i.MX Linux® User's Guide

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:

Contents

- `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 6Quad SABRE-SD board and platform
- i.MX 6DualLite SABRE-SD platform
- i.MX 6Quad SABRE-AI platform
- i.MX 6SoloX SABRE-SD platform
- i.MX 7Dual SABRE-SD platform
- i.MX 6QuadPlus SABRE-AI platform
- i.MX 6QuadPlus SABRE-SD platform
- i.MX 6UltraLite EVK platform
- i.MX 6ULL EVK platform
- i.MX 6ULZ EVK platform
- i.MX 7ULP EVK platform
- i.MX 8QuadMax MEK board
- i.MX 8QuadXPlus MEK platform
- i.MX 8M Quad EVK platform
- i.MX 8M Mini EVK Board
- i.MX 8M Nano 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.
- SoloX or SX refers to the i.MX 6SoloX SABRE-SD and SABRE-AI boards.
- UL refers to the i.MX 6UltraLite board.
- ULL refers to the i.MX 6ULL board.
- ULZ refers to the i.MX 6ULZ board.
- 7ULP refers to the i.MX 7 Ultra Low Power platform.
- 8QXP refers to the 8QuadXPlus platform.
- 8QM refers to the 8QuadMax platform.
- 8MQ refers to the 8M Quad platform.
- 8MM refers to the 8M Mini platform.
- 8MN refers to the 8M Nano platform.

1.4 References

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

- i.MX 6 Family: 6QuadPlus, 6Quad, 6DualLite, 6SoloX, 6SLL, 6UltraLite, 6ULL, 6ULZ
- i.MX 7 Family: 7Dual, 7ULP
- i.MX 8 Family: 8QuadMax
- i.MX 8M Family: 8M Quad, 8M Mini, 8M Nano
- i.MX 8X Family: 8QuadXPlus

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

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) - Describes the board support package for NXP development systems using Yocto Project to set up host, install tool chain, and build source code to create images.
- *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 Porting Guide* (IMXXBSPPG) - Contains the instructions on porting the BSP to a new board.
- *i.MX VPU Application Programming Interface Linux® Reference Manual* (IMXVPUAPI) - Provides the reference information on the VPU API on i.MX 6 VPU.

The quick start guides contain basic information on the board and setting it up. They are on the NXP website.

- SABRE Platform Quick Start Guide (IMX6QSDPQSG)
- SABRE Board Quick Start Guide (IMX6QSDBQSG)
- i.MX 6UltraLite EVK Quick Start Guide (IMX6ULTRALITEQSG)
- i.MX 6ULL EVK Quick Start Guide (IMX6ULLQSG)
- SABRE Automotive Infotainment Quick Start Guide (IMX6SABREINFOQSG)
- i.MX 7Dual SABRE-SD Quick Start Guide (SABRESDBIMX7DUALQSG)
- i.MX 8M Quad Evaluation Kit Quick Start Guide (IMX8MQUADEVKQSG)
- i.MX 8M Mini Evaluation Kit Quick Start Guide (8MMINIEVKQSG)
- i.MX 8M Nano Evaluation Kit Quick Start Guide (8MNANOEVKQSG)
- i.MX 8QuadXPlus Multisensory Enablement Kit Quick Start Guide (IMX8QUADXPLUSQSG)
- i.MX 8QuadMax Multisensory Enablement Kit Quick Start Guide (IMX8QUADMAXQSG)

Documentation is available online at nxp.com.

- i.MX 6 information is at nxp.com/iMX6series
- i.MX SABRE information is at nxp.com/imxSABRE
- i.MX 6UltraLite information is at nxp.com/iMX6UL
- i.MX 6ULL information is at nxp.com/iMX6ULL
- i.MX 7Dual information is at nxp.com/iMX7D
- i.MX 7ULP information is at nxp.com/imx7ulp
- i.MX 8 information is at nxp.com/imx8
- i.MX 6ULZ information is at nxp.com/imx6ulz

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.
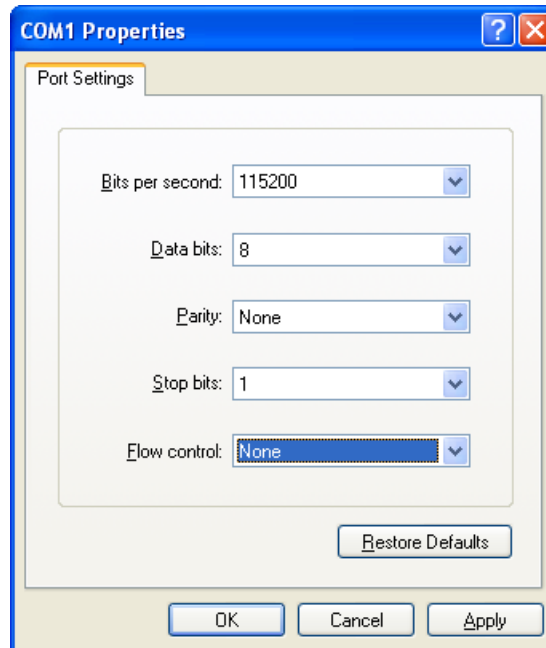
3 Basic Terminal Setup

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

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.

The i.MX 6Quad/QuadPlus/DualLite SABRE-AI boards connect to the host server using a serial cable.

The other i.MX boards connect the host driver using the micro-B USB connector.

1. Connect the target and the PC running Windows OS using a cable mentioned above.
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 6 and i.MX 7, the following elements are needed:
- Bootloader (U-Boot)
- Linux kernel image (zImage)

- A device tree file (.dtb) for the board being used
- A root file system (rootfs) for the particular Linux image
- Arm Cortex-M4 image for i.MX 7ULP

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, System controller firmware, and the SECO firmware since B0.
- Arm Cortex-M4 image
- 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 8M Quad, i.MX 8M Mini, and i.MX 8M Nano, 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 backend is XWayland. 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 (8M SoC), DDR firmware (8M Quad), and HDMI firmware (8M Quad), and SECO firmware for B0 (8QXP/8QuadMax).

On i.MX 8M SoC, 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 5.4.3 version of the Linux kernel and the device tree files associated with each platform.

The same kernel image is used for all the i.MX 6 and i.MX 7 with name zImage. 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 example, the device tree file of the i.MX 8QuadMax MEK board is Image-imx8qm-mek.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   Universal update utility

The Universal Update Utility (UUU) runs on a Windows or Linux OS host and is used to download images to different devices on an i.MX board.

4.2.1   Downloading UUU

Download UUU version 1.2.130 or later from https://github.com/NXPmicro/mfgtools/releases.

4.2.2   Using UUU

Follow these instructions to use the UUU for i.MX 6, i.MX 7, i.MX 8:

1. Connect a USB cable from a computer to the USB OTG/TYPE C 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 serial download mode mode. See Section "Serial download mode for the Manufacturing Tool" in this document.

For detailed usage of UUU, see github.com/NXPmicro/mfgtools/wiki.

For example, the following command writes rootfs.wic into eMMC.

```
uuu -b emmc_all <bootloader> <rootfs.wic>
```

The following command decompresses bz2 file and writes into eMMC:

```
uuu -b emmc_all <bootloader> <rootfs.wic.bz2/*>
```

The following command executes downloading and bootloader (SPL and U-Boot) by USB:

```
uuu -b spl <bootloader>
```

The follow command burns into eMMC (If only one board is supported in such a release package and the board supports eMMC chip):

```
uuu <release package>.zip
```

NOTE

For i.MX 8QuadXPlus B0, UUU flashes the eMMC image to boot partition with 32 KB offset. It may not be compatible with all eMMC devices. It is recommended to enable eMMC fastboot mode and use the UUU kernel version script to flash the eMMC image to boot partition with 0 offset.

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, and usually only the 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.

A .wic image contains all four images properly configured for an SD card. The release contains a pre-built .wic image that is built specifically for the one board configuration. It runs the Wayland 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 .wic image can be loaded and the individual parts can be overwritten with the specific components.

The rootfs on the default .wic 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
0x400 bytes (2)	0x9FFC00 bytes (20478)	RAW	i.MX 6 and i.MX 7 U-Boot and reserved area

Table continues on the next page...

Table 1.   Image layout (continued)

Start address (sectors)	Size (sectors)	Format	Description
0x8400 (66)	0x9F7C00 (20414)	RAW	i.MX 8M Quad and i.MX 8M Mini imx-boot reserved area
0x8000 (64)	0x9F800 (20416)	RAW	i.MX 8QuadMax/ 8QuadXPlus/8M Nano
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 .wic) 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>.wic 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 8QuadMax A0, i.MX 8QuadXPlus A0, i.MX 8M Quad, and i.MX 8M Mini
- 32 - for i.MX 8QuadXPlus B0, i.MX 8QuadMax B0, and i.MX 8M Nano

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 `imx-image-multimedia-imx7ulpevk.tar.bz2` to `/home/user/rootfs` for example:

```
$ cd /home/user/rootfs
$ tar -jxvf imx-image-multimedia-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 UUU. 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 Using eMMC

There is an eMMC chip on i.MX SABRE boards, i.MX 8 MEK boards and i.MX 8M EVK boards. It is accessed through SDHC4 on i.MX 6 SABRE boards or SDHC3 on i.MX 7Dual SABRE-SD board, SDHC1 on i.MX 8 MEK boards and i.MX 8M EVK boards. The i.MX 7ULP EVK board also supports to rework eMMC on the MicroSD port. The following steps describe how to use this memory device.

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 `uuu`
`L5.4.3_1.0.0_images_MX7ULPEVK.zip\uuu_sd_m4.auto` to burn both BSP and
Arm Cortex-M4 images.

1. Execute the following command on the U-Boot console to clean up the environments stored on eMMC:

```
U-Boot > env default -f -a
U-Boot > save
U-Boot > reset
```

2. Configure the boot pin. Power on the board and set the U-Boot environment variables as required. For example,

```
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 fake MAC address via 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 to the TFTP server. Then download it to RAM:

```
U-Boot > dhcp
```

4. Query the information about the eMMC chip.

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

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

```
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 into eMMC. For example, the command below writes the image with the size
0x800000 from ${loadaddr} to the offset 0x100000 of the eMMC chip. Here, the following equations are used: 0x800
=0x100000/0x200, 0x4000=0x800000/0x200. The block size of this card is 0x200. This example assumes that the
kernel image is less than 0x800000 bytes. If the kernel image exceeds 0x800000, enlarge the image length.

```
### Select mmc dev 2 (USDHC4) on the i.MX 6 SABRESD board:
U-Boot > mmc dev 2 0
### Select mmc dev 1 (USDHC3) on the i.MX 7Dual SABRESD board:
U-Boot > mmc dev 1 0
### Select mmc dev 1 (USDHC2) on the i.MX 6UltraLite EVK board:
U-Boot > mmc dev 1 0
### Select mmc dev 0 (USDHC1) on the i.MX 7ULP EVK board:
U-Boot > mmc dev 0 0
### Select mmc dev 0 (eMMC0) on the i.MX 8QuadMax MEK board and iMX 8QuadXPlus MEK
board:
U-Boot > mmc dev 0 0
### Suppose kernel zImage is less than 8 MB:
U-Boot > tftpboot ${loadaddr} ${bootfile}
U-Boot > mmc write ${loadaddr} 0x800 0x4000
```

7. Program the dtb file located in RAM at ${fdt_addr} into the eMMC chip.

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

8. Boot up the system through the rootfs in eMMC, using the HannStar LVDS as display. The kernel MMC module now
uses the fixed mmcblk indices for the USDHC slots. The eMMC/SD4 slot on the i.MX 6 SABRE boards is mmcblk3.
The eMMC5.0 on the i.MX 8QuadMax MEK board, i.MX 8QuadXPlus MEK board, and i.MX 8M Quad EVK board

are mmcblk0. The eMMC5.0/SD3 slot on the i.MX 7Dual SABRE board is mmcblk2. eMMC is not populated on the i.MX 7Dual SABRE board.

