AUG
Android™ User's Guide

Rev. P9.0.0_2.3.3-AUTO — 30 March 2020

Contents

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 Pie 9.0 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
```

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-p9.0.0_2.3.5-auto.tar.gz
```

3 Building the Android platform for i.MX

3.1 Getting i.MX Android release source code

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

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

Assume you have i.MX Android proprietary source code package imx-p9.0.0_2.3.5-auto.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-p9.0.0_2.3.5-auto/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).

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

The build configuration command **lunch** can be issued with an argument <Build name>-<Build type> string, such as **lunch mek_8q_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 8QuadXPlus MEK Board with EVS function enabled in the Arm Cortex-M4 CPU core
mek_8q_car2	i.MX 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 Cortex-M4 image.

   Download the GCC tool chain from Arm website, such as "gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2". Extract it to your installation directory, and export the directory as "export ARMGCC_DIR=<install_dir>/gcc-arm-none-eabi-7-2018-q2-update" and add it to /etc/profile. Upgrade the cmake version to or above 3.13.0. For details, see How to customize the rearview camera.

2. Change to the top level build directory.

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

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

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

4. Execute the Android **lunch** command. In this example, the setup is for the production image of i.MX 8QuadXPlus MEK Board/Platform device with user type.

   ```
   $ lunch mek_8q_car-userdebug
   ```

5. Execute the **make** command to generate the image.

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

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

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

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

- root/: root file system (including init, init.rc). Mounted at /.
- system/: Android system binary/libraries. Mounted at /system.
- recovery/: root file system when booting in "recovery" mode. Not used directly.
- dtbo-imx8qxp.img: board's device tree binary. It is used to support the LVDS-to-HDMI display for i.MX 8QuadXPlus 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".
- partition-table.img: GPT partition table image. Used for 16 GB SD card.
- partition-table-7GB.img: GPT partition table image. Used for 8 GB SD card.
- partition-table-28GB.img: GPT partition table image. Used for 32 GB SD card.
- spl-imx8qxp.bin: a composite image including Seco firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK with b0 chip.
- u-boot-imx8qxp-mek-uuu.imx: U-Boot image used by UUU for i.MX 8QuadXPlus MEK with b0 chip. It is not flashed to MMC.
- bootloader-imx8qxp.img: the next loader image after SPL. It includes the Arm trusted firmware, trusty OS, and U-Boot proper for i.MX 8QuadXPlus MEK with b0 chip.
- spl-imx8qxp-c0.bin: a composite image includes Seco firmware, SCU firmware, Cortex-M4 image, and SPL for i.MX 8QuadXPlus MEK with c0 chip.
- u-boot-imx8qxp-mek-c0-uuu.imx: U-Boot image used by UUU for i.MX 8QuadXPlus MEK with c0 chip. It will not be flashed to mmc.
- bootloader-imx8qxp-c0.img: the next loader image after SPL. It includes the Arm trusted firmware, trusty OS, and U-Boot proper for i.MX 8QuadXPlus MEK with c0 chip.
- 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 uImage separately, see Building a kernel image.
- To build boot.img, see Building boot.img.
- To build dtbo.img, see Building dtbo.img.

3.2.1 Configuration examples of building i.MX devices

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

Table 3. i.MX device lunch examples

Build name	Description
i.MX 8QuadXPlus MEK Board with EVS function enabled in the Arm Cortex-M4 CPU core	$ lunch mek_8q_car-userdebug
i.MX 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.

NOTE

To pass CTS, use **user** build mode.

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

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

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

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

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

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

There are two methods for the build of Android image.

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

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

Table 4. Android system image production build method 1

i.MX development tool	Description	Image build command
Evaluation Kit	i.MX 8QuadXPlus MEK with EVS function enabled in the Cortex-M4 CPU core	$ make -j4 PRODUCT-mek_8q_car-userdebug
Evaluation Kit	i.MX 8QuadXPlus MEK without EVS function enabled in the Cortex-M4 CPU core	$ make -j4 PRODUCT-mek_8q_car2-userdebug

Method 2: Set the environment and then use `lunch` command to configure argument. See table below. An example for the i.MX 8QuadXPlus MEK with the EVS function enabled in the Cortex-M4 CPU core is as follows:

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

Table 5. Android system image production build method 2

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 8QuadXPlus MEK with EVS function enabled in the Cortex-M4 CPU core	mek_8q_car-userdebug
Evaluation Kit	i.MX 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:

```
$ 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 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 bootloader -j4
```

3.4 Building a kernel image

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

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

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

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

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

Generate ".config" according to the default configuration file under arch/arm64/configs/android_car_defconfig.

To build the kernel Image for i.MX 8QuadXPlus with EVS function enabled in the Arm Cortex-M4 CPU core, use the following commands:

```
$ make android_car_defconfig
$ make KCFLAGS=-mno-android
```

Generate ".config" according to the default configuration file under arch/arm64/configs/android_car2_defconfig.

To build the kernel image for i.MX 8QuadXPlus without EVS function enabled in the Arm Cortex-M4 CPU core:

```
$ make android_car2_defconfig
$ make KCFLAGS=-mno-android
```

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

3.5 Building boot.img

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

```
# Boot image for i.MX 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.

To generate dtbo.img under the Android environment, use the following commands:

```
# dtbo image for i.MX 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 6. Image packages

Image package	Description
android_p9.0.0_2.3.5-auto_image_8qmek.tar.gz	Prebuilt-image for i.MX 8QuadXPlus MEK board with EVS function enabled in the Arm Cortex-M4 CPU core, which includes NXP extended features.
android_p9.0.0_2.3.5-auto_image_8qmek2.tar.gz	Prebuilt-image and UUU script files for i.MX 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_p9.0.0_2.3.5-auto_image_8qmek.tar.gz image package. Images are almost the same for i.MX 8QuadXPlus MEK with/without EVS function enabled in the Arm Cortex-M4 CPU core.

