AUG

Android User's Guide

Rev. android-12.0.0_2.0.0 — 28 July 2022

User guide

Document information

Information	Content
Keywords	Android, i.MX, android-12.0.0_2.0.0
Abstract	This document provides the technical information related to the i.MX 8 series devices.

1 Overview

This document provides the technical information related to 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 and 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 18.04 64-bit version is the most tested environment for the Android 12.0 build.

To synchronize the code and build images of this release, the computer should at least have:

- 450 GB free disk space
- 16 GB RAM

Note:

The minimum required amount of free memory is around 16GB, and even with that, some configurations may not work.

Enlarge the physical RAM capacity is a way to avoid potential build errors related to memory.

With 16GB RAM, if you run into segfaults or other errors related to memory when build the images, try reducing your `-j` value. In the demonstration commands in the following part of this document, the `-j` value is 4.

After installing the computer running Linux OS, check whether all the necessary packages are installed for an Android build. See "Establishing a Build Environment" 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 \
                     zlib1g-dev liblz-dev \
                     liblzo2-2 liblzo2-dev \
                     lzop \
                     git curl \
                     u-boot-tools \
                     mtd-utils \
                     android-sdk-libsparse-utils android-sdk-
ext4-utils \
                     device-tree-compiler \
                     gdisk \
                     m4 \
                     zlib1g-dev \
                     bison \
                     flex make \
                     libssl-dev \
                     gcc-multilib \
                     libghc-gnutls-dev \
                     swig \
                     libdw-dev \
                     dwarves
```

Note:

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 set up a computer running Linux OS, unpack the Android release package by using the following command:

```
$ cd ~ (or any other directory you like)
$ tar xzvf imx-android-12.0.0_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-12.0.0_2.0.0.tar.gz` under the `~/.` 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-12.0.0_2.0.0/imx_android_setup.sh
# By default, after preceding command finishes execution,
 current working directory changed to the i.MX Android source
 code root directory.
# ${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`
```

Note:

In the `imx_android_setup.sh` script, a `.xml` file that contains the code repository information is specified. To make the code synchronized by this script the same as the release state, code repository revision is specified with the release tag in this file. The release tag is static and is not moved after the code is published, so no matter when `imx_android_setup.sh` is executed, the working area of the code repositories synchronized by this script are the same as release state and images being built are the same as prebuilt images.

If a critical issue bugfix is published, another `.xml` file is published to reflect those changes on the source code. Then customers need to modify the `imx_android_setup.sh`. For this release, make the following changes on the script.

```
diff --git a/imx_android_setup.sh b/imx_android_setup.sh
index 324ec67..4618679 100644
--- a/imx_android_setup.sh
+++ b/imx_android_setup.sh
@@ -26,7 +26,7 @@ if [ ! -d "$android_builddir" ]; then
     # Create android build dir if it does not exist.
     mkdir "$android_builddir"
     cd "$android_builddir"
-    repo init -u https://source.codeaurora.org/
external/imx/imx-manifest.git -b imx-android-12 -m imx-
android-12.0.0_2.0.0.xml
+    repo init -u https://source.codeaurora.org/
external/imx/imx-manifest.git -b imx-android-12 -m
 rel_android-12.0.0_2.0.0.xml
       rc=$?
       if [ "$rc" != 0 ]; then
          echo
 "---------------------------------------------------"
```

The wireless-regdb repository may fail to be synchronized with the following log:

```
fatal: unable to access 'https://git.kernel.org/pub/
scm/linux/kernel/git/sforshee/wireless-regdb/': server
 certificate verification failed. CAfile: /etc/ssl/certs/ca-
certificates.crt CRLfile: none
```

If this issue is encountered, execute the following command on the host as a fix:

```
$ git config --global http.sslVerify false
```

3.2 Building Android images

The Android image can be built after the source code has been downloaded (Section 3.1 Section 3.1).

This section provides an overview of how to use Android build system and what NXP did on it. Then it provides an example of how to build Android images for a specific board as well as preparation steps. Customers could follow these steps to do the preparation work and build the images.

First, the source build/envsetup.sh command is executed to import shell functions that are defined in ${MY_ANDROID}/build/envsetup.sh.

Then, the lunch <ProductName-BuildMode> command is executed to set up the build configuration.

The "Product Name" is the Android device name found in directory ${MY_ANDROID}/device/nxp/. Search for the keyword PRODUCT_NAME under this directory for the product names. The following table lists the i.MX product names.

Table 1. i.MX product names

Product name	Description
evk_8mm	i.MX 8M Mini EVK Board
evk_8mn	i.MX 8M Nano EVK Board
evk_8mp	i.MX 8M Plus EVK Board
evk_8mq	i.MX 8M Quad EVK Board
evk_8ulp	i.MX 8ULP 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 mode

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

This lunch command can be executed with an argument of ProductName-BuildMode, such as lunch evk_8mm-userdebug. It can also be issued without the argument and a menu presents for choosing a target.

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 behavior of the i.MX Android build system used to be aligned with the original Android platform. The make command can start the build process and all images are 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 Android 10.0. 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 the 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 is compiled, so customers can still build the i.MX Android images with only one command with this script.

i.MX Android platform needs some preparation for the first time when building the images. The image build steps are as follows:

1. Prepare the build environment for U-Boot and Linux kernel.
   This step is mandatory because there is no GCC cross-compile tool chain in the one in AOSP codebase.
   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](https://developer.arm.com) page. It is recommended to use the 9.2 version for this release. You can download `gcc-arm-9.2-2019.12-x86_64-aarch64-none-elf.tar.xz` or `gcc-arm-9.2-2019.12-x86_64-aarch64-none-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-9.2-2019.12-x86_64-aarch64-none-elf.tar.xz"
    is used
   $ sudo tar -xvJf gcc-arm-9.2-2019.12-x86_64-aarch64-none-
   elf.tar.xz -C /opt
   $ export AARCH64_GCC_CROSS_COMPILE=/opt/gcc-
   arm-9.2-2019.12-x86_64-aarch64-none-elf/bin/aarch64-none-
   elf-
   # if "gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-
   gnu.tar.xz" is used
   $ sudo tar -xvJf gcc-arm-9.2-2019.12-x86_64-aarch64-none-
   linux-gnu.tar.xz -C /opt
   $ export AARCH64_GCC_CROSS_COMPILE=/opt/gcc-
   arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu/bin/aarch64-
   none-linux-gnu-
   ```

   c. Follow below steps to set external clang tools for kernel building.

   ```
   sudo git clone https://android.googlesource.com/platform/
   prebuilts/clang/host/linux-x86 /opt/prebuilt-android-clang
   cd /opt/prebuilt-android-clang
   sudo git checkout 0fc0715d9392ca616605c07750211d7ca71f4e36
   export CLANG_PATH=/opt/prebuilt-android-clang
   ```

   The preceding export commands can be added to `/etc/profile`. When the host boots up, `AARCH64_GCC_CROSS_COMPILE` and `CLANG_PATH` are 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:

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

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

To change `BUILD_ID` and `BUILD_NUMBER`, update `build_id.mk` in the `${MY_ANDROID}/device/nxp/` 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, it is used to generate system.img together with files in `system/`.
- `system/`: Android system binary/libraries. It is used to generate `system.img` together with files in `root/`.
- `recovery/`: root file system, integrated into `vendor_boot.img` as a part of the ramdisk and used by the Linux kernel when the system boots up.
- `vendor_ramdisk/`: Integrated into `vendor_boot.img` as part of the ramdisk and used by the Linux kernel when the system boots up.
- `ramdisk/`: integrated into boot image as part of the ramdisk and used by linux kernel when system boot up. Because GKI is enabled on i.MX 8M Mini EVK, this is integrated into the `boot-imx.img`.
- `ramdisk.img`: Ramdisk image generated from `ramdisk/`. Not directly used.
- `dtbo-imx8mm.img`: Board's device tree binary. It is used to support MIPI-to-HDMI output on the i.MX 8M Mini EVK LPDDR4 board.
- `dtbo-imx8mm-m4.img`: Board's device tree binary. It is used to support MIPI-to-HDMI output and audio playback based on Cortex-M4 FreeRTOS on the i.MX 8M Mini EVK LPDDR4 board.
- `dtbo-imx8mm-mipi-panel.img`: Board's device tree binary. It is used to support rm67199 MIPI Panel output on the i.MX 8M Mini EVK LPDDR4 board.
- `dtbo-imx8mm-mipi-panel-rm67191.img`: Board's device tree binary. It is used to support rm67191 MIPI Panel output on the i.MX 8M Mini EVK LPDDR4 board.
- `dtbo-imx8mm-ddr4.img`: Board's device tree binary. It is used to support MIPI-to-HDMI output on the i.MX 8M Mini EVK DDR4 board.
- `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`.
- `vbmeta-imx8mm-mipi-panel-rm67191.img`: Android Verify boot metadata image for `dtbo-imx8mm-mipi-panel-rm67191.img`.

- `vbmeta-imx8mm-ddr4.img`: Android Verify boot metadata image for `dtbo-imx8mm-ddr4.img`.
- `system.img`: EXT4 image generated from `system/` and `root/`.
- `system_ext.img`: EXT4 image generated from `system_ext/`.
- `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 LPDDR4 board.
- `u-boot-imx8mm-trusty.imx`: U-Boot image with Trusty OS integrated for i.MX 8M Mini EVK LPDDR4 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 LPDDR4 board.
- `u-boot-imx8mm-evk-uuu.imx`: U-Boot image used by UUU for i.MX 8M Mini EVK LPDDR4 board. It is not flashed to MMC.
- `u-boot-imx8mm-ddr4.imx`: U-Boot image for i.MX 8M Mini EVK DDR4 board.
- `u-boot-imx8mm-ddr4-evk-uuu.imx`: U-Boot image used by UUU for i.MX 8M Mini EVK DDR4 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, which is generated with `system.img`, `system_ext.img`, `vendor.img`, and `product.img`.
- `boot.img`: a composite image which includes the AOSP generic kernel Image, generic ramdisk and boot parameters.
- `boot-imx.img`: a composite image which includes the kernel Image built from i.MX Kernel tree, generic ramdisk and boot parameters.
- `vendor_boot.img`: A composite image, which includes vendor 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 Section 3.3.*
- *To build the kernel uImage separately, see Section 3.4.*
- *To build `boot.img`, see Section 3.5.*
- *To build `dtbo.img`, see Section 3.6.*

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

Board name	Lunch command
i.MX 8M Mini EVK board	`$ lunch evk_8mm-userdebug`
i.MX 8M Nano EVK board	`$ lunch evk_8mn-userdebug`
i.MX 8M Plus EVK board	`$ lunch evk_8mp-userdebug`
i.MX 8M Quad EVK board	`$ lunch evk_8mq-userdebug`
i.MX 8ULP EVK Board	`$ lunch evk_8ulp-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.

`PRODUCT_PACKAGES_ENG`, `PRODUCT_PACKAGES_DEBUG` and `PRODUCT_PACKAGES` can be used to specify the modules to be installed in the appropriate product makefiles.

The modules specified by `PRODUCT_PACKAGES` are always installed. For the effect of `PRODUCT_PACKAGES_ENG` and `PRODUCT_PACKAGES_DEBUG`, check the description below.

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

- `eng`: development configuration with additional debugging tools
  - Installs modules specified by `PRODUCT_PACKAGES_ENG` and/or `PRODUCT_PACKAGES_DEBUG`.
  - Installs modules according to the product definition files.
  - `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.
  - `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 specified by `PRODUCT_PACKAGES_DEBUG`.
  - Installs modules according to the product definition files.
  - `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
```

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 the `${MY_ANDROID}/vendor/partner_gms` folder. Make sure that the `product.mk` file has the following line:

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

Then build the images. The GMS package is then 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/nxp/imx8m/evk_8mm/evk_8mm.mk`.

3.3 Building U-Boot images

The U-Boot images can be generated separately. For example, you can generate a U-Boot image for i.MX 8M Mini EVK as follows:

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

For other platform, use **lunch** `<ProductName-BuildMode>` to set up the build configuration. For detailed build configuration, see Section 3.2. Multiple U-Boot variants are generated for different purposes. For more details, check `{MY_Android}/device/nxp/{MY_PLATFORM}/{MY_PRODUCT}/UbootKernelBoardConfig.mk`. The following table lists the U-Boot configurations and images for i.MX 8M Mini EVK.

Table 4. U-Boot configurations and images for i.MX 8M Mini EVK

SoC	U-Boot configurations	Generated images	Description
i.MX 8M Mini	`imx8mm_evk_android_defconfig`	`u-boot-imx8mm.imx`	Default i.MX 8M Mini U-Boot image if trusty is not enabled.
i.MX 8M Mini	`imx8mm_evk_android_dual_defconfig`	`spl-imx8mm-dual.bin`, `bootloader-imx8mm-dual.img`	i.MX 8M Mini U-Boot image with dual-bootloader feature enabled.
i.MX 8M Mini	`imx8mm_evk_android_trusty_defconfig`	`u-boot-imx8mm-trusty.imx`	Default i.MX 8M Mini U-Boot image if trusty is enabled.
i.MX 8M Mini	`imx8mm_evk_android_trusty_secure_unlock_defconfig`	`u-boot-imx8mm-trusty-secure-unlock.imx`	i.MX 8M Mini U-Boot image with trusty and secure unlock feature enabled.
i.MX 8M Mini	`imx8mm_evk_android_trusty_dual_defconfig`	`spl-imx8mm-trusty-dual.bin`, `bootloader-imx8mm-trusty-dual.img`	i.MX 8M Mini U-Boot image with trusty and dual-bootloader feature enabled.
i.MX 8M Mini	`imx8mm_ddr4_evk_android_defconfig`	`u-boot-imx8mm-ddr4.imx`	i.MX 8M Mini U-Boot image with DDR4 DRAM chip.
i.MX 8M Mini	`imx8mm_evk_android_uuu_defconfig`	`u-boot-imx8mm-evk-uuu.imx`	U-Boot image meant for flashing images for i.MX 8M Mini EVK. It should not be shipped to end users.
i.MX 8M Mini	`imx8mm_ddr4_evk_android_uuu_defconfig`	`u-boot-imx8mm-ddr4-evk-uuu.imx`	U-Boot image meant for flashing images for i.MX 8M Mini EVK with DDR4 DRAM chip. It should not be shipped to end users.

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

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 <ProductName-buildMode>` to set up the build configuration. For detailed build configuration, see Section Section 3.2.

3.6 Building dtbo.img

DTBO image holds the device tree binary of the board.

The following commands are 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 <ProductName-buildMode>` to set up the build configuration. For detailed build configuration, see Section Section 3.2.

4 Running the Android Platform with a Prebuilt Image

To test the Android platform before building any code, use the prebuilt images from the following packages and go to Section 5 and Section 6.

Table 5. Image packages

Image package	Description
`android-12.0.0_2.0.0_image_8mmevk.tar.gz`	Prebuilt-image for i.MX 8M Mini EVK LPDDR4 board, which includes NXP extended features.
`android-12.0.0_2.0.0_image_8mnevk.tar.gz`	Prebuilt-image for i.MX 8M Nano EVK board, which includes NXP extended features.

Table 5. Image packages*...continued*

Image package	Description
android-12.0.0_2.0.0_image_8mpevk.tar.gz	Prebuilt-image for i.MX 8M Plus EVK board, which includes NXP extended features.
android-12.0.0_2.0.0_image_8mqevk.tar.gz	Prebuilt-image for i.MX 8M Quad EVK board, which includes NXP extended features.
android-12.0.0_2.0.0_image_8ulpevk.tar.gz	Prebuilt-image for i.MX 8ULP EVK board, which includes NXP extended features.

The following tables list the detailed contents of the `android-12.0.0_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 LPDDR4 board.
spl-imx8mm-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i.MX 8M Mini EVK LPDDR4 board.
bootloader-imx8mm-dual.img	An image containing U-Boot proper and ATF. It is for i.MX 8M Mini EVK LPDDR4 board.
bootloader-imx8mm-trusty-dual.img	An image containing U-Boot proper, ATF, and Trusty OS. It is for i.MX 8M Mini EVK LPDDR4 board.
u-boot-imx8mm.imx	An image containing U-Boot and ATF for i.MX 8M Mini EVK LPDDR4 board.
u-boot-imx8mm-trusty.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Mini EVK LPDDR4 board.
u-boot-imx8mm-trusty-secure-unlock.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Mini EVK LPDDR4 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 EVK LPDDR4 board. It is not flashed to MMC.
u-boot-imx8mm-ddr4.imx	An image containing U-Boot and ATF for i.MX 8M Mini EVK DDR4 board.
u-boot-imx8mm-ddr4-evk-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8M Mini EVK DDR4 board. It is not flashed to MMC.
boot.img	Boot image for i.MX 8M Mini EVK board, it contains the AOSP generic kernel image, generic ramdisk and default kernel command line.
boot-imx.img	Boot image for i.MX 8M Mini EVK board, it contains the kernel image built from i.MX Kernel tree, generic ramdisk and default kernel command line.
vendor_boot.img	Vendor boot image for i.MX 8M Mini EVK board, it contains vendor ramdisk and default kernel command line.
system.img	System image for i.MX 8M Mini EVK board.
system_ext.img	System extension 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.

Table 6. Images for i.MX 8M Mini*...continued*

i.MX 8M Mini EVK Image	Description
`super.img`	Super image generated from `system.img`, `system_ext.img`, `vendor.img`, and `product.img`.
`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 MIPI-to-HDMI output on i.MX 8M Mini EVK LPDDR4 board.
`dtbo-imx8mm-m4.img`	Device Tree image for i.MX 8M Mini EVK board to support MIPI-to-HDMI output and audio playback based on Cortex-M4 Free RTOS i.MX 8M Mini EVK LPDDR4 board.
`dtbo-imx8mm-mipi-panel.img`	Device Tree image for i.MX 8M Mini EVK board to support RM67199 MIPI panel output i.MX 8M Mini EVK LPDDR4 board.
`dtbo-imx8mm-mipi-panel-rm67191.img`	Device Tree image for i.MX 8M Mini EVK board to support RM67191 MIPI panel output i.MX 8M Mini EVK LPDDR4 board.
`dtbo-imx8mm-ddr4.img`	Device Tree image for i.MX 8M Mini EVK board to support MIPI-to-HDMI output on i.MX 8M Mini EVK DDR4 board.
`vbmeta-imx8mm.img`	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support MIPI-to-HDMI output on i.MX 8M Mini EVK LPDDR4 board.
`vbmeta-imx8mm-m4.img`	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support MIPI-to-HDMI output and Cortex-M4 playback on i.MX 8M Mini EVK LPDDR4 board.
`vbmeta-imx8mm-mipi-panel.img`	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support RM67199 MIPI panel output on i.MX 8M Mini EVK DDR4 board.
`vbmeta-imx8mm-mipi-panel-rm67191.img`	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support RM67191 MIPI panel output on i.MX 8M Mini EVK DDR4 board.
`vbmeta-imx8mm-ddr4.img`	Android Verify Boot metadata image for i.MX 8M Mini EVK board to support MIPI-to-HDMI output on i.MX 8M Mini EVK DDR4 board.
`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 the `android-12.0.0_2.0.0_image_8mnevk.tar.gz` image package.

Table 7. Images for i.MX 8M Nano

i.MX 8M Nano EVK Image	Descriptions
spl-imx8mn-dual.bin	Secondary program loader image without Trusty related configurations for i.MX 8M Nano EVK LPDDR4 board.
spl-imx8mn-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i.MX 8M Nano EVK LPDDR4 board.
bootloader-imx8mn-dual.img	An image containing U-Boot proper and ATF. It is for i.MX 8M Nano EVK LPDDR4 board.
bootloader-imx8mn-trusty-dual.img	An image containing U-Boot proper, ATF, and Trusty OS. It is for i.MX 8M Nano EVK LPDDR4 board.
u-boot-imx8mn.imx	An image containing U-Boot and ATF for i.MX 8M Nano EVK LPDDR4 board.
u-boot-imx8mn-trusty.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Nano EVK LPDDR4 board.
u-boot-imx8mn-trusty-secure-unlock.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Nano EVK LPDDR4 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 EVK LPDDR4 board. It is not flashed to MMC.
u-boot-imx8mn-ddr4.imx	An image containing U-Boot and ATF for i.MX 8M Nano EVK DDR4 board.
u-boot-imx8mn-ddr4-evk-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8M Nano EVK DDR4 board. It is not flashed to MMC.
boot.img	Boot image for i.MX 8M Nano EVK board, it contains the AOSP generic kernel image, generic ramdisk and default kernel command line.
boot-imx.img	Boot image for i.MX 8M Nano EVK board, it contains the kernel image built from i.MX Kernel tree, generic ramdisk and default kernel command line.
vendor_boot.img	Vendor boot image for i.MX 8M Nano EVK board, it contains vendor ramdisk and default kernel command line.
system.img	System image for i.MX 8M Nano EVK board.
system_ext.img	System extension 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.
super.img	Super image generated from system.img, system_ext.img, vendor.img, and product.img.
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 7. Images for i.MX 8M Nano*...continued*

i.MX 8M Nano EVK Image	Descriptions
`dtbo-imx8mn.img`	Device Tree image for i.MX 8M Nano EVK LPDDR4 board to support MIPI-to-HDMI output.
`dtbo-imx8mn-rpmsg.img`	Device Tree image for i.MX 8M Nano EVK LPDDR4 board to support MIPI-to-HDMI output and MCU image.
`dtbo-imx8mn-mipi-panel.img`	Device Tree image for i.MX 8M Nano EVK LPDDR4 board to support RM67199 MIPI panel output.
`dtbo-imx8mn-mipi-panel-rm67191.img`	Device Tree image for i.MX 8M Nano EVK LPDDR4 board to support RM67191 MIPI panel output.
`dtbo-imx8mn-ddr4.img`	Device Tree image for i.MX 8M Nano EVK DDR4 board to support MIPI-to-HDMI output.
`dtbo-imx8mn-ddr4-rpmsg.img`	Device Tree image for i.MX 8M Nano EVK DDR4 board to support MIPI-to-HDMI output and MCU image.
`dtbo-imx8mn-ddr4-mipi-panel.img`	Device Tree image for i.MX 8M Nano EVK DDR4 board to support RM67199 MIPI panel output.
`dtbo-imx8mn-ddr4-mipi-panel-rm67199.img`	Device Tree image for i.MX 8M Nano EVK DDR4 board to support RM67191 MIPI panel output.
`vbmeta-imx8mn.img`	Device Tree image for i.MX 8M Nano EVK LPDDR4 board to support MIPI-to-HDMI output.
`vbmeta-imx8mn-rpmsg.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK LPDDR4 board to support MIPI-to-HDMI output and MCU image.
`vbmeta-imx8mn-mipi-panel.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK LPDDR4 board to support RM67199 MIPI panel output.
`vbmeta-imx8mn-mipi-panel-rm67191.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK LPDDR4 board to support RM67191 MIPI panel output.
`vbmeta-imx8mn-ddr4.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK DDR4 board to support MIPI-to-HDMI output.
`vbmeta-imx8mn-ddr4-rpmsg.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK DDR4 board to support MIPI-to-HDMI output and MCU image.
`vbmeta-imx8mn-ddr4-mipi-panel.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK DDR4 board to support RM67199 MIPI panel output.
`vbmeta-imx8mn-ddr4-mipi-panel-rm67191.img`	Android Verify Boot metadata image for i.MX 8M Nano EVK DDR4 board to support RM67191 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 the `android-12.0.0_2.0.0_image_8mpevk.tar.gz` image package.

Table 8. Images for i.MX 8M Plus

i.MX 8M Plus EVK Image	Description
spl-imx8mp-dual.bin	Secondary program loader image without Trusty related configurations for i.MX 8M Plus EVK board.
spl-imx8mp-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i. MX 8M Plus EVK board.
bootloader-imx8mp-dual.img	An image containing U-Boot proper and ATF. It is for i.MX 8M Plus EVK board.
bootloader-imx8mp-trusty-dual.img	An image containing U-Boot proper, ATF, and Trusty OS. It is for i.MX 8M Plus EVK board.
u-boot-imx8mp.imx	An image containing U-Boot and ATF for i.MX 8M Plus EVK board.
u-boot-imx8mp-trusty.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Plus EVK board.
u-boot-imx8mp-trusty-secure-unlock.imx	An image containing U-Boot, ATF, and Trusty OS for i.MX 8M Plus EVK board. It is a demonstration of secure unlock mechanism.
u-boot-imx8mp-evk-uuu.imx	An image containing U-Boot and ATF, used by UUU for i.MX 8M Plus board. It is not flashed to MMC.
boot.img	Boot image for i.MX 8MPlus EVK board, it contains the AOSP generic kernel image, generic ramdisk and default kernel command line.
boot-imx.img	Boot image for i.MX 8MPlus EVK board, it contains the kernel image built from i.MX Kernel tree, generic ramdisk and default kernel command line.
vendor_boot.img	Vendor boot image for i.MX 8MPlus EVK board, it contains vendor ramdisk and default kernel command line.
system.img	System image for i.MX 8M Plus EVK board.
system_ext.img	System extension image for i.MX 8M Plus EVK board.
vendor.img	Vendor image for i.MX 8M Plus EVK board.
product.img	Product image for i.MX 8M Plus EVK board.
super.img	Super image generated from system.img, system_ext.img, vendor.img, and product.img.
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 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.
imx8mp_mcu_demo.img	MCU image for i.MX 8M Plus EVK board.
dtbo-imx8mp.img	Device Tree image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support two basler cameras plug in CSI1 and CSI2 port.

Table 8. Images for i.MX 8M Plus*...continued*

i.MX 8M Plus EVK Image	Description
dtbo-imx8mp-basler-ov5640.img	Device Tree image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support basler camera plug in CSI1 port and OV5640 camera plug in CSI2 port.
dtbo-imx8mp-basler.img	Device Tree image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support only basler camera plug-in CSI1 slot.
dtbo-imx8mp-ov5640.img	Device Tree image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support only OV5640 camera plugin CSI1 slot
dtbo-imx8mp-lvds-panel.img	Device Tree image for i.MX 8M Plus EVK board to support LVDS panel output.
dtbo-imx8mp-lvds.img	Device Tree image for i.MX 8MPlus EVK board to support dual-channel LVDS to HDMI output.
dtbo-imx8mp-mipi-panel.img	Device Tree image for i.MX 8M Plus EVK board to support RM67199 MIPI panel output.
dtbo-imx8mp-mipi-panel-rm67191.img	Device Tree image for i.MX 8M Plus EVK board to support RM67191 MIPI panel output.
dtbo-imx8mp-rpmsg.img	Device Tree image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output and MCU image.
dtbo-imx8mp-sof.img	Device Tree image for i.MX 8MPlus EVK board to support the Sound Open Firmware audio output.
vbmeta-imx8mp.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support two basler cameras plug in CSI1 and CSI2 port..
vbmeta-imx8mp-basler-ov5640.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support basler camera plug in CSI1 port and OV5640 camera plug in CSI2 port.
vbmeta-imx8mp-basler.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support only basler camera plug-in CSI1 slot.
vbmeta-imx8mp-ov5640.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output, and support only OV5640 camera plug-in CSI1 slot.
vbmeta-imx8mp-lvds-panel.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support LVDS panel output.
vbmeta-imx8mp-lvds.img	Android Verify Boot metadata image for i.MX 8MPlus EVK board to support dual-channel LVDS to HDMI output.
vbmeta-imx8mp-mipi-panel.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support RM67199 MIPI panel output.
vbmeta-imx8mp-mipi-panel-rm67191.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support RM67191 MIPI panel output.
vbmeta-imx8mp-rpmsg.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support MIPI-to-HDMI output and MCU image.
vbmeta-imx8mp-sof.img	Android Verify Boot metadata image for i.MX 8M Plus EVK board to support the Sound Open Firmware audio output.

Table 8. Images for i.MX 8M Plus*...continued*

i.MX 8M Plus EVK Image	Description
`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 the default AVB private key.

The following tables list the detailed contents of the `android-12.0.0_2.0.0_image_8mqevk.tar.gz` image package.

Table 9. 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 8MQuad EVK board, it contains the AOSP generic kernel image, generic ramdisk and default kernel command line.
`boot-imx.img`	Boot image for i.MX 8MQuad EVK board, it contains the kernel image built from i.MX Kernel tree, generic ramdisk and default kernel command line.
`vendor_boot.img`	Vendor boot image for i.MX 8M Quad EVK board, it contains vendor ramdisk and default kernel command line.
`system.img`	System image for i.MX 8M Quad EVK board.
`system_ext.img`	System extension 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.
`super.img`	Super image generated from `system.img`, `system_ext.img`, `vendor.img`, and `product.img`.
`partition-table.img`	GPT partition table image for single bootloader condition. Used for 16 GB SD card and eMMC.

Table 9. 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 16 GB SD card and eMMC.
partition-table-28GB.img	GPT partition table image for 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.
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 RM67199 MIPI panel output.
dtbo-imx8mq-mipi-panel-rm67191.img	Device Tree image for i.MX 8M Quad EVK REV A board to support RM67191 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 RM67199 MIPI panel output.
vbmeta-imx8mq-mipi-panel-rm67199.img	Android Verify Boot metadata image for i.MX 8M Quad EVK REV A board to support RM67191 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 the default AVB private key.

The following tables list the detailed contents of the `android-12.0.0_1.0.0_image_8ulpevk.tar.gz` image package.

Table 10. Images for i.MX 8ULP EVK

i.MX 8ULP EVK Image	Description
spl-imx8ulp-dual.bin	Secondary program loader image without Trusty related configurations for i.MX 8ULP EVK board.
spl-imx8ulp-trusty-dual.bin	Secondary program loader image with Trusty related configurations for i.MX 8ULP EVK board.
bootloader-imx8ulp-dual.img	An image containing U-Boot proper and ATF for i.MX 8ULP EVK board.
bootloader-imx8ulp-trusty-dual.img	An image containing U-Boot proper, ATF, and Trusty OS for i.MX 8ULP EVK board.

Table 10. Images for i.MX 8ULP EVK*...continued*

i.MX 8ULP EVK Image	Description
`u-boot-imx8ulp-9x9.imx`	An image containing uboot and ATF for i.MX 8ULP EVK 9x9 board.
`u-boot-imx8ulp-trusty-9x9.imx`	An image containing uboot, ATF and Trusty OS for i.MX 8ULP EVK 9x9 board.
`u-boot-imx8ulp-9x9-evk-uuu.imx`	An image containing uboot and ATF, used by UUU for i.MX 8ULP EVK 9x9 board,it will not be flashed to MMC.
`u-boot-imx8ulp.imx`	An image containing U-Boot and ATF for i.MX 8ULP EVK board.
`u-boot-imx8ulp-trusty.imx`	An image containing U-Boot ATF and Trusty OS for i.MX 8ULP EVK board.
`u-boot-imx8ulp-trusty-secure-unlock.imx`	An image containing U-Boot, ATF, and Trusty OS for i.MX 8ULP EVK board. It is a demonstration of secure unlock mechanism.
`u-boot-imx8ulp-evk-uuu.imx`	An image containing U-Boot and ATF, used by UUU for i.MX 8ULP EVK board. It is not flashed to MMC.
`boot.img`	Boot image for i.MX 8ULP EVK board, it contains the AOSP generic kernel image, generic ramdisk and default kernel command line.
`boot-imx.img`	Boot image for i.MX 8ULP EVK board, it contains the kernel image built from i.MX Kernel tree, generic ramdisk and default kernel command line.
`vendor_boot.img`	Vendor boot image for i.MX 8ULP EVK board. it contains vendor ramdisk and default kernel command line.
`system.img`	System image for i.MX 8ULP EVK board.
`system_ext.img`	System extension image for i.MX 8ULP EVK board.
`vendor.img`	Vendor image for i.MX 8ULP EVK board.
`product.img`	Product image for i.MX 8ULP EVK board.
`super.img`	Super image generated from `system.img`, `system_ext.img`, `vendor.img`, and `product.img`.
`partition-table.img`	GPT partition table image for single-bootloader condition. Used for 16 GB eMMC.
`partition-table-dual.img`	GPT partition table image for dual-bootloader condition. Used for 16 GB eMMC.
`partition-table-28GB.img`	GPT partition table image for single-bootloader condition. Used for 32 GB eMMC.
`partition-table-28GB-dual.img`	GPT partition table image for dual-bootloader condition. Used for 32 GB eMMC.
`dtbo-imx8ulp-9x9.img`	Device Tree image for i.MX 8ULP EVK 9x9 board to support MIPI panel output.
`dtbo-imx8ulp-9x9-hdmi.img`	Device Tree image for i.MX 8ULP EVK 9x9 board to support HDMI output.
`dtbo-imx8ulp.img`	Device Tree image for i.MX 8ULP EVK board to support MIPI panel output.
`dtbo-imx8ulp-hdmi.img`	Device Tree image for i.MX 8ULP EVK board to support HDMI output.

Table 10. Images for i.MX 8ULP EVK...*continued*

i.MX 8ULP EVK Image	Description
`dtbo-imx8ulp-epdc.img`	Device Tree image for i.MX 8ULP EVK board to support EPDC output.
`dtbo-imx8ulp-sof.img`	Device Tree image for i.MX 8ULP EVK board to support the Sound Open Firmware audio output.
`vbmeta-imx8ulp-9x9.img`	Android Verify Boot metadata image for i.MX 8ULP EVK 9x9 board to support MIPI panel output.
`vbmeta-imx8ulp-9x9-hdmi.img`	Android Verify Boot metadata image for i.MX 8ULP EVK 9x9 board to support HDMI output.
`vbmeta-imx8ulp.img`	Android Verify Boot metadata image for i.MX 8ULP EVK board to support MIPI panel output.
`vbmeta-imx8ulp-hdmi.img`	Android Verify Boot metadata image for i.MX 8ULP EVK board to support HDMI output.
`vbmeta-imx8ulp-epdc.img`	Android Verify Boot metadata image for i.MX 8ULP EVK board to support EPDC output.
`vbmeta-imx8ulp-sof.img`	Android Verify Boot metadata image for i.MX 8ULP EVK board to support the Sound Open Firmware audio 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 the default AVB private key.

5 Programming Images

The images from the prebuilt release package or created from source code contain the U-Boot bootloader, system image, GPT image, vendor image, and vbmeta image. At a minimum, the storage devices on the development system (MMC/SD or NAND) must be programmed with the U-Boot bootloader. The i.MX 8 series boot process determines what storage device to access based on the switch settings. When the bootloader 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 Section 6.

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

- UUU to download all images to the eMMC or SD card.
- `imx-sdcard-partition.sh` to download all images to the SD card.
- `fastboot_imx_flashall` script to download all images to the eMMC or 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 vendor boot image: `vendor_boot.img`
- Android system image: `system.img`
- Android system extension image: `system_ext.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 partition is used to put the unpacked codes/data of the applications, system configuration database, and so on. In normal boot mode, the root file system is first 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 11. Storage partitions

Partition type/ index	Name	Start offset	Size	File system	Content
N/A	`bootloader0`	Listed in the following table	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 + part of recovery ramdisk	`boot.img`
4 (6)	`boot_b`	`Follow boot_a`	64 MB	`boot.img` format, a kernel + part of recovery ramdisk	`boot.img`
5 (7)	`vendor_boot_a`	`Follow boot_b`	64 MB	Part of recovery ramdisk	`vendor_boot.img`
6 (8)	`vendor_boot_a`	`Follow boot_b`	64 MB	Part of recovery ramdisk	`vendor_boot.img`
7 (9)	`misc`	`Follow boot_b`	4 MB	N/A	For recovery storage bootloader message, reserve.
8 (10)	`metadata`	`Follow misc`	16 MB	N/A	Metadata of OTA update, remount, etc.

Table 11. Storage partitions*...continued*

Partition type/ index	Name	Start offset	Size	File system	Content
9 (11)	presistdata	Follow metadata	1 MB	N/A	Option to operate unlock\unlock.
10 (12)	userdata	Follow presistdata	4096 MB	N/A	system.img, system_ext.img, vendor.img, and product.img
11 (13)	userdata	Follow super	Remained space		Application data storage for system application. And for internal media partition, in the /mnt/sdcard/ directory.
12 (14)	fbmisc	Follow userdata	1 MB	N/A	To store the state of lock/unlock.
13 (15)	vbmeta_b	Follow fbmisc	1 MB	N/A	To store the verify boot's metadata.
14 (16)	vbmeta_b	Follow vbmeta_a	1 MB	N/A	To store the verify boot's metadata.

Table 12. bootloader0 offset

SoC	bootloader0 offset in eMMC boot0 partition	bootloader0 offset in SD card
i.MX 8M Mini	33 KB	33 KB
i.MX 8M Nano	0	32 KB
i.MX 8M Plus	0	32 KB
i.MX 8M Quad	33 KB	33 KB
i.MX 8ULP	0	32 KB
i.MX 8Quad Max Rev.B	0	32 KB
i.MX 8QuadXPlus Rev.B	32 KB	32 KB
i.MX 8QuadXPlus Rev.C	0	32 KB

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:

```
$ sudo ${MY_ANDROID}/device/nxp/common/tools/imx-sdcard-
partition.sh -f <soc_name> /dev/sdX
```

```
# <soc_name> can be imx8mm,imx8mn,imx8mp,imx8mq, imx8ulp,
 imx8qm,imx8qxp.
```

Note:

- *If the SD card is 16 GB, use `sudo ${MY_ANDROID}/device/nxp/common/tools/imx-sdcard-partition.sh -f <soc_name> /dev/sdX` to flash images in the current working directory.*
- *If the SD card is 32 GB, use `sudo ${MY_ANDROID}/device/nxp/common/tools/imx-sdcard-partition.sh -f <soc_name> -c 28 /dev/sdX` to flash images in the current working directory.*
- *`/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 13. 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 automatically flash all images with only one command.

With virtual A/B feature enabled, your host fastboot tool version should be equal to or later than 30.0.4. You can download the host fastboot tool from the Android website or build it with the Android project. Based on Section Section 3.2, follow the steps below 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
 is flashed;
                                Otherwise U-Boot image is
 flashed.
                            For Android Automative:
                                Only dual-bootloader feature is
 supported. By default, spl&bootloader image is flashed.
      -d dtb_feature    Flashes 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
```

Note:

- *`-f` option is mandatory. The SoC name can be `imx8mm`, `imx8mn`, `imx8mp`, `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/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_android/evk_8mm/`: Images to be flashed are in the directory of `/imx_android/evk_8mm/`.
- `-u trusty`: Flashes `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 14. Relationship between 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`, `vendor_boot_a`, `vendor_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`, `system_ext_a`, `system_ext_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 recognize 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 SD/eMMC

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 15. 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

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 16. 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 1-4 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 there is no `bootargs` defined in U-Boot.

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

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

Table 17. Boot device switch settings

Boot mode switch	SW4
SD boot	0011
eMMC boot	0010
Download mode	0001

- To boot from SD, change the board Boot_Mode switch SW4 to 0011 (from 1-4 bit).
- To boot from eMMC, change the board Boot_Mode switch SW4 to 0010 (from 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 8M Quad EVK board

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

All information provided in this document is subject to legal disclaimers.

Table 18. Boot device switch settings

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

Table 19. 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.

6.1.5 Booting from eMMC on the i.MX 8ULP EVK board

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

Table 20. Boot device switch settings

Boot mode switch	SW5 (from 1-8 bit)
eMMC boot	00000001
Download mode	00000010

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
```

6.1.6 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 21. 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.

6.1.7 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 22. 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 configurations.

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 Section 6.2.2, `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` variable in your U-Boot environment). To boot up the Android system, do the following:

  ```
  > boota
  ```

  To boot into recovery mode, execute the following command:

  ```
  > boota recovery
  ```

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 23. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
`console`	Where to output kernel log by `printk`.	`console=ttymxc0`	i.MX 8M Mini uses `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.

Table 23. Kernel boot parameters*...continued*

Kernel parameter	Description	Typical value	Used when
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. • For i.MX 8M Quad, it is 1280 MB by default. • For i.MX 8QuadXPlus and 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 also allocates memory from CMA and Cortex-M4 cannot use the memory larger than 0x DFFFFFFFF.
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	Setting this argument also bypasses 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	-

Table 23. Kernel boot parameters*...continued*

Kernel parameter	Description	Typical value	Used when
`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 60 fps. To configure fps, change this value to 4kp60/4kp50/4kp30. • 1080p display should be configured as: `androidboot.displaymode=1080p`. The default fps is 60fps. To configure fps, change this value to 1080p60/1080p50/ 1080p30. • 720p display should be configured as: `androidboot.displaymode=720p`. The default fps is 60fps. To configure fps, change this value to 720p60/720p50/ 720p30. • 480p display should be configured as: `androidboot.displaymode=480p`. The default FPS is 60fps. To configure fps, change this value to 480p60/480p50/ 480p30. • For other displaymode which is not 4k/1080p/720p/480p or fps is not 60/50/30, for example: 1024x768p24 display should be configured as: `androidboot.displaymode=1024x768p24`. • 1080p60 display can be configured as: `androidboot.displaymode=1920x1080p60` or `androidboot.displaymode=1080p`.	The system will find out and work at the best display mode, and display mode can be changed through this `bootargs`.

Table 23. Kernel boot parameters*...continued*

Kernel parameter	Description	Typical value	Used when
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	-
moal.mod_para	It is used to set driver load arguments for NXP mxmdriver Wi-Fi driver.	• moal.mod_para=wifi_mod_para_sd8987.conf • moal.mod_para=wifi_mod_para_powersave.conf	-
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	-
swiotlb	It is used to configure the SWIOTLB size. The kernel default value is 64 MB.	swiotlb=65536	i.MX 8M Plus EVK is configured to 128 MB (swiotlb=65536) to fix SWIOTLB overflow issue of the Wi-Fi driver.
androidboot.vendor.sysrq	It is used to enable sysrq.	androidboot.vendor.sysrq=1	-

Table 23. Kernel boot parameters*...continued*

Kernel parameter	Description	Typical value	Used when
androidboot. powersave.usb	It is used to enable USB runtime_pm (auto).	androidboot. powersave. usb=true	-

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 convenience 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/product/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 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}
$ out/host/linux-x86/bin/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.

7.1.4 Building an OTA package to include a dual-bootloader image

The dual-bootloader feature divides the default `u-boot.imx` into two parts: `spl.bin` and `bootloader.img`. `spl.bin` leads to the bootloader0 partition, which is managed by U-Boot itself, while `bootloader.img` leads to the bootloader_a/bootloader_b partitions, which are managed by GPT. Taking i.MX 8M Mini as an example, the layout of the dual-bootloader images is as follows.

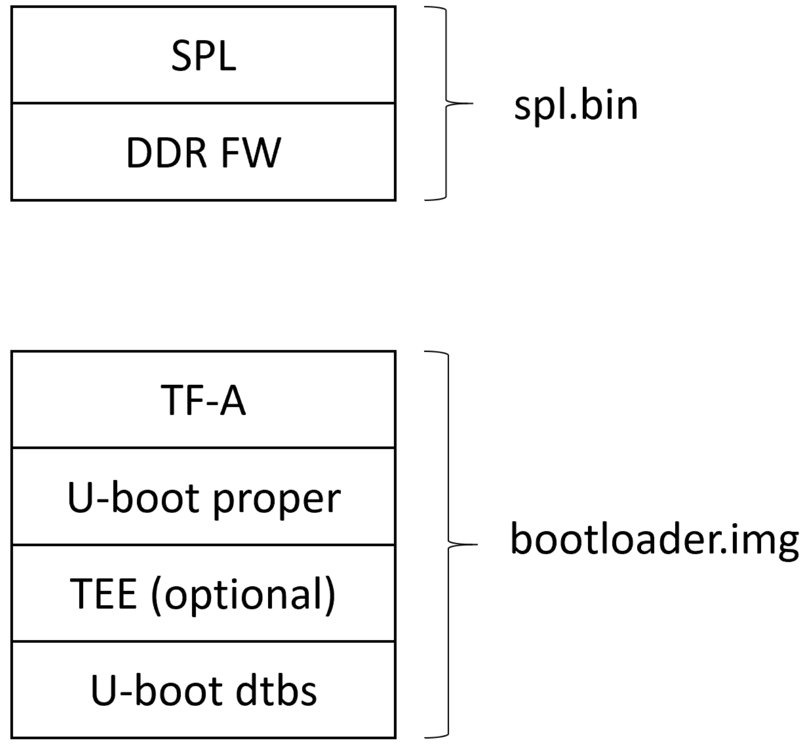

Figure 1. Dual-bootloader image layout

The dual-bootloader feature is optional, but it is useful in some cases as it can provide a secure way to update the bootloader image. To include `bootloader.img` into the OTA package, set some configurations. Taking i.MX 8M Mini as an example, add the following changes to `${MY_ANDROID}/device/nxp`:

```
diff --git a/imx8m/evk_8mm/AndroidBoard.mk b/imx8m/evk_8mm/
AndroidBoard.mk
index 75193153..35bae3b4 100644
--- a/imx8m/evk_8mm/AndroidBoard.mk
+++ b/imx8m/evk_8mm/AndroidBoard.mk
@@ -7,3 +7,6 @@ include $(CONFIG_REPO_PATH)/common/build/gpt.mk
 include $(FSL_PROPRIETARY_PATH)/fsl-proprietary/media-profile/
media-profile.mk
 include $(FSL_PROPRIETARY_PATH)/fsl-proprietary/sensor/fsl-
sensor.mk
 -include $(IMX_MEDIA_CODEC_XML_PATH)/mediacodec-profile/
mediacodec-profile.mk
+
+BOARD_PACK_RADIOIMAGES += bootloader.img
+INSTALLED_RADIOIMAGE_TARGET  += $(PRODUCT_OUT)/bootloader.img
```

```
diff --git a/imx8m/evk_8mm/BoardConfig.mk b/imx8m/evk_8mm/
BoardConfig.mk
index 6648989a..885a6563 100644
--- a/imx8m/evk_8mm/BoardConfig.mk
+++ b/imx8m/evk_8mm/BoardConfig.mk
@@ -179,3 +179,7 @@ BOARD_SEPOLICY_DIRS += \
        $(IMX_DEVICE_PATH)/sepolicy_drm
 endif
+AB_OTA_PARTITIONS += bootloader
+
+PRODUCT_COPY_FILES += \
+    $(OUT_DIR)/target/product/$(PRODUCT_DEVICE)/
obj/UBOOT_COLLECTION/bootloader-imx8mm-trusty-
dual.img:bootloader.img
```

On the HAB CLOSED boards, replace `bootloader-imx8mm-trusty-dual.img` with the `bootloader.img`, which is signed by Code Signing Tool (CST). For how to sign the bootloader image with CST, see the *i.MX Android Security User's Guide* (ASUG).

7.1.5 Building an OTA package with postinstall command

Postinstall is a mechanism to execute a specified command in the updated partition during OTA process. To enable this mechanism, add some build configurations.

This release provides a demonstration for enabling the vendor partition postinstall command. You can find the following code in the repository under the `${MY_ANDROID}/device/nxp` directory:

```
AB_OTA_POSTINSTALL_CONFIG += \
RUN_POSTINSTALL_vendor=true \
POSTINSTALL_PATH_vendor=bin/imx_ota_postinstall \
FILESYSTEM_TYPE_vendor=ext4 \
POSTINSTALL_OPTIONAL_vendor=false
```

The preceding configurations are as follows:

- The vendor partition postinstall command is enabled.
- After the vendor partition is updated, the vendor partition with updated image is mounted on the `/postinstall` directory, and the `/postinstall/bin/imx_ota_postinstall` command is executed.
- The updated vendor partition is of Ext4 type.
- The vendor partition postinstall command is not optional. If the command fails, the whole OTA process will not be marked as success.

As you can find in the source code, the preceding configurations do not take effect by default unless a variable named `IMX_OTA_POSTINSTALL` is assigned with an appropriate value. For example, assign a value when executing the command to build an OTA package as follows:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ ./imx-make.sh bootloader kernel -j4
$ make otapackage -j4 IMX_OTA_POSTINSTALL=1
```

This postinstall mechanism is not mutually exclusive with full update package or incremental update package. It can be used with both of them.

In the demonstration, `imx_ota_postinstall` corresponds to a shell script, and the source code is under the `${MY_ANDROID}/vendor/nxp-opensource/imx/ota_postinstall/` directory. It is used to update the `bootloader0` partition, which does not have a/b slot.

Note: *You should be aware of the risk that the update of the `bootloader0` partition may fail and there is no way to roll back.*

During the execution of this command, it invokes the `dd` command to write the file `/postinstall/etc/bootloader0.img` to the appropriate offset of the boot device. You can modify the configuration source code to decide which file is copied to the vendor partition and named as `bootloader0.img`. Taking i.MX 8M Mini EVK as an example, the following code lines in the release code can copy the U-Boot image with Trusty OS to vendor partition and name it as `bootloader0.img`. If dual-bootloader feature is enabled, the SPL image should be copied. If the board is closed, the image should be signed first.

```
PRODUCT_COPY_FILES += \
  $(OUT_DIR)/target/product/$(firstword $(PRODUCT_DEVICE))/
obj/UBOOT_COLLECTION/u-boot-imx8mm-trusty.imx:
$(TARGET_COPY_OUT_VENDOR)/etc/bootloader0.img
```

See the *i.MX Android Security User's Guide* (ASUG) about how to sign the bootloader0 image with CST. In the default configuration, an SPL image is copied to be `bootloader0.img` because dual-bootloader is recomended.

You can implement your own postinstall command and perform the operations as needed during the OTA process.

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 Section 7.1.2 and Section 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 shows "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 storage: `adb push payload.bin /data/ota_package`.
3. Cat the content of `payload_properties.txt` as follows:
   - `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:///
   data/ota_package/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 displays "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 (for example, 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 Section 7.1.2 and Section 7.1.3.*

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 Automotive 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/nxp/imx8m/evk_8mm/BoardConfig.mk`:

```
TARGET_BOARD_DTS_CONFIG ?= imx8mm-ddr4:imx8mm-ddr4-evk.dtb
TARGET_BOARD_DTS_CONFIG += imx8mm:imx8mm-evk-usd-wifi.dtb
TARGET_BOARD_DTS_CONFIG += imx8mm-mipi-panel:imx8mm-evk-
rm67199.dtb
TARGET_BOARD_DTS_CONFIG += imx8mm-mipi-panel-rm67191:imx8mm-
evk-rm67191.dtb
TARGET_BOARD_DTS_CONFIG += imx8mm-m4:imx8mm-evk-rpmsg.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 rm67199 MIPI panel display, modify this BOARD_PREBUILT_DTBOIMAGE as follows:

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

To generate an OTA package for `evk_8mm` with rm67191 MIPI panel display, modify this BOARD_PREBUILT_DTBOIMAGE as follows:

```
BOARD_PREBUILT_DTBOIMAGE := out/target/product/evk_8mm/dtbo-
imx8mm-mipi-panel-rm67191.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 reads 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/nxp/. 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 camera. 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 goes 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 24. 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 has 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 25. 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.
`orientation`	If (`PixelArrayWidth` > `PixelArrayHeight`), and the screen is portrait(w < h), set it to **90**. If (`PixelArrayWidth` < `PixelArrayHeight`), and the screen is landscape(w > h), set it to **90**. Otherwise, set it to **0**.
`FocalLength`	Focal length.
`MinFrameDuration`	Minimum FPS.
`MaxFrameDuration`	Maximum FPS.
`MaxJpegSize`	Maximum JPEG size.
`PhysicalWidth`	`PixelArrayWidth * siz_of_one_pixel` (For OV5640, it is 1.4 um; For max9286, it is 4.2 um.)
`PhysicalHeight`	`PixelArrayHeight * siz_of_one_pixel` (For OV5640, it is 1.4 um; For max9286, it is 4.2 um.)

8.1.3 Making cameras work on i.MX 8M Plus EVK with non-default images

The default image for i.MX 8M Plus EVK supports basler + basler and the cameras can work after the image is flashed and boot up. To make cameras work with non-default images, execute the following additional commands:

- Basler (CSI1) + OV5640 (CSI2) or only Basler (CSI1) on the host

```
# flash the image
sudo ./fastboot_imx_flashall.sh -f imx8mp -a -e -d basler-
ov5640 // or "-d basler" for Only basler(CSI1)

# set bootargs
# In serial console, enter into uboot command mode, run below
 commads:
# If enable basler 4k size, also add
 androidboot.camera.ispsensor.maxsize=4k.
setenv append_bootargs androidboot.camera.layout=basler-ov5640
saveenv
boota
```

- Only OV5640 (CSI1) on the host

```
# flash the image
sudo ./fastboot_imx_flashall.sh -f imx8mp -a -e -d ov5640

# set bootargs
# In serial console, enter into uboot command mode, run below
 commad:
setenv append_bootargs androidboot.camera.layout=only-ov5640
saveenv
boota
```

Note:

`-d ov5640` can be replaced by one of below:

-d lvds, -d lvds-panel, -d mipi-panel, -d mipi-panel-rm67191, -d rpmsg, -d sof.

- Only OS08A20 (CSI1)

```
# flash the image
sudo ./fastboot_imx_flashall.sh -f imx8mp -a -e -d os08a20

# set bootargs
# In serial console, enter into uboot command mode, run below
 commads:
# If enable os08a20 4k size, also add
 androidboot.camera.ispsensor.maxsize=4k.
setenv append_bootargs androidboot.camera.layout=os08a20-
ov5640
saveenv
boota
```

- OS08A20 (CSI1) + OS08A20 (CSI2)

```
# flash the image
sudo ./fastboot_imx_flashall.sh -f imx8mp -a -e -d dual-
os08a20

# set bootargs
# In serial console, enter into uboot command mode, run below
 commad:
setenv append_bootargs androidboot.camera.layout=dual-os08a20
saveenv
boota
```

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 keep 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 adds `AUDIO_OUTPUT_FLAG_DIRECT` for this track, and this stream plays from `DirectOutputThread`.

In low-power audio mode, the default audio period time is 500 milliseconds, and the whole buffer can hold 20 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 20
> setprop vendor.audio.lpa.period_ms 500
```

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

1. Add `-d m4 -m` or `-d rpmsg -m` when flashing images to support audio playback based on MCU FreeRTOS, for example:
   - For i.MX 8M Mini: `uuu_imx_android_flash.sh -f imx8mm -e -d m4 -m`
   - For i.MX 8M Plus: `uuu_imx_android_flash.sh -f imx8mp -e -d rpmsg -m`
2. Add `bootmcu` to `bootcmd` and add `androidboot.lpa.enable=1 snd_pcm.max_alloc_per_card=134217728` to `append_bootargs` in U-Boot command line.

   ```
   setenv bootcmd "bootmcu && boota"
   setenv append_bootargs androidboot.lpa.enable=1
    snd_pcm.max_alloc_per_card=134217728
   saveenv
   ```
3. Boot up the system, and push `.wav` audio files to `/sdcard/`. It is better to use a long duration audio file.
4. Open the `DirectAudioPlayer` application, and select a file from the spinner. The file selected is listed under the spinner.
5. Click the **Play** button to play audio.
6. Press the ON/OFF button on the board. The system then enters suspend mode, and the audio can keep playing.

Note:

- *Only i.MX 8M Mini EVK Board and i.MX 8M Plus EVK Board support this feature.*
  - *For i.MX 8M Mini EVK Board, the audio is output from "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).*
  - *For i.MX 8M Plus EVK Board, the audio is output from the HEADPHONE jack.*
- *`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 median are not supported. For example, it does not support playing 44100 Hz audio.*
- *`DirectAudioPlayer` supports 24/32 bits `.wav` file with sampling rates no more than 192000.*

8.2.2 Supporting a new sound card

Perform the following steps to support a new sound card on the Android system:

1. Add a new audio configuration JSON file.
   Each sound card needs one JSON file under the `/vendor/etc/configs/audio` folder of the board, so that Android audio HAL code can manage this card. The content of the JSON file mainly includes the card's driver name, supported output/input device type, and mixer controls that need to be configured.
   See `${MY_ANDROID}/device/nxp/common/audio-json/readme.txt` for details to create such a JSON file. After that, copy the JSON file to the board by the following command in Android makefile:

   ```
   PRODUCT_COPY_FILES += \
   ```

```
device/nxp/common/audio-json/xxx_config.json:
$(TARGET_COPY_OUT_VENDOR)/etc/configs/audio/xxx_config.json
```

2. Configure the audio mix port, device port, and route in `${MY_ANDROID}/device/nxp/imx8m/evk_8mp/audio_policy_configuration.xml`.
   - Mix ports describe the possible configuration profiles for streams that can be opened at the audio HAL for playback and capture.
   - Device ports describe the devices that can be attached with their type.
   - Routes describe which mix port can route to which device.

   Take the following configuration as an example. It means that the system supports three output devices: speaker, headphone, and HDMI. If the speaker or headphone is connected, it expects that the frameworks can deliver 16 bit, 48 kHz, and stereo streams to them. If an HDMI device is connected, it expects 24 bit, 48 kHz, and stereo streams.

```
<mixPort name="primary output" role="source"
 flags="AUDIO_OUTPUT_FLAG_PRIMARY">
    <profile name="" format="AUDIO_FORMAT_PCM_16_BIT"
             samplingRates="48000"
 channelMasks="AUDIO_CHANNEL_OUT_STEREO"/>
</mixPort>
<mixPort name="hdmi output" role="source">
    <profile name="" format="AUDIO_FORMAT_PCM_8_24_BIT"
             samplingRates="48000"
 channelMasks="AUDIO_CHANNEL_OUT_STEREO"/>
</mixPort>
<devicePort tagName="Speaker" type="AUDIO_DEVICE_OUT_SPEAKER"
 role="sink" >
</devicePort>
<devicePort tagName="Wired Headphones"
 type="AUDIO_DEVICE_OUT_WIRED_HEADPHONE" role="sink">
</devicePort>
<devicePort tagName="HDMI Out"
 type="AUDIO_DEVICE_OUT_AUX_DIGITAL" role="sink">
</devicePort>
<route type="mix" sink="Speaker"
       sources="primary output"/>
<route type="mix" sink="Wired Headphones"
       sources="primary output"/>
<route type="mix" sink="HDMI Out"
       sources="hdmi output"/>
```

3. (Optional) Support device hot plug.

   Android frameworks support dynamically switching default output device by catching the device's hot-plug uevent. The uevent can be sent in the kernel by extcon driver.

   a. Declare which device type supports:

```
static const unsigned int xxx_extcon_cables[] = {
   EXTCON_JACK_HEADPHONE,
   EXTCON_NONE,
};
struct extcon_dev xxx_edev;
```

   b. Allocate and register the extcon device:

```
xxx_edev = devm_extcon_dev_allocate(&pdev->dev,
 xxx_extcon_cables);
devm_extcon_dev_register(&pdev->dev, xxx_edev);
```

c. When the device is connected, execute the following command to tell frameworks that the headphone device has been connected:

```
extcon_set_state_sync(extcon_dev, EXTCON_JACK_HEADPHONE,
 1);
```

d. When the device is disconnected, execute the following command:

```
extcon_set_state_sync(extcon_dev, EXTCON_JACK_HEADPHONE,
 0).
```

8.2.3 Enabling powersave mode

By default, the DRAM speed is 4000 MT/s, the GIC frequency is 500 MHz, and VDD_SOC is 0.95 V. A powersaving mode can be achieved with the following conditions:

- DRAM speed is 2400 MT/s.
- VDD_SOC is 0.85 V.
- Prohibit the eMMC module, FEC module, BT module, and Wi-Fi module from requesting high bus frequency.
- Disable LDB, ISP, and HDMI.
- USB power domain is active when the USB is in use, and enters suspending when the USB is not in use.
- When playing local audio and output with Bluetooth headset, playing local audio through LPA and output with wired headset, playing online audio and ouput with wired headset at the time of screen off, the DRAM speed is 400 MT/s and the GIC frequency is 100 MHz.

Perform the following steps to enable powersave mode:

1. Download the GCC tool chain from the arm Developers GNU-RM Downloads page. It is recommended to download the `7-2018-q2-update` version. Extract it to the installation directory, and export the directory as `export ARMGCC_DIR=<install_dir>/gcc-arm-none-eabi-7-2018-q2-update` and add it to `/etc/profile`.
2. Upgrade the CMake version to or higher than 3.13.0. If the CMake version on your machine is not higher than 3.13.0, you can execute the following commands to upgrade it:

```
wget https://github.com/Kitware/CMake/releases/download/
v3.13.2/cmake-3.13.2.tar.gz
tar -xzvf cmake-3.13.2.tar.gz; cd cmake-3.13.2;
sudo ./bootstrap
sudo make
sudo make install
```

3. Build image with `POWERSAVE=true`.

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

Perform the following steps to play audio in powersaving mode with the MCU image:

1. Use `-u trusty-powersave -d rpmsg -m` when flashing images to enable powersave mode, for example:

```
# For imx8mp
sudo uuu_imx_android_flash.sh -f imx8mp -e -u trusty-
powersave -d rpmsg -m
```

All information provided in this document is subject to legal disclaimers.

2. Set `bootargs` in U-Boot command line:

   ```
   setenv append_bootargs androidboot.lpa.enable=1
    snd_pcm.max_alloc_per_card=134217728
   saveenv
   ```

3. Set `bootcmd` in U-Boot command line:

   ```
   setenv bootcmd "bootmcu && boota"
   saveenv
   ```

   Make sure that only "MIPI DSI", "Debug UART", and "Power" ports are connected on the board.
4. To play local audio through LPA and output with wired headset:
   1. Boot up the system.
   2. Push the `.wav` audio files to `/sdcard/`. It is better to use a long duration audio file.
   3. Open the DirectAudioPlayer application. Select a file from the spinner, and the file selected is listed under the spinner.
   4. Click the Play button to play audio.
   5. Press the power key on the board to make the system enter suspend mode, and the audio can keep playing.

Perform the following steps to play audio in powersaving mode without the MCU image:

1. Use `-u trusty-powersave -d powersave-non-rpmsg` when flashing images to enable the powersave mode, for example:

   ```
   # For imx8mp
   sudo uuu_imx_android_flash.sh -f imx8mp -e -u trusty-
   powersave -d powersave-non-rpmsg
   ```

   Make sure that only "MIPI DSI", "Debug UART", and "Power" ports are connected on the board.
2. To play audio and output with Bluetooth headset:
   1. Boot up the system.
   2. Push the `.mp3` audio files to `/sdcard/`. It is better to use a long duration audio file.
   3. Connect a Bluetooth headset.
   4. Play the `.mp3` audio file and turn of the screen.
3. To play online audio and ouput with wired headset:
   1. Boot up the system.
   2. Connect to the Wi-Fi access point.
   3. Open the Spotify application and play audio and turn off the screen.

Note: *Only the i.MX 8M Plus EVK Board supports this feature.*

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/nxp/` 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/nxp/` and recompiling the code or setting in U-Boot command line as `bootargs` during boot up.

Note:

- *For the i.MX 8M Mini EVK board, the source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mm/BoardConfig.mk`.*
- *For the i.MX 8M Nano EVK board, the source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mn/BoardConfig.mk`.*
- *For the i.MX 8M Plus EVK board, the source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mp/BoardConfig.mk`.*
- *For the i.MX 8MQuad EVK board, the source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mq/BoardConfig.mk`.*
- *For the i.MX 8ULP EVK board, the source folder is `${MY_ANDROID}/device/nxp/imx8ulp/evk_8ulp/BoardConfig.mk`.*
- *For the i.MX 8QuadMax/8QuadXPlus MEK board, the source folder is `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk`.*

8.3.2 Enabling multiple-display function

The following boards support more than one display.

Table 26. Boards supporting multiple displays

Board	Number of displays	Display port
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
i.MX 8M Plus EVK	3	MIPI-DSI, LVDS0, HDMI

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

The three displays on i.MX 8M Plus EVK are enabled by default.

To evaluate the multiple-display feature with physical 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/nxp/` directory.

Take i.MX 8M Plus EVK 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"
```

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

```
getevent -i | grep location
location: "usb-xhci-hcd.0.auto-1.3.4/input0"
location: "usb-xhci-hcd.0.auto-1.2.4/input0"
location: "usb-xhci-hcd.0.auto-1.1.4/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-hcd.0.auto-1.1.4/
input0" />
    <port display="1" input="usb-xhci-hcd.0.auto-1.2.4/
input0" />
    <port display="2" input="usb-xhci-hcd.0.auto-1.3.4/
input0" />
</ports>
```

To make the modifications take effect, modify the source file under the `${MY_ANDROID}/device/nxp/` directory and rebuild the images. Keep the connection of display 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, 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 platform. 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 pop-up menu. The selected application is then displayed on the 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 the A2DP_sink profile is disabled.

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

The following is an example to set A2DP_sink enabled and A2DP disabled for the i.MX 8M Mini board.

The file is `${MY_ANDROID}/device/nxp/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 fec2840430..c1c963bef3 100644
--- a/configs/imx8qm_mek_android_defconfig
+++ b/configs/imx8qm_mek_android_defconfig
@@ -136,7 +136,7 @@ CONFIG_SPL_PHY=y
CONFIG_SPL_USB_GADGET=y
CONFIG_SPL_USB_SDP_SUPPORT=y
-CONFIG_SPL_SDP_USB_DEV=1
+CONFIG_SPL_SDP_USB_DEV=0
CONFIG_SDP_LOADADDR=0x80400000
CONFIG_FASTBOOT=y
@@ -147,7 +147,7 @@ CONFIG_FASTBOOT_UUU_SUPPORT=n
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 1fb6b45768..c60f924f02 100644
--- a/include/configs/imx8qm_mek_android.h
+++ b/include/configs/imx8qm_mek_android.h
@@ -19,7 +19,6 @@
#define IMX_HDMITX_FIRMWARE_SIZE 0x20000
#define IMX_HDMIRX_FIRMWARE_SIZE 0x20000
-#define CONFIG_FASTBOOT_USB_DEV 1
#undef CONFIG_EXTRA_ENV_SETTINGS
#undef CONFIG_BOOTCOMMAND
```

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 2dbd3f3f91..57aec56b0c 100644
--- a/configs/imx8qxp_mek_android_defconfig
+++ b/configs/imx8qxp_mek_android_defconfig
@@ -138,7 +138,7 @@ CONFIG_SPL_PHY=y
```

```
CONFIG_SPL_USB_GADGET=y
CONFIG_SPL_USB_SDP_SUPPORT=y
-CONFIG_SPL_SDP_USB_DEV=1
+CONFIG_SPL_SDP_USB_DEV=0
CONFIG_SDP_LOADADDR=0x80400000
CONFIG_FASTBOOT=y
@@ -149,7 +149,7 @@ CONFIG_FASTBOOT_UUU_SUPPORT=n
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 7e70e92f49..d8e420114f 100644
--- a/include/configs/imx8qxp_mek_android.h
+++ b/include/configs/imx8qxp_mek_android.h
@@ -16,8 +16,6 @@
#define FSL_FASTBOOT_FB_DEV "mmc"
-#define CONFIG_FASTBOOT_USB_DEV 1
-
#undef CONFIG_EXTRA_ENV_SETTINGS
#undef CONFIG_BOOTCOMMAND
```

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 the related `defconfig` files:

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

8.5.2 Changing the VID/PID values of the USB Gadget

8.5.2.1 USB Gadget in U-Boot

The USB Gadget functions in the U-Boot stage include fastboot and SPL Serial Download Protocol (SDP).

The VID/PID values for fastboot are **0x1fc9/0x0152**, they are configured with two `defconfig` items as follows. They can be found in the `defconfig` file.

```
CONFIG_USB_GADGET_VENDOR_NUM=0x1fc9
CONFIG_USB_GADGET_PRODUCT_NUM=0x0152
```

The VID/PID values for SPL SDP are **0x1fc9/0x0151**. The VID value is the same as before, and the PID value is changed to **0x0151** with the following function. The corresponding source code file is `${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/arch/arm/mach-imx/spl.c`.

```
int g_dnl_bind_fixup(struct usb_device_descriptor *dev, const
 char *name)
{
    put_unaligned(0x0151, &dev->idProduct);
    return 0;
}
```

The UUU tool relies on the VID/PID value, the reference values can be found in the UUU source code config.cpp. Therefoe, if the values are changed, UUU may not work. But the

U-Boot image used with UUU is not flashed to the board, so the one in prebuilt images can be used during development if the VID/PID values need to be changed.

8.5.2.2 USB Gadget on the Android platform

The are many VID/PID value sets on the Android platform. They are set in the USB Gadget HAL with the following function. The corresponding source code file is `${MY_ANDROID}/vendor/nxp-opensource/imx/usb/UsbGadget.cpp`. Search the name of the following function in the source code file. Different PID/VID values are used when the Gadget provides different functions. Change the values based on your requirement.

```
static V1_0::Status setVidPid(const char *vid, const char *pid)
```

8.5.2.3 USB Gadget in Recovery

The USB Gadget functions in Recovery include `adb` and `fastbootd`. The VID/PID values are set in `${MY_ANDROID}/bootable/recovery/etc/init.rc`. The following lines can be found in the file:

```
write /config/usb_gadget/g1/idVendor 0x18D1
write /config/usb_gadget/g1/idProduct 0xD001
write /config/usb_gadget/g1/idProduct 0x4EE0
```

Change the value in preceding lines based on your requirement.

8.6 Trusty OS/security configuration

Trusty OS firmware is used in i.MX Android 12 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-12 -m imx-trusty-
android-12.0.0_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` file are 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 uses 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 `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/tee-imx8mn.bin`.*
- *For i.MX 8M Plus EVK, use `make imx8mp` to build the Trusty OS image, and copy the file `lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/ uboot-firmware/tee-imx8mp.bin`.*
- *For i.MX 8M Quad EVK, use `make imx8m` to build the Trusty OS image, and copy the final `lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8m/tee-imx8mq.bin`.*
- *For i.MX 8ULP EVK, use `make imx8ulp` to build the Trusty OS image, and copy the final lk.bin to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8ulp/tee-imx8ulp.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 codebase.*

8.6.1 Initializing the secure storage for Trusty OS

Trusty OS uses the secure storage to protect userdata. This secure storage is based 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 CAAM hardware bound key or vendor specified key are both supported. The RPMB key cannot be changed once it is set.

- To set a **CAAM hardware bound** key, perform the following steps:
  Make your board enter fastboot mode, and then execute the following commands on the host side:
  - `fastboot oem set-rpmb-hardware-key`
    After the board is rebooted, the RPMB service in Trusty OS is initialized successfully.
- To set a **vendor specified** key, perform the following steps:
  Make your board enter fastboot mode, and then execute the following commands on the host side:
  - `fastboot stage < path-to-your-rpmb-key >`
  - `fastboot oem set-rpmb-staged-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_test.bin
```

–
```
echo -n "RPMB" > rpmb_key_test.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' >>
 rpmb_key_test.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.

- ***Note:***
  For more details, see the i.MX Android Security User's Guide (ASUG).

8.6.2 Provisioning the AVB key

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

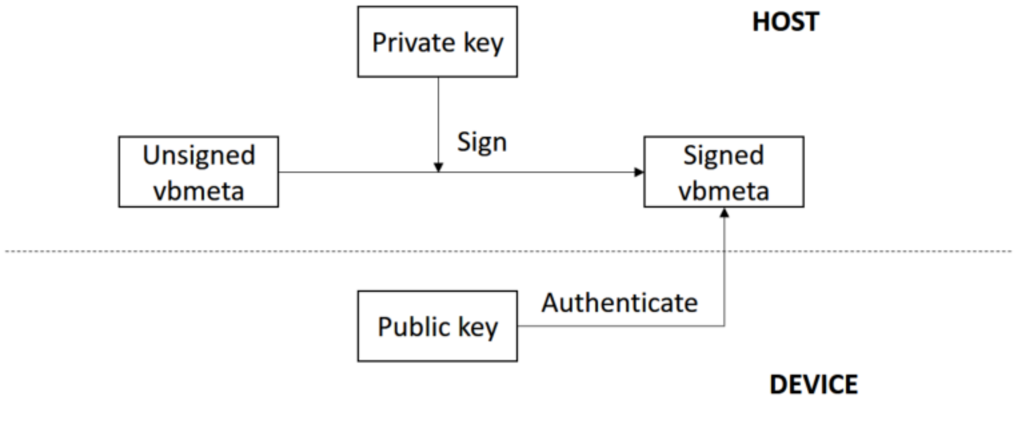

Figure 2. Relationship between AVB key and vbmeta

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 signs the vbmeta struct in vbmeta image with the private key above and stores one copy of the public key in the signed vbmeta image. During AVB verification, the U-Boot validates the public key first, and then uses 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 AVB boot key

The boot image is built as chained partition and the vbmeta struct in boot image is signed by a pair of asymmetric keys (AVB boot key. For more information about the chained partition, see https://android.googlesource.com/platform/external/avb/+/master/README.md.

By default, the Android platform uses the test AVB boot key to sign the boot image. It is located at:

```
${MY_ANDROID}/external/avb/test/data/testkey_rsa2048.pem
```

Custom keys should be used for production. See Section Section 8.6.2.1 to generate the custom private key. The AVB boot key and algorithm can be overridden by setting the following configurations:

```
BOARD_AVB_BOOT_ALGORITHM := <algorithm-type>
```

```
BOARD_AVB_BOOT_KEY_PATH := <key-path>
```

8.6.4 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, 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
  ```

  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
  ```

  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, download the SCFW porting kit on the i.MX Software and Development Tools page. For this release, click "Embedded Linux", and then click the "RELEASES" tab. Find the Linux 5.15.32_2.0.0 release and download its corresponding SCFW Porting kit. Then decompress the file with the following commands:

```
tar -zxvf imx-scfw-porting-kit-1.13.0.tar.gz
```

```
cd packages
chmod a+x imx-scfw-porting-kit-1.13.0.bin
./imx-scfw-porting-kit-1.13.0.bin
cd imx-scfw-porting-kit-1.13.0/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 the 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 packages 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/nxp`.

Note:

- *For i.MX 8M Mini EVK Board, the source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mm/BoardConfig.mk`.*
- *For i.MX 8M Nano EVK Board, source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mn/BoardConfig.mk`.*
- *For i.MX 8M Plus EVK Board, source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mp/BoardConfig.mk`.*
- *For i.MX 8M Quad EVK Board, the source folder is `${MY_ANDROID}/device/nxp/imx8m/evk_8mq/BoardConfig.mk`.*
- *For i.MX 8ULP EVK Board, the source folder is `${MY_ANDROID}/device/nxp/imx8ulp/evk_8ulp/BoardConfig.mk`.*
- *For i.MX 8QuadMax/8QuadXPlus MEK, the source folder is `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk`.*

8.8.2 Modifying the super partition

The partition of super is used to hold logical partitions. Metadata describing the layout of logical partitions in super partition is at the beginning of the super partition. When the system boots up, the init program parses the metadata in super partition and creates logical partitions to mount.

With virtual A/B feature, the super partition can only have the size for one slot of logical partitions. Now the size of super partition is 4.0 GB. 10 MB reserved in this 4.0 GB for metadata. You can find the code as follows in `${MY_ANDROID}/device/nxp`:

```
BOARD_SUPER_PARTITION_SIZE := 4294967296
BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4284481536
```

Refer to the following patch to change the super partition size to 4 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": "4096 MiB",
    +            "size": "3584 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 := 4294967296
    -    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4284481536
    +    BOARD_SUPER_PARTITION_SIZE := 3758096384
    +    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 3747610624
       endif
       ifeq ($(IMX_NO_PRODUCT_PARTITION),true)
         BOARD_NXP_DYNAMIC_PARTITIONS_PARTITION_LIST := system
 system_ext vendor
    diff --git a/imx8q/BoardConfigCommon.mk b/imx8q/
BoardConfigCommon.mk
    index 85d3561..c7352a2 100644
    --- a/imx8q/BoardConfigCommon.mk
    +++ b/imx8q/BoardConfigCommon.mk
    @@ -164,8 +164,8 @@ ifeq
 ($(TARGET_USE_DYNAMIC_PARTITIONS),true)
           BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4024434688
         endif
       else
    -    BOARD_SUPER_PARTITION_SIZE := 4294967296
    -    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 4284481536
    +    BOARD_SUPER_PARTITION_SIZE := 3758096384
    +    BOARD_NXP_DYNAMIC_PARTITIONS_SIZE := 3747610624
       endif
       ifeq ($(IMX_NO_PRODUCT_PARTITION),true)
         BOARD_NXP_DYNAMIC_PARTITIONS_PARTITION_LIST := system
 system_ext vendor
```

8.9 Notices before the debugging work

When doing the customization work, you may need to do some debugging work. The debugging work will be convenient and flexible if the read-only filesystems are remounted as writable, so that files in it can be replaced with the `adb push` command. It helps to avoid flashing the images again and saves time.

To remount the read-only filesystems, perform the following steps:

1. Unlock the device.
2. Boot up the system to Android platform.
3. Execute the following commands on the host. The second command takes seconds to finish.

   ```
   $ adb root
   $ adb disable-verity
   ```

4. Reboot the device, and execute the following command on the host:

   ```
   $ adb root
   $ adb remount
   ```

Then, the images can be pushed to the board with the `adb push` command. Before the further debugging work, be aware of the following notices:

- Do not erase the "userdata" partition after `adb disable-verity` is executed.
  With the dynamic partition feature enabled in i.MX Android images, and the size is not specified for `system`, `system_ext`, `vendor`, and `product` partitions when building the images. `overlayfs` is used when remounting the read-only filesystems. An upper

directory that can be written in `overlayfs` is needed in this condition. When the `adb push` command is executed, the files are pushed to the upper directory of `overlayfs`, while the original read-only filesystems are not modified.
i.MX Android images use only one partition named "super" to store images in logical partitions, and `ext4` filesystem is used for the userdata partition, which is mounted on `/data`. When executing the `adb disable-verity` command, an image is allocated under `/data/gsi/remount/scratch.img.0000`. Its size is half the size of the "super" partition and should not be greater than 2 GB. The layout information of this image is stored in `/metadata/gsi/remount/lpmetadata` in the format logical partition metadata.
When rebooting the system, at the first stage of the init program, the information in `/metadata/gsi/remount/lpmetadata` is used to create a logical partition named "scratch", and it is mounted on `/mnt/scratch`. This is used as the upper directory in overlayfs used in remount. When the `adb push` command is executed to modify the originally read-only filesystems, files are written to the "scratch" partition.
At the first stage of the init program, the userdata partition is not mounted. The code judges whether the backing image of the scratch partition exists in the userdata partition by checking whether the `/metadata/gsi/remount/lpmetadata` file can be accessed. Therefore, if the userdata partition is erased, but the logical partition is still created, this could be catastrophic and may make the system crash.

- To modify the files from the console, execute `remount` on the console first.
  `adb` and `sh` are in different mount namespaces. `adb remount` does not change the mount status that `sh` sees.
- For MEK boards, if files need to be pushed to `/vendor/etc`, push them to another path.
  Images for i.MX 8Quad Max MEK and i.MX 8QuadXPlus MEK are built together with one target. Media codec configuration files' names and paths are hardcoded in framework, while these two SoCs need different media codec configurations. It means that the media codec configuration files for these two boards with different content should have the same name and being accessed with the same path. Therefore, overlayfs is used, and images for these two boards have different overlayfs upper directories. The mount command can be found in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/init.rc`:

```
mount overlay overlay /vendor/etc ro lowerdir=/vendor/
vendor_overlay_soc/${ro.boot.soc_type}/vendor/etc:/vendor/
etc,override_creds=off
```

  The value of `${ro.boot.soc_type}` can be `imx8qxp` or `imx8qm` here.
  With the preceding command executed, access to files under `/vendor/etc` can access files both under `/vendor/etc` and `/vendor/vendor_overlay_soc/${ro.boot.soc_type}/vendor/etc`. The `/vendor/vendor_overlay_soc/${ro.boot.soc_type}/vendor/etc:/vendor/etc` directory is the upper directory in overlayfs and `/vendor/etc` is both the lower directory and mount point.
  After remount, the lower directory `/vendor/etc` is still read-only, and files can be pushed to other sub-paths under `/vendor` except `/vendor/etc`. To push a modified file, which should be accessed from `/vendor/etc`, push it to `/vendor/vendor_overlay_soc/${ro.boot.soc_type}/vendor/etc`, and then reboot the system to make it take effect.
  For example, if you modified the file `cdnhdmi_config.json`, a file should be under `/vendor/etc/configs/audio/`. Execute the following commands on the console:

```
su
umask 000
cd /vendor/vendor_overlay_soc/imx8qm/vendor/etc/
```

```
mkdir -p configs/audio/
```

Then, execute the following commands on the host:

```
sudo adb push cdnhdmi_config.json /vendor/vendor_overlay_soc/
imx8qm/vendor/etc/
```

At last, reboot the device to make this change take effect.

There are two limitations here:

– To delete a file under `/vendor/etc/`, you can only rebuild the image and flash the vendor image again.

– The overlayfs is mounted with a command in an init `.rc` file. The init `.rc` files are all parsed by the init program before the overlayfs is mounted. Therefore, to modify init `.rc` files under `/vendor/etc`, you can only rebuild the image and flash the vendor image again.

9 Generic Kernel Image (GKI) Development

9.1 GKI introduction

The Generic Kernel Image (GKI) project addresses kernel fragmentation by unifying the core kernel and moving SoC and board support out of the core kernel into loadable modules. The GKI kernel presents a stable Kernel Module Interface (KMI) for kernel modules, so modules and kernel can be updated independently.

Devices that launch with the Android 12 (2021) platform release using kernel versions v5.10 or higher are required to ship with the GKI kernel.

The following boards have enabled GKI:

- i.MX 8M Mini Board
- i.MX 8M Nano Board
- i.MX 8M Plus EVK Board
- i.MX 8M Quad EVK Board
- i.MX 8ULP EVK Board

9.2 Changes after GKI enabled

There are some changes after GKI is enabled.

- `boot.img`
  After GKI is enabled, the `boot.img` is a composite image which includes the AOSP generic kernel Image, ramdisk, and boot parameters.
  It is built from one prebuilt `boot.img`, stored in the android source code `${MY_ANDROID}/vendor/nxp/fsl-proprietary/gki/boot.img`. This `boot.img` is certified and released from AOSP, then signed with avb key to generate final `boot.img`.
  Since kernel 5.15 GKI image released by AOSP is not compatible with Android 12, it needs to be built locally with following steps:
  – download Google Released android13-5.15 GKI `Image`.
  – `cp Image {MY_ANDROID}/out/target/product/xxx/obj/KERNEL_OBJ/arch/arm64/boot/Image.lz4`
  – `TARGET_IMX_KERNEL=true make bootimage`

  - `cp {MY_ANDROID}/out/target/product/xxx/boot.img {MY_ANDROID}/vendor/nxp/fsl-proprietary/gki/boot.img`

  By default, the UUU and fastboot script flash this image.
  To build `boot.img`, run `./imx-make.sh` or `make bootimage`.

- `boot-imx.img`
  `boot-imx.img` is built from the i.MX kernel tree for debug purpose. By default, it is built out by `imx-make.sh` with `TARGET_IMX_KERNEL=true`, then renamed from `boot.img` to `boot-imx.img`. For details, see the last piece of code in the `imx-make.sh` build script.
  Note: *`boot.img` and `boot-imx.img` are generated by the `imx-make.sh` script as follows:*

```
TARGET_IMX_KERNEL=true make ${parallel_option}
 ${build_bootimage} ${build_vendorbootimage}
 ${build_dtboimage} ${build_vendorimage} || exit
if [ ${TARGET_PRODUCT} = "evk_8mp" ] || [ ${TARGET_PRODUCT} =
 "evk_8mn" ] \
|| [ ${TARGET_PRODUCT} = "evk_8ulp" ] \
|| [ ${TARGET_PRODUCT} = "evk_8mm" ] || [ ${TARGET_PRODUCT} =
 "evk_8mq" ]; then
mv ${OUT}/boot.img ${OUT}/boot-imx.img
make bootimage
fi
```

  To build `boot-imx.img`, run `./imx-make.sh` or `TARGET_IMX_KERNEL=true make bootimage && mv ${OUT}/boot.img ${OUT}/boot-imx.img`.

- Kernel `defconfig`
  Kernel `.config` is generated by one generic `gki_defconfig` along with one board specific `config`, like `imx8mm_gki.fragment`.
- Driver modules
  As GKI requires, all vendor drivers need to be built as module. Their configurations are set to `m` in above-mentioned board specific configuration file.
  In addition, explicitly install those modules on board by adding them to the following two Android predefined macros. For example, see `${MY_ANDROID}/device/nxp/imx8m/evk_8mm/SharedBoardConfig.mk`:
  - `BOARD_VENDOR_RAMDISK_KERNEL_MODULES`
    Modules under this macro are copied to `${MY_ANDROID}/out/target/product/evk_8mm/vendor_ramdisk/lib/modules`, and then built as `vendor_boot.img`.
    They are installed to the kernel in the first stage of initialization. In general, put essential modules here and be careful of the sequence.
  - `BOARD_VENDOR_KERNEL_MODULES`
    Modules under this macro are copied to `${MY_ANDROID}/out/target/product/evk_8mm/vendor/lib/modules`, and then built as `vendor.img`.
    They are installed later than `vendor_ramdisk`, after the Android file system is ready.

9.3 How to add new drivers

Perform the following steps to add new drivers (Taking i.MX 8M Mini as an example):

1. Set the driver configuration to `m` in the configuration fragment file of the board:

```
diff --git a/arch/arm64/configs/imx8mm_gki.fragment b/arch/
arm64/configs/imx8mm_gki.fragment
index 594bf1228f72..b5585c423bbf 100644
```

```
--- a/arch/arm64/configs/imx8mm_gki.fragment
+++ b/arch/arm64/configs/imx8mm_gki.fragment
@@ -109,3 +109,5 @@ CONFIG_DMABUF_IMX=m
 # CONFIG_IMX_SENTNL_MU is not set
 # CONFIG_IMX_RPMSG_TTY is not set
+CONFIG_ZRAM=m
+CONFIG_ZSMALLOC=m
```

2. Add the driver `.ko` files to the board:

```
diff --git a/imx8m/evk_8mm/SharedBoardConfig.mk b/imx8m/
evk_8mm/SharedBoardConfig.mk
index df7850b0b285..84d136c224cd 100644
--- a/imx8m/evk_8mm/SharedBoardConfig.mk
+++ b/imx8m/evk_8mm/SharedBoardConfig.mk
@@ -102,6 +104,10 @@ endif
 ifeq ($(IMX8MM_USES_GKI),true)
 BOARD_VENDOR_RAMDISK_KERNEL_MODULES +=     \
+$(KERNEL_OUT)/mm/zsmalloc.ko \
+$(KERNEL_OUT)/crypto/lzo.ko \
+$(KERNEL_OUT)/crypto/lzo-rle.ko \
+$(KERNEL_OUT)/drivers/block/zram/zram.ko \
 $(KERNEL_OUT)/drivers/soc/imx/soc-imx8m.ko \
```

Note: *If other driver modules depend on them, put them before others.*

3. Fix symbol issues.
   If some symbols are not exported but used by the added driver modules, perform the following steps to export them:
   a. Export symbols with `EXPORT_SYMBOL_GPL(xxx)`.
      Note: *If symbols are in core kernel code (which means not in loadable modules), such changes must upstream to the AOSP GKI Kernel tree.*
   b. Add symbols to the AOSP GKI Kernel tree `android/abi_gki_aarch64.xml`.

9.4 How to build GKI locally

In development stage, it is useful to build a GKI image locally to verify drivers.

1. Prepare the GKI Kernel build repo (Taking 5.15 kernel as an example):

```
mkdir gki && cd gki
repo init -u https://android.googlesource.com/kernel/manifest
 -b common-android13-5.15
repo sync
```

2. (Optional) Enable the early console.
   Early console is useful, if the system is stuck at "Starting kernel ...".
   Apply the following changes in the GKI Kernel tree: `gki/common`:

```
diff --git a/arch/arm64/configs/gki_defconfig b/arch/arm64/
configs/gki_defconfig
index 56d405007e96..d3d922d17a67 100644
--- a/arch/arm64/configs/gki_defconfig
+++ b/arch/arm64/configs/gki_defconfig
@@ -372,6 +372,7 @@ CONFIG_SERIAL_AMBA_PL011=y
 CONFIG_SERIAL_AMBA_PL011_CONSOLE=y
 CONFIG_SERIAL_SAMSUNG=y
 CONFIG_SERIAL_SAMSUNG_CONSOLE=y
+CONFIG_SERIAL_IMX_EARLYCON=y
 CONFIG_SERIAL_MSM_GENI_EARLY_CONSOLE=y
 CONFIG_SERIAL_SPRD=y
```

```
diff --git a/drivers/tty/serial/Kconfig b/drivers/tty/serial/
Kconfig
index 940db397dae2..29ef9b4bdb87 100644
--- a/drivers/tty/serial/Kconfig
+++ b/drivers/tty/serial/Kconfig
@@ -517,7 +517,6 @@ config SERIAL_IMX_CONSOLE
 config SERIAL_IMX_EARLYCON
 bool "Earlycon on IMX serial port"
-depends on ARCH_MXC || COMPILE_TEST
 depends on OF
 select SERIAL_EARLYCON
 select SERIAL_CORE_CONSOLE
```

3. Build the Kernel Image.

```
BUILD_CONFIG=common/build.config.gki.aarch64 build/build.sh
```

The GKI `Image` is built as `gki/out/android13-5.15/common/arch/arm64/boot/Image`.

4. Build Android `boot.img`:

```
cp gki/out/android13-5.15/common/arch/arm64/boot/Image
 ${MY_ANDROID}/out/target/product/evk_8mm/obj/KERNEL_OBJ/
arch/arm64/boot/Image.lz4
cd ${MY_ANDROID}
TARGET_IMX_KERNEL=true make bootimage
```

9.5 How to export new symbols

AOSP GKI image only exports those symbols listed at `android/abi_gki_aarch64.xml`. To update them, see the official document: https://source.android.com/devices/architecture/kernel/abi-monitor.

The following is a quick start guide to export new symbols.

1. Generate the device symbol list (`android/abi_gki_aarch64_imx`).

```
mkdir gki && cd gki (Make sure folder gki is not inside of
 ${MY_ANDROID})
repo init -u https://android.googlesource.com/kernel/manifest
 -b common-android13-5.15
repo sync
cd common
```

Note: *Switch kernel in this common folder from AOSP to own device kernel and apply all your local patches which may require new symbols.*

```
git remote add device <device kernel git URL>
git remote update
git checkout device/<device kernel branch>
git apply <all device patches if needed>
cd ..
(Due to ISP and wifi code is out of kernel tree, set it
 explicitly to collect their symbols)
ln -s ${MY_ANDROID}/vendor/nxp-opensource/
verisilicon_sw_isp_vvcam verisilicon_sw_isp_vvcam
ln -s ${MY_ANDROID}/vendor/nxp-opensource/nxp-mwifiex nxp-
mwifiex
BUILD_FOR_GKI=yes BUILD_CONFIG=common/build.config.imx
 EXT_MODULES_MAKEFILE="verisilicon_sw_isp_vvcam/vvcam/v4l2/
```

```
Kbuild" EXT_MODULES="nxp-mwifiex/mxm_wifiex/wlan_src" build/
build_abi.sh --update-symbol-list -j8
```

Then, `common/android/abi_gki_aarch64_imx` is updated.

2. Update the AOSP symbol list (`android/abi_gki_aarch64.xml`).

```
cd gki
cp common/android/abi_gki_aarch64_imx /tmp/
abi_gki_aarch64_imx
cd common
```

Note: *Switch the kernel in this common folder from own device kernel to the AOSP kernel.*

```
git reset --hard
git checkout aosp/android13-5.15
# (Note: the prebuilt GKI boot.img is based on
 android13-5.15-2022-06_r1 tag)
cp /tmp/abi_gki_aarch64_imx android/abi_gki_aarch64_imx
cd ..
BUILD_CONFIG=common/build.config.gki.aarch64 build/
build_abi.sh LTO=thin --update -j8
```

Then, `common/android/abi_gki_aarch64.xml` is updated.

3. Build Android `boot.img` locally.

```
cp gki/out/android13-5.15/common/arch/arm64/boot/Image \
        ${MY_ANDROID}/out/target/product/evk_8mm/obj/
KERNEL_OBJ/arch/arm64/boot/Image.lz4
cd ${MY_ANDROID}
TARGET_IMX_KERNEL=true make bootimage
# Then the boot.img built locally will export those symbols
```

4. If you want AOSP released GKI image to export these symbols, upstream the two files to AOSP:
`android/abi_gki_aarch64_imx android/abi_gki_aarch64.xml`

9.6 How to update the GKI image

Download GKI `boot.img` from Google. Put `boot.img` in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/gki/boot.img`. Run the following command to build signed `boot.img`:

```
./imx-make.sh bootimage
or
make bootimage
```

10 Revision History

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.

Revision history*...continued*

Revision number	Date	Substantive changes
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_2.1.0	04/2020	i.MX 8M Plus Alpha and i.MX 8QuadXPlus Beta release.
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.
android-10.0.0_2.3.0	07/2020	i.MX 8M Plus EVK Beta1 release, and all the other i.MX 8 GA release.
android-11.0.0_1.0.0	12/2020	i.MX 8M Plus EVK Beta release, and all the other i.MX 8 GA release.
android-11.0.0_2.0.0	04/2021	i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Plus, and i.MX 8M Quad GA release.
android-11.0.0_2.2.0	07/2021	i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Plus, and i.MX 8M Quad GA release.
android-11.0.0_2.4.0	10/2021	i.MX 8ULP EVK Alpha release, i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Plus, and i.MX 8M Quad GA release.
android-11.0.0_2.6.0	01/2022	i.MX 8ULP EVK Beta release, i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Plus, and i.MX 8M Quad GA release.
android-12.0.0_1.0.0	03/2022	i.MX 8ULP EVK Beta release, i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Plus, and i.MX 8M Quad GA release.
android-12.0.0_2.0.0	07/2022	i.MX 8ULP EVK Beta release, i.MX 8M Mini, i.MX 8M Nano, i.MX 8M Plus, and i.MX 8M Quad GA release.

11 Legal information

11.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

11.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

11.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

AUG

Android User's Guide

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

Date of release: 28 July 2022
Document identifier: AUG
Document number: