Document Number: IMXLUG
Rev. L4.9.88_2.1.0_8mm-alpha, 06/2018

i.MX Linux® User's Guide

Contents

1 Overview

This document describes how to build and install the i.MX Linux® OS BSP, where BSP stands for Board Support Package, on the i.MX platform. It also covers special i.MX features and how to use them.

The document also provides the steps to run the i.MX platform, including board DIP switch settings, and instructions on configuring and using the U-Boot bootloader.

The later chapters describe how to use some i.MX special features when running the Linux OS kernel.

Features covered in this guide may be specific to particular boards or SOCs. For the capabilities of a particular board or SOC, see the *i.MX Linux® Release Notes* (IMXLXRN).

1.1 Audience

This document is intended for software, hardware, and system engineers who are planning to use the product, and for anyone who wants to know more about the product.

1.2 Conventions

This document uses the following conventions:

- `Courier New` font: This font is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.

1.3 Supported hardware SoCs and boards

These are the systems covered in this guide:

- i.MX 8MQuad EVK platform
- i.MX 8MMini EVK Board

Some abbreviations are used in places in this document.

- SABRE-SD refers to the i.MX 6Quad SABRE-SD, i.MX 6DualLite SABRE-SD, i.MX 6QuadPlus SABRE-SD, and i.MX 7Dual SABRE-SD boards.
- SABRE-AI refers to the i.MX 6Quad SABRE-AI, i.MX 6DualLite SABRE-AI, and i.MX 6QuadPlus SABRE-AI boards.
- SoloLite refers to the i.MX 6SoloLite Board.
- 8MQ refers to the 8MQuad platform.

1.4 References

i.MX has multiple families supported in software. The folllowing are the listed families and SoCs per family. The i.MX Linux® Release Notes will describe which SoC is supported in current release. Some previously released SoC might be buildable in current release but not validated if they are at the previous validated level.

- i.MX 6 Family: 6QuadPlus, 6Quad, 6DualLite, 6SoloX, 6SoloLite, 6SLL 6UltraLite, 6ULL
- i.MX 7 Family: 7Dual, 7ULP
- i.MX 8 Family: 8QuadMax
- i.MX 8M Family: 8MQuad, 8MMini
- i.MX 8X Family: 8QuadXPlus

This release includes the following references and additional information.

- *i.MX Linux® Release Notes* (IMXLXRN) - Provides the release information.
- *i.MX Linux® User's Guide* (IMXLUG) - Contains the information on installing U-Boot and Linux OS and using i.MX-specific features.
- *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG) - Contains the instructions for setting up and building Linux OS in the Yocto Project.
- *i.MX Reference Manual* (IMXLXRM) - Contains the information on Linux drivers for i.MX.
- *i.MX Graphics User's Guide* (IMXGRAPHICUG) - Describes the graphics features.
- *i.MX BSP Porting Guide* (IMXXBSPPG) - Contains the instructions on porting the BSP to a new board.

2 Introduction

The i.MX Linux BSP is a collection of binary files, source code, and support files that can be used to create a U-Boot bootloader, a Linux kernel image, and a root file system for i.MX development systems. The Yocto Project is the framework of choice to build the images described in this document, although other methods can be used.

All the information on how to set up the Linux OS host, how to run and configure a Yocto Project, generate an image, and generate a rootfs, are covered in the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG).

When Linux OS is running, this guide provides information on how to use some special features that i.MX SoCs provide. The release notes provide the features that are supported on a particular board.

3 Basic Terminal Setup

The i.MX boards can communicate with a host server (Windows® OS or Linux OS) using a serial cable. Common serial communication programs such as HyperTerminal, Tera Term, or PuTTY can be used. The example below describes the serial terminal setup using HyperTerminal on a host running Windows OS.

1. Connect the target and the PC running Windows OS using a serial cable or a micro-B USB cable on the i.MX 8 board.
2. Open HyperTerminal on the PC running Windows OS and select the settings as shown in the following figure.

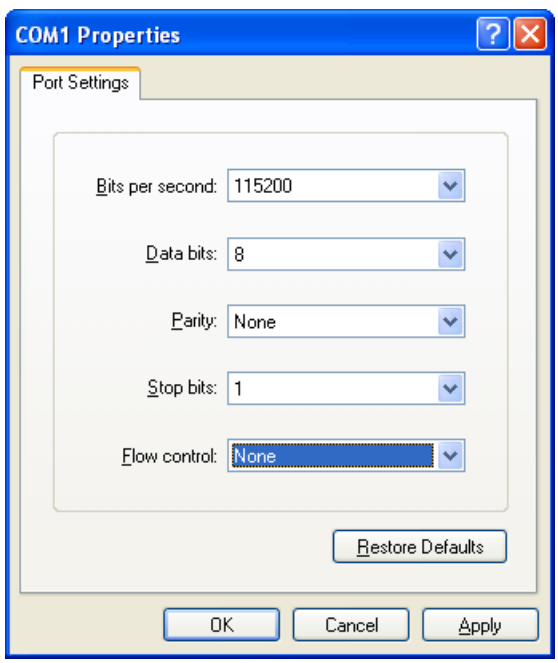

Figure 1. Teraterm settings for terminal setup

The i.MX 8 board connects the host driver using the micro USB connector. The USB to serial driver can be found under www.ftdichip.com/Drivers/VCP.htm. The FT4232 USB to serial convertor provides four serial ports. The i.MX 8 board uses the first port for the Arm® Cortex®-A cores console and the second port for SCU's console. Users need to select the first port (COM) in the terminal setup.

4 Booting Linux OS

Before booting the Linux OS kernel on an i.MX board, copy the images (U-Boot, Linux kernel, device tree, and rootfs) to a boot device and set the boot switches to boot that device. There are various ways to boot the Linux OS for different boards, boot devices, and results desired. This section describes how to prepare a boot device, where files need to be in the memory map, how to set switches for booting, and how to boot Linux OS from U-Boot.

4.1 Software overview

This section describes the software needed for the board to be able to boot and run Linux OS.

To boot a Linux image on i.MX 8QuadMax and i.MX 8QuadXPlus, four elements are needed:

- Bootloader (imx-boot built by imx-mkimage), which includes U-Boot, Arm Trusted Firmware, DCD file, and System controller firmware.
- Linux kernel image (Image built by linux-imx)
- A device tree file (.dtb) for the board being used
- A root file system (rootfs) for the particular Linux image

On i.MX 8MQuad and i.MX 8MMini, four elements are needed:

- imx-boot (built by imx-mkimage), which includes SPL, U-Boot, ARM trusted firmware, DDR firmware, and HDMI firmware
- Linux kernel image
- A device tree file (.dtb) for the board being used.
- A root file system (rootfs) for the particular Linux image

The system can be configured for a specific graphical backend. For i.MX 8, the graphical backends are X11, XWayland, and Frame Buffer. For i.MX 7ULP, the default backend is XWayland.

4.1.1 Bootloader

U-Boot is the tool recommended as the bootloader for i.MX 6 and i.MX 7. i.MX 8 requires a bootloader that includes U-boot as well as other components described below. U-Boot must be loaded onto a device to be able to boot from it. U-Boot images are board-specific and can be configured to support booting from different sources.

The pre-built or Yocto project default bootloader names start with the name of the bootloader followed by the name of the platform and board and followed by the name of the device that this image is configured to boot from: `u-boot-[platform][board]_[machine_configuration].bin`. If no boot device is specified, it boots from SD/MMC.

The manufacturing tool can be used to load U-Boot onto all devices with i.MX 6 and i.MX 7. U-Boot can be loaded directly onto an SD card using the Linux `dd` command. U-Boot can be used to load a U-Boot image onto some other devices.

On i.MX 8, the U-Boot cannot boot the device by itself. The i.MX 8 pre-built images or Yocto Project default bootloader is imx-boot for the SD card, which is created by the imx-mkimage. The imx-boot binary includes the Uboot, ARM trusted firmware, DCD file (8QuadMax/8QuadXPlus), system controller firmware (8QuadMax/8QuadXPlus), SPL (8MQuad and 8MMini), DDR firmware (8MQuad), and HDMI firmware (8MQuad).

On i.MX 8MQuad, the second program loader (SPL) is enabled in U-Boot. SPL is implemented as the first-level bootloader running on TCML (due to OCRAM size limitation). It is used to initialize DDR and load U-Boot, U-Boot DTB, Arm trusted firmware, and TEE OS (optional) from the boot device into the memory. After SPL completes loading the images, it jumps to the Arm trusted firmware BL31 directly. The BL31 starts the optional BL32 (TEE OS) and BL33 (u-boot) for continue booting kernel.

In imx-boot, the SPL is packed with DDR Firmware together, so that ROM can load them into Arm Cortex-M4 TCML. The U-Boot, U-Boot DTB, Arm Trusted firmware, and TEE OS (optional) are packed into a FIT image, which is finally built into imx-boot.

4.1.2 Linux kernel image and device tree

This i.MX BSP contains a pre-built kernel image based on the 4.9.88 version of the Linux kernel and the device tree files associated with each platform.

Device trees are tree data structures, which describe the hardware configuration allowing a common kernel to be booted with different pin settings for different boards or configurations. Device tree files use the .dtb extension. The configuration for a device tree can be found in the Linux source code under arch/arm/boot/dts in the *.dts files.

The i.MX Linux delivery package contains pre-built device tree files for the i.MX boards in various configurations. File names for the prebuilt images are named Image-[platform]-[board]-[configuration].dtb.

For i.MX 6 and i.MX 7, the *ldo.dtb device trees are used for LDO-enabled feature support. By default, the LDO bypass is enabled. If your board has the CPU set to 1.2 GHz, you should use the *ldo.dtb device tree instead of the default, because LDO bypass mode is not supported on the CPU at 1.2 GHZ. The device tree *hdcp.dtb is used to enable the HDCP feature because of a pin conflict, which requires this to be configured at build time.

On i.MX 8, the kernel is 64 bit and device trees are located in the arch/arm64/boot/dts/freescale folder and use the dts extension. The kernel is built using linux-imx software provided in the release package and the file name starting with Image.

4.1.3 Root file system

The root file system package (or rootfs) provides busybox, common libraries, and other fundamental elements.

The i.MX BSP package contains several root file systems. They are named with the following convention: `[image name]-[backend]-[platform][board].[ext4|sdcard]`. The ext4 extension indicates a standard file system. It can be mounted as NFS, or its contents can be stored on a boot media such as an SD/MMC card.