```
U-Boot > setenv mmcboot 'run bootargs_base mmcargs; mmc dev 2;
mmc read ${loadaddr} 0x800 0x4000; mmc read ${fdt_addr} 0x5000 0x800;bootz ${loadaddr}
- ${fdt_addr} '
U-Boot > setenv bootcmd 'run mmcboot'
U-Boot > saveenv
```

9. Boot up the system through the rootfs in eMMC, using the CLAA WVGA panel as display:
   - For i.MX 6 boards:

     ```
     U-Boot > setenv mmcargs 'setenv bootargs ${bootargs}
     root=/dev/mmcblk3p2 rootwait rw video=mxcfb0:dev=lcd,CLAA-WVGA,if=RGB565 ip=dhcp'
     ```

   - For i.MX 7Dual SABRE boards:

     ```
     U-Boot > setenv mmcargs 'setenv bootargs ${bootargs}
     root=/dev/mmcblk2p2 rootwait rw video=mxcfb0:dev=lcd,CLAA-WVGA,if=RGB565 ip=dhcp'
     ```

10. Boot up the system through rootfs in eMMC, using HDMI as display:
    - For i.MX 6 boards:

      ```
      U-Boot > setenv mmcargs 'setenv bootargs ${bootargs} root=/dev/mmcblk3p2 rootwait
      rw video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24'
      ```

    - For i.MX 7Dual SABRE boards:

      ```
      U-Boot > setenv mmcargs 'setenv bootargs ${bootargs} root=/dev/mmcblk2p2 rootwait
      rw video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24'
      ```

    - For i.MX 8QuadMax/8QuadXPlus, the following display kernel parameters are supported:
      1. Pick a particular video mode for legacy FB emulation since system startup.

         ```
         video=HDMI-A-{n}: {video_mode}
         ```

         n can be 1 to the maximum number of HDMI connectors in the system. video_mode should be the one that the monitor on the connector supports. For example, `video=HDMI-A-1:1920x1080@60`. By default, if there is no parameter in the command line, the system uses the video mode that the monitor recommends.

      2. Enable or disable legacy FB emulation.

         ```
         drm_kms_helper.fbdev_emulation=0 or 1
         ```

         0 to disable, 1 to enable. By default, if there is no parameter in the command line, the emulation is enabled.

      3. Set legacy FB emulation framebuffer's bits per pixel (bpp) parameter.

         ```
         imxdrm.legacyfb_depth=16 or 24 or 32
         ```

         By default, if there is no parameter in the command line, bpp is 16.

To program the rootfs to MMC/SD, see Using an i.MX board as the host server to create a rootfs or Preparing an SD/MMC card to boot.

4.4.1.2 Flashing U-Boot on SPI-NOR from U-Boot

Flashing directly to SPI-NOR with TFTPBoot is limited to i.MX 6 SABRE-AI boards. To flash U-Boot on SPI-NOR, perform the following steps:

1. Boot from an SD card.
2. Set Jumper J3 to position: 2-3.
3. Fetch the U-Boot image with built-in SPI-NOR support. This example uses `u-boot.imx`.

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

```
U-Boot > tftpboot ${loadaddr} u-boot.imx
```

4. Flash the U-Boot image in SPI-NOR.

```
U-Boot > sf probe
U-Boot > sf erase 0 0x80000
U-Boot > sf write ${loadaddr} 0x400 0x7FC00
```

5. Set boot switches to boot from SPI-NOR on SABRE-AI.
   - S2-1 1
   - S2-2 1
   - S2-3 0
   - S2-4 0
   - S1-[1:10] X
6. Reboot the target board.

4.4.1.3 Flashing U-Boot on Parallel NOR from U-Boot

Flashing directly to Parallel NOR with TFTPBoot is limited to i.MX 6 SABRE-AI boards. To flash U-Boot on Parallel NOR, perform the following steps:

1. Check the jumper J3, should not between pins 2 and 3.
2. Update the SD U-Boot with EIM NOR version. For details on commands, see Copying a bootloader image. Then boot from the SD card.
3. TFTP the U-Boot image.

```
tftpboot ${loadaddr} u-boot.imx
```
4. Flash the U-Boot image.

```
cp.b ${loadaddr} 0x08001000 ${filesize}
```
5. Change boot switches and reboot.

```
S2 all 0 S1-6 1 others 0
```
6. By default, rootfs is mounted on NFS.

4.4.1.4 Flashing an Arm Cortex-M4 image on QuadSPI

i.MX 6SoloX SABRE-SD/SABRE-AI, i.MX 7ULP EVK, and i.MX 7Dual SABRE-SD boards have the Arm Cortex-M4 processor and QuadSPI memory that can be used to flash an image to it.

NOTE

> For i.MX 7ULP, use MfgTool to burn the Arm Cortex-M4 image to QuadSPI. The script
> mfgtool2-yocto-mx-evk-qspi-nor-m4.vbs is dedicated to burn the Arm Cortex-M4 image.
> The users can also use other scripts to burn both the Arm Cortex-M4 and BSP images.
> See the SD and eMMC sections.

i.MX U-Boot provides a reference script on i.MX 7Dual SABRESD and i.MX 6SoloX SABRE-AI/SABRE-SD to flash the Arm Cortex-M4 image from the SD card. To execute the script, perform the following steps:

1. Copy the Arm Cortex-M4 image to the first VFAT partition of the boot SD card. Name the file to "m4_qspi.bin".
2. Boot from the SD card.
3. Flash the Arm Cortex-M4 image from the SD card to the NOR flash on QuadSPI2 PortB CS0 on the i.MX 6SoloX SABRE-SD board or QuadSPI1 PortB CS0 on the i.MX 6SoloX SABRE-AI board or QuadSPI1 PortA CS0 offset 1 MB on the i.MX 7Dual SABRE-SD board.

```
U-Boot > run update_m4_from_sd
```

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

Alternatively, users can flash the Arm Cortex-M4 image from TFTP by performing the following steps:

1. Boot from the SD card.
2. TFTP the Arm Cortex-M4 image.

   ```
   U-Boot > tftp ${loadaddr} m4_qspi.bin
   ```
3. Select the NOR flash on QuadSPI2 PortB CS0 on the i.MX 6SoloX SABRE-SD board or QuadSPI1 PortB CS0 on the i.MX 6SoloX SABRE-AI board.

   ```
   U-Boot > sf probe 1:0
   ```

   Select the NOR flash on QuadSPI1 PortA CS0 on the i.MX 7Dual SABRE-SD board and i.MX 7ULP EVK board.

   ```
   U-Boot > sf probe 0:0
   ```
4. Flash the Arm Cortex-M4 image to the selected NOR flash. The erase size is ${filesize}, around 64 Kbytes. This example assumes that it is 128 Kbytes.

   ```
   U-Boot > sf erase 0x0 0x20000
   U-Boot > sf write ${loadaddr} 0x0 ${filesize}
   ```

   i.MX 7Dual SABRE-SD needs to program the Arm Cortex-M4 images to 1 MB offset, because the first 1 MB is used by the U-Boot image in QuadSPI.

   ```
   U-Boot > sf erase 0x100000 0x20000
   U-Boot > sf write ${loadaddr} 0x100000 ${filesize}
   ```

NOTE

On i.MX 7Dual SABRE-SD, the Arm Cortex-M4 image on QuadSPI is supported only when the U-Boot image is built by the target mx7dsabresd_qspi1_defconfig booted by U-Boot from QuadSPI.

The default U-Boot for i.MX 7Dual SABRESD boards uses the Cortex-M4 image from the SD card and runs it on OCRAM.

On i.MX 7ULP EVK, the Arm Cortex-M4 image needs to be programmed. Otherwise, it will not boot.

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@ ~$ 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,
   ```

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

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@ ~$ 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 /imx-image-multimedia.ext3 /mnt/tmpmnt
cd /mnt
mkdir mmcblk0p1
mount -t ext3 /dev/$PARTITION /mnt/mmcblk0p1

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

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

```
     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 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

The following table shows the DIP switch settings for booting from the SD card slot labeled SD1 on the i.MX 7Dual SABRE-SD boards.

Table 2. Booting from SD1 on i.MX 7Dual SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW2	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
SW3	ON	OFF	-	-	-	-	-	-

The following table shows the DIP switch settings for booting from the SD card slot labeled SD1 on the i.MX 7ULP EVK boards.

Table 3. Booting from SD1 on i.MX 7ULP EVK

Switch	D1	D2	D3	D4
SW1	ON	OFF	OFF	ON

The following table shows the bootcfg pin settings for booting from the SD card slot labeled SD1 on the i.MX 8QuadMax MEK boards.

Table 4. Booting from SD1 on i.MX 8QuadMax MEK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW2	OFF	OFF	ON	ON	OFF	OFF	-	-

The following table shows the bootcfg pin settings for booting from the SD card slot labeled SD1 on the i.MX 8QuadXPlus MEK boards.

Table 5.   Booting from SD1 on i.MX 8QuadXPlus MEK

Switch	D1	D2	D3	D4
SW2	ON	ON	OFF	OFF

4.5.2   Booting from an SD card in slot SD3

The SD card slot that is labeled SD3 indicates that this slot is connected to the uSDHC pin SD3 on the processor. Most boards label this slot as SD3. This slot is referred to as SD3 in this document.

i.MX 6 SABRE-AI boards

The following table shows the DIP switch settings to boot from an SD card in slot SD3 on i.MX 6 SABRE-AI boards.

Table 6.   Booting from an SD card in slot SD3 on i.MX 6 SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
S1	X	X	X	OFF	ON	X	X	X	X	X
S2	X	OFF	ON	OFF	-	-	-	-	-	-
S3	OFF	OFF	ON	OFF	-	-	-	-	-	-

i.MX 6SoloX SABRE-AI boards

The following table shows the DIP switch settings to boot from an SD card in slot SD3 on i.MX 6SoloX SABRE-AI boards.

Table 7.   Booting from an MMC card in Slot SD3 on i.MX 6SoloX SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8
S4	OFF	ON	OFF	X	OFF	OFF	ON	OFF
S3	X	OFF	OFF	OFF	ON	ON	OFF	OFF
S1	OFF	OFF	ON	OFF	-	-	-	-

i.MX 6 SABRE-SD boards

The following table shows the DIP switch settings for booting from SD3, also labeled as J507. The SD3 slot is located between the HDMI and UART ports.

Table 8.   Booting from an SD card in slot SD3 on i.MX 6 SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW6	OFF	ON	OFF	OFF	OFF	OFF	ON	OFF

4.5.3   Booting from an SD card in slot SD4

The following table describes the dip switch settings for booting from an SD card in slot SD4.

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

The SD4 slot is on the center of the edge of the SoloX board.

Table 9.   Booting from an SD card in slot SD4 on i.MX 6SoloX SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW10	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW11	OFF	OFF	ON	ON	ON	OFF	OFF	OFF
SW12	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF

Table 10.   Booting from an MMC card in slot SD4 on i.MX 6SoloX SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW10	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW11	OFF	OFF	ON	ON	ON	OFF	OFF	OFF
SW12	OFF	ON	ON	OFF	OFF	OFF	OFF	OFF

4.5.4   Booting from eMMC

eMMC 4.4 is a chip permanently attached to the board that uses the SD4 pin connections from the i.MX 6 processor. For more information on switch settings, see table "MMC/eMMC Boot Fusemap" in the IC reference manual.

The following table shows the boot switch settings to boot from eMMC4.4 (SDIN5C2-8G) on i.MX 6 SABRE-SD boards.

Table 11.   Booting from eMMC on i.MX 6 SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW6	ON	ON	OFF	ON	OFF	ON	ON	OFF

i.MX 7Dual is different from i.MX 6. The eMMC uses the SD3 pin connections from the i.MX 7Dual processor.

Table 12.   Booting from eMMC on i.MX 7Dual SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW2	OFF	ON	OFF	ON	OFF	OFF	OFF	OFF
SW3	ON	OFF	-	-	-	-	-	-

The following table shows the boot switch settings to boot from eMMC4.4 on the i.MX 7ULP EVK boards.

Table 13.   Booting from eMMC on i.MX 7ULP EVK

Switch	D1	D2	D3	D4
SW1	ON	OFF	OFF	OFF

The following table shows the boot switch settings to boot from eMMC5.0 on the i.MX 8QuadMax MEK boards.

Table 14.   Booting from eMMC on i.MX 8QuadMax MEK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW2	OFF	OFF	OFF	ON	OFF	OFF	-	-

The following table shows the boot switch settings to boot from eMMC5.0 on the i.MX 8QuadXPlus MEK boards.

Table 15.   Booting from eMMC on i.MX 8QuadXPlus MEK

Switch	D1	D2	D3	D4
SW2	OFF	ON	OFF	OFF

The following table shows the boot switch settings to boot from eMMC5.0 on the i.MX 8M Quad EVK boards.

Table 16.   Booting from eMMC on i.MX 8M Quad EVK

Switch	D1	D2	D3	D4
SW801	OFF	OFF	ON	OFF

The following table shows the boot switch settings to boot from eMMC5.1 on the i.MX 8M Mini EVK boards.

Table 17.   Booting from eMMC on i.MX 8M Mini EVK

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

The following table shows the boot switch settings to boot from eMMC5.1 on the i.MX 8M Nano EVK boards.

Table 18.   Booting from eMMC on i.MX 8M Nano EVK

Switch	D1	D2	D3	D4
SW1101	OFF	ON	OFF	OFF

4.5.5   Booting from SATA

The following switch settings enable booting from SATA.

SATA booting is supported only by the i.MX 6Quad/6QuadPlus SABRE boards.

Table 19.   Booting from SATA on i.MX 6 SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW6	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF

4.5.6 Booting from NAND

The following table shows the DIP switch settings needed to boot from NAND on i.MX 6 SABRE-AI boards.

Table 20. Booting from NAND on i.MX 6 SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
S1	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
S2	OFF	OFF	OFF	ON	-	-	-	-	-	-
S3	OFF	OFF	>ON	OFF	-	-	-	-	-	-

The following table shows the DIP switch settings needed to boot from NAND for i.MX 7Dual SABRE-SD boards.

Table 21. Booting from NAND on i.MX 7Dual SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
S2	OFF	ON	ON	X	X	X	X	OFF		
S3	ON	OFF	X	X	X	X	X	X		

The following table shows the DIP switch settings needed to boot from NAND for i.MX 8M Mini DDR4 EVK boards.

Table 22. Booting from NAND on i.MX 8M Mini DDR4 EVK

Switch	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
SW1101	OFF	ON	OFF	OFF	OFF	OFF	OFF	ON		
SW1102	OFF	OFF	OFF	ON	ON	ON	ON	OFF		

4.5.7 Booting from SPI-NOR

Enable booting from SPI NOR on i.MX 6 SABRE-AI boards by placing a jumper on J3 between pins 2 and 3.

Table 23. Booting from SPI-NOR on i.MX 6 SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
S1	X	X	X	X	X	X	X	X	X	X
S2	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
S3	OFF	OFF	ON	OFF	-	-	-	-	-	-

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

4.5.8   Booting from EIM (Parallel) NOR

The following table shows the DIP switch settings to boot from NOR.

Table 24.   Booting from EIM NOR on i.MX 6 SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
S1	X	X	X	OFF	OFF	ON	X	X	X	X
S2	X	OFF	OFF	OFF	-	-	-	-	-	-
S3	OFF	OFF	ON	OFF	-	-	-	-	-	-

NOTE

SPI and EIM NOR have pin conflicts on i.MX 6 SABRE-AI boards. Neither can be used for the same configuration. The default U-Boot configuration is set to SPI NOR.

4.5.9   Booting from QuadSPI or FlexSPI

The following tables list the DIP switch settings for booting from QuadSPI.

Table 25.   Booting from QuadSPI on i.MX 6SoloX SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW10	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW11	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW12	OFF	OFF	OFF	ON	ON	OFF	OFF	OFF

Table 26.   Booting from QuadSPI on i.MX 6SoloX SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW4	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
SW3	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW1	OFF	OFF	ON	OFF	-	-	-	-

Table 27.   Booting from QuadSPI on i.MX 7Dual SABRE-SD

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW2	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW3	ON	OFF	-	-	-	-	-	-

Table 28.   Booting from QuadSPI on i.MX 6UltraLite EVK and i.MX 6ULL EVK

Switch	D1	D2	D3	D4
SW601	OFF	OFF	OFF	OFF
SW602	ON	OFF	-	-

Table 29. Booting from FlexSPI on i.MX 8QuadMax MEK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW2	OFF	OFF	OFF	ON	ON	OFF	-	-

Table 30. Booting from FlexSPI on i.MX 8QuadXPlus MEK

Switch	D1	D2	D3	D4
SW2	OFF	ON	ON	OFF

Table 31. Booting from FlexSPI on i.MX 8M Mini LPDDR4 EVK

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

Table 32. Booting from QuadSPI on i.MX Nano EVK

Switch	D1	D2	D3	D4
SW601	OFF	OFF	OFF	OFF
SW602	ON	OFF	-	-

4.5.10 Serial download mode for the Manufacturing Tool

No dedicated boot DIP switches are reserved for serial download mode on i.MX 6 SABRE-SD. There are various ways to enter serial download mode. One way is to set the boot mode to boot from SD slot SD3 (set SW6 DIP switches 2 and 7 to **on**, and the rest are **off**). Do not insert the SD card into slot SD3, and power on the board. After the message "HID Compliant device" is displayed, the board enters serial download mode. Then insert the SD card into SD slot SD3. Another way to do this is to configure an invalid boot switch setting, such as setting all the DIP switches of SW6 to off.

The following table shows the boot switch settings for i.MX 6 SABRE-AI boards, which are used to enter serial download mode for the Manufacturing Tool. If the boot image in the boot media is not validated, the system also enters the serial download mode.

Table 33. Setup for the Manufacturing Tool on i.MX 6 SABRE-AI

Switch	D1	D2	D3	D4
S3	OFF	ON	OFF	OFF

Table 34. Setup for the Manufacturing Tool on i.MX 7Dual SABRE-SD

Switch	D1	D2	D3	D4
S3	OFF	ON	-	-

Table 35.   Setup for Manufacturing Tool on i.MX 6UltraLite EVK and i.MX 6ULL EVK

Switch	D1	D2
SW602	OFF	ON

Table 36.   Setup for Manufacturing Tool on i.MX 7ULP EVK

Switch	D1	D2	D3	D4
SW1	OFF	ON	-	-

Table 37.   Setup for Manufacturing Tool on i.MX 8M Quad EVK

Switch	D1	D2
SW802	OFF	ON

Table 38.   Setup for Manufacturing Tool on i.MX 8M Mini 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

Table 39.   Setup for Manufacturing Tool on i.MX 8M Nano EVK

Switch	D1	D2	D3	D4
SW1101	ON	OFF	OFF	OFF

Table 40.   Setup for Manufacturing Tool on i.MX 8QuadMax MEK

Switch	D1	D2	D3	D4	D5	D6
SW2	OFF	OFF	ON	OFF	OFF	OFF

Table 41.   Setup for Manufacturing Tool on i.MX 8QuadXPlus MEK

Switch	D1	D2	D3	D4
SW2	ON	OFF	OFF	OFF

NOTE

If the SD card with bootable image is plugged in SD2 (baseboard), ROM will not fall
back into the serial download mode.

4.5.11 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/5.4.3/environment-setup-aarch64-poky-linux
export ARCH=arm64
```
- For i.MX 6 and i.MX 7 builds on the host machine, set the environment with the following command before building.

```
source /opt/fsl-imx-fb/5.4.3/environment-setup-cortexa9hf-neon-poky-linux-gnueab
export ARCH=arm
```
- 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
```
- Users can also build SPL for i.MX 8QuadMax and i.MX 8QuadXPlus MEK boards. It is needed when booting with OPTEE image.
  For i.MX 8QuadMax MEK:

```
make clean
make imx8qm_mek_spl_defconfig
make
```

For i.MX 8QuadXPlus MEK:

```
make clean
make imx8qxp_mek_spl_defconfig
make
```
- To build U-Boot for i.MX 8M Quad EVK:

```
make clean
make imx8mq_evk_defconfig
make
```
- For i.MX 8M LPDDR4 EVK:

```
make clean
make imx8mm_evk_defconfig
make
```
- For i.MX 8M DDR4 EVK:

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

```
make clean
make imx8mm_ddr4_evk_defconfig
make
```
• To build an i.MX 6 or i.MX 7 U-Boot in the standalone environment, find the configuration for the target boot. In the following example, i.MX 6ULL is the target.

```
Download source by cloning with git clone https://source.codeaurora.org/external/imx/
uboot-imx -b imx_v2019.04_5.4.3_1.0.0
cd uboot-imx
make clean
make mx6ull_14x14_evk_defconfig
make
```
• To build the kernel in the standalone environment for i.MX 8, execute the following commands:

```
make imx_v8_defconfig
make
```
• To build the kernel in the standalone environment for i.MX 6 and i.MX 7, execute the following commands:

```
Download source by cloning with git clone https://source.codeaurora.org/external/imx/
linux-imx -b imx_5.4.3-1.0.0
cd linux-imx
make imx_v7_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.

4.5.12   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. Copy the SECO firmware container image (ahab-container.img) to imx-mkimage/iMX8QM/.
5. Run `make SOC=iMX8QM flash` to generate flash.bin.
6. If using OPTEE, copy tee.bin to imx-mkimage/iMX8QM/ and copy u-boot/spl/u-boot-spl.bin to imx-mkimage/iMX8QM/. Run `make SOC=iMX8QM flash_spl` 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. Copy the SECO firmware container image (ahab-container.img) to imx-mkimage/iMX8QX/.
5. Run `make SOC=iMX8QX flash` to generate flash.bin.
6. If using OPTEE, copy tee.bin to imx-mkimage/iMX8QX/ and copy u-boot/spl/u-boot-spl.bin to imx-mkimage/iMX8QX/. Run `make SOC=iMX8QX flash_spl` to generate flash.bin.

The following is a matrix table for targets of i.MX 8QuadMax and i.MX 8QuadXPlus

Table 42. Matrix table for targets of i.MX 8QuadMax and i.MX 8QuadXPlus

-	OPTEE	U-Boot	SPL	Cortex-M4
flash_spl	Yes	Yes	Yes	No
flash	No	Yes	No	No
flash_linux_m4	Yes	Yes	Yes	Yes
flash_regression_linux_m4	No	Yes	No	Yes

For i.MX 8M EVK boards 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 8M Quad EVK) or fsl-imx8mm-evk (for i.MX 8M Mini LPDDR4 EVK) or fsl-imx8mm-ddr4-evk (for i.MX 8M Mini DDR4 EVK) from u-boot/arch/arm/dts/ to imx-mkimage/iMX8M/.
5. 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. For i.MX 8M Quad and i.MX 8M Mini LPDDR4 EVK, 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/.

   For i.MX 8M Mini DDR4 EVK, copy ddr4_imem_1d.bin, ddr4_dmem_1d.bin, ddr4_imem_2d.bin, and ddr4_dmem_2d.bin from firmware/ddr/synopsys of firmware-imx package to imx-mkimage/iMX8M.

8. For i.MX 8M Quad EVK, run `make SOC=iMX8M flash_evk` to generate flash.bin (imx-boot image) with HDMI FW included. For i.MX 8M Mini LPDDR4 EVK, run `make SOC=iMX8MM flash_evk` to generate flash.bin (imx-boot image). For i.MX 8M Mini DDR4 EVK, run `make SOC=iMX8MM flash_ddr4_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/SATA memory map

The MMC/SD/SATA 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.

4.6.2   NAND flash memory map

The NAND flash memory map is configured from the Linux kernel command line.

For example:

```
mtdparts=gpmi-nand:64m(boot),16m(kernel),16m(dtb),-(rootfs)
```

4.6.3   Parallel NOR flash memory map

The default configuration contains only one parallel NOR partition. The parallel NOR device is generally 4 MB. U-Boot is loaded at the beginning of parallel NOR so that the device can boot from it. The default configuration is that on boot up, U-Boot loads the kernel, DTB, and root file system from the SD/MMC card into DDRAM. The end user can change the default settings according to their needs. More partitions can be added through the kernel command line. The memory type designator for the command below consists of the NOR address and the designator. This information can be found in the imx .dtsi device tree file in arch/arm/boot/dts. The following is an example of what might be added to the Linux boot command line:

```
mtdparts=8000000.nor:1m(uboot),-(rootfs)
```

The address for parallel NOR is 0x8000000 for i.MX 6 SABRE-AI.

4.6.4   SPI-NOR flash memory map

The SPI-NOR flash memory can be configured using the Linux kernel command line.

U-Boot should be loaded at the 1 KB offset of the SPI-NOR memory, so that the device can boot from it. The default configuration is that on boot up, U-Boot loads the kernel, DTB, and root file system from the SD/MMC card into DDRAM. The end user can change the default settings according to their needs. More partitions can be added through the kernel command line. The following is an example of what might be added to the Linux boot command line:

```
mtdparts=spi32766.0:768k(uboot),8k(env),128k(dtb),-(kernel)
```

4.6.5   QuadSPI flash memory map

The QuadSPI flash memory can be configured using the Linux kernel command line.

U-Boot is loaded at the beginning of the QuadSPI memory so that the device can boot from it. The default configuration is that on boot up, U-Boot loads the kernel, DTB, and root file system from the SD/MMC card into DDRAM. The end user can change the default settings according to their requirements. More partitions can be added through the kernel command line. The following is an example of what might be added to the Linux boot command line:

```
mtdparts=21e4000.qspi:1m(uboot),8m(kernel),1m(dtb),-(user)
```

U-Boot has the mapping below to help in accessing the QuadSPI flash in U-Boot for non-parallel mode.

Table 43.   U-Boot mapping for QuadSPI

Device on hardware	Device in U-Boot	Memory address in U-Boot	Remark
QuadSPI1 Port A CS0	sf probe 0:0 on i.MX 6SoloX SABRE-AI board, i.MX 7Dual SABRE-SD board, i.MX 6UltraLite EVK board, i.MX 8QuadMax MEK and QuadXPlus MEK	0x60000000 0x08000000	-
QuadSPI1 Port B CS0	sf probe 1:0 on i.MX 6 SoloX SABRE-AI board	0x68000000	-
QuadSPI2 Port A CS0	sf probe 0:0 on i.MX 6SoloX SABRE-SD board	0x70000000	-
QuadSPI2 Port B CS0	sf probe 1:0 on i.MX 6SoloX SABRE-SD board	0x78000000	-

5   Enabling Solo Emulation

Solo emulation can be enabled on the i.MX 6DualLite SABRE-SD and i.MX 6DualLite SABRE-AI boards. This is achieved by using a specific U-Boot configuration in the bootloader build process.

When this Solo emulation is enabled on the i.MX 6DualLite SABRE platforms, the capabilities of the i.MX 6DualLite change to the following:

- For solo emulation, we use DualLite DTB and need to add maxcpus=1 to bootcmd of U-Boot.
- 32-bit data bus on DDR RAM
- 1 GB of RAM for i.MX 6DualLite SABRE-AI
- 512 MB of RAM for i.MX 6DualLite SABRE-SD

To build U-Boot for an i.MX 6Solo on an i.MX 6DualLite SABRE-SD card, use the following command:

```
MACHINE=imx6solosabresd bitbake u-boot-imx
```

To build U-Boot for an i.MX 6Solo on an i.MX 6DualLite SABRE-AI card, use the following command:

```
MACHINE=imx6solosabreauto bitbake u-boot-imx
```

6 Power Management

The i.MX power management uses the standard Linux interface. Check the standard Linux power documentation for information on the standard commands. The *i.MX Linux*® *Reference Manual* (IMXLXRM) contains information on the power modes that are available and other i.MX-specific information in the power management section.

