ```
   d. Unlock the device. Execute the following command on the host side:

      ```
      fastboot oem unlock
      ```
   e. Wait until the unlock process is complete.
2. Disable DM-verity.
   a. Boot up the device.
   b. Disable the DM-verity feature. Execute the following command on the host side:

      ```
      adb root
      adb disable-verity
      adb reboot
      ```

7 Over-The-Air (OTA) Update

7.1 Building OTA update packages

7.1.1 Building target files

You can use the following commands to generate target files under the Android environment:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q-userdebug
$ make target-files-package -j4
```

After building is complete, you can find the target files in the following path:

```
${MY_ANDROID}/out/target/product/mek_8q/obj/PACKAGING/target_files_intermediates/mek_8q-
target_files-${date}.zip
```

7.1.2 Building a full update package

A full update is one where the entire final state of the device (system, boot, and vendor partitions) is contained in the package.

You can use the following commands to build a full update package under the Android environment:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q-userdebug
$ make otapackage -j4
```

After building is complete, you can find the OTA packages in the following path:

```
${MY_ANDROID}/out/target/proudct/mek_8q/mek_8q-ota-${date}.zip
```

`mek_8q-ota-${date}.zip` includes `payload.bin` and `payload_properties.txt`. The two files are used for full update.

NOTE

- `${date}` is the BUILD_NUMBER in `build_id.mk`.

7.1.3 Building an incremental update package

An incremental update contains a set of binary patches to be applied to the data that is already on the device. This can result in considerably smaller update packages:

- Files that have not changed do not need to be included.
- Files that have changed are often very similar to their previous versions, so the package only needs to contain encoding of the differences between the two files. You can install the incremental update package only on a device that has the old or source build used when constructing the package.

Before building an incremental update package, see Section 7.1.1 to build two target files:

- PREVIOUS-target_files.zip: one old package that has already been applied on the device.
- NEW-target_files.zip: the latest package that is waiting to be applied on the device.

Then use the following commands to generate the incremental update package under the Android environment:

```
$ cd ${MY_ANDROID}
$ ./build/tools/releasetools/ota_from_target_files -i PREVIOUS-target_files.zip NEW-
target_files.zip incremental_ota_update.zip
```

`${MY_ANDROID}/incremental_ota_update.zip` includes `payload.bin` and `payload_properties.txt`. The two files are used for incremental update.

7.2 Implementing OTA update

7.2.1 Using update_engine_client to update the Android platform

update_engine_client is a pre-built tool to support A/B (seamless) system updates.

- Copy `ota_update.zip` or `incremental_ota_update.zip` (generated on 7.1.2 and 7.1.3) to the HTTP server (for example, 192.168.1.1:/var/www/).
- Unzip the packages to get `payload.bin` and `payload_properties.txt`.
- Cat the content of `payload_properties.txt` like this:
  - FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
  - FILE_SIZE=379074366
  - METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
  - METADATA_SIZE=46866
- Input the following command on the board's console to update:

  ```
  update_engine_client --payload=http://192.168.1.1:10888/payload.bin --update --
  headers="FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
  FILE_SIZE=379074366
  METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ/de8Dgp9zFXt8Fo
  +Hxccp465uTOvKNsteWU=
  METADATA_SIZE=46866"
  ```

NOTE

Make sure to use a new line for every payload_properties parameter here.

- The system will update in the background. After it finishes, it will show "Update successfully applied, waiting to reboot" in the logcat.

7.2.2 Using a customized application to update the Android platform

There is a reference OTA application unter `${MY_ANDROID}/vendor/nxp-opensource/fsl_imx_demo/FSLOta`, which can do the OTA operations:

1. Get `payload_properties.txt` and `payload.bin` from a specific address.
2. Use the update_engine service to update the Android platform.

Perform the following steps to use this application:

1. Set up the HTTP server (eg., lighttpd, apache).
   You need one HTTP server to hold OTA packages.
   - For full OTA update, execute the following commands:

     ```
     cp ${MY_ANDROID}/out/target/product/mek_8q/system/build.prop ${server_ota_folder}
     cp ${MY_ANDROID}/out/target/product/mek_8q/mek_8q-ota-${date}.zip $
     {server_ota_folder}
     cd ${server_ota_folder}
     unzip mek_8q-ota-${date}.zip
     ```
   - For incremental OTA update, execute the following commands:

```
cp ${old_build.prop} ${server_ota_folder}/old_build.prop
cp ${MY_ANDROID}/out/target/product/mek_8q/system/build.prop ${server_ota_folder}/
build_diff.prop
mkdir ${server_ota_folder}/diff_ota
cp ${MY_ANDROID}/incremental_ota_update.zip ${server_ota_folder}/diff_ota
cd ${server_ota_folder}/diff_ota
unzip incremental_ota_update.zip
mv payload.bin payload_diff.bin
mv payload_properties.txt payload_properties_diff.txt
mv payload_diff.bin payload_properties_diff.txt ${server_ota_folder}
cd ${server_ota_folder}
echo -n "base." >> build_diff.prop
grep "ro.build.date.utc" old_build.prop >> build_diff.prop
```

For example, the server_ota_folder content is like this (Make sure that you have at least 6 files as follows in ${server_ota_folder}, or the OTA application will be aborted):

```
build@server:/var/www/mek_8q_oreo_8$ ls
build.prop build_diff.prop payload.bin payload_diff.bin payload_properties.txt
payload_properties_diff.txt
```

NOTE

- server_ota_folder: ${http_root}/mek_8q_${ota_folder_suffix}_${version}.
- ${old_build.prop} is the old image's build.prop.
- mek_8q-ota-${date}-${soc}.zip and incremental_ota_update.zip are built from Section 7.1.2 "Building a full update package" and Section 7.1.3 "Building an incremental update package".
- ${ota_folder_suffix} is stored at board's /vendor/etc/ota.conf.
- ${version} can be obtained by the following command on the board's console: $getprop ro.build.version.release.
- These file and folder names should align with this example, or modify the OTA application source code correspondingly.

2. Configure the OTA server IP address and HTTP port number.
   The OTA configuration file (/vendor/etc/ota.conf) content is like this:

```
server=192.168.1.100
port=10888
ota_folder_suffix=oreo
```

Modify it to fit the environment.

3. Open the OTA application and click the **Update** button.
   The reference application is a dialogue box activity, and can be enabled through the **Settings** -> **About tablet** -> **Additional system Update** menu. There are two buttons on the dialogue box:
   - **Upgrade**: Performs full OTA.
   - **Diff Upgrade**: Performs incremental OTA.

Click one button to update the Android platform. After update is complete, click the **Reboot** button on the dialogue box.

NOTE

- This application uses the "ro.build.date.utc=1528987645" property to decide whether it can perform full OTA or incremental OTA.
- local utc = $getprop ro.build.date.utc.
- remote utc = cat ${server_ota_folder}/build.prop | grep "ro.build.date.utc".
- remote diff utc = cat ${server_ota_folder}/build_diff.prop | grep "ro.build.date.utc".
- remote diff base utc = cat ${server_ota_folder}/build_diff.prop | grep "base.ro.build.date.utc" (base.ro.build.date.utc should be added manually, which is the "ro.build.date.utc" value in PREVIOUS-target_files.zip's system/build.prop).
- Full OTA condition:

    - local utc < remote utc
- Incremental OTA condition:
    - local utc = remote diff base utc
    - local utc < remote diff utc

NOTE

The OTA package includes the DTBO image, which stores the board's DTB. There may be many DTS for one board. For example, in ${MY_ANDROID}/device/fsl/imx8q/mek_8q/BoardConfig.mk:

```
TARGET_BOARD_DTS_CONFIG ?= imx8mm:fsl-imx8mm-evk.dtb imx8mm-mipi-
panel:fsl-imx8mm-evk-rm67191.dtb imx8mm-dsd:fsl-imx8mm-evk-
ak4497.dtb imx8mm-m4:fsl-imx8mm-evk-m4.dtb
```

There is one variable to specify which dtbo image is stored in the OTA package:

```
BOARD_PREBUILT_DTBOIMAGE := out/target/product/mek_8q/dtbo-
imx8mm.img
```

Therefore, the default OTA package can only be applied for mek_8q with single HDMI display. To generate the OTA package for mek_8q with MIPI panel display, modify this BOARD_PREBUILT_DTBOIMAGE as follows:

```
BOARD_PREBUILT_DTBOIMAGE := out/target/product/mek_8q/dtbo-imx8mm-
mipi-panel.img
```

For detailed information about A/B OTA updates, see https://source.android.com/devices/tech/ota/ab/.

8 Customized Configuration

8.1 How to change boot command line in boot.img

After boot.img is used, the default kernel boot command line is stored inside the image. It packages together during android build.

You can change this by changing BOARD_KERNEL_CMDLINE's definition in BoardConfig.mk file under ${MY_ANDROID}/device/fsl.

NOTE

- For the i.MX 8M Mini board, the source folder is ${MY_ANDROID}/device/fsl/imx8m/mek_8q/BoardConfig.mk.
- For the i.MX 8MQuad EVK board, the source folder is ${MY_ANDROID}/device/fsl/imx8m/evk_8mq/BoardConfig.mk.
- For the i.MX 8QuadMax/8QuadXPlus MEK board, the source folder is ${MY_ANDROID}/device/fsl/imx8q/mek_8q/BoardConfig.mk.

8.2 How to configure the rear and front cameras

Property "back_camera_name" and "front_camera_name" are used to configure which camera to be used as the rear camera or front camera.

The name should be either `v4l2_dbg_chip_ident.match.name` returned from v4l2's IOCTL VIDIOC_DBG_G_CHIP_IDENT or `v4l2_capability.driver` returned from v4l2's IOCTL VIDIOC_QUERYCAP.

Camera HAL goes through all the V4L2 devices in the system. Camera HAL chooses the first matched name in property settings as the corresponding camera. Comma is used as a delimiter of different camera name among multiple-camera selection.

The following is an example set in ${MY_ANDROID}/device/fsl/imx8m/mek_8q/init.rc.

```
setprop back_camera_name mx6s-csi
setprop front_camera_name uvc
```

`media_profiles_V1_0.xml` in `/vendor/etc` is used to configure the parameters used in the recording video. NXP provides several media profile examples that help customer align the parameters with their camera module capability and device definition.

Table 13. Media profile parameters

Profile file name	Rear camera	Front camera
media_profiles_1080p.xml	Maximum to 1080P, 30FPS and 8 Mbps for recording video	Maximum to 720P, 30FPS, and 3 Mbps for recording video
media_profiles_720p.xml	Maximum to 720P, 30FPS, and 3 Mbps for recording video	Maximum to 720P, 30FPS, and 3 Mbps for recording video
media_profiles_480p.xml	Maximum to 480P, 30FPS, and 2 Mbps for recording video	Maximum to 480P, 30FPS, and 2 Mbps for recording video
media_profiles_qvga.xml	Maximum to QVGA, 15FPS, and 128 Kbps for recording video	Maximum to QVGA, 15FPS, and 128 Kbps for recording video

NOTE

Because not all UVC cameras can have 1080P, 30FPS resolution setting, it is recommended that `media_profiles_480p.xml` is used for any board's configuration, which defines the UVC as the rear camera or front camera.

8.3 How to configure the logical display density

The Android UI framework defines a set of standard logical densities to help application developers target application resources.

Device implementations must report one of the following logical Android framework densities:

- 120 dpi, known as 'ldpi'
- 160 dpi, known as 'mdpi'
- 213 dpi, known as 'tvdpi'
- 240 dpi, known as 'hdpi'
- 320 dpi, known as 'xhdpi'
- 480 dpi, known as 'xxhdpi'

Device implementations should define the standard Android framework density that is numerically closest to the physical density of the screen, unless that logical density pushes the reported screen size below the minimum supported.

To configure the logical display density for framework, you must define the following line in `init.rc under ${MY_ANDROID}/device/fsl/`:

```
setprop ro.sf.lcd_density <density>
```

NOTE

- For the i.MX 8QuadMax/8QuadXPlus MEK board, the source folder is ${MY_ANDROID}/device/fsl/imx8q/mek_8q/init.rc.

8.4 How to enable USB 2.0 in U-Boot for i.MX 8QuadMax/8QuadXPlus

There are both USB 2.0 and USB 3.0 ports on i.MX 8QuadMax/8QuadXPlus MEK board. Because U-Boot can support only one USB gadget driver, the USB 3.0 port is enabled by default. To use the USB 2.0 port, modify the configurations to enable it and disable the USB 3.0 gadget driver.

For i.MX 8QuadMax, make the following changes under ${MY_ANDROID}/vendor/nxp-opensource/uboot-imx:

```
diff --git a/configs/imx8qm_mek_android_defconfig b/configs/imx8qm_mek_android_defconfig
index cf14544..ae670cb 100644
--- a/configs/imx8qm_mek_android_defconfig
+++ b/configs/imx8qm_mek_android_defconfig
@@ -31,14 +31,12 @@ CONFIG_USB_TCPC=y

 CONFIG_CMD_USB_MASS_STORAGE=y
 CONFIG_USB_GADGET=y
-# CONFIG_CI_UDC=y
+CONFIG_CI_UDC=y
 CONFIG_USB_GADGET_DOWNLOAD=y
 CONFIG_USB_GADGET_MANUFACTURER="FSL"
 CONFIG_USB_GADGET_VENDOR_NUM=0x18d1
 CONFIG_USB_GADGET_PRODUCT_NUM=0x0d02

-CONFIG_USB_CDNS3=y
-CONFIG_USB_CDNS3_GADGET=y
 CONFIG_USB_GADGET_DUALSPEED=y

 CONFIG_CMD_GPIO=y
diff --git a/include/configs/imx8qm_mek_android.h b/include/configs/imx8qm_mek_android.h
index bc8a5ec..5f5b00b 100644
--- a/include/configs/imx8qm_mek_android.h
+++ b/include/configs/imx8qm_mek_android.h
@@ -40,7 +40,7 @@
  #define CONFIG_FASTBOOT_FLASH

  #define CONFIG_FSL_FASTBOOT
-#define CONFIG_FASTBOOT_USB_DEV 1
+#define CONFIG_FASTBOOT_USB_DEV 0
 #define CONFIG_ANDROID_RECOVERY
```

For i.MX 8QuadXPlus, make the following changes under ${MY_ANDROID}/vendor/nxp-opensource/uboot-imx:

```
diff --git a/configs/imx8qxp_mek_android_defconfig b/configs/imx8qxp_mek_android_defconfig
index ee02e02..e4bbace 100644
--- a/configs/imx8qxp_mek_android_defconfig
+++ b/configs/imx8qxp_mek_android_defconfig
@@ -33,14 +33,14 @@ CONFIG_USB_TCPC=y

 CONFIG_CMD_USB_MASS_STORAGE=y
 CONFIG_USB_GADGET=y
-# CONFIG_CI_UDC=y
+CONFIG_CI_UDC=y
 CONFIG_USB_GADGET_DOWNLOAD=y
 CONFIG_USB_GADGET_MANUFACTURER="FSL"
 CONFIG_USB_GADGET_VENDOR_NUM=0x18d1
 CONFIG_USB_GADGET_PRODUCT_NUM=0x0d02

-CONFIG_USB_CDNS3=y
-CONFIG_USB_CDNS3_GADGET=y
```

```
 CONFIG_USB_GADGET_DUALSPEED=y

 CONFIG_CMD_GPIO=y
diff --git a/include/configs/imx8qxp_mek_android.h b/include/configs/imx8qxp_mek_android.h
index 55f2e15..39022ae 100644
--- a/include/configs/imx8qxp_mek_android.h
+++ b/include/configs/imx8qxp_mek_android.h
@@ -35,7 +35,7 @@
 #define CONFIG_FASTBOOT_FLASH

 #define CONFIG_FSL_FASTBOOT
-#define CONFIG_FASTBOOT_USB_DEV 1
+#define CONFIG_FASTBOOT_USB_DEV 0
 #define CONFIG_ANDROID_RECOVERY
```

To enable USB 2.0 for U-Boot used by UUU, for c language header files, apply the same changes above. For defconfig files, apply the changes above respectively on imx8qm_mek_android_uuu_defconfig and imx8qxp_mek_android_uuu_defconfig. The two defconfig files are specially for U-Boot used by UUU.

9 Revision History

Table 14. Revision history

Revision number	Date	Substantive changes
O8.0.0_1.2.0_8QXP-EAR	12/2017	Initial release
O8.1.0_1.2.0_8QXP-PRC	03/2018	i.MX 8QuadXPlus PRC/Beta release
O8.1.0_1.2.0_8QXP-beta2	08/2018	i.MX 8QuadXPlus Beta2 release
O8.1.0_2.0.0-beta	01/2019	i.MX 8QuadXPlus/8QuadMax Beta release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number AUG
Revision O8.1.0_2.0.0-beta, 01/2019