The graphical backend to be used is also defined by the rootfs.

4.2 Manufacturing Tool

The Manufacturing Tool (MfgTool) runs on a Windows OS host and is used to download images to different devices on an i.MX board. The tar.gz file can be found with the pre-built images.

4.2.1 Configuring MfgTool

Decompress mfgtools-with-rootfs.zip and mfgtools-without-rootfs.zip.

Instructions for MfgTool V2 can be found in the file `Profiles/Linux/OS Firmware/ucl2.xml`. Read and update the `ucl2.xml` file to understand the operations before using MfgTool.

Skip this step if a board in the supported list is used.

It is important to correctly configure the `cfg.ini` and `UICfg.ini` files. For example, if only one board is supported, `PortMgrDlg=1` should be set in `UICfg.ini`. If four boards are supported, `PortMgrDlg=4` should be set. An incorrect configuration causes MfgTool to malfunction.

4.2.2 Using MfgTool

Follow these instructions to use the MfgTool V2 for i.MX 6, i.MX 7, and i.MX 8MQuad:

1. Connect a USB cable from a computer to the USB OTG port on the board.
2. Connect a USB cable from the OTG-to-UART port to the computer for console output.
3. Open a Terminal emulator program. See Section "Basic Terminal Setup" in this document.
4. Set the boot pin to MfgTool mode. See Section "Serial download mode for the Manufacturing Tool" in this document.
5. Choose the correct vbs file and double-click it to launch the MfgTool host tool.
6. The default profile of the Manufacturing Tool assumes that your file system is packed and compressed using the bzip2 algorithm. An example can be found in the MfgTool release package in the folder `Profiles/Linux/OS Firmware/`

`files`. To create this file, run the following commands as a root user on Linux OS. You can also modify the profile to support other formats.

7. After the image downloading is done, set the boot pin to boot up the board. See Section "How to boot the i.MX boards" in this document.

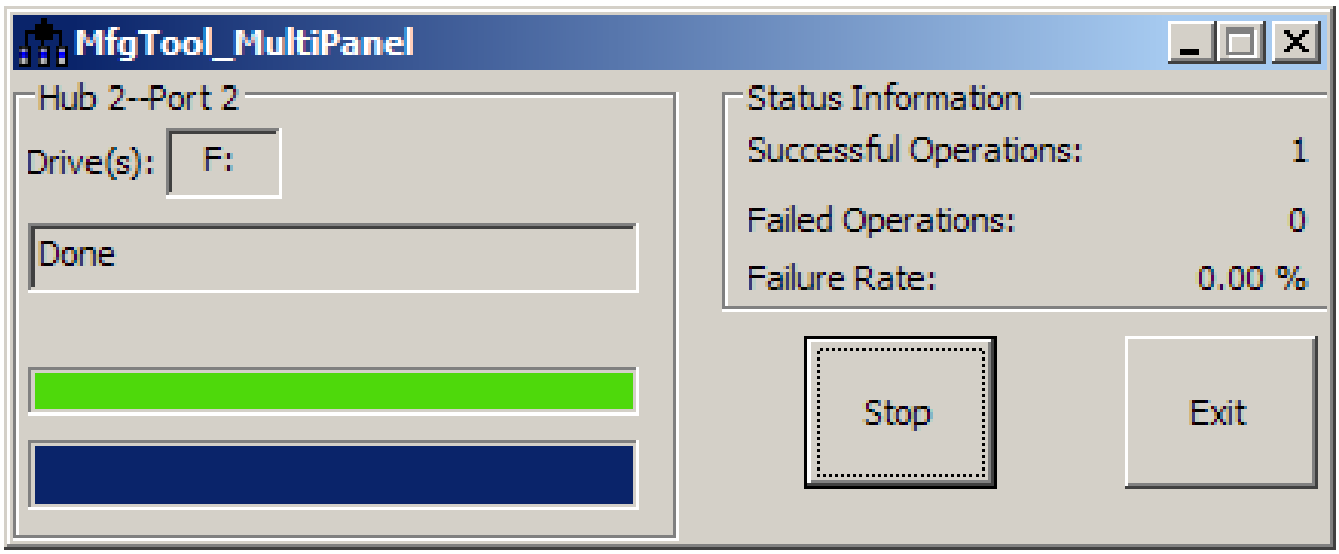

Figure 2. Programming SD with the Manufacturing Tool – image downloading

NOTE

The Manufacturing Tool may sometimes report an error message when it is downloading the file system to an SD card. This can be caused by insufficient space on the SD card due to a small partition size. To fix this, decompress the file `Profiles/Linux/OS Firmware/mksdcard.sh.tar` and modify the script to increase the size of the partition and create more partitions according to your file system requirements. After the modification is done, compress the script again.

4.3 Preparing an SD/MMC card to boot

This section describes the steps to prepare an SD/MMC card to boot up an i.MX board using a Linux host machine. These instructions apply to SD and MMC cards although for brevity, often only SD card is listed.

For a Linux image to be able to run, four separate pieces are needed:

- Linux OS kernel image (zImage/Image)
- Device tree file (*.dtb)
- Bootloader image
- Root file system (i.e., EXT4)

The Yocto Project build creates an SD card image that can be flashed directly. This is the simplest way to load everything needed onto the card with one command.

An .sdcard image contains all four images properly configured for an SD card. The release contains a pre-built .sdcard image that is built specifically for the one board configuration. It runs the X11 graphical backend. It does not run on other boards unless U-Boot, the device tree, and rootfs are changed.

When more flexibility is desired, the individual components can be loaded separately, and those instructions are included here as well. An SD card can be loaded with the individual components one-by-one or the .sdcard image can be loaded and the individual parts can be overwritten with the specific components.

The rootfs on the default .sdcard image is limited to a bit less than 4 GB, but re-partitioning and re-loading the rootfs can increase that to the size of the card. The rootfs can also be changed to specify the graphical backend that is used.

The device tree file (.dtb) contains board and configuration-specific changes to the kernel. Change the device tree file to change the kernel for a different i.MX board or configuration.

By default, the release uses the following layout for the images on the SD card. The kernel image and DTB move to use the FAT partition without a fixed raw address on the SD card. The users have to change the U-Boot boot environment if the fixed raw address is required.

Table 1. Image layout

Start address (sectors)	Size (sectors)	Format	Description
0xa00000 bytes (20480)	500 Mbytes (1024000)	FAT	Kernel zImage and DTBs
0x25800000 bytes (1228800)	Remaining space	Ext3/Ext4	Rootfs

4.3.1 Preparing the card

An SD/MMC card reader, such as a USB card reader, is required. It is used to transfer the bootloader and kernel images to initialize the partition table and copy the root file system. To simplify the instructions, it is assumed that a 4 GB SD/MMC card is used.

Any Linux distribution can be used for the following procedure.

The Linux kernel running on the Linux host assigns a device node to the SD/MMC card reader. The kernel might decide the device node name or udev rules might be used. In the following instructions, it is assumed that udev is not used.

To identify the device node assigned to the SD/MMC card, carry out the following command:

```
$ cat /proc/partitions
major minor  #blocks  name
   8     0   78125000 sda
   8     1   75095811 sda1
   8     2          1 sda2
   8     5    3028221 sda5
   8    32  488386584 sdc
   8    33  488386552 sdc1
   8    16    3921920 sdb
   8    18    3905535 sdb1
```

In this example, the device node assigned is /dev/sdb (a block is 1024 Bytes).

NOTE

Make sure that the device node is correct for the SD/MMC card. Otherwise, it may damage your operating system or data on your computer.

4.3.2 Copying the full SD card image

The SD card image (with the extension .sdcard) contains U-Boot, the Linux image and device trees, and the rootfs for a 4 GB SD card. The image can be installed on the SD card with one command if flexibility is not required.

Carry out the following command to copy the SD card image to the SD/MMC card. Change sdx below to match the one used by the SD card.

```
$ sudo dd if=<image name>.sdcard of=/dev/sdx bs=1M && sync
```

The entire contents of the SD card are replaced. If the SD card is larger than 4 GB, the additional space is not accessible.

4.3.3 Partitioning the SD/MMC card

The full SD card image already contains partitions. This section describes how to set up the partitions manually. This needs to be done to individually load the bootloader, kernel, and rootfs.

There are various ways to partition an SD card. Essentially, the bootloader image needs to be at the beginning of the card, followed by the Linux image and the device tree file. These can either be in separate partitions or not. The root file system needs to be in a partition that starts after the Linux section. Make sure that each section has enough space. The example below creates two partitions.

On most Linux host operating systems, the SD card is mounted automatically upon insertion. Therefore, before running fdisk, make sure that the SD card is unmounted if it was previously mounted (through `sudo umount /dev/sdx`).

Start by running fdisk with root permissions. Use the instructions above to determine the card ID. We are using `sdx` here as an example.

```
$ sudo fdisk /dev/sdx
```

Type the following parameters (each followed by <ENTER>):

```
p          [lists the current partitions]
d          [to delete existing partitions.  Repeat this until no unnecessary partitions
             are reported by the 'p' command to start fresh.]

n          [create a new partition]
p          [create a primary partition - use for both partitions]
1          [the first partition]
20480      [starting at offset sector]
1024000    [size for the first partition to be used for the boot images]
p          [to check the partitions]

n
p
2
1228800    [starting at offset sector, which leaves enough space for the kernel,
                the bootloader and its configuration data]
<enter>    [using the default value will create a partition that extends to
                the last sector of the media]
p          [to check the partitions]
w          [this writes the partition table to the media and fdisk exits]
```

4.3.4 Copying a bootloader image

This section describes how to load only the bootloader image when the full SD card image is not used. For i.MX 6 and i.MX 7, execute the following command to copy the U-Boot image to the SD/MMC card.

```
$ sudo dd if=<U-Boot image> of=/dev/sdx bs=1k seek=<offset> conv=fsync
```

Where offset is:

- 1 - for i.MX 6 or i.MX 7
- 33 - for i.MX 8

NOTE

For i.MX 8QuadMax, i.MX 8QuadXPlus, i.MX 8MQuad, and i.MX 8MMini, the offset should be 33k, so bs=1k, seek=33.

The first 16 KB of the SD/MMC card, which includes the partition table, is reserved.

4.3.5 Copying the root file system (rootfs)

This section describes how to load the rootfs image when the full SD card image is not used.

Copy the target file system to a partition that only contains the rootfs. This example uses partition 2 for the rootfs. First format the partition. The file system format ext3 or ext4 is a good option for the removable media due to the built-in journaling. Replace sdx with the partition in use in your configuration.

```
$ sudo mkfs.ext3 /dev/sdx2
Or
$ sudo mkfs.ext4 /dev/sdx2
```

Copy the target file system to the partition:

```
$ mkdir /home/user/mountpoint
$ sudo mount /dev/sdx2 /home/user/mountpoint
```

Extract a rootfs package to a directory: for example, extract `fsl-image-qt5-validation-imx-imx7ulpevk.tar.bz2` to `/home/user/rootfs` for example:

```
$ cd /home/user/rootfs
$ tar -jxvf fsl-image-qt5-validation-imx-imx7ulpevk.tar.bz2
```

The rootfs directory needs to be created manually.

Assume that the root file system files are located in `/home/user/rootfs` as in the previous step:

```
$ cd /home/user/rootfs
$ sudo cp -a * /home/user/mountpoint
$ sudo umount /home/user/mountpoint
$ sudo umount /home/user/rootfs
$ sync
```

The file system content is now on the media.

NOTE

Copying the file system takes several minutes depending on the size of your rootfs.

4.4 Downloading images

Images can be downloaded to a device using a U-Boot image that is already loaded on the boot device or by using the Manufacturing Tool, MfgTool. Use a terminal program to communicate with the i.MX boards.

4.4.1 Downloading images using U-Boot

The following sections describe how to download images using the U-Boot bootloader.

The commands described below are generally useful when using U-Boot. Additional commands and information can be found by typing `help` at the U-Boot prompt.

The U-Boot `print` command can be used to check environment variable values.

The `setenv` command can be used to set environment variable values.

4.4.1.1 Downloading an image to MMC/SD

This section describes how to download U-Boot to an MMC/SD card that is not the one used to boot from.

Insert an MMC/SD card into the SD card slot. This is slot SD3 on i.MX 6 SABRE boards and SD1 on i.MX 6SoloLite boards, SD2 on i.MX 6UltraLite EVK board and i.MX 6ULL EVK board, SD1 on i.MX 7Dual SABRE-SD board and i.MX 7ULP EVK board (MicroSD), and SD1 on i.MX 8QuadMax MEK board, QuadXPlus MEK board, and i.MX 8MQuad EVK board.

NOTE

To enable the full features for i.MX 7ULP, burn the Arm® Cortex®-M4 image to QuadSPI. It is recommended to use the mfgtool script "mfgtool2-yocto-mx-evk-sdcard-sd1-m4-ulp.vbs" to burn both BSP and Arm Cortex-M4 images.

For i.MX 7ULP, to burn the Arm Cortext-M4 image to QuadSPI, perform the following steps:

1. Copy the Arm Cortex-M4 image to the SD card vfat partition, insert the SD card, and then boot to the U-Boot console.
2. Probe the Quard SPI in U-Boot, and erase an enough big size QuardSPI flash space for this Arm Cortext-M4 image.

   ```
   U-Boot   >   sf probe
   U-Boot   >    sf erase 0x0 0x30000;
   ```
3. Read the Arm Cortext-M4 image (in the first vfat partition on the SD card) to memory address 0x0x62000000, the Arm Cortext-M4 image name is sdk20-app.img here.

   ```
   U-Boot   >    fatload mmc 0:1 0x62000000 <m4_binary>.img;
   ```
4. Write the Arm Cortext-M4 image to the QuardSPI.

   ```
   U-Boot   >     sf write 0x62000000 0x0 0x30000
   ```

To flash the original U-Boot, see Section Preparing an SD/MMC card to boot.

The U-Boot bootloader is able to download images from a TFTP server into RAM and to write from RAM to an SD card. For this operation, the Ethernet interface is used and U-Boot environment variables are initialized for network communications.

The boot media contains U-Boot, which is executed upon power-on. Press any key before the value of the U-Boot environment variable, "bootdelay", decreases and before it times out. The default setting is 1 second to display the U-Boot prompt.

1. To clean up the environment variables stored on MMC/SD to their defaults, execute the following command in the U-Boot console:

   ```
   U-Boot > env default -f -a
   U-Boot > saveenv
   U-Boot > reset
   ```
2. Configure the U-Boot environment for network communications. The folllowing is an example. The lines preceded by the "#" character are comments and have no effect.

   ```
   U-Boot > setenv serverip <your TFTPserver ip>
   U-Boot > setenv bootfile <your kernel zImage/Image name on the TFTP server>
   U-Boot > setenv fdt_file <your dtb image name on the TFTP server>
   ```

   The user can set a fake MAC address through ethaddr enviroment if the MAC address is not fused.

   ```
   U-Boot > setenv ethaddr 00:01:02:03:04:05
   U-Boot > save
   ```
3. Copy zImage/Image to the TFTP server. Then download it to RAM:

   ```
   U-Boot > dhcp
   ```
4. Query the information about the MMC/SD card.

   ```
   U-Boot > mmc dev
   U-Boot > mmcinfo
   ```

5. Check the usage of the "mmc" command. The "blk#" is equal to "<the offset of read/write>/<block length of the card>". The "cnt" is equal to "<the size of read/write>/<block length of the card>".

```
U-Boot > help mmc
mmc - MMC sub system

Usage:
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
```

6. Program the kernel zImage/Image located in RAM at `${loadaddr}` into the SD card. For example, the command to write the image with the size 0x800000 from ${loadaddr} to the offset of 0x100000 of the microSD card. See the following examples for the definition of the mmc parameters.

```
blk# = (microSD Offset)/(SD block length) = 0x100000/0x200 = 0x800

cnt = (image Size)/(SD block length) = 0x800000/0x200 = 0x4000
```

   This example assumes that the kernel image is equal to 0x800000. If the kernel image exceeds 0x800000, increase the image length. After issuing the TFTP command, filesize of the U-Boot environment variable is set with the number of bytes transferred. This can be checked to determine the correct size needed for the calculation. Use the U-Boot command printenv to see the value.

```
U-Boot >  mmc dev 2 0
U-Boot > tftpboot ${loadaddr} ${bootfile}
### Suppose the kernel zImage is less than 8M.
U-Boot > mmc write ${loadaddr} 0x800 0x4000
```

7. Program the dtb file located in RAM at `${fdt_addr}` into the microSD.

```
U-Boot > tftpboot ${fdt_addr} ${fdt_file}
U-Boot > mmc write ${fdt_addr} 0x5000 0x800
```

4.4.2 Using an i.MX board as the host server to create a rootfs

Linux OS provides multiple methods to program images to the storage device. This section describes how to use the i.MX platform as a Linux host server to create the rootfs on an MMC/SD card or the SATA device. The following example is for an SD card. The device file node name needs to be changed for a SATA device.

1. Boot from NFS or other storage. Determine your SD card device ID. It could be mmcblk* or sd*. (The index is determined by the USDHC controller index.) Check the partition information with the command:

```
$ cat /proc/partitions
```

2. To create a partition on the MMC/SD card, use the fdisk command (requires root privileges) in the Linux console:

```
root@freescale ~$ sudo fdisk /dev/$SD
```

   Replace $SD above with the name of your device.
3. If this is a new SD card, you may get the following message:

```
The device contains neither a valid DOS partition table, nor Sun, SGI or OSF disk label
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that the previous content
won't be recoverable.
The number of cylinders for this disk is set to 124368.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
```

```
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
```

The usual prompt and commands to partition the card are as follows. Text in boldface indicates what the user types.

```
Command (m for help): p

Disk /dev/sdd: 3965 MB, 3965190144 bytes
4 heads, 32 sectors/track, 60504 cylinders, total 7744512 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00080bff

        Device Boot      Start         End      Blocks   Id System
```

4. As described in Flash memory maps, the rootfs partition should be located after the kernel image. The first 0x800000 bytes can be reserved for MBR, bootloader, and kernel sections. From the log shown above, the Units of the current MMC/SD card is 32768 bytes. The beginning cylinder of the first partition can be set to "0x300000/32768 = 96." The last cylinder can be set according to the rootfs size. Create a new partition by typing the letters in bold:

```
   Command (m for help): n
    e extended
    p primary partition (1-4)
   Select (default p): p
   Partition number (1-4): 1
   First cylinder (1-124368, default 1): 96
   Last cylinder or +size or +sizeM or +sizeK (96-124368, default 124368): Using
default value 124368

   Command (m for help): w
   The partition table has been altered!
   Calling ioctl() to re-read $SD partition table
```

5. Check the partitions (see above) to determine the name of the partition. $PARTITION is used here to indicate the partition to be formatted. Format the MMC/SD partitions as ext3 or ext4 type. For example, to use ext3:

```
root@freescale ~$ mkfs.ext3 /dev/$PARTITION
mke2fs 1.42 (29-Nov-2011)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
248992 inodes, 994184 blocks
49709 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1019215872
31 block groups
32768 blocks per group, 32768 fragments per group
8032 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 20 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

6. Copy the rootfs contents to the MMC/SD card. The name may vary from the one used below. Check the directory for the rootfs desired. (Copy the *.ext2 to NFS rootfs).

```
mkdir /mnt/tmpmnt
mount -t ext3 -o loop /fsl-image-gui-imx6qsabresd.ext3 /mnt/tmpmnt
cd /mnt
mkdir mmcblk0p1
mount -t ext3 /dev/$PARTITION /mnt/mmcblk0p1

cp -af /mnt/tmpmnt/* /mnt/mmcblk0p1/
```

```
umount /mnt/mmcblk0p1
umount /mnt/tmpmnt
```

7. Type sync to write the contents to MMC/SD.
8. Type poweroff to power down the system. Follow the instructions in Running Linux OS on the target to boot the image from the MMC/SD card.

NOTE

By default, v2013.04 and later versions of U-Boot support loading the kernel image and DTB file from the SD/MMC vfat partition by using the fatload command. To use this feature, perform the following steps:

1. Format the first partition (for example 50 MB) of the SD/MMC card with vfat filesystem.
2. Copy zImage and the DTB file into the VFAT partition after you mount the VFAT partition into your host computer.
3. Make sure that the zImage and DTB file name are synchronized with the file name pointed to by the U-Boot environment variables: fdt_file and image. Use the print command under U-Boot to display these two environment variables. For example:

   ```
   print fdt_file image
   ```
4. U-Boot loads the kernel image and the DTB file from your VFAT partition automatically when you boot from the SD/MMC card.

The following is an example to format the first partition to a 50 MB vfat filesystem and format the second partition to an ext4 filesystem:

```
~$ fdisk /dev/sdb

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-30318591, default 2048): 4096
Last sector, +sectors or +size{K,M,G} (4096-30318591, default 30318591): +50M

Command (m for help): p

Disk /dev/sdb: 15.5 GB, 15523119104 bytes
64 heads, 32 sectors/track, 14804 cylinders, total 30318592 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x3302445d

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            4096      106495       51200   83  Linux

Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First sector (2048-30318591, default 2048): 106496
Last sector, +sectors or +size{K,M,G} (106496-30318591, default 30318591):
Using default value 30318591

Command (m for help): p

Disk /dev/sdb: 15.5 GB, 15523119104 bytes
64 heads, 32 sectors/track, 14804 cylinders, total 30318592 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disk identifier: 0x3302445d

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            4096      106495       51200   83  Linux
/dev/sdb2          106496    30318591    15106048   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.


~$ mkfs.vfat /dev/mmcblk0p1
~$ mkfs.ext4 /dev/mmcblk0p2
```

4.5 How to boot the i.MX boards

When U-Boot is loaded onto one of the devices that support booting, the DIP switches can be used to boot from that device. The boot modes of the i.MX boards are controlled by the boot configuration DIP switches on the board. For help locating the boot configuration switches, see the quick start guide for the specific board as listed under References above.

The following sections list basic boot setup configurations. The tables below represent the DIP switch settings for the switch blocks on the specified boards. An X means that particular switch setting does not affect this action.

4.5.1 Booting from an SD card in slot SD1

4.5.2 Serial download mode for the Manufacturing Tool

Table 2. Setup for Manufacturing Tool on i.MX 8MQuad EVK

Switch	D1	D2
SW802	OFF	ON

Table 3. Setup for Manufacturing Tool on i.MX 8MMini EVK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW1101	ON	OFF	X	X	X	X	X	X
SW1102	X	X	X	X	X	X	X	X

NOTE

If the SD card with bootable image is plugged in SD2, because of the MFG boot on SD2, it will not enter the serial download mode.

4.5.3 How to build U-Boot and Kernel in standalone environment

To build U-Boot and Kernel in a standalone environment, perform the following steps:

First, generate an SDK, which includes the tools, toolchain, and small rootfs to compile against to put on the host machine.

- Generate an SDK from the Yocto Project build environment with the following command. To set up the Yocto Project build environment, follow the steps in the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG). In the following command, set Target-Machine to the machine you are building for. The populate_sdk generates a script file that sets up a standalone environment without Yocto Project. This SDK should be updated for each release to pick up the latest headers, toolchain, and tools from the current release.

  ```
  DISTRO=fsl-imx-fb MACHINE=Target-Machine bitbake core-image-minimal -c populate_sdk
  ```
- From the build directory, the bitbake was run in, copy the sh file in tmp/deploy/sdk to the host machine to build on and execute the script to install the SDK. The default location is in /opt but can be placed anywhere on the host machine.

On the host machine, these are the steps to build U-Boot and Kernel:

- For i.MX 8 builds on the host machine, set the environment with the following command before building.

  ```
  source /opt/fsl-imx-xwayland/4.9.88/environment-setup-aarch64-poky-linux
  export ARCH=arm64
  ```
- To build an i.MX 8 U-Boot in the standalone environment, find the configuration for the target boot. In the following example, i.MX 8QuadMax MEK board is the target and it runs on the Arm Cortex-A53 core by default.

  ```
  make clean
  make imx8qm_mek_defconfig
  make
  ```

  For i.MX 8QuadXPlus MEK board:

  ```
  make clean
  make imx8qxp_mek_defconfig
  make
  ```
- To build U-Boot for i.MX 8MQuad EVK:

  ```
  make clean
  make imx8mq_evk_defconfig
  make
  ```

NOTE

Users need to modify configurations for fused parts. For example, the i.MX 6UltraLite has four parts, G0, G1, G2, and G3.

The fused modules are as follows:

- G0: TSC ， ADC2, FLEXCAN1, FLEXCAN2, FREQ_MON, TEMP_MON, VOLT_MONLCDIF, CSI, ENET2, CAAM, USB_OTG2, SAI23, BEE, UART5678, PWM5678, ECSPI34, I2C34, GPT2, and EPIT2.
- G1: TSC, ADC2, FLEXCAN2, FREQ_MON, TEMP_MON, VOLT_MON, LCDIF, CSI, ENET2, and BEE.
- G2: FREQ_MON, TEMP_MON, VOLT_MON, and BEE.
- G3: No fused module.

U-Boot configuration changes:

G0:

```
/* #define CONFIG_VIDEO */
#define CONFIG_FEC_ENET_DEV 0
/* #define CONFIG_CMD_BEE */
#define CONFIG_USB_MAX_CONTROLLER_COUNT 1
```

G1:

```
/* #define CONFIG_VIDEO */
#define CONFIG_FEC_ENET_DEV 0
/* #define CONFIG_CMD_BEE */
```

G2:

```
/* #define CONFIG_CMD_BEE */
```

G3: No change.

4.5.4 How to build imx-boot image by using imx-mkimage

For i.MX 8QuadMax, to build imx-boot image by using imx-mkimage, perform the following steps:

1. Copy u-boot.bin from u-boot/u-boot.bin to imx-mkimage/iMX8QM/.
2. Copy scfw_tcm.bin from SCFW porting kit to imx-mkimage/iMX8QM/.
3. Copy bl31.bin from ARM Trusted Firmware (imx-atf) to imx-mkimage/iMX8QM/.
4. Run `make SOC=iMX8QM flash` to generate flash.bin.

For i.MX 8QuadXPlus, to build imx-boot image by using imx-mkimage, perform the following steps:

1. Copy u-boot.bin from u-boot/u-boot.bin to imx-mkimage/iMX8QX/.
2. Copy scfw_tcm.bin from SCFW porting kit to imx-mkimage/iMX8QX/.
3. Copy bl31.bin from ARM Trusted Firmware (imx-atf) to imx-mkimage/iMX8QX/.
4. Run `make SOC=iMX8QX flash to generate flash.bin`.

For i.MX 8MQuad EVK and i.MX 8MMini EVK, to build imx-boot image by using imx-mkimage, perform the following steps:

1. Copy and rename mkimage from u-boot/tools/mkimage to imx-mkimage/iMX8M/mkimage_uboot.
2. Copy u-boot-spl.bin from u-boot/spl/u-boot-spl.bin to imx-mkimage/iMX8M/.
3. Copy u-boot-nodtb.bin from u-boot/u-boot-nodtb.bin to imx-mkimage/iMX8M/.
4. Copy fsl-imx8mq-evk.dtb (for i.MX 8MQuad EVK) or fsl-imx8mm-evk (for i.MX 8MMini EVK) from u-boot/arch/arm/dts/ to imx-mkimage/iMX8M/.
5. (Only for i.MX 8MQuad EVK) Copy bl31.bin from ARM Trusted Firmware (imx-atf) to imx-mkimage/iMX8M/.
6. Copy firmware/hdmi/cadence/signed_hdmi_imx8m.bin from firmware-imx package to imx-mkimage/iMX8M/.
7. Copy lpddr4_pmu_train_1d_dmem.bin, lpddr4_pmu_train_1d_imem.bin, lpddr4_pmu_train_2d_dmem.bin, and lpddr4_pmu_train_2d_imem.bin from firmware/ddr/synopsys of firmware-imx package to imx-mkimage/iMX8M/.
8. For i.MX 8MQuad EVK, run `make SOC=iMX8M flash_evk` to generate flash.bin (imx-boot image) with HDMI FW included. For i.MX 8MMini EVK, run `make SOC=iMX8MM flash_evk` to generate flash.bin (imx-boot image).

4.6 Flash memory maps

This section describes the software layout in memory on memory devices used on the i.MX boards.

This information is useful for understanding subsequent sections about image downloading and how the images are placed in memory.

The mtdparts directive can be used in the Linux boot command to specify memory mapping. The following example briefly describes how to use memory maps. Memory is allocated in the order of how it is listed. The dash (-) indicates the the rest of the memory.

```
mtdparts=[memory type designator]:[size]([name of partition]),[size]([name of partition]),-
([name of final partition])
```

4.6.1 MMC/SD memory map

The MMC/SD memory scheme is different from the NAND and NOR flash, which are deployed in the BSP software. The MMC/SD/SATA must keep the first sector (512 bytes) as the Master Boot Record (MBR) to use MMC/SD as the rootfs.

Upon boot-up, the MBR is executed to look up the partition table to determine which partition to use for booting. The bootloader should be after the MBR. The kernel image and rootfs may be stored at any address after the bootloader. By default, the the U-Boot boot arguments uses the first FAT partition for kernel and DTB, and the following ext3 partition for the root file system. Alternatively, users can store the kernel and the DTB in any raw memory area after the bootloader. The boot arguments must be updated to match any changed memory addresses.

The MBR can be generated through the fdisk command when creating partitions in MMC/SD cards on a Linux host server.

4.7 Running Linux OS on the target

This section explains how to run a Linux image on the target using U-Boot.

These instructions assume that you have downloaded the kernel image using the instructions in Downloading images or Preparing an SD/MMC card to boot. If you have not set up your Serial Terminal, see Basic Terminal Setup.

The basic procedure for running Linux OS on an i.MX board is as follows. This document uses a specific set of environment variable names to make it easier to describe the settings. Each type of setting is described in its own section as follows.

1. Power on the board.
2. When U-Boot comes up, set the environment variables specific to your machine and configuration. Common settings are described below and settings specific to a device are described in separate sections.
3. Save the environment setup:

   ```
   U-Boot > saveenv
   ```
4. Run the boot command:

   ```
   U-Boot > run bootcmd
   ```

The commands `env default -f -a` and `saveenv` can be used to return to the default environment.

Specifying the console

The console for debug and command-line control can be specified on the Linux boot command line. The i.MX 6Quad SABRE-AI board uses ttymxc2, so it is not same for all boards. It is usually specified as follows, but the baudrate and the port can be modified. Therefore, for NFS, it might be `ttymxc3`.

```
U-Boot > setenv consoleinfo 'console=ttymxc2,115200'
```

For the i.MX 7ULP EVK, i.MX 8QuadMax MEK boards, and i.MX 8QuadXPlus MEK board, change to " console=ttyLP0,115200".

Specifying displays

The display information can be specified on the Linux boot command line. It is not dependent on the source of the Linux image. If nothing is specified for the display, the settings in the device tree are used. Add `${displayinfo}` to the environment macro containing `bootargs`. The specific parameters can be found in the *i.MX Linux® Release Notes* (IMXLXRN). The following are some examples of these parameters.

- `U-Boot > setenv displayinfo 'video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24'` for an HDMI display
- `U-Boot > setenv displayinfo 'video=mxcfb1:dev=ldb video=mxcfb0:dev=hdmi, 1920x1080M@60,if=RGB24'` for LVDS and HDMI dual displays

- `U-Boot > setenv displayinfo 'video=mxcfb0:dev=lcd,if=RGB565'` for an LCD
- `U-Boot > setenv displayinfo 'video=mxcepdcfb:E060SCM,bpp=16 max17135:pass=2,vcom=-2030000'` for an EPDC connection
- `U-Boot > setenv displayinfo 'video=mxcfb0:mxcfb0:dev=lcd,if=RGB565 video=mxcfb1:dev=hdmi,1920x1080M@60,if=RGB24'` for LCD and HDMI dual displays

Specifying memory addresses

The addresses in the memory where the kernel and device tree are loaded to do not change based on the device that runs Linux OS. The instructions in this chapter use the environment variables `loadaddr` and `ftd_addr` to indicate these values. The following table shows the addresses used on different i.MX boards.

Table 4. Board-specific default values

Variable	6Quad, 6QuadPlus, and 6DualLite SABRE (AI and SD)	6SoloLite EVK	6SoloX EVK	7Dual SABRE-SD	6UltraLite EVK and i.MX 6ULL EVK	7ULP EVK	8QuadMax and 8QuadXPlus	8MQuad EVK	Description
loadaddr	0x12000000	0x80800000	0x80800000	0x80800000	0x80800000	0x60800000	0x80280000	0x40480000	Address in the memory the kernel are loaded to
fdt_addr	0x18000000	0x83000000	0x83000000	0x83000000	0x83000000	0x63000000	0x83000000	0x43000000	Address in the memory the device tree code are copied to

In addition, `fdt_file` is used to specify the filename of the device tree file. The commands used to set the U-Boot environment variables are as follows:

```
U-Boot > setenv loadaddr 0x80080000
U-Boot > setenv fdtaddr 0x80f00000
U-Boot > setenv fdt_file fsl-imx7ulp-evk.dtb
```

Specifying the location of the root file system

The rootfs can be located on a device on the board or on NFS. The settings below show some options for specifying these.

- `U-Boot > setenv rootfsinfo 'root=/dev/nfs ip=dhcp nfsroot=${serverip}:${nfsroot},v3,tcp'`
- `U-Boot > setenv rootfsinfo 'root=/dev/nfs ip=dhcp weim-nor nfsroot=${serverip}:${nfsroot},v3,tcp'`
- `U-Boot > setenv rootfsinfo 'ubi.mtd=4 root=ubi0:rootfs rootfstype=ubifs rootwait rw mtdparts=gpmi-nand:64m(boot),16m(kernel),16m(dtb),-(rootfs)'`
- `U-Boot > setenv rootfsinfo 'root=/dev/mmcblk0p2 rootwait rw'`

Special settings

SoloLite, Solo, and UltraLite can specify `uart_from_osc` on the command line to specify that the OSC clock rather than PLL3 should be used. This allows the system to enter low power mode.

```
U-Boot > setenv special 'uart_from_osc'
```

Creating the boot command line

For clarification, this document groups the bootargs into one macro as follows:

```
U-Boot > setenv bootargsset 'setenv bootargs ${consoleinfo} ${rootfsinfo} ${displayinfo} $
{special}'
```

The executed boot command is then as follows. Arguments vary by device.

```
U-Boot > setenv bootcmd 'run bootargsset; {settings-for-device}; bootz ${loadaddr} - $
{fdt_addr}'
```

4.7.1 Running Linux OS from MMC/SD

This scenario assumes that the board is configured to boot U-Boot, that the Linux kernel image is named zImage and is stored on the SD card in an MSDOS FAT partition, and one or more device tree files are also stored in this partition. The rootfs is also stored on the SD/MMC card in another partition.

When U-Boot boots up, it detects the slot where it is booting from and automatically sets `mmcdev` and `mmcroot` to use the rootfs on that SD card. In this scenario, the same SD card can be used to boot from any SD card slot on an i.MX 6/7 board, without changing any U-Boot settings. From the U-Boot command line, type `boot` to run Linux OS.

The following instructions can be used if the default settings are not desired.

Set `mmcautodetect` to "no" to turn off the automatic setting of the SD card slot in `mmcdev` and `mmcroot`. The U-Boot `mmcdev` is based on the soldered SD/MMC connections, so it varies depending on the board. The U-Boot mmc dev 0 is the lowest numbered SD slot present, 1 is the next, and so on. The Linux kernel, though, indexes all the uSDHC controllers whether they are present or not. The following table shows this mapping.

Table 5. Linux uSDHC relationships

uSDHC	mmcroot
uSDHC 1	mmcblk0*
uSDHC 2	mmcblk1*
uSDHC 3	mmcblk2*
uSDHC 4	mmcblk3*

In the default configuration of the SD card and the example here, U-Boot is at the 1024 byte offset or 33 KB offset for i.MX 8 before the first partition, partition 1 is the partition with the Linux kernel and device trees, and partition 2 is the rootfs.

Setting up the environment variables

For convenience, this document uses a standard set of variables to describe the information in the Linux command line. The values used here may be different for different machines or configurations. By default, U-Boot supports setting `mmcdev` and `mmcroot` automatically based on the uSDHC slot that we boot from. This assumes zImage, the device tree file (DTB), and the rootfs are on the same SD/MMC card. To set these environment variables manually, set `mmcautodetect` to `no` to disable the feature.

The following is one way to set up the items needed to boot Linux OS.

```
U-Boot > setenv mmcpart 1
U-Boot > setenv loadfdt 'fatload mmc ${mmcdev}:${mmcpart} ${fdt_addr} ${fdt_file}'
U-Boot > setenv loadkernel 'fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} zImage'
U-Boot > setenv bootcmd 'mmc dev ${mmcdev}; run loadkernel; run mmcargs; run loadfdt; bootz $
{loadaddr} - ${fdt_addr};'
```

The descriptions of the variables used above are as follows:

- `mmcpart` - This is the partition on the MMC/SD card containing the kernel image.
- `mmcroot` - The location of the root file system on the MMC SD card along with directives for the boot command for the rootfs.

NOTE

The U-Boot environment on the pre-built SD card does not match this. It is more complex so that it can automatically deal with more variations. The example above is designed to be easier to understand and use manually.

Reading the kernel image from eMMC

eMMC has user area, boot partition 1, and boot partition 2. To switch between the eMMC partitions, the user needs to use the command `mmc dev [dev id] [partition id]`. For example,

```
mmc dev 2 0 ---> user area
mmc dev 2 1 ---> boot partition 1
mmc dev 2 2 ---> boot partition 2
```

4.7.2 Running the Linux image from NFS

To boot from NFS, set the following environment variables at the U-Boot prompt:

```
U-Boot > setenv serverip 10.192.225.216
U-Boot > setenv image <your kernel zImage name on the TFTP server>
U-Boot > setenv fdt_file <your dtb image name on the TFTP server>
U-Boot > setenv rootfsinfo 'setenv bootargs ${bootargs} root=/dev/nfs ip=dhcp \
               nfsroot=${serverip}:/data/rootfs_home/rootfs_mx6,v3,tcp'
U-Boot > setenv bootcmd_net 'run rootfsinfo; dhcp ${image}; dhcp ${fdt_addr} \
               ${fdt_file}; booti ${loadaddr} - ${fdt_addr}'
U-Boot > setenv bootcmd 'run bootcmd_net'
```

NOTE

If the MAC address has not been burned into the fuses, set the MAC address to use the network in U-Boot.

```
setenv ethaddr xx:xx:xx:xx:xx:xx
```

4.7.3 Linux OS login

The default login username for the i.MX Linux OS is `root` with no password.

5 Multimedia

i.MX provides audio optimized software codecs, parsers, hardware acceleration units, and associated plugins. The i.MX provides GStreamer plugins to access the i.MX multimedia libraries and hardware acceleration units. This chapter provides various multimedia use cases with GStreamer command line examples.

5.1 i.MX multimedia packages

Due to license limitations, i.MX multimedia packages can be found in two locations:

- Standard packages: provided on the NXP mirror.
- Limited access packages: provided on nxp.com with controlled access.

For details, see the *i.MX Release Notes* (IMXLXRN).

5.2 Building limited access packages

Place the limited access package in the `downloads` directory and read the `readme` file in each package.

For example, README-microsoft in the package `imxcodec-microsoft-$version.tar.gz`.

5.3 Multimedia use cases

GStreamer is the default multimedia framework on Linux OS. The following sections provide examples of GStreamer commands to perform the specific functions indicated. The following table shows how this document refers to common functions and what the actual command is.

Table 6. Command mapping

Variable	$GSTL	$PLAYBIN	$GPLAY	$GSTINSPECT
GStreamer 1.x	gst-launch-1.0	playbin	gplay-1.0	gst-inspect-1.0

One option is to set these as environment variables as shown in the following examples. Use the full path to the command on your system.

```
export GSTL=gst-launch-1.0
export PLAYBIN=playbin
export GPLAY=gplay-1.0
export GSTINSPECT=gst-inspect-1.0
```

In this document, variables are often used to describe the command parameters that have multiple options. These variables are of the format $description where the type of values that can be used are described. The possible options can be found in the Section about Multimedia in the *i.MX Linux® Release Notes* (IMXLXRN) for i.MX-specific options, or at "gstreamer.freedesktop.org/ for the community options.

The GStreamer command line pipes the input through various plugins. Each plugin section of the command line is marked by an exclamation mark (!). Each plugin can have arguments of its own that appear on the command line after the plugin name and before the next exclamation mark (!). Use $GSTINSPECT $plugin to get information on a plugin and what arguments it can use.

Square brackets ([ ]) indicate optional parts of the command line.

5.3.1 Playback use cases

Playback use cases include the following:

- Audio-only playback
- Video-only playback
- Audio/Video file playback
- Other methods for playback

5.3.1.1 Audio-only playback

An audio-only playback command uses this format:

```
$GSTL filesrc location=$clip_name [typefind=true] ! [$id3parse] ! queue !
$audio_parser_plugins
     ! $audio_decoder_plugin ! $audio_sink_plugin
```

If the file to be played contains an ID3 header, use the ID3 parser. If the file does not have an ID3 header, this has no effect.

This example plays an MP3 file in the audio jack output.

```
$GSTL filesrc location=test.mp3 ! id3demux ! queue ! mpegaudioparse ! beepdec ! alsasink
```

This example plays an MP3 file in the HDMI audio output.

```
GSTL filesrc location=test.mp3 ! id3demux ! queue ! mpegaudioparse ! beepdec ! alsasink
device=sysdefault:CARD=imxaudiohdmi
```

5.3.1.2 Video-only playback

```
$GSTL filesrc location=test.video typefind=true
     ! $capsfilter ! $demuxer_plugin ! queue max-size-time=0
     ! $video_decoder_plugin ! $video_sink_plugin
```

This is an example of an MP4 video file playback:

```
$GSTL filesrc location=test.mp4 typefind=true
     ! video/quicktime ! aiurdemux ! queue max-size-time=0
     ! avdec_h264 ! autovideosink
```

5.3.1.3 Audio/Video file playback

This is an example of a command to play a video file with audio:

```
$GSTL filesrc location=test_file typefind=true ! $capsfilter
     ! $demuxer_plugin name=demux demux.
     ! queue max-size-buffers=0 max-size-time=0 ! $video_decoder_plugin
     ! $video_sink_plugin demux.
     ! queue max-size-buffers=0 max-size-time=0 ! $audio_decoder_plugin
     ! $audio_sink_plugin
```

This is an example of an AVI file:

```
$GSTL filesrc location=test.avi typefind=true ! video/x-msvideo
     ! aiurdemux name=demux demux.
     ! queue max-size-buffers=0 max-size-time=0 ! avdec_h264
     ! autovideosink demux.
     ! queue max-size-buffers=0 max-size-time=0 ! beepdec
     ! alsasink
```

For the plattforms without VPU hardware, $video_deocder_plugin could be a software decoder plugin like avdec_h264.

5.3.1.4 Multichannel audio playback

For the multichannel audio playback settings to be used when pulseaudio is enabled, see Pulseaudio input/output settings.

5.3.1.5 Other methods for playback

Use the $PLAYBIN plugin or the i.MX $GPLAY command line player for media file playback.

```
$GSTL $PLAYBIN uri=file:///mnt/sdcard/test.avi
$GPLAY /mnt/sdcard/test.avi
```

5.3.1.6 Video playback to multiple displays

Video playback to multiple displays can be supported by a video sink plugin. The video sink for multidisplay mode does not work on i.MX 8 family SoCs.

This use case requires that the system boots in multiple-display mode (dual/triple/four, the number of displays supported is determined by the SOC and the BSP). For how to configure the system to boot in this mode, see the *i.MX BSP Porting Guide* (IMXBSPPG).

To configure the video sink plugin for multidisplay mode, see Overlaysink usage later in this document.

5.3.1.6.1 Playing different videos on different displays

The command line to play two videos on different displays might look like this:

```
$GSTL $PLAYBIN uri=file:///$file1 playbin uri=file:///$file2 video-sink="overlaysink
    display-master=false display-slave=true"
```

5.3.1.6.2 Routing the same video to different displays

A video can be displayed on multiple displays with a command as follows:

```
$GSTL $PLAYBIN uri=file:///$filename video-sink="overlaysink display-slave=true"
```

5.3.1.6.3 Multiple videos overlay

The `overlaysink` plugin provides support for compositing multiple videos together and rendering them to the same display. The result might look like the following image.

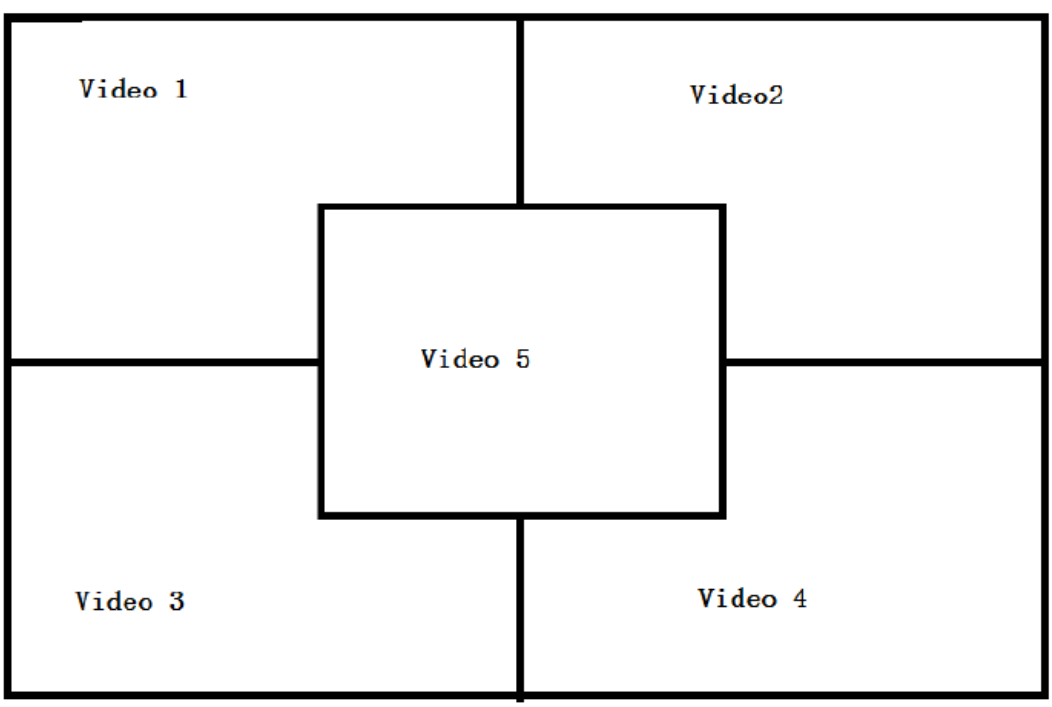

Figure 3. Multiple video overlay

```
gst-launch-1.0 playbin uri=file://$FILE1
    video-sink="overlaysink overlay-width=512 overlay-height=384"
    playbin uri=file://$FILE2 flags=0x41
    video-sink="overlaysink overlay-left=512 overlay-width=512 overlay-height=384"
    playbin uri=file://$FILE3 flags=0x41
    video-sink="overlaysink overlay-top=384 overlay-width=512 overlay-height=384"
    playbin uri=file://$FILE4 flags=0x41
    video-sink="overlaysink overlay-left=512 overlay-top=384 overlay-width=512
    overlay-height=384"
    playbin uri=file://$FILE5 flags=0x41
    video-sink="overlaysink overlay-left=352 overlay-top=264 overlay-width=320
    overlay-height=240 zorder=1"
```

5.3.2 Audio encoding

Here are some examples for MP3 encoding.

```
$GSTL filesrc location=test.wav ! wavparse ! avenc_mp2
     ! filesink location=output.mp3
```

5.3.3 Audio recording

The following examples show how to make an MP3 or WMA audio recording.

- MP3 recording

  ```
  $GSTL pulsesrc num-buffers=$NUMBER blocksize=$SIZE ! avenc_mp2
       ! filesink location=output.mp3
  ```

  NOTE

  The recording duration is calculated as $NUMBER * $SIZE * 8 / (samplerate * channel * bit width).

  Therefore, to record 10 seconds of a stereo channel sample with a 44.1K sample rate and a 16 bit width, use the following command:

  ```
  $GSTL alsasrc num-buffers=430 blocksize=4096 ! avenc_mp2
       ! filesink location=output.mp3
  ```

5.3.4 Camera preview

This example displays what the camera sees. It is only available on platforms with a camera.

```
$GSTL v4l2src ! 'video/x-raw, format=(string)$FORMAT,
     width=$WIDTH, height=$HEIGHT, framerate=(fraction)30/1'
     ! v4l2sink
```

Camera preview example:

```
$GSTL v4l2src device=/dev/video1 ! 'video/x-raw,
     format=(string)UYVY,width=640,height=480,framerate=(fraction)30/1'
     ! autovideosink
```

Parameter comments:

- Get the camera support format and resolution using gst-inspect-1.0 v4l2src.

- Set caps filter according to the camera's supported capabilities if the user needs other format or resolution.
- Ensure that the right caps filter has been set, which also needs to be supported by v4l2sink.

5.3.5 HTTP streaming

The HTTP streaming includes the following:

- Manual pipeline

```
$GSTL souphttpsrc location= http://SERVER/test.avi ! typefind
     ! aiurdemux name=demux demux. ! queue max-size-buffers=0 max-size-time=0
     ! vpudec ! $video_sink_plugin demux. ! queue max-size-buffers=0 max-size-time=0
     ! beepdec ! $audio_sink_plugin
```

- PLAYBIN

```
$GSTL $PLAYBIN uri=http://SERVER/test.avi
```

- GPLAY

```
$GPLAY http://SERVER/test.avi
```

5.3.6 Real Time Streaming Protocol (RTSP) playback

Use the following command to see the GStreamer RTP depacketize plugins:

```
$GSTINSPECT | grep depay
```

RTSP streams can be played with a manual pipeline or by using playbin. The format of the commands is as follows.

- Manual pipeline

```
$GSTL rtspsrc location=$RTSP_URI name=source
     ! queue ! $video_rtp_depacketize_plugin ! $vpu_dec ! $video_sink_plugin source.
     ! queue ! $audio_rtp_depacketize_plugin ! $audio_parse_plugin ! beepdec !
$audio_sink_plugin
```

- PLAYBIN

```
$GSTL $PLAYBIN uri=$RTSP_URI
```

Two properties of `rtspsrc` that are useful for RTSP streaming are:

- **Latency**: This is the extra added latency of the pipeline, with the default value of 200 ms. If you need low-latency RTSP streaming playback, set this property to a smaller value.
- **Buffer-mode**: This property is used to control the buffering algorithm in use. It includes four modes:
  - None: Outgoing timestamps are calculated directly from the RTP timestamps, not good for real-time applications.
  - Slave: Calculates the skew between the sender and receiver and produces smoothed adjusted outgoing timestamps, good for low latency communications.
  - Buffer: Buffer packets between low and high watermarks, good for streaming communication.
  - Auto: Chooses the three modes above depending on the stream. This is the default setting.

To pause or resume the RTSP streaming playback, use a buffer-mode of slave or none for `rtspsrc`, as in `buffer-mode=buffer`. After resuming, the timestamp is forced to start from 0, and this causes buffers to be dropped after resuming.

Manual pipeline example:

```
$GSTL rtspsrc location=rtsp://10.192.241.11:8554/test name=source
     ! queue ! rtph264depay ! avdec_h264 ! overlaysink source.
     ! queue ! rtpmp4gdepay ! aacparse ! beepdec ! pulsesink
```

Playback does not exit automatically in GStreamer 1.x, if `buffer-mode` is set to `buffer` in the `rtpsrc` plugin.

5.3.7 RTP/UDP MPEGTS streaming

There are some points to keep in mind when doing RTP/UDP MPEGTS Streaming:

- The source file that the UDP/RTP server sends must be in TS format.
- Start the server one second earlier than the time client starts.
- Two properties of `aiurdemux` that are useful for UDP/RTP TS streaming are:

  streaming-latency: This is the extra added latency of the pipeline, and the default value is 400 ms. This value is designed for the situation when the client starts first. If the value is too small, the whole pipeline may not run due to lack of audio or video buffers. In that situation, you should cancel the current command and restart the pipeline. If the value is too large, wait for a long time to see the video after starting the server.

  low_latency_tolerance: This value is a range that total latency can jitter around streaming-latency. This property is disabled by default. When the user sets this value, the maximum latency is (streaming-latency + low_latency_tolerance).

The UDP MPEGTS streaming command line format looks like this:

```
$GSTL udpsrc do-timestamp=false uri=$UDP_URI caps="video/mpegts"
     ! aiurdemux streaming_latency=400 name=d d. ! queue ! $vpu_dec
     !   queue ! $video_render_sink sync=true d. ! queue ! beepdec ! $audio_sink_plugin
sync=true

$GSTL udpsrc do-timestamp=false uri=udp://10.192.241.255:10000 caps="video/mpegts"
     ! aiurdemux streaming_latency=400 name=d d. ! queue ! vpudec
     ! queue ! overlaysink sync=true d. ! queue ! beepdec ! pulsesink sync=true
```

The format for a RTP MPEGTS streaming command is covered as follows:

```
$GSTL udpsrc do-timestamp=false uri=$RTP_URI caps="application/x-rtp"
     ! rtpmp2tdepay ! aiurdemux streaming_latency=400 name=d d. ! queue ! $vpu_dec
     ! queue ! $video_render_sink sync=true d. ! queue ! beepdec ! $audio_sink_plugin
sync=true

$GSTL udpsrc do-timestamp=false uri=udp://10.192.241.255:10000 caps="application/x-rtp"
     ! rtpmp2tdepay ! aiurdemux streaming_latency=400 name=d d.
     ! queue ! vpudec ! queue ! overlaysink sync=true d. ! queue ! beepdec
     ! pulsesink sync=true
```

5.3.8 Video conversion

There are three video conversion plugins, imxvideoconvert_ipu, imxvideoconvert_g2d, and imxvideoconvert_pxp. All of them can be used to perform video color space conversion, resize, and rotate. imxvideoconvert_ipu can also be used to perform video deinterlacing. They can be used to connect before ximagesink to enable the video rendering on X Windows or used in transcoding to change video size, rotation, or deinterlacing.

Use gst-inspect-1.0 to get each convertor's capability and supported input and output formats. Note that imxvideoconvert_g2d can only perform color space converting to RGB space.

Color Space Conversion (CSC)

```
gst-launch-1.0 videotestsrc ! video/x-raw,format=NV12 ! imxvideoconvert_{xxx} ! video/x-
raw,format=RGB16 ! ximagesink display=:0
```

Resize

```
gst-launch-1.0 videotestsrc ! video/x-raw,format=NV12,width=800,height=600 !
imxvideoconvert_{xxx} ! video/x-raw, width=640, height=480 ! ximagesink display=:0
```

Rotate

```
gst-launch-1.0 videotestsrc ! imxvideoconvert_{xxx} rotation=2 ! ximagesink display=:0
```

5.3.9 Video composition

imxcompositor_g2d uses corresponding hardware to accelerate video composition. It can be used to composite multiple videos into one. The video position, size, and rotation can be specified while composition. Video color space conversion is also performed automatically if input and output video are not same. Each video can be set to an alpha and z-order value to get alpha blending and video blending sequence.

Note that imxcompositor_g2d can only output RGB color space format. Use gst-inspect-1.0 to get more detailed information, including the supported input and output video format.

- Composite two videos into one.

  ```
  gst-launch-1.0 imxcompositor_{xxx} name=comp sink_1::xpos=160 sink_1::ypos=120 !
  overlaysink videotestsrc ! comp.sink_0 videotestsrc ! comp.sink_1
  ```
- Composite two videos into one with red background color.

  ```
  gst-launch-1.0 imxcompositor_{xxx} background=0x000000FF name=comp sink_1::xpos=160
  sink_1::ypos=120 ! overlaysink videotestsrc ! comp.sink_0 videotestsrc ! comp.sink_1
  ```
- Composite two videos into one with CSC, resize, and rotate.

  ```
  gst-launch-1.0 imxcompositor_{xxx} name=comp sink_0::width=640 sink_0::height=480
        sink_1::xpos=160 sink_1::ypos=120 sink_1::width=640 sink_1::height=480
  sink_1::rotate=1 !
        video/x-raw,format=RGB16 ! overlaysink videotestsrc !
        video/x-raw,format=NV12,width=320,height=240 ! comp.sink_0 videotestsrc !
        video/x-raw,format=I420,width=320,height=240 ! comp.sink_1
  ```
- Composite three videos into one with CSC, resize, rotate, alpha, z-order, and keep aspect ratio.

  ```
  gst-launch-1.0 imxcompositor_{xxx} name=comp sink_0::width=640 sink_0::height=480
        sink_0::alpha=0.5 sink_0::z-order=3 sink_1::alpha=0.8 sink_1::z-order=2
  sink_1::xpos=160
        sink_1::ypos=120 sink_1::width=640 sink_1::height=480 sink_1::rotate=1
  sink_2::xpos=320
        sink_2::ypos=240 sink_2::width=500 sink_2::height=500 sink_2::alpha=0.6
        sink_2::keep-ratio=true ! video/x-raw,format=RGB16 ! overlaysink videotestsrc !
        video/x-raw,format=NV12,width=320,height=240 ! comp.sink_0 videotestsrc !
        video/x-raw,format=I420,width=320,height=240 ! comp.sink_1 videotestsrc !
        video/x-raw,format=RGB16,width=320,height=240 ! comp.sink_2
  ```

5.4 Pulseaudio input/output settings

If pulseaudio is installed in the rootfs, the pulseaudio input/output settings may need to be set. Pulse audio is only available for the X11 back-end Yocto Project rootfs.

Audio output settings

Use the `pactl` command to list all the available audio sinks:

```
$ pactl list sinks
```

A list of available audio sinks are displayed:

```
Sink #0
        State: SUSPENDED
        Name: alsa_output.platform-soc-audio.1.analog-stereo
        Description: sgtl5000-audio Analog Stereo
   ...
   ...
Sink #1
        State: SUSPENDED
        Name: alsa_output.platform-soc-audio.4.analog-stereo
        Description: imx-hdmi-soc Analog Stereo
   ...
   ...
```

Use the `pacmd` command to set the default audio sink according to the sink number in the list shown above:

```
$ pacmd set-default-sink $sink-number
```

$sink-number could be 0 or 1 in the example above.

After setting the default sink, use the command below to verify the audio path:

```
$ gst-launch audiotestsrc ! pulsesink
```

Audio input settings

Use the `pactl` command to list all the available audio sources:

```
$ pactl list sources
```

A list of available audio sources are displayed:

```
Source #0
          State: SUSPENDED
          Name: alsa_output.platform-soc-audio.1.analog-stereo.monitor
          Description: Monitor of sgtl5000-audio Analog Stereo
   ...
   ...

Source #1
          State: SUSPENDED
          Name: alsa_input.platform-soc-audio.1.analog-stereo
          Description: sgtl5000-audio Analog Stereo ...
   ...
   ...
```

Use the `pacmd` command to set the default audio source according to the source number in the list shown above:

```
$ pacmd set-default-source $sink-number
```

$sink-number could be 0 or 1 in the example above. If record and playback at the same time is not needed, there is no need to set the monitor mode.

The pulseaudio I/O path setting status can be checked with:

```
$ pactl stat
```

Multichannel output support settings

For those boards that need to output multiple channels, these are the steps needed to enable the multichannel output profile:

1. Use the `pacmd` command to list the available cards:

   ```
   $ pacmd list-cards
   ```

   The available sound cards and the profiles supported are listed.

   ```
   2 card(s) available.
       index: 0
           name: <alsa_card.platform-sound-cs42888.34>
   ```

```
        driver: <module-alsa-card.c>
        owner module: 6
        properties:
                alsa.card = "0"
                alsa.card_name = "cs42888-audio"
        ...
        ...
        profiles:
                input:analog-mono: Analog Mono Input (priority 1, available: unknown)
                input:analog-stereo: Analog Stereo Input (priority 60, available:
unknown)
        ...
        ...
    active profile: <output:analog-stereo+input:analog-stereo>
        ...
        ...
```

2. Use the `pacmd` command to set the profile for particular features.

   ```
   $ pacmd set-card-profile $CARD $PROFILE
   ```

   `$CARD` is the card name listed by `pacmd list-cards` (for example, alsa_card.platform-sound-cs42888.34 in the example above), and `$PROFILE` is the profile name. These are also listed by `pamcd list-cards`. (for example, output:analog-surround-51 in the example above).
3. After setting the card profile, use `$ pactl list sinks` and `$pacmd set-default-sink $sink-number` to set the default sink.

5.5 Installing gstreamer1.0-libav into rootfs

The following steps show how to install `gstreamer1.0-libav` into a rootfs image.

1. Add the following lines into the configuration file `conf/local.conf`.

   ```
   IMAGE_INSTALL_append = " gstreamer1.0-libav"
   LICENSE_FLAGS_WHITELIST = "commercial"
   ```
2. Build gstreamer1.0-libav.

   ```
   $ bitbake gstreamer1.0-libav
   ```
3. Build the rootfs image.

   ```
   $ bitbake
   $ <image_name>
   ```

6 Graphics

There are a number of graphics tools, tests, and example programs that are built and installed in the Linux rootfs. There are some variation on what is included based on the build and packages selected, the board, and the backend specified. This section describes some of them.

The kernel module version of graphics used on the system can be found by running the following command on the board:

```
dmesg | grep Galcore
```

The user-side GPU drivers version of graphics can be displayed using the following command on the board:

```
grep VERSION /usr/lib/libGAL*
```

When reporting problems with graphics, this version number is needed.

6.1 imx-gpu-sdk

This graphics package contains source for several graphics examples for OpenGLES 2.0 and OpenGLES 3.0 APIs for X11, Framebuffer, and XWayland graphical backends. These applications show that the graphics acceleration is working for different APIs. The package includes samples, demo code, and documentation for working with the i.MX family of graphic cores. More details about this SDK are in the *i.MX Graphics User's Guide*. This SDK is only supported for hardware that has OpenGLES hardware acceleration.

6.2 G2D-imx-samples

The G2D Application Programming Interface (API) is designed to make it easy to use and understand the 2D BLT functions. It allows the user to implement customized applications with simple interfaces. It is hardware and platform independent when using 2D graphics.

The G2D API supports the following features and more:

- Simple BLT operation from source to destination
- Alpha blend for source and destination with Porter-Duff rules
- High-performance memory copy from source to destination
- Up-scaling and down-scaling from source to destination
- 90/180/270 degree rotation from source to destination
- Horizontal and vertical flip from source to destination
- Enhanced visual quality with dither for pixel precision-loss
- High performance memory clear for destination
- Pixel-level cropping for source surface
- Global alpha blend for source only
- Asynchronous mode and synchronization
- Contiguous memory allocator
- VG engine

The G2D API document includes the detailed interface description and sample code for reference. The API is designed with C-Style code and can be used in both C and C++ applications.

The G2D is supported on all i.MX. The hardware that supports G2D is described below. For more details look at the i.MX Release Notes in the Frame Buffer to see which hardware is used for G2D.

- For i.MX 6 with GPU, the G2D uses the 2D GPU.
- For i.MX with PXP, the G2D uses the PXP with limited G2D features.

The following is the directory structure for the G2D test applications.

- `g2d`
  - g2d_test
  - Overlay Test
    - g2d_overlay_test

6.3 viv_samples

The directory `viv_samples` is found under `/opt`. It contains binary samples for OpenGL ES 1.1/2.0 and OpenVG 1.1.

The following are the basic sanity tests, which help to make sure that the system is configured correctly.

- cl11: This contains unit tests and FFT samples for OpenCL 1.1 Embedded Profile. OpenCL is implemented on the i.MX 6Quad, i.MX 6QuadPlus, and i.MX 8 boards.
- es20: This contains tests for Open GLES 2.0.
  - vv_launcher
    - coverflow.sh
    - vv_launcher
- tiger: A simple OpenVG application with a rotating tiger head. This is to demonstrate OpenVG.
- vdk: Contains sanity tests for OpenGL ES 1.1 and OpenGL ES 2.0.

The tiger and VDK tests show that hardware acceleration is being used. They will not run without it.

- UnitTest
  - clinfo
  - loadstore
  - math
  - threadwalker
  - test_vivante
    - functions_and_kernels
    - illegal_vector_sizes
    - initializers
    - multi_dimensional_arrays
    - reserved_data_types
    - structs_and_enums
    - unions
    - unsupported_extensions
  - fft

6.4 Qt 5

Qt 5 is built into the Linux image in the Yocto Project environment with the command `bitkake fsl-image-qt5`.

To run the Qt 5 examples on the board, the platform and input plugin need to be specified. Different backends require different graphics plugins, as shown in the following table.

Table 7. Graphics plugins for backends

Backend	Graphics
FB	eglfs
XWayland	wayland-egl
X11	xcb

It is often useful to specifically set the display variable and allow access to the X server. The commands below perform this operation. Check the xhost man page for additional ways to use that command. These commands often fix the problem that causes the "Could not connect to display" error message.

```
export DISPLAY=:0.0
xhost +
```

The command lines for some of the Qt 5 sample applications are as follows. For XWayland, it sometimes helps to add `--fullscreen` to the command line. The Qt 5 desktop may contain links to these examples.

- `Qt5_CinematicExperience -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0`
- `/usr/share/qt5nmapcarousedemo-1.0/Qt5_NMap_CarouselDemo -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0`

- `/usr/bin/qt5/qmlscene -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0 /usr/share/qt5ledscreen-1.0/example_hello.qml`
- `/usr/bin/qt5/qmlscene -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0 /usr/share/qt5ledscreen-1.0/example_billboard.qml`
- `/usr/bin/qt5/qmlscene -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0 /usr/share/qt5ledscreen-1.0/example_combo.qml`

Some examples must be run from a particular directory. The first column in the following table shows the directory and the second column shows the command to run in that directory. The examples are usually installed in `/usr/share`.

Table 8. Example directories and command lines

Directory	Command
qt5everywheredemo-1.0	./QtDemo -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0
qtpatientcare-1.0	./patientcare -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0
qtsmarthome-1.0	./smarthome -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0
quitbattery-1.0.0	./QUItBattery -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0

7 Security

Using the i.MX CryptoDev security driver causes the system to run much faster than without it.

The CAAM drivers are accelerated through the CryptoDev interface. The openssl command can be used to show the system speed without CryptoDev .

```
openssl speed -evp aes-128-cbc -engine cryptodev
```

An example of the key portion of the output is as follows. Library load errors may occur but they can be ignored.

```
Doing aes-128-cbc for 3s on 16 size blocks: 4177732 aes-128-cbc's in 2.99s
Doing aes-128-cbc for 3s on 64 size blocks: 1149097 aes-128-cbc's in 3.01s
Doing aes-128-cbc for 3s on 256 size blocks: 297714 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 75118 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks: 9414 aes-128-cbc's in 3.00s
```

Start CryptoDev and run the openssl command again. This time you should be able to see that the timeing values show the accelerated values. As the block sizes increase, the elapsed time decreases.

```
modprobe cryptodev
openssl speed -evp aes-128-cbc -engine cryptodev
```

Here is an example of the accelerated output:

```
Doing aes-128-cbc for 3s on 16 size blocks: 36915 aes-128-cbc's in 0.10s
Doing aes-128-cbc for 3s on 64 size blocks: 34651 aes-128-cbc's in 0.05s
Doing aes-128-cbc for 3s on 256 size blocks: 25926 aes-128-cbc's in 0.10s
Doing aes-128-cbc for 3s on 1024 size blocks: 20274 aes-128-cbc's in 0.04s
Doing aes-128-cbc for 3s on 8192 size blocks: 5656 aes-128-cbc's in 0.02s
```

8 Connectivity

This section describes the connectivity for Bluetooth wireless technology and Wi-Fi, as well as for USB type-C.

8.1 Connectivity for Bluetooth wireless technology and Wi-Fi

The officially supported Wi-Fi chip with our BSP is Murata 1PJ module based on Qualcomm QCA9377-3.

The QCA9377-3 is a single-die wireless local area network (WLAN) and Bluetooth (BT) combination solution to support 1 x 1 IEEE 802.11a/b/g/n/ac WLAN standards and BT 4.1 + HS, enabling seamless integration of LAN/BT and low-energy technology.

- Wi-Fi driver

  The i.MX release uses Qualcomm QCA9377 LEA-2.0 Wi-Fi driver to support Murata 1PJ, which can support SDIO3.0.

- Firmware

  The NXP release package already includes Wi-Fi firmware, but it requires to accept NXP license.

The wireless driver supports wpa_supplicant, which is a WEP/WPA/WPA2 encryption authenticated tool.

To run Wi-Fi, execute the following commands:

```
wpa_passphrase  SSID SSID_PASSWD >> /etc/wpa_supplicant.conf
wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant.conf -D nl80211
udhcpc -i wlan0
```

8.2 Connectivity for USB type-C

The following describes the connectivity for USB type-C and power delivery connection on the i.MX 8QuadXPlus MEK board.

- The Linux release includes USB type-C and PD stack, which is enabled by default. The specific power parameters are passed in by DTS. The following fsl-imx8qxp-mek is an example:

```
typec_ptn5110: typec@50 {
        compatible = "usb,tcpci";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_typec>;
        reg = <0x50>;
        interrupt-parent = <&gpio1>;
        interrupts = <3 IRQ_TYPE_LEVEL_LOW>;
        ss-sel-gpios = <&gpio5 9 GPIO_ACTIVE_LOW>;
        reset-gpios = <&pca9557_a 7 GPIO_ACTIVE_HIGH>;
        src-pdos = <0x380190c8>;
        snk-pdos = <0x380190c8 0x3802d0c8>;
        max-snk-mv = <9000>;
        max-snk-ma = <1000>;
        op-snk-mw = <9000>;
        port-type = "drp";
        sink-disable;
        default-role = "source";
        status = "okay";
};
```

  For power capability related configuration, users need to check the PD specification to see how to composite the PDO value. To make it support power source role for more voltages, specify the source PDO. The i.MX 8QuadXPlus board can support 5 V and 12 V power supply.

- The Linux BSP of the Alpha and Beta releases on the i.MX 8QuadXPlus MEK platform only supports power source role for 5 V.

- Users can use /sys/kernel/debug/tcpm/2-0050 to check the power delivery state, which is for debugging purpose information.
- Booting only by type-C port power supply is not supported on the Alpha release.

9 Revision History

This table provides the revision history.

Table 9. Revision history

Revision number	Date	Substantive changes
L4.9.51_imx8qxp-alpha	11/2017	Initial release
L4.9.51_imx8qm-beta1	12/2017	Added i.MX 8QuadMax
L4.9.51_imx8mq-beta	12/2017	Added i.MX 8MQuad
L4.9.51_8qm-beta2/8qxp-beta	02/2018	Added i.MX 8QuadMax Beta2 and i.MX 8QuadXPlus Beta
L4.9.51_imx8mq-ga	03/2018	Added i.MX 8MQuad GA
L4.9.88_2.0.0-ga	05/2018	i.MX 7ULP and i.MX 8MQuad GA release
L4.9.88_2.1.0_8mm-alpha	06/2018	i.MX 8MMini 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. Arm, AMBA, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and µVision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, Mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. 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 IMXLUG
Revision L4.9.88_2.1.0_8mm-alpha, 06/2018