There are three main power management techniques on i.MX boards: suspend and resume commands, CPU frequency scaling, and bus frequency scaling. They are described in the following sections.

6.1 Suspend and resume

The power state can be changed by setting the standard Linux state, `/sys/power/state`. The command used to set the power state into suspend mode, available from the command line, is `echo mem > /sys/power/state`. The value `mem` can be replaced by any of the valid power states, as described by the *i.MX Linux*® *Reference Manual* (IMXLXRM).

Use one of the following methods to wake up the system from suspend mode.

- The debug UART can be set as a wakeup source with:

```
echo enabled > /sys/class/tty/ttymxc0/power/wakeup
```

NOTE

It is ttylp0 for i.MX 8QuadXPlus and i.MX 8QuadMax.

- RTC can be used to enter and exit from suspend mode by using the command:

```
/unit_test/SRTC/rtcwakeup.out -d rtc0 -m mem -s 10
```

This command indicates to sleep for 10 secs. This command automatically sets the power state to `mem` mode.

6.2 CPU frequency scaling

Scaling governors are used in the Linux kernel to set the CPU frequency. CPU frequencies can be scaled automatically depending on the system load either in response to ACPI events or manually by userspace programs. For more information about governors, read governors.txt from www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

The following are some of the more frequently used commands:

These commands return information about the system and the current settings.

- The kernel is pre-configured to support only certain frequencies. The list of frequencies currently supported can be obtained from:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```
- To get the available scaling governors:

```
cat /sys/devices/system/cpu/*/cpufreq/scaling_available_governors
```
- To check the current CPU frequency:

```
cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_cur_freq
```

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

The frequency is displayed depending on the governor set.
- To check the maximum frequency:

  ```
  cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_max_freq
  ```
- To check the minimum frequency:

  ```
  cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_min_freq
  ```

These commands set a constant CPU frequency:

- Use the maximum frequency:

  ```
  echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  ```
- Use the current frequency to be the constant frequency:

  ```
  echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  ```
- The following two commands set the scaling governor to a specified frequency, if that frequency is supported. If the frequency is not supported, the closest supported frequency is used:

  ```
  echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  echo <frequency> > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
  ```

6.3 Bus frequency scaling

This release does not support the bus frequency scaling feature on i.MX 7ULP EVK.

This release does not support the bus frequency scaling feature on i.MX 8QuadXPlus and i.MX 8QuadMax.

The system automatically adjusts the bus frequency (DDR, AHB, etc.) for optimal performance based on the devices that are active.

The bus frequency driver is enabled by default. The following DDR frequencies are supported:

- Normal DDR frequency – Default frequency in U-Boot
- Audio DDR frequency – 50 MHz on i.MX 6Quad, i.MX 6DualLite, and i.MX 6SoloX, 100 and 100 MHz on i.MX 7Dual
- Low power idle DDR frequency – 24 MHz

To enter a low power idle DDR frequency, ensure that all devices that require high DDR frequency are disabled. Most drivers do active clock management, but certain commands can be used to avoid waiting for timeouts to occur:

`echo 1 > /sys/class/graphics/fb0/blank` -> to blank the display (may need to blank fb1, fb2, and so on, if more than one display is active).

`ifconfig eth0 down` -> disables the Ethernet module. On i.MX 6SoloX, i.MX 7Dual, i.MX 6UltraLite, and i.MX 6UltraLiteLite should also disable Ethernet 1 (eth1).

On most systems, the chip enters low power IDLE mode after the above two commands are executed.

To manipulate bus frequency, use the following commands to achieve the results desired:

`cat /sys/bus/platform/drivers/imx_busfreq/soc\:busfreq/enable` -> displays the status of bus frequency.

`echo 0 > /sys/bus/platform/drivers/imx_busfreq/soc\:busfreq/enable` -> disables bus frequency.

`echo 1 > /sys/bus/platform/drivers/imx_busfreq/soc\:busfreq/enable` -> enables bus frequency.

The *i.MX Linux® Reference Manual* (IMXLXRM) has more information on the bus frequency in the chapter about DVFS.

7 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.

7.1 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).

7.2 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`.

7.3 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 44. 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.

7.3.1 Playback use cases

Playback use cases include the following:

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

7.3.1.1 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 ! pulsesink
```

7.3.1.2 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
      ! vpudec ! autovideosink
```

7.3.1.3 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 platforms without VPU hardware, $video_decoder_plugin could be a software decoder plugin like avdec_h264.

7.3.1.4   Multichannel audio playback

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

7.3.1.5   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
```

7.3.1.6   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).

7.3.1.6.1   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"
```

7.3.1.6.2   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"
```

7.3.1.6.3   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.
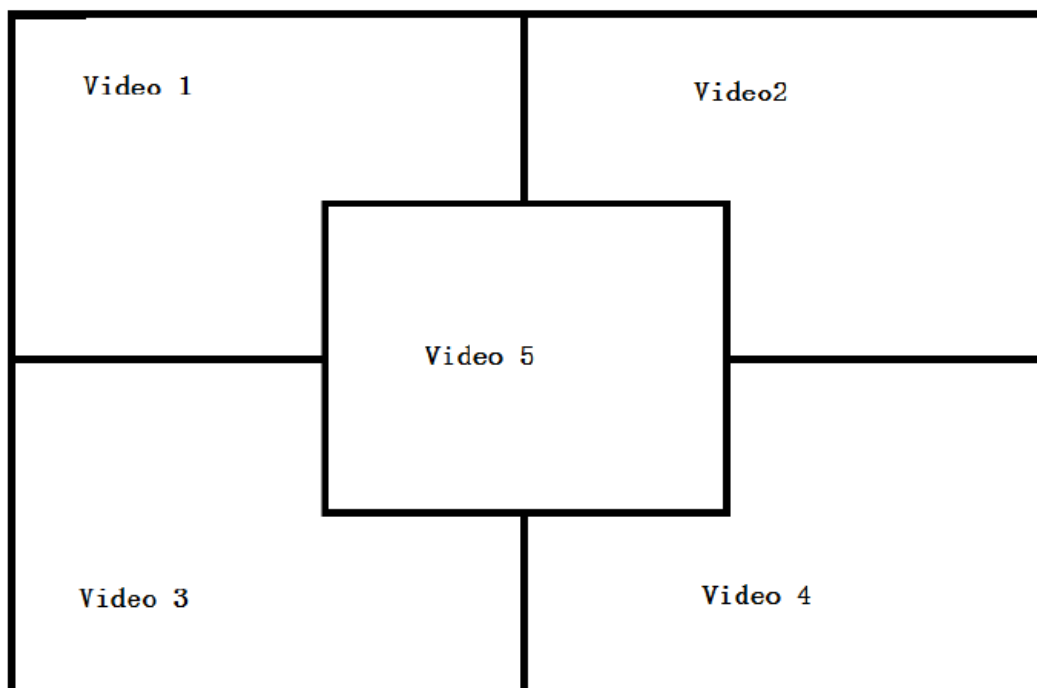
Figure 2. Multiple video overlay

```
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"
```

7.3.2   Audio encoding

Here are some examples for MP3 encoding.

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

7.3.3   Video encoding

The commands below presents some suggestions on how to use the plugins accelerated by VPU hardware to encode some media files (though they only work on a SoC with a VPU).

VPU video encoding only works on SoC with VPU encoder support.

```
$GSTL filesrc location=test.yuv
    ! videoparse format=2 witdh=$WIDTH height=$HEIGHT framerate=30/1
    ! vpuenc_xxx ! $MUXER ! filesink location=$output
```

- The target encoder codec type can be MPEG4, H263, H264, or MJPEG.

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

- The vpuenc_xxx can be vpuenc_mpeg4, vpuenc_h263, vpuenc_h264, or vpuenc_jpeg.
- The $MUXER can be set to qtmux, matroskamux, mp4mux, avimux, or flvmux.
- Different muxers support different encoded codec types. Use $GSTINSPECT and $MUXER to see the capabilities of the muxer to be used.

7.3.4 Transcoding

Transcoding is converting a file from one video encoding to another.

VPU video encoding only works on SoC with VPU encoder support.

```
$GSTL filesrc location=$filename typefind=true ! $capsfilter ! aiurdemux
! vpudec ! imxvideoconvert_ipu ! $CAPS1 ! vpuenc_xxx ! matroskamux ! filesink
location=720p.mkv
```

capsfilter is the container's mime type. CAPS1 is the target video resolution, and the vpuenc_xxx can be vpuenc_mpeg4, vpuenc_h263, vpuenc_h264, and vpuenc_jpeg..

For example:

```
gst-launch-1.0 filesrc location=$FILE.mp4 typefind=true ! video/quicktime ! aiurdemux !
vpudec ! imxvideoconvert_ipu ! video/x-raw,format=NV12,width=1280,height=720 ! vpuenc_h263 !
matroskamux ! filesink location=$FILE.mkv
```

7.3.5 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
```

7.3.6 Video recording

Video recording is done using the camera input, so this activity only applies to platforms with a camera. Different cameras need to be set with different capture modes for special resolutions. See Chapter "Supporting the i.MX 6Dual/6Quad/6Solo/6DualLite Camera Sensor with CSI" in the *i.MX BSP Porting Guide* (IMXBSPPG).

VPU video encoding only works on SoC with VPU encoder support.

Use the $GSTINSPECT command to obtain more information about the codec property.

An example of recording might look like this:

```
$GSTL imxv4l2src device=$DEVICE num-buffers=300 ! $INPUT_CAPS ! queue ! vpuenc_h264
    ! $ $MUXER ! filesink location=output.$EXTENSION
```

- $DEVICE could be set to /dev/video, /dev/video0, or /dev/video1 according to the system video input device.
- $INPUT_CAPS should be set to 'video/x-raw,format=(string)NV12,width=1920,height=1080,framerate=(fraction)30/1'.
- $MUXER can be set as to qtmux, matroskamux, mp4mux, avimux, or flvmux.
- $EXTENSION is filename extension according to the muxer type.

7.3.7 Audio/Video recording

This is an example of a command used to record audio and video together:

```
$GSTL -e imxv4l2src device=$DEVICE ! $INPUT_CAPS ! queue ! vpuenc_h264 ! queue
    ! mux. pulsesrc ! 'audio/x-raw, rate=44100, channels=2' ! imxmp3enc ! queue
    ! mux. $MUXER name=mux ! filesink location= output.$EXTENSION
```

- $INPUT_CAPS should be set to 'video/x-raw, format=(string)NV12, width=1920, height=1080, framerate=(fraction)30/1'.
- $MUXER can be set as to qtmux, matroskamux, mp4mux, avimux, or flvmux.

Common parameters are as follws:

- -e indicates to send EOS when the user presses **Ctrl+C** to avoid output corruption.
- $EXTENSION is the filename extension according to the multiplexer type.

7.3.8 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.

7.3.9 Recording the TV-in source

The TV-in source plugin gets video frames from the TV decoder. It is based on the V4l2 capture interface. A command line example is as follows:

```
gst-launch-1.0 v4l2src ! autovideosink
```

NOTE

The TV decoder is ADV7180. It supports NTSC and PAL TV mode. The output video frame is interlaced, so the sink plugin needs to enable deinterlace. The default value of v4l2sink deinterface is True.

7.3.10   Web camera

The following command line is an example of how to record and transfer web camera input.

```
$GSTL v4l2src device=/dev/video1 ! vpuenc_h264 ! rtph264pay ! udpsink host=$HOST_IP
```

HOST_IP is the IP/multicast group to send the packets to.

This command line is an example of how to receive and display web camera input.

```
$GSTL udpsrc ! buffer-size=204800 (example number) application/x-rtp ! rtph264depay !
vpudec ! v4l2sink
```

7.3.11   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
  ```

7.3.12   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.

7.3.13 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
```

7.3.14 RTSP streaming server

The RTSP streaming server use case is based on the open source gst-rtsp-server package. It uses the i.MX aiurdemux plugin to demultiplex the file to audio or video elementary streams and to send them out through RTP. Start the RTSP streaming server on one board, and play it on another board with the RTSP streaming playback commands.

The gst-rtsp-server package is not installed by default in the Yocto Project release. Follow these steps to build and install it.

1. Enable the layer meta-openembedded/meta-multimedia:

   Add the line `BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-multimedia"` to the configuration file `$yocto_root/build/conf/bblayers.conf`.

2. Include gst-rtsp-server into the image build:

   Add the line `IMAGE_INSTALL_append += "gstreamer1.0-rtsp-server"` to the configuration file `$yocto_root/build/conf/local.conf`.

3. Run `bitbake` for your image to build with gst-rstp-server.
4. You can find the test-uri binary in the folder:

   $yocto_root/build/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/gstreamer1.0-rtsp-server/$version/build/examples/.libs

5. Flash the image.

   Copy test-uri into `/usr/bin` in the rootfs on your board and assign execute permission to it.

Some information on running the tool is as follows:

- Command:

  ```
  test-uri $RTSP_URI
  ```

  For example:

  ```
  test-uri file:///home/root/temp/TestSource/mp4/1.mp4
  ```
- Server address:

  ```
  rtsp://$SERVER_IP:8554/test
  ```

  For example:

  ```
  rtsp://10.192.241.106:8554/test
  ```
- Client operations supported are Play, Stop, Pause, Resume, and Seek.

7.3.15 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.