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

Table 7. Images for i.MX 8QuadXPlus MEK

i.MX 8QuadXPlus MEK images	Description
spl-imx8qxp.bin	The secondary program loader (SPL) for i.MX 8QuadXPlus MEK board with b0 chip.
u-boot-imx8qxp-mek-uuu.imx	The bootloader used by UUU for i.MX 8QuadXPlus MEK board with b0 chip. It is not flashed to MMC
bootloader-imx8qxp.img	The next loader image after SPL for i.MX 8QuadXPlus MEK board with b0 chip.
spl-imx8qxp-c0.bin	The secondary program loader (SPL) for i.MX 8QuadXPlus MEK board with c0 chip.
u-boot-imx8qxp-mek-c0-uuu.imx	The bootloader used by UUU for i.MX 8QuadXPlus MEK board with c0 chip. It is not flashed to MMC.
bootloader-imx8qxp-c0.img	The next loader image after SPL for i.MX 8QuadXPlus MEK board with c0 chip.
boot.img	Boot image for to support LVDS-to-HDMI display.
partition-table.img	GPT table image for 16 GB boot storage.
partition-table-7GB.img	GPT table image for 8 GB boot storage.
partition-table-28GB.img	GPT table image for 32 GB boot storage.
vbmeta-imx8qxp.img	Android Verify Boot metadata Image for i.MX 8QuadXPlus MEK board to support LVDS-to-HDMI display.
system.img	System image.
vendor.img	Vendor image.
dtbo-imx8qxp.img	Device Tree Image for i.MX 8QuadXPlus MEK.
rpmb_key_test.bin	Prebuilt test RPMB key. Iit 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 minium, 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 parition is used to put the unpacked codes/data of the applications, system configuration database, etc. In normal boot mode, the root file system is mounted from the system partition. In recovery mode, the root file system is mounted from the boot partition.

Table 8. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader0	32 KB (i.MX 8QuadXPlus)	4 MB	N/A	spl.bin
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	2560 MB	EXT4. Mount as / system	Android system files under / system/dir

Table continues on the next page...

Table 8. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
8	system_b	Follow system_a	2560 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	presistdata	Follow metadata	1 MB	N/A	Option to operate unlock \unlock
12	vendor_a	Follow persistdata	256 MB	EXT4. Mount at / vendor	vendor.img
13	vendor_b	Follow vendor_a	256 MB	EXT4. Mount at / vendor	vendor.img
14	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.
15	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
16	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
17	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 9. fastboot_imx_flashall script

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

With the help of fasboot_imx_flashall scripts, you do not need to use fastboot to flash Android images one by one manually. These scripts 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: 7 / 14 / 28
                        If it is not set, use partition-table.img (default).
                        If it is set to 7, use partition-table-7GB.img for 8 GB SD card.
                        If it is set to 14, use partition-table-14GB.img for 16 GB SD card.
                        If it is set to 28, use partition-table-28GB.img for 32 GB SD card.
                        Make sure that the corresponding file exists on your platform.
      -m               Flashes the Cortex-M4 image.
      -u uboot_feature  Flashes U-Boot or SPL &bootloader images with "uboot_feature" in their names
                        For Standard Android:
                          If not set, the default U-Boot image is flashed.
                        For Android Automative:
                          If not set, the default SPL&bootloader images are flashed.
      -d dtb_feature    flash dtbo, vbmeta and recovery image file with "dtb_feature" in their names
                          If not set, default dtbo, vbmeta and recovery images are flashed.
      -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 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 imx8qxp -a -e -D /imx_pi9.0/mek_8q_car/
```

Option explanations:

- -f imx8qxp: Flashes images for i.MX 8QuadXPlus MEK Board.
- -a: Only flashes slot a.

- -e: Erases user data after all image files are flashed.
- -D /imx_pi9.0/mek_8q_car/: Images to be flashed are in the directory of /imx_pi9.0/mek_8q_car/.

6 Booting

This chapter describes booting from MMC.

6.1 Booting from eMMC

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

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

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 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 configuartions.

6.2.1 U-Boot environment

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

```
> boota
```

To boot into recovery mode, execute the following command:

```
> boota recovery
```

6.2.2 Kernel command line (bootargs)

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

Table 11. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
console	Where to output kernel log by printk.	console=ttymxc0	-
init	Tells kernel where the init file is located.	init=/init	All use cases. "init" in the Android platform is located in "/" instead of in "/sbin".
androidboot.console	The Android shell console. It should be the same as console=.	androidboot.console=ttymxc0	To use the default shell job control, such as Ctrl+C to terminate a running process, set this for the kernel.
cma	CMA memory size for GPU/VPU physical memory allocation.	cma=800M@0x960M-0xe00M	Start address is 0x96000000 and end address is 0xDFFFFFFFF. The CMA size can be configured to other value, but cannot exceed 1184 MB, because the Cortex-M4 core will also allocate memory from CMA and Cortex-M4 cannot use the memory larger than 0xDFFFFFFFF.
androidboot.selinux	Argument to disable selinux check and enable serial input when connecting a host computer to the target board's USB UART port. For details about selinux, see Security-Enhanced Linux in Android.	androidboot.selinux=permissive	Android 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.

Table continues on the next page...

Table 11. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
androidboot.fbTileSupport	It is used to enable framebuffer super tile output.	androidboot.fbTileSupport=enable	-
firmware_class.path	It is used to set the Wi-Fi firmware path.	firmware_class.path=/vendor/firmware	-
androidboot.wificountrycode=CN	It is used to set Wi-Fi country code. Different countries use different Wi-Fi channels.	androidboot.wificountrycode=CN	-
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=33554432	It is 128 MB by default. i.MX 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 convience for developers, but the device is unprotected.

To disable DM-verity, perform the following steps:

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

```
reboot bootloader
```

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

```
fastboot oem unlock
```

e. Wait until the unlock process is complete.

2. Disable DM-verity.

a. Boot up the device.

b. Disable the DM-verity feature. Execute the following command on the host side:

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

7 Over-The-Air (OTA) Update

This section provides an example for the i.MX 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
$ 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
$ make otapackage -j4
```

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

```
${MY_ANDROID}/out/target/proudct/mek_8q_car/mek_8q_car-ota-${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=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
   - FILE_SIZE=379074366
   - METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
   - 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=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
   FILE_SIZE=379074366
   METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ/de8Dgp9zFXt8Fo+Hxccp465uTOvKNsteWU=
   METADATA_SIZE=46866"
   ```

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

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

1. Unzip 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=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
   - FILE_SIZE=379074366
   - METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
   - METADATA_SIZE=46866
4. Input the following command in board's console to update:

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

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

NOTE

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

7.2.2 Using a customized application to update the Android platform

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

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

Perform the following steps to use this application:

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

   You need one HTTP server to hold OTA packages.

