AUG
Android™ User's Guide

Rev. android-10.0.0_2.0.0 — 20 May 2020 User's Guide

Contents

1 Overview

This document describes how to build Android 10.0 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

2.1 Setting up your computer

To build the Android source files, use a computer running the Linux OS. The Ubuntu 16.04 64-bit version and openjdk-8-jdk is the most tested environment for the Android 10.0 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
sudo apt-get install m4
sudo apt-get install libz-dev
sudo apt-get install bison
sudo apt-get install flex
sudo apt-get install libssl-dev
```

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 you like)
$ tar xzvf imx-android-10.0.0_2.0.0.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-android-10.0.0_2.0.0.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-android-10.0.0_2.0.0/imx_android_setup.sh
# By default, the imx_android_setup.sh script will create the source code build environemnt in the
folder `pwd`/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=`pwd`/android_build
```

3.2 Building Android images

The Android image can be built after the source code has been downloaded (Section 3.1 "Getting i.MX Android release source code").

Command **source build/envsetup.sh** to import shell functions first needs to be executed to import shell functions in **${MY_ANDROID}/build/envsetup.sh**.

Commands **lunch <BuildName-BuildMode>** to set up the build configuration.

After the two commands above are executed, the build process is not started yet. It is at a stage that the next command is necessary to be used to start the build process. The behaviour of the i.MX Android build system is used to be aligned with the original Android platform. The **make** command can start the build process and all images will be built out. There are some differences. A shell script named "imx-make.sh" is provided and its symlink file can be found under ${MY_ANDROID} directory, and "./imx-make.sh " should be executed first to start the build process.

The original purpose of this "imx-make.sh" is to build U-Boot/kernel before building Android images.

Google started to put a limit on the host tools used when compiling Android code from Android10.0. Some host tools necessary for building U-Boot/kernel now cannot be used in the Android build system, which is under the control of soong_ui, so U-Boot/kernel cannot be built together with Android images. Google also recommends to use prebuilt binaries for U-Boot/kernel in Android build system. It takes some steps to build U-Boot/kernel to binaries and put these binaries in proper directories, so some specific Android images depending on these binaries can be built without error. "imx-make.sh" is then added to do these steps to simplify the build work. After U-Boot/kernel are compiled, any build commands in standard Android can be used.

"imx-make.sh" can also start the soong_ui with the "make" function in "${MY_ANDROID}/build/envsetup.sh" to build the Android images after U-Boot/kernel are compiled, so customers can still build the i.MX Android images with just one command with this script.

The build configuration command **lunch** can be issued with an argument <Build name>-<Build mode> string, such as **lunch evk_8mm-userdebug**, or can be issued without the argument, which will present a menu of options to select.

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
evk_8mm	i.MX 8M Mini EVK Board
evk_8mn	i.MX 8M Nano EVK Board
evk_8mq	i.MX 8M Quad EVK Board
mek_8q	i.MX 8QuadMax/i.MX 8QuadXPlus MEK Board

The "Build mode" is used to specify what debug options are provided in the final image. The following table lists the build modes.

Table 2. Build types

Build mode	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. Prepare the build environment for U-Boot. This step is optional, if there is still GCC cross-compile tool chain in AOSP codebase, and this tool chain outputs the following log indicating that this tool chain will be deprecated.

```
    Android GCC has been deprecated in favor of Clang, and will be removed from
    Android in 2020-01 as per the deprecation plan in:
    https://android.googlesource.com/platform/prebuilts/clang/host/linux-x86/+/master/
GCC_4_9_DEPRECATION.md
```

   An approach is provided to use the self-installed GCC cross-compile tool chain.

   a. Download the tool chain for the A-profile architecture on arm Developer GNU-A Downloads page. It is recommended to use the 8.3 version for this release. You can download the "gcc-arm-8.3-2019.03-x86_64-aarch64-elf.tar.xz" or "gcc-arm-8.3-2019.03-x86_64-aarch64-linux-gnu.tar.xz". The first one is dedicated for compiling bare-metal programs, and the second one can also be used to compile the application programs.

   b. Decompress the file into a path on local disk, for example, to "/opt/". Export a variable named "AARCH64_GCC_CROSS_COMPILE" to point to the tool as follows:

```
# if "gcc-arm-8.3-2019.03-x86_64-aarch64-elf.tar.xz" is used
export AARCH64_GCC_CROSS_COMPILE=/opt/gcc-arm-8.3-2019.03-x86_64-aarch64-elf/bin/
```

```
aarch64-elf-
    # if "gcc-arm-8.3-2019.03-x86_64-aarch64-linux-gnu.tar.xz" is used
    export AARCH64_GCC_CROSS_COMPILE=/opt/gcc-arm-8.3-2019.03-x86_64-aarch64-linux-gnu/bin/
aarch64-linux-gnu-
```

The preceding command can be added to "/etc/profile". When the host boots up, "AARCH64_GCC_CROSS_COMPILE" is set and can be directly used.

2. Change to the top level build directory.

```
$ cd ${MY_ANDROID}
```

3. Set up the environment for building. This only configures the current terminal.

```
$ source build/envsetup.sh
```

4. Execute the Android **lunch** command. In this example, the setup is for the production image of i.MX 8M Mini EVK Board/ Platform device with userdebug type.

```
$ lunch evk_8mm-userdebug
```

5. Execute the **imx-make.sh** script to generate the image.

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

The commands below can achieve the same result:

```
$ ./imx-make.sh bootloader kernel -j4 2>&1 | tee build-log.txt
# Build U-Boot/kernel with imx-make.sh first, but not to build Android images.
$ make -j4 2>&1 | tee -a build-log.txt
# Start the process of build Android images with "make" function.
```

The output of **make** command will be written to standard output and build-log.txt. If there is any errors when building the image, error logs can be found in the build-log.txt file for checking.

To change BUILD_ID & BUILD_NUMBER, update build_id.mk in ${MY_ANDROID}/device/fsl/ directory. For details, see the Android™ Frequently Asked Questions.

The following outputs are generated by default in ${MY_ANDROID}/out/target/product/evk_8mm:

- **root/**: root file system (including init, init.rc). Mounted at /.
- **system/**: Android system binary/libraries. Mounted at /system.
- **recovery/**: root file system when booting in "recovery" mode. Not used directly.
- **dtbo-imx8mm.img**: Board's device tree binary. It is used to support MIPI-DSI-to-HDMI output and Direct Stream Digital (DSD) playback on boards.
- **dtbo-imx8mm-m4.img**: Board's device tree binary. It is used to support MIPI-DSI-to-HDMI output and audio playback based on Cortex-M4 FreeRTOS on boards with LPDDR4.
- **dtbo-imx8mm-mipi-panel**: Board's device tree binary. It is used to support MIPI Panel output on boards with LPDDR4.
- **vbmeta-imx8mm.img**: Android Verify boot metadata image for dtbo-imx8mm.img.
- **vbmeta-imx8mm-m4.img**: Android Verify boot metadata image for dtbo-imx8mm-m4.img.
- **vbmeta-imx8mm-mipi-panel.img**: Android Verify boot metadata image for dtbo-imx8mm-mipi-panel.img.
- **ramdisk.img**: Ramdisk image generated from "root/". Not directly used.
- **system.img:** EXT4 image generated from "system/" and "root/".
- **product.img**: EXT4 image generated from "product/".

- **partition-table.img**: GPT partition table image for single bootloader condition. Used for 16 GB sdcard and eMMC.
- **partition-table-dual.img**: GPT partition table image for dual bootloader condition. Used for 16 GB sdcard and eMMC.
- **partition-table-28GB.img**: GPT partition table image for single bootloader condition. Used for 32 GB sdcard.
- **partition-table-28GB-dual.img**: GPT partition table image for dual bootloader condition. Used for 32 GB sdcard.
- **u-boot-imx8mm.imx**: U-Boot image without Trusty OS integrated for i.MX 8M Mini EVK on board.
- **u-boot-imx8mm-trusty.imx**: U-Boot image with Trusty OS integrated for i.MX 8M Mini EVK with LPDDR4 on board.
- **u-boot-imx8mm-trusty-secure-unlock.imx**: U-Boot image with Trusty OS integrated and demonstration secure unlock mechanism for i.MX 8M Mini EVK with LPDDR4 on board.
- **u-boot-imx8mm-evk-uuu.imx**: U-Boot image used by UUU for i.MX 8M Mini EVK on board. It is not flashed to MMC.
- **spl-imx8mm-dual.bin**: SPL image without Trusty related configuration for i.MX 8M Mini EVK with LPDDR4 on board.
- **spl-imx8mm-trusty-dual.bin**: SPL image with Trusty related configuration for i.MX 8M Mini EVK with LPDDR4 on board.
- **bootloader-imx8mm-dual.img**: Bootloader image without Trusty OS integrated for i.MX 8M Mini EVK with LPDDR4 on board.
- **bootloader-imx8mm-trusty-dual.img**: Bootloader image with Trusty OS integrated for i.MX 8M Mini EVK with LPDDR4 on board.
- **imx8mm_mcu_demo.img**: MCU FreeRTOS image to support audio playback on MCU side.
- **vendor.img**: Vendor image, which holds platform binaries. Mounted at **/vendor**.
- **super.img**: super image that is generated with system.img, vendor.img, and product.img.
- **boot.img**: A composite image, which includes the kernel Image, ramdisk, and boot parameters.
- **rpmb_key_test.bin**: Prebuilt test RPMB key. Can be used to set the RPMB key as fixed 32 bytes 0x00.
- **testkey_public_rsa4096.bin**: Prebuilt AVB public key. It is extracted from the default AVB private key.

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 with userdebug build mode. After the desired i.MX device is set up, the `imx-make.sh` script is used to start the build.

Table 3. i.MX device lunch examples

Build name	Description
i.MX 8M Mini EVK LPDDR4 board	$ lunch evk_8mm-userdebug
i.MX 8M Nano EVK board	$ lunch evk_8mn-userdebug
i.MX 8M Quad EVK board	$ lunch evk_8mq-userdebug
i.MX 8QuadMax/i.MX 8QuadXPlus MEK board	$ lunch mek_8q-userdebug

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 makefile variable `LOCAL_MODULE_TAGS`, which can be one or more values of `optional` (default), `debug`, `eng`, and `tests`. The values of `debug` and `eng` are deprecated. It is recommended to use `PRODUCT_PACKAGES_ENG` and `PRODUCT_PACKAGES_DEBUG` to specify the modules in the appropriate product makefiles.

If a module does not specify a tag with `$(LOCAL_MODULE_TAGS)`, its tag is `optional` by default. An optional module is installed only if it is required by product configuration with the makefile variable `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 via LOCAL_MODULE_TAGS, or specified by PRODUCT_PACKAGES_ENG and/or PRODUCT_PACKAGES_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 via LOCAL_MODULE_TAGS, or specified by PRODUCT_PACKAGES_DEBUG.
  - ro.debuggable=1
  - adb is enabled by default.

To build of Android images, an example for the i.MX 8M Mini EVK LPDDR4 target is:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh   #set env
$ lunch evk_8mm-userdebug
$ ./imx-make.sh -j4
```

The commands below can achieve the same result.

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh bootloader kernel -j4
$ make -j4
```

Table 4. Android system image production build

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 8M Mini EVK LPDDR4	evk_8mm-userdebug
Evaluation Kit	i.MX 8M Nano EVK	evk_8mn-userdebug
Evaluation Kit	i.MX 8M Quad EVK	evk_8mq-userdebug
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK	mek_8q-userdebug

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

3.2.3 Building with GMS package

Get the Google Mobile Services (GMS) package from Google. Put the GMS package into **${MY_ANDROID}/vendor/partner_gms** folder. Make sure **product.mk*** file have the following line:

```
$(call inherit-product-if-exists, vendor/partner_gms/products/gms.mk)
```

Then build the images. The GMS package will be installed into the target images.

NOTE

product.mk means the build target make file. For example, for i.MX 8M Mini EVK Board, the product.mk is named `device/fsl/imx8m/evk_8mm/evk_8mm.mk`.

3.3 Building U-Boot images

Use the following command to generate `u-boot.imx` under the Android environment:

```
# U-Boot image for i.MX 8M Mini EVK LPDDR4 board
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh bootloader -j4
```

For other platforms, use `lunch <buildName-buildMode>` to set up the build configuration. For detailed build configuration, see Section 3.2 "Building Android images".

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}
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh kernel -c -j4
```

The kernel images are found in ${MY_ANDROID}/out/target/product/evk_8mm/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 the system.

The following commands are used to generate boot.img under Android environment:

```
# Boot image for i.MX 8M Mini EVK LPDDR4 board
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh bootimage -j4
```

The commands below can achieve the same result:

```
# Boot image for i.MX 8M Mini EVK board
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh kernel -j4
$ make bootimage -j4
```

For other platforms, use lunch <buildName-buildMode> to set up the build configuration. For detailed build configuration, see Section 3.2 "Building Android images".

3.6 Building dtbo.img

Dtbo image holds the device tree binary of the board.

The following commands is used to generate dtbo.img under Android environment:

```
# dtbo image for i.MX 8M Mini EVK LPDDR4 board
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh dtboimage -j4
```

The commands below can achieve the same result:

```
# dtbo image for i.MX 8M Mini EVK board
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh kernel -j4
$ make dtboimage -j4
```

For other platforms, use lunch <buildName-buildMode> to set up the build configuration. For detailed build configuration, see Section 3.2 "Building Android images".

4 Running the Android Platform with a Prebuilt Image

Table 5. Image packages

Image package	Description
android-10.0.0_2.0.0_image_8mmevk.tar.gz	Prebuilt-image for i.MX 8M Mini EVK LPDDR4 board, which includes NXP extended features.
android-10.0.0_2.0.0_image_8mnevk.tar.gz	Prebuilt-image for i.MX 8M Nano EVK board, which includes NXP extended features.
android-10.0.0_2.0.0_image_8mqevk.tar.gz	Prebuilt-image for i.MX 8M Quad EVK board, which includes NXP extended features.
android-10.0.0_2.0.0_image_8qmek.tar.gz	Prebuilt-image for i.MX 8QuadMax/8QuadXPlus MEK board, which includes NXP extended features.

The following tables list the detailed contents of android-10.0.0_2.0.0_image_8mmevk.tar.gz image package.

Table 6. Images for i.MX 8M Mini

i.MX 8M Mini EVK image	Description
spl-imx8mm-dual.bin	Secondary program loader image without Trusty related configurations for i.MX 8M Mini EVK board with LPDDR4 on board.
spl-imx8mm-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i.MX 8M Mini EVK board with LPDDR4 on board.
bootloader-imx8mm-dual.img	An image containing U-Boot proper and ATF. It is for i.MX 8M Mini EVK board with LPDDR4 on board.
bootloader-imx8mm-trusty-dual.img	An image containing U-Boot proper, ATF and Trusty OS. It is for i.MX 8M Mini EVK board with LPDDR4 on board.
u-boot-imx8mm.imx	An image containing U-Boot and ATF for i.MX 8M Mini EVK board with LPDDR4 on board.
u-boot-imx8mm-trusty.imx	An image containing U-Boot, ATF and Trusty OS for i.MX 8M Mini EVK board with LPDDR4 on board.
u-boot-imx8mm-trusty-secure-unlock.imx	An image containing U-Boot, ATF and Trusty OS for i.MX 8M Mini EVK board with LPDDR4 on board. It is a demonstration of secure unlock mechanism.
u-boot-imx8mm-evk-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8M Mini board with LPDDR4. It is not flashed to MMC.
boot.img	Boot image for i.MX 8M Mini EVK board, which contains kernel, ramdisk, and default kernel commandline.
system.img	System image for i.MX 8M Mini EVK board.
vendor.img	Vendor image for i.MX 8M Mini EVK board.
product.img	Product image for i.MX 8M Mini EVK board.
partition-table.img	GPT partition table image for single bootloader condition. Used for 16 GB SD card and eMMC.
partition-table-dual.img	GPT partition table image for dual bootloader condition. Used for 16 GB sdcard and eMMC.
partition-table-28GB.img	GPT partition table image for single bootloader condition. Used for 32 GB sdcard.
partition-table-28GB-dual.img	GPT partition table image for dual bootloader condition. Used for 32 GB SD card.
imx8mm_mcu_demo.img	The MCU FreeRTOS image for i.MX 8M Mini EVK board.
dtbo-imx8mm.img	Device Tree image for i.MX 8M Mini EVK board to support LPDDR4 and MIPI-DSI-to-HDMI output and DSD playback.
dtbo-imx8mm-m4.img	Device Tree image for i.MX 8M Mini EVK board to support LPDDR4, MIPI-DSI-to-HDMI output, and audio playback based on Cortex-M4 FreeRTOS.
dtbo-imx8mm-mipi-panel.img	Device Tree image for i.MX 8M Mini EVK board to support LPDDR4 and MIPI panel output.
vbmeta-imx8mm.img	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support LPDDR4 and MIPI-DSI-to-HDMI output.
vbmeta-imx8mm-m4.img	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support LPDDR4, MIPI-DSI-to-HDMI output, and Cortex-M4 playback.

Table continues on the next page...

Table 6. Images for i.MX 8M Mini (continued)

i.MX 8M Mini EVK image	Description
vbmeta-imx8mm-mipi-panel.img	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support LPDDR4 and MIPI panel output.
rpmb_key_test.bin	Prebuilt test RPMB key, which can be used to set the RPMB key as fixed 32 bytes 0x00.
testkey_public_rsa4096.bin	Prebuilt AVB public key, which is extracted from the default AVB private key.

The following tables list the detailed contents of android-10.0.0_2.0.0_image_8mnevk.tar.gz image package.

Table 7. Images for i.MX 8M Nano

i.MX 8M Nano EVK Images	Descriptions
spl-imx8mn-dual.bin	Secondary program loader image without Trusty related configurations for i.MX 8M Nano EVK board.
spl-imx8mn-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i.MX 8M Nano EVK board.
bootloader-imx8mn-dual.img	An image containing U-Boot proper and ATF. It is for i.MX 8M Nano EVK board.
bootloader-imx8mn-trusty-dual.img	An image containing U-Boot proper, ATF, and Trusty OS. It is for i.MX 8M Nano EVK board.
u-boot-imx8mn.imx	An image containing U-Boot and ATF for i.MX 8M Nano EVK board.
u-boot-imx8mn-trusty.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Nano EVK board with LPDDR4 on board.
u-boot-imx8mn-trusty-secure-unlock.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Nano EVK board. It is a demonstration of secure unlock mechanism.
u-boot-imx8mn-evk-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8M Nano board. It is not flashed to MMC.
boot.img	Boot image for i.MX 8M Nano EVK board. It contains kernel, ramdisk, and default kernel commandline.
system.img	System image for i.MX 8M Nano EVK board.
vendor.img	Vendor image for i.MX 8M Nano EVK board.
product.img	Product image for i.MX 8M Nano EVK board.
partition-table.img	GPT partition table image for single-bootloader condition. Used for 16 GB SD card and eMMC.
partition-table-dual.img	GPT partition table image for dual-bootloader condition. Used for 16 GB SD card and eMMC.
partition-table-28GB.img	GPT partition table image for SD single bootloader condition. Used for 32 GB SD card.
partition-table-28GB-dual.img	GPT partition table image for dual-bootloader condition. Used for 32 GB SD card.
imx8mn_mcu_demo.img	The MCU demonstration image for i.MX 8M Nano EVK board.

Table continues on the next page...

Table 7. Images for i.MX 8M Nano (continued)

i.MX 8M Nano EVK Images	Descriptions
dtbo-imx8mn.img	Device Tree image for i.MX 8M Nano EVK board to support MIPI-DSI-to-HDMI output.
dtbo-imx8mn-rpmsg.img	Device Tree image for i.MX 8M Nano EVK board to support MIPI-DSI-to-HDMI output and MCU image.
dtbo-imx8mn-mipi-panel.img	Device Tree image for i.MX 8M Nano EVK board to support MIPI panel output.
vbmeta-imx8mn.img	Android Verify Boot metadata image for i.MX 8M Nano EVK board to support MIPI-DSI-to-HDMI output.
vbmeta-imx8mn-rpmsg.img	Android Verify Boot metadata image for i.MX 8M Nano EVK board to support MIPI-DSI-to-HDMI output and MCU image.
vbmeta-imx8mn-mipi-panel.img	Android Verify Boot metadata image for i.MX 8M Nano EVK board to support MIPI panel output.
rpmb_key_test.bin	Prebuilt test RPMB key. It can be used to set the RPMB key as fixed 32 bytes 0x00.
testkey_public_rsa4096.bin	Prebuilt AVB public key. It is extracted from default AVB private key.

The following tables list the detailed contents of android-10.0.0_2.0.0_image_8mqevk.tar.gz image package.

Table 8. Images for i.MX 8M Quad EVK

i.MX 8M Quad EVK image	Description
spl-imx8mq-dual.bin	Secondary program loader image without Trusty related configurations for i.MX 8M Quad EVK board.
spl-imx8mq-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i.MX 8M Quad EVK board.
bootloader-imx8mq-dual.img	An image containing U-Boot proper and ATF. It is for i.MX 8M Quad EVK board.
bootloader-imx8mq-trusty-dual.img	An image containing U-Boot proper, ATF and Trusty OS. It is for i.MX 8M Quad EVK board.
u-boot-imx8mq.imx	An image containing U-Boot and ATF for i.MX 8M Quad EVK board.
u-boot-imx8mq-trusty.imx	An image containing U-Boot, ATF and Trusty OS for i.MX 8M Quad EVK board.
u-boot-imx8mq-trusty-secure-unlock.imx	An image containing U-Boot, ATF and Trusty OS for i.MX 8M Quad EVK board. It is a demonstration of secure unlock mechanism.
u-boot-imx8mq-evk-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8M Quad EVK board. It is not flashed to mmc.
boot.img	Boot image for i.MX 8M Quad EVK board.
system.img	System image for i.MX 8M Quad EVK board.
vendor.img	Vendor image for i.MX 8M Quad EVK board.
product.img	Product image for i.MX 8M Quad EVK board.
partition-table.img	GPT partition table image for single bootloader condition. Used for 16 GB sdcard and eMMC.

Table continues on the next page...

Table 8. Images for i.MX 8M Quad EVK (continued)

i.MX 8M Quad EVK image	Description
partition-table-dual.img	GPT partition table image for dual bootloader condition. Used for 16GB sdcard and eMMC.
partition-table-28GB.img	GPT partition table image for single bootloader condition. Used for 32 GB sdcard.
partition-table-28GB-dual.img	GPT partition table image for dual bootloader condition. Used for 32 GB sdcard.
dtbo-imx8mq.img	Device Tree image for i.MX 8M Quad EVK REV A board to support HDMI output and DSD playback.
dtbo-imx8mq-mipi.img	Device Tree image for i.MX 8M Quad EVK REV A board to support MIPI-DSI-to-HDMI output.
dtbo-imx8mq-dual.img	Device Tree image for i.MX 8M Quad EVK REV A board to support HDMI and MIPI-DSI-to-HDMI dual output.
dtbo-imx8mq-mipi-panel.img	Device Tree image for i.MX 8M Quad EVK REV A board to support MIPI panel output.
vbmeta-imx8mq.img	Android Verify Boot metadata image for i.MX 8M Quad EVK REV A board to support HDMI output.
vbmeta-imx8mq-mipi.img	Android Verify Boot metadata image for i.MX 8M Quad EVK REV A board to support MIPI-DSI-to-HDMI output.
vbmeta-imx8mq-dual.img	Android Verify Boot metadata image for i.MX 8M Quad EVK REV A board to support HDMI and MIPI-DSI-to-HDMI dual output.
vbmeta-imx8mq-mipi-panel.img	Android Verify Boot metadata image for i.MX 8M Quad EVK REV A board to support MIPI panel output.
rpmb_key_test.bin	Prebuilt test RPMB key, which can be used to set the RPMB key as fixed 32 bytes 0x00.
testkey_public_rsa4096.bin	Prebuilt AVB public key. It is extracted from default AVB private key.

The following tables list the detailed contents of android-10.0.0_2.0.0_image_8qmek.tar.gz image package.

Table 9. Images for i.MX 8QuadMax/8QuadXPlus MEK

i.MX 8QuadMax/8QuadXPlus MEK image	Description
u-boot-imx8qm.imx	An image containing U-Boot and ATF for for i.MX 8QuadMax MEK board.
u-boot-imx8qm-trusty.imx	An image containing U-Boot, ATF and Trusty OS for for i.MX 8QuadMax MEK board.
u-boot-imx8qxp.imx	An image containing U-Boot and ATF for i.MX 8QuadXPlus MEK board.
u-boot-imx8qxp-trusty.imx	An image containing U-Boot, ATF and Trusty OS for i.MX 8QuadXPlus MEK board.
u-boot-imx8qm-mek-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8QuadMax MEK board. It is not flashed to MMC.
u-boot-imx8qxp-mek-uuu.imx	An image containing U-Boot and ATF, 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.

Table continues on the next page...

Table 9. Images for i.MX 8QuadMax/8QuadXPlus MEK (continued)

i.MX 8QuadMax/8QuadXPlus MEK image	Description
vendor.img	Vendor image for i.MX 8QuadMax/8QuadXPlus MEK board.
product.img	Product image for i.MX 8QuadMax/8QuadXPlus MEK board.
partition-table.img	GPT partition table image for 16 GB SD card and eMMC.
partition-table-28GB.img	GPT partition table image for 32 GB boot sdcard.
vbmeta-imx8qm.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support LVDS-to-HDMI/MIPI-DSI-to-HDMI display.
vbmeta-imx8qm-md.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support multiple displays.
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-DSI-to-HDMI or dual LVDS-to-HDMI displays with dual cameras support.
dtbo-imx8qm.img	Device Tree image for i.MX 8QuadMax MEK board to support LVDS-to-HDMI/MIPI-DSI-to-HDMI display.
dtbo-imx8qm-md.img	Device Tree image for i.MX 8QuadMax MEK board to support multiple displays.
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-DSI-to-HDMI or dual LVDS-to-HDMI display with dual cameras support.
rpmb_key_test.bin	Prebuilt test RPMB key, which can be used to set the RPMB key as fixed 32 bytes 0x00.
testkey_public_rsa4096.bin	Prebuilt AVB public key. It is extracted from default AVB private key.

NOTE

boot.img is an Android image that stores Image and ramdisk together. It can also store other information such as the kernel boot command line and machine name. This information can be configured in android.mk. It can avoid touching 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 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
- Android product image: product.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 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 firstly mounted with ramdisk from boot partition, and then the logical system partition is mounted and switched as root. In recovery mode, the root file system is mounted with ramdisk from the boot partition.

Table 10. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader0	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	spl.imx/u-boot.imx
(1)	bootloader_a	8 MB	4 MB	N/A	bootloader.img
(2)	bootloader_b	Following bootloader_a	4 MB	N/A	bootloader.img
1/(3)	dtbo_a	8 MB (following bootloader_b)	4 MB	N/A	dtbo.img
2/(4)	dtbo_b	Follow dtbo_a	4 MB	N/A	dtbo.img
3 (5)	boot_a	Follow dtbo_b	64 MB	boot.img format, a kernel + recovery ramdisk	boot.img
4 (6)	boot_b	Follow boot_a	64 MB	boot.img format, a kernel + recovery ramdisk	boot.img

Table continues on the next page...

Table 10. Storage partitions (continued)

5 (7)	misc	Follow boot_b	4 MB	N/A	For recovery storage bootloader message, reserve.
6 (8)	metadata	Follow misc	2 MB	N/A	For system slide show
7 (9)	presistdata	Follow metadata	1 MB	N/A	the option to operate unlock\unlock
8 (10)	super	Follow presistdata	7168 MB	N/A	system.img, vendor.img, and product.img
9 (11)	userdata	Follow super	Remained space	EXT4. Mount at /data	Application data storage for system application. And for internal media partition, in /mnt/sdcard/ dir.
10 (12)	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
11 (13)	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
12 (14)	vbmeta_b	Follow vbmeta_a	1 MB	N/A	For storing the verify boot's metadata

NOTE

For the preceding table, in the "Partition Type/Index" column and "Start offset" column, the contents in brackets is specific for dual-bootloader condition.

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 imx8mm,imx8mn,imx8mq,imx8qm,imx8qxp.
```

NOTE

- 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, product image, and vbmeta image in your current directory, or use `-D <directory_containing_images>` to specify the directory path in which there are the images to be flashed.
- 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 11. 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.

fastboot can be built with Android build system. Based on Section 3, which introduces how to build android images, perform the following steps to build fastboot:

```
    $ cd ${MY_ANDROID}
    $ make -j4 fastboot
```

After the build process finishes building fastboot, the directory to find the fastboot is as follows:

- Linux version binary file: ${MY_ANDROID}/out/host/linux-x86/bin
- Windows version binary file: ${MY_ANDROID}/out/host/windows-x86/bin

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 MCU image.
      -u uboot_feature  Flashes U-Boot or spl&bootloader images with "uboot_feature" in their names
                            For Standard Android:
                                If the parameter after "-u" option contains the string of "dual", the
spl&bootloader image will be flashed;
                                Otherwise U-Boot image will be flashed.
                            For Android Automative:
                                Only dual-bootloader feature is supported. By default, spl&bootloader
image will be flashed.
```

```
      -d dtb_feature    flash dtbo, vbmeta and recovery image file with "dtb_feature" in their names
                            If 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
      -super            Do not generate super.img when flash the images with dynamic partition feature
enabled.
                        Use the super.img already existed together with other images.
```

NOTE

- -f option is mandatory. SoC name can be imx8mm, imx8mn, imx8mq, 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 imx8mm -a -e -u trusty -D /imx_android-10.0/evk_8mm/
```

Options explanation:

- -f imx8mm: flashes images for i.MX 8M Mini EVK Board.
- -a: Only flashes slot a.
- -e: Erases user data after all image files are flashed.
- -D /imx_pi9.0/evk_8mm/: images to be flashed are in the directory of /imx_android-10.0/evk_8mm/.
- -u trusty: Flashes the "u-boot-imx8mm-trusty.imx".

5.1.4 Downloading a single image with fastboot

Sometimes only a single image needs to be flashed again with fastboot for debug purpose.

With dynamic partition feature enabled, fastboot is also implemented in userspace (recovery) in addition to the implementation in U-Boot. The partitions are categorized into three. Fastboot implemented in U-Boot and userspace can individually recognize part of the partitions. The relationship between them are listed in the following table.

Table 12. Relationship betwen partitions

Partition category	Partition	Can be recognized by
U-Boot hard-coded partition	bootloader0, gpt, mcu_os	U-Boot fastboot
EFI partition	boot_a, boot_b, dtbo_a, dtbo_b, vbmeta_a, vbmeta_b, misc, metadata, presistdata, super, userdata, fbmisc	U-Boot fastboot, userspace fastboot
Logical partition	system_a, system_b, vendor_a, vendor_b, product_a, product_b	userspace fastboot

To enter U-Boot fastboot mode, for example, make the board enter U-Boot command mode, and execute the following command on the console:

```
> fastboot 0
```

To enter userspace fastboot mode, two commands are provided as follows for different conditions. You may need root permission on Linux OS:

```
# board in U-Boot fastboot mode, execute the following command on the host
$ fastboot reboot fastboot

# board boot up to the Android system, execute the following command on the host
$ adb reboot fastboot
```

To use fastboot tool on the host to operate on a specific partition, choose the proper fastboot implemented on the device, which can recoginize the partition to be operated on. For example, to flash the system.img to the partition of system_a, make the board enter userspace fastboot mode, and execute the following command on the host:

```
$ fastboot flash system_a system.img
```

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 8M Mini EVK board

The following tables list the boot switch settings to control the boot storage for Rev. C boards with LPDDR4.

Table 13. Boot device switch settings

Boot device switch	SW1101 (1-10 bit)	SW1102 (1-10 bit)
SD boot	0110110010	0001101000
Download mode	1010xxxxxx	xxxxxxxxxx
eMMC boot	0110110001	0001010100

For Rev. C boards with LPDDR4:

To test booting from SD, change the board Boot_Mode switch to SW1101 0110110010 (1-10 bit) and SW1102 0001101000 (1-10 bit).

To test booting from eMMC, change the board Boot_Mode switch to SW1101 0110110010 (1-10 bit) and SW1102 0001010100 (1-10 bit).

The default environment is in boot.img. To use the default environment in boot.img, do not set bootargs environment in U-Boot.

To clear the bootargs environment being set and saved before, use the following command:

```
U-Boot > setenv bootargs
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 bootargs defined in U-Boot.

6.1.2 Booting from SD/eMMC on the i.MX 8M Nano board

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

Table 14. Boot device switch settings

Boot mode switch	SW1101 (from 1-4 bit)
SD boot	1100
eMMC boot	0100
Download mode	1000

- To boot from SD, change the board Boot_Mode switch to SW1101 1100 (from 1-4 bit).
- To boot from eMMC, change the board Boot_Mode switch to SW1101 0100 (from 1-8 bit).

The default environment is in boot.img. To use the default environment in boot.img, do not set bootargs environment in U-Boot.

To clear the bootargs environment being set and saved before, use the following command:

```
U-Boot > setenv bootargs
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 if there is no bootargs defined in U-Boot.

6.1.3 Booting from SD/eMMC on the i.MX 8M Quad EVK board

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

Table 15. Boot device switch settings

Boot device switch	External SDcard	eMMC
SW01 (1-2 bit)	1100	0010

Table 16. Boot mode switch settings

Boot mode switch	Download Mode (MfgTool mode)	Boot mode
SW02 (1-2 bit)	01	10

To test booting from SD, change the board Boot_Mode switch to 10 (1-2 bit) and SW801 1100 (1-4 bit).

To test booting from eMMC, change the board Boot_Mode switch to 10 (1-2 bit) and SW801 0010 (1-4 bit).

The default environment is in boot.img. To use the default environment in boot.img, do not set bootargs environment in U-Boot.

To clear the bootargs environment being set and saved before, use the following command:

```
U-Boot > setenv bootargs
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 if there is no bootargs defined in U-Boot.

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

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

Table 17. Boot device switch settings

Boot mode switch	SW2 (from 1-6 bit)
SD boot	001100
eMMC boot	000100
Download mode	001000

To test booting from SD, change the board Boot_Mode switch to 001100 (1-6 bit).

To test booting from eMMC, change the board Boot_Mode switch to 000100 (1-6 bit).

The default environment is in boot.img. To use the default environment in boot.img, do not set bootargs environment in U-Boot.

To clear the bootargs environment being set and saved before, use the following command:

```
U-Boot > setenv bootargs
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 if there is no bootargs defined in U-Boot.

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

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

Table 18. Boot device switch settings

Boot mode switch	SW2 (from 1-4 bit)
SD boot	1100
eMMC boot	0100
Download mode	1000

To test booting from SD, change the board Boot_Mode switch to 1100 (1-4 bit).

To test booting from eMMC, change the board Boot_Mode switch to 0100 (1-4 bit).

The default environment is in boot.img. To use the default environment in boot.img, do not set bootargs environment in U-Boot.

To clear the bootargs environment being set and saved before, use the following command:

```
U-Boot > setenv bootargs
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 if there is no bootargs defined in U-Boot.

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 command 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. For i.MX 8M Mini EVK, eMMC is is mmc1. For i.MX 8M Quad EVK, i.MX 8QuadMax MEK, and i.MX 8QuadXPlus MEK, eMMC is mmc0. You can add partition ID after mmcX.

  Add partition ID after mmcX.

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

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 19. 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=ttymxc1.
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 19. 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 10.0 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-DSI-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	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 19. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
		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, 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-DSI-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	-
androidboot.wificountrycode=CN	It is used to set Wi-Fi country code. Different countries use different Wi-Fi channels. For details, see the i.MX Android Frequently Asked Questions.	androidboot.wificountrycode=CN	-
transparent_hugepage	It is used to change the sysfs boot time defaults of Transparent Hugepage support.	transparent_hugepage=never/always/madvise	-
loop.max_part	Defines how many partitions to be able to manage per loop device.	loop.max_part=7	-

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. Execute the following command on the target side to make the board enter fastboot mode:

```
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 evk_8mm-userdebug
$ ./imx-make.sh bootloader kernel -j4
$ make target-files-package -j4
```

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

```
${MY_ANDROID}/out/target/product/evk_8mm/obj/PACKAGING/target_files_intermediates/evk_8mm-ota-**.zip
```

7.1.2 Building a full update package

A full update is one where the entire final state of the device (system, boot, product, 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 evk_8mm-userdebug
$ ./imx-make.sh bootloader kernel -j4
$ make otapackage -j4
```

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

```
${MY_ANDROID}/out/target/proudct/evk_8mm/evk_8mm-ota-**.zip
```

`evk_8mm-ota-**.zip` includes `payload.bin` and `payload_properties.txt`. These two files are used for full update, which is called `full-ota.zip` for convenience.

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.zip
```

`${MY_ANDROID}/incremental-ota.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. It supports update system from a remote server or board's storage.

To update system from a remote server, perform the following steps:

1. Copy `full-ota.zip` or `incremental-ota.zip` (generated on 7.1.2 and 7.1.3) to the HTTP server (for example, 192.168.1.1:/var/www/).
2. Unzip the packages to get `payload.bin` and `payload_properties.txt`.
3. Cat the content of `payload_properties.txt` like this:
   - FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
   - FILE_SIZE=379074366
   - METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
   - METADATA_SIZE=46866

4. Input the following command on the board's console to update:

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

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

To update system from board's storage, perform the following steps:

1. Unzip `full-ota.zip` or `incremental-ota.zip` (Generated on 7.1.2 and 7.1.3) to get `payload.bin` and `payload_properties.txt`.
2. Push payload.bin to board's /sdcard dir: `adb push payload.bin /sdcard/`.
3. Cat the content of payload_properties.txt like this:
   - FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
   - FILE_SIZE=379074366
   - METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
   - METADATA_SIZE=46866
4. Input the following command on the board's console to update:

```
su
update_engine_client --payload=file:///sdcard/payload.bin --update --
headers="FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
METADATA_SIZE=46866"
```

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

NOTE

Make sure that the -- header equals to the exact content of payload_properties.txt without "space" or "return" character.

7.2.2 Using a customized application to update the Android platform

Google has provided a reference OTA application (named as SystemUpdaterSample) under `${MY_ANDROID}/bootable/recovery/updater_sample`, which can do the OTA operations. Perform the following steps to use this application:

1. Generate json configuration file from the OTA package.

```
PYTHONPATH=$MY_ANDROID/build/make/tools/releasetools:$PYTHONPATH \
bootable/recovery/updater_sample/tools/gen_update_config.py \
--ab_install_type=STREAMING \
--ab_force_switch_slot \
full-ota.zip \
full-ota.json \
http://192.168.1.1:10888/full-ota.zip
```

And you can use the following command to generate incremental OTA json file:

```
PYTHONPATH=$MY_ANDROID/build/make/tools/releasetools:$PYTHONPATH \
bootable/recovery/updater_sample/tools/gen_update_config.py \
--ab_install_type=STREAMING \
--ab_force_switch_slot \
incremental-ota.zip \
incremental-ota.json \
http://192.168.1.1:10888/incremental-ota.zip
```

NOTE

http://192.168.1.1:10888/full-ota.zip is a remote server address, which can hold the OTA package.

2. Set up the HTTP server (eg., lighttpd, apache).

   You need one HTTP server to hold OTA packages.

```
scp full-ota.zip ${server_ota_folder}
scp incremental-ota.zip ${server_ota_folder}
```

NOTE

- `server_ota_folder` is one folder on your remote server to hold OTA packages.
- `full-ota.zip` and `incremental-ota.zip` are built from Building a full update package and Building an incremental update package.

3. Push json files to the board.

   a. Use the following command to push json files to the board:

```
adb push full-ota.json /data/local/tmp
adb push incremental-ota.json /data/local/tmp
```

   b. Use the following command to move json files to the private folder of the SystemUpdaterSample application:

```
su
mkdir -m 777 -p /data/user/0/com.example.android.systemupdatersample/files
mkdir -m 777 -p /data/user/0/com.example.android.systemupdatersample/files/configs
cp /data/local/tmp/*.json /data/user/0/com.example.android.systemupdatersample/files/
configs
chmod 777 /data/user/0/com.example.android.systemupdatersample/files/configs/*.json
```

NOTE

If you use the Android Automative system, move json files to the `user/10` folder as follows:

```
su
mkdir -m 777 -p /data/user/10/com.example.android.systemupdatersample/files
mkdir -m 777 -p /data/user/10/com.example.android.systemupdatersample/files/
configs
cp /data/local/tmp/*.json /data/user/10/com.example.android.systemupdatersample/
files/configs
chmod 777 /data/user/10/com.example.android.systemupdatersample/files/configs/
*.json
```

4. Open the SystemUpdaterSample OTA application.

There are many buttons on the UI. The following are their brief description:

```
Reload - reloads update configs from device storage.
View config - shows selected update config.
Apply - applies selected update config.
Stop - cancel running update, calls UpdateEngine#cancel.
Reset - reset update, calls UpdateEngine#resetStatus, can be called only when update is not
running.
Suspend - suspend running update, uses UpdateEngine#cancel.
Resume - resumes suspended update, uses UpdateEngine#applyPayload.
Switch Slot - if ab_config.force_switch_slot config set true, this button will be enabled after
payload is applied, to switch A/B slot on next reboot.
```

First, choose the desired json configuration file. Then, click the **APPLY** button to do the update. After update is complete, you can see "SUCCESS" in the **Engine error** text field, and "REBOOT_REQUIRED" in the **Updater state** text field. Finally, reboot the board to finish the whole OTA update.

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/imx8m/evk_8mm/BoardConfig.mk::

```
TARGET_BOARD_DTS_CONFIG ?= imx8mm:fsl-imx8mm-trusty-evk.dtb
TARGET_BOARD_DTS_CONFIG += imx8mm-mipi-panel:fsl-imx8mm-evk-rm67191.dtb
TARGET_BOARD_DTS_CONFIG += 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/evk_8mm/dtbo-imx8mm.img
```

Therefore, the default OTA package can only be applied for evk_8mm with single MIPI-DSI-to-HDMI display. To generate an OTA package for evk_8mm with MIPI panel display, modify this BOARD_PREBUILT_DTBOIMAGE as follows:

```
BOARD_PREBUILT_DTBOIMAGE := out/target/product/evk_8mm/dtbo-imx8mm-mipi-
panel.img
```

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

For detailed information about the SystemUpdaterSample application, see https://android.googlesource.com/platform/bootable/recovery/+/refs/heads/master/updater_sample/.

8 Customized Configuration

8.1 Camera configuration

Camera HAL on running will read the information in `/vendor/etc/configs/camera_config_${ro.boot.soc_type}.json` to configure the camera. `${ro.boot.soc_type}` is the value of property `ro.boot.soc_type`. The source of this json file is in the repository under `${MY_ANDROID}/device/fsl/`. To configure the camera, make modifications on this source file.

Some parameters have default values in the camera HAL. It is not necessary to set these parameters in the json file if the default values can have cameras work normally.

8.1.1 Configuring the rear and front cameras

`camera_type` and `camera_name` can be used together in the camera configuration json file to specify the camera used as the front or rear camera.

The value of `camera_type` can be "front" and "back". "front" represents the front camera, and "back" represents the rear camera.

The value of "camera_name" represents the camer. It should be either `v4l2_dbg_chip_ident.match.name` returned from `v4l2's VIDIOC_DBG_G_CHIP_IDENT ioctl` or `v4l2_capability.driver` returned from `v4l2's VIDIOC_QUERYCAP ioctl`. `v4l2_dbg_chip_ident` and `v4l2_capability` are structure types defined in camera HAL. Camera HAL will go through all the V4L2 device present in the system to find the corresponding camera and output the information to logcat.

`OmitFrame` is used to skip the first several frames. `cam_blit_csc` is used to specify the hardware used to do csc in camera HAL. `cam_blit_copy` is used to specify the hardware used to do memory copy in camera HAL.

`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 20. 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.

8.1.2 Configuring camera sensor parameters

Camera sensor parameters are used to calculate view angle when doing panorama. The focal length and sensitive element size should be customized based on the camera sensor being used. The release have the parameters for OV5640 as the rear camera.

The following table lists the parameters for camera sensor. These parameters can be configured in the camera configuration json file.

Table 21. Camera sensor parameters

Parameter	Description
ActiveArrayWidth	Maximum active pixel width for camera sensor
ActiveArrayHeight	Maximum active pixel height for camera sensor
PixelArrayWidth	Maximum pixel width for camera sensor
PixelArrayHeight	Maximum pixel height for camera sensor
FocalLength	Focal length
MinFrameDuration	Minimum FPS
MaxFrameDuration	Maximum FPS
MaxJpegSize	Maximum JPEG size
PhysicalWidth	PixelArrayWidth * siz_of_one_pixel Ffor OV5640, it is 1.4 um; For max9286, it is 4.2 um.)

Table continues on the next page...

Table 21. Camera sensor parameters (continued)

PhysicalHeight	PixelArrayHeight * siz_of_one_pixel (For OV5640, it is 1.4 um; For max9286, it is 4.2 um.)

8.2 Audio configuration

8.2.1 Enabling low-power audio

The "DirectAudioPlayer" application is provided to support audio playback from DirectOutputThread. The source code is in `${MY_ANDROID}/vendor/nxp-opensource/fsl_imx_demo/DirectAudioPlayer`. After the "vendor.audio.lpa.enable" property is set to 1, low-power audio can be enabled. In this situation, audio can keey playing even if the system enters suspending mode.

By default, the music stream plays from MixedThread. To make stream play from DirectOutputThread, add the AUDIO_OUTPUT_FLAG_DIRECT flag to the related tracks. On the Android Application layer, there is no AUDIO_OUTPUT_FLAG_DIRECT flag to specify DirectOutputThread explicitly. Instead, use FLAG_HW_AV_SYNC when there is "new AudioTrack" in the application. Then the Android audio framework will add AUDIO_OUTPUT_FLAG_DIRECT for this track, and this stream will play from DirectOutputThread.

In low-power audio mode, the default audio period time is 1 second, and the whole buffer can hold 60 seconds data. These two parameters can be configured by the `vendor.audio.lpa.period_ms` and `vendor.audio.lpa.hold_second` properties as follows:

```
> setprop vendor.audio.lpa.hold_second 60
> setprop vendor.audio.lpa.period_ms 1000
```

To enable low-power audio, perform the following steps:

1. Flash boot-imx8mm-m4.img, imx8mm_m4_demo.img, and vbmeta-imx8mm-m4.img to support audio playback based on Cortex-M4 FreeRTOS.
2. Add `bootmcu` to `bootcmd` in U-Boot command line, see Section 3.4.2 "Booting with Single MIPI-DSI-to-HDMI display and audio playback based on Cortex-M4 FreeRTOS" in the *Android™ Quick Start Guide* (AQSUG).
3. Run the following command to enable low-power audio mode:

   ```
   > su
   > setprop vendor.audio.lpa.enable 1
   > pkill audioserver
   ```

4. Push the .wav audio files to /sdcard/. It is better to use a long duration audio file.
5. Disable the following system sounds:

   ```
   Settings -> Sound -> Touch sounds
   Settings -> Sound -> Screen locking sounds
   Settings -> Sound -> Charging sounds
   ```

6. Open the DirectAudioPlayer application, and select a file from the spinner. The file selected is listed under the spinner.
7. Click the **Play** button to play audio.
8. Press the ON/OFF button on the board. The system then enters suspend mode, and the audio can keep playing.

NOTE

- Only the i.MX 8M Mini EVK board supports this feature. The audio is output from the "LPA output" port on the audio expansion board. See Figure "i.MX 8M Mini EVK with audio board" in the *Android™ Quick Start Guide* (AQSUG).
- DirectAudioPlayer supports limited audio files, which is declared in device's audio_policy_configuration.xml with AUDIO_OUTPUT_FLAG_DIRECT|AUDIO_OUTPUT_FLAG_HW_AV_SYNC flag. Other medias are not supported. For example, it does not support playing 44100Hz audio.
- DirectAudioPlayer supports 24/32 bits wav file with sampling rates no more than 192000.

8.3 Display configuration

8.3.1 Configuring 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.

The default display density value is defined in `${MY_ANDROID}/device/fsl/` as follows:

```
BOARD_KERNEL_CMDLINE += androidboot.lcd_density=240
```

The display density value can be changed by modifying the related lines mentioned above in files under `${MY_ANDROID}/device/fsl/` and recompiling the code or setting in U-Boot command line as bootargs during boot up.

NOTE

- For i.MX 8M Mini EVK Board, the source folder is ${MY_ANDROID}/device/fsl/imx8m/evk_8mm/BoardConfig.mk.
- For i.MX 8M Nano EVK Board, source folder is ${MY_ANDROID}/device/fsl/imx8m/evk_8mn/BoardConfig.mk.
- For i.MX 8MQuad EVK Board, the source folder is ${MY_ANDROID}/device/fsl/imx8m/evk_8mq/BoardConfig.mk.
- For i.MX 8QuadMax/8QuadXPlus MEK, the source folder is ${MY_ANDROID}/device/fsl/imx8q/mek_8q/BoardConfig.mk.

8.3.2 Enabling multiple-display function

The following boards support more than one displays.

Table 22. Boards supporting multiple displays

Board	Number of displays	Display port

Table continues on the next page...

Table 22. Boards supporting multiple displays (continued)

i.MX 8QuadMax MEK	4	• If physical HDMI is used: HDMI_TX, LVDS0_CH0, LVDS1_CH0, MIPI_DSI1 • If physical HDMI is not used: LVDS0_CH0 and LVDS1_CH0, MIPI_DSI0 and MIPI_DSI1
i.MX 8QuadXPlus MEK	2	DSI0/LVDSI0, DSI1/LVDSI1
i.MX 8M Quad EVK	2	HDMI, MIPI-DSI-to-HDMI

The two displays on i.MX 8QuadXPlus MEK are enabled by default.

To evaluate the multiple-display feature with phyical HDMI on i.MX 8QuadMax MEK, flash `dtbo-imx8qm-md.img`.

To evaluate the multiple-display feature on i.MX 8MQuad EVK, flash `dtbo-imx8mq-dual.img`.

8.3.2.1 Binding the display port with the input port

The display port and input port are bound together based on the input device location and display-id. `/vendor/etc/input-port-associations.xml` is used to do this work when the system is running, but the input device location and display-id changes with the change of connection forms of these ports with corresponding input and display devices, which means the input location and display-id need to be retrieved before the connection is fixed.

The source file of `/vendor/etc/input-port-associations.xml` is in the repository under the `${MY_ANDROID}/device/fsl/` directory.

Take i.MX 8QuadMax MEK as an example:

1. Use the following commands to obtain the display port number:

```
dumpsys SurfaceFlinger --display-id
Display 4693505326422272 (HWC display 0): port=0 pnpId=DEL displayName="DELL P2314T"
Display 4693505326422273 (HWC display 1): port=1 pnpId=DEL displayName="DELL P2314T"
Display 4692921138614786 (HWC display 2): port=2 pnpId=DEL displayName="DELL S2740L"
Display 18309706364381699 (HWC display 3): port=3 pnpId=PHL displayName="PHL 245C5"
```

2. Use the following commands to obtain the touch input location:

```
getevent -i | grep location
location: "usb-xhci-cdns3-1.3.4/input0"
location: "usb-xhci-cdns3-1.2.4/input0"
```

3. Bind the display port and input location as follows and modify the configuration file. This file needs to be modified according to actual connection. One display port can be bound with multiple input ports.

```
<ports>
    <port display="0" input="usb-xhci-cdns3-1.1.4/input0" />
    <port display="1" input="usb-xhci-cdns3-1.2.4/input0" />
    <port display="2" input="usb-xhci-cdns3-1.3.4/input0" />
    <port display="3" input="usb-xhci-cdns3-1.4.4/input0" />
    <port display="0" input="usb-xhci-cdns3-1.4/input0" />
    <port display="0" input="usb-ci_hdrc.0-1.4/input0" />
</ports>
```

To make the modifications take effect, modify the source file under the `${MY_ANDROID}/device/fsl/` directory and re-build the images. Keep the connection of diplay devices and input devices unchanged and reflash the images. Or you can disable dm-verity on the board and then use the `adb push` command to push the file to the vendor partition to overwrite the original one.

8.3.2.2 Enabling multi-client input method

Only multi-client IMEs can support typing at the same time with different displays. The following is the way to enable the pre-installed multi-client IME.

```
# Enable multi-client IME for the side-loaded sample multi-client IME
adb root
adb shell setprop persist.debug.multi_client_ime
com.example.android.multiclientinputmethod/.MultiClientInputMethod
adb reboot
```

To disable multi-client IME on non-supported devices again, just clear `persist.debug.multi_client_ime` as follows. Reboot is still required for this to take effect.

```
# Disable multi-client IME again
adb root
adb shell "setprop persist.debug.multi_client_ime ''"
adb reboot
```

The pre-installed multi-client IME in the system is just a sample multi-client IME from AOSP. The performance is not as good as default Google Input Method Editor. If users want to develop multi-client IMEs, see the document in source code (`${MY_ANDROID}/frameworks/base/services/core/java/com/android/server/inputmethod/multi-client-ime.md`).

8.3.2.3 Launching applications on different displays

To launch a certain application to certain display, you need to select the Home application (MultiDisplay or Quickstep). The MultiDisplay is the new launcher for multi-display feature. The Quickstep is the original launcher of Android. If Quickstep is selected as Home application, you can also tap the "MD Launcher" application to get multi-display home screen. Select different display ports on the top of the popup menu, the application user selected will show on specific display port.

8.4 Wi-Fi/Bluetooth configuration

8.4.1 Enabling or disabling Bluetooth profile

Default enabled Bluetooth profiles for Android build are configured in this file: `${MY_ANDROID}/packages/apps/Bluetooth/res/values/config.xml`.

For example, `<bool name="profile_supported_a2dp">true</bool>` indicates that the A2DP profile is enabled. `<bool name="profile_supported_a2dp_sink">false</bool>` indicates that A2DP_sink profile is disabled.

To change enabled Bluetooth profiles, add an overlay file in `${MY_ANDROID}/device/fsl/` to overwrite the default Bluetooth profile configuration.

The following is an example to set A2DP_sink enabled and A2DP disabled for the evk_8mm board.

The file is `${MY_ANDROID}/device/fsl/imx8m/evk_8mm/overlay/packages/apps/Bluetooth/res/values/config.xml`.

```
<resources>
  <bool name="profile_supported_a2dp">false</bool>
  <bool name="profile_supported_a2dp_sink">true</bool>
</resources>
```

8.5 USB configuration

8.5.1 Enabling USB 2.0 in U-Boot for i.MX 8QuadMax/8QuadXPlus MEK

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, to enable USB 2.0 for the u-boot-imx8qm.imx, 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 af6c9e4a87..680d664a7d 100644
--- a/configs/imx8qm_mek_android_defconfig
+++ b/configs/imx8qm_mek_android_defconfig
@@ -115,13 +115,11 @@ CONFIG_DM_USB_GADGET=y
 CONFIG_SPL_DM_USB_GADGET=y
 CONFIG_USB=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=0x0525
 CONFIG_USB_GADGET_PRODUCT_NUM=0xa4a5
-CONFIG_USB_CDNS3=y
-CONFIG_USB_CDNS3_GADGET=y
 CONFIG_USB_GADGET_DUALSPEED=y

 CONFIG_SPL_USB_GADGET=y
@@ -138,7 +136,7 @@ CONFIG_FSL_FASTBOOT=y
 CONFIG_FASTBOOT_BUF_ADDR=0x98000000
 CONFIG_FASTBOOT_BUF_SIZE=0x19000000
 CONFIG_FASTBOOT_FLASH=y
-CONFIG_FASTBOOT_USB_DEV=1
+CONFIG_FASTBOOT_USB_DEV=0

 CONFIG_BOOTAUX_RESERVED_MEM_BASE=0x88000000
 CONFIG_BOOTAUX_RESERVED_MEM_SIZE=0x01000000
diff --git a/include/configs/imx8qm_mek_android.h b/include/configs/imx8qm_mek_android.h
index 10b334da41..3c62b8b54d 100644
--- a/include/configs/imx8qm_mek_android.h
+++ b/include/configs/imx8qm_mek_android.h
@@ -45,7 +45,6 @@

-#define CONFIG_FASTBOOT_USB_DEV 1
 #define CONFIG_ANDROID_RECOVERY

 #define CONFIG_CMD_BOOTA
```

For i.MX 8QuadXPlus, to enable USB 2.0 for the u-boot-imx8qxp.imx, 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 b468515014..314c2fcd53 100644
--- a/configs/imx8qxp_mek_android_defconfig
+++ b/configs/imx8qxp_mek_android_defconfig
@@ -114,13 +114,11 @@ CONFIG_DM_USB_GADGET=y
 CONFIG_SPL_DM_USB_GADGET=y
 CONFIG_USB=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=0x0525
 CONFIG_USB_GADGET_PRODUCT_NUM=0xa4a5
-CONFIG_USB_CDNS3=y
-CONFIG_USB_CDNS3_GADGET=y
 CONFIG_USB_GADGET_DUALSPEED=y

 CONFIG_SPL_USB_GADGET=y
@@ -137,7 +135,7 @@ CONFIG_FSL_FASTBOOT=y
 CONFIG_FASTBOOT_BUF_ADDR=0x98000000
 CONFIG_FASTBOOT_BUF_SIZE=0x19000000
 CONFIG_FASTBOOT_FLASH=y
-CONFIG_FASTBOOT_USB_DEV=1
+CONFIG_FASTBOOT_USB_DEV=0

 CONFIG_SYS_I2C_IMX_VIRT_I2C=y
 CONFIG_I2C_MUX_IMX_VIRT=y
diff --git a/include/configs/imx8qxp_mek_android.h b/include/configs/imx8qxp_mek_android.h
index a9542f48d6..1f4cb9be17 100644
--- a/include/configs/imx8qxp_mek_android.h
+++ b/include/configs/imx8qxp_mek_android.h
@@ -37,7 +37,6 @@
 #define CONFIG_SYS_MALLOC_LEN           (64 * SZ_1M)
 #endif

-#define CONFIG_FASTBOOT_USB_DEV 1
 #define CONFIG_ANDROID_RECOVERY

 #define CONFIG_CMD_BOOTA
```

More than one defconfig files are used to build U-Boot images for one platform. Make the same changes on defconfig files as above to enable USB 2.0 for other U-Boot images. You can use the following command under the `${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/` directory to list all related defconfig files:

```
ls configs | grep "imx8q.*android.*"
```

8.6 Trusty OS/security configuration

Trusty OS firmware is used in i.MX Android 10 release as TEE, which supports security features.

The i.MX Trusty OS is based on the AOSP Trusty OS and supports for i.MX 8M Mini EVK, i.MX 8M Quad EVK, i.MX 8QuadMax MEK, and i.MX 8QuadXplus MEK Board. This section provides some basic configurations to make Trusty OS work on EVK/MEK boards. For more configurations about security related features, see the *i.MX Android Security User's Guide* (ASUG).

Customers can modify the Trusty OS code to make different configurations and enable different features. Firs, use the following commands to fetch code and build the target Trusty OS binary.

```
    # firslty create a directory for Trusty OS code and enter into this directory
   $ repo init -u https://source.codeaurora.org/external/imx/imx-manifest.git -b imx-android-10 -m imx-
trusty-android-10.0.0_2.0.0.xml
    $ repo sync
    $ source trusty/vendor/google/aosp/scripts/envsetup.sh
    $ make imx8mm #i.MX 8M Mini EVK Board
    $ cp ${TRUSTY_REPO_ROOT}/build-imx8mm/lk.bin ${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-
firmware/imx8m/tee-imx8mm.bin
```

Then, build the images, and the `tee-imx8mm.bin` will be integrated into `u-boot-imx8mm-trusty.imx`, `u-boot-imx8mm-trusty-secure-unlock.imx`, and `bootloader-imx8mm-trusty-dual.img`.

Flash the `u-boot-imx8mm-trusty.imx` file to the target device.

NOTE

- For i.MX 8M Nano EVK, it use the same Trusty target as i.MX 8M Mini EVK. Use `make imx8mm` to build the Trusty OS image, and copy the file `lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/tee-imx8mn.bin`.
- For i.MX 8MQuad EVK, use `make imx8m` to build the Trusty OS image, and copy the final `lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8m/tee-imx8mq.bin`.
- For i.MX 8QuadMax MEK, use `make imx8qm` to build the Trusty OS image, and copy the final `lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/tee-imx8qm.bin`.
- For i.MX 8QuadXPlus MEK, use `make imx8qxp` to build the Trusty OS image, and copy the final `lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/tee-imx8qx.bin`.
- `${TRUSTY_REPO_ROOT}` is the root directory of the Trusty OS codebase.
- `${MY_ANDROID}` is the root directory of the Android 10 codebase.

8.6.1 Initializing the secure storage for Trusty OS

Trusty OS uses the secure storage to protect userdata. This secure storage is baed on RPMB on the eMMC chip. RPMB needs to be initialized with a key, and default execution flow of images does not make this initialization.

Initialize the RPMB with a specified key or random key are both supported. Note that the RPMB key cannot be changed once it is set.

- To set a **specified** key, perform the following steps:

  Make your board enter fastboot mode, enter the following commands on the host side:

  — `fastboot stage < path-to-your-rpmb-key >`

  — `fastboot oem set-rpmb-key`

  After the board is rebooted, the RPMB service in Trusty OS is initialized successfully.

NOTE

— The RPMB key should start with magic "RPMB" and be followed with 32 bytes hexadecimal key.

— A prebuilt `rpmb_key_test.bin` whose key is fixed 32 bytes hexadecimal 0x00 is provided. It is generated with the following shell commands:

◦ touch rpmb_key.bin

◦ echo -n "RPMB" > rpmb_key.bin

◦ echo -n -e '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' >> rpmb_key.bin

The '\xHH' means eight-bit character whose value is the hexadecimal value 'HH'. You can replace "00" above with the key you want to set.

- To set a **random** key, perform the following steps:

  Make your board enter fastboot mode, enter the following commands on the host side:

  — fastboot oem set-rpmb-random-key

  After the board is rebooted, the RPMB service in Trusty OS is initialized successfully.

NOTE

The random key is generated on device and is invisible to anyone. Your device may no longer boot up if the RPMB key message was destroyed.

8.6.2 Provisioning the AVB key

The AVB key consists of a public key and a private key. The private key is used by the host to sign the vbmeta image, and the public key is used by AVB to authenticate the vbmeta image. The following figure shows the relationships between the AVB key and vbmeta. Without Trusty OS, the public key is hard-coded in U-Boot. With Trusty OS, it is saved in the secure storage.

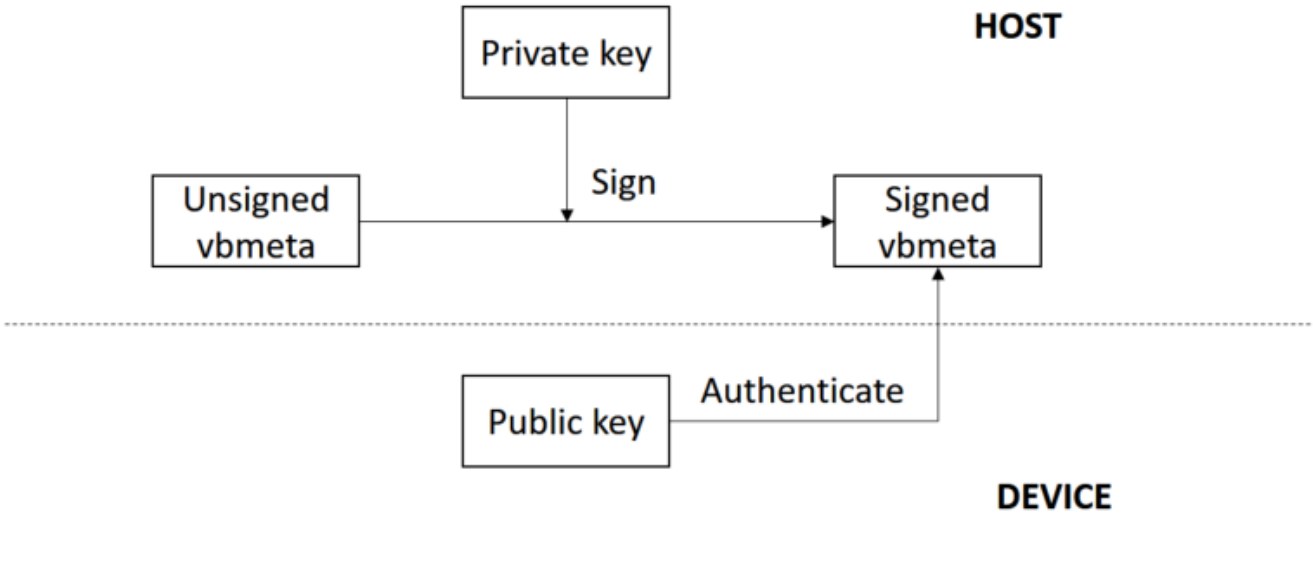

Figure 1. Relationship between AVB key and vbmeta

8.6.2.1 Generating the AVB key to sign images

The OpenSSL provides some commands to generate the private key. For example, you can use the following commands to generate the RSA-4096 private key test_rsa4096_private.pem:

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform PEM -out test_rsa4096_private.pem
```

The public key can be extracted from the private key. The avbtool in `${MY_ANDROID}/external/avb` supports such commands. You can get the public key `test_rsa4096_public.bin` with the commands:

```
avbtool extract_public_key --key test_rsa4096_private.pem --output test_rsa4096_public.bin
```

By default, the Android build system uses the algorithm SHA256_RSA4096 with the private key from `${MY_ANDROID}/external/avb/test/data/testkey_rsa4096.pem`. This can be overwritten by setting the `BOARD_AVB_ALGORITHM` and `BOARD_AVB_KEY_PATH` to use different algorithm and private key:

```
BOARD_AVB_ALGORITHM := <algorithm-type>
BOARD_AVB_KEY_PATH := <key-path>
```

Algorithm SHA256_RSA4096 is recommended, so Cryptographic Acceleration and Assurance Module (CAAM) can help accelerate the hash calculation. The Android build system will sign the vbmeta image with the private key above and will store one copy of the public key in the signed vbmeta image. During AVB verify, the U-Boot will validate the public key first and then use the public key to authenticate the signed vbmeta image.

8.6.2.2 Storing the AVB public key to a secure storage

The public key must be stored in the Trusty OS backed RPMB for Android if Trusty OS is enabled. Perform the following steps to set the public key.

Make your board enter fastboot mode and enter the following commands on the host side:

```
fastboot stage ${your-key-directory}/test_rsa4096_public.bin
fastboot oem set-public-key
```

The public key test_rsa4096_public.bin should be extracted from the private key you have specified. But if you do not specify any private key, you should set the public key as prebuilt testkey_public_rsa4096.bin, which is extracted to form the default private key testkey_rsa4096.pem.

8.6.3 Key attestation

The keystore key attestation aims to provide a way to strongly determine if an asymmetric key pair is hardware-backed, what the properties of the key are, and what constraints are applied to its usage.

Google provides the attestation "keybox" that contains private keys (RSA and ECDSA) and the corresponding certificate chains to partners from the Android Partner Front End (APFE). After retrieving the "keybox" from Google, you need to parse the "keybox" and provision the keys and certificates to secure storage. Both keys and certificates should be **Distinguished Encoding Rules (DER)** encoded.

Fastboot commands are provided to provision the attestation keys and certificates. Make sure that the secure storage is properly initialized for Trusty OS:

- Set RSA private key:

  ```
  fastboot stage < path-to-rsa-private-key >
  fastboot oem set-rsa-atte-key
  ```

- Set ECDSA private key:

  ```
  fastboot stage < path-to-ecdsa-private-key >
  fastboot oem set-ec-atte-key
  ```

- Append RSA certificate chain:

  ```
  fastboot stage < path-to-rsa-atte-cert >
  fastboot oem append-rsa-atte-cert
  ```

  Note that this command may need to be executed multiple times to append the whole certificate chain.

- Append ECDSA certificate chain:

  ```
  fastboot stage < path-to-ecdsa-cert >
  fastboot oem append-ec-atte-cert
  ```

  Note that this command may need to be executed multiple times to append the whole certificate chain.

After provisioning all the keys and certificates, the keystore attestation feature should work properly.

Besides, secure provision provides a way to prevent the plaintext attestation keys and certificates from exposure. For more details, see the *i.MX Android Security User's Guide* (ASUG).

8.7 SCFW configuration

SCFW is a binary stored in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware`, built into bootloader.

To customize SCFW, you need to download the SCFW porting kit on the i.MX Software and Development Tools page. For this release, choose **Linux -> Linux 5.4.3_1.0.0 -> SCFW Porting Kit** to download the porting kit. Then decompress the file with the following commands:

```
tar -zxvf imx-scfw-porting-kit-1.2.10.1.tar.gz
cd packages
chmod a+x imx-scfw-porting-kit-1.2.10.1.bin
./imx-scfw-porting-kit-1.2.10.1.bin
cd imx-scfw-porting-kit-1.2.10.1/src
tar -zxvf scfw_export_mx8qm_b0.tar.gz       # for i.MX 8QuadMax MEK
tar -zxvf scfw_export_mx8qx_b0.tar.gz       # for i.MX 8QuadXPlus MEK
```

The SCFW porting kit contains prebuilt binaries, libraries, and configuration files. For the board configuration file, take i.MX 8QuadXPlus MEK as an example, it is `scfw_export_mx8qx_b0/platform/board/mx8qx_mek/board.c`. Based on this file, some changes are made for Android and the file is stored in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q/board-imx8qxp.c`.

You can copy `board.c in vendor/nxp/fsl-proprietary` to SCFW porting kit, modify it, and then build the SCFW.

The following are steps to build Android SCFW (taking i.MX 8QuadXPlus as example):

1. Download GCC tool from the arm Developer GNU-RM Downloads page. It is suggested to download the version of "6-2017-q2-update" as it is verified.
2. Unzip the GCC tool to /opt/scfw_gcc.
3. Export `TOOLS="/opt/scfw-gcc"`.
4. Copy the board configuration file from `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q/board-imx8qxp.c` to porting kit.

   ```
   cp ${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q/board-imx8qxp.c
   scfw_export_mx8qx_b0/platform/board/mx8qx_mek/board.c
   ```

5. Build SCFW.

   ```
   cd scfw_export_mx8qx_b0     # enter the directory just uncompressed for i.MX 8QuadXPlus MEK
   make clean
   make qx R=B0 B=mek
   ```

6. Copy the SCFW binary to the uboot-firmware folder.

   ```
   cp build_mx8qx_b0/scfw_tcm.bin ${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q/
   mx8qx-scfw-tcm.bin
   ```

7. Build the bootloader.

   ```
   cd ${MY_ANDROID}
   ./imx-make.sh bootloader -j4
   ```

NOTE

To build SCFW for i.MX 8QuadMax MEK, use "qm" to replace "qx" in the steps above.

8.8 Miscellaneous configurations

8.8.1 Changing the boot command line in boot.img

After using boot.img, we store the default kernel boot command line inside this image. It will package together during Android build.

You can change this by changing the value of `BOARD_KERNEL_CMDLINE` in the `BoardConfig.mk` file under `${MY_ANDROID}/device/fsl`.

NOTE

- For i.MX 8M Mini EVK Board, the source folder is `${MY_ANDROID}/device/fsl/imx8m/evk_8mm/BoardConfig.mk`.
- For i.MX 8M Nano EVK Board, source folder is `${MY_ANDROID}/device/fsl/imx8m/evk_8mn/BoardConfig.mk`.
- For i.MX 8M Quad EVK Board, the source folder is `${MY_ANDROID}/device/fsl/imx8m/evk_8mq/BoardConfig.mk`.
- For i.MX 8QuadMax/8QuadXPlus MEK, the source folder is `${MY_ANDROID}/device/fsl/imx8q/mek_8q/BoardConfig.mk`.

8.8.2 Modifying the super partition

The partition of super is used to hold logical partitions. The following figure shows the dynamic partitions with a/b slot feature. The size of the logical partition is dynamically determined by the size of the raw image file. The remaining space is used together by the partitions in a group, so it can reduce the condition that some partitions cannot hold the new image while other partitions have a lot of unused space in OTA.

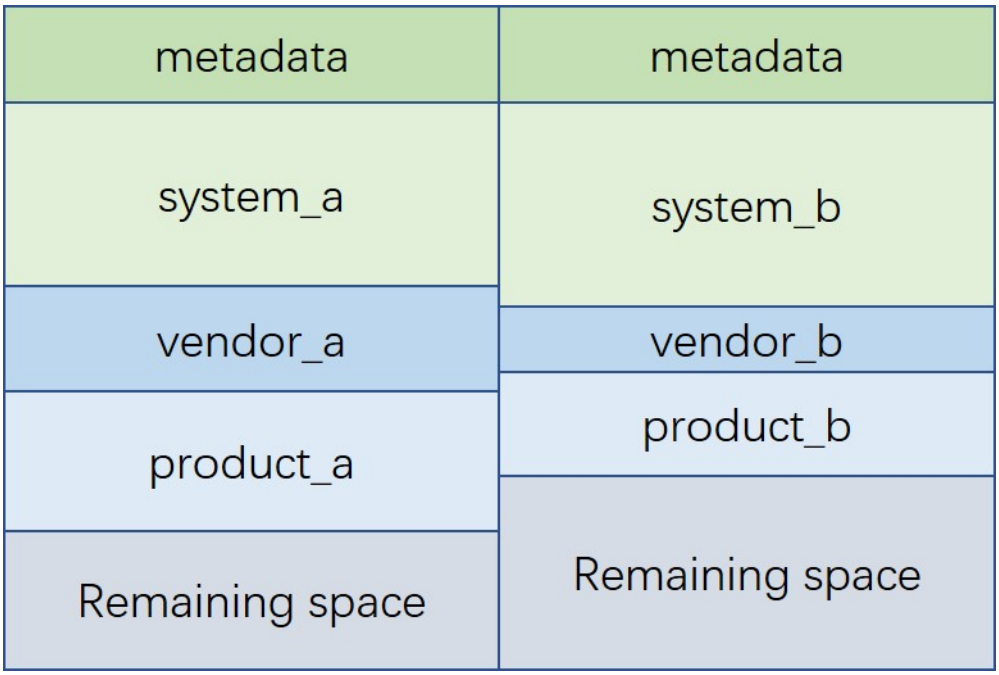

Figure 2. Brief schematic structure in super partition

Now the size of the super partition is 7 GB, 3.5 GB for each slot, and 10 MB reserved in this 3.5 GB for metadata. You can find code as follows in `${MY_ANDROID}/device/fsl`:

```
BOARD_SUPER_PARTITION_SIZE := 7516192768
BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 3747610624
```

Refer to the following patch to change the super partition size to 8 GB:

```
diff --git a/common/partition/device-partitions-13GB-ab_super.bpt b/common/partition/device-
partitions-13GB-ab_super.bpt
index e6e7f1a..829821c 100644
```

```
--- a/common/partition/device-partitions-13GB-ab_super.bpt
+++ b/common/partition/device-partitions-13GB-ab_super.bpt
    @@ -39,7 +39,7 @@
             },
             {
                 "label": "super",
    -            "size": "7168 MiB",
    +            "size": "8192 MiB",
                 "guid": "auto",
                 "type_guid": "c1dedb9a-a0d3-42e4-b74d-0acf96833624"
             },
    diff --git a/imx8m/BoardConfigCommon.mk b/imx8m/BoardConfigCommon.mk
    index 20d65a3..ae42220 100644
    --- a/imx8m/BoardConfigCommon.mk
    +++ b/imx8m/BoardConfigCommon.mk
    @@ -135,8 +135,8 @@ ifeq ($(TARGET_USE_DYNAMIC_PARTITIONS),true)
           BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4024434688
         endif
       else
    -    BOARD_SUPER_PARTITION_SIZE := 7516192768
    -    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 3747610624
    +    BOARD_SUPER_PARTITION_SIZE := 8589934592
    +    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4284481536
       endif
       ifeq ($(IMX_NO_PRODUCT_PARTITION),true)
         BOARD_NXP_DYNAMIC_PARTITIONS_PARTITION_LIST := system vendor
    diff --git a/imx8q/BoardConfigCommon.mk b/imx8q/BoardConfigCommon.mk
    index 85d3561..c7352a2 100644
    --- a/imx8q/BoardConfigCommon.mk
    +++ b/imx8q/BoardConfigCommon.mk
    @@ -164,8 +164,8 @@ ifeq ($(TARGET_USE_DYNAMIC_PARTITIONS),true)
           BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4024434688
         endif
       else
    -    BOARD_SUPER_PARTITION_SIZE := 7516192768
    -    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 3747610624
    +    BOARD_SUPER_PARTITION_SIZE := 8589934592
    +    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4284481536
       endif
       ifeq ($(IMX_NO_PRODUCT_PARTITION),true)
         BOARD_NXP_DYNAMIC_PARTITIONS_PARTITION_LIST := system vendor
```

You may also need to change the flash script, which can generate `super.img` when flashing images. The flash scripts include `fastboot_imx_flashall.sh`, `fastboot_imx_flashall.bat`, `uuu_imx_android_flash.sh`, `uuu_imx_android_flash.bat`, and `fsl-sdcard-partition.sh`. The following is an example on the `uuu_imx_android_flash` script:

```
    diff --git a/common/tools/uuu_imx_android_flash.bat b/common/tools/uuu_imx_android_flash.bat
    index 8eecb29..21af031 100755
    --- a/common/tools/uuu_imx_android_flash.bat
    +++ b/common/tools/uuu_imx_android_flash.bat
    @@ -885,8 +885,8 @@ if %support_dualslot% == 1 (
             set lpmake_vendor_image_b=--image vendor_b=%image_directory%%vendor_file%
             set lpmake_product_image_b=--image product_b=%image_directory%%product_file%
         )
    -    %image_directory%lpmake.exe --metadata-size 65536 --super-name super --metadata-slots 3 --
device super:7516192768 ^
    -        --group nxp_dynamic_partitions_a:3747610624 --group nxp_dynamic_partitions_b:3747610624 ^
    +    %image_directory%lpmake.exe --metadata-size 65536 --super-name super --metadata-slots 3 --
device super:8589934592 ^
```

```
    +        --group nxp_dynamic_partitions_a:4284481536 --group nxp_dynamic_partitions_b:4284481536 ^
             --partition system_a:readonly:0:nxp_dynamic_partitions_a !lpmake_system_image_a! ^
             --partition system_b:readonly:0:nxp_dynamic_partitions_b !lpmake_system_image_b! ^
             --partition vendor_a:readonly:0:nxp_dynamic_partitions_a !lpmake_vendor_image_a! ^
    diff --git a/common/tools/uuu_imx_android_flash.sh b/common/tools/uuu_imx_android_flash.sh
    index 7872b7a..bfbf501 100755
    --- a/common/tools/uuu_imx_android_flash.sh
    +++ b/common/tools/uuu_imx_android_flash.sh
    @@ -324,8 +324,8 @@ function make_super_image
             fi
         fi

    -        ${sym_link_directory}lpmake --metadata-size 65536 --super-name super --metadata-slots 3 --
device super:7516192768 \
    -          --group nxp_dynamic_partitions_a:3747610624 --group nxp_dynamic_partitions_b:3747610624
\
    +        ${sym_link_directory}lpmake --metadata-size 65536 --super-name super --metadata-slots 3 --
device super:8589934592 \
    +          --group nxp_dynamic_partitions_a:4284481536 --group nxp_dynamic_partitions_b:4284481536
\
                --partition system_a:readonly:0:nxp_dynamic_partitions_a ${lpmake_system_image_a} \
                --partition system_b:readonly:0:nxp_dynamic_partitions_b ${lpmake_system_image_b} \
                --partition vendor_a:readonly:0:nxp_dynamic_partitions_a ${lpmake_vendor_image_a} \
```

9 Revision History

Table 23. Revision history

Revision number	Date	Substantive changes
P9.0.0_1.0.0-beta	11/2018	Initial release
P9.0.0_1.0.0-ga	01/2019	i.MX 8M, i.MX 8QuadMax, i.MX 8QuadXPlus GA release.
P9.0.0_2.0.0-ga	04/2019	i.MX 8M, i.MX 8QuadMax, i.MX 8QuadXPlus GA release.
P9.0.0_2.0.0-ga	08/2019	Updated the location of the SCFW porting kit.
android-10.0.0_1.0.0	02/2020	i.MX 8M Mini, i.MX 8M Quad, i.MX 8QuadMax, and i.MX 8QuadXPlus GA release.
android-10.0.0_1.0.0	03/2020	Deleted the Android 10 image.
android-10.0.0_2.0.0	05/2020	i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Quad, i.MX 8QuadMax, and i.MX 8QuadXPlus GA release.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, UMEMS, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 20 May 2020

Document identifier: AUG