<u>Color Space Conversion (CSC)</u>

```
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
```

Deinterlacing with i.MX with IPU

```
gst-launch-1.0 playbin uri=file://$FILE video-sink="imxvideoconvert_ipu deinterlace=3 !
ximagesink display=:0 sync=false"
```

Transcoding with i.MX with VPU

```
gst-launch-1.0 filesrc location=$FILE.mp4 typefind=true ! video/quicktime ! aiurdemux !
vpudec ! imxvideoconvert_ipu ! video/x-raw,format=NV12,width=1280,height=720 ! vpuenc_h263 !
avimux ! filesink location=$FILE.avi
```

Combination with i.MX with IPU or VPU

It is possible to combine CSC, resize, rotate, and deinterlace at one time. Both of imxvideoconvert_ipu and imxvideoconvert_g2d can be used at the same time in a pipeline. The following is an example:

```
gst-launch-1.0 videotestsrc ! video/x-raw,format=I420,width=1280,height=800,interlace-
mode=interleaved ! imxvideoconvert_ipu rotation=2 deinterlace=3 ! video/x-
raw,format=NV12,width=800,height=600 ! vpuenc_h264 ! vpudec ! imxvideoconvert_g2d
rotation=3 ! video/x-raw,format=RGB16,width=640,height=480 ! ximagesink sync=false display=:0
```

7.3.16   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
  ```

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

```
        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
```

7.4 PulseAudio input/output settings

If PulseAudio is installed in the rootfs, the PulseAudio input/output settings may need to be set.

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.

7.5 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>
```

8 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.

8.1 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.

8.2 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

8.3 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

8.4 Qt 5

Qt 5 is built into the Linux image in the Yocto Project environment with the command `bitbake imx-image-full`. For more details on Qt enablement checkout the README in the meta-imx repo and the IMX Yocto User's Guide.

9 Security

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

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

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
```

10   Connectivity

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

10.1   Connectivity for Bluetooth wireless technology and Wi-Fi

Bluetooth and Wi-Fi are supported on i.MX through on-board chip solutions and external hardware. The following table lists the various on-board chips and external solutions. When i.MX uses the Muarata baseboard, it requires a hardware modification on boards.

Table 45.   On-board chips and external solutions for Bluetooth and Wi-Fi support

SoC	On-board chip	PCIe	uSD-to-M.2 baseboard
8QuadXPlus	-	1CX/1FD M.2 Murata	-
8QuadMax	-	1CX/1FD M.2 Murata	-
8MQuad	Cypress 1CX	1CX/1FD M.2 Murata	-
8MMini	Cypress 1MW	1CX/1FD M.2 Murata	-
8MMini	Qualcomm 1PJ	-	-
7ULP	Cypress 1DX	-	-
7Dual	Broadcom ZP	-	Cypress 1MW
6QuadPlus/Quad/Dual/Solo	-	-	Cypress 1MW
6SLL/6UltraLite/6ULL/6ULZ	-	-	Cypress 1MW

Cypress and Murata solutions are the official supported solution for all i.MX except for the i.MX 8M Mini LPDDR4 EVK board which has one board that supports the Murata 1PJ module based on Qualcomm QCA9377-3.

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

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.

The wireless driver supports wpa_supplicant, which is a WEP/WPA/WPA2 encryption authenticated tool.
- Wi-Fi driver
    - Cypress FMAC driver supports Murata 1MW/TypeZP/1DX that supports SDIO interface, Murata 1FD/1CX that supports PCIe interface.
    - Qualcomm LEA-2.0 Wi-Fi driver supports Murata 1PJ that supports SDIO interface.
- Firmware

    The NXP release package already includes all Cypress and Qualcomm Wi-Fi/Bluetooth firmware. It requires to accept NXP license.

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
```

To run Bluetooth with BlueZ stack, execute the following commands:

```
hciattach <device> <type> 3000000 flow -b -t 120
hciconfig hci0 up
hciconfig hci0 piscan
hciconfig -a
hcitool dev
```

NOTE
- Device: /dev/ttymxcN or /dev/ttyLPN.
- Different boards have different devices.
- Type: qca for Qualcomm, bcm43xx for Cypress.

The i.MX 6 boards require board rework to support the bluetooth WiFI enablement as well as running with the btwifi device tress. Below is a list of the hardware modifications required and possibly conflicts caused by these modifications.

- i.MX 6QuadPlus/Quad/Dual/DualLite/Solo: Please refer to: https://community.nxp.com/docs/DOC-94235. Note this change will have a pin conflict with: EPDC/SPI-NOR/GPIO-LED
- i.MX 6SoloX: Install R328, disconnect R327. Connect with SD2 slot and BLUETOOTH CABLE CONNECTOR J19. It has no Pin conflict with other modules.
- i.MX 6SLL: Install R127, double check to ensure R126,R128 is installed. Connect with SD3 slot and BLUETOOTH CABLE CONNECTOR J4. It has no Pin conflict with other modules.
- i.MX 6UL/ULL/ULZ: Install R1701, It has no Pin conflict with other modules

10.2 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>;
```

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

```
                    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.

11 Xen

11.1 Xen overview

Xen project is a hypevisor using a microkernel design, providing services that allow multiple operating systems to run on the same hardware with hardware virtualization capability. It is widely used on Cloud and is also used in embedded systems.

11.2 Basic architecture

The following figure shows the basic architecture of Xen.
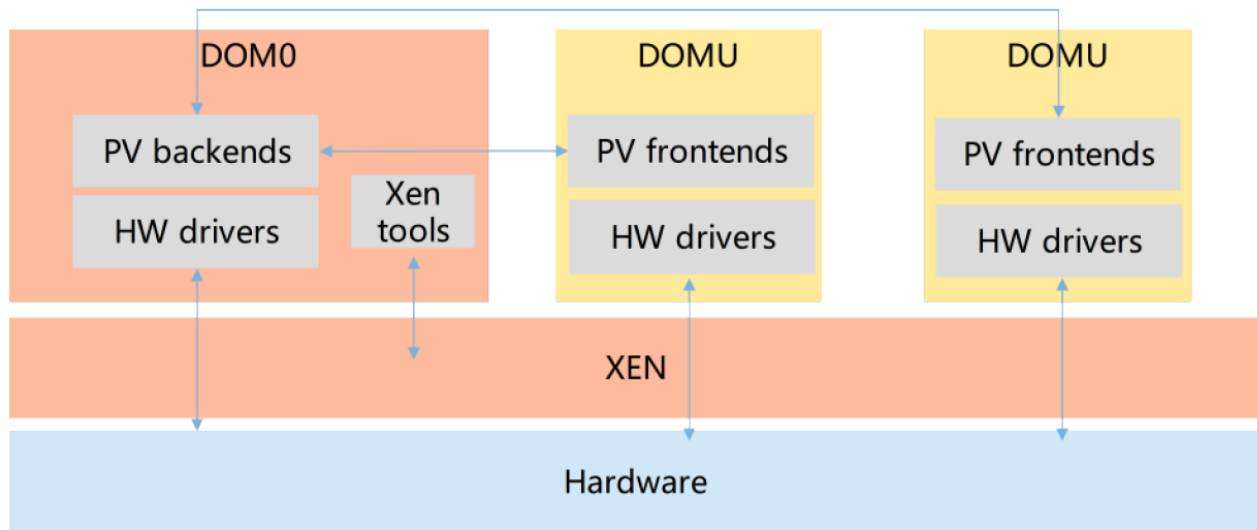
Figure 3. Basic architecture of Xen

11.3   Xen xl

For detailed Xen xl usage, see https://xenbits.xen.org/docs/unstable/man/xl.cfg.5.html.

11.4   How to boot multiple operating systems on i.MX 8QuadMax EVK

To boot two Yocto operating systems, partition the SD card (no less than 16 GB) with three partitions. The first partition is FAT, used to hold Xen, image, and imx8qm-mek-dom0.dtb. The second and third partitions are used to hold Yocto rootfs.

In the U-Boot stage:
   - To boot from the SD card, run `xenmmcboot`.
   - To boot from the network, run `xennetboot`.

After the first Linux OS is boot up, perform the following steps:
   1. Copy `Image` and `imx8qm-mek-domu.dtb` to /home/root/xen.
   2. Create a new file named `domu.cfg`. See below. Then, xl creates domu.cfg.
   3. Use `xl list` to view the running domains.
   4. Use `xl console DomU` to attach to the console of DomU.
   5. Press **ctrl +]** to exit the console of DomU.

The content of the `domu.cfg` file is as follows:

```
kernel = "/home/root/xen/Image"
device_tree = "/home/root/xen/imx8qm-mek-domu.dtb"
disk = [ 'format=raw,access=rw,vdev=xvda,target=/dev/mmcblk1p3' ]
cmdline = "no_console_suspend console=hvc0 root=/dev/xvda rw"
name = "DomU"
on_reboot="soft-reset"
memory = "3584"
vcpus = 1
cpus = ['3']
dtdev = [
```

```
#usbotg3:
'/usb3@5b110000',
#camera:
'/camera/isi@58100000',
#cryto:
'/caam@0x31400000',
#vpu:
'/vpu_decoder@2c000000',
#usdhc1:
'/usdhc@5b010000',
#dpu2:
'/dpu@57180000',
#gpu_3d1:
'/gpu@54100000',
#usb_otg1:
'/usb@5b0d0000',
#usbphy1:
#'/usbphy@0x5b100000',
#edma01:
'/dma-controller1@5a1f0000',
#cm41:
'/cm41@1',
#pciea:
'/pcie@0x5f000000',
]
iomem = [
#usbotg3:
"0x5b110,0x10@0x5b110",
"0x5b130,0x10@0x5b130",
"0x5b140,0x10@0x5b140",
"0x5b160,0x40@0x5b160",
"0x5b120,0x10@0x5b120",
#usbotg3_lpcg:
"0x5b280,0x1@0x5b280",
#camera
#isi_lpcg:
"0x58500,0x1@0x58500",
"0x58510,0x1@0x58510",
"0x58520,0x1@0x58520",
"0x58530,0x1@0x58530",
#isi_0:
"0x58100,0x10@0x58100",
#isi_1:
"0x58110,0x10@0x58110",
#isi_2:
"0x58120,0x10@0x58120",
#isi_3:
"0x58130,0x10@0x58130",
#mipi_csi_0:
"0x58227,0x1@0x58227",
"0x58221,0x1@0x58221",
#i2c0_mipi_csi0:
"0x58226,0x1@0x58226",
#irqsteer_csi0:
"0x58220,0x1@0x58220",
#gpio0_mipi_csi0:
"0x58222,0x1@0x58222",
#lpcg:
"0x58223,0x1@0x58223",
"0x58580,0x1@0x58580",
#end camera
#cryto:
"0x31400,0x400@0x31400",
#caam_sm:
"0x31800,0x10@0x31800",
#vpu mu_m0:
"0x2d000,0x10@0x2d000",
#vpu mu1_m0:
"0x2d020,0x10@0x2d020",
#vpu mu2_m0:
```

```
"0x2d040,0x10@0x2d040",
#vpu_decoder:
"0x2c000,0x1000@0x2c000",
#vpu_decoder_csr:
"0x2d080,0x1@0x2d080",
#vpu_decoder:
"0x2d000,0x1000@0x2d000",
#vpu_reserved:
"0x84000,0x2000@0x84000",
"0x86000,0x2000@0x86000",
"0x90400,0x1000@0x90400",
"0x91400,0x1000@0x91400",
"0x94400,0x1800@0x94400",
#hsio_pcie_x2_lpcg:
"0x5f050,0x10@0x5f050",
#hsio_pcie_x1_lpcg:
"0x5f060,0x10@0x5f060",
#hsio_phy_x2_lpcg:
"0x5f080,0x10@0x5f080",
#hsio_pcie_x2_crr2:
"0x5f0c0,0x10@0x5f0c0",
#hsio:
"0x5f080,0xf0@0x5f080",
#pciea:
"0x5f000,0x10@0x5f000",
"0x6ff00,0x80@0x6ff00",
#mu2:
"0x5d1d0,0x10@0x5d1d0",
#usb_otg1:usbmisc1:
"0x5b0d0,0x1@0x5b0d0",
#usbphy1:
"0x5b100,0x1@0x5b100",
#usblpcg:
"0x5b270,0x1@0x5b270",
#usdhc1:
"0x5b010,0x10@5b010",
#usdhc1_lpcg:
"0x5b200,0x1@5b200",
#lvds_region2 irqsteer_lvds2:
"0x57240,0x10@0x57240",
#dpu2_irqsteer:
"0x57000,0x10@0x57000",
#dpu2:
"0x57180,0x40@0x57180",
#pixel_combiner2:
"0x57020,0x10@0x57020",
#dc_1_lpcg:
"0x57010,0x10@0x57010",
#gpu_3d1:
"0x54100,0x40@0x54100",
#prg, dpr:
"0x57040,0xf0@0x57040",
#edma01:
"0x5a2e0,0x20@0x5a2e0",
#lpuart1_lpcg:
"0x5a470,0x10@0x5a470",
#lpuart1:
"0x5a070,0x1@0x5a070",
#mu6_lpcg1:
"0x5d610, 0x10@0x5d610",
#rpmsg1:
"0x90000, 0x400@0x90000",
#mu_rpmsg1_A:
"0x5d210, 0x10@0x5d210",
#The following iomem is to let CM41 with SMMU enabled work correctly
#mu_rpmsg1_B:
"0x5d2a0, 0x10@0x5d2a0",
#mu6_lpcg1_b:
"0x5d6a0, 0x10@0x5d6a0",
#mu7_lpcg1_b:
```

```
"0x5d6b0, 0x10@0x5d6b0",
#irqsteer_cm41:
"0x51080, 0x10@0x51080",
#lpuart2_lpcg:
"0x5a480,0x10@0x5a480",
#lpuart2:
"0x5a080,0x1@0x5a080",
]
irqs = [
#usbotg3,
303,
#gpio4:
172,
#irqsteer_csi0
352,
#isi
329,
330,
331,
332,
#crypto
180,
485,
486,
#vpu
504,
505,
506,
#pciea
102,
104,
#usdhc1
264,
210,
#usbotg1
299,
#irqsteer_lvds1
90,
#dpu2
184, 185, 186, 187, 188, 189, 190, 191, 194, 195,
#gpu_3d1
97,
#lpuart1,
378,
#edma01,
468, 469,
#mu_rpmsg1,
217,
]
serial="pty"
#vif = [ 'bridge=br0' ]
vi2c = ["backend=0,be-adapter=5a800000.i2c,slaves=0x51;0x44;0x1e;0x20;0x60"]
```

12 NXP eIQ Machine Learning

The NXP® eIQ™ for i.MX toolkit provides a set of libraries and development tools for machine learning applications targeting NXP microcontrollers and application processors. The toolkit is contained in the meta-imx/meta-ml layer. The following sections describe how to run OpenCV, Arm Compute Library, TensorFlow Lite, Arm NN, and ONNX demos on i.MX Linux BSPs.

12.1 OpenCV machine learning demos

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

12.1.1 Downloading OpenCV demos

OpenCV DNN demos are located at /usr/share/OpenCV/samples/bin/. Input data, model configurations, and model weights are not located in this directory, because of size. These files should be downloaded to the device before running the demos:

- Download the opencv_extra-4.1.2.zip package from this link: https://github.com/opencv/opencv_extra/releases/tag/4.1.2.
- Decompress the file `opencv_extra-4.1.2` to the SD card folder (/home/root or /tmp).
- Go to `opencv_extra-4.1.2/testdata/dnn/` and run `python download_models.py`. The script downloads NN models, configuration files, and input images for some OpenCV examples. This operation may take a while. Copy these dependencies to

  `/usr/share/OpenCV/samples/bin`
- Download the configuration model file from https://github.com/opencv/opencv/blob/4.1.2/samples/dnn/models.yml. The model.yml file contains preprocessing parameters for some DNN examples, which accepts

  `'--zoo'`

  parameter. Copy the model file to

  `/usr/share/OpenCV/samples/bin`

12.1.2 OpenCV DNN demos

The OpenCV DNN module implements an inference engine and does not provide any functionalities for neural network training.

12.1.2.1 Image classification demo

This demo performs image classification using a pretrained SqueezeNet network. Demo dependencies (taken from opencv_extra-4.1.2 package):

- dog416.png
- squeezenet_v1.1.caffemodel
- squeezenet_v1.1.prototxt

Other demo dependencies:

- classification_classes_ILSVRC2012.txt from

  `/usr/share/OpenCV/samples/data/dnn`
- models.yml from github

Run the C++ example with image input from the default location:

```
./example_dnn_classification --input=dog416.png --zoo=models.yml squeezenet
```
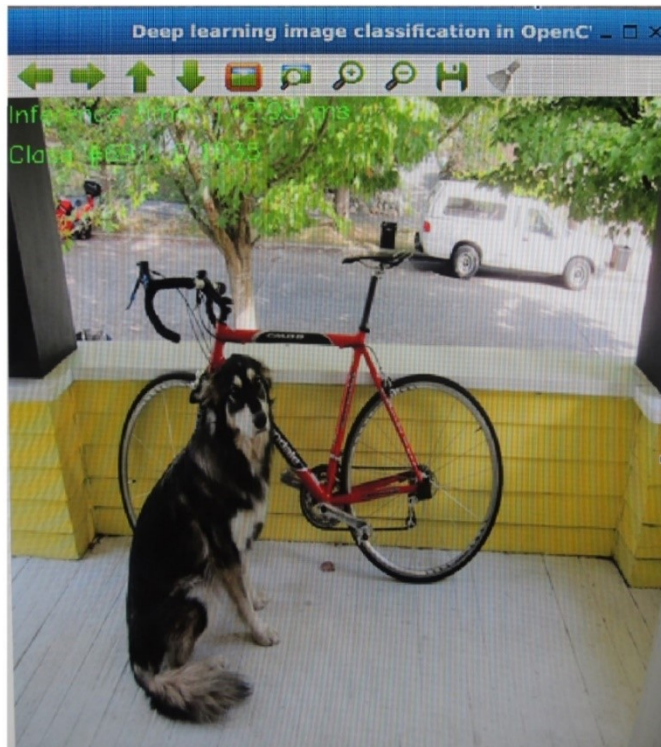
Figure 4. Image classification graphics output

Run the C++ example with the live camera input from the default location

```
./example_dnn_classification --zoo=models.yml squeezenet
```

NOTE

Only port 0 is supported for the live camera input.

12.1.2.2   YOLO object detection example

The YOLO demo performs object detection using You Only Look Once (YOLO) detector (https://arxiv.org/pdf/
1506.02640v5.pdf). It detects objects on camera/video/image. Find out more information about this demo, see https://
docs.opencv.org/4.1.2/da/d9d/tutorial_dnn_yolo.html.

Demo dependencies (taken from opencv_extra-4.1.2 package):

- dog416.png
- yolov3.weights
- yolov3.cfg

Other demo dependencies:

- models.yml
- object_detection_classes_yolov3.txt from

```
/usr/share/OpenCV/samples/data/dnn
```

Running the C++ example with image input from the default location:

```
./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 -input=dog416.png -
rgb -zoo=models.yml yolo
```
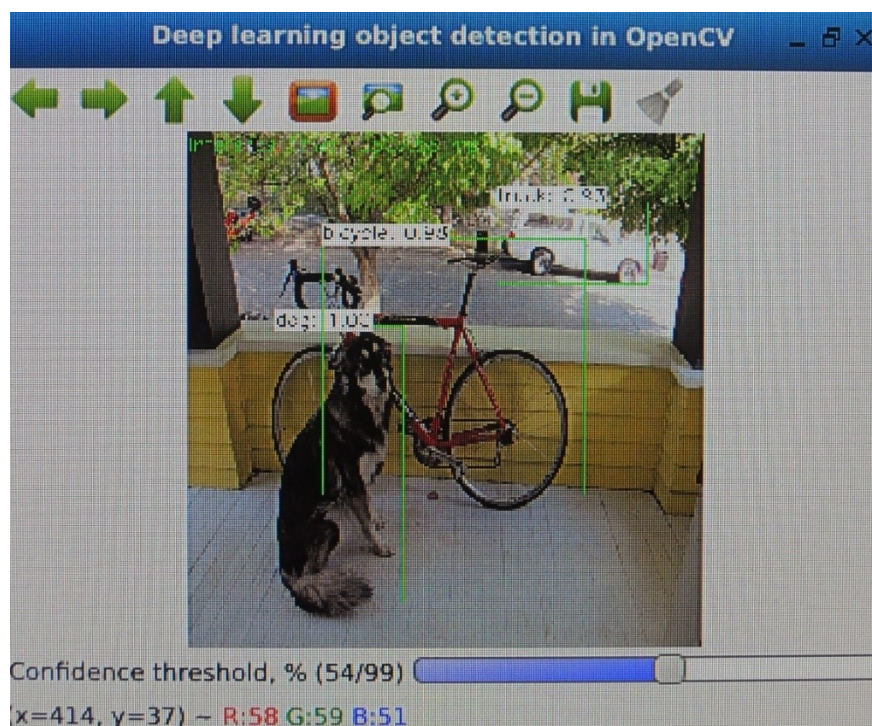
Figure 5. YOLO object detection graphics output

Run the C++ example with the live camera connected to the port 3:

```
./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 --device=3 -rgb -
zoo=models.yml yolo
```

NOTE

> Choose the right port where the camera is currently connected. Use `ls /dev/video*` command to check it.
>
> Running this example with live camera input is quite slow, because of running example on the CPU only.

12.1.2.3 Image segmentation demo

The image segmentation means dividing the image into groups of pixels based on some criteria grouping based on color, texture, or some other criteria. Demo dependencies (taken from opencv_extra-4.1.2 package):

- dog416.png
- fcn8s-heavy-pascal.caffemodel
- fcn8s-heavy-pascal.prototxt

Other demo dependencies are models.yml. Running the C++ example with image input from the default location:

```
./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --input=dog416.png --
zoo=models.yml fcn8s
```

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020
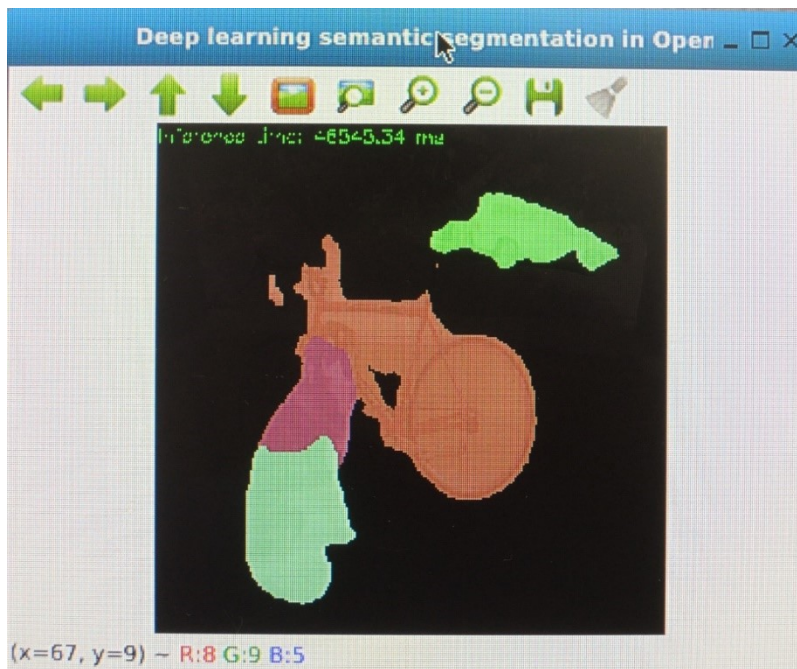
Figure 6. Image segmentation graphics output

Run the C++ example with the live camera connected to the port 3:

```
./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --device=3 --
zoo=models.yml fcn8s
```

NOTE

Choose the right port where the camera is currently connected. Use `ls /dev/video*`
command to check it.

Running this example with live camera input is quite slow, because of running this
example on the CPU only.

12.1.2.4 Image colorization demo

This sample demonstrates recoloring grayscale images with DNN. The demo supports input images only, not the live camera input. Demo dependencies (taken from opencv_extra-4.1.2 package) are:

- colorization_release_v2.caffemodel
- colorization_deploy_v2.prototxt

Other demo dependencies are basketball1.png. Running the C++ example with image input from the default location:

```
./example_dnn_colorization --model=colorization_release_v2.caffemodel --
proto=colorization_deploy_v2.prototxt --image=../data/basketball1.png
```

Figure 7. Image colorization graphics output

12.1.2.5   Human pose detection demo

This application demonstrates human or hand pose detection with a pretrained OpenPose DNN. The demo supports input images only and no live camera input. Demo dependencies (taken from opencv_extra-4.1.2 package):

- grace_hopper_227.png
- openpose_pose_coco.caffemodel
- openpose_pose_coco.prototxt

Running the C++ example with image input from the default location:

```
./example_dnn_openpose --model=openpose_pose_coco.caffemodel --
proto=openpose_pose_coco.prototxt --image=grace_hopper_227.png --width=227 --height=227 --
dataset=COCO
```
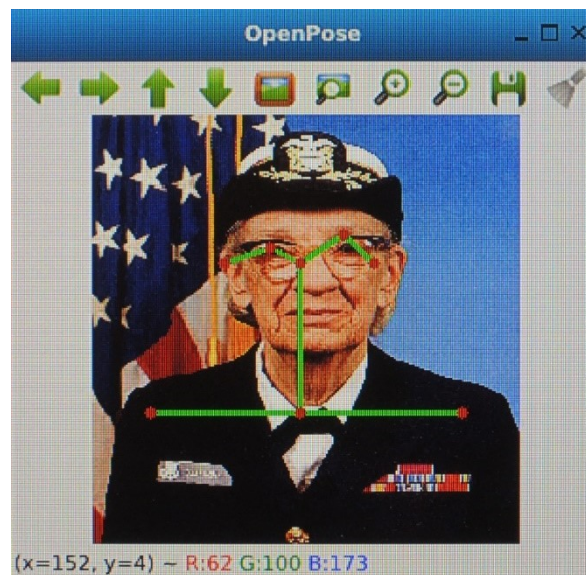

Figure 8. Human pose estimation graphics output

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

12.1.2.6   CNN image classification example

This demo performs image classification using a pretrained SqueezeNet network. The demo supports input images only, not the live camera input. Demo dependencies (taken from opencv_extra-4.1.2 package):