   - For full OTA update, execute the following commands:

```
cp ${MY_ANDROID}/out/target/product/mek_8q/system/build.prop ${server_ota_folder}
cp ${MY_ANDROID}/out/target/product/mek_8q/mek_8q_car-ota-${date}.zip ${server_ota_folder}
cd ${server_ota_folder}
unzip mek_8q_car-ota-${date}.zip
```

   - For incremental OTA update, execute the following commands:

```
cp ${old_build.prop} ${server_ota_folder}/old_build.prop
cp ${MY_ANDROID}/out/target/product/mek_8q/system/build.prop ${server_ota_folder}/
build_diff.prop
mkdir ${server_ota_folder}/diff_ota
cp ${MY_ANDROID}/incremental_ota_update.zip ${server_ota_folder}/diff_ota
cd ${server_ota_folder}/diff_ota
unzip incremental_ota_update.zip
mv payload.bin payload_diff.bin
```

```
mv payload_properties.txt payload_properties_diff.txt
mv payload_diff.bin payload_properties_diff.txt ${server_ota_folder}
cd ${server_ota_folder}
echo -n "base." >> build_diff.prop
grep "ro.build.date.utc" old_build.prop >> build_diff.prop
```

For example, the server_ota_folder content is as follows. Make sure you have at least the following 6 files in `${server_ota_folder}`; otherwise, the OTA application will abort.

```
build@server:/var/www/mek_8q_car_pie_9$ ls
build.prop build_diff.prop payload.bin payload_diff.bin payload_properties.txt
payload_properties_diff.txt
```

NOTE

- server_ota_folder: ${http_root}/mek_8q_car_${ota_folder_suffix}_${version}.
- ${old_build.prop} is the old image's build.prop.
- mek_8q_car-ota-${date}.zip and incremental_ota_update.zip are built from Section 7.1.2 "Building a full update package" and Section 7.1.3 "Building an incremental update package".
- ${ota_folder_suffix} is stored at board's /vendor/etc/ota.conf.
- ${version} can be obtained by the following command on the board's console: `$getprop ro.build.version.release`.
- These file and folder names should align with this example, or modify the OTA application source code correspondingly.

2. Configure the OTA server IP address and HTTP port number.

   The OTA configuration file (/vendor/etc/ota.conf) content is like this:

   ```
   server=192.168.1.100
   port=10888
   ota_folder_suffix=pie
   ```

   Modify it to fit the environment.

3. Open the OTA application and click the **Update** button.

   The reference application is a dialogue box activity, and can be enabled through the **Settings** -> **About tablet** -> **Additional system Update** menu. There are two buttons on the dialogue box:

   - **Upgrade**: Performs full OTA.
   - **Diff Upgrade**: Performs incremental OTA.

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

NOTE

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

NOTE

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

```
TARGET_BOARD_DTS_CONFIG := imx8qm:fsl-imx8qm-mek-car.dtb
TARGET_BOARD_DTS_CONFIG += imx8qm-xen:fsl-imx8qm-mek-domu-car.dtb
TARGET_BOARD_DTS_CONFIG += imx8qxp:fsl-imx8qxp-mek-car.dtb
TARGET_BOARD_DTS_CONFIG := imx8qm:fsl-imx8qm-mek-car2.dtb
TARGET_BOARD_DTS_CONFIG += imx8qxp:fsl-imx8qxp-mek-car2.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 to the i.MX 8QuadMax MEK board. To generate an OTA package for the i.MX 8QuadXPlus MEK board, modify BOARD_PREBUILT_DTBOIMAGE as follows:

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

The OTA package includes 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/bootloader-imx8qm.img
```

To generate the OTA package for i.MX 8QuadXPlus MEK, modify BOARD_OTA_BOOTLOADERIMAGE as follows:

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

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

8 Customized Configuration

8.1 How to change the boot command line in boot.img

When 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.

8.2 How to configure the logical display density

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

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

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

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

To configure the logical display density for framework, you must define the following line in `${MY_ANDROID}/device/fsl/imx8q/mek_8q/init_car.rc`:

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

8.3 How to use an application and add it into the launcher

Only some applications that are contained in car_facet_package_filters can be displayed in the launcher. To start a certain application, use `adb install` and `adb shell am start` to start the related application:

```
  > adb install xxxx.apk
  > adb shell am start xxxx(package of apk, e.g: com.android.cts.verifier)
```

For example, play video with CactusPlayer.apk:

```
  > adb install CactusPlayer.apk
  > adb shell am start -n com.freescale.cactusplayer/com.freescale.cactusplayer.VideoPlayer -d xxx.mp4
```

To display an application in the launcher, add the application package name (e.g., com.freescale.cactusplayer&com.android.cts.verifier) into car_facet_package_filters. ${MY_ANDROID}/packages/services/Car/car_product/overlay/frameworks/base/packages/SystemUI/res/values/arrays_car.xml:

```
diff --git a/car_product/overlay/frameworks/base/packages/SystemUI/res/values/arrays_car.xml b/
car_product/overlay/frameworks/base/packages/SystemUI/res/values/arrays_car.xml
index 94a6d45..8d7c71d 100644
--- a/car_product/overlay/frameworks/base/packages/SystemUI/res/values/arrays_car.xml
+++ b/car_product/overlay/frameworks/base/packages/SystemUI/res/values/arrays_car.xml
@@ -57,6 +57,6 @@
         <item>com.android.car.dialer</item>
         <item>com.android.car.overview</item>
         <item></item>
-
<item>com.android.car.hvac;com.android.settings;com.android.car.settings;com.android.vending;com.googl
e.android.car.bugreport;...;com.google.android.projection.sink</item>
+
<item>com.android.car.hvac;com.android.settings;com.android.car.settings;com.android.vending;com.googl
e.android.car.bugreport;...;com.google.android.projection.sink;com.freescale.cactusplayer;com.android.
cts.verifier</item>
```

```
    </array>
</resources>
```

8.4 Trusty OS build and configuration

8.4.1 How to fetch and build the Trusty OS

i.MX Android Automotive Pie uses the Trusty OS firmware as TEE that supports security features. Users can modify the Trusty OS code to support different configurations and features.

In this release, the i.MX Trusty OS is based on AOSP Trusty OS. NXP adds the i.MX 8QuadXPlus support on it.

To fetch and build the target Trusty OS binary, use the following commands:

```
   $repo init -u https://source.codeaurora.org/external/imx/imx-manifest.git -b imx-android-pie -m imx-trusty-p9.0.0_2.3.5-auto.xml
    $repo sync
    $source trusty/vendor/google/aosp/scripts/envsetup.sh
    $make imx8qxp #for i.MX 8QuadXPlus
    $cp ${TRUSTY_REPO_ROOT}/build-imx8qxp/lk.bin ${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/tee-imx8qx.bin
```

Then build the images and flash the spl-imx8qxp.bin and bootloader-imx8qxp.img files to the target device.

NOTE

- `${TRUSTY_REPO_ROOT}` is the root directory of the Trusty OS repository.
- `${MY_ANDROID}` is the root directory of the Android Automotive Pie repository.

8.4.2 How to initialize the secure storage for the Trusty OS

Security storage is based on RPMB on the eMMC chip. By default, the RPMB key is not initialized by images.

