

Android™ User's Guide

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

For any problems on the building process related to the jack server, see the Android website source.android.com/source/jack.html.

1	Overview.....	1
2	Preparation.....	1
3	Building the Android platform for i.MX.....	2
4	Running the Android Platform with a Prebuilt Image.....	7
5	Programming Images.....	8
6	Bootting.....	11
7	Over-The-Air (OTA) Update.....	14
8	Customized Configuration.....	17
9	Revision History.....	23



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 liblz2-2 liblz2-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_1.0.2-auto-alpha.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_1.0.2-auto-alpha.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_1.0.2-auto-alpha/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-userdebug	i.MX 8QuadXPlus/8QuadMax MEK Board

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. Change to the top level build directory.

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

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

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

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

```
$ lunch mek_8q_car-userdebug
```

4. 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.

Building the Android platform for i.MX

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.
- 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.
- dtbo-imx8qm-xen.img: board's device tree binary. It is used to support the LVDS-to-HDMI display on Xen.
- dtbo-imx8qxp.img: board's device tree binary. It is used to support the LVDS-to-HDMI display.
- vbmeta-imx8qm.img: Android Verify boot metadata image for boot-imx8qm.img. It is used to support the LVDS-to-HDMI display.
- vbmeta-imx8qm-xen.img: Android Verify boot metadata image for dtbo-imx8qm-xen.img. It is used to support the LVDS-to-HDMI display on Xen.
- vbmeta-imx8qxp.img: Android Verify boot metadata image for boot-imx8qxp.img. It is used to support the LVDS-to-HDMI display.
- 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.
- u-boot-imx8qm.imx: U-Boot image with no padding for i.MX 8QuadMax MEK.
- u-boot-imx8qm-xen.imx: U-Boot image with no padding for i.MX 8QuadMax MEK on Xen.
- u-boot-imx8qxp.imx: U-Boot image with no padding for i.MX 8QuadXPlus MEK.
- uuu-u-boot-imx8qm.imx: U-Boot image used by UUU for i.MX 8QuadMax MEK. It is not flashed to MMC.
- uuu-u-boot-imx8qxp.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.

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/8QuadMax MEK Board	\$ lunch mek_8q_car-userdebug

3.2.2 User build mode

A production release Android system image is created by using the **userdebug** Build Type. For configuration options, see Table "Build types" in Section [Building Android images](#).

The notable differences between the **user** and **eng** build types are as follows:

- Limited Android System image access for security reasons.
- Lack of debugging tools.
- Installation modules tagged with user.
- APKs and tools according to product definition files, which are found in PRODUCT_PACKAGES in the sources folder `${MY_ANDROID}/device/fs/imx8/imx8.mk`. To add customized packages, add the package `MODULE_NAME` or `PACKAGE_NAME` to this list.
- The properties are set as: `ro.secure=1` and `ro.debuggable=0`.
- adb is disabled 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 PRODUCT-XXX userdebug 2>&1 | tee build-log.txt #XXX depends on different
boards, 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/8QuadMax MEK	\$ make PRODUCT-mek_8q_car-userdebug

Method 2: Set the environment and then use lunch command to configure argument. See table below. An example for the i.MX 8QuadXPlus/8QuadMax MEK board 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/8QuadMax MEK	mek_8q_car-userdebug

To create Android platform 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 8QuadMax/8QuadXPlus MEK board
$ 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 8QuadMax/8QuadXPlus, use the following commands:

```
$ make android_car_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 8QuadMax/8QuadXPlus MEK board
$ 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 8QuadMax/8QuadXPlus MEK board
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch mek_8q_car-userdebug
$ make dtboimage -j4
```

4 Running the Android Platform with a Prebuilt Image

To test the Android platform before building any code, use the prebuilt images from the following packages and go to "Programming Images" and "Bootting".

Table 6. Image packages

Image package	Description
android_p9.0.0_1.0.2-auto-alpha_image_8qmek.tar.gz	Prebuilt-image for i.MX 8QuadXPlus/8QuadMax MEK board, which includes NXP extended features.

The following tables list the detailed contents of android_p9.0.0_1.0.2-auto-alpha_image_8qmek.tar.gz image package.

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/8QuadMax MEK image	Description
/u-boot-imx8qm.imx	Bootloader (with padding) for i.MX 8QuadMax MEK board
/u-boot-imx8qm-xen.imx	The bootloader (with padding) for i.MX 8QuadMax MEK board on Xen
/u-boot-imx8qxp.imx	Bootloader (with padding) for i.MX 8QuadXPlus MEK board
/uuu-u-boot-imx8qm.imx	Bootloader used by UUU for i.MX 8QuadMax MEK board. It is not flashed to MMC.
/uuu-u-boot-imx8qxp.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-7GB.img	GPT table image for 8 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-xen.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support LVDS-to-HDMI display on Xen
/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
/dtbo-imx8qm.img	Device tree image for i.MX 8QuadMax
/dtbo-imx8qm-xen.img	Device tree image for i.MX 8QuadMax on Xen
/dtbo-imx8qxp.img	Device tree image for i.MX 8QuadXPlus

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 [Bootimg](#).

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:

- U-Boot image: u-boot.imx
- GPT table image: partition-table.img
- Android dtbo image: dtbo.img
- Android boot image: boot.img
- Android system image: system.img
- Android 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 8. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader	0 KB (i.MX 8QuadMax) or 32 KB (i.MX 8QuadXPlus)	4 MB	N/A	bootloader
1	dtbo_a	8 MB	4 MB	N/A	dtbo.img
2	dtbo_b	Follow dtbo_a	4 MB	N/A	dtbo.img

Table continues on the next page...

Table 8. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
3	boot_a	Follow dtbo_b	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
4	boot_b	Follow boot_a	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
5	system_a	Follow boot_b	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
6	system_b	Follow system_a	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
7	misc	Follow system_b	4 MB	N/A	For recovery store bootloader message, reserve
8	metadata	Follow datafootor	2 MB	N/A	For system slide show
9	persistdata	Follow metadata	1 MB	N/A	Option to operate unlock \unlock
10	vendor_a	Follow persistdata	112 MB	EXT4. Mount at / vendor	vendor.img
11	vendor_b	Follow vendor_a	112 MB	EXT4. Mount at / vendor	vendor.img
12	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.
13	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
14	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
15	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)*, or use format tools in the prebuilt directory.

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.

Programming Images

A new fastboot_imx_flashall script is supported to use fastboot to flash the Android system image into the board. It is more flexible. Only require board can enter fastboot mode and the device is unlocked. 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 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.

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.
-d dev	Flash dtbo, vbmeta, and recovery image file with dev. If it is not set, use default dtbo, vbmeta, and recovery image.
-e	Erases user data after all image files are flashed.
-l	Locks the device after all image files are flashed.
-D directory	Directory of images. If this script is execute in the directory of the images, it does not need to use this option.
-s ser_num	Serial number of the board. If only one board connected to computer, it does not need to use this option

NOTE

- -f option is mandatory. 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_pi9.0/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_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/8QuadMax MEK board

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

Table 10. Boot switch settings for i.MX 8QuadMax

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

Table 11. 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. To use the default environment in boot.img, use the following command:

```
U-Boot > setenv bootargs
```

To clear the bootargs environment, use the following command:

```
U-Boot > setenv bootcmd boota mmc0
U-Boot > setenv bootargs console=ttyLP0,115200 earlycon=lpuart32,0x5a060000,115200
androidboot.console=ttyLP0 androidboot.xen_boot=default init=/init consoleblank=0
androidboot.hardware=freescale androidboot.fbTileSupport=enable cma=800M@0x960M-0xe00M
androidboot.primary_display=imx-drm firmware_class.path=/vendor/firmware [Optional]
U-Boot > saveenv [Save the environments]
```

NOTE

bootargs environment is an optional setting for boota. The boot.img includes a default bootargs, which is used if there is no definition about the bootargs environment.

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 `zImage` 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 from `mmcX`, do the following:

```
> boota mmcX
```

To read the boot partition (the partition store `boot.img`, in this instance, `mmcblk0p1`), the `X` is the eMMC bus number, which is the hardware eMMC bus number, in SABRE-SD boards. eMMC is `mmc2` or you can add the partition ID after `mmcX`.

Add partition ID after `mmcX`.

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

If you have read the `boot.img` into memory, use this command to boot:

```
> boota 0XXXXXXXXX
```

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 12. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
console	Where to output kernel log by <code>printk</code> .	<code>console=ttymx0</code>	i.MX 8QuadMax MEK uses <code>console=ttyLP0</code>
init	Tells kernel where the init file is located.	<code>init=/init</code>	All use cases. "init" in the Android platform is located in "/" instead of in "/sbin".

Table continues on the next page...

Table 12. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
androidboot.console	The Android shell console. It should be the same as console=.	androidboot.console=ttymx0	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 on i.MX 8MQuad EVK.	androidboot.fbTileSupport=enable	-
firmware_class.path	It is used to set the Wi-Fi firmware path.	firmware_class.path=/vendor/firmware	-
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	-

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.

Over-The-Air (OTA) Update

- b. Choose **Settings -> Developer Options -> OEM Unlocking** to enable OEM unlocking.
- c. Enter Fastboot mode on the device. Execute the following command on the target side:

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

```
fastboot oem unlock
```
- e. Wait until the unlock process is complete.
2. Disable DM-verity.
 - a. Boot up the device.
 - b. Disable the DM-verity feature. Execute the following command on the host side:

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

7 Over-The-Air (OTA) Update

7.1 Building OTA update packages

7.1.1 Building target files

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

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

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

- 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/`).
- Unzip the packages to get `payload.bin` and `payload_properties.txt`.
- Cat the content of `payload_properties.txt` like this:
 - `FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVt1BeyOigpCCgkoOfHKY=`
 - `FILE_SIZE=379074366`
 - `METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ=`
 - `METADATA_SIZE=46866`
- 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=0fSBbXonyTjaAzMpwTBgM9AVt1BeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ/de8Dgp9zFXt8Fo
+Hxccp465uTOvKNsteWU=
METADATA_SIZE=46866"
```

NOTE

Make sure to use a new line for every `payload_properties` parameter here.

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

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
cp ${MY_ANDROID}/incremental_ota_update.zip ${server_ota_folder}
cd ${server_ota_folder}
unzip incremental_ota_update.zip
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 like this:

```
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 imx8qm-xen:fsl-imx8qm-mek-domu.dtb imx8qxp:fsl-imx8qxp-mek-car.dtb.

The OTA package only includes the first DTS_CONFIG definition DTS: fsl-imx8qm-mek-car.dtb, so the default OTA package can only be applied for the i.MX 8QuadMax MEK board. To generate an OTA package for the i.MX 8QuadXPlus MEK board, modify TARGET_BOARD_DTS_CONFIG as follows:

```
TARGET_BOARD_DTS_CONFIG := imx8qxp:fsl-imx8qxp-mek-car.dtb
imx8qm:fsl-imx8qm-mek-car.dtb imx8qm-xen:fsl-imx8qm-mek-domu.dtb
```

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/{product}/BoardConfig.mk file.

NOTE

Replace {product} with your product, such as mek_8q.

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/fs1/{product}/init_car.rc`:

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

NOTE

Replace `{product}` with your product, such as `mek_8q`.

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.google.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.google.android.car.bugreport;...;com.google.android.projection.sink;com.freescale.cactusp
```

```
layer;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 and i.MX 8QuadMax 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_1.0.2-auto-alpha.xml
$repo sync
$source trusty/vendor/google/aosp/scripts/envsetup.sh
$make imx8qm #for i.MX 8QuadMax
$cp ${TRUSTY_REPO_ROOT}/build-imx8qm/lk.bin ${MY_ANDROID}/vendor/nxp/fsl-proprietary/
uboot-firmware/imx8q/tee-imx8qm.bin
```

Then build the images and flash the u-boot-imx8qm.imx file to the target device.

NOTE

- For i.MX 8QuadXPlus, only replace `imx8qm` with `imx8qxp` in the commands above.
- `${TRUSTY_REPO_ROOT}` is the root directory of the Trusty OS repository.
- `${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. In addition, in the Alpha version, the RPMB key is the fixed 32-byte 0x00 one. Therefore, the user needs to initialize the RPMB key manually. The RPMB key cannot be changed once it is set.

1. Generate the RPMB key `rpmb_key_alpha.bin` on the Linux host and run the following commands:

```
echo -n "RPMB" > rpmb_key_alpha.bin
head -c 32 /dev/zero >> rpmb_key_alpha.bin
```

If you have difficulty to generate the RPMB key, use the prebuilt `rpmb_key_alpha.bin` in tools.

2. Make the board enter fastboot mode and flash the generated key to the board with the following commands:

```
fastboot stage rpmb_key_alpha.bin
fastboot oem set-rpmb-key
```

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

8.5 Rearview camera on the i.MX device

Exterior View System (EVS) is supported 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.

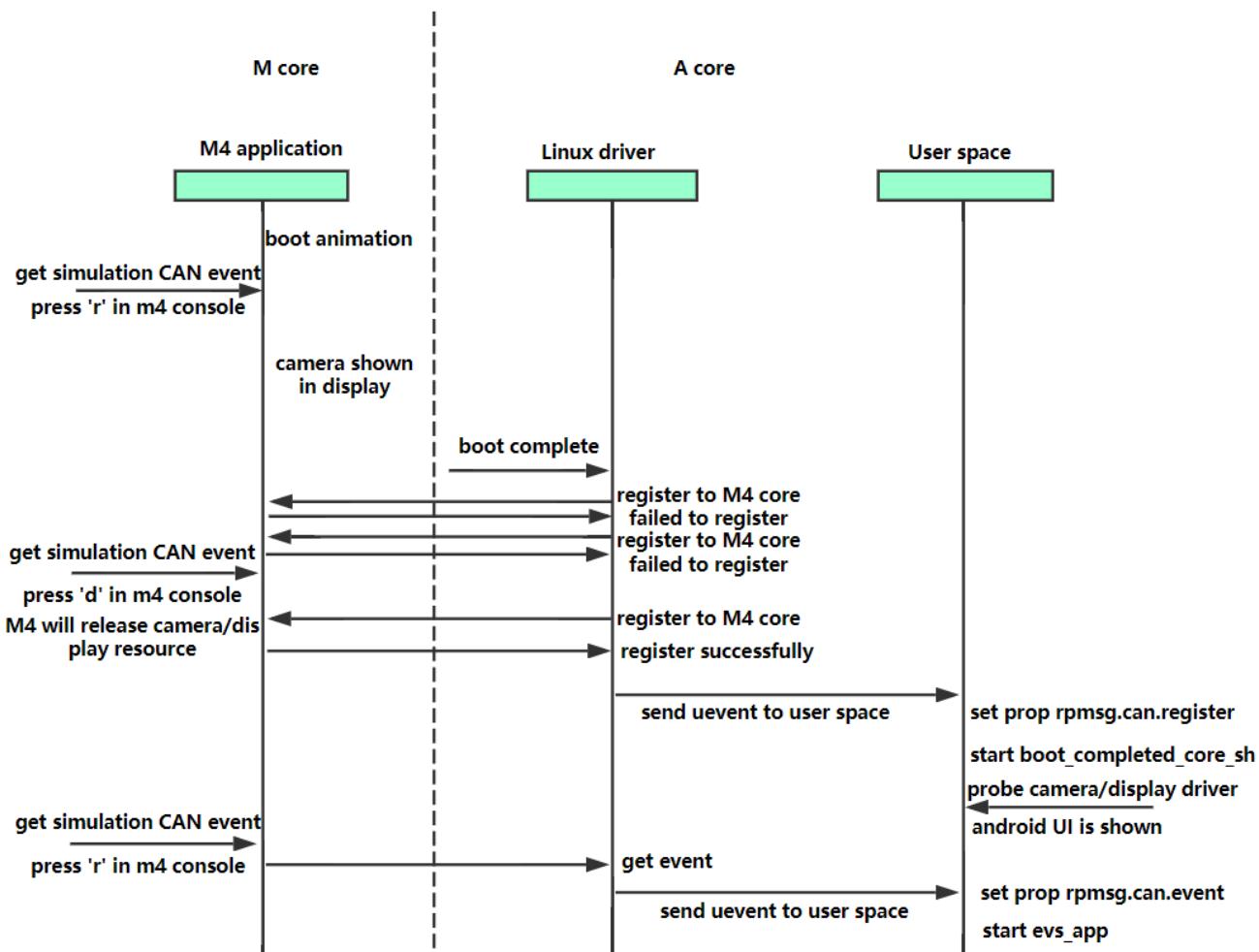


Figure 1. Sequence chart of EVS

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.
 - Cortex-M4 console on i.MX 8QuadMax MEK board: RS232 port on the base board.
3. Press **r** on the Cortex-M4 console when the board is powered on. The rearview camera appears on the screen. Press **d** when the following log is displayed on the Android console, and the Android UI appears on the screen.

```
can_rpmsg virtio2.rpmsg-can-channel.-1.1: rpmsg not ack 1!
```

4. Press **r** on the Cortex-M4 console after Android system boot is complete. The rearview camera appears on the screen. Press **d** on the Cortex-M4 console, and the Android UI appears on the screen.

NOTE

- Pressing **r** on the Cortex-M4 console means that the Cortex-M4 core gets the reverse signal. Pressing **d** on the Cortex-M4 console means that the Cortex-M4 core gets the drive signal.

8.5.2 How to customize the rearview camera

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

- Takes over control of the camera/display before Android OS is ready.
- Gets the CAN 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: <https://mcuxpresso.nxp.com/en/select>.

To update the Cortex-M4 image, run the following commands:

```
# Build Cortex-M4 images:
export ARMGCC_DIR=<path_to_GNUARM_GCC_installation_dir>
cd sdk_mcu/mek8m_Cortex-M4/boards/mekmimx8qm/demo_apps/rear_view_camera/cCortex-M4_core1/armgcc/
./build_all.sh
# The Cortex-M4 image is located in release/Cortex-M4_image.bin.
cp Cortex-M4_image.bin ${MY_ANDROID}/vendor/nxp/fsl-proprietary/mcu-sdk/imx8q/imx8qm_Cortex-M4_1_tcm_auto.bin
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/
```

8.6 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:

Customized Configuration

- 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.

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, `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.

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 `${MY_ANDROID}/vendor/nxp-opensource/kernel_imx/arch/arm64/configs/android_car_config`:
 - `CONFIG_SQUASHFS=y`
 - `CONFIG_SQUASHFS_LZ4=y`
 - `CONFIG_SQUASHFS_XATTR=y`
 - `CONFIG_SQUASHFS_DECOMP_MULTI=y`
2. Add the following configurations in `${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.

8.6.5 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, it takes 18.85 seconds for the i.MX 8QuadXPlus MEK board and 14.45 seconds for the i.MX 8QuadMax MEK board to boot the Android Automotive.

NOTE

As the HDMI monitor is used by the release prebuilt image, the HDMI monitor's initialization time should be excluded from the measured boot time. Different HDMI monitors may have different initialization time, which may vary from 1 to 3 seconds.

To get the initialization time of HDMI monitor, perform the following steps:

1. Connect HDMI-1 to the PC.
2. Connect HDMI-2 to the board.
3. Make the PC and board booted and displayed correctly and stop at the display of the board.
4. Use the HDMI monitor input source select menu to switch the current display from "board" to "PC". When pressing the button to start the "switching" operation, start the stopwatch at the same time.
5. When seeing the PC display on the HDMI monitor, stop the stopwatch.
6. The time from changing the HDMI monitor input source as PC to seeing the PC display on the HDMI monitor is the response time of this HDMI display.

9 Revision History

Table 13. 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

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, and UMEMS 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, μ Vision, 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.

© 2018 NXP B.V.

Document Number AUG
Revision P9.0.0_1.0.2-AUTO-alpha, 11/2018