- space_shuttle.jpg

Other demo dependencies:

- Download SqueezeNet.caffemodel model weight file form this link: https://github.com/kvmanohar22/caffe/tree/obj_detect_loss/proto
- Download SqueezeNet_deploy.prototxt model definition file from this link: https://github.com/opencv/opencv_contrib/tree/4.1.2/modules/dnn_objdetect/samples/data

Running the C++ example with image input from the default location:

```
./example_dnn_objdetect_image_classification SqueezeNet_deploy.prototxt
SqueezeNet.caffemodel space_shuttle.jpg
```

Running the model on the space_shuttle.jpg image produces the following text results in the console:

```
Best class Index: 812
Time taken: 0.649153
Probability: 15.8467
```

12.1.2.7   Text detection

This demo is used for text detection in the image using EAST algorithm. Demo dependencies are as follows:

- frozen_east_text_detection.pb (taken from the opencv_extra-4.1.2 package)
- imageTextN.png

Running the C++ example with image input from the default location:

```
./example_dnn_text_detection --model=frozen_east_text_detection.pb --input=../data/
imageTextN.png
```
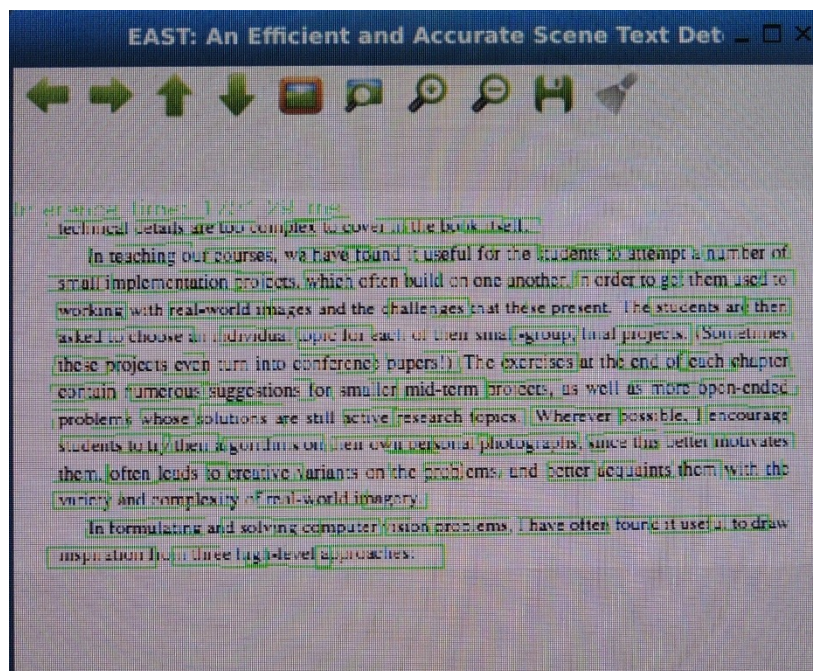
Figure 9. Text detection graphics output

NOTE
This example accepts the PNG image format only.

Running the C++ example with the live camera input from the default location:

```
./example_dnn_text_detection --model=frozen_east_text_detection.pb
```

NOTE
Only port 0 is supported for the live camera input.

12.1.3   OpenCV classical machine learning demos

After deploying OpenCV on the target device, Non-Neural Networks demos are installed in the rootfs in

```
/usr/share/OpenCV/samples/bin/
```

12.1.3.1   SVM introduction

This example demonstrates how to create and train an SVM model using training data. Once the model is trained, labels for test data are predicted. The full description of the example can be found in tutorial_introduction_to_svm. For displaying the result, an image with Qt5 enabled is required.

After running the demo, the graphics result is shown on the screen:

```
./example_tutorial_introduction_to_svm
```

Result:

- The code opens an image and shows the training examples of both classes. The points of one class are represented with white circles, and other class uses black points.

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020

- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into a blue region and a green region. The boundary between both regions is the optimal separating hyperplane.
- Finally, the support vectors are shown using gray rings around the training examples.
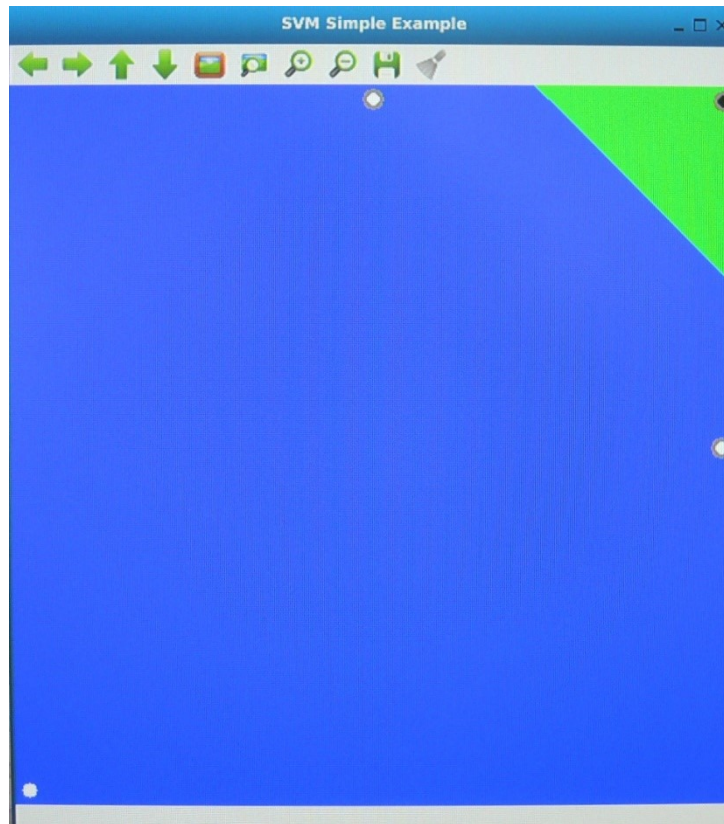
Figure 10. SVM introduction graphics output

12.1.3.2   SVM for non-linearly separable data

This example deals with non-linearly separable data and shows how to set parameters of SVM with linear kernel for this data. For more details, go to Support Vector Machines for Non-Linearly Separable Data.

After running the demo, the graphics result is shown on the screen (it requires Qt5 support):

```
./example_tutorial_non_linear_svms
```

Result:

- The code opens an image and shows the training data of both classes. The points of one class are represented with light green, the other class uses light blue points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into blue green regions. The boundary between both regions is the separating hyperplane. Since the training data is non-linearly separable, some of the examples of both classes are misclassified; some green points lay on the blue region and some blue points lay on the green one.
- Finally, the support vectors are shown using gray rings around the training examples.
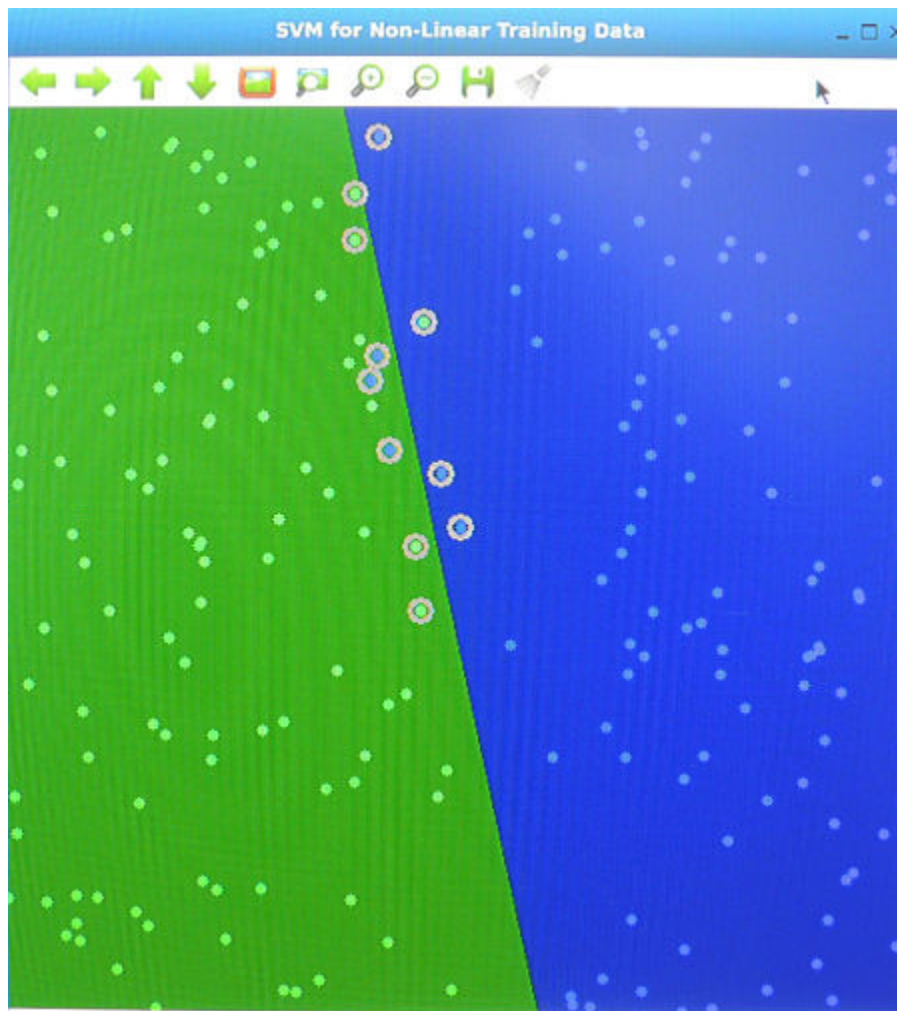
Figure 11. SVM for Non-linear training data

12.1.3.3   Prinicipal Component Analysis (PCA) introduction

Principal Component Analysis (PCA) is a statistical method that extracts the most important features of a dataset. In this tutorial you will learn how to use PCA to calculate the orientation of an object. For more details, check the OpenCV tutorial Introduction to Principal Component Analysis (PCA).

After running the demo, the graphics result is shown on the screen (it requires Qt 5 support):

```
./example_tutorial_introduction_to_pca ../data/pca_test1.jpg
```

Results:

- Open an image (loaded from ../data/pca_test1.jpg)
- Find the orientation of the detected objects of interest
- Visualizes the result by drawing the contours of the detected objects of interest, the center point, and the x-axis, y-axis regarding the extracted orientation.

i.MX Linux® User's Guide, Rev. L5.4.3_1.0.0, 03/2020
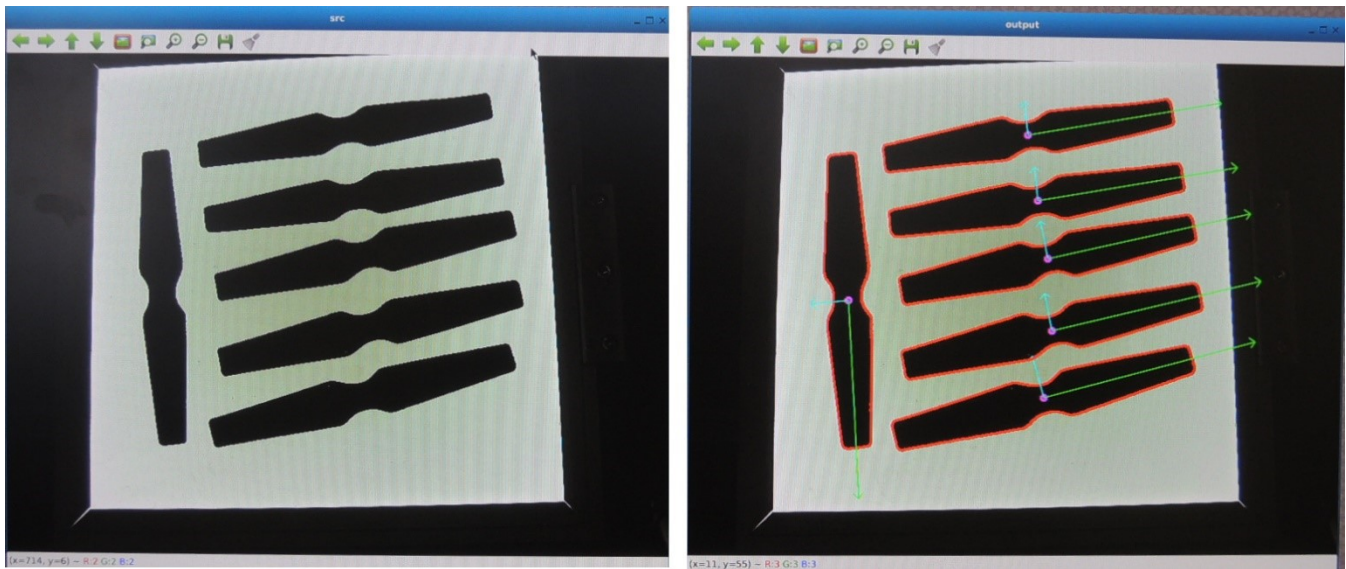
Figure 12. PCA graphics output

12.1.3.4   Logistic regression

In this sample, logistic regression is used for prediction of two characters (0 or 1) from an image. First, every image matrix is reshaped from its original size of 28x28 to 1x784. A logistic regression model is created and trained on 20 images. After training, the model can predict labels of test images. Source code is located on the link logistic_regression and can be run by typing the following command:

Demo dependencies (preparing the train data files):

```
wget https://raw.githubusercontent.com/opencv/opencv/4.1.2/samples/data/data01.xml
```

After running the demo, the graphics result is shown on the screen (it requires Qt5 support):

```
./example_cpp_logistic_regression
```

The results are as follows:

- Training and test data are shown.
- Comparison between original and predicted labels is displayed.

The console text output is as follows (the trained model reaches 95% accuracy):

```
original vs predicted:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
saving the classifier to NewLR_Trained.xml
loading a new classifier from NewLR_Trained.xml
predicting the dataset using the loaded classifier...done!
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
```
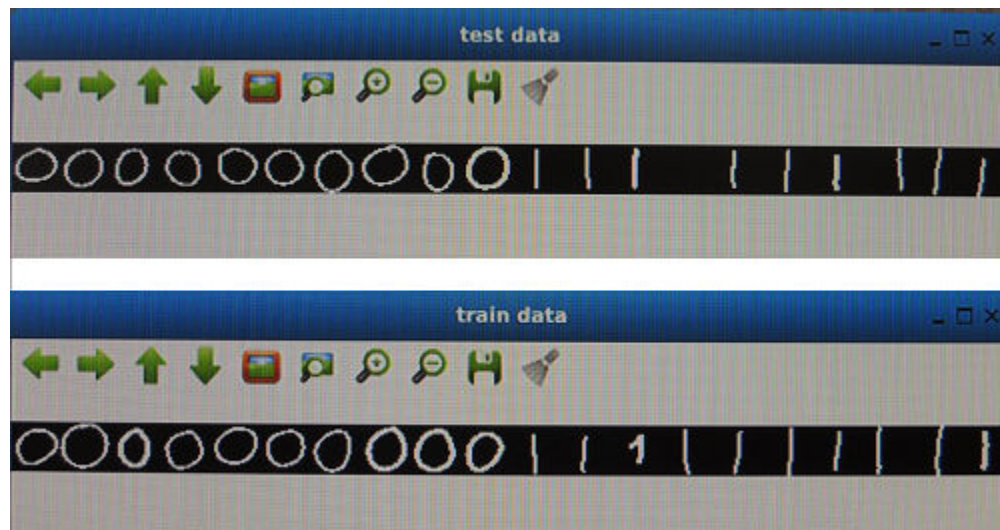
Figure 13. Logistic regression graphics output

12.2   Arm Compute Library

Arm Compute Library is a collection of low-level functions optimized for Arm CPU and GPU architectures targeted at image processing, computer vision, and machine learning.

12.2.1   Running a DNN with random weights and inputs

Arm compute library comes with examples for most common DNN architectures like: AlexNet, MobileNet, ResNet, Inception v3, Inception v4, Squeezenet, etc.

All available examples can be found in this example build location:

```
/usr/share/arm-compute-library/build/examples
```

Each model architecture can be tested with graph_[dnn_model] application.

For example, to run the MobileNet v2 DNN model with random weights, run the example application without any argument:

```
$: ./graph_mobilenet_v2
```

The application will create the DNN with random weights and run it with random inputs. If inference finishes, the "Test passed" message should be printed.

12.2.2   Running AlexNet using graph API

In 2012, AlexNet shot to fame when it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual challenge that aims to evaluate algorithms for object detection and image classification. AlexNet is made up of eight trainable layers: five convolution layers and three fully connected layers. All the trainable layers are followed by a ReLu activation function, except for the last fully connected layer, where the Softmax function is used.

Location of the C++ AlexNet example implementation using the graph API is in this folder:

```
/usr/share/arm-compute-library/build/examples
```

Demo instructions:

- Download the archive file to the example location folder from this link: compute_library_alexnet.zip.
- Create a new sub-folder and unzip the file.
- Run the example with command line arguments from the default location:

```
$: mkdir assets_alexnet
$: unzip compute_library_alexnet.zip -d assets_alexnet
```

Set environment variables for execution:

```
$: export LD_LIBRARY_PATH=/usr/share/arm-compute-library/build/examples/
$: export PATH_ASSETS=/usr/share/arm-compute-library/build/examples/assets_alexnet/
$: ./graph_alexnet --data=$PATH_ASSETS --image=$PATH_ASSETS/go_kart.ppm --labels=
$PATH_ASSETS/labels.txt
```

The output of the successful classification is as follows:

```
---------- Top 5 predictions ----------

0.6988 - [id = 573], n03444034 go-kart
0.1242 - [id = 751], n04037443 racer, race car, racing car
0.0351 - [id = 518], n03127747 crash helmet
0.0290 - [id = 817], n04285008 sports car, sport car
0.0099 - [id = 981], n09835506 ballplayer, baseball player

Test passed
```

12.3 TensorFlow Lite

TensorFlow Lite is a light-weight version of and a next step from TensorFlow. TensorFlow Lite is an open-source software library focused on running machine learning models on mobile and embedded devices (available at www.tensorflow.org/lite). It enables on-device machine learning inference with low latency and small binary size. TensorFlow Lite also supports hardware acceleration using Android™ OS neural network APIs.

12.3.1 Running image classification example

This simple example is pre-installed by default on the prepared image with machine learning enablement. Its name is 'label_image' and classifies images of clothing, like hat, shirts, etc. A proper model file for this example is not included on the target image by default, because of its size. The example binary file location is:

```
/usr/bin/tensorflow-lite-1.13.2/examples
```

Demo instructions

- Download the TensorFlow model file to the example folder. It can be the same model file as the previous benchmark example in opencv. Download it from here: http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_1.0_224_quant.tgz
- Unpack the model file:

  ```
  $: tar -xzvf mobilenet_v1_1.0_224_quant.tgz
  ```
- To run the inference using the CPU module, use the following command line arguments:

  ```
  $: ./label_image -m mobilenet_v1_1.0_224_quant.tflite -t 1 -i grace_hopper.bmp -l
  labels.txt
  ```

The output of the successful classification for the 'grace_hopper' input image is as follows:

```
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
invoked
average time: 330.473 ms
0.780392: 653 military uniform
0.105882: 907 Windsor tie
0.0156863: 458 bow tie
0.0117647: 466 bulletproof vest
0.00784314: 835 suit
```

To run the inference using the GPU module, use the following command line arguments:

```
$: ./label_image -m mobilenet_v1_1.0_224_quant.tflite -t 1 -i grace_hopper.bmp -l
labels.txt -a 1
```

The output with acceleration enabled should be the following:

```
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
invoked
first time: 1733.25 ms

invoked
average time: 19.803 ms
0.784314: 653 military uniform
0.105882: 907 Windsor tie
0.0156863: 458 bow tie
0.0117647: 466 bulletproof vest
0.00784314: 668 mortarboard
```

NOTE

> The TensorFlow Lite library uses the NN API driver implementation from the GPU driver for running the inference using the GPU module. The implemented NN API version is 1.0, which has limitations in supported tensor data types and operations, compared to the feature set of TensorFlow Lite. Therefore, some models may work without acceleration enabled, but may fail when using the NN API. For the full list of supported features, see the NN HAL versions section of the NN API documentation: https://source.android.com/devices/neural-networks#hal-versions.

> The first iteration of model inference using the NN API always takes many times longer, because of model graph initialization needed by the GPU module. The iterations following the graph initialization will be performed many times faster.

12.4 Arm NN

Arm NN SDK provides a set of tests which can also be considered as demos showing what Arm NN does and how to use it. They load neural network models of various formats (Caffe, TensorFlow, TensorFlowLite, ONNX), run the inference on a specified input data and output the inference result. Arm NN tests are built by default when building the Yocto image and are installed in the rootfs in /usr/bin. Note that input data, model configurations, and model weights are not distributed with Arm NN. The user must download them separately and make sure they are available on the device before running the tests. However, Arm NN tests don't come with a documentation. Input file names are hardcoded, so investigate the code to find out what input file names are expected.

To help in getting started with Arm NN, the following sections provide details about how to prepare the input data and how to run Arm NN tests. All of them use well known neural network models, therefore, with only few exceptions, such pretrained networks are available freely on the Internet. Input images, models, formats and their content was deduced using code analysis. However, this was not possible for all the tests, because either models are not publicly available or it is not possible to deduce clearly what input files are required by the application. General work-flow is first to prepare data on a host machine and then to deploy it on the board, where the actual Arm NN tests will be run.

The following sections assume that neural network model files are stored in a folder called models and input image files are stored in a folder called data. Both of them are created inside a folder called ArmnnTests. Create this folder structure on the larger partition using the following commands:

```
$ mkdir ArmnnTests
$ cd ArmnnTests
$ mkdir data
$ mkdir models
```

12.4.1 Caffe tests

Arm NN SDK provides the following set of tests for Caffe models:

```
/usr/bin/CaffeAlexNet-Armnn
/usr/bin/CaffeCifar10AcrossChannels-Armnn
/usr/bin/CaffeInception_BN-Armnn
/usr/bin/CaffeMnist-Armnn
/usr/bin/CaffeResNet-Armnn
/usr/bin/CaffeVGG-Armnn
/usr/bin/CaffeYolo-Armnn
```

Two important limitations might require preprocessing of the Caffe model file prior to running an Arm NN Caffe test. First, Arm NN tests require batch size to be set to 1. Second, Arm NN does not support all Caffe syntaxes, therefore some older neural network model files will require updates to the latest Caffe syntax.

Details about how to perform these preprocessing steps are described on Arm NN GitHub page. Install Caffe on the host. Also check Arm NN documentation for Caffe support.

For example, if a Caffe model has a batch size different from one or uses an older Caffe version defined by files model_name.prototxt and model_name.caffemodel, create a copy of the .prototxt file (new_model_name.prototxt), modify this file to use the new Caffe syntax and change the batch size to 1 and finally run the following python script:

```
import caffe

net = caffe.Net('model_name.prototxt', 'model_name.caffemodel', caffe.TEST)
new_net = caffe.Net('new_model_name.prototxt', 'model_name.caffemodel', caffe.TEST)
new_net.save('new_model_name.caffemodel')
```

The following sections explain how to run each of the tests, except for CaffeCifar10AcrossChannels-Armnn, CaffeVGG-Armnn, and CaffeYolo-Armnn. For the first and the second ones, a pretrained model is not publicly available. For the third one, it is very difficult to discover the exact content of the input image originally used by this test.

12.4.1.1 CaffeAlexNet-Armnn

Follow the steps below:

1. Download the model files:
   - https://raw.githubusercontent.com/BVLC/caffe/master/models/bvlc_alexnet/deploy.prototxt
   - http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel
2. Transform the network as explained in the Caffe Test section..
3. Rename bvlc_alexnet.caffemodel to bvlc_alexnet_1.caffemodel
4. Copy bvlc_alexnet_1.caffemodel to the models folder on the device
5. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: CaffeAlexNet-Armnn --data-dir=data --model-dir=models
```

12.4.1.2   CaffeInception_BN-Armnn

Follow the steps below:

1. Download the model files:
   - https://raw.githubusercontent.com/pertusa/InceptionBN-21K-for-Caffe/master/deploy.prototxt
   - http://www.dlsi.ua.es/~pertusa/deep/Inception21k.caffemodel
2. Transform the network as explained in the introductory part of Caffe Tests chapter.
3. Rename Inception21k.caffemodel to Inception-BN-batchsize1.caffemodel
4. Copy Inception-BN-batchsize1.caffemodel to the models folder on the device
5. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: CaffeInception_BN-Armnn --data-dir=data --model-dir=models
```

12.4.1.3   CaffeMnist-Armnn

Follow the steps below:

1. Download the model files:
   - https://raw.githubusercontent.com/BVLC/caffe/master/examples/mnist/lenet.prototxt
   - https://github.com/ARM-software/ML-examples/blob/master/armnn-mnist/model/lenet_iter_9000.caffemodel
2. Transform the network as explained in the introductory part of Caffe Tests chapter.
3. Download the two archives below and unpack them:
   - http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
   - http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
4. Copy lenet_iter_9000.caffemodel to the models folder on the device
5. Copy t10k-images.idx3-ubyte and t10k-labels.idx1-ubyte to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: CaffeMnist-Armnn --data-dir=data --model-dir=models
```

12.4.1.4   CaffeResNet-Armnn

Follow the steps below:

1. Download the model file for ResNet50 at https://onedrive.live.com/?authkey=%21AAFW2-FVoxeVRck&cid=4006CBB8476FF777&id=4006CBB8476FF777%2117895&parId=4006CBB8476FF777%2117887&o=OneUp
2. Rename RestNet-50-model.caffemodel to ResNet_50_ilsvrc15_model.caffemodel
3. Copy ResNet_50_ilsvrc15_model.caffemodel to the models folder.
4. Download the image file and copy it to the data folder: https://raw.githubusercontent.com/ameroyer/PIC/d136e9ceded0ceb700898725405d8eb7bd273bbe/val_samples/ILSVRC2012_val_00000018.JPEG
5. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: CaffeResNet-Armnn --data-dir=data --model-dir=models
```

12.4.2   TensorFlow tests

Arm NN SDK provides the following set of tests for TensorFlow models:

```
/usr/bin/TfCifar10-Armnn
/usr/bin/TfInceptionV3-Armnn
/usr/bin/TfMnist-Armnn
/usr/bin/TfMobileNet-Armnn
/usr/bin/TfResNext-Armnