NXP Semiconductors
User's Guide

Document Number: AUG
Rev. O8.1.0_1.4.0_7ULP-ga, 06/2018

Android™ User's Guide

Contents

1 Overview....1
2 Preparation....1
3 Building the Android platform for i.MX....2
4 Running the Android Platform with a Prebuilt Image....7
5 Programming Images....8
6 Booting....10
7 Android Platform Update....12
8 Revision History....14

1 Overview

This document describes how to build Android Oreo 8.1 platform for the i.MX 7ULP devices. It provides instructions for:

- Configuring a Linux® OS build machine.
- Downloading, patching, and building the software components that create the Android™ system image.
- Building from sources and using pre-built images.
- Copying the images to boot media.
- Hardware/software configurations for programming the boot media and running the images.

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

2 Preparation

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.

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 is the most tested environment for the Android Oreo 8.1 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
```

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 i.MX 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-o8.1.0_1.4.0_7ulp-ga.tar.gz
```

3 Building the Android platform for i.MX

3.1 Getting Android source code (Android/kernel/U-Boot)

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

- NXP i.MX public source code, which is maintained in 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.

To generate the i.MX Android release source code build environment, follow the steps below. Assume you had i.MX Android proprietary source code package imx-o8.1.0_1.4.0_7ulp-ga.tar.gz under the ~/. directory.

```
$ mkdir ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo  > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=${PATH}:~/bin
```

```
$ source ~/imx-o8.1.0_1.4.0_7ulp_ga/imx_android_setup.sh
# By default, the imx_android_setup.sh script will create the source code build environment
in the folder ~/android_build
# ${MY_ANDROID} will be refered as the i.MX Android source code root directory in all i.MX
Andorid 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) and patched (Section 3.2).

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 evk_7ulp-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
evk_7ulp		i.MX 7ULP EVK 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	Production ready image similar with "user" but with root access and debug tools
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 6DualQuad SABRE Board/Platform device with user type.

   ```
   $ lunch evk_7ulp-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.

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/evk_7ulp:

- 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.
- boot-imx7ulp.img: a composite image for i.MX 7ULP EVK-SD, which includes the kernel zImage, ramdisk, board's device tree binary, and boot parameters.
- ramdisk.img: ramdisk image generated from "root/". Not used directly.
- system.img: EXT4 image generated from "system/". It can be programmed to "SYSTEM" partition on SD/eMMC card with "dd".
- recovery-imx7ulp.img: EXT4 image for i.MX 7ULP EVK-SD, which is generated from "recovery/". It can be programmed to "RECOVERY" partition on SD/eMMC card with "dd".
- partition-table.img: GPT partition table image, used for 8 GB SD card.
- partition-table-14GB.img: GPT partition table image, used for 16 GB SD card.
- partition-table-28GB.img: GPT partition table image, used for 32 GB SD card.
- u-boot-imx7ulp.imx: U-Boot image with no padding for i.MX 7ULP EVK-SD.
- vendor.img: vendor image, which holds platform binaries, mounted at /vendor.
- imx7ulp_m4_demo.img: prebuilt image for Arm Cortex-M4 core, used to supply some essential functions to run Linux OS.

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.

The default build is configured for internal eMMC boot storage. See Building Android image for the SD card on the SABRE-SD Board for the configuration steps to make SD card in Slot 3 as the boot storage.

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 7ULP EVK Board	$ lunch evk_7ulp-userdebug

After the `lunch` command is executed, the **make** command is issued:

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

3.2.2 User build mode

A production release Android system image is created by using the **user** 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/fsl/imx6/imx6.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: Setting the environment first, and then issuing the make command:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh   #set env
$ make PRODUCT-XXX user 2>&1 | tee build-log.txt
```

XXX depends on different boards. See the table below:

Table 4. Android system image production build method 1

i.MX development tool	Description	Image build command
Evaluation Kit	i.MX 7ULP	$ make PRODUCT-evk_7ulp-userdebug

Another method to create Android System Images is setting the environment and then issuing the `make` command. To set up the environment, execute the script `${MY_ANDROID}/build/envsetup.sh`. The `lunch` command and configuration argument for the Development Tool is then executed. To start the build, use the `make` command. The following table lists the `lunch` configuration values.

Table 5. Android system image production build method 2

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 7ULP	evk_7ulp-userdebug

An example for the i.MX 7ULP is:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_7ulp-userdebug
$ make
```

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

```
$ make otapackage
```

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

All the output files to create a bootable Android image should be all created in {MY_ANDROID}/out/target/product/<board>.

3.3 Building U-Boot images

After you set up U-Boot using the steps outlined above, you can find the tool (mkimage) under tools/.

```
$ cd ${MY_ANDROID}/vendor/nxp-opensource/uboot-imx
$ export ARCH=arm
$ export CROSS_COMPILE=${MY_ANDROID}/prebuilts/gcc/linux-x86/arm/arm-linux-
```

```
androideabi-4.9/bin/arm-linux-androideabi-
$ make distclean
```

For i.MX 7ULP EVK-SD SD:

```
# To build uboot.imx that is used in Android platform
$ make mx7ulp_evk_android_config
# To build uboot.img that is used in MFGTOOL
$ make mx7ulp_evk_defconfig
$ make
```

"u-boot.imx" is generated if you have a successful build.

NOTE

Any image that should be loaded by U-Boot must have an unique image head. For example, data must be added at the head of the loaded image to tell U-Boot about the image (in other words, whether it's a kernel or ramfs) and how to load the image (in other words, load/execute address). Before loading any image into RAM by U-Boot, you need a tool to add this information and generate a new image which can be recognized by U-Boot. The tool is delivered together with U-Boot. After you set up U-Boot using the steps outlined above, find the tool (mkimage) under tools/. The process of using mkimage to generate an image (for example, kernel image and ramfs image), which is to be loaded by U-Boot, is outlined in the subsequent sections of this document.

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=arm
$ export CROSS_COMPILE=${MY_ANDROID}/prebuilts/gcc/linux-x86/arm/arm-linux-
androideabi-4.9/bin/arm-linux-androideabi-

# Generate ".config" according to the default config file under arch/arm/configs.
# To build the kernel zImage for i.MX 7ULP
$ make imx_v7_android_defconfig
$ make KCFLAGS=-mno-android

# To build the zImage which is used in MFGTOOL
# zImage is under mfgtools\Profiles\Linux\OS Firmware\firmware\
$ make defconfig
$ make KCFLAGS=-mno-android -j4
```

With a successful build in either of the above case, the generated kernel images are:

```
${MY_ANDROID}/out/target/product/evk_7ulp/obj/KERNEL_OBJ/arch/arm/boot/Image
```

3.5 Building boot.img

boot.img and boota are default booting command and image.

As outlined in Running the Android Platform with a Prebuilt Image, we use boota as the default image to boot, not the uramdisk and uImage we used before.

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

```
# Boot image for i.MX 7ULP EVK-SD board
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_7ulp-userdebug
$ make bootimage
```

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 "Download Images" and "Boot".

Table 6. Image packages

Image package	Description
android_o8.1.0_1.4.0_7ulp-ga_image_7ulpevk.tar.gz	Prebuilt image for the i.MX 7ULP EVK board, which includes extended features.

The following table lists the detailed contents of android_o8.1.0_1.4.0_7ulp-ga_image_7ulpevk.tar.gz image packages.

It shows the prebuilt images to support the system boot from SD on the i.MX 7ULP EVK board.

Table 7. Images for i.MX 7ULP EVK SD boot

EVK SD image	Description
/u-boot-imx7ulp.imx	Bootloader (with padding) for the i.MX7ULP EVK-SD board
/partition-table.img	GPT table image for 8 GB SD card
/partition-table-14GB.img	GPT table Image for 16 GB SD card
/partition-table-28GB.img	GPT table Image for 32 GB SD card
/boot-imx7ulp.img	Boot Image for i.MX 7ULP EVK-SD board
/system.img	System Boot Image
/recovery-imx7d.img	Recovery Image for i.MX 7ULP EVK-SD board
/vendor.img	Vendor Image
/imx7ulp_m4_demo.img	Prebuilt image for Arm Cortex-M4 core on SoC

NOTE

boot.img is an Android image that stores zImage 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 is used to override any bootloader default boot arguments without changing it in the bootloader source code.

5 Programming Images

The images from the prebuilt release package or created from source code contain the U-Boot boot loader, system image, and recovery image. At a minium, the storage devices on the development system (SD) must be programmed with the U-Boot boot loader. The i.MX 7 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:

- MFGTool or fsl-sdcard-partition.sh to download all images to SD card.

5.1 System on SD

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

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

- U-Boot image: u-boot.imx
- boot image: boot.img
- Android system image: system.img
- Recovery image: recovery.img
- GPT table image: partition-table.img
- Vendor image: vendor.img
- vbmeta image: vbmeta.img
- Cortex-M4 image: imx7ulp_m4_demo.img

5.1.1 Storage partitions

To create storage partitions, use MFGTool as described in the *Android Quick Start Guide* (AQSUG), or use format tools in the prebuild directory.

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

- [Partition type/index] which is defined in the GPT.
- [Name] is only meaningful in the Android platform. Ignore it when creating these partitions.
- [Start Offset] shows where partition is started, unit in MB.

The SYSTEM partition is used to put the built Android system image. The DATA is used to put applications' unpacked codes/data, system configuration database, etc. In normal boot mode, the root file system is mounted from uramdisk. In recovery mode, the root file system is mounted from the RECOVERY partition.

Table 8. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	BOOT Loader	1 KB	1 MB - 1K	N/A	bootloader
1	Boot	2 MB	32 MB	boot.img format, kernel + ramdisk	boot.img

Table continues on the next page...

Table 8. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
2	Recovery	Follow Boot	32 MB	boot.img format, kernel + ramdisk	recovery.img
3	SYSTEM	Follow Recovery	1536 MB	EXT4. Mount as / system	Android system files under / system/ dir
4	CACHE	Follow SYSTEM.	512 MB	EXT4. Mount as / cache.	Android cache for image store of OTA
5	Misc	Follow CACHE	4 MB	N/A	For recovery store bootloader message, reserve
6	DATAFOOTER	Follow Misc	2 MB	N/A	For crypto footer of DATA partition encryption
7	METADATA	Follow Dtafootor	2 MB	N/A	For system slide show
8	PRESISTER	Follow Metadata	1 MB	N/A	Option to operate unlock \unlock
9	VENDOR	Follow PRESISTER	112 MB	EXT4. Mount as / vendor	vendor.img
10	DATA	Follow VENDOR	Total - Other images	EXT4. Mount at /data	Application data storage for system application, and for internal media partition, in /mnt/sdcard/ dir
11	FBMISC	Follow Data	1 MB	N/A	To store the state of lock \unlock
12	VBMETA	Follow FBMISC	1 MB	N/A	vbmeta.img, used to hold verification data

To create these partitions, use MFGTool described in the *Android Quick Start Guide* (AQSUG), or use script fsl-sdcard-partition.sh in the source code directory.

The script below can be used to partition an SD card as shown in the partition table above:

```
$ cd ${MY_ANDROID}/
$ sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> /dev/sdX
# <soc_name> can be imx7ulp.
```

NOTE

- The minimum size of the SD card is 8 GB.
- Use a USB SD card dongle. Do not write the card in the MMC slot of the Ubuntu computer.
- If the SD card is 8 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> /dev/sdX` to flash images.
- If the SD card is 16 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> -c 14 /dev/sdX` to flash images.
- If the SD card is 32 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> -c 28 /dev/sdX` to flash images.
- /dev/sdX, the X is the disk index from 'a' to 'z'. That may be different on each computer running Linux OS.
- Unmount all the SD card partitions before running the script.
- Put the related bootloader, boot image, system image, recovery image, GPT table image, and vendor image in your current directory. This script requires to install the simg2img tool on the computer. simg2img is a tool that converts the sparse system

image to raw system image on the Linux OS host computer. The android-tools-fsutils package includes the simg2img command for Ubuntu Linux OS.

5.1.2 Downloading images with MFGTool

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

6 Booting

This chapter describes booting from SD.

6.1 Booting from SD

This section describes how to boot from SD on the i.MX 7ULP EVK SD board.

6.1.1 Booting from SD on the i.MX 7ULP EVK-SD board

This section contains boot switch information and steps needed to bootup from SD.

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

Table 9. Boot switch settings

Boot switch	Download mode (MFGTool mode)	SD boot
SW4 (from 1-4 bit)	01xx	1001

To boot from the SD card, perform the following operations:

Change the board SW1 switch to 1001 (from 1-4 bit) . The default environment in boot.img is booting from SD. To use the default environment in boot.img, use the following command:

```
U-Boot > setenv bootargs
To clear the bootargs env, you can use the following command

U-Boot > setenv bootargs console=ttyLP0,115200 init=/init androidboot.console=ttyLP0
consoleblank=0 androidboot.hardware=freescale vmalloc=128M cma=448M [Optional]
U-Boot > saveenv                          [Save the environments]
```

6.2 Boot-up configurations

This section explains the common U-Boot environments used for NFS, MMC/SD boot, and kernel command line.

6.2.1 U-Boot environment

If you do not define the bootargs environment, it uses the default bootargs inside the image.

- bootcmd is the first variable to run after U-Boot boot.
- bootargs is 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, it uses the default bootargs inside the image.

  To use the default environment in boot.img, use the following command to clear the bootargs environment.

  ```
  > setenv bootargs
  ```
- dhcp: get the IP address by BOOTP protocol, and load the kernel image ($bootfile env) from the TFTP server.
- booti:

  booti 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" var in your U-Boot environment). To boot from mmcX, do the following:

  ```
  > booti mmcX
  ```

  To read the boot partition (the partition store boot.img, in this instance, mmcblk0p1), the X is the MMC bus number, which is the hardware MMC bus number, in i.MX 6Dual/6Quad SABRE-SD and i.MX 6Solo/6DualLite SABRE-SD boards. eMMC is mmc3.

  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 from

  ```
  > boota 0xXXXXXXXX
  ```
- bootm (only works in for the NFS) starts running the kernel. For other use cases, use booti command.

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 10. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
console	Where to output kernel log by printk.	console=ttymxc0,115200	All use cases.
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.hardware	Specifies hardware name for this product. Android init process loads the configuration file init.$	androidboot.hardware=freescale	All use cases.

Table continues on the next page...

Table 10. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
	(androidboot.hardware).rc in the root directory.		
vmalloc	vmalloc virtual range size for kernel.	vmalloc=128M	vmalloc=<size>
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=448M	It is 448 MB by default.
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 Oreo 8.1 CTS requirement: The serial input should be disabled by default. Setting this argument enables console serial input, which violates the CTS requirement.

7 Android Platform Update

7.1 Building an Android update package

Android build system supports auto generation of update.zip function. It generates the updater_script and all system.img files. You can use the following command to generate an OTA package under Android environment:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_7ulp-userdebug
$ make otapackage -j4
```

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

```
out/target/proudct/evk_7ulp/evk_7ulp-ota-${date}-${soc}.zip
```

7.2 Updating the Android platform

7.2.1 Using ADB to update Android platform

This section describes how to use ADB to update Android platform:

1. Enter recovery mode.

   There are two methods to enter recovery mode:

   Method 1: If the board supports physical **Power/VOLUME DOWN/VOLUME UP** keys, press **VOLUME DOWN** and **Power** to enter recovery mode when the system is powered on.

   Method 2: Enter the following command on the board console:

   ```
   $ reboot recovery
   ```

   - When the system completes booting up into recovery mode, it displays an Android Robot Logo.
   - Move the menu item by **VOLUME UP** and **VOLUME DOWN** button.
   - Select the **Apply update from ADB** item by **Power Key**.
2. Download the OTA package.

   You can build an OTA package from "7.1 Build Android Update Package". Make sure your host has the ADB driver, and then connect the board with your host.

   Execute the following command:

   ```
   $ adb sideload $YOUR_UPDATE_PACKAGE.zip
   ```

   After the package is sent, the system starts updating the firmware with the updated file.

7.2.2 Using an application to update Android platform

There is a reference OTA application under `vendor/nxp-opensource/fsl_imx_demo/FSLOta`, which can do the OTA work.

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, and then put the following files to `${http_root}/evk_7ulp_${ota_folder_suffix}_${version}`:

   ```
   ${MY_ANDROID}/out/target/product/evk_7ulp/system/build.prop
   ${MY_ANDROID}/out/target/product/evk_7ulp/evk_7ulp-ota-${date}-${soc}.zip
   ```

   NOTE

   - evk_7ulp-ota-${date}-${soc}.zip is built from 7.1 Build Android Update Package.
     - There may be two or more OTA packages like this:

       evk_7ulp-ota-20180529-imx7ulp-mipi.zip and evk_7ulp-ota-20180529-imx7ulp.zip

     - The OTA application only supports updating evk_7ulp-ota-20180529-imx7ulp.zip.
     - To update evk_7ulp-ota-20180529-imx7ulp-mipi.zip, change its name to evk_7ulp-ota-20180529-imx7ulp.zip.
   - ${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 | grep "ro.build.version.release"
```

2. Configure the OTA server IP address and port number.
   The OTA configuration file (/vendor/etc/ota.conf) content is like this:

   ```
   server=192.168.1.100
   port=10888
   ota_folder_suffix=oreo
   ```

   Modify it to fit your environment.

3. Open the OTA application.

   The reference application is a dialog activity, and can be enabled in the **Settings** -> **About tablet** -> **Additional system Update** menu. The application downloads this `build.prop`, parse it to get the build time, and compare with the build time of itself.

   If the server build is newer, it shows the version information and package size, and prompts user to upgrade. Click the **Update** button to update Android platform. Then you can see the download starting.

   After downloading is finished, the title changes to "Verify package". During this time, it is actually doing the RecoverySystem.verifyPackage() API to verify whether the package is complete, such as an MD5 checksum checking.

   It also checks whether the key chain in the package aligns with the key chain in the device. After the verification is finished, it calls the RecoverySystem.installPackage() API to install the package. It writes a recovery command and stores it into the `/cache/recovery/command` file, and then reboots. After reboot, the system boots to recovery mode.

   After it installs the update package, it displays the "Android Robot" spinning on the screen. If there is an error, it displays "Error Robot" and stops spinning. Press the **MENU** button to view the log output on the screen.

8 Revision History

Table 11. Revision history

Revision number	Date	Substantive changes
N7.1.2_1.1.0_7ULP-PRC	06/2017	Initial release
O8.1.0_1.4.0_7ULP-ga	06/2018	i.MX 7ULP GA release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

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

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

NXP, the NXP logo, Freescale, and the Freescale logo are trademarks of NXP B.V. Arm, the Arm logo, and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All other product or service names are the property of their respective owners. All rights reserved.
© 2018 NXP B.V.

Document Number: AUG
Rev. O8.1.0_1.4.0_7ULP-ga
06/2018