You can use both the specified RPMB key or random RPMB key. The RPMB key cannot be changed once it is set.

- To set a specified RPMB key, perform the following operations:

  Make your board enter fastboot mode. Execute the commands on the host side:

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

  After the board is reboot, 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 with the fixed key of 32 bytes hexadecimal 0x00 is provided. It is generated with the following shell commands:

  ```
  touch rpmb_key.bin
  echo -n "RPMB" > rpmb_key.bin
  echo -n -e
  '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
  x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' >>
  rpmb_key.bin
  ```

  The '\xHH' means 8-bit character whose value is the hexadecimal value 'HH'. You can replace above "00" with the key you want to set.

- To set a random RPMB key, perform the following operations:

Make your board enter fastboot mode. Execute the commands on the host side:

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

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

NOTE

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

8.5 Rearview camera on the i.MX device

Exterior View System (EVS) is supppoted in the i.MX Android auto package. This feature supports fastboot camera that starts camera within 1 second when the board is powered on. Arm Cortex-M4 takes over the control of the camera/display before Android OS boot is complete.

The following figure is the sequence chart of EVS.

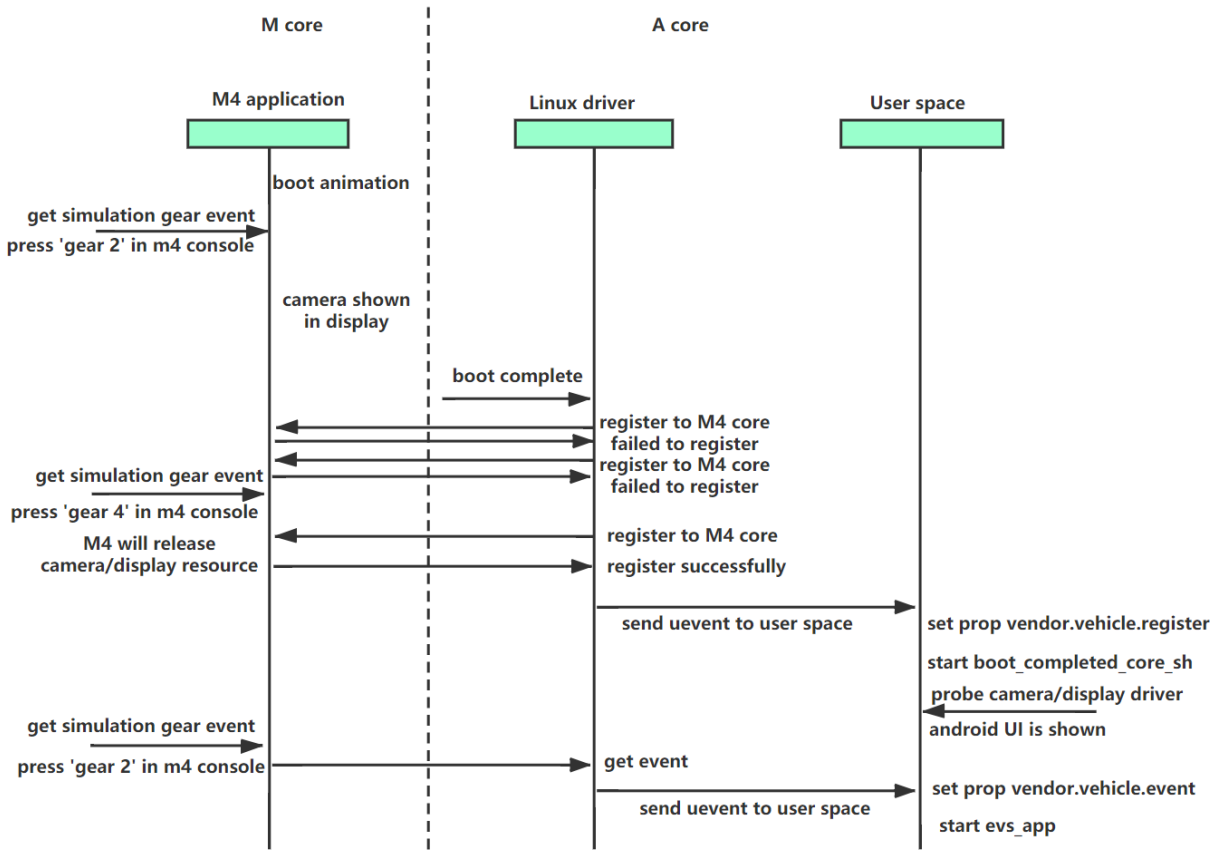

Figure 1. Sequence chart of EVS

8.5.1 How to demo the rearview camera

To demo the rearview camera, perform the following steps:

1. Connect the camera as quick start.
2. Open the Cortex-M4 console.
   - Cortex-M4 console on the i.MX 8QuadXPlus MEK board: The USB-to-UART port has two consoles, one Cortex-A core console and one Cortex-M4 console.

3. Input 'gear 2' on the Cortex-M4 console when the board is powered on, rearview camera appears on the screen.

   Input 'gear 4' when you see the following log printed on the Android console. Android UI appears on the screen.

4. Press 'gear 2' on the Cortex-M4 console after the Android system boot is complete. The rearview camera appears on the screen.

   Press 'gear 4' on the Cortex-M4 console. Android UI appears on the screen.

NOTE

- When you press 'gear 2' on the Cortex-M4 console, the Cortex-M4 core gets the reverse signal.
- When you press 'gear 4' on the Cortex-M4 console, the Cortex-M4 core gets the drive signal.

8.5.2 How to customize the rearview camera

The Cortex-M4 core runs in DDR on the i.MX board. It provides the following functions:

- Takes over control of the camera/display before Android OS is ready.
- Gets the vehicle event and passes this event to the Cortex-A core.

To customize the bootanimation and add the CAN bus event, see the details from the Cortex-M4 source code: `${MY_ANDROID}/vendor/nxp/mcu-sdk-auto`.

To update the Cortex-M4 image, perform the following steps:

1. Prepare the Cortex-M4 image build environment:

   ```
   export ARMGCC_DIR=<path_to_GNUARM_GCC_installation_dir>
   ```

   Make sure the cmake version is equal to or later than 3.13.0. If not, update the cmake version as follows:

   ```
   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
   ```

2. Run the command:

   ```
   make bootloader -j4
   ```

To customize EVS in Android OS, use the following commands:

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

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

These protocol includes the communication commands between Cortex-A core to Cortex-M4 core and related response packet.

Table 12. SRTM AUTO Control Category Command Table (Cortex-A to Cortex-M4)

Category	Version	Type	Command	Data	Function
0x08	0x0100	REQUEST	REGISTER	Data[0-3]: clientIdData[4]: reservedData[5]:	Register RPMSG client. clientId indicates different client. partition indicates the

Table continues on the next page...

Table 12. SRTM AUTO Control Category Command Table (Cortex-A to Cortex-M4) (continued)

Category	Version	Type	Command	Data	Function
				partitionData[6-15]: reserved	Android Xen partition. Partition:0xFF: This parameter is invalid.
0x08	0x0100	REQUEST	UNREGISTER	Data[0-3]: clientIdData[4]: reservedData[5]: causeOfData[6-15]: reserved	Unregister 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	Send 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. Example:controlCode: 0x0000: air conditioner temperaturecontrolParam: 4bytes(float): temperatureIndex: left or right.
0x08	0x0100	REQUEST	PWR_REPORT	Data[0-3]: clientIdData[4]: reservedData[5-6]: androidPwrStateData[7-10]: time_postponeData[11-15]: reserved	Report 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	Get information from Cortex-M4 side. Android platform and Cortex-M4 should have the same information table. The information includes 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	Response 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	Response current power state of Android platform.
0x08	0x0100	RESPONSE	VSTATE	Data[0-3]: clientIdData[4]: retCodeData[5-6]: unitTypeData[7-15]: reserved	Response to the control command from Cortex-M4 side, and “state” indicates the current IVI state.

Table 13. SRTM AUTO Control Category Command Table (Cortex-M4 to 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 operation. 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]: actionStateData[7-15]: reserved	Response the result of the control request. MCU will do some actions to 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]: reserved	Response the GET_INFO request. infoIndex should be the same as the request index. The length of infoData should be specific according to infoIndex. These information includes sensor data, fuel data, and battery data. 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	Notify Android platform that why VMCU boot the Cortex-A core (Android). It will be sent after the MCU send the 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	Request Android platform 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	Request Vehicle state to Android platform (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.

8.6 Boot time tuning

8.6.1 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.

8.6.2 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 from U-Boot and Linux kernel to save its initialization time and image size.
- Removed the unused driver from U-Boot and Linux kernel.
- Make some drivers as the kernel module and load them when Android boot is completed. For example, the connectivity devices and camera driver are initialized after the Android Automotive Launcher UI is showed on the display. This makes the Android Automotive Launcher UI shown earlier.
- Removed the unused device from the Android Framework, such as Ethernet and Sensors.
- Refined the Android Verify Boot procedure.
- Optimized the Android Framework to make service executed on different CPUs.
- Delayed Zygote32 to when UI shown.
- Removed some unused service in Android Framework.

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

8.6.3 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, `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.

8.6.4 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%. Thus, it can shorten the automotive boot time. 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 configurationsg in ${MY_ANDROID}/device/fsl/imx8q/mek_8q/BoardConfig.mk:

```
BOARD_SYSTEMIMAGE_FILE_SYSTEM_TYPE := squashfs
```

This is a balance of CPU processing time and eMMC access time. Customers should try both the conditions of `squashfs` enabled and not enabled to decide whether `squashfs` can be used to get a shorter boot time.

8.6.5 How to measure the boot time

Per the definition of the boot time described in Section 8.6.1, users need to measure the boot time duration from power-on to when the display shows the desktop.

Pay attention to the following:

- Keep the device in lock state by `$fastboot oem lock`.
- Make sure that the device is powered down safely. `$setprop sys.powerctl shutdown` makes the device 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 Section 8.6.3 has been done.

In this release, according to the measurement above, the boot time performance is obtained as in the following table.

Table 14. Boot time performance

Platform	mek_8q_car Build	mek_8q_car2 Build
i.MX 8QuadXPlus MEK	14.3s	15.3s

8.7 How to enable USB 2.0 in U-Boot for i.MX 8QuadXPlus

There are both USB 2.0 and USB 3.0 ports on i.MX 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 8QuadXPlus MEK, make the following changes under ${MY_ANDROID}/vendor/nxp-opensource/uboot-imx:

```
diff --git a/configs/imx8qxp_mek_androidauto_trusty_defconfig b/configs/
imx8qxp_mek_androidauto_trusty_defconfig
index 644e47a..500adb4 100644
--- a/configs/imx8qxp_mek_androidauto_trusty_defconfig
+++ b/configs/imx8qxp_mek_androidauto_trusty_defconfig
@@ -31,14 +31,14 @@ CONFIG_CMD_USB=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=0x18d1
 CONFIG_USB_GADGET_PRODUCT_NUM=0x0d02
```

```
-CONFIG_USB_CDNS3=y
-CONFIG_USB_CDNS3_GADGET=y
 CONFIG_USB_GADGET_DUALSPEED=y

 CONFIG_DM_GPIO=y
diff --git a/include/configs/imx8qxp_mek_android_auto.h b/include/configs/imx8qxp_mek_android_auto.h
index 97f6487..db9438a 100644
--- a/include/configs/imx8qxp_mek_android_auto.h
+++ b/include/configs/imx8qxp_mek_android_auto.h
@@ -50,7 +50,7 @@

 #define CONFIG_SKIP_RESOURCE_CHECING
 #define CONFIG_FSL_FASTBOOT
-#define CONFIG_FASTBOOT_USB_DEV 1
+#define CONFIG_FASTBOOT_USB_DEV 0
 #define CONFIG_ANDROID_RECOVERY
diff --git a/arch/arm/dts/fsl-imx8qxp-mek-auto.dts b/arch/arm/dts/fsl-imx8qxp-mek-auto.dts
index e105f68..2553422 100644
--- a/arch/arm/dts/fsl-imx8qxp-mek-auto.dts
+++ b/arch/arm/dts/fsl-imx8qxp-mek-auto.dts
@@ -67,10 +67,6 @@
         status = "disabled";
      };

-&usbotg1 {
-    -       status = "disabled";
-    -};
-
 &usb2 {
              status = "disabled";
           };
```

To enable USB 2.0 for U-Boot used by UUU, for c language header files, apply the same changes above. For defconfig files, apply the changes above respectively on imx8qxp_mek_android_uuu_defconfig. This defconfig file is specially for U-Boot used by UUU.

8.8 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 relationships between the private key, the public key, and the vbmeta are as follows:

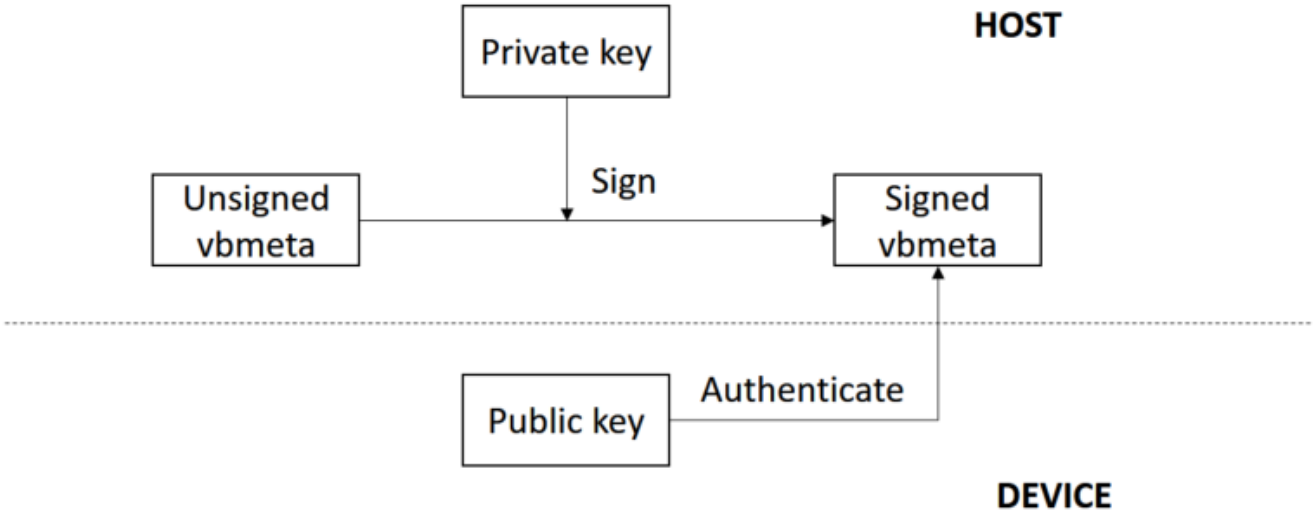

Figure 2. Relationship between AVB key and vbmeta

8.8.1 How to specify the AVB key

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 for i.MX 8QuadXPlus whose Cryptographic Acceleration and Assurance Module (CAAM) can help accelerate the hash calculation.

You can specify the private key for i.MX 8QuadXPlus with the following changes under ${MY_ANDROID}/device/fsl:

```
diff --git a/imx8q/mek_8q/BoardConfig.mk b/imx8q/mek_8q/BoardConfig.mk
index 8e367bb..e1385f9 100644
--- a/imx8q/mek_8q/BoardConfig.mk
+++ b/imx8q/mek_8q/BoardConfig.mk
@@ -207,7 +207,7 @@ BOARD_AVB_ENABLE := true
 ifeq ($(PRODUCT_IMX_CAR),true)
 BOARD_AVB_ALGORITHM := SHA256_RSA4096
 # The testkey_rsa4096.pem is copied from external/avb/test/data/testkey_rsa4096.pem
-BOARD_AVB_KEY_PATH := device/fsl/common/security/testkey_rsa4096.pem
+BOARD_AVB_KEY_PATH := ${your-key-directory}/test_rsa4096_private.pem
```

```
endif
TARGET_USES_MKE2FS := true
```

The Android build system signes the vbmeta image with the private key above and stores one copy of the public key in the signed vbmeta image. During AVB verification, U-Boot validates the public key first and then uses the public key to authenticate the signed vbmeta image.

8.8.2 How to set the vbmeta public key

The public key should be stored in Trusty OS backed RPMB for Android Auto. Perform the following steps to set the public key.

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

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

The public key test_rsa4096_public.bin should be extracted from the 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.

8.9 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).

8.10 How to prolong 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.

8.10.1 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 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 {
```

2. 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
```

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

8.10.2 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
```

8.11 Cluster display in i.MX device

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

To demostrate cluster display, connect 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
```

8.12 How to change SCFW

SCFW is a binary stored in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware`, built into bootloader. To change SCFW, you need SCFW porting kit and specified board configuration file. SCFW porting kit contains prebuilt binaries and libraries.

Specified board configuration file is stored in SCFW porting kit, for example (i.MX 8QuadXPlus): `imx-scfw-porting-kit/src/scfw_export_mx8qx_b0/platform/board/mx8qx_mek/board.c`.

There is another board configuration file stored in `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/board-imx8qxp.c`.

You can copy `board.c` from `vendor/nxp/fsl-propeirtary` 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: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads/6-2017-q2-update.
2. Unzip the GCC tool to /opt/scfw_gcc.
3. Export `TOOLS="/opt/scfw-gcc"`.
4. Download SCFW porting kit to `${MY_ANDROID}` as `imx-scfw-porting-kit.bin`. You can download the corresponding version SCFW from here: L4.14.98_2.0.0_SCFWKIT-1.2.
5. Unzip the porting kit and SCFW for i.MX 8QuadXPlus.

```
./imx-scfw-porting-kit.bin
cd imx-scfw-porting-kit/src
tar xf scfw_export_mx8qx_b0.tar.gz
```

6. Copy THE board configuration file from `${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/board-imx8qxp.c` to porting kit.

```
cp ${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car/board-imx8qxp.c
scfw_export_mx8qx_b0/platform/board/mx8qx_mek/board.c
```

7. Build SCFW.

```
cd ${MY_ANDROID}/imx-scfw-porting-kit/src/scfw_export_mx8qx_b0
make clean
make qx R=B0 B=mek
```

8. 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
```

9. Build the bootloader.

```
cd ${MY_ANDROID}
make bootloader
```

8.13 How to configure car audio route

In Android car image, different streams route to different sound cards. Once configured, the route is static decided, unlike the dynamic route in ANDROID standard image.

In the car image release version, 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. The example shows that music and navigation go through the extended audio board, and others go through the audio jack on the CPU board.

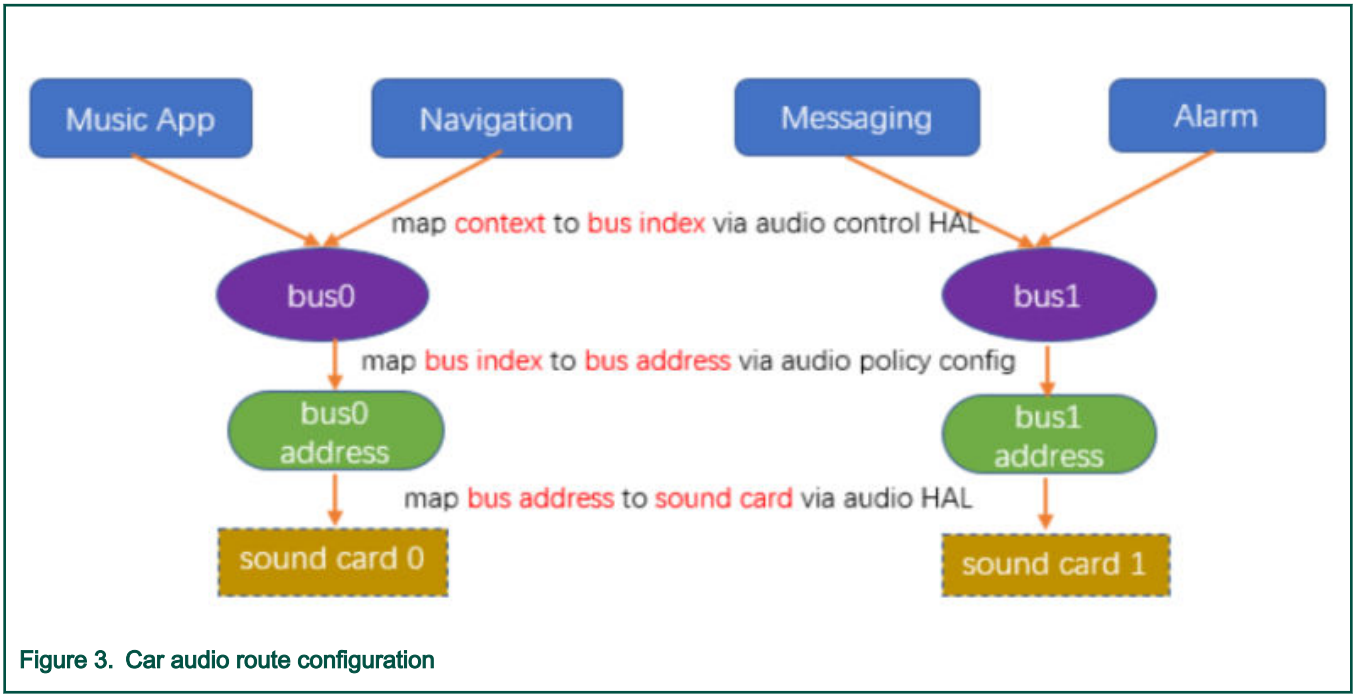

Figure 3. Car audio route configuration

1. Map the context to bus index in `${MY_ANDROID}/hardware/interfaces/automotive/audiocontrol/1.0/default/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
};
```

2. 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>
```

3. 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"`.
4. Build the image.

9 EVS/HVAC Function

9.1 EVS/HVAC functions for car image

9.1.1 HVAC

The following table lists the HVAC test items.

Table 15. HVAC test items

	AP-> Cortex-M4	Cortex-M4 -> AP (input on Cortex-M4 console)	comment
AC ON	Cortex-M4 console has the following print when ac on: Android control: AC_ON, ON/OFF	`=>report ac_on 0/2` AC in panel will be closed/open.	-

Table continues on the next page...

Table 15. HVAC test items (continued)

	AP-> Cortex-M4	Cortex-M4 -> AP (input on Cortex-M4 console)	comment
Fan direction	Android control: FAN_DIRECTION, 0x2 Typical value: 0x2 (to face) 0x4 (to floor) 0x06 (to face & floor) 0x0a (to floor & defrost) Pi 9.0: typical value: 0x2 (to face) 0x4 (to floor) 0x06 (to face & floor) 0x0c (to floor & defrost)	N	Fan direction Control from Cotex-M4 core is not supported by the default HVAC Android application.
Fan speed	Android control: FAN_SPEED, 0x6 Typical value: 0x00(off)/ 0x02/0x04/0x06/0x08/0x0a/ 0x0c(MAX)	`=>report fan_speed 2/4/6/8/10/12` 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	HVAC power-on control from Cotex-M4 core is not supported by the default HVAC Android application.
AUTO ON	Cortex-M4 console has the following print when HVAC is auto: Android control: AUTO_ON, ON/OFF	`=>report auto_on 0/2` AUTO in panel will be closed/open	-

Defrost	Left one:
	Android control: DEFROST, index=1, on/off
	Right one:
	Android control: DEFROST, index=2, on/off
		Left one:
		`=>report defrost 0/2 1`
		defrost in panel will be closed/open.
		Right one:
		`=>report defrost 0/2 2`
		defrost in panel will be closed/open
			-

Temperature	Left temp +-: Or 8.1: Android control: AC_TEMP, index=1, temp=16.16	`=>report ac_temp 23.45 1/4`	You can calculate the Fahrenheit value as follows: Fahrenheit = 32 + 1.8 * Centigrade

Table continues on the next page...

Table 15. HVAC test items (continued)

	AP-> Cortex-M4	Cortex-M4 -> AP (input on Cortex-M4 console)	comment
	Pi 9.0: Android control: AC_TEMP, index=49, temp=16.16 Right temp +-: Or 8.1: Android control: AC_TEMP, index=4, temp=21.21 Pi9.0: Android control: AC_TEMP, index=68, temp=18.18	Sends 23.45 Centigrade value to Android side. The left/right HVAC temperature bar will change to 74.	**Fahrenheit: the number shown in HVAC** **Centigrade: 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 0/2` RECIRC in panel will be closed/open	-

9.1.2 Multi-camera EVS test procedure

This function is supported on the i.MX 8QuadXPlus MEK board with silicon revision C0 chip. The i.MX 8QuadXPlus MEK board with silicon revision B0 chip does not support it.

- The relationship between the orientation of cameras and hardware connection is listed in the following table.

Table 16. Relationship between the orientation of cameras and hardware connection

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

- The application has the following logic when handling the vehicle information.

```
If (gear state == reverse)
    Show reverse camera
Else if (turn signal == right)
    Show right camera
Else if(turn signal == left)
    Show left camera
Else if(gear state == park)
    Show overall camera.
Else
    No camera info shown
```

- The meaning of commands input on the Cortex-M4 console is listed in the following table.

Table 17. Meaning of commands input on the Cortex-M4 console

Command	Meaning
turn 0	Not turn
turn 1	Turn right
turn 2	Turn left
gear 1	Park
gear 2	Reverse
gear 4	Drive

The test steps are as follows. You may design your own test case.

1. Enter `su && start evs_app` on the Cortex-A core console to start evs_app.

   You can also start the rearview camera on the Cortex-M4 side (`gear 2`). The display should be the reverse camera shown.

2. Enter `gear 1` on the Cortex-M4 console. It shows the overall camera on the display as follows.

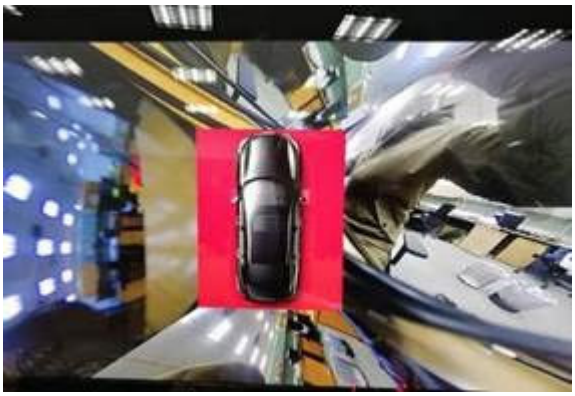

Figure 4. Overall camera

3. Enter `turn 1` on the Cortex-M4 console. It has the left camera shown on the display.
4. Enter `turn 2` on the Cortex-M4 console. It has the right camera shown on the display.
5. Enter `turn 0` on the Cortex-M4 console. The overall camera is shown.
6. Stop evs through `stop evs_app` on the Cortex-M4 core console.

NOTE

You can enter `gear 2` anytime in boot process to test the rear view camera.

9.2 EVS/HVAC functions for car2 image

9.2.1 HVAC

The following table lists the HVAC test items.

Table 18. HVAC test items

	AP-> dummy vehicle driver	Cortex-M4 -> dummy vehicle driver	Comment
AC ON	AP Console has the following print when AC is OFF/ON: Set fan AC on with value 0/2	`echo 0/2 > sys/devices/platform/vehicle-dummy/ac_on` AC in panel will be closed/open.	-
Fan direction	Set fan direction with value 8. Or 8.1: Typical value: 0x2 (to face) 0x4 (to floor) 0x06 (to face & floor) 0x0a (to floor & defrost) Pi9.0: Typical value: 0x2 (to face) 0x4 (to floor) 0x06 (to face & floor) 0x0c (to floor & defrost)	N	Fan direction Control from Cotex-M4 core is not supported by the default HVAC Android application.
Fan speed	Set fan speed with value 8. Typical value: 0x00(off)/0x02/0x04/0x06/0x08/0x0a/0x0c(MAX)	`echo 2/4/6/8/10/12 > 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	HVAC power-on control from Cotex-M4 core is not supported by the default HVAC Android application.
AUTO ON	Set Auto ON with value 0/2 Set Auto OFF/ON	`echo 0/2 > sys/devices/platform/vehicle-dummy/auto_on` AUTO in panel will be closed/open.	-
Defrost	Left one: Set defroster index 1 with value 0/2 Right one: Set defroster index 2 with value 0/2	Left one: `echo 0/2 > sys/devices/platform/vehicle-dummy/defrost_right` defrost in panel will be closed/open.	-

Table continues on the next page...

Table 18. HVAC test items (continued)

	AP-> dummy vehicle driver	Cortex-M4 -> dummy vehicle driver	Comment
		Right one: `echo 0/2 > sys/devices/platform/vehicle-dummy/defrost_right` defrost in panel will be closed/open.	
Temperature	left temp +-: or 8.1: Set temp index 1 with value 1097859072 Pi 9.0: Set temp index 49 with value 1097859072 Right temp +-: Or 8.1: Set temp index 4 with value 1100422258 Pi 9.0: Set temp index 68 with value 1100422258	`echo 1095528903 > sys/devices/platform/vehicle-dummy/temp_left` The left HVAC temperature bar will change to 55.	You can calculate the Fahrenheit temperature value as follows: Fahrenheit = 32 + 1.8 * Centigrade Fahrenheit: the number shown in HVAC Centigrade: 1095528903 is the float of Centigrade. You can use the following tool to convert: http://www.23bei.com/tool-23.html#
RECIRC	recirc on: Set recirc ON with value 0/2	`echo 0/2 > sys/devices/platform/vehicle-dummy/recirc_on` RECIRC in panel will be closed/open.	-

9.2.2 Multi-camera EVS test procedure

This function is supported on the i.MX 8QuadXPlus MEK board with silicon revision C0 chip. The i.MX 8QuadXPlus MEK board with silicon revision B0 chip does not support it.

- The relationship between the orientation of cameras and hardware connection is listed in the following table.

Table 19. Relationship between the orientation of cameras and hardware connection

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

- The application has the following logic when handling the vehicle information.

```
If (gear state == reverse)
    Show reverse camera
Else if (turn signal == right)
    Show right camera
Else if(turn signal == left)
    Show left camera
Else if(gear state == park)
    Show overall camera.
Else
    No camera info shown
```

- The meaning of commands input on the Cortex-A console is listed in the following table.

Table 20. Meaning of commands input on the Cortex-A 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

The test steps are as follows. You may design your own test case.

1. Enter `su && start evs_app` on the Cortex-A core console to start evs_app.

   You can also start the rearview camera (`echo 2 > sys/devices/platform/vehicle-dummy/gear`) on the Cortex-A side. The display should be the reverse camera shown.

2. Enter `'echo 1 > sys/devices/platform/vehicle-dummy/gear` on the Cortex-A console. It shows the overall camera on the display as follows.

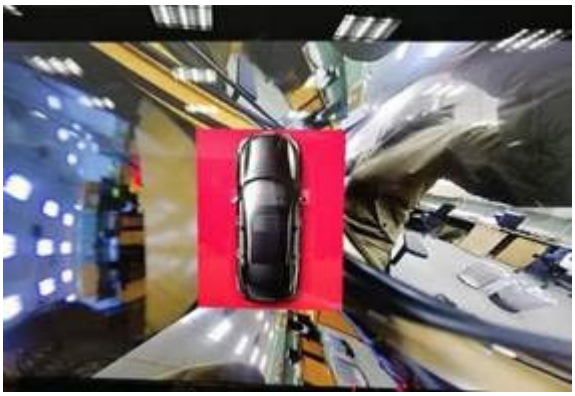

Figure 5. Overall camera

3. Enter `echo 1 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It has the left camera shown on the display.
4. Enter `echo 2 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. It has the right camera shown on the display.
5. Enter `echo 0 > sys/devices/platform/vehicle-dummy/turn` on the Cortex-A console. The overall camera is shown.
6. Stop evs through `stop evs_app` on the Cortex-A core console.

NOTE

You only can test rearview camera (not support multi-camera) at the kernel stage with the following commands on the i.MX 8QuadXPlus board:

```
echo 2 > sys/devices/platform/vehicle-dummy/gear
```

10 Revision History

Table 21. 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
P9.0.0_2.3.3-AUTO	02/2020	i.MX 8QuadXPlus MEK GA release
P9.0.0_2.3.3-AUTO	03/2020	Added the multi-camera EVS test procedure.

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.

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

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

Date of release: 30 March 2020

Document identifier: AUG