UG10176

Android Automotive User's Guide

Rev. automotive-15.0.0_2.1.0 —
14 October 2025

User guide

Document information

Information	Content
Keywords	Android, Automotive, i.MX, automotive-15.0.0_2.1.0, UG10176
Abstract	This document describes how to configure a Linux build machine and provides the steps to download, patch, and build the software components that create the Android system image when working with the sources.

1 Overview

This document provides the technical information related to the i.MX 8 and i.MX 95 devices:

- Instructions for building from sources or using pre-built images.
- Instructions for copying images to boot media.
- Hardware/software configurations for programming the boot media and running the images.

This document describes how to configure a Linux build machine and provides the steps to download, patch, and build the software components that create the Android system image when working with the sources.

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 64bit version is the most tested environment for the Android 15.0 build.

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

- 32 GB RAM
- 450 GB hard disk

Note:

- *The minimum required amount of free memory is around 24 GB, even with which, some configurations may not work. Enlarging the physical RAM capacity is a way to avoid potential build errors related to the memory.*
- *With 24 GB RAM, if you run into segfaults or other errors related to memory when building the images, try to reduce your `-j` value. In the demonstration commands in the following part of this document, the `-j` value is **4**.*

After the setup of Linux PC, check whether you have all the necessary packages installed for an Android build. See "Setting up your machine" on the Android website.

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 \
device-tree-compiler \
gdisk \
m4 \
bison \
flex make \
libssl-dev \
gcc-multilib \
libgnutls28-dev \
swig \
liblz4-tool \
libdw-dev \
dwarves \
```

```
bc cpio tar lz4 rsync \
ninja-build clang \
build-essential \
libncurses5 \
xxd \
unzip \
efitools
```

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"*
- *To build Android in the Docker container, skip the step of installing preceding packages, and refer to Section 3.3 to build the Docker image. It has the full i.MX Android build environment.*

2.2 Unpacking the Android release package

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

```
$ cd ~ (or any other directory you like)
$ tar -xzvf imx-automotive-15.0.0_2.1.0.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 GitHub 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 have the i.MX Android proprietary source code package `imx-automotive-15.0.0_2.1.0.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-automotive-15.0.0_2.1.0/imx_android_setup.sh
# By default, after the preceding command is executed, the current working
 directory changes to the i.MX Android source code root directory.
# ${MY_ANDROID} will be referred as the i.MX Android source code root directory
 in all i.MX Android release documentation.
$ export MY_ANDROID=`pwd`
```

Note:

In the `imx_android_setup.sh` script, a `.xml` file that contains the code repositories' information is specified. Code repository revision is specified with the release tag in this file. The release tag should not be moved when the code is externally released, so no matter when the `imx_android_setup.sh` is executed, the working areas of code repositories synchronized by this script are the same.

If the released code is critically fixed, another `.xml` file is created to help customers to synchronize the code. Then customers need to modify `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 "$android_builddir"
-   repo init -u https://github.com/nxp-imx/imx-manifest -b imx-android-15 -m
 imx-automotive-15.0.0_2.1.0.xml
+   repo init -u https://github.com/nxp-imx/imx-manifest -b imx-android-15 -m
 rel_automotive-15.0.0_2.1.0.xml
    rc=$?
    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 occurs, execute the following command on the host to fix it:

```
$ 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).

Execute the `source build/envsetup.sh` command to import shell functions in `${MY_ANDROID}/build/envsetup.sh`.

Execute the `lunch <BuildName-BuildMode>` command to set up the build configuration.

The `Product Name` is the Android device name found in the directory `${MY_ANDROID}/device/nxp/`. Search for the keyword `PRODUCT_NAME` under this directory for the product name.

Table 1. Build names

Build name	Description
`mek_8q_car`	i.MX 8QuadMax/8QuadXPlus MEK Board with the Exterior View System (EVS) function enabled on the Arm Cortex-M4 CPU core
`mek_8q_car2`	i.MX 8QuadMax/8QuadXPlus MEK Board with EVS function enabled on the Arm Cortex-A CPU cores (Power mode switch demo is running on the Cortex-M4 core in this configuration)
`evk_95_car`	i.MX 95 EVK Board with the Exterior View System (EVS) function enabled on the Arm Cortex-M7 CPU core.
`evk_95_car2`	i.MX 95 EVK Board with EVS function enabled on the Arm Cortex-A CPU cores (Power mode switch demo is running on the Cortex-M7 core in this configuration)

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 modes

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

After the two commands above are executed, then the build process starts. The behavior of the i.MX Android build system used to be aligned with the original Android system. The command of `make` could start the build process and all images were built out before. There are some differences now. A shell script named `imx-make.sh` is provided and its symlink file can be found under the `${MY_ANDROID}` directory. `./imx-make.sh` should be executed to start the build process.

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

Google puts a limit on the host tools used when compiling Android code from the Android 10.0 platform. 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 using prebuilt binaries for U-Boot/kernel in the Android build system. It takes some steps to build U-Boot/kernel to binaries and puts 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 perform these steps to simplify the build work. After U-Boot/kernel are compiled, any build commands in the standard Android can be used.

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

The build configuration command `lunch` can be issued with an argument `<Build name>-nxp_stable-<Build type>` string, such as `lunch mek_8q_car-nxp_stable-userdebug`, or can be issued without the argument presenting a menu of selection.

Do some preparations for the first time when building the images. A detailed example of image building steps is as follows:

1. Prepare the build environment for U-Boot and kernel.
   This step is mandatory because there is no GCC cross-compile tool chain in the AOSP codebase.
   An approach is provided to use the self-installed GCC cross-compile tool chain for both `AArch32` and `AArch64` platforms.
   First, download the tool chain for the A-profile architecture on the arm Developer GNU-A Downloads page. It is recommended to use the 12.3.Rel1 version for this release. For `AArch32` build, you can download the BareMetal target `arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi.tar.xz`. For `AArch64` build, you can download the GNU/Linux target `arm-gnu-toolchain-12.3.rel1-x86_64-aarch64-none-linux-gnu.tar.xz`.
   Then, uncompress the file into a path on the local disk. For example, to `/opt/`. Export a variable named `AARCH32_GCC_CROSS_COMPILE` and `AARCH64_GCC_CROSS_COMPILE` to point to the tool as follows:

```
# For AArch32 toolchain
$ sudo tar -xvJf arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi.tar.xz -C /
opt
$ export AARCH32_GCC_CROSS_COMPILE=/opt/arm-gnu-toolchain-12.3.rel1-x86_64-
arm-none-eabi/bin/arm-none-eabi-
# For AArch64 toolchain
```

```
$ sudo tar -xvJf arm-gnu-toolchain-12.3.rel1-x86_64-aarch64-none-linux-
gnu.tar.xz -C /opt
$ export AARCH64_GCC_CROSS_COMPILE=/opt/arm-gnu-toolchain-12.3.rel1-x86_64-
aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
```

Finally, follow the steps below to set external clang tools for kernel building.

```
$ sudo git clone --no-checkout --depth 1 https://android.googlesource.com/
platform/prebuilts/clang/host/linux-x86 /opt/android-kernel-prebuilts-6.12/
clang/host/linux-x86
$ cd /opt/android-kernel-prebuilts-6.12/clang/host/linux-x86
$ sudo git fetch --depth 1 origin 66acdd82ee62e4aaa4248f03191c59dfed9db193
$ sudo git checkout 66acdd82ee62e4aaa4248f03191c59dfed9db193

$ sudo git clone --no-checkout --depth 1 https://android.googlesource.com/
kernel/prebuilts/build-tools /opt/android-kernel-prebuilts-6.12/kernel-build-
tools
$ cd /opt/android-kernel-prebuilts-6.12/kernel-build-tools
$ sudo git fetch --depth 1 origin 3c5e4f14b451ec85167c38b917d2459687abd7f4
$ sudo git checkout 3c5e4f14b451ec85167c38b917d2459687abd7f4

$ sudo git clone --no-checkout --depth 1 https://android.googlesource.com/
platform/prebuilts/rust /opt/android-kernel-prebuilts-6.12/rust
$ cd /opt/android-kernel-prebuilts-6.12/rust
$ sudo git fetch --depth 1 origin 5156e7f81ae254c79ee736e44c960e75ad685c67
$ sudo git checkout 5156e7f81ae254c79ee736e44c960e75ad685c67

$ sudo git clone --no-checkout --depth 1 https://android.googlesource.com/
platform/prebuilts/clang-tools /opt/android-kernel-prebuilts-6.12/clang-tools
$ cd /opt/android-kernel-prebuilts-6.12/clang-tools
$ sudo git fetch --depth 1 origin 17329f6590e2872dcf04a0c96a176be089470cd9
$ sudo git checkout 17329f6590e2872dcf04a0c96a176be089470cd9

$ export KERNEL_PREBUILTS_PATH=/opt/android-kernel-prebuilts-6.12
```

The final export command can be added to `/etc/profile`. When the host boots up, `AARCH32_GCC_CROSS_COMPILE`, `AARCH64_GCC_CROSS_COMPILE` and `KERNEL_PREBUILTS_PATH` are set and can be directly used.

Note: *To build Android in the Docker container, skip the step of installing the preceding packages, and refer to Section 3.3 to build the Docker image. It has the full i.MX Android build environment.*

2. Prepare the build environment for the Arm Cortex-M4 image. Download the GCC tool chain from the Arm Developers GNU-RM Downloads page. It is recommended to download the `9-2020-q2-update` version. Extract it to your installation directory, for example, `/opt`. Then, export a variable named `ARMGCC_DIR` to point to the tool as follows:

```
$ sudo tar -jxvf gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2 -C /
opt
$ export ARMGCC_DIR=/opt/gcc-arm-none-eabi-9-2020-q2-update
```

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

Upgrade the CMake version to 3.13.0 or higher. If the CMake version on your machine is not higher than 3.13.0, you can follow the steps below 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. Change to the top-level build directory.

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

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

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

5. Execute the Android lunch command.
   In this example, the setup is for the production image of i.MX 8QuadMax/8QuadXPlus MEK Board/Platform device with EVS function enabled in the Cortex-M4 CPU core.

```
$ lunch mek_8q_car-nxp_stable-userdebug
```

6. 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.

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

The output of the `make` command is written to the standard output and `build-log.txt`. If there are 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 detailed steps, see the i.MX Android Frequently Asked Questions.

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

- `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 when booting in "recovery" mode. Not used directly.
- `dtbo-imx8qm.img`: Board's device tree binary. It is used to support the LVDS-to-HDMI display for i.MX 8QuadMax MEK.
- `dtbo-imx8qm-revd.img`: Board's device tree binary. It is used to support the LVDS-to-HDMI display for i.MX 8QuadMax MEK rev.E.
- `dtbo-imx8qm-md.img`: Board's device tree binary. It is used to support multiple-display feature for i.MX 8QuadMax MEK.
- `dtbo-imx8qm-md-revd.img`: Board's device tree binary. It is used to support multiple-display feature for i.MX 8QuadMax MEK rev.E.
- `dtbo-imx8qm-sof.img`: Board's device tree binary. It is used to support the SOF for i.MX 8QuadMax MEK.
- `dtbo-imx8qm-sof-revd.img`: Board's device tree binary. It is used to support the SOF for i.MX 8QuadMax MEK rev.E.
- `dtbo-imx8qxp.img`: Board's device tree binary. It is used to support the LVDS-to-HDMI display for i.MX 8QuadXPlus MEK.
- `dtbo-imx8qxp-sof.img`: Board's device tree binary. It is used to support the SOF for i.MX 8QuadXPlus MEK.
- `vbmeta-imx8qm.img`: Android Verify boot metadata image for `dtbo-imx8qm.img`. It is used to support the LVDS-to-HDMI display for i.MX 8QuadMax MEK.
- `vbmeta-imx8qm-revd.img`: Android Verify boot metadata image for `dtbo-imx8qm-revd.img`. It is used to support the LVDS-to-HDMI display for i.MX 8QuadMax MEK rev.E.
- `vbmeta-imx8qm-md.img`: Android Verify boot metadata image for `dtbo-imx8qm-md.img`. It is used to support the multiple-display feature for i.MX 8QuadMax MEK.

- `vbmeta-imx8qm-md-revd.img`: Android Verify boot metadata image for `dtbo-imx8qm-md-revd.img`. It is used to support the multiple-display feature for i.MX 8QuadMax MEK rev.E.
- `vbmeta-imx8qm-sof.img`: Android Verify boot metadata image for `dtbo-imx8qm-sof.img`. It is used to support the SOF feature for i.MX 8QuadMax MEK.
- `vbmeta-imx8qm-sof-revd.img`: Android Verify boot metadata image for `dtbo-imx8qm-sof-revd.img`. It is used to support the SOF feature for i.MX 8QuadMax MEK rev.E.
- `vbmeta-imx8qxp.img`: Android Verify boot metadata image for `dtbo-imx8qxp.img`. It is used to support the LVDS-to-HDMI display for i.MX 8QuadXPlus MEK.
- `vbmeta-imx8qxp-sof.img`: Android Verify boot metadata image for `dtbo-imx8qxp-sof.img`. It is used to support the SOF feature for i.MX 8QuadXPlus MEK.
- `ramdisk.img`: Ramdisk image generated from `root/`. Not directly used.
- `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. Used for 16 GB boot storage.
- `partition-table-28GB.img`: GPT partition table image. Used for 32 GB boot storage.
- `spl-imx8qm.bin`: A composite image, which includes SECO firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadMax MEK.
- `spl-imx8qm-secure-unlock.bin`: A composite image, which includes SECO firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadMax MEK. It is a demonstration of the secure unlock mechanism.
- `spl-imx8qxp.bin`: A composite image, which includes SECO firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK with silicon revision B0 chip.
- `spl-imx8qxp-secure-unlock.bin`: A composite image, which includes SECO firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK with silicon revision B0 chip. It is a demonstration of the secure unlock mechanism.
- `spl-imx8qxp-c0.bin`: A composite image, which includes SECO firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK with silicon revision C0 chip.
- `bootloader-imx8qm.img`: The next loader image after SPL. It includes the Arm trusted firmware, Trusty OS, and U-Boot proper for i.MX 8QuadMax MEK.
- `bootloader-imx8qm-secure-unlock.img`: The next loader image after SPL. It includes the Arm trusted firmware, trusty OS, and U-Boot proper for i.MX 8QuadMax MEK. It is a demonstration of the secure unlock mechanism.
- `bootloader-imx8qxp.img`: The next loader image after SPL. It includes the Arm trusted firmware, Trusty OS, and U-Boot proper for i.MX 8QuadXPlus MEK with silicon revision B0 chip.
- `bootloader-imx8qxp-secure-unlock.img`: The next loader image after SPL. It includes the Arm trusted firmware, Trusty OS, and U-Boot proper for i.MX 8QuadXPlus MEK with silicon revision B0 chip. It is a demonstration secure unlock mechanism.
- `bootloader-imx8qxp-c0.img`: The next loader image after SPL. It includes the Arm trusted firmware, Trusty OS, and U-Boot proper for i.MX 8QuadXPlus MEK with silicon revision C0 chip.
- `u-boot-imx8qm-mek-uuu.imx`: U-Boot image used by UUU for i.MX 8QuadMax MEK. It is not flashed to MMC.
- `u-boot-imx8qxp-mek-uuu.imx`: U-Boot image used by UUU for i.MX 8QuadXPlus MEK with silicon revision B0 chip. It is not flashed to MMC.
- `u-boot-imx8qxp-mek-c0-uuu.imx`: U-Boot image used by UUU for i.MX 8QuadXPlus MEK with silicon revision C0 chip. It is not flashed to MMC.
- `vendor.img`: Vendor image, which holds platform binaries. Mounted at `/vendor`.
- `boot.img`: A composite image that includes the kernel Image, ramdisk, and boot parameters.
- `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.

Note:

- *To build the U-Boot image separately, see Section 3.4.*
- *To build the kernel uImage separately, see Section 3.5.*
- *To build `boot.img`, see Section 3.6.*
- *To build `dtbo.img`, see Section 3.7.*

3.2.1 Configuration examples of building i.MX devices

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

Table 3. i.MX device lunch examples

Build name	Lunch command
i.MX 8QuadXPlus/8QuadMax MEK Board with the EVS function enabled on the Arm Cortex-M4 CPU core	`$ lunch mek_8q_car-nxp_stable-userdebug`
i.MX 8QuadMax/8QuadXPlus MEK Board with the EVS function enabled on the Arm Cortex-A CPU cores (Power mode switch demo is running on the Cortex-M4 core in this configuration)	`$ lunch mek_8q_car2-nxp_stable-userdebug`
i.MX 95 EVK Board with the EVS function enabled on the Arm Cortex-M7 CPU core	`$ lunch evk_95_car-nxp_stable-userdebug`
i.MX 95 EVK Board with the EVS function enabled on the Arm Cortex-A CPU cores (Power mode switch demo is running on the Cortex-M7 core in this configuration)	`$ lunch evk_95_car2-nxp_stable-userdebug`

3.2.2 Build mode selection

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

Note: *To pass CTS, use `user` build mode.*

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 good for user to test with greater diagnosis capabilities.

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

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

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

- `eng`: development configuration with additional debugging tools
  - Installs modules tagged with: `eng` and/or `debug` through `LOCAL_MODULE_TAGS`, or specified by `PRODUCT_PACKAGES_ENG` and/or `PRODUCT_PACKAGES_DEBUG`.
  - Installs modules according to the product definition files, in addition to tagged modules.

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

There are two methods for the build of Android image.

To build Android images, an example for the i.MX 8QuadMax/8QuadXPlus MEK with the EVS function enabled in the Cortex-M4 CPU core is:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh -j4
```

The commands below can achieve the same result:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-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 Build with the GAS package

Get the Google Automobile Services (GAS) package from Google. Put the GAS package into the `${MY_ANDROID}/vendor/partner_gas` directory. Make sure the `product.mk*` file includes the following command line:

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

Then build the images. The GAS package is then installed into the target images.

3.3 Building an Android image with Docker

The Dockerfile can be found in the directory `${MY_ANDROID}/device/nxp/common/dockerbuild/`, which sets up an Ubuntu 20.04 image ready to build the i.MX Android OS. You can use it to generate your own Docker image with the full i.MX Android build environment. The process is as follows:

1. Build the Docker image.

```
$ cd ${Dockerfile_path}
# ${Dockerfile_path} can be ${MY_ANDROID}/device/nxp/common/dockerbuild/, or
 another path that you moved the Dockerfile to.
$ docker build --no-cache --build-arg userid=$(id -u) --build-arg groupid=
$(id -g) --build-arg username=$(id -un) -t <docker_image_name> .
# <docker_image_name> can be whatever you want, such as 'android-build'.
# '.' means using the current directory as the build context, it specifies
 where to find the files for the “context” of the build on the Docker daemon.
```

2. Start up a new container and mount your Android source codes to it with the following:

```
$ docker run --privileged -it -v ${MY_ANDROID}:/home/$(id -un)/android_src
 <docker_image_name>
> cd ~/android_src; source build/envsetup.sh
> lunch mek_8q_car2-nxp_stable-userdebug
> ./imx-make.sh -j4 2>&1 | tee build-log.txt
```

3. Get the image that you want.

```
> exit
$ cd ${MY_ANDROID}/out/target/product/mek_8q
```

Note:

- *If it fails to `apt` install packages in the process of Docker image build, configure the HTTP proxy.*
  1. *Copy your host `apt.conf` with `cp /etc/apt/apt.conf ${Dockerfile_path}/apt.conf`, or create a stripped-down version.*
  2. *Refer to the related content in the Dockerfile, and remove the symbol "#" to solve the issue.*
- *If it fails to install Clang tools in the process of Docker image build, refer to the related content in Dockerfile, remove the symbol "#" and try to build it again.*
- *If you manage the Docker as a non-root user, prefix the docker command with `sudo`, such as `sudo docker build ... & sudo docker run .....`*
- *You can use the command `docker images` to see the existing Docker image and use `docker ps -a` to see the existing container. For other Docker commands, learn them from the Docker Docs website.*
- *The Android build content above takes the i.MX 8QuadMax/QuadXPlus MEK as an example. To build other board images or a single image, refer to the other sections.*

3.4 Building U-Boot images

The U-Boot images can be generated separately. For example, you can generate a U-Boot image for the i.MX 8QuadMax/8QuadXPlus MEK board with the EVS function enabled in the Arm Cortex-M4 CPU core as follows:

```
# U-Boot image for 8QuadMax/8QuadXPlus MEK board with EVS function enabled in
 the Arm Cortex-M4 CPU core
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh bootloader -j4
```

Multiple U-Boot variants are generated for different purposes. You can check `{MY_Android}/device/nxp/imx8q/mek_8q/UbootKernelBoardConfig.mk` for more details. The following table lists the U-Boot configurations and images for the `lunch` target `mek_8q_car-nxp_stable-userdebug`. Similar variants are generated for the i.MX 95 EVK board. You can check `{MY_Android}/device/nxp/imx9/evk_95/UbootKernelBoardConfig.mk` for more details.

Table 4. U-Boot configurations and images

SoC	U-Boot configurations	Generated images	Description
i.MX 8QuadMax	`imx8qm_mek_androidauto_trusty_defconfig`	`spl-imx8qm.bin`, `bootloader-imx8qm.img`	Default i.MX 8QuadMax Android Auto image.
i.MX 8QuadMax	`imx8qm_mek_androidauto_trusty_secure_unlock_defconfig`	`spl-imx8qm-secure-unlock.bin`, `bootloader-imx8qm-secure-unlock.img`	i.MX 8QuadMax Android Auto image with secure unlock feature enabled. For more details about secure unlock, see Section "Secure unlock" in the *i.MX Android Security User's Guide* (UG10158).
i.MX 8QuadXPlus B0 chip	`imx8qxp_mek_androidauto_trusty_defconfig`	`spl-imx8qxp.bin`, `bootloader-imx8qxp.img`	Default i.MX 8QuadXPlus B0 chip Android Auto image
i.MX 8QuadXPlus C0 chip	`imx8qxp_mek_androidauto_trusty_defconfig`	`spl-imx8qxp-c0.bin`, `bootloader-imx8qxp-c0.img`	Default i.MX 8QuadXPlus C0 chip Android Auto image
i.MX 8QuadXPlus B0 chip	`imx8qxp_mek_androidauto_trusty_secure_unlock_defconfig`	`spl-imx8qxp-secure-unlock.bin`, `bootloader-imx8qxp-secure-unlock.img`	i.MX 8QuadXPlus B0 chip Android Auto image with secure unlock feature enabled. For more details about secure unlock, see Section "Secure unlock" in the *i.MX Android Security User's Guide* (UG10158).
i.MX 8QuadMax	`imx8qm_mek_android_uuu_defconfig`	`u-boot-imx8qm-mek-uuu.imx`	U-Boot image aims to flash images for i.MX 8QuadMax. This should not be shipped to end users.
i.MX 8QuadXPlus B0 chip	`imx8qxp_mek_android_uuu_defconfig`	`u-boot-imx8qxp-mek-uuu.imx`	U-Boot image aims to flash images for i.MX 8QuadXPlus B0 chip. This should not be shipped to end users.
i.MX 8QuadXPlus C0 chip	`imx8qxp_mek_android_uuu_defconfig`	`u-boot-imx8qxp-mek-c0-uuu.imx`	U-Boot image aims to flash images for i.MX 8QuadXPlus C0 chip. This should not be shipped to end users.

3.5 Building a kernel image

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

To build out the kernel image independently from the default Android build command:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh kernel -j4
```

With a successful build in the use case above, the generated kernel image is: `${MY_ANDROID}/out/target/product/mek_8q/obj/KERNEL_OBJ/arch/arm64/boot/Image`.

3.6 Building boot.img

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

```
# Boot image for i.MX 8QuadMax/8QuadXPlus MEK board with EVS function enabled in
 the Arm Cortex-M4 CPU core
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh bootimage -j4
```

The following commands can achieve the same result:

```
# Boot image for i.MX 8QuadMax/8QuadXPlus MEK board with EVS function enabled in
 the Arm Cortex-M4 CPU core
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh kernel -j4
$ make bootimage -j4
```

3.7 Building dtbo.img

DTBO image holds the device tree binary of the board.

The following commands are used to generate `dtbo.img` under the Android environment:

```
# dtbo image for i.MX 8QuadMax/8QuadXPlus MEK board with EVS function enabled in
 the Arm Cortex-M4 CPU core
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh dtboimage -j4
```

The following commands can achieve the same result:

```
# dtbo image for i.MX 8QuadMax/8QuadXPlus MEK board with EVS function enabled in
 the Arm Cortex-M4 CPU core
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh kernel -j4
$ make dtboimage -j4
```

4 Running the Android Platform with a Prebuilt Image

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

Table 5. Image packages

Image package	Description
`automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz`	Prebuilt image for the i.MX 8QuadXPlus/8QuadMax MEK board with EVS function enabled in the Arm Cortex-M4 CPU core, which includes NXP extended features.

Table 5. Image packages*...continued*

Image package	Description
android_automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz	Prebuilt image and UUU script files for the i.MX 8QuadMax/8QuadXPlus MEK board without EVS function enabled in the Arm Cortex-M4 CPU core, which includes NXP extended features.
android_automotive-15.0.0_2.1.0_image_95evk_car.tar.gz	Prebuilt image for the i.MX 95 EVK board with EVS function enabled in the Arm Cortex-M7 CPU core, which includes NXP extended features.
android_automotive-15.0.0_2.1.0_image_95evk_car2.tar.gz	Prebuilt image and UUU script files for the i.MX 95 EVK board without EVS function enabled in the Arm Cortex-M7 CPU core, which includes NXP extended features.

The following tables list the detailed contents of the android_automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz image package.

Table 6. Images for the i.MX 8QuadXPlus and i.MX 8QuadMax MEK boards

i.MX 8QuadXPlus/8QuadMax MEK image	Description
spl-imx8qm.bin	The secondary program loader (SPL) for the i.MX 8QuadMax MEK board.
spl-imx8qm-secure-unlock.bin	The secondary program loader (SPL) with Trusty and secure unlock related configurations for the i.MX 8QuadMax MEK board.
spl-imx8qxp.bin	The secondary program loader (SPL) for the i.MX 8QuadXPlus MEK board with silicon revision B0 chip.
spl-imx8qxp-secure-unlock.bin	The secondary program loader (SPL) with Trusty and secure unlock related configurations for the i.MX 8QuadXPlus MEK board with silicon revision B0 chip.
spl-imx8qxp-c0.bin	The secondary program loader (SPL) for the i.MX 8QuadXPlus MEK board with silicon revision C0 chip.
bootloader-imx8qm.img	The next loader image after SPL for the i.MX 8QuadMax MEK board.
bootloader-imx8qm-secure-unlock.img	The next loader image after SPL for the i.MX 8QuadMax MEK board, including the Arm trusted firmware, Trusty OS, and U-Boot proper.
bootloader-imx8qxp.img	The next loader image after SPL for the i.MX 8QuadXPlus MEK board with silicon revision B0 chip.
bootloader-imx8qxp-secure-unlock.img	The next loader image after SPL for the i.MX 8QuadXPlus MEK board with silicon revision B0 chip, including the Arm trusted firmware, Trusty OS, and U-Boot proper.
bootloader-imx8qxp-c0.img	The next loader image after SPL for the i.MX 8QuadXPlus MEK board with silicon revision C0 chip.
u-boot-imx8qm-mek-uuu.imx	Bootloader used by UUU for the i.MX 8QuadMax MEK board. It is not flashed to MMC.
u-boot-imx8qxp-mek-uuu.imx	The bootloader used by UUU for the i.MX 8QuadXPlus MEK board with silicon revision B0 chip. It is not flashed to MMC.
u-boot-imx8qxp-mek-c0-uuu.imx	The bootloader used by UUU for the i.MX 8QuadXPlus MEK board with silicon revision C0 chip. It is not flashed to MMC.
partition-table.img	GPT table image for 16 GB boot storage
partition-table-28GB.img	GPT table image for 32 GB boot storage
vbmeta-imx8qm.img	Android Verify Boot metadata image for the i.MX 8QuadMax MEK board to support LVDS-to-HDMI display

Table 6. Images for the i.MX 8QuadXPlus and i.MX 8QuadMax MEK boards*...continued*

i.MX 8QuadXPlus/8QuadMax MEK image	Description
vbmeta-imx8qm-md.img	Android Verify Boot metadata image for the i.MX 8QuadMax MEK board to support the multiple-display feature.
vbmeta-imx8qm-sof.img	Android Verify Boot metadata image for the i.MX 8QuadMax MEK board to support the SOF DSP feature.
vbmeta-imx8qxp.img	Android Verify Boot metadata image for the i.MX 8QuadXPlus MEK board to support LVDS-to-HDMI display
vbmeta-imx8qxp-sof.img	Android Verify Boot metadata image for the i.MX 8QuadXPlus MEK board to support the SOF DSP feature.
system.img	System Boot image
system_ext.img	System extension image.
vendor.img	Vendor image, which holds platform binaries. Mounted at /vendor.
vendor_dlkm.img	Vendor DLKM image, which holds a dynamically loadable kernel module. Mounted at /vendor_dlkm.
product.img	Product image.
dtbo-imx8qm.img	Device tree image for the i.MX 8QuadMax
dtbo-imx8qm-md.img	Device tree image for the i.MX 8QuadMax to support the multiple-display feature.
dtbo-imx8qm-sof.img	Device tree image for the i.MX 8QuadMax to support the SOF DSP feature.
dtbo-imx8qxp.img	Device tree image for the i.MX 8QuadXPlus
dtbo-imx8qxp-sof.img	Device tree image for the i.MX 8QuadXPlus to support the SOF DSP feature.
boot.img	A composite image, which includes the AOSP generic kernel image and boot parameters.
init_boot.img	Generic ramdisk.
vendor_boot.img	A composite image, which includes vendor ramdisk and boot parameters.
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_automotive-15.0.0_2.1.0_image_95evk_car2.tar.gz image package.

Table 7. Images for i.MX 95 EVK board

i.MX 95 EVK/Verdin image	Description
spl-imx95.bin	The secondary program loader (SPL) for the i.MX 95 EVK board.
bootloader-imx95.img	The next loader image after SPL for the i.MX 95 EVK board.
u-boot-imx95-evk-uuu.imx	Bootloader used by UUU for the i.MX 95 EVK board. It is not flashed to MMC.
partition-table.img	GPT table image for 16 GB boot storage.
partition-table-28GB.img	GPT table image for 32 GB boot storage.
vbmeta-imx95.img	Android Verify Boot metadata image for the i.MX 95 EVK board to support MIPI-to-HDMI display.

Table 7. Images for i.MX 95 EVK board*...continued*

i.MX 95 EVK/Verdin image	Description
`vbmeta-imx95-lvds0.img`	Android Verify Boot metadata image for the i.MX 95 EVK board to support LVDS-to-HDMI display.
`vbmeta-imx95-mipi-lvds1.img`	Android Verify Boot metadata image for the i.MX 95 EVK board to support MIPI-to-HDMI and LVDS-to-HDM displays (multiple displays).
`system.img`	System Boot image.
`system_ext.img`	System extension image.
`vendor.img`	Vendor image, which holds platform binaries. Mounted at `/vendor`.
`vendor_dlkm.img`	Vendor DLKM image, which holds a dynamically loadable kernel module. Mounted at `/vendor_dlkm`.
`product.img`	Product image.
`dtbo-imx95.img`	Device tree image for the i.MX 95 EVK board to support MIPI-to-HDMI display.
`dtbo-imx95-ap1302.img`	Device tree image for the i.MX 95 EVK board with Car type image to support MIPI-to-HDMI display and AP1302 camera.
`dtbo-imx95-lvds0.img`	Device tree image for the i.MX 95 EVK board with Car2 type image to support LVDS-to-HDMI display.
`dtbo-imx95-mipi-lvds1.img`	Device tree image for the i.MX 95 EVK board to support MIPI-to-HDMI and LVDS-to-HDM displays (multiple displays).
`dtbo-imx95-mipi-lvds1-ap1302.img`	Device tree image for the i.MX 95 EVK board with Car type image to support MIPI-to-HDMI and LVDS-to-HDM displays (multiple displays) and the AP1302 camera.
`imx95-verdin.img`	Device tree image for the i.MX 95 Verdin EVK board to support HDMI display (LT8912).
`imx95-verdin-ap1302.img`	Device tree image for the i.MX 95 Verdin EVK board with Car type image to support HDMI display (LT8912) and RPI-CAM-MIPI camera (AP1302).
`imx95-verdin-adv7535.img`	Device tree image for the i.MX 95 Verdin EVK board to support MIPI-to-HDMI display (ADV7535).
`imx95-verdin-adv7535-ap1302.img`	Device tree image for the i.MX 95 Verdin EVK board with Car type image to support MIPI-to-HDMI display (ADV7535) and the RPI-CAM-MIPI camera (AP1302).
`boot.img`	A composite image, which includes the AOSP generic kernel image and boot parameters.
`init_boot.img`	Generic ramdisk.
`vendor_boot.img`	A composite image, which includes vendor ramdisk and boot parameters.
`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.

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

5 Programming Images

The images from the prebuilt release package or created from source code contain the U-Boot bootloader , system image, gpt image, vendor image, and vbmeta image. At a minimum, the storage devices on the NXP development system (eMMC) must be programmed with the U-Boot bootloader. The i.MX 8 and i.MX 9 series boot process determines what storage device to access based on the Boot 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 6.

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

- UUU to download all images to the eMMC storage.
- `fastboot_imx_flashall` script to download all images to the eMMC storage.

5.1 System on eMMC

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

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

- Secondary program loader image: `spl.bin`
- Android bootloader image: `bootloader.img`
- 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 vendor image: `vendor.img`
- Android vendor dynamically loadable kernel module image: `vendor_dlkm.img`
- Android Verify boot metadata image: `vbmeta.img`

5.1.1 Storage partitions

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

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

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

Table 8. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	`bootloader0`	0 KB (i.MX 8Quad Max, i.MX 8Quad XPlus C0) or 32KB (i.MX 8QuadXPlus B0)	4 MB	N/A	`spl.bin`
1	`bootloader_a`	8 MB	4 MB	N/A	`bootloader.img`

Table 8. Storage partitions*...continued*

Partition type/index	Name	Start offset	Size	File system	Content
2	bootloader_b	Follow bootloader_a	4 MB	N/A	bootloader.img
3	dtbo_a	Follow bootloader_b	4 MB	N/A	dtbo.img
4	dtbo_b	Follow dtbo_a	4 MB	N/A	dtbo.img
5	boot_a	Follow dtbo_b	64 MB	boot.img format, a kernel + recovery ramdisk	boot.img
6	boot_b	Follow boot_a	64 MB	boot.img format, a kernel + recovery ramdisk	boot.img
7	vendor_boot_a	Follow boot_a	64 MB	Part of recovery ramdisk	vendor_boot.img
8	vendor_boot_b	Follow boot_b	64 MB	Part of recovery ramdisk	vendor_boot.img
9	misc	Follow boot_ b	4 MB	N/A	For recovery storage bootloader message, reserve
10	metadata	Follow misc	16 MB	N/A	Metadata of OTA update, remount, etc.
11	presistdata	Follow metadata	1 MB	N/A	Option to operate lock \unlock
12	super.img	Follow presistdata	4096 MB	N/A	system.img, system_ext.img, vendor.img, vendor_dlkm.img, and product.img
13	userdata	Follow super	Remained space	EXT4. Mount at /data	Application data storage for system application, and for internal media partition, in the /mnt/sdcard/ directory
14	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock/ unlock
15	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
16	vbmeta_b	Follow vbmeta_a	1 MB	N/A	For storing the verify boot's metadata

Partitions are created by UUU utility, burning Android automotive images (by partition.img). Using UUU is described in the *Android Quick Start Guide* (AQSUG).

5.1.2 Downloading images with UUU

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

There is another tool of `fastboot_imx_flashall` script, which uses fastboot to flash the Android System Image into the board. It requires the target board to be able to enter fastboot mode and the device is unlocked. There is no need to change the boot mode with this `fastboot_imx_flashall` script.

The table below lists the `fastboot_imx_flashall` scripts.

Table 9. 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 `fasboot_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 line of command.

With the virtual A/B feature enabled, your host fastboot tool version should be equal to or greater than 30.0.4. You can download the host fastboot tool from the Android website or you can build it with the Android project. Based on Section 3, which describes how to build Android images, perform the following steps to build fastboot:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ 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 as 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
      -m                Flashes the Cortex-M4 image.
      -u uboot_feature  Flashes U-Boot or SPL&bootloader images with
 "uboot_feature" in their names. For QXP C0 revision please use -u c0.
                              For Android Automotive:
                                  Only dual-bootloader feature is supported. By
 default, SPL&bootloader image is flashed.
                                  For i.MX 8QuadXPlus C0 revision, use the -u c0
 parameter.
      -d dtb_feature    Flashes dtbo, vbmeta, and recovery image file with
 "dtb_feature" in their names.
                              If it is not set, use the 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 is connected to computer, it does not
 need to use this option
```

Note:

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

Example:

```
sudo ./fastboot_imx_flashall.sh -f imx8qm -a -e -D /imx_android-15.0/mek_8q_car/
```

Option explanations:

- `-f imx8qm`: Flashes images for the i.MX 8QuadMax MEK Board.
- `-a`: Only flashes slot a.
- `-e`: Erases user data after all image files are flashed.
- `-D /imx_android-15.0/mek_8q_car/`: Images to be flashed are in the directory of `/imx_android-15.0/mek_8q_car/`.

5.1.4 Downloading a single image with fastboot

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

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

Table 10. Partition categories

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

Note:

Logical partitions only exist if the dynamic partition feature is enabled.

To enter U-Boot fastboot mode, for example, you can first make the board enter U-Boot command mode, and then 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 the 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
```

The `adb` binary for the Microsoft Windows host can be obtained from https://dl.google.com/android/repository/platform-tools-latest-windows.zip. The `adb` binary for the GNU/Linux host can be installed through a packaging manager of your distribution. The following is an example for Ubuntu distribution:

```
$ sudo apt-get install adb
```

To use the fastboot tool on the host to operate on a specific partition, choose the proper fastboot implemented on the device that can recognize the partition to be operated on. For example, images in `automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz` have dynamic partition feature enabled. To flash `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.

6.1 Booting from eMMC

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

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

Table 11. Boot switch settings for i.MX 8QuadMax

i.MX 8QuadMax boot switch	download Mode (UUU mode)	eMMC boot
SW2 Boot_Mode (1-6 bit)	001000	000100

Table 12. Boot switch settings for i.MX 8QuadXPlus

i.MX 8QuadXPlus boot switch	download Mode (UUU mode)	eMMC boot
SW2 Boot_Mode (1-4 bit)	1000	0100

Boot from eMMC

Change the board Boot_Mode switch to 000100 (1-6 bit) for i.MX 8QuadMax.

Change the board Boot_Mode switch to 0100 (1-4 bit) for i.MX 8QuadXPlus.

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

Note:

`bootargs` is an optional setting for `boota`. The `boot.img` includes a default `bootargs`, which will be used if there is no `bootargs` defined in U-Boot.

6.1.2 Booting from eMMC on the i.MX 95 EVK board

The following table lists the boot switch settings to control the boot storage.

Table 13. Boot switch settings for i.MX 95

i.MX 95 boot switch	Download mode (UUU mode)	eMMC boot
SW7 Boot_Mode (1-4 bit)	1001	1010

Boot from eMMC

Change the board Boot_Mode switch to 1010 (1-4 bit).

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

Note:

`bootargs` 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.

6.2 Boot-up configurations

This section describes 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 variable 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 variable, it uses the default `bootargs` inside the image. If you have the environment variable, it is then used.
  To use the default environment in `boot.img`, use the following command to clear the `bootargs` environment variable.

  ```
  > setenv bootargs
  ```

  If the environment variable `append_bootargs` is set, the value of `append_bootargs` is appended to `bootargs` automatically.
- `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 the system, execute the following command:

  ```
  U-Boot=> boota
  ```

  To boot into recovery mode, execute the following command:

  ```
  U-Boot=> 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 14. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
`console`	Where to output kernel log by printk.	`console=ttymxc0`	i.MX 8QuadMax MEK uses `console=ttyLP0`.
`init`	Informs the kernel where the `init` file is located.	`init=/init`	All use cases. `init` in the Android platform is located in `/` instead of in `/sbin`.
`androidboot.console`	The Android shell console. It should be the same as `console=`.	`androidboot.console=ttymxc0`	To use the default shell job control, such as Ctrl+C to terminate a running process, set this for the kernel.
`cma`	CMA memory size for GPU/VPU physical memory allocation.	`cma=1184M@0x960M-0xe00M`	The start address is 0x9600 0000 and the end address is 0xDFFFFFFF. The CMA size can be configured to other value, but cannot exceed 1184 MB, because the Cortex-M core also allocates the memory from CMA, and Cortex-M cannot use the memory larger than 0xDFFFFFFF.
`androidboot.selinux`	Argument to disable SELinux check. For details about SELinux, see Security-Enhanced Linux in Android.	`androidboot.selinux=permissive`	Setting this argument also bypasses all the SELinux rules defined in the Android system. It is recommended to set this argument for internal developers.
`androidboot.fbTileSupport`	It is used to enable the framebuffer super tile output.	`androidboot.fbTileSupport=enable`	-
`firmware_class.path`	It is used to set the Wi-Fi firmware path.	`firmware_class.path=/vendor/firmware`	-
`androidboot.wificountrycode=US`	It is used to set the Wi-Fi country code. Different countries use different Wi-Fi channels. For details, see the i.MX Android Frequently Asked Questions.	`androidboot.wificountrycode=US`	-
`transparent_hugepage`	It is used to change the `sysfs` boot time defaults of Transparent Hugepage support.	`transparent_hugepage=never/always/madvise`	-
`galcore.contiguousSize`	It is used to configure the GPU reserved memory.	`galcore.contiguousSize=33554432`	It is 128 MB by default. i.MX 8QuadMax/8QuadXPlus automatically configure it to 32 MB to shorten the GPU driver initialization time.
`androidboot.vendor.sysrq`	It is used to enable `sysrq`.	`androidboot.vendor.sysrq=1`	-

6.2.3 DM-verity configuration

DM-verity (device-mapper-verity) provides transparent integrity checking of block devices. It can prevent the 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. Enable **Developer mode**. click 7 times on the **Settings** -> **About** -> **Build number** menu.
   c. Choose **Settings** -> **Developer Options** -> **OEM Unlocking** to enable OEM unlocking.
   d. Execute the following command on the target side to make the board enter fastboot mode:

```
reboot bootloader
```

   e. Unlock the device. Execute the following command on the host side:

```
fastboot oem unlock
```

   f. 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

This section provides an example for the i.MX 8QuadMax/8QuadXPlus MEK Board with EVS function enabled in the Arm Cortex-M4 CPU core to build and implement OTA update.

For other platforms, use "`lunch `" to set up the build configuration. For detailed build configuration, see Section 3.2.

7.1 Building OTA update packages

7.1.1 Building target files

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

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-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:

```
${MY_ANDROID}/out/target/product/mek_8q_car/obj/PACKAGING/
target_files_intermediates/mek_8q_car-target_files-${date}.zip
```

7.1.2 Building a full update package

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

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

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-nxp_stable-userdebug
$ ./imx-make.sh bootloader kernel -j4
$ make otapackage -j4
```

Note:

The command line `$ make otapackage -j4` is used for i.MX 8QuadMax. For i.MX 8QuadXPlus, use the command line `make OTA_TARGET=8qxp otapackage -j4`.

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

```
${MY_ANDROID}/out/target/product/mek_8q_car/mek_8q_car-ota-${date}.zip
```

`mek_8q_car-ota-${date}.zip` includes `payload.bin` and `payload_properties.txt`. The two files are used for full update.

Note:

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

7.1.3 Building an incremental update package

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

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

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

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

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

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

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

7.2 Implementing OTA update

7.2.1 Using update_engine_client to update the Android platform

`update_engine_client` is a pre-built tool to support A/B (seamless) system updates. It supports updating system from a remote server or board's storage.

To update the 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. Log in to the ADB shell and execute the following command to update:

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

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

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

1. Unzip `full-ota.zip` or `incremental_ota.zip` (Generated on Section 7.1.2 and Section 7.1.3) to get `payload.bin` and `payload_properties.txt`.
2. Push `payload.bin` to board's storage:

```
adb root
adb push payload.bin /data/ota_package
```

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 in board's console to update:

```
update_engine_client --payload=file:///data/ota_package/payload.bin --update
 --headers="FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ/de8Dgp9zFXt8Fo
+Hxccp465uTOvKNsteWU=
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.

Note:

Make sure that the -- header equals to the exact content of `payload_properties.txt`. No more "space" or "return" characters.

7.2.2 Using a customized application to update the Android platform

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

1. Generate a JSON configuration file from the OTA package.

```
out/host/linux-x86/bin/gen_update_config \
--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 an incremental OTA JSON file:

```
out/host/linux-x86/bin/gen_update_config \
--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 your 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.
   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
```

   Then 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. Their brief description is as follows:

```
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, and then click the **APPLY** button to do the update. After the update is complete, you can see "SUCCESS" in the **Engine error** text field, and "REBOOT_REQUIRED" in the **Updater state** text field. Then, reboot the board to finish the whole OTA update.

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

```
TARGET_BOARD_DTS_CONFIG := imx8qm:imx8qm-mek-car.dtb imx8qxp:imx8qxp-mek-car.dtb
```

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

```
BOARD_PREBUILT_DTBOIMAGE := out/target/product/mek_8q/dtbo-imx8qm.img
```

Therefore, the default OTA package can only be applied for the `mek_8qm` board.To generate an OTA package for `mek_8qxp`, modify this `BOARD_PREBUILT_DTBOIMAGE` as follows:

```
BOARD_PREBUILT_DTBOIMAGE := out/target/product/mek_8q/dtbo-imx8qxp.img
```

The OTA package includes the bootloader image, which is specified by the following variable in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk`:

```
BOARD_OTA_BOOTLOADERIMAGE := out/target/product/mek_8q/obj/UBOOT_COLLECTION/
bootloader-imx8qm.img
```

To generate an OTA package for `mek_8qxp`, modify `BOARD_OTA_BOOTLOADERIMAGE` as follows:

```
BOARD_OTA_BOOTLOADERIMAGE := out/target/product/mek_8q/obj/UBOOT_COLLECTION/
bootloader-imx8qxp.img
```

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

For 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

Exterior View System (EVS) is supported in i.MX Android Automotive release. This feature supports a fastboot camera, which starts camera within 1 second when the board is powered on.

This section describes how this feature is implemented and how the interfaces are used to control the EVS function. This can help customers to do customization work on the EVS function.

8.1.1 Switching between camera models on i.MX 95 EVK

The default `evk_95` and `verdin` images use OX03C10 cameras, which do not support the fast boot camera feature. Instead, the AP1302 (`RPI_CAM_MIPI`) is required for this feature. This section describes how to set up the i.MX 95 device for the AP1302 camera, as it is required to download the AP1302 firmware and flash it along with the Device Tree configured for AP1302 and a Linux kernel boot parameter.

8.1.1.1 Obtaining the AP1302 firmware

The AP1302 camera third-party firmware is not a part of this release. Therefore, it needs to be downloaded manually and added into both Android OS and Cortex-M7 application, the `rear_view_camera_sm` demo, which requires compilation of the bootloader.

To get the AP1302 firmware into the EVK board running the Android Automotive image, perform the following steps:

1. Obtain the AP1302 firmware: ap1302_60fps_ar0144_27M_2Lane_awb_tuning.bin.
2. Rename the firmware `ap1302_60fps_ar0144_27M_2Lane_awb_tuning.bin` to `ap130x_ar0144_single_fw.bin`.

```
mv ap1302_60fps_ar0144_27M_2Lane_awb_tuning.bin ap130x_ar0144_single_fw.bin
```

3. Remount the file systems on the device to get the write permission on the vendor partition using ADB. When the remount command is run for the first time, the device needs to be rebooted for the command to work.

```
adb root
adb remount
# Reboot device if ADB request reboot. Then send the remount command again:
adb reboot
adb root
adb remount
```

4. Upload the firmware and restart the EVK:

```
adb push ap130x_ar0144_single_fw.bin /vendor/firmware
adb reboot
```

5. Add the AP1302 firmware into the fastboot camera demo `rear_view_camera_sm`. Run this command in the root of Android sources, the same folder in which the `lunch` and `imx_make.sh` commands are executed.

```
xxd -i ap130x_ar0144_single_fw.bin > vendor/nxp/mcu-sdk-auto/SDK_EVK-MIMX95/
boards/\
```

```
imx95lpd5evk19/demo_apps/rear_view_camera_sm/cm7/
ap130x_ar0144_single_fw_bin.c
```

6. Make the generated variables in `const` of `ap130x_ar0144_single_fw_bin.c`.
   a. In a text editor, open the file `vendor/nxp/mcu-sdk-auto/SDK_EVK-MIMX95/boards/imx95lpd5evk19/demo_apps/rear_view_camera_sm/cm7/ap130x_ar0144_single_fw_bin.c`.
   b. Add `const` qualifier to the `ap130x_ar0144_single_fw_bin` and `ap130x_ar0144_single_fw_bin_len` variables, and then save the file. The result looks like this:

```
const unsigned char ap130x_ar0144_single_fw_bin[] = {
    0x17, 0x23, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, ...
}
const unsigned int ap130x_ar0144_single_fw_bin_len = 80528;
```

7. Rebuild the bootloader and flash it into the i.MX 95 EVK.
   a. Set up the build environment as described in Section 3.4.
   b. Rebuild the bootloader.

```
./imx-make.sh -j14 bootloader
```

   c. Set the EVK board into fastboot mode. For details, see Section 5.

```
# In U-Boot console
fastboot 0
```

   d. Flash the rebuilt bootloader images.

```
cd out_95/target/product/evk_95
sudo fastboot flash bootloader0 spl-imx95.bin
sudo fastboot flash bootloader_a bootloader-imx95.img
```

8. Having firmware ready for the camera, configure the Android platform to use it. This is described in Section 8.1.1.2.

8.1.1.2 Change camera type in Device-Tree and U-Boot

By default, the i.MX 95 image is configured for the OX03C10 camera.

To use AP1302, perform the following steps:

1. Flash the `ap1302` Device Tree Blob that you select into `evk_95` (for example, `dtbo-imx95-ap1302.img`):

```
fastboot flash dtbo dtbo-imx95-ap1302.img
```

   Note: *List of Device Tree configurations could be found in Section 4.*

2. Update `boot_args`. In U-Boot command mode, run the following command:

```
setenv append_bootargs androidboot.camera.layout=ap1302
saveenv
boot
```

To switch back to OX03C10, perform the following steps:

1. Flash the Device Tree Blob that you select without `ap1302` in the name (for example, `dtbo-imx95.img`):

```
fastboot flash dtbo dtbo-imx95.img
```

2. Update `boot_args`. In U-Boot command mode, run the following command:

```
setenv append_bootargs
```

```
saveenv
boot
```

8.1.2 Interfaces to control the EVS function

8.1.2.1 Starting the EVS function with images in `automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz`

With images in `automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz`, the Arm Cortex-A core runs Android Automotive system and the Arm Cortex-M core runs RTOS collaborate to realize this EVS function. The work sequence chart of EVS is shown in the following figure. It starts with the board power on.

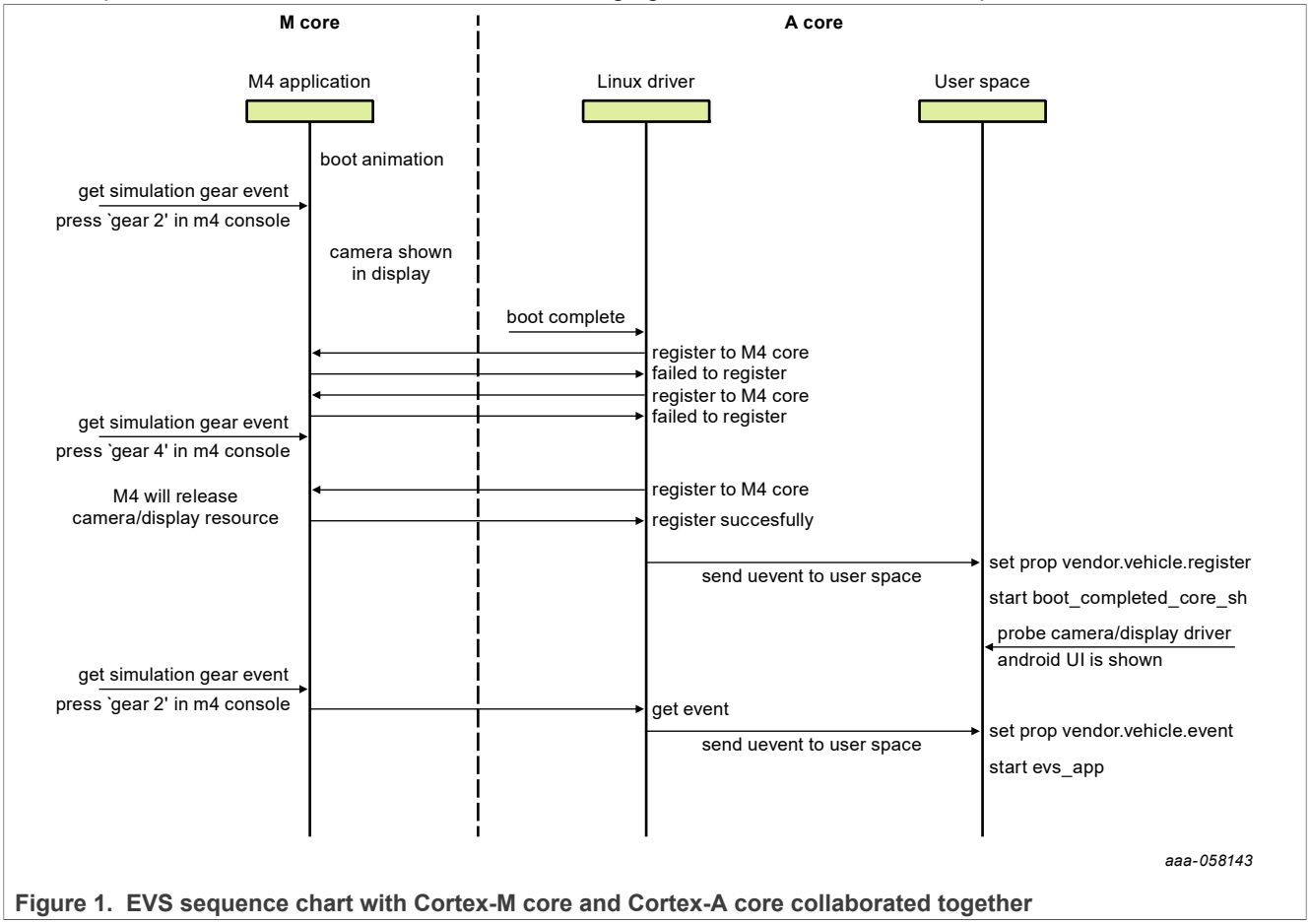

Figure 1. EVS sequence chart with Cortex-M core and Cortex-A core collaborated together

Rear view camera (RVC) is only supported in Android cars. The following is the registration process of the vehicle client.

1. Set `vendor.all.system_server.ready` to **1** in `frameworks/base/packages/SystemUI/src/com/android/systemui/SystemUIApplication.java`.
2. Write **1** to `/sys/devices/platform/vehicle_rpmsg_m4/register` in AP. Register the RPMsg client to the Cortex-M side.
3. Cortex-M4 releases camera/display resource and sends Response of RPMsg client register. If the registration status is successful, go to Step 5; otherwise, go to Step 4.
4. AP gets state values `VEHICLE_GEAR` and `register_ready`.
5. Send `extcon_set_state_sync` to `evs_service` in AP. `vendor.vehicle.register` is then set.
6. Start `boot_completed_core_sh`, which probes the display/camera modules.

i.MX 8QuadMax MEK and i.MX 8QuadXPlus MEK both support single-rearview camera. To start the single-rearview camera:

1. Connect the camera as described in the *Android Automotive Quick Start Guide* (UG10177).
2. Open the Cortex-M4 console.
   Cortex-M4 console on the i.MX 8QuadXPlus MEK board: USB-to-UART port has two consoles. One is a Cortex-A core console, and the other one is a Cortex-M4 console.
   Cortex-M4 console on i.MX 8QuadMax MEK board: RS-232 port on the base board.
3. Input `gear 2` on the Cortex-M4 console when the board is powered on and Android Automotive running on Cortex-A core is not fully booted. The rearview camera appears on the screen.
   Input `gear 4` when Android Automotive is fully booted. The Android UI appears on the screen.
4. Input `gear 2` on the Cortex-M4 console after Android system boot is complete. The rearview camera appears on the screen.
   Input `gear 4` on the Cortex-M4 console. The Android UI appears on the screen.

Note:

- *Inputting `gear 2` on the Cortex-M4 console indicates that the Cortex-M4 core gets the reverse signal.*
- *Inputting `gear 4` on the Cortex-M4 console indicates that the Cortex-M4 core gets the drive signal.*

i.MX 8QuadMax MEK and i.MX 8QuadXPlus MEK with silicon revision C0 chip also support multiple EVS cameras. The relationship between the orientation of cameras and hardware connection is shown as follows.

Table 15. Relationship between the orientation of cameras and hardware connection

Hardware connection	Camera orientation
IN0	Rear
IN1	Front
IN2	Right
IN3	Left

The logic to handle the vehicle information is shown with the following pseudo code:

```
if (gear state == reverse)
    show rear camera view
else if (turn signal == right)
    show right camera view
else if (turn signal == left)
    show left camera
else if (gear state == park)
    show all cameras' view
else
    show no camera view
```

The meaning of the commands input on the Cortex-M4 console is as follows.

Table 16. Meaning of commands input on the Cortex-M4 console

Command	Meaning
`turn 0`	Not turn
`turn 1`	Turn right
`turn 2`	Turn left
`gear 1`	Park
`gear 2`	Reverse

Table 16. Meaning of commands input on the Cortex-M4 console*...continued*

Command	Meaning
`gear 4`	Drive

To start the multiple-EVS-camera function:

1. Input `su && start evs_app` on the AP console to start `evs_app`. You can also start the rearview camera on the Cortex-M4 console with gear 2. The display should be rear camera view.
2. Input gear 1 on the Cortex-M4 console. All cameras must be connected to the board.

Figure 2. All cameras' views on the display

3. Input `turn 1` on the Cortex-M4 console. It shows the right camera view on the display.
4. Input `turn 2` on the Cortex-M4 console. It shows the left camera view on the display.
5. Input `turn 0` on the Cortex-M4 console. It shows all cameras' views on the display.
6. Stop EVS with stop `evs_app` on the Cortex-A core console.

Note:

You can input `gear 2` on the Cortex-M4 console anytime in the boot process to start the rearview camera.

8.1.2.2 Starting the EVS function with images in `automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz`

With images in `automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz`, the EVS function is realized on Android Automotive running on the Cortex-A core.

i.MX 8QuadMax MEK and i.MX 8QuadXPlus MEK both support a single-rearview camera. To start the single-rearview camera:

1. Connect the camera as described in the *Android Automotive Quick Start Guide* (UG10177).
2. Open the Cortex-A core console.
   Input `su && start evs_app` on the Cortex-A console to start `evs_app`. You can also start the rearview camera with `echo 2 > /sys/devices/platform/vehicle-dummy/gear` on the Cortex-A console. The display should be rear camera view. Input `stop evs_app` on the Cortex-A console to stop the rearview camera EVS function.

i.MX 8QuadMax MEK and i.MX 8QuadXPlus MEK with silicon revision C0 chip can also support multiple EVS cameras.

The relationship between the orientation of cameras and hardware connection is shown as follows.

Table 17. Relationship between the orientation of cameras and hardware connection

Hardware connection	Camera orientation
IN0	Rear
IN1	Front
IN2	Right
IN3	Left

The logic to handle the vehicle information is shown with the following pseudo code:

```
if (gear state == reverse)
    show rear camera view
else if (turn signal == right)
    show right camera view
else if (turn signal == left)
    show left camera
else if (gear state == park)
    show all cameras' view
else
    show no camera view
```

The meaning of the commands input on the Cortex-A core console is as follows.

Table 18. Meaning of commands input on the Cortex-A core console

Command	Meaning
`echo 0 > /sys/devices/platform/vehicle-dummy/turn`	Not turn
`echo 1 > /sys/devices/platform/vehicle-dummy/turn`	Turn right
`echo 2 > /sys/devices/platform/vehicle-dummy/turn`	Turn left
`echo 1 > /sys/devices/platform/vehicle-dummy/gear`	Park
`echo 2 > /sys/devices/platform/vehicle-dummy/gear`	Reverse
`echo 4 > /sys/devices/platform/vehicle-dummy/gear`	Drive

To start the multiple-EVS-camera function:

1. Input `su && start evs_app` on the Cortex-A console to start `evs_app`. You can also start the rearview camera with `echo 2 > sys/devices/platform/vehicle-dummy/gear` on the Cortex-A console. The display should be rear camera view.
2. Input `echo 1 > sys/devices/platform/vehicle-dummy/gear` on the Cortex-A console. All cameras must be connected to the board.

Figure 3. All cameras' views on the display

3. Input `echo 1 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It shows the right camera view on the display.
4. Input `echo 2 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It shows the left camera view on the display.
5. Input `echo 0 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It shows all cameras' views on the display.
6. Stop EVS with stop `evs_app` on the Cortex-A console.

8.1.3 EVS related code

For images in `automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz`, the Cortex-M4 core runs with its code on the DDR on the i.MX board. It is responsible for the following work:

- Take over control of the camera/display before the Android system is fully booted.
- Get the vehicle event and pass this event to the Cortex-A core.

Source code for the Cortex-M4 core is in the `${MY_ANDROID}/vendor/nxp/mcu-sdk-auto` directory.

After modifying the Cortex-M4 core source code, execute the following command to build and update the Cortex-M4 image:

```
cd ${MY_ANDROID}
source build/envsetup.sh
lunch mek_8q_car-nxp_stable-userdebug
./imx-make.sh bootloader -j4
```

The directory of EVS related code running on the Cortex-A core is listed as follows:

- EVS HAL: `${MY_ANDROID}/vendor/nxp-opensource/imx/evs_hal`
- EVS service: `${MY_ANDROID}/vendor/nxp-opensource/imx/evs/evs_service`
- EVS kernel driver: `${MY_ANDROID}/vendor/nxp-opensource/kernel_imx/drivers/mxc/vehicle`
- EVS application: `${MY_ANDROID}/vendor/nxp-opensource/imx/evs/evs_app`

After modifying the Cortex-A core source code, build the whole system to update Android Automotive images.

8.1.4 Communication protocol between Cortex-A core and Cortex-M4 core

Images in `automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz` are built with the target lunched with `mek_8q_car-userdebug`. The EVS function in this package is realized with both the Cortex-A core and the Cortex-M4 core.

The following table lists the communication commands and related response packet between the Cortex-A core and the Cortex-M4 core.

Table 19. SRTM AUTO Control Category Command Table (Cortex-A -> Cortex-M4)

Category	Version	Type	Command	Data	Function
0x08	0x0100	REQUEST	REGISTER	Data[0-3]: clientId Data[4]: reserved Data[5]: partition Data[6-15]: reserved	Registers the RPMsg client. clientId indicates different clients. partition indicates the Android partition. Partition: 0xFF: This parameter is invalid.
0x08	0x0100	REQUEST	UNREGISTER	Data[0-3]: clientId Data[4]: reserved Data[5]: causeOf Data[6-15]: reserved	Unregisters the RPMsg client. Cortex-M and remote processor cannot communicate again. The causeOf parameter can indicate the reason for unregister. causeOf: 0x00: AP powers off.
0x08	0x0100	REQUEST	CONTROL	Data[0-3]: clientId Data[4]: reserved Data[5-6]: controlCode Data[7-10]: timeout Data[11-15]: control Param Data[15]: index	Sends a control command to Cortex-M to request Cortex-M to do some actions. It needs to complete and give a response to Android in "timeout" ms. Reserved for future. Example: • controlCode: 0x0000: Air conditioner temperature • controlParam: 4bytes (float): Temperature • Index: Left or right.
0x08	0x0100	REQUEST	PWR_REPORT	Data[0-3]: clientId Data[4]: reserved Data[5-6]: androidPwr State Data[7-10]: time_ postpone Data[11-15]: reserved	Reports the Android power state. androidPwrState: • 0x0000: BOOT_COMPLETE • 0x0001: DEEP_SLEEP_ENTRY • 0x0002: DEEP_SLEEP_EXIT • 0x0003: SHUTDOWN_POSTPONE • 0x0004: SHUTDOWN_START • 0x0005: DISPLAY_OFF • 0x0006: DISPLAY_ON
0x08	0x0100	REQUEST	GET_INFO	Data[0-3]: clientId Data[4]: reserved Data[5-6]: infoIndex Data[7-15]: reserved	Gets information from the Cortex-M side. Android and Cortex-M should have the same information table. The information includes the sensor data, fuel data, battery data, and so on. infoIndex:0x0001: Vehicle unique ID.
0x08	0x0100	RESPONSE	BOOT_REASON	Data[0-3]: clientId Data[4]: retCode Data[5-15]: reserved	Responds to Cortex-M's boot reason request (USER_POWER_ON, DOOR_OPEN, DOOR_UNLOCK, REMOTE_START, TIMER).
0x08	0x0100	RESPONSE	PWR_CTRL	Data[0-3]: clientId Data[4]: retCode Data[5-6]: androidPwr State Data[7-15]: reserved	Responds to the current power state of the Android OS.

Table 19. SRTM AUTO Control Category Command Table (Cortex-A -> Cortex-M4)*...continued*

Category	Version	Type	Command	Data	Function
0x08	0x0100	RES PONSE	VSTATE	Data[0-3]: clientId Data[4]: retCode Data[5-6]: unitType Data[7-15]: reserved	Responds to the control command from the Cortex-M side. state indicates the current IVI state.

Table 20. SRTM AUTO Control Category Command Table (Cortex-M4 -> Cortex-A)

Category	Version	Type	Command	Data	Function
0x08	0x0100	RES PONSE	REGISTER	Data[0-3]: clientId Data[4]: retCode Data[5-6]: mcuOperate Mode Data[7-15]: reserved	Response of the RPMsg client register (success, failed). mcuOperateMode: indicates the Cortex-M work state. • SHARED_RESOURCE_FREE: 0x0000. • SHARED_RESOURCE_OCCUPIED: 0x0001.
0x08	0x0100	RES PONSE	UNREGI STER	Data[0-3]: clientId Data[4]: retCode Data[5-15]: reserved	Response of the RPMsg client unregister.
0x08	0x0100	RES PONSE	CONTROL	Data[0-3]: clientId Data[4]: retCode Data[5-6]: actionState Data[7-15]: reserved	Response to the result of the control request. The MCU performs some actions to complete Android's request. actionState is not used currently.
0x08	0x0100	RES PONSE	PWR_ REPORT	Data[0-3]: clientId Data[4]: retCode Data[5-15]: reserved	Response to Android power state report.
0x08	0x0100	RES PONSE	GET_INFO	Data[0-3]: clientId Data[4]: retCode Data[5-6]: infoIndex Data[7-14]: data Data[15]: reserve	Response to the GET_INFO request. infoIndex should be the same as request index. The length of infoData should be specific according to info Index. The information includes sensor data, fuel data, and battery data. And it is a response packet to Android's request.
0x08	0x0100	REQUEST	BOOT_ REASON	Data[0-3]: clientId Data[4]: reserved Data[5]: bootReason Data[6-15]: reserved	Notifies the Android system why VMCU boots the Cortex-A core (Android). It is sent after the MCU sends a normal drive command to the Android system. bootReason: • 0x00: USER_POWER_ON • 0x01: DOOR_OPEN • 0x02: DOOR_UNLOCK • 0x03: REMOTE_START
0x08	0x0100	REQUEST	PWR_CTRL	Data[0-3]: clientId Data[4]: reserved Data[5-6]: powerState Req Data[7-8]: addition Param Data[9-15]: reserved	Requests the Android system to enter a specific power state (ON_DISP_OFF, ON_FULL, SHUTDOWN_PREPARE) powerStateReq: • 0x0000: ON_DISP_OFF • 0x0001: ON_FULL • 0x0002: SHUTDOWN_PREPARE

Table 20. SRTM AUTO Control Category Command Table (Cortex-M4 -> Cortex-A)*...continued*

Category	Version	Type	Command	Data	Function
0x08	0x0100	REQUEST	VSTATE	Data[0-3]: clientId Data[4]: reserved Data[5-6]: unitType Data[7-10]: stateValue Data[11-15]: reserved	Requests the Vehicle state to Android (Door open/close/lock/unlock, Fan on/off/speed/recycle/direction, AC on/off/temperature, heater on/off/power, defrost on/off/front/back) (mute/unmute, volume adjust, rear view camera on/off, lights on/off …) unitType indicates the type of each unit of vehicle, such as door, fan, and air condition. stateValue indicates the unit state parameter.

8.1.5 Delay of camera/display module probe

The RVC is occupied by the Cortex-M4 core in early stage when booting up in an Android car. AP needs to separate camera/display resource in boot stage. There are two resources that need to pay attention in AP boot stage: clock and power domain.

1. Separate clock in boot stage.
   a. Add CONFIG_VEHICLE_CLK_POST_INIT, which does not register camera/display related CLK in clk-imx8qxp.c and clk-imx8qm.c.
   b. Add clk-post-imx8qm.c and clk-post-imx8qxp.c, which are probed in notice_evs_released.
2. Separate power domain in boot stage.
   SC_R_CSI_0/SC_R_LVDS_1/SC_R_DC_1/SC_R_ISI_CH0 are used at Cortex-M4 side. The related power domain used in DTS needs to be removed under the DTS node vehicle_rpmsg_m4.
   - The node whose power domain is pd_dc1 needs to be moved into vehicle_rpmsg_m4.
   - The node whose power domain is under pd_dc1 (such as pd_mipi1/pd_lvds1/pd_mipi1_i2c0/..) needs to be moved into the DTS node vehicle_rpmsg_m4.
   - The node whose power domain is pd_isi_ch0 needs to be moved into the DTS node vehicle_rpmsg_m4.
   - The node whose power domain is under pd_isi_ch0 (such as pd_csi0/pd_csi1/..) needs to be moved into the DTS node vehicle_rpmsg_m4.
   - The camera node needs to be moved into the DTS node vehicle_rpmsg_m4.

8.2 Audio configuration

8.2.1 Routing audio stream to different sound cards

In Android Automotive, different audio streams are routed to different sound cards. When configured, the route is statically decided, unlike the dynamically routed in the standard Android image.

In the Android Automotive release, the route is configured as follows:

- CPU board audio jack (WM896x codec) is routed to bus0_media_out. This audio bus plays all types of sounds such as media, calls, alarms, and alerts.
- Extended audio board (CS42888) is routed to bus100_audio_zone_1 audio bus. This audio bus plays all types of sounds. The bus is for emulation of the passenger rear audio zone and is optional.

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/imx8q/mek_8q/BoardConfig.mk` as shown below:

```
BOARD_KERNEL_CMDLINE += androidboot.lcd_density=200
```

The display density value can be changed by modifying the related lines mentioned above in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk` and then recompiling the code or setting (the density value) in U-Boot command line as `bootargs` during boot-up.

8.3.2 Starting the cluster display

Cluster display is supported in the i.MX Android Automotive release package. With this feature, two displays connected to the board can display different content.

To do customization work on this function, you need to know how this function can be started and controlled.

i.MX 8QuadMax and i.MX 8QuadXPlus MEK:

To start the cluster display, connect the two i.MX mini SAS cables with the LVDS-to-HDMI adapters to the "LVDS0" and "LVDS1" ports of the board. After the system boots into the Android launcher, different content is displayed on the two displays connected to the board.

i.MX 95 EVK:

To evaluate the cluster display on i.MX 95 EVK, flash `dtbo-imx95-mipi-lvds1.img`, `vbmeta-imx95-mipi-lvds1.img` and connect the two mini SAS cables with the MIPI-to-HDMI and LVDS-to-HDMI adapters to the "MIPI_DSI" and "LVDS1" ports of the board.

8.3.3 Enabling the multiple-display function

The following boards support more than one display.

Table 21. Displays supported by different boards

Board	Number of displays	Display port
i.MX 8QuadMax MEK	4	If a physical HDMI output (J6) is used: HDMI_TX, LVDS0_CH0, LVDS1_CH0, MIPI_DSI1

Table 21. Displays supported by different boards*...continued*

Board	Number of displays	Display port
		If a physical HDMI output (J6) is not used: LVDS0_CH0 and LVDS1_CH0, MIPI_DSI0 and MIPI_DSI1
i.MX 8QuadXPlus MEK	2	DSI0/LVDSI0, DSI1/LVDSI1
i.MX 95 EVK	2	MIPI_DSI, LVDS1

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

The multiple-display without physical HDMI output (J6) displays on i.MX 8QuadMax MEK are enabled by default.

To evaluate the multiple-display with physical HDMI output (J6) on i.MX 8QuadMax MEK, flash `dtbo-imx8qm-md.img`, `vbmeta-imx8qm-md.img`, and `u-boot-imx8qm-md.imx`.

To evaluate the multiple-display on i.MX 95 EVK, flash `dtbo-imx95-mipi-lvds1.img` and `vbmeta-imx95-mipi-lvds1.img`.

8.3.3.1 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 vary with the connection forms of these ports with corresponding input and display devices, which means that 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 8QuadMax MEK as an example:

1. Use the following commands to get 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=NXP displayName="NXP
 Android"
Display 4692921138614786 (HWC display 2): port=2 pnpId=NXP displayName="NXP
 Android"
Display 18309706364381699 (HWC display 3): port=3 pnpId=NXP displayName="NXP
 Android"
```

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

```
getevent -i | grep location location:
location: "usb-xhci-hcd.1.auto-1.3.4/input0"
location: "usb-xhci-hcd.1.auto-1.2.4/input0"
```

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

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

```
</ports>
```

To make the modifications take effect, you can modify the source file under the `${MY_ANDROID}/device/nxp/` directory and re-build the images. Keep the connection of display devices and input devices unchanged and reflash the images. You can also 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.

8.3.3.2 Enabling multi-client input method

Only multi-client IMEs 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, clear `persist.debug.multi_client_ime` as follows. Then, reboot the system to make it 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 a sample multi-client IME from AOSP. The performance is not as good as the default Google Input Method Editor. To develop multi-client IME, see the document in source code (`${MY_ANDROID}/frameworks/base/services/core/java/com/android/server/inputmethod/multi-client-ime.md`).

8.3.3.3 User's zone configuration in multi-display mode

The multi-zone launcher application is automatically started on the secondary displays in multi-display mode. The multi-zone launcher application provides the interface for a multi-user and zone support, supporting multiple-user settings, applications, and data. Android Automotive relies on Android's multi-user support to provide a shared device experience, wherein each device user is intended to be used by a different physical person and zone. The display output port (DSI, LVDS, HDMI) and a passenger zone configuration is bound with the `display-id`. `/vendor/etc/displayconfig/display_layout_configuration.xml` is used to do this work when the system is running, but the passenger zone configuration and `display-id` vary with the connection forms of these ports with corresponding input and display devices, which means that the passenger zone and `display-id` need to be retrieved before the connection is fixed.

The source file of `/vendor/etc/displayconfig/display_layout_configuration.xml` is in the repository under the `${MY_ANDROID}/device/nxp/` directory.

Take i.MX 8QuadMax MEK as an example:

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

```
dumpsys SurfaceFlinger --display-id
Display 4616379502225925632 (HWC display 0): port=0 pnpId=DEL
 displayName="DELL U2412M"
Display 4616379120519781377 (HWC display 1): port=1 pnpId=DEL
 displayName="DELL P2412H"
Display 4616379502225925634 (HWC display 2): port=2 pnpId=DEL
 displayName="DELL U2412M"
```

```
Display 4616378750182017539 (HWC display 3): port=3 pnpId=DEL
 displayName="DELL U2417H"
```

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

```
<layouts>
  <layout>
    <!-- Use the default state -->
    <state>-1</state>

    <!-- Primary display port -->
    <display enabled="true" defaultDisplay="true"/>
      <address/>4616379120519781376</address/>
    </display/>

    <!-- Display cluster port -->
    <display enabled="true" defaultDisplay="false"/>
      <address/>4616379502225925633</address/>
    </display/>

    <!-- Passenger 1 display port -->
    <display enabled="true" defaultDisplay="false"
 displayGroup="passenger_display1"/>
      <address/>4616379502225925634</address/>
    </display/>

    <!-- Passenger 2 display port -->
    <display enabled="true" defaultDisplay="false"
 displayGroup="passenger_display2"/>
      <address/>4616378750182017539</address/>
    </display/>
  </layout>
</layouts>
```

To make the modifications take effect, you can modify the source file under the `${MY_ANDROID}/device/nxp/` directory and re-build the images. Keep the connection of the display devices and input devices unchanged and reflash the images. You can also 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.

8.3.4 Configuring the primary display resolution

The whole Android UI stack needs a display resolution to be defined before Android framework boots up.

In normal Android and car2 build, the display resolution is obtained when enumerating `/dev/dri/cardX` in display HAL. The system selects the best aligned resolution when the `ro.boot.displaymode` property is set, or select the default "1080p60" when the property is not set.

In car build, the predefined resolution is defined by the `ro.boot.fake.ui_resolution` property and it should be aligned with physical display device. When the physical display is ready, the PollFileThread gets the event and enumerates the `/dev/dri/cardX` again to configure the physical display.

When the MCU takes over the display, the resolution of the display is hardcoded in the MCU-SDK code by macro `APP_FRAME_HEIGHT` and `APP_FRAME_WIDTH` in the `isi_example.h` file. This resolution should align with Android UI settings, or the display experience is different.

8.4 HVAC configuration

HVAC is short for "Heating, Ventilation and Air Conditioning". This section describes the interfaces to control the HVAC system. It helps customers to do customization work on HVAC.

8.4.1 Interfaces to control the HVAC system

For images in `automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz` built with the lunch target `mek_8q_car-userdebug`, see the following table to control the HVAC system.

Table 22. HVAC test items for automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz

	AP-> Cortex-M	Cortex-M -> AP (input on the Cortex-M console)	Comment
AC ON	The Cortex-M console has the following print when AC is on: `Android control: AC_ON, on/off`	`=>report ac_on 0/1` The AC on the panel is on/off.	-
Fan direction	`Android control: FAN_DIRECTION, 0x2` Typical value: 0x1 (to face) 0x2 (to floor) 0x03 (to face and floor) 0x06 (to floor and defrost)	`=>report fan_direction 0x1/0x2/0x03/0x06` It sets the fan direction.	-
Fan speed	`Android control: FAN_SPEED, 0x6` Typical value: 0x00(off)/0x01/0x02/0x03/0x04/0x05/0x06(MAX)	`=>report fan_speed 1/2/3/4/5/6` It sets the fan speed.	-
HVAC power on	The cortex-M console has the following print when HVAC is on: `Android control: HVAC_POWER_ON, on/off`	`=>report hvac_power 0/1` It sets the HVAC power.	-
AUTO ON	The Cortex-M Console has the following print when HVAC is auto: `Android control: AUTO_ON, on/off`	`=>report auto_on 0/1` AUTO on the panel is on/off.	-
Defrost	Left one: `Android control: DEFROST, index=1, on/off` Right one: `Android control: DEFROST, index=2, on/off`	Left one: `=>report defrost 0/1 1` The defrost on the panel is on/off. Right one: `=>report defrost 0/1 2` The defrost on the panel is on/off.	-
Temperature	Left temp +-: `Android control: AC_TEMP, index=32, temp=16.16` Right temp +-: `Android control: AC_TEMP, index=64, temp=18.18`	`=>report ac_temp 23.45 32/ 64` Sends the 23.45 Centigrade value to the Android side, and the left/right HVAC temperature bar changes to 74.	The formula for Celsius to Fahrenheit conversion is as follows: cDegrees = ((fDegrees - MIN_Fahrenheit) / (MAX_FAHRENHEIT - MIN_Fahrenheit)) * (MAX_

Table 22. HVAC test items for automotive-15.0.0_2.1.0_image_8qmek_car.tar.gz*...continued*

	AP-> Cortex-M	Cortex-M -> AP (input on the Cortex-M console)	Comment
			CELSIUS- MIN_CELSIUS) + MIN_CELSIUS Where, MIN_Fahrenheit = 60, MAX_FAHRENHEIT = 84, MAX_CELSIUS = 28, MIN_CELSIUS = 16
RECIRC	The Cortex-M console has the following print when recirc is on: `Android control: RECIRC_ON, off/on`	`=>report recirc_on 0/1` RECIRC on the panel is on/off.	-
SEAT TEMPERATURE	Left one: `Android control: SEAT_TEMP, index=1, values 0,1,2,3` Right one: `Android control: SEAT_TEMP, index=4, values 0,1,2,3`	`=>report seat_temp 1/4 0/1/2/3`	-

For images in `automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz` built with the lunch target `mek_8q_car2-userdebug`, see the following table to control the HVAC system.

Table 23. HVAC test items for automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz

	AP-> dummy vehicle driver	Cortex-M -> dummy vehicle driver	Comment
AC ON	The AP Console has the following print when AC is off/on: `set fan AC on with value 0/1`	`echo 0/1 > /sys/devices/platform/vehicle-dummy/ac_on` The AC on the panel is on/off.	
Fan direction	Set fan direction with value 8 Typical value: 0x1 (to face) 0x2 (to floor) 0x03 (face and floor) 0x06 (to floor and defrost)	`echo 1/2/3/6 > /sys/devices/platform/vehicle-dummy/fan_direction`	
Fan speed	Set fan speed with value 8 Typical value: 0x00(off)/0x01/0x02/0x03/0x04/0x05/0x06(MAX)	`echo 1/2/3/4/5/6 > /sys/devices/platform/vehicle-dummy/fan_speed` It sets the fan speed.	
HVAC power on	HVAC on: `Android control: HVAC_POWER_ON, on/off`	`echo 0/1 > /sys/devices/platform/vehicle-dummy/hvac_on`	
AUTO ON	Set auto on with value 0/1 Set auto off/on	`echo 0/1 > /sys/devices/platform/vehicle-dummy/auto_on` AUTO on the panel is on/off.	

Table 23. HVAC test items for automotive-15.0.0_2.1.0_image_8qmek_car2.tar.gz*...continued*

	AP-> dummy vehicle driver	Cortex-M -> dummy vehicle driver	Comment
Defrost	Left one: `set defroster index 1 with value 0/1` Right one: `set defroster index 2 with value 0/1`	Left one: `echo 0/1 > /sys/devices/platform/vehicle-dummy/defrost_right` The defrost on the panel is close/open. Right one: `echo 0/1 > /sys/devices/platform/vehicle-dummy/defrost_right` The defrost on the panel is on/off.	
Temperature	Left temp +-: set `temp index 32` with value 1097859072 Right temp +-: set `temp index 64` with value 1100422258	`echo 1095528903 > /sys/devices/platform/vehicle-dummy/temp_left` The left HVAC temperature bar changes to 55.	The formula for Celsius to Fahrenheit conversion is as follows: cDegrees = ((fDegrees - MIN_Fahrenheit) / (MAX_FAHRENHEIT - MIN_Fahrenheit)) * (MAX_CELSIUS- MIN_CELSIUS) + MIN_CELSIUS Where, MIN_Fahrenheit = 60, MAX_FAHRENHEIT = 84, MAX_CELSIUS = 28, MIN_CELSIUS = 16
RECIRC	Recirc on: set recirc on with value 0/1	`echo 0/1 > /sys/devices/platform/vehicle-dummy/recirc_on` RECIRC on the panel is on/off.	
SEAT TEMPERATURE	Control seat temperature with values 0/1/2/3/4. Value 0 means OFF.	`echo 0/1/2/3 > /sys/devices/platform/ vehicle-dummy/seat_temp_left` `echo 0/1/2/3 > /sys/devices/platform/ vehicle-dummy/seat_temp_right`	-

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 MEK, 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_androidauto_trusty_defconfig b/configs/
imx8qm_mek_androidauto_trusty_defconfig
index 9ceb9d58f1..a54766eb6a 100644
--- a/configs/imx8qm_mek_androidauto_trusty_defconfig
+++ b/configs/imx8qm_mek_androidauto_trusty_defconfig
@@ -101,13 +101,11 @@ CONFIG_SPL_DM_USB_GADGET=y
CONFIG_USB=y
CONFIG_USB_GADGET=y
-#CONFIG_CI_UDC=y
+CONFIG_CI_UDC=y
CONFIG_USB_GADGET_DOWNLOAD=y
CONFIG_USB_GADGET_MANUFACTURER="FSL"
CONFIG_USB_GADGET_VENDOR_NUM=0x0525
CONFIG_USB_GADGET_PRODUCT_NUM=0xa4a5
-CONFIG_USB_CDNS3=y
-CONFIG_USB_CDNS3_GADGET=y
CONFIG_USB_GADGET_DUALSPEED=y
CONFIG_SPL_USB_GADGET=y
@@ -124,7 +122,7 @@ CONFIG_FSL_FASTBOOT=y
CONFIG_FASTBOOT_BUF_ADDR=0x98000000
CONFIG_FASTBOOT_BUF_SIZE=0x19000000
CONFIG_FASTBOOT_FLASH=y
-CONFIG_FASTBOOT_USB_DEV=1
+CONFIG_FASTBOOT_USB_DEV=0
CONFIG_BOOTAUX_RESERVED_MEM_BASE=0x88800000
CONFIG_BOOTAUX_RESERVED_MEM_SIZE=0x02000000
diff --git a/include/configs/imx8qm_mek_android_auto.h b/include/configs/
imx8qm_mek_android_auto.h
index 793530c61a..5bef17b451 100644
--- a/include/configs/imx8qm_mek_android_auto.h
+++ b/include/configs/imx8qm_mek_android_auto.h
@@ -51,7 +51,6 @@
#define CONFIG_SYS_MALLOC_LEN           (64 * SZ_1M)
#endif
-#define CONFIG_FASTBOOT_USB_DEV 1
#define CONFIG_ANDROID_RECOVERY
#define CONFIG_CMD_BOOTA
```

For i.MX 8QuadXPlus, to enable USB2.0 for the `u-boot-imx8qxp.imx`, make the following changes under `${MY_ANDROID}/vendor/nxp-opensource/uboot-imx`:

```
diff --git a/configs/imx8qxp_mek_androidauto_trusty_defconfig b/configs/
imx8qxp_mek_androidauto_trusty_defconfig
index e3f60821b0..6b59fa71ab 100644
--- a/configs/imx8qxp_mek_androidauto_trusty_defconfig
+++ b/configs/imx8qxp_mek_androidauto_trusty_defconfig
@@ -103,13 +103,11 @@ CONFIG_SPL_DM_USB_GADGET=y
CONFIG_USB=y
CONFIG_USB_GADGET=y
-#CONFIG_CI_UDC=y
+CONFIG_CI_UDC=y
CONFIG_USB_GADGET_DOWNLOAD=y
CONFIG_USB_GADGET_MANUFACTURER="FSL"
CONFIG_USB_GADGET_VENDOR_NUM=0x0525
CONFIG_USB_GADGET_PRODUCT_NUM=0xa4a5
-CONFIG_USB_CDNS3=y
```

```
-CONFIG_USB_CDNS3_GADGET=y
CONFIG_USB_GADGET_DUALSPEED=y
CONFIG_SPL_USB_GADGET=y
CONFIG_SPL_USB_SDP_SUPPORT=y
@@ -124,7 +122,7 @@ CONFIG_FSL_FASTBOOT=y
CONFIG_FASTBOOT_BUF_ADDR=0x98000000
CONFIG_FASTBOOT_BUF_SIZE=0x19000000
CONFIG_FASTBOOT_FLASH=y
-CONFIG_FASTBOOT_USB_DEV=1
+CONFIG_FASTBOOT_USB_DEV=0
CONFIG_SYS_I2C_IMX_VIRT_I2C=y
CONFIG_I2C_MUX_IMX_VIRT=y
CONFIG_IMX_VSERVICE_SHARED_BUFFER=0x90000000
diff --git a/include/configs/imx8qxp_mek_android_auto.h b/include/configs/
imx8qxp_mek_android_auto.h
index 95ec29d307..376b306c72 100644
--- a/include/configs/imx8qxp_mek_android_auto.h
+++ b/include/configs/imx8qxp_mek_android_auto.h
@@ -45,7 +45,6 @@
#endif
#define CONFIG_SKIP_RESOURCE_CHECKING
-#define CONFIG_FASTBOOT_USB_DEV 1
#define CONFIG_ANDROID_RECOVERY
#define CONFIG_CMD_BOOTA
```

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

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

Note:

U-Boot used by UUU is compiled with `imx8qm_mek_android.h` and `imx8qxp_mek_android.h`, not the `imx8qm_mek_android_auto.h` and `imx8qxp_mek_android_auto.h` listed above.

8.6 Trusty OS/security configuration

Trusty OS firmware is used in the i.MX Android 15 release as TEE, which supports security features.

The i.MX Trusty OS is based on the AOSP Trusty OS and supports the i.MX 8QuadMax MEK and i.MX 8QuadXplus MEK boards. This section describes some basic configurations to make the Trusty OS work on MEK boards. For more configurations about security-related features, see the *i.MX Android Security User's Guide* (UG10158).

Customers can modify the Trusty OS code to make different configurations and enable different features. First, u

1. Use the following commands to fetch code to build the target Trusty OS binary. Create a directory for the Trusty OS code and enter this directory.

```
$ repo init -u https://github.com/nxp-imx/imx-manifest.git -b imx-android-15
 -m imx-trusty-automotive-15.0.0_2.1.0.xml
$ repo sync
$ source trusty/vendor/google/aosp/scripts/envsetup.sh
$ ./trusty/vendor/google/aosp/scripts/build.py imx8qxp #i.MX 8QuadXPlus MEK
$ cp ${TRUSTY_REPO_ROOT}/build-imx8qxp/lk.bin ${MY_ANDROID}/vendor/nxp/fsl-
proprietary/uboot-firmware/imx8q_car/tee-imx8qx.bin
```

2. Build the images, and `tee-imx8qx.bin` is integrated into `bootloader-imx8qxp.img` and `bootloader-imx8qxp-secure-unlock.img`. Flash the `spl-imx8qxp.bin` and `bootloader-imx8qxp.img` files to the target device.

Note:

- *For i.MX 8QuadMax MEK, use `make imx8qm_a72` to build the Trusty OS image, and copy `final lk.bin` to `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/tee-imx8qm.bin`.*
- *`${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 the default execution flow of images does not make this initialization.

The RPMB can be initialized with the hardware bound key or vendor specified key. The RPMB key cannot be changed once it is set.

- To set a **hardware bound** key, perform the following operation:
  Make your board enter fastboot mode, and then execute the following command 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 operation:
  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 the Trusty OS is initialized successfully.
  Note:
  – *This method does not work on the platforms without CAAM (for example, i.MX 95).*
  – *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\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 (UG10158).

8.6.2 AVB key provision

The AVB key consists of a pair of public and private keys. The private key is used by the host to sign the vbmeta image. The public key is used by AVB to authenticate the vbmeta image. The following figure shows the

relationships between the private key and vbmeta. Without Trusty OS, the public key is hard-coded in U-Boot. With Trusty OS, it is saved in secure storage.

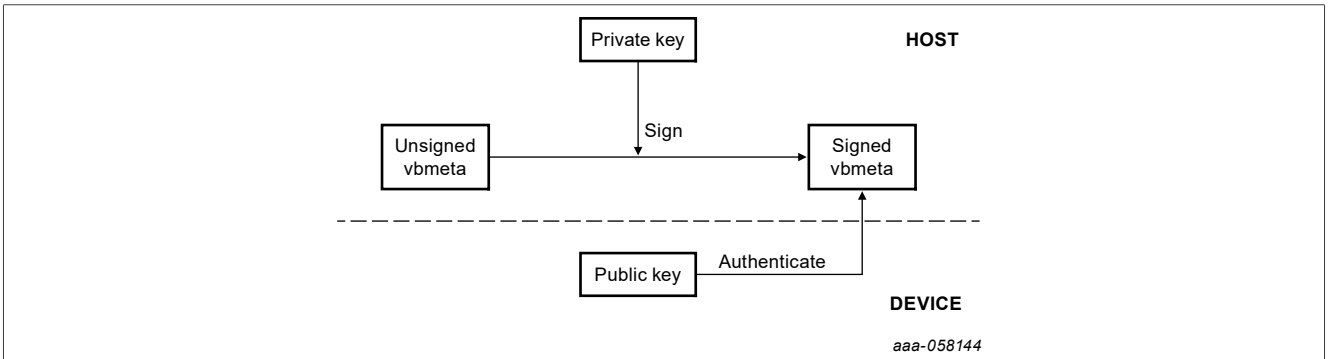

Figure 4. Relationship between AVB key and vbmeta

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 following 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 overridden 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 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 How to set the vbmeta public key

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

Make your board enter fastboot mode, and execute 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 specified private key. If no private key is specified, set the public key as prebuilt `testkey_public_rsa4096.bin`, which is extracted from the default private key `testkey_rsa4096.pem`.

8.6.3 Key attestation

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

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

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

- Set RSA private key:

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

- Set ECDSA private key:

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

- Append RSA certificate chain:

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

Note:
This command may need to be executed multiple times to append the whole certificate chain.

- Append ECDSA certificate chain:

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

Note:
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 the 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 LF6.6.52_2.2.0 release and download its corresponding SCFW Porting kit. Then, decompress the file with the following commands:

```
tar -zxvf imx-scfw-porting-kit-1.18.0.tar.gz
cd packages
chmod a+x imx-scfw-porting-kit-1.18.0.bin
./imx-scfw-porting-kit-1.18.0.bin
```

```
cd imx-scfw-porting-kit-1.18.0/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, taking i.MX 8QuadXPlus MEK as an example, it is the `scfw_export_mx8qx_b0/platform/board/mx8qx_mek/board.c`. Based on this file, some changes are made for Android Automotive and the file is stored in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/board-imx8qxp.c`.

You can copy `board-imx8qxp.c/board-imx8qm.c in vendor/nxp/fsl-proprietary` to the SCFW porting kit, modify it, and then build the SCFW.

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

1. Download the GCC tool from the arm Developer GNU-RM Downloads page. It is recommended 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_car/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 Power state configuration

Android automotive power HAL supports power request property, which can be used to control the system power state: ON, OFF, or suspend.

It is assumed that the power state of the Cortex-A core is controlled by separate power controller. In the following use case, MCU and dummy vehicle driver play the role of power controller in the car and car2 image accordingly.

Connect the board to a BT device to better show the system power state.

Table 24. Power state configuration

	Power control from car MCU console	Power control from car2 AP console	Comment
shutdown now	power 1 1	`echo "1 1" > /sys/devices/platform/vehicle-dummy/power_req`	The system shuts down right now. Then, long press the power-on key to wake up the system.
suspend	power 1 2	`echo "1 2" > /sys/devices/platform/vehicle-dummy/power_req`	The system disconnects from BT, waits for all tasks to be done, and then enter suspend mode. Press the power-on key to wake up the system. BT is connected again. The system wakes up by itself every 60 seconds due to battery health checking.
shutdown postpone	power 1 3	`echo "1 3" > /sys/devices/platform/vehicle-dummy/power_req`	The system waits for all tasks to be done, and then shuts down.
cancel	power 2 0	`echo "2 0" > /sys/devices/platform/vehicle-dummy/power_req`	Cancel the shutdown and suspend command if it has not been executed. First, enter `power 1 3` for car image or echo `1 3` to `power_req` for car2 image. The system disconnects from the BT, turns off the display, and prepares for shutdown. Before the system shuts down, enter `power 2 0` for car image (or `echo 2 0` to `power_req` for car2 image). The system cancels shutdown command, turns on the display, and connects BT.

8.9 Boot time tuning

8.9.1 Boot time overview

In this document, the boot time is the time it takes the board the to start from cold boot to when Android Automotive Launcher UI appears on the display screen when the hardware is not in the first-time boot from factory. Due to the fact that the first successful boot sets up the accelerating software executing environment, it takes longer time to boot.

NXP makes the boot time shorter in U-Boot, Linux kernel, and Android framework. To improve the debug efficiency, some debug purpose modules and interfaces are kept in the release. Before the product is ready to ship, these modules and interfaces can be configured to save the boot time and make the boot time performance best in the final product.

8.9.2 What NXP did to tune the boot time

To make Android Automotive boot faster, lots of changes were made on different modules to achieve better performance. The following changes impact the boot time:

- Removed the debug command in U-Boot and Linux kernel to save its initialization time and image size.
- Built Linux kernel as zImage to save the image size.
- Removed unused driver in U-Boot and Linux kernel.
- Make some drivers as kernel module, and load them when Android boot is completed so that the connectivity devices and camera driver are initialized after the Android Automotive Launcher UI is shown on the display. This makes the Android Automotive Launcher UI show earlier.
- Removed unused device from Android Framework, such as Ethernet, Sensors.
- Refined Android Verify Boot procedure.
- Optimized Android Framework to make service execute on different CPUs.

- Delayed some non-critical services for SystemUI module of Android after boot is completed.
- Delayed Zygote32 to when UI is shown.
- Delayed Bluetooth service to when UI is shown.
- Removed some unused service in Android Framework.
- Booted from the Cortex-A72 core instead of Cortex-A53 (only for i.MX 8QuadMax MEK).

All the changes above do not impact any of the functions and the performance except the boot time.

8.9.3 How to get the shorter boot time

For debug and development purpose, the U-Boot boot delay and the logs in U-Boot, Trusty OS, and Linux kernel are enabled by default. In field measurement, the Linux kernel dmesg takes about 1.15 seconds during the boot process because UART is a slow device. Therefore, before the final product, it is recommended to remove the U-Boot delay and the logs in U-Boot, Trusty OS, and Linux Kernel by performing the following operations:

1. Set `CONFIG_BOOTDELAY=-2` in the U-Boot defconfig file to remove boot delay.
2. Remove `CONFIG_SPL_SERIAL_SUPPORT=y` in U-Boot defconfig file to disable logs at SPL stage.
3. Set `CONFIG_SERIAL_PRESENT=n` in U-Boot defconfig file to disable logs in U-Boot proper. Disable the UART node in U-Boot DTS. Take i.MX 8QuadMax as example:

```
--- a/arch/arm/dts/fsl-imx8qm-mek-auto.dts
+++ b/arch/arm/dts/fsl-imx8qm-mek-auto.dts
diff --git a/arch/arm/dts/fsl-imx8qm-mek-auto.dts b/arch/arm/dts/fsl-imx8qm-
mek-auto.dts
index 461ee46fa8..58356e1466 100644
--- a/arch/arm/dts/fsl-imx8qm-mek-auto.dts
+++ b/arch/arm/dts/fsl-imx8qm-mek-auto.dts
@@ -54,6 +54,10 @@
     };
};
+&lpuart0 {
+   status = "disabled";
+};
/delete-node/ &pd_dc0;
/delete-node/ &pd_dc1;
/delete-node/ &pd_isi_ch0;
•Disable "DEBUG" in Trusty OS to remove TA logs like below:
diff --git a/project/imx8-inc.mk b/project/imx8-inc.mk
index e58c15a..8c20e99 100644
--- a/project/imx8-inc.mk
+++ b/project/imx8-inc.mk
@@ -16,7 +16,7 @@
 LOCAL_DIR := $(GET_LOCAL_DIR)
-DEBUG := 1
+DEBUG := 0
 WITH_SMP := 1
 SMP_MAX_CPUS ?= 4
 SMP_CPU_CLUSTER_SHIFT ?= 2
```

4. Modify the Linux bootargs in build system. Append `loglevel=0` in it, which will prevent the dmesg printing on the console when the system is booted.
5. By default, the images are built by `userdebug` build. When it is changed to `user` build, it saves about 0.5 seconds boot time.

Note:

When setting `loglevel=0`, the debug message is not displayed directly to the console. To check it, however, you can use the `$dmesg` command in the shell to output it.

8.9.4 How to build system.img with squashfs files system type

The default file system of `system.img` is ext4. After the `system.img` file system type is changed to `squashfs`, the `system.img` size can be reduced to about 50%. Smaller storage size costs more CPU resource but less eMMC IO operation, so this is a balanced option between IO and CPU loading. By default, this is not enabled. If the target device has a strong CPU but weak eMMC, `squashfs` is an option for boot time tuning.

To change the default file system type to `squashfs`, perform the following steps:

1. Add the following Linux kernel macro in `${MY_ANDROID}/vendor/nxp-opensource/kernel_imx/arch/arm64/configs/android_car_config`:
   - `CONFIG_SQUASHFS=y`
   - `CONFIG_SQUASHFS_LZ4=y`
   - `CONFIG_SQUASHFS_XATTR=y`
   - `CONFIG_SQUASHFS_DECOMP_MULTI=y`
2. Add the following configuration in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk`:

```
BOARD_SYSTEMIMAGE_FILE_SYSTEM_TYPE := squashfs
```

Rebuild the whole images for the `mek_8q` board. It can shorten the automotive boot time for the i.MX 8QuadMax MEK Board, but there is no boot time optimization on the i.MX 8QuadXPlus MEK Board.

8.9.5 How to measure the boot time

Per the definition of the boot time described in Section 8.9.1, users need to measure the boot time duration from power-on to when the display shows the desktop.

Pay attention to the following:

- Keep the device in lock state by `$fastboot oem lock`.
- Make sure that the device is powered down safely. `$setprop sys.powerctl shutdown` makes the device power down safely. Or the fsck scans the storage during the booting time and it costs 1 to 2 seconds.
- Make sure the action of Section 8.9.3 has been done.

The boot time is different for different boot that the AOSP Android Framework schedules the system services. To evaluate the boot time performance, calculate the average values based on about 50 times boot. According to the boot time analyzing tools provided by Google (https://source.android.com/devices/tech/perf/boot-times), evaluate the time by that first `sys.boot_completed=1` shown from initialization logs. The process is easier. You can also get the bootanylaze tool from `${MY_ANDROID}/system/extras/boottime_tools/bootanalyze`. To make sure that this log is printed, append `printk.devkmsg=on` in bootargs. Based on the timestamp for the first time, `sys.boot_completed=1` is displayed in the log. This is the boot time from kernel started to Android Framework boot completed.

Then, evaluate the boot time for the modules, which boot before the Linux kernel. It is easy to evaluate it by adding the following codes to print the timer value before jumping to Linux in U-Boot.

```
diff --git a/arch/arm/lib/bootm.c b/arch/arm/lib/bootm.c
index 96cac780b5..aae07a98ba 100644
--- a/arch/arm/lib/bootm.c
+++ b/arch/arm/lib/bootm.c
@@ -423,6 +423,8 @@ int do_bootm_linux(int flag, int argc, char * const argv[],
        return 0;
 }
```

```
+       printf("%d\n", get_timer(0));
+
if (flag & (BOOTM_STATE_OS_GO | BOOTM_STATE_OS_FAKE_GO)) {
        boot_jump_linux(images, flag);
        return 0;
```

The boot evaluation by software is the sum of the timestamp for the first time we see `sys.boot_completed=1` and the timer values printed in U-Boot.

8.10 Configuration for the load orders of driver modules

8.10.1 Why does Android Automotive have driver load orders

As the boot time performance of Android Automotive is important, make Linux kernel boot as soon as possible to enable some critical services earlier. Therefore, some drivers that are not critical for the Android Automotive booting are not loaded during the early boot stage. The set of drivers is built into kernel modules during build time and are loaded and probed after the Android Automotive key service boots successfully. This makes the display and UI ready earlier.

In this release, the following module-related drivers are probe before the initialization process starts:

- Camera (only in `mek_8q_car`)
- USB
- Wi-Fi

8.10.2 How does the non-critical driver load

In i.MX Android Automotive, there are two kinds of build. The `mek_8q_car` and `mek_8q_car2`. `mek_8q_car` have special design to support the EVS features, which use the Arm Cortex-M4 core to handle camera-related modules before the Android display related service is ready. Therefore, `mek_8q_car` and `mek_8q_car2` loads different driver modules in different stages.

In i.MX Android Automotive, all kernel driver modules are loaded in `init.rc` by the script named init.insmod.sh.

For `mek_8q_car`, when the EVS service running in the Arm Cortex-M4 core releases the hardware resource for camera modules, Android Automotive loads the camera-related driver modules. This typically happens when the `late_start` service is triggered in `init.rc`, if the EVS service running in Android Automotive is initialized successfully. This part of drivers is listed in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/setup.core.cfg`. After the core drivers are probed successfully, it triggers low-priority driver modules to load and probe by triggering the service named `boot_completed_main_sh`, which loads drivers listed in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/setup.main.cfg` in `init.rc`. The "main" drivers are the rest of driver modules.

For `mek_8q_car2`, not like `mek_8q_car`, it has no "core" driver modules to be loaded and probed during the boot process. As all necessary camera driver modules are built in inside the kernel image, like `mek_8q_car`, the "main" drivers are the same ones like rfkill for BT, USB, and Wi-Fi. The driver load and probe are triggered once `sys.boot_completed` property is set to be 1. This is handled in `init.rc`.

8.10.3 How to change driver load orders

Generally, the driver follows the priority below to be loaded:

- Built-in
- Listed in `early.init_car_gki.cfg`

- Listed in `setup.main.gki.cfg`

In each cfg file, the drivers are loaded one by one. To change the driver load orders, in `early.init.cfg` or `setup.main.cfg`, just change the text list order. If some built-in drivers need to be loaded in low priority, follow the changes below:

- In the kernel defconfig file, mark specific CONFIG to be `m` instead of *y* .
- Modify the `BOARD_VENDOR_KERNEL_MODULES` in `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk` to copy the specific .ko files to the target image.
- Add the driver module name in `early.init_car_gki.cfg` or `setup.main.gki.cfg` based on its loading priority.

8.11 Dual-bootloader configuration

8.11.1 Dual-bootloader layout

Dual-bootloader feature splits the default `u-boot.imx` into two parts: `spl.bin` and `bootloader.img`. The `spl.bin` goes to the bootloader0 partition, which is managed by U-Boot itself. The bootloader.img goes to the bootloader_a/bootloader_b partitions, which are managed by GPT and thus gets a chance to be updated.

The layout of dual-bootloader is as follows (taking i.MX 8Quad as an example):

The `bootloader.img` contains U-Boot proper, Arm Trusted Firmware, and Trusty OS. All of them can be updated easily through OTA to fix some power or security issues.

8.11.2 Configuring dual-bootloader

Dual-bootloader feature is enabled for Android Automotive by default. It is enabled by configuring `CONFIG_DUAL_BOOTLOADER` in U-Boot. Take i.MX 8Quad as an example:

```
diff --git a/configs/imx8qm_mek_androidauto_trusty_defconfig b/configs/
imx8qm_mek_androidauto_trusty_defconfig
index 82ec5ca..e0b210e 100644
--- a/configs/imx8qm_mek_androidauto_trusty_defconfig
+++ b/configs/imx8qm_mek_androidauto_trusty_defconfig
@@ -170,4 +170,4 @@ CONFIG_APPEND_BOOTARGS=y
CONFIG_LIBAVB=y
CONFIG_SHA256=y
CONFIG_SPL_MMC_WRITE=y
+CONFIG_DUAL_BOOTLOADER=y
diff --git a/configs/imx8qxp_mek_androidauto_trusty_defconfig b/configs/
imx8qxp_mek_androidauto_trusty_defconfig
index 30fe32d..2f709d2 100644
--- a/configs/imx8qxp_mek_androidauto_trusty_defconfig
+++ b/configs/imx8qxp_mek_androidauto_trusty_defconfig
@@ -179,4 +179,4 @@ CONFIG_APPEND_BOOTARGS=y
CONFIG_LIBAVB=y
CONFIG_SHA256=y
CONFIG_SPL_MMC_WRITE=y
+CONFIG_DUAL_BOOTLOADER=y
```

Then, `imx-mkimage` needs to pack `spl.bin` and `bootloader.img` separately. Taking i.MX 8QuadMax and i.MX 8QuadXPlus as an example, two targets are used to handle the dual-bootloader image generation with Cortex-M4 images in `imx-mkimage`:

```
i.MX 8QuadMax:    flash_b0_spl_container_m4_1_trusty
```

```
i.MX 8QuadXPlus:  flash_all_spl_container_ddr_car
```

When Trusty OS is enabled, bootloader rollback index can be used to prevent rollback attack. For more details to set the bootloader rollback index, see Section 2.3.5 in the *i.MX Android Security User's Guide* (ASUG).

Besides, after enabling dual-bootloader, the steps to sign images with the CST tool are different. For more information, see Section 2.1 in the *i.MX Android Security User's Guide* (ASUG).

8.12 Miscellaneous configuration

8.12.1 Changing boot command line in boot.img

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

You can change this by changing BOARD_KERNEL_CMDLINE's definition in the `${MY_ANDROID}/device/nxp/imx8q/mek_8q/BoardConfig.mk` file.

8.12.2 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 make changes to 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

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 14 (2023) platform release using kernel versions v5.15 or higher are required to ship with the GKI kernel.

The following boards have enabled GKI:

- i.MX 8QuadMax MEK Board
- i.MX 8QuadXPlus MEK Board
- i.MX 95 EVK/Verdin Board

9.1 Changes after GKI enabled

- `boot.img`
  After GKI is enabled, `boot.img` is a composite image, which includes the Android Open Source Project (AOSP) generic kernel image 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, and then signed with the AVB key to generate the final `boot.img`.
  By default, the UUU and fastboot script flash this image.
  To build `boot.img`, run `./imx-make.sh` or `make bootimage`.
- `system_dlkm.img`
  `system_dlkm.img` is signed by Google using the kernel build-time key pair and is compatible only with the GKI it is built with. There is no ABI stability between `boot.img` and `system_dlkm.img`. For modules to load correctly during runtime, `boot.img` and `system_dlkm.img` must be built and updated together.
- `boot-imx.img`
  `boot-imx.img` is built from the i.MX kernel tree for debugging purposes. By default, it is built out by `imx-make.sh` with `TARGET_IMX_KERNEL=true`, and 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} ${build_vendordlkmimage} || exit
if [ -n "${build_bootimage}" ] || [ ${build_whole_android_flag} -eq 1 ]; then
if [ ${TARGET_PRODUCT} = "evk_8mp" ] || [ ${TARGET_PRODUCT} = "evk_8mn" ] \
|| [ ${TARGET_PRODUCT} = "evk_8ulp" ] || [ ${TARGET_PRODUCT} = "mek_8q" ] \
|| [ ${TARGET_PRODUCT} = "mek_8q_car" ] || [ ${TARGET_PRODUCT} =
 "mek_8q_car2" ] \
|| [ ${TARGET_PRODUCT} = "evk_95_car" ] || [ ${TARGET_PRODUCT} =
 "evk_95_car2" ] \
|| [ ${TARGET_PRODUCT} = "evk_8mm" ] || [ ${TARGET_PRODUCT} = "evk_8mq" ]; then
    if [ ${sign_gki} -eq 1 ]; then
        mv ${OUT}/boot.img ${OUT}/boot-imx.img
        make bootimage
    fi
fi
fi
```

  To build `boot-imx.img`, run `./imx-make.sh` or `TARGET_IMX_KERNEL=true make bootimage && mv ${OUT}/boot.img ${OUT}/boot-imx.img`.
- Kernel defconfig
  The kernel `.config` is generated by one generic `gki_defconfig` along with one board-specific config, like `imx8q_car_gki.fragment`.
- Driver modules
  As GKI requires, all vendor drivers need to be built as modules. Their configs are set as "`m`" in the board-specific configuration file mentioned above. In addition, explicitly install those modules on board by adding them to the following two Android predefined macros. For example, see `${MY_ANDROID}/device/nxp/imx8q/mek_8q/SharedBoardConfig.mk`:
  - `BOARD_VENDOR_RAMDISK_KERNEL_MODULES`
    Modules under this macro are copied to `${MY_ANDROID}/out/target/product/mek_8qvendor_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/mek_8q/vendor_dlkm/lib/modules, and then built as vendor_dlkm.img. They are installed later than vendor_ramdisk, after the Android filesystem is ready.

Note:

Due to SoC errata TKT340553 in i.MX 8QuadMax, GKI is not fully enabled. The boot_8q_car.img and system_dlkm_staging_8q_car are built locally for both i.MX 8QuadMax and i.MX 8QuadXPlus.

9.2 How to add new drivers

To add new drivers, perform the following steps:

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

```
diff --git a/arch/arm64/configs/imx8q_car_gki.fragment b/arch/arm64/configs/
imx8q_car_gki.fragment
index 594bf1228f72..b5585c423bbf 100644
--- a/arch/arm64/configs/imx8q_car_gki.fragment
+++ b/arch/arm64/configs/imx8q_car_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:
   Note: *If other driver modules depend on them, put them before others.*

```
diff --git a/imx8q/mek_8q/SharedBoardConfig.mk b/imx8q/mek_8q/
SharedBoardConfig.mk
index df7850b0b285..84d136c224cd 100644
--- a/imx8q/mek_8q/SharedBoardConfig.mk
+++ b/imx8q/mek_8q/SharedBoardConfig.mk
@@ -102,6 +104,10 @@ endif
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 \
```

3. Fix the 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 the symbols are in the core kernel code (which means not in the loadable modules), such changes must upstream to the AOSP GKI Kernel tree.*
   b. Add symbols to the AOSP GKI Kernel tree gki/aarch64/abi.stg.

9.3 How to build GKI locally

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

1. Prepare the GKI Kernel build repository (taking the 6.6 kernel as an example):

```
mkdir gki && cd gki
```

```
repo init -u https://android.googlesource.com/kernel/manifest -b common-
android15-6.6
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`.

```
{MY_ANDROID}/vendor/nxp-opensource/imx-gki/debug_patches/0001-MA-19811-
ttyimx_
earlycon-Support-lpuart-earlycon.patch
```

3. Build the GKI Image.

```
tools/bazel run //common:kernel_aarch64_dist
```

   The GKI `boot.img` is obtained from `out/kernel_aarch64/dist/boot.img`. The GKI `system_dlkm_staging_archive.tar.gz` is obtained from `out/kernel_aarch64/dist/system_dlkm_staging_archive.tar.gz`.
4. Build Android `boot.img` and `system_dlkm.img`.

```
cp out/kernel_aarch64/dist/boot.img {MY_ANDROID}/vendor/nxp-opensource/imx-
gki/boot_8q[95]_car.img
cp system_dlkm_staging_archive.tar.gz {MY_ANDROID}/vendor/nxp-opensource/imx-
gki/system_dlkm_staging_archive.tar.gz
```

5. Build Android `boot_8q_car.img` and `system_dlkm_8q.img` (only for i.MX 8QuadXPlus and i.MX 8QuadMax MEK boards).
   To address TKT340553 Errata and support for multiple-state domains, i.MX 8QuadXPlus and i.MX 8QuadMax require `boot_8q_car.img` and `system_dlkm_8q.img`. This `boot_8q_car.img` and `system_dlkm_staging_8q_car` are built locally with `aosp/android16-6.12`. Then, the following patches from `{MY_ANDROID}/ vendor/nxp-opensource/imx-gki/boot_8q_patches` are added:

```
0001-MLK-16005-2-arm64-tlb-add-the-SW-workaround-for-i.MX.patch
0002-ANDROID-ABI-Update-symbol-list-for-imx.patch
0003-PM-Domains-Move-the-Subdomain-check-into-_genpd_powe.patch
0004-PM-Domains-Support-enter-deepest-state-for-multiple-.patch
0005-PM-Domains-Choose-the-deepest-state-to-enter-if-no-d.patch
0006-PM-Domains-remove-no-governor-for-states-warning.patch
0007-Car-support-enable-performance-gov.patch
```

   Then update the AOSP symbol list according to Section 9.4. These patches are going upstream.
6. Build Android `boot_95_car.img` and `system_dlkm_95.img` (only for i.MX 95 board).
   To avoid receive timeouts when using SCMI for data transfer, i.MX 95 requires `boot_95_car.img` and `system_dlkm_95.img`. The `boot_95_car.img` and `system_dlkm_staging_95.img` are built locally with `aosp/ android16-6.12`. Then, the following patch from `{MY_ANDROID}/vendor/nxpopensource/imx-gki/boot_95_patches` are added:

```
0001-ILIE-12-include-videodev2.h-Add-meta-formats-used-fo.patch
0002-ILIE-17-media-v4l2-core-Add-meta-neoisp-formats-desc.patch
0003-PCI-dwc-Fix-resume-failure-if-no-EP-is-connected-at-.patch
0004-LF-13477-PCI-dwc-i.MX6QP-suspend-resume-hang-on-PCIe.patch
0005-Car-support-enable-performance-gov.patch
```

   Then update the AOSP symbol list according to Section 9.4. These patches are going upstream.

9.4 How to export new symbols

AOSP GKI image only exports those symbols listed at `gki/aarch64/abi.stg`. 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. Check the AOSP symbol list (`gki/aarch64/abi.stg`)

```
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-
android16-6.12
repo sync
cd common
```

   Check `gki/aarch64/abi.stg` for the symbol you need. If it is already there, you only need to find a release from Android GKI Release Builds that includes the required symbol. Then, see Section 9.5 to update `boot.img` and `system_dlkm.img`.

2. Generate the device symbol list (`gki/aarch64/symbols/imx`). If you do not find the symbol you need in `gki/aarch64/abi.stg`, continue to work in the common folder.
   Note: *Switch the kernel in this common folder from AOSP to its own device kernel and apply all your local patches that may require new symbols.*

```
git remote add device https://github.com/nxp-imx/linux-imx.git
git remote update
git fetch device --tags
git checkout automotive-15.0.0_2.1.0
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
tools/bazel run //common:imx_abi_update_symbol_list
```

   Then, `common/gki/aarch64/symbols/imx` is updated.

3. Update the AOSP symbol list (`android/abi_gki_aarch64.stg`).

```
cd gki
cp common/gki/aarch64/symbols/imx /tmp/imx
cd common
```

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

```
git reset --hard
git checkout aosp/android16-6.12
cp /tmp/imx gki/aarch64/symbols/imx
```

   Note: *Verify new symbols. If any existing symbols are removed, add them back. Then, keep what you need and remove the extras; otherwise,* `kernel_aarch64_abi_update` *or upstream will fail.*

```
cd ..
tools/bazel run //common:kernel_aarch64_abi_update
```

   Then, `common/gki/aarch64/abi.stg` is updated.

4. Build Android `boot_8q_car.img` and `system_dlkm.img` locally.

```
cp out/kernel_aarch64/dist/boot.img {MY_ANDROID}/vendor/nxp-opensource/imx-
gki/boot_8q_car.img
cp system_dlkm_staging_archive.tar.gz {MY_ANDROID}/vendor/nxp-opensource/imx-
gki/system_dlkm_staging_archive.tar.gz
cd {MY_ANDROID}/vendor/nxp-opensource/imx-gki
```

```
tar -xzf system_dlkm_staging_archive.tar.gz -C system_dlkm_staging_8q_car
```

5. If you want AOSP released GKI image to export these symbols, upstream the two files `gki/aarch64/abi.stg` and `gki/aarch64/symbols/imx` to AOSP.

9.5 How to update the GKI image

To update the GKI image, perform the following steps:

1. 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
```

2. Download GKI `system_dlkm_staging_archive.tar.gz` from Google. Put `system_dlkm_staging_archive.tar.gz` in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/gki/system_dlkm_staging_archive.tar.gz`. Unzip `system_dlkm_staging_archive.tar.gz to system_dlkm_staging`. Run the following command to build `system_dlkm.img`.

```
make system_dlkmimage
```

3. Get `boot.img` and `system_dlkm_staging_archive.tar.gz` from the https://source.android.com/docs/core/architecture/kernel/gki-release-builds.

10 Acronyms

Table 25. Acronyms

Acronym	Description
AOSP	Android Open Source Project
BT	Bluetooth
CST	(NXP) Code Signing Tool
eMMC	Embedded Multi-Media Card
EVK	Evaluation Kit
EVS	Android Exterior View System
GAS	Google Automotive Services
GCC	GNU Compiler collection
GPT	GUID partition table
HVAC	Heating, ventilation, and air conditioning
MEK	Multisensory Enablement Kit
OS	Operating system
PC	Personal (host) computer
SoC	System on Chip
SPL	U-Boot Secondary Program Loader
OTA	Over-The-Air programming
SOF	Sound Open Firmware
U-Boot	Universal Boot Loader

11 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

12 Revision History

This table provides the revision history.

Table 26. Revision history

Document ID	Release date	Description
UG10176 v.automotive-15.0.0_2.1.0	14 October 2025	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release, i.MX 95 EVK (Silicon 19x19 Revision A1, B0) and i.MX 95 Verdin (Silicon 19x19 Revision A1, B0) Beta
UG10176 v.automotive-15.0.0_1.3.0	21 July 2025	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release, i.MX 95 EVK (Silicon Revision A1 19x19) and i.MX 95 Verdin (Silicon Revision A1 19x19) Beta
UG10176 v.automotive-15.0.0_1.1.0	15 May 2025	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release, i.MX 95 EVK (Silicon Revision A1 19x19) Beta, and i.MX 95 Verdin (Silicon Revision A1 19x19) Experimental
UG10176 v.automotive-14.0.0_2.3.0	23 January 2025	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release, i.MX 95 EVK (Silicon Revision A1 19x19) Beta, and i.MX 95 Verdin (Silicon Revision A1 19x19) Experimental
UG10176 v.automotive-14.0.0_2.1.0	7 November 2024	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release, i.MX 95 EVK (Silicon Revision A1 19x19) Alpha (EAR)
AAUG_14.0.0_1.1.0	20 June 2024	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-13.0.0_2.3.0	4 January 2024	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release

Table 26. Revision history...*continued*

Document ID	Release date	Description
automotive-13.0.0_2.1.0	10/2023	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-13.0.0_1.3.0	07/2023	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-13.0.0_1.1.0	05/2023	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-12.1.0_1.1.0	12/2022	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-12.0.0_2.1.0	09/2022	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-12.0.0_1.1.0	06/2022	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-11.0.0_2.5.0	03/2022	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-11.0.0_2.3.0	12/2021	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0, C0) GA release
automotive-11.0.0_2.1.0	11/2021	Added the examples for i.MX 8QuadXPlus and upgraded the tool version
android-11.0.0_1.1.0-AUTO	01/2021	i.MX 8QuadXPlus/8QuadMax MEK GA release
android-10.0.0_2.4.0	07/2020	i.MX 8QuadMax MEK GA release
android-10.0.0_2.2.0-AUTO	06/2020	i.MX 8QuadXPlus/8QuadMax MEK GA release
automotive-10.0.0_1.1.0	03/2020	Deleted the Android 10 image
automotive-10.0.0_1.1.0	03/2020	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0) GA release
P9.0.0_2.1.0-AUTO-ga	08/2019	Updated the location of the SCFW porting kit
P9.0.0_2.1.0-AUTO-ga	04/2019	i.MX 8QuadXPlus/8QuadMax Automotive GA release
P9.0.0_1.0.2-AUTO-beta	01/2019	i.MX 8QuadXPlus/8QuadMax Automotive Beta release
P9.0.0_1.0.2-AUTO-alpha	11/2018	i.MX 8QuadXPlus/8QuadMax Automotive Alpha release
O8.1.0_1.1.0_AUTO-beta	05/2018	i.MX 8QuadXPlus/8QuadMax Beta release
O8.1.0_1.1.0_AUTO-EAR	02/2018	Initial release

Legal information

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.

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.

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 https://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.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

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.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Contents