

1 Overview

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

- Instructions for building from sources or using pre-built images.
- Copying the 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

The minimum recommended system requirements are as follows:

- 16 GB RAM
- 300 GB hard disk

2.1 Setting up your computer

To build the Android source files, use a computer running the Linux OS. The Ubuntu 16.04 64bit version and openjdk-8-jdk of Ubuntu are the most tested environment for the Android 10.0 build.

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

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

```
$ sudo apt-get install uuid uuid-dev
$ sudo apt-get install zlib1g-dev liblz-dev
$ sudo apt-get install liblzo2-2 liblzo2-dev
$ sudo apt-get install lzop
$ sudo apt-get install git-core curl
$ sudo apt-get install u-boot-tools
$ sudo apt-get install mtd-utils
$ sudo apt-get install android-tools-fsutils
$ sudo apt-get install openjdk-8-jdk
$ sudo apt-get install device-tree-compiler
$ sudo apt-get install gdisk
$ sudo apt-get install liblz4-tool
$ sudo apt-get install m4
$ sudo apt-get install libz-dev
$ sudo apt-get install bison
```

Contents

1 Overview.....	1
2 Preparation.....	1
3 Building the Android platform for i.MX...	2
4 Running the Android Platform with a Prebuilt Image.....	8
5 Programming Images.....	10
6 Booting.....	13
7 Over-The-Air (OTA) Update.....	17
8 Customized Configuration.....	21
9 Revision History.....	48



```
$ sudo apt-get install flex
$ sudo apt-get install libssl-dev
```

NOTE

If you have trouble installing the JDK in Ubuntu, see [How to install misc JDK in Ubuntu for Android build](#).

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

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

2.2 Unpacking the Android release package

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

```
$ cd ~ (or any other directory you like)
$ tar xzvf imx-automotive-10.0.0_1.1.0.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 [CodeAurora Forum repository](#).
- AOSP Android public source code, which is maintained in [android.gogglesource.com](#).
- NXP i.MX Android proprietary source code package, which is maintained in [www.NXP.com](#).

Assume you have i.MX Android proprietary source code package `imx-automotive-10.0.0_1.1.0.tar.gz` under `~/` directory. To generate the i.MX Android release source code build environment, execute the following commands:

```
$ mkdir ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=${PATH}:~/bin
$ source ~/imx-automotive-10.0.0_1.1.0/imx_android_setup.sh
# By default, the imx_android_setup.sh script will create the source code build environemnt in the
# folder ~/android_build
# ${MY_ANDROID} will be refered as the i.MX Android source code root directory in all i.MX Android
# release documentation.
$ export MY_ANDROID=~/android_build
```

3.2 Building Android images

Building the Android image is performed when the source code has been downloaded (Section 3.1 [Getting i.MX Android release source code](#)).

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

Command `lunch <BuildName-BuildMode>` is used to set up the build configuration.

After the two commands above are executed, then the build process starts. The behaviour of the i.MX Android build system used to be aligned with the original Android system. The command of `make` can start the build process and all images will be built out. There are some differences. A shell script named `imx-make.sh` is provided and its symlink file can be found under 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 Android10.0 platform. Some host tools necessary for building U-Boot/kernel now cannot be used in Android build system, which is under the control of `soong_ui`, so U-Boot/kernel cannot be built together with Android images. Google also recommends to use prebuilt binaries for U-Boot/kernel in Android build system. It takes some steps to build U-Boot/kernel to binaries and 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 do these steps to simplify the build work. After U-Boot/kernel are compiled, any build commands in standard Android can be used.

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

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

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

Table 1. Build names

Build name	Description
mek_8q_car	i.MX 8QuadMax/8QuadXPlus MEK Board with EVS function enabled in the Arm Cortex-M4 CPU core
mek_8q_car2	i.MX 8QuadMax/8QuadXPlus MEK Board without EVS function enabled in the Arm Cortex-M4 CPU core

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

Table 2. Build types

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

Android build steps are as follows:

1. Prepare the build environment for U-Boot.

This step is optional if there is still GCC cross-compile tool chain in AOSP codebase, and this tool chain outputs the following log indicating that this tool chain will be deprecated.

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

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

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

Then, decompress the file into a path on local disk. For example, to `/opt/`, export a variable named `AARCH64_GCC_CROSS_COMPILE` to point to the tool as follows:

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

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

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

2. 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 "7-2018-q2-update" version. Extract it to your installation directory, for example, `/opt`. Then, export the directory as `export ARMGCC_DIR=/opt/gcc-arm-none-eabi-7-2018-q2-update` and add this to `/etc/profile`.

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-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.

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

The output of `make` command will be written to 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` & `BUILD_NUMBER` changing, update `build_id.mk` in the `${MY_ANDROID}/device/fsl/imx8q/mek_8q/` directory. For detailed steps, check [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 (including `init`, `init.rc`). Mounted at `/`.
- `system/`: Android system binary/libraries. Mounted at `/system`.
- `data/`: Android data area. Mounted at `/data`.
- `recovery/`: root file system when booting in "recovery" mode. Not used directly.

- dtbo-imx8qm.img: board's device tree binary. It is used to support the LVDS-to-HDMI display for i.MX 8QuadMax MEK.
- dtbo-imx8qm-md.img: Board's device tree binary. It is used to support multiple-display feature for i.MX 8QuadMax MEK.
- dtbo-imx8qxp.img: board's device tree binary. It is used to support the LVDS-to-HDMI display 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-md.img: Android Verify boot metadata image for dtbo-imx8qm.img. It is used to support multiple-display feature for i.MX 8QuadMax MEK.
- 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.
- ramdisk.img: Ramdisk image generated from "root/". Not directly used.
- system.img: EXT4 image generated from "system/". Can be programmed to "SYSTEM" partition on SD/eMMC card with "dd".
- 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 includes Seco firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadMax MEK.
- spl-imx8qxp.bin: a composite image includes Seco firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK.
- spl-imx8qm-secure-unlock.bin: a composite image includes Seco firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadMax MEK. It is a demonstration secure unlock mechanism.
- spl-imx8qxp-secure-unlock.bin: a composite image includes Seco firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK. It is a demonstration secure unlock mechanism.
- 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-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.
- 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 secure unlock mechanism.
- 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. It is a demonstration secure unlock mechanism.
- u-boot-imx8qm-mek-uuu.imx: U-Boot image used by UUU for i.MX 8QuadMax MEK. It is not flashed to MMC.
- u-boot-imx8qxp-mek-uuu.imx: U-Boot image used by UUU for i.MX 8QuadXPlus MEK. It is not flashed to MMC.
- vendor.img: vendor image, which holds platform binaries. Mounted at /vendor.
- boot.img: a composite image that includes the kernel Image, ramdisk, and boot parameters.
- 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 [Building U-Boot images](#).
- To build the kernel ulmage separately, see [Building a kernel image](#).
- To build boot.img, see [Building boot.img](#).
- To build dtbo.img, see [Building dtbo.img](#).

3.2.1 Configuration examples of building i.MX devices

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

Table 3. i.MX device lunch examples

Build name	Description
i.MX 8QuadXPlus/8QuadMax MEK Board with EVS function enabled in the Arm Cortex-M4 CPU core	\$ lunch mek_8q_car-userdebug
i.MX 8QuadMax/8QuadXPlus MEK Board without EVS function enabled in the Arm Cortex-M4 CPU core	\$ lunch mek_8q_car2-userdebug

3.2.2 Build mode selection

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

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

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

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

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

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

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

There are two methods for the build of Android image.

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

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

The commands below can achieve the same result:

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

Table 4. Android system image production build method

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK with EVS function enabled in the Cortex-M4 CPU core	mek_8q_car-userdebug
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK without EVS function enabled in the Cortex-M4 CPU core	mek_8q_car2-userdebug

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

```
$ ./imx-make.sh bootloader kernel -j4
$ make otapackage -j4
```

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

3.3 Building U-Boot images

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

```
# 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-userdebug
$ ./imx-make.sh bootloader -j4
```

3.4 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-userdebug
$ ./imx-make.sh kernel -j4
```

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

3.5 Building boot.img

`boot.img` can be built only after kernel is built as described in Section [Building a kernel image](#). 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-userdebug
$ make bootimage -j4
```

3.6 Building dtbo.img

DTBO image holds the device tree binary of the board.

`dtbo.img` can be built only after kernel is built as described in Section [Building a kernel image](#). 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-userdebug
$ make dtboimage -j4
```

4 Running the Android Platform with a Prebuilt Image

Table 5. Image packages

Image package	Description
<code>android_automotive-10.0.0_1.1.0_image_8qmek.tar.gz</code>	Prebuilt-image for i.MX 8QuadXPlus/8QuadMax MEK board with EVS function enabled in the Arm Cortex-M4 CPU core, which includes NXP extended features.
<code>android_automotive-10.0.0_1.1.0_image_8qmek2.tar.gz</code>	Prebuilt-image and UUU script files for i.MX 8QuadMax/8QuadXPlus MEK board without EVS function enabled in the Arm Cortex-M4 CPU core, which includes NXP extended features.

The following tables list the detailed contents of `android_automotive-10.0.0_1.1.0_image_8qmek.tar.gz` image package. Images are almost the same for i.MX 8QuadMax and i.MX 8QuadXPlus MEK with/without EVS function enabled in the Arm Cortex-M4 CPU core, except that there is no Xen support for `android_automotive-10.0.0_1.1.0_image_8qmek.tar.gz`.

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

Table 6. Images for i.MX 8QuadXPlus MEK

i.MX 8QuadXPlus/8QuadMax MEK image	Description
<code>spl-imx8qm.bin</code>	The secondary program loader (SPL) for i.MX 8QuadMax MEK board.

Table continues on the next page...

Table 6. Images for i.MX 8QuadXPlus MEK (continued)

spl-imx8qxp.bin	The secondary program loader (SPL) for i.MX 8QuadXPlus MEK board.
spl-imx8qm-secure-unlock.bin	The secondary program loader (SPL) for i.MX 8QuadMax MEK board.
spl-imx8qxp-secure-unlock.bin	The secondary program loader (SPL) for i.MX 8QuadXPlus MEK board.
bootloader-imx8qm.img	The next loader image after SPL for the i.MX 8QuadMax MEK board.
bootloader-imx8qxp.img	The next loader image after SPL for the i.MX 8QuadXPlus 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-secure-unlock.img	The next loader image after SPL for i.MX 8QuadXPlus MEK board, including the arm trusted firmware, Trusty OS, and U-Boot proper.
u-boot-imx8qm-mek-uuu.imx	Bootloader used by UUU for i.MX 8QuadMax MEK board. It is not flashed to MMC.
u-boot-imx8qxp-mek-uuu.imx	Bootloader used by UUU for i.MX 8QuadXPlus MEK board. It is not flashed to MMC.
boot.img	Boot image to support LVDS-to-HDMI display.
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 i.MX 8QuadMax MEK board to support LVDS-to-HDMI display
vbmeta-imx8qm-md.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support multiple-display feature.
vbmeta-imx8qxp.img	Android Verify Boot metadata image for i.MX 8QuadXPlus MEK board to support LVDS-to-HDMI display
system.img	System Boot image
vendor.img	Vendor image
product.img	Product image.
dtbo-imx8qm.img	Device tree image for i.MX 8QuadMax
dtbo-imx8qm-md.img	Device Tree Image for i.MX 8QuadMax MEK.
dtbo-imx8qxp.img	Device tree image for i.MX 8QuadXPlus
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 boot loader code to change any default boot arguments.

5 Programming Images

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

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

- UUU to download all images to the eMMC 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 system image: system.img
- Android vendor image: vendor.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, etc. In normal boot mode, the root file system is mounted from the system partition. In recovery mode, the root file system is mounted from the boot partition.

Table 7. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader0	0 KB (i.MX 8QuadMax) or 32 KB (i.MX 8QuadXPlus)	4 MB	N/A	spl.bin

Table continues on the next page...

Table 7. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
1	bootloader_a	8 MB	4 MB	N/A	bootloader.img
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	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
6	boot_b	Follow boot_a	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
7	system_a	Follow boot_b	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
8	system_b	Follow system_a	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
9	misc	Follow system_b	4 MB	N/A	For recovery storage bootloader message, reserve
10	metadata	Follow metafooter	2 MB	N/A	For system slide show
11	persistdata	Follow metadata	1 MB	N/A	Option to operate unlock \unlock
12	vendor_a	Follow persistdata	512 MB	EXT4. Mount at / vendor	vendor.img
13	vendor_b	Follow vendor_a	512 MB	EXT4. Mount at / vendor	vendor.img
14	product_a	Follow vendor_b	1792 MB	EXT4. Mount at / product	product.img
15	product_b	Follow product_a	1792 MB	EXT4. Mount at / product	product.img
16	userdata	Follow vendor_b	Remained space	EXT4. Mount at / data	Application data storage for system application, and for internal media partition, in /mnt/sdcard/ dir.
17	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
18	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata

Table continues on the next page...

Table 7. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
19	vbmeta_b	Follow vbmeta_a	1 MB	N/A	For storing the verify boot's metadata

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

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 *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 firstly, 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 board. It requires the target board 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 8. fastboot_imx_flashall script

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

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

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

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

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

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

The way to use these scripts is follows:

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

```
Options:
  -h           Displays this help message
  -f soc_name  Flashes the Android image file with soc_name
  -a           Only flashes the image to slot_a
  -b           Only flashes the image to slot_b
  -c card_size Optional setting: 28
               If it is not set, use partition-table.img (default).
               If it is set to 28, use partition-table-28GB.img for 32 GB SD card.
```

```

Make sure that the corresponding file exists on your platform.
-m          Flashes the Cortex-M4 image.
-u uboot_feature  Flashes U-Boot or SPL&bootloader images with "uboot_feature" in their names.
                For Standard Android:
                    If the parameter after "-u" option contains the string of "dual", the
                    SPL&bootloader image is flashed;
                    otherwise, U-Boot image is flashed.
                For Android Automotive:
                    Only dual-bootloader feature is supported. By default, SPL&bootloader
                    image is flashed.
-d dtb_feature  Flashes dtbo, vbmeta, and recovery image file with "dtb_feature" in their names.
                If 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 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-10.0/mek_8q_car/
```

Option explanations:

- -f imx8qm: Flashes images for i.MX 8QuadMax MEK Board.
- -a: Only flashes slot a.
- -e: Erases user data after all image files are flashed.
- -D /imx_android-10.0/mek_8q_car/: Images to be flashed are in the directory of /imx_android-10.0/mek_8q_car/.

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 9. 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 10. Boot switch settings for i.MX 8QuadXPlus

i.MX 8QuadXPlus boot switch	download Mode (MFGTool 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.

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

To clear the bootargs environment, use the following command:

```
U-Boot > setenv bootargs
U-Boot > saveenv          #Save the environments
```

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.

6.2 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 [Kernel command line \(bootargs\)](#), 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 11. 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=ttyLPO
init	Tells kernel where the init file is located.	init=/init	All use cases. "init" in the Android platform is located in "/" instead of in "/sbin".
androidboot.console	The Android shell console. It should be the same as console=.	androidboot.console=ttymxc0	To use the default shell job control, such as Ctrl+C to terminate a running process, set this for the kernel.
cma	CMA memory size for GPU/VPU physical memory allocation.	cma=800M@0x960M-0xe00M	Start address is 0x96000000 and end address is 0xDFFFFFFF. The CMA size can be configured to other value, but cannot exceed 1184 MB, because the Cortex-M4 core will also allocate memory from CMA and Cortex-M4 cannot use the memory larger than 0xDFFFFFFF.
androidboot.selinux	Argument to disable selinux check and enable serial input when connecting a host computer to the target board's USB UART port. For details about selinux, see Security-Enhanced Linux in Android .	androidboot.selinux=permissive	Android Pie 9.0 CTS requirement: serial input should be disabled by default. Setting this argument enables console serial input, which will violate the CTS requirement. Setting this argument will also bypass all the selinux rules defined in Android system. It is recommended to set this argument for internal developer.
androidboot.fbTileSupport	It is used to enable 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=CN	It is used to set Wi-Fi country code. Different countries use	androidboot.wificountrycode=CN	-

Table continues on the next page...

Table 11. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
	different Wi-Fi channels.		
androidboot.xen_boot	It is used to configure which environment automotive works at, normal environment or Xen environment.	Normal environment: androidboot.xen_boot=default Xen environment: androidboot.xen_boot=xen	-
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=33554 432	It is 128 MB by default. i.MX 8QuadMax/ 8QuadXPlus automatically configures it to 32 MB to shorten the GPU driver initialization time.

6.2.3 DM-verity configuration

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

To disable DM-verity, perform the following steps:

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

```
reboot bootloader
```

- d. Unlock the device. Execute the following command on the host side:

```
fastboot oem unlock
```

- e. Wait until the unlock process is complete.

2. Disable DM-verity.
 - a. Boot up the device.
 - b. Disable the DM-verity feature. Execute the following command on the host side:

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

7 Over-The-Air (OTA) Update

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 "[Building Android images](#)".

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-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-userdebug
$ ./imx-make.sh bootloader kernel -j4
$ make otapackage -j4
```

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

```
${MY_ANDROID}/out/target/product/mek_8q_car/mek_8q_car-ota-${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 `ota_update.zip` or `incremental_ota_update.zip` (generated on 7.1.2 and 7.1.3) to the HTTP server (for example, `192.168.1.1:/var/www/`).
2. Unzip the packages to get `payload.bin` and `payload_properties.txt`.
3. Cat the content of `payload_properties.txt` like this:
 - `FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtIBeyOigpCCgkoOfHKY=`
 - `FILE_SIZE=379074366`
 - `METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ=`
 - `METADATA_SIZE=46866`
4. Input the following command on the board's console to update:

```
update_engine_client --payload=http://192.168.1.1:10888/payload.bin --update --
headers="FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtIBeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ/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 `ota_update.zip` or `incremental_ota_update.zip` (Generated on 7.1.2 and 7.1.3) to get `payload.bin` and `payload_properties.txt`.
2. Push `payload.bin` to board's `/sdcard` dir: `adb push payload.bin /sdcard/`.
3. Cat the content of `payload_properties.txt` like this:
 - `FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtIBeyOigpCCgkoOfHKY=`
 - `FILE_SIZE=379074366`
 - `METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ=`
 - `METADATA_SIZE=46866`

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

```
update_engine_client --payload=file:///sdcard/payload.bin --update --
headers="FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVt1BeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ/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" character.

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.

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

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

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

NOTE

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

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

You need one HTTP server to hold OTA packages.

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

NOTE

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

3. Push json files to the board.

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/fsl/imx8q/mek_8q/BoardConfig.mk`:

```
TARGET_BOARD_DTS_CONFIG := imx8qm:fsl-imx8qm-mek-car.dtb imx8qm-xen:fsl-imx8qm-mek-domu.dtb
imx8qxp:fsl-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/fsl/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.gogglesource.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 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 Interfaces to control the EVS function

Starting the EVS function with images in automotive-10.0.0_1.1.0_image_8qmek.tar.gz

With images in automotive-10.0.0_1.1.0_image_8qmek.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 the board power-on.

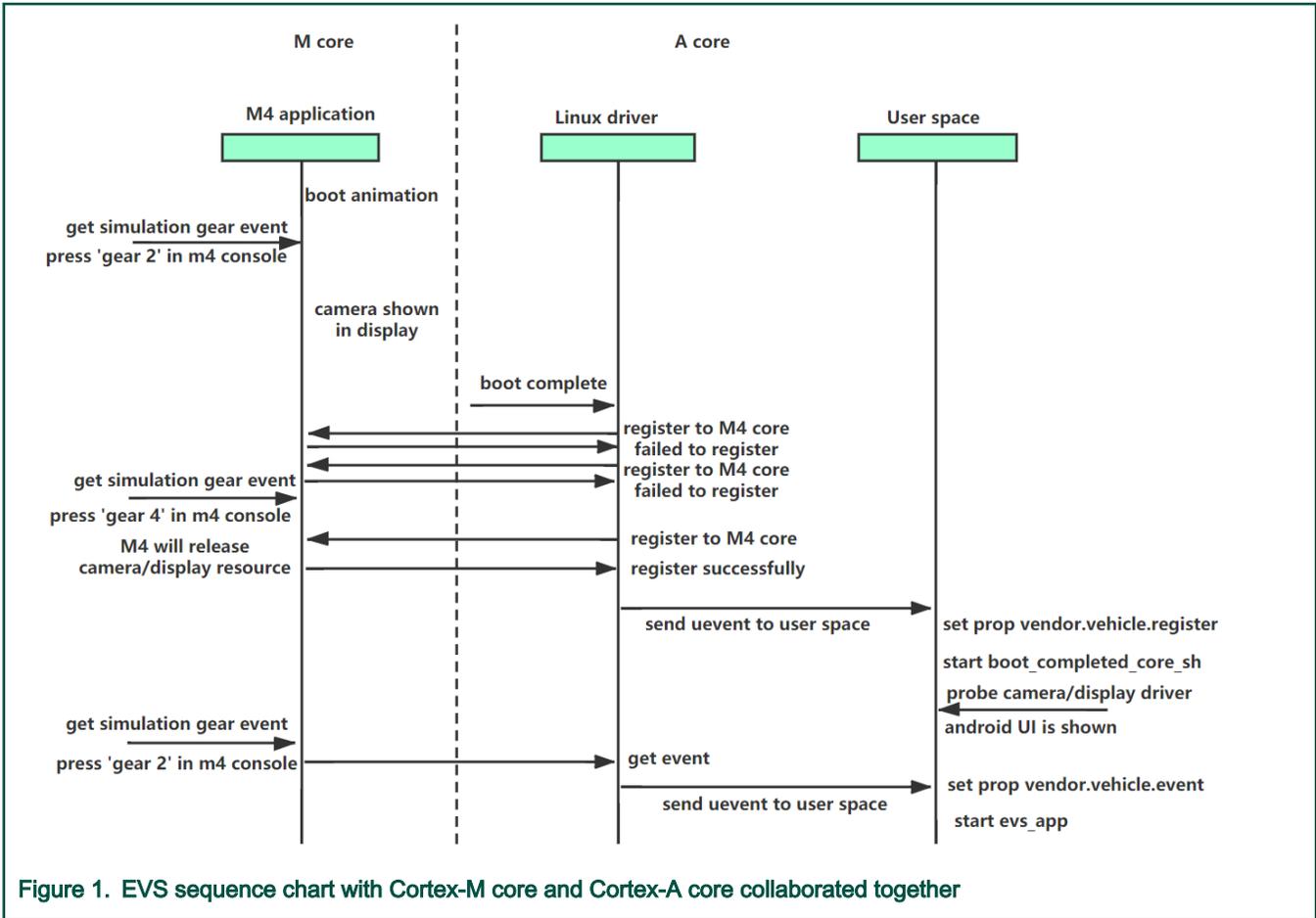


Figure 1. EVS sequence chart with Cortex-M core and Cortex-A core collaborated together

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-M4 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 single-rearview camera:

1. Connect the camera as described in *i.MX Android Quick Start Guide (AQSUG)*.
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 Cortex-A core console, and the other one is Cortex-M4 console.

Cortex-M4 console on i.MX 8QuadMax MEK board: RS232 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 will appear on the screen.

Input `gear 4` when Android Automotive is fully booted. The Android UI will appear on the screen.

- Input `gear 2` on the Cortex-M4 console after Android system boot is complete. The rearview camera will appear on the screen.

Input `gear 4` on the Cortex-M4 console. The Android UI will appear 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 also supports multiple EVS cameras. The relationship between the orientation of cameras and hardware connection is show as follows:

Table 12. 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 veiw
    
```

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

Table 13. 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	Teverse
gear 4	Drive

To start multiple-EVS-camera function:

- 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.
- Input `gear 1` on the Cortex-M4 console. It shows all cameras' views on the display as follows.

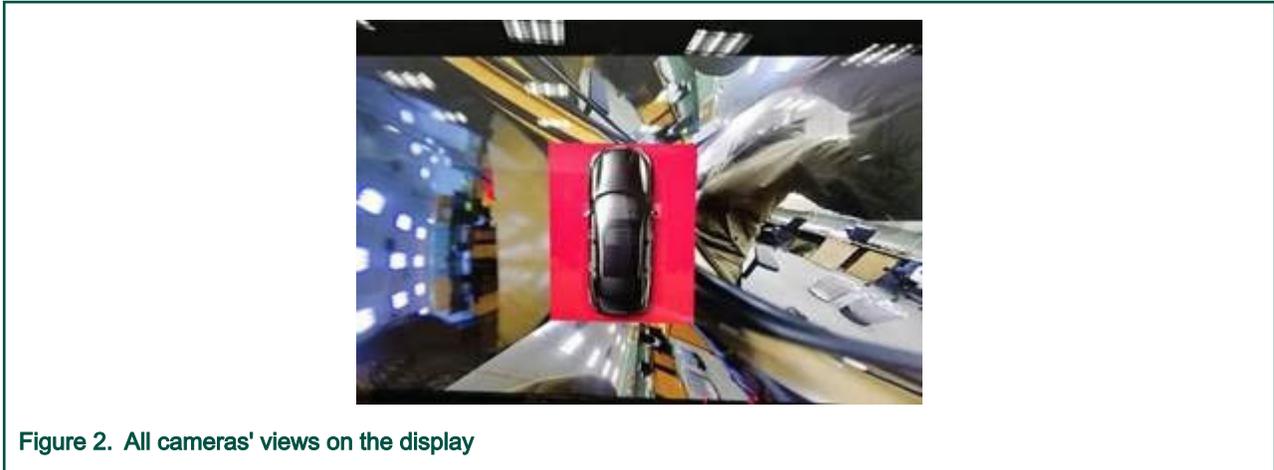


Figure 2. All cameras' views on the display

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 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 boot process to start the rearview camera.

Starting the EVS function with images in automotive-10.0.0_1.1.0_image_8qmek2.tar.gz

With images in `automotive-10.0.0_1.1.0_image_8qmek2.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 single-rearview camera. To start single-rearview camera:

1. Connect the camera as described in *i.MX Android Quick Start Guide (AQSUG)*.
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 can also support multiple EVS cameras.

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

Table 14. 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 veiw
    
```

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

Table 15. 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 multiple-EVS-camera function:

1. Input `su && start evs_app` on the Cortex-A console to start `evs_app`. You can also start 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. It shows all cameras' views on display as follows.

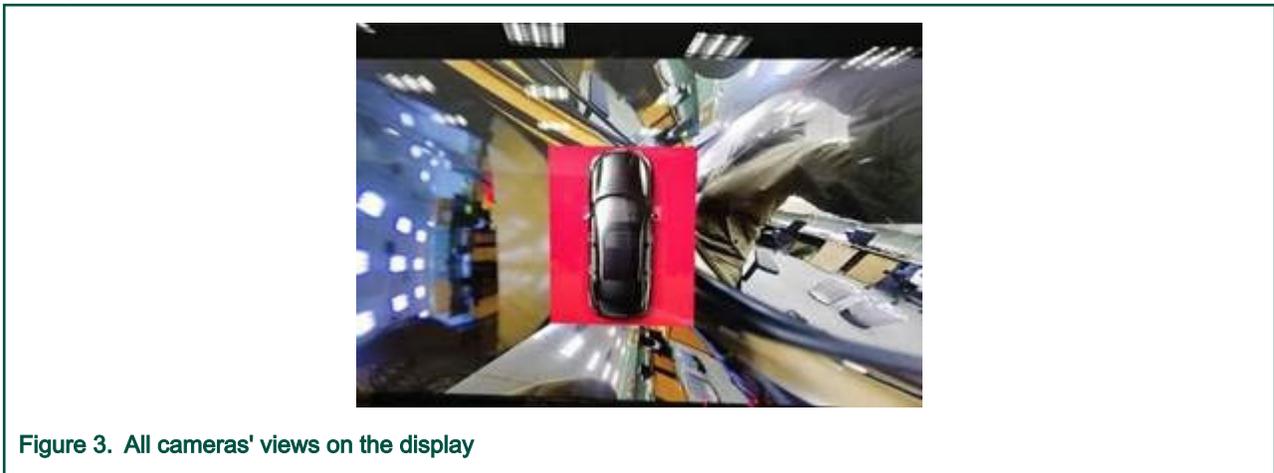


Figure 3. All cameras' views on the display

3. Input `echo 1 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It shows right camera view on the display.
4. Input `echo 2 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It shows 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.

8.1.2 EVS related code

For images in automotive-10.0.0_1.1.0_image_8qmek.tar.gz, the Cortex-M4 core runs with its code on the DDR on i.MX board. It is responsible for the following work:

- Take over control of 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 `#{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-userdebug
./imx-make.sh bootloader -j4
```

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

- EVS hal: `#{MY_ANDROID}/vendor/nxp-opensource/imx/evs`
- EVS service: `#{MY_ANDROID}/vendor/nxp-opensource/imx/evs_service`
- EVS kernel driver: `#{MY_ANDROID}/vendor/nxp-opensource/kernel_imx/drivers/mxc/vehicle`
- EVS application: `#{MY_ANDROID}/packages/services/Car/evs/app/`

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

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

Images in automotive-10.0.0_1.1.0_image_8qmek.tar.gz is built with target lunched with `mek_8q_car-userdebug`. EVS function in this package is realized with both Cortex-A core and the Cortex-M4 core.

The communication commands and related response packet between the Cortex-A core and the Cortex-M4 core are listed as follows.

Table 16. SRTM AUTO Control Category Command Table (Cortex-A -> Cortex-M4)

Category	Version	Type	Command	Data	Function
0x08	0x0100	REQUEST	REGISTER	Data[0-3]: clientIdData[4]: reservedData[5]: partitionData[6-15]: reserved	Registers the RPMSG client. <code>clientId</code> indicates different clients. <code>partition</code> indicate the Android Xen partition. Partition:0xFF: This parameter is invalid.
0x08	0x0100	REQUEST	UNREGISTER	Data[0-3]: clientIdData[4]: reservedData[5]: causeOfData[6-15]: reserved	Unregisters the RPMSG client. Cortex-M4 and remote processor cannot communicate again. "causeOf" parameter can indicate the reason of unregister.causeOf:0x00: AP will power off.
0x08	0x0100	REQUEST	CONTROL	Data[0-3]: clientIdData[4]: reservedData[5-6]: controlCodeData[7-10]: timeoutData[11-15]: controlParamData[15]:index	Sends control command to Cortex-M4 to request Cortex-M4 to do some actions. It needs to complete and give a response to Android in "timeout" ms. Reserved for future.

Table continues on the next page...

Table 16. SRTM AUTO Control Category Command Table (Cortex-A -> Cortex-M4) (continued)

					Example:controlCode: 0x0000: air conditioner temperaturecontrolParam: 4bytes(float): temperatureIndex: left or right.
0x08	0x0100	REQUEST	PWR_REP ORT	Data[0-3]: clientIdData[4]: reservedData[5-6]: androidPwrStateData[7-10]: time_postponeData[11-15]: reserved	Reports Android power stateandroidPwrState:0x0000: BOOT_COMPLETE0x0001: DEEP_SLEEP_ENTRY0x0002: DEEP_SLEEP_EXIT0x0003: SHUTDOWN_POSTPONE0x0004: SHUTDOWN_START0x0005: DISPLAY_OFF0x0006: DISPLAY_ON.
0x08	0x0100	REQUEST	GET_INFO	Data[0-3]: clientIdData[4]: reservedData[5-6]: infoIndexData[7-15]: reserved	Gets information from Cortex-M4 side. Android and Cortex-M4 should have the same information table. The information includes the sensor data, fuel data, battery data, etc. infoIndex:0x0001: vehicle unique ID.
0x08	0x0100	RESPONSE	BOOT_REASON	Data[0-3]: clientIdData[4]: retCodeData[5-15]: reserved	Responds to Cortex-M4's boot reason request (USER_POWER_ON, DOOR_OPEN, DOOR_UNLOCK, REMOTE_START, TIMER).
0x08	0x0100	RESPONSE	PWR_CTRL	Data[0-3]: clientIdData[4]: retCodeData[5-6]: androidPwrStateData[7-15]: reserved	Responds the current power state of Android.
0x08	0x0100	RESPONSE	VSTATE	Data[0-3]: clientIdData[4]: retCodeData[5-6]: unitTypeData[7-15]: reserved	Responds to the control command from Cortex-M4 side. state indicates the current IVI state.

Table 17. SRTM AUTO Control Category Command Table (Cortex-M4 -> Cortex-A)

Category	Version	Type	Command	Data	Function
0x08	0x0100	RESPONSE	REGISTER	Data[0-3]: clientIdData[4]: retCodeData[5-6]: mcuOperateModeData[7-15]: reserved	Response of RPMSG client register. (success, failed), mcuOperateMode indicates Cortex-M4 work statemcuOperateMode:SHARED_RESOURCE_FREE: 0x0000SHARED_RESOURCE_OCCUPIED: 0x0001.
0x08	0x0100	RESPONSE	UNREGISTER	Data[0-3]: clientIdData[4]: retCodeData[5-15]: reserved	Response of RPMSG client unregister.
0x08	0x0100	RESPONSE	CONTROL	Data[0-3]: clientIdData[4]: retCodeData[5-6]:	Response the result of the control request. MCU will do some actions to

Table continues on the next page...

Table 17. SRTM AUTO Control Category Command Table (Cortex-M4 -> Cortex-A) (continued)

				actionStateData[7-15]: reserved	complete Android's request. actionState is not used currently.
0x08	0x0100	RESPONSE	PWR_REPORT	Data[0-3]: clientIdData[4]: retCodeData[5-15]: reserved	Response to Android power state report.
0x08	0x0100	RESPONSE	GET_INFO	Data[0-3]: clientIdData[4]: retCodeData[5-6]: infoIndexData[7-14]: dataData[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 infoIndex. The information includes sensor data, fuel data, battery data. And it is a response packet to Android's request.
0x08	0x0100	REQUEST	BOOT_REASON	Data[0-3]: clientIdData[4]: reservedData[5]: bootReasonData[6-15]: reserved	Notifies Android system that why VMCU boots the Cortex-A core(Android). It is sent after MCU sends normal drive command to android.bootReason:0x00: USER_POWER_ON0x01: DOOR_OPEN0x02: DOOR_UNLOCK0x03: REMOTE_START.
0x08	0x0100	REQUEST	PWR_CTRL	Data[0-3]: clientIdData[4]: reservedData[5-6]: powerStateReqData[7-8]: additionParamData[9-15]: reserved	Requests Android to enter specific power state (ON_DISP_OFF, ON_FULL, SHUTDOWN_PREPARE) powerStateReq:0x0000: ON_DISP_OFF0x0001: ON_FULL0x0002: SHUTDOWN_PREPARE
0x08	0x0100	REQUEST	VSTATE	Data[0-3]: clientIdData[4]: reservedData[5-6]: unitTypeData[7-10]: stateValueData[11-15]: reserved	Requests 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, air condition, etc. stateValue indicates the unit state parameter.

8.1.4 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 in 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 route to different sound cards. When configured, the route is statically decided, unlike the dynamically routed in standard Android image.

In the Android Automotive release, the route is configured as follows: Alarm, notification, and system sounds are played from the audio jack on the CPU board. Other sounds such as music are played from the extended audio board. The following are steps to change the route. For example, music and navigation go through the extended audio board, and others go through the audio jack on the CPU board.

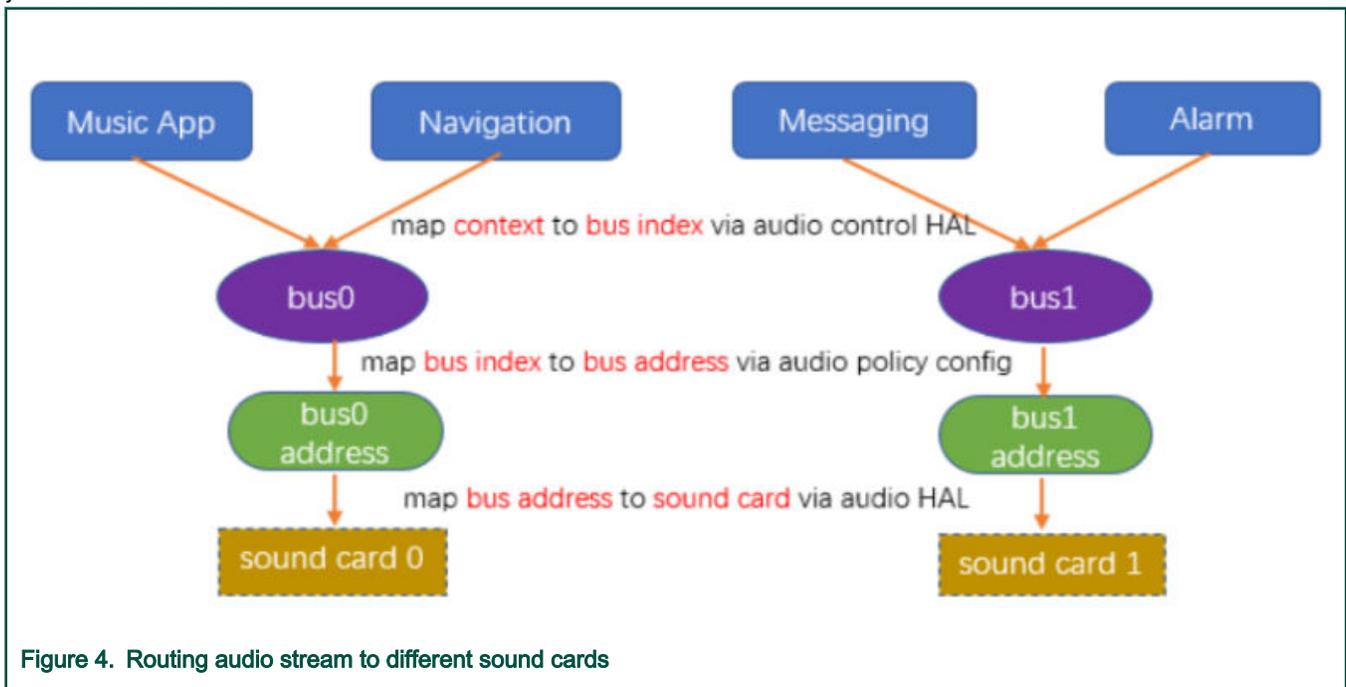


Figure 4. Routing audio stream to different sound cards

1. Map the context to bus index in `$(MY_ANDROID)/vendor/nxp-opensource/imx/audiocontrol/AudioControl.cpp`.

```
static int sContextToBusMap[] = {
    -1,    // INVALID
    0,    // MUSIC_CONTEXT
    0,    // NAVIGATION_CONTEXT
    1,    // VOICE_COMMAND_CONTEXT
    1,    // CALL_RING_CONTEXT
    1,    // CALL_CONTEXT
    1,    // ALARM_CONTEXT
    1,    // NOTIFICATION_CONTEXT
}
```

```
1, // SYSTEM_SOUND_CONTEXT
};
```

- Map the bus index to bus address in `$(MY_ANDROID)/device/fsl/imx8q/mek_8q/audio_policy_configuration_car.xml`. The bus index "0/1" is parsed from the `tagName`.

```
<devicePort tagName="bus0_media_out" role="sink" type="AUDIO_DEVICE_OUT_BUS"
  address="bus0_media_out">
  <gains>
    <gain name="" mode="AUDIO_GAIN_MODE_JOINT"
      minValueMB="-3200" maxValueMB="600" defaultValueMB="0" stepValueMB="100"/>
  </gains>
</devicePort>
<devicePort tagName="bus1_system_sound_out" role="sink" type="AUDIO_DEVICE_OUT_BUS"
  address="bus1_system_sound_out">
  <gains>
    <gain name="" mode="AUDIO_GAIN_MODE_JOINT"
      minValueMB="-3200" maxValueMB="600" defaultValueMB="0" stepValueMB="100"/>
  </gains>
</devicePort>
```

- Bind the bus address to a specific sound card in `$(MY_ANDROID)/vendor/nxp-opensource/imx/alsa`.

In `config_cs42888.h`, `cs42888_card.bus_name = "bus0_media_out"`.

In `wm8960.h`, `wm8960_card.bus_name = "bus1_system_sound_out"`.

- Build the image.

8.3 Display configuration

8.3.1 Configuring the logical display density

The Android UI framework defines a set of standard logical densities to help application developers target application resources. Device implementations must report one of the following logical Android framework densities:

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

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

The default display density value is defined in `$(MY_ANDROID)/device/fsl/imx8q/mek_8q/BoardConfig.mk` as shown below:

```
# Set the density to 213 tvdpi to match CDD.
BOARD_KERNEL_CMDLINE += androidboot.lcd_density=213
```

The display density value can be changed by modifying the related lines mentioned above in `$(MY_ANDROID)/device/fsl/imx8q/mek_8q/BoardConfig.mk` and recompile the code or set 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 controled.

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

The following two commands can be executed on the board console to simulate key input to select the menu on the cluster display:

```
dumpsys activity service android.car.cluster.sample/.SampleClusterServiceImpl injectKey 22
dumpsys activity service android.car.cluster.sample/.SampleClusterServiceImpl injectKey 21
```

8.3.3 Enabling the multiple-display function

The following boards support more than one display.

Table 18. Displays supported by different boards

Board	Number of displays	Display port
i.MX 8QuadMax MEK	4	If physical HDMI is used: HDMI_TX, LVDS0_CH0, LVDS1_CH0, MIPI_DSI1 If physical HDMI is not used: LVDS0_CH0 and LVDS1_CH0, MIPI_DSI0 and MIPI_DSI1
i.MX 8QuadXPlus MEK	2	DSI0/LVDSI0, DSI1/LVDSI1
i.MX 8MQuad EVK	2	HDMI, MIPI-to-HDMI

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

- To evaluate the multiple-display feature with physical HDMI on i.MX 8QuadMax MEK, flash "dtbo-imx8qm-hdmi.img".
- To evaluate the multiple display feature on i.MX 8MQuad EVK, flash "dtbo-imx8mq-dual.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/fsl/` 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=DEL displayName="DELL P2314T"
Display 4692921138614786 (HWC display 2): port=2 pnpId=DEL displayName="DELL S2740L"
Display 18309706364381699 (HWC display 3): port=3 pnpId=PHL displayName="PHL 245C5"
```

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

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

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

To make the modifications take effect, you can modify the source file under the `/${MY_ANDROID}/device/fs1/` 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 Launching applications on different displays

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

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-10.0.0_1.1.0_image_8qmek.tar.gz built with the lunch target "mek_8q_car-userdebug", see the following table to control the HVAC system.

Table 19. HVAC test items for automotive-10.0.0_1.1.0_image_8qmek.tar.gz

	AP-> Cortex-M4	Cortex-M4 -> AP (input on the Cortex-M4 console)	Comments
AC ON	Cortex-M4 Console has the following print when AC is on: Android control: AC_ON, on/off	=>report ac_on 0/1 AC on the panel is close/open.	
Fan direction	Android control: FAN_DIRECTION, 0x2 Typical value: 0x1 (to face) 0x2 (to floor) 0x03 (to face & floor) 0x06 (to floor & defrost)	N	Google APK issue for not support Cortex-M4->AP
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	Cortex-M4 Console has the following print when HVAC is on: Android control: HVAC_POWER_ON, on/off	N	Google APK issue for not support Cortex-M4->AP
AUTO ON	Cortex-M4 Console has the following print when HVAC is auto: Android control: AUTO_ON, on/off	=>report auto_on 0/1 AUTO in panel is close/open.	
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 defrost in panel is close/open. Right one:=>report defrost 0/1 2 defrost in panel is close/open.	
Temperature	Left temp +/-: Android control: AC_TEMP, index=49, temp=16.16 Right temp +/-: Android control: AC_TEMP, index=68, temp=18.18	=>report ac_temp 23.45 49/68 sends 23.45 Centigrade value to the Android side, and the left/right HVAC temp bar changes to 74.	You can calculate the Fahrenheit temp with the following ways: Fahrenheit = 32 + 1.8 * Centigrade Fahrenheit: the num shown in

Table continues on the next page...

Table 19. HVAC test items for automotive-10.0.0_1.1.0_image_8qmek.tar.gz (continued)

			hvacCentigrade: printed on the Cortex-M4 console.
RECIRC	Cortex-M4 Console has the following print when recirc is on: Android control: RECIRC_ON, off/on	=>report recirc_on 0/1 RECIRC in panel is close/open.	

For images in automotive-10.0.0_1.1.0_image_8qmek2.tar.gz built with the lunch target "mek_8q_car2-userdebug", see the following table to control the HVAC system.

Table 20. HVAC test items for automotive-10.0.0_1.1.0_image_8qmek2.tar.gz

	AP-> dummy vehicle driver	Cortex-M4 -> dummy vehicle driver	comments
AC ON	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 AC on the panel is close/open.	
Fan direction	Set fan direction with value 8 Typical value: 0x1 (to face) 0x2 (to floor) 0x03 (to face & floor) 0x06 (to floor & defrost)	N	Google APK issue for not support Cortex-M4-> dummy vehicle driver
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	N	Google APK issue for not support Cortex-M4-> dummy vehicle driver
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 cloase/open	
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 defrost on the panel is close/open. Right one: echo 0/1 > sys/devices/platform/vehicle-dummy/defrost_right defrost on the panel is close/open.	

Table continues on the next page...

Table 20. HVAC test items for automotive-10.0.0_1.1.0_image_8qmek2.tar.gz (continued)

Temperature	<p>Left temp +/-: set temp index 49 with value 1097859072</p> <p>Right temp +/-: set temp index 68 with value 1100422258</p>	<p>echo 1095528903 > sys/devices/platform/vehicle-dummy/temp_left</p> <p>The left HVAC temp bar changes to 55.</p>	<p>You can calculate the Fahrenheit temp as follows:</p> <p>Fahrenheit = 32 + 1.8 * Centigrade</p> <p>Fahrenheit: the num shown in HVAC</p> <p>Centigrade: 1095528903 is the float of Centigrade. You can use the following tool to convert:http://www.23bei.com/tool-23.html#</p>
RECIRC	<p>Recirc on: set recirc on with value 0/1</p>	<p>echo 0/1 > sys/devices/platform/vehicle-dummy/recirc_on</p> <p>RECIRC on the panel is close/open</p>	

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

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

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 Storage configuration

8.6.1 Prolonging eMMC's lifespan

For Android Automotive device, internal storage uses an Embedded MultiMediaCard (eMMC) with thousands of erase/write cycles. If the eMMC fails, the system can become unusable. As vehicles have long lifespans (typically over 10 years). The eMMC must be extremely reliable.

This section provides some methods to help prolong eMMC's lifespan.

8.6.1.1 Enabling adoptable storage

Adoptable storage can make external storage (such as SD cards) to work as internal storage, which can be used to install applications and store application data.

When the external storage media is used, it is formatted and encrypted to only work with a single Android device at one time. Because the media is strongly tied to the Android device that uses it, it can safely store both applications and private data for all users.

To enable adoptable storage, perform the following steps:

1. Enable SDHC node in i.MX 8QuadMax DTS (vendor/nxp-opensource/kernel_imx).

```
diff --git a/arch/arm64/boot/dts/freescale/fsl-imx8qm-mek-car.dts b/arch/arm64/boot/dts/freescale/fsl-imx8qm-mek-car.dts
index fad12072611d..2a261b911acc 100644
--- a/arch/arm64/boot/dts/freescale/fsl-imx8qm-mek-car.dts
+++ b/arch/arm64/boot/dts/freescale/fsl-imx8qm-mek-car.dts
@@ -175,7 +175,7 @@
 };

 &usdhc2 {
-   status = "disabled";
+   status = "okay";
 };

 &amix {
```

2. Enable SDHC node in i.MX 8QuadXPlus DTS (vendor/nxp-opensource/kernel_imx).

```
diff --git a/arch/arm64/boot/dts/freescale/fsl-imx8qxp-mek-car.dts b/arch/arm64/boot/dts/freescale/fsl-imx8qxp-mek-car.dts
index 0ed717953287..c02f63ec2e82 100644
--- a/arch/arm64/boot/dts/freescale/fsl-imx8qxp-mek-car.dts
+++ b/arch/arm64/boot/dts/freescale/fsl-imx8qxp-mek-car.dts
@@ -168,7 +168,7 @@
```

```
};

&usdhc2 {
-   status = "disabled";
+   status = "okay";
};

&amix {
```

3. Add the SDHC node in fstab (device/fsl).

```
diff --git a/imx8q/mek_8q/fstab.freescale.car b/imx8q/mek_8q/fstab.freescale.car
index 9f4442d0..3be100ae 100644
--- a/imx8q/mek_8q/fstab.freescale.car
+++ b/imx8q/mek_8q/fstab.freescale.car
@@ -5,6 +5,7 @@

 /devices/platform/passthrough/5b0d0000.usb/ci_hdrc.0/* auto auto defaults voldmanaged=usb:auto
 /devices/platform/5b0d0000.usb/ci_hdrc.0/* auto auto defaults voldmanaged=usb:auto
+ /devices/platform/5b020000.usdhc/mmc_host* auto auto defaults
voldmanaged=sdcard:auto,encryptable=userdata
 /dev/block/by-name/system          /                ext4
ro,barrier=1                        wait,slotselect
 /dev/block/by-name/userdata       /data            ext4
nosuid,nodev,nodiratime,noatime,nomblk_io_submit,noauto_da_alloc,errors=panic
latemount,wait,formattable,fileencryption=software
 /dev/block/by-name/misc          /misc            emmc
defaults                            defaults
```

4. Use Settings->Storage to configure an SD card as adoptable storage.

8.6.1.2 Limiting third-party application

To protect the internal storage of Android Automotive system, users can configure whether the third-party applications can be installed on the internal storage (applications can write only to the partition on which they were installed). For example, to configure it on the mek_8q board, add the following configuration in the resource overlay (device/fsl/imx8q/mek_8q/overlay_car/frameworks/base/core/res/res/values/config.xml):

```
<bool name="config_allow3rdPartyAppOnInternal">false</bool>
```

After making this configuration, any third-party applications cannot be installed on the internal storage. To install applications, enable the adoptable storage, and use the following command to specify the application installed on external storage:

```
adb install --install-location 2 app.apk
```

8.7 Trusty OS/security configuration

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

The i.MX Trusty OS is based on the AOSP Trusty OS and supports i.MX 8QuadMax MEK and i.MX 8QuadXplus MEK Board. This section describes some basic configurations to make Trusty OS work on

```
# Create a directory for Trusty OS code and enter into this directory
$ repo init -u https://source.codeaurora.org/external/imx/imx-manifest.git -b imx-android-10 -m imx-trusty-automotive-10.0.0_1.1.0.xml
$ repo sync
$ source trusty/vendor/google/aosp/scripts/envsetup.sh
$ make 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
```

EVK/MEK boards. For more configurations about security related features, see the *i.MX Android Security User's Guide* (ASUG). Customers can modify the Trusty OS code to make different configurations and enable different features. First, use the following commands to fetch code to build the target Trusty OS binary.

Then, 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 the 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 10 codebase.

8.7.1 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.

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

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

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

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

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

NOTE

- The RPMB key should start with magic "RPMB" and be followed with 32 bytes hexadecimal key.
- A prebuilt `rpmb_key_test.bin` whose key is fixed 32 bytes hexadecimal `0x00` is provided. It is generated with the following shell commands:
 - `touch rpmb_key.bin`
 - `echo -n "RPMB" > rpmb_key.bin`
 - `echo -n -e '\x00' >> rpmb_key.bin`

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

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

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

```
— fastboot oem set-rpmb-random-key
```

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

NOTE

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

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

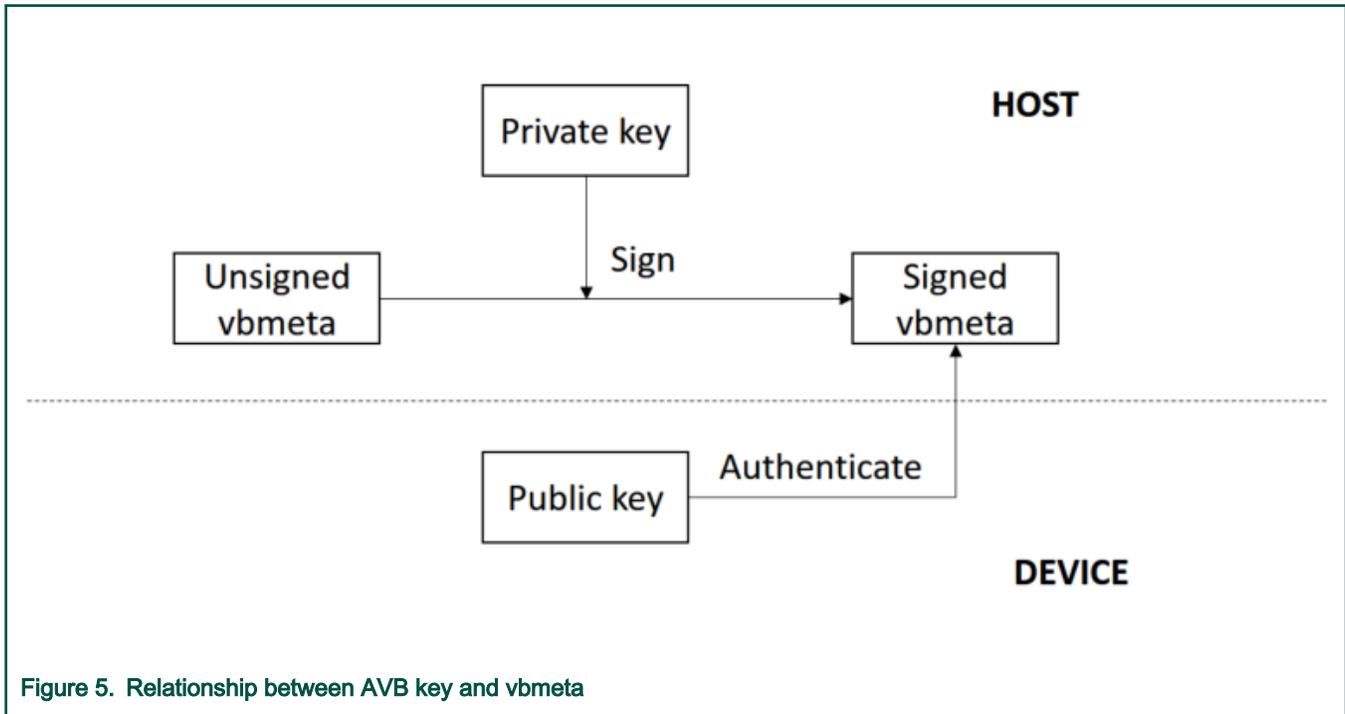


Figure 5. Relationship between AVB key and vbmeta

8.7.2.1 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 use the public key to authenticate the signed vbmeta image.

8.7.2.2 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 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`.

8.7.3 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)*.

8.8 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 "Linux -> Linux 4.19.35_1.1.0 -> SCFW Porting Kit" to download the porting kit. Then, decompress the file with the following commands:

```
tar -zxvf imx-scfw-porting-kit-1.2.7.1.tar.gz
cd packages
chmod a+x imx-scfw-porting-kit-1.2.7.1.bin
./imx-scfw-porting-kit-1.2.7.1.bin
cd imx-scfw-porting-kit-1.2.7.1/src
tar -zxvf scfw_export_mx8qm_b0.tar.gz      # for i.MX 8QuadMax MEK
tar -zxvf scfw_export_mx8qx_b0.tar.gz     # for i.MX 8QuadXPlus MEK
```

The SCFW porting kit contains prebuilt binaries, libraries, and configuration files. For the board configuration file, 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.

8.9 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 21. 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. Sometimes the system wakes up by itself.
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.

8.10 Boot time tuning

8.10.1 Boot time overview

In this document, the boot time is the duration from the time the hardware is started from cold boot to that the Android Automotive Launcher UI is showed on the display screen when the hardware is not in the first time boot from factory. Because the very first successfully boot sets up the accelerating software executing environment, it costs a 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.

8.10.2 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 executed on different CPUs.
- Delayed some non-critical services for SystemUI module of Android after boot is completed.
- Delayed Zygoter32 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.

8.10.3 How to get the shorter boot time

For debug and development purpose, the U-Boot boot delay and Linux kernel dmesg are enable by default. The Linux kernel dmesg is printed by UART. In field measurement, the Linux kernel dmesg costs about 1.15 seconds during the boot process because UART is the slow device. Therefore, before the final product, remove the U-Boot delay and Linux kernel dmesg by the following operations:

- Set `CONFIG_BOOTDELAY=-2` in the U-Boot defconfig file, `imx8qm_mek_androidauto_trusty_defconfig` for i.MX 8QuadMax MEK and `imx8qxp_mek_androidauto_trusty_defconfig` for i.MX 8QuadXPlus MEK in `$(MY_ANDROID)/vendor/nxp-opensource/uboot-imx/configs`.
- Modify the Linux bootargs in build system. See Section 8.1. Appending `loglevel=0` to it will prevent the dmesg to be printed to console during the boot.
- By default, the images are built by `userdebug` build. When it is changed to `user` build, about 0.5 seconds boot time is saved.

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.

8.10.4 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 have 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 `$(MY_ANDROID)/vendor/nxp-opensource/kernel_imx/arch/arm64/configs/android_car_config`:
 - `CONFIG_SQUASHFS=y`
 - `CONFIG_SQUASHFS_LZ4=y`
 - `CONFIG_SQUASHFS_XATTR=y`
 - `CONFIG_SQUASHFS_DECOMP_MULTI=y`
2. Add the following configurations in `$(MY_ANDROID)/device/fsl/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.

8.10.5 How to measure the boot time

Per the definition of the boot time described in [Boot time overview](#), 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 powered down safely. Or the fsck scans the storage during the booting time and it costs 1 to 2 seconds.
- Make sure the action of [How to get the shorter boot time](#) 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 more easier. You can also get the bootanlyaze 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.

In this release, based on the Android bootanlyaze tool, the boot time performance is obtained as listed in the following table.

Table 22. Boot time performance

Platform	mek_8q_car Build	mek_8q_car2 Build
i.MX 8QuadXPlus MEK	21.4s	21.9s
i.MX 8QuadMax MEK	13.9s	14.2s

8.11 Configuration for the load orders of driver modules

8.11.1 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 are build 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`)
- Rfkill
- USB
- Wi-Fi

8.11.2 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.inssmod.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 are listed in `$(MY_ANDROID)/device/fsl/imx8q/mek_8q/setup.core.cfg`. After the core drivers are probed successfully, it triggers low-priority driver modules to load and probe by triggering the service named `boot_completed_main_sh`, which loads drivers listed in `$(MY_ANDROID)/device/fsl/imx8q/mek_8q/setup.main.cfg` in `init.rc`. The "main" drivers are the rest of driver modules, such as `rfkill` for BT, USB, and Wi-Fi.

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`.

8.11.3 How to change driver load orders

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

- Built-in
- Listed in `setup.core.cfg`
- Listed in `setup.main.cfg`

In each `cfg` file, the drivers are loaded one by one. To change the driver load orders, in `setup.core.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/fsl/imx8q/mek_8q/SharedBoardConfig.mk` to copy the specific `.ko` files to the target image.
- Add the driver module name in `setup.core.cfg` or `setup.main.cfg` based on its loading priority.

8.12 Dual-bootloader configuration

8.12.1 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.

8.12.2 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_l_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)*.

8.13 Miscellaneous configuration

8.13.1 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/fsl/imx8q/mek_8q/BoardConfig.mk` file.

8.13.2 Changing the partition size

Partition name and size is defined in `partition-table.img`, which is built from `${MY_ANDROID}/device/fsl/common/partition/device-partitions.bpt`. Follow the patch below in `${MY_ANDROID}/device/fsl` to adjust the partition size (taking the system partition as an example, change its size from 1536 MB to 1792 MB):

```
diff --git a/common/partition/device-partitions-13GB-ab.bpt b/common/partition/device-partitions-13GB-
ab.bpt
index f25697855..5432b5f21 100644
--- a/common/partition/device-partitions-13GB-ab.bpt
+++ b/common/partition/device-partitions-13GB-ab.bpt
@@ -22,7 +22,7 @@
```

```

        {
            "ab": true,
            "label": "system",
-           "size": "1536 MiB",
+           "size": "1792 MiB",
            "guid": "auto",
            "type_guid": "0f2778c4-5cc1-4300-8670-6c88b7e57ed6"
        },
diff --git a/imx8m/BoardConfigCommon.mk b/imx8m/BoardConfigCommon.mk
index 74ca3d0bb..0d0cef236 100644
--- a/imx8m/BoardConfigCommon.mk
+++ b/imx8m/BoardConfigCommon.mk
@@ -119,7 +119,7 @@ BOARD_BOOTIMAGE_PARTITION_SIZE := 50331648
 ifeq ($(IMX_NO_PRODUCT_PARTITION),true)
 BOARD_SYSTEMIMAGE_PARTITION_SIZE := 2952790016
 else
-BOARD_SYSTEMIMAGE_PARTITION_SIZE := 1610612736
+BOARD_SYSTEMIMAGE_PARTITION_SIZE := 1879048192
 endif

 BOARD_VENDORIMAGE_PARTITION_SIZE := 536870912

```

The following table lists the minimum requirement of the partition size:

Table 23. Minimum requirement of the partition size

Partition name	Partition size with GMS built-in	Partition size without GMS built-in
system	1.6 GB	1.2 GB
vendor	280 MB	280 MB
product	350 GB	350 MB

9 Revision History

Table 24. Revision history

Revision number	Date	Substantive changes
O8.1.0_1.1.0_AUTO-EAR	02/2018	Initial release
O8.1.0_1.1.0_AUTO-beta	05/2018	i.MX 8QuadXPlus/8QuadMax Beta release
P9.0.0_1.0.2-AUTO-alpha	11/2018	i.MX 8QuadXPlus/8QuadMax Automotive Alpha release
P9.0.0_1.0.2-AUTO-beta	01/2019	i.MX 8QuadXPlus/8QuadMax Automotive Beta release
P9.0.0_2.1.0-AUTO-ga	04/2019	i.MX 8QuadXPlus/8QuadMax Automotive GA release
P9.0.0_2.1.0-AUTO-ga	08/2019	Updated the location of the SCFW porting kit.
automotive-10.0.0_1.1.0	03/2020	i.MX 8QuadXPlus/8QuadMax MEK (Silicon Revision B0) GA release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

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

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

© NXP B.V. 2018-2020.

All rights reserved.

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

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

Date of release: 24 March 2020

Document identifier: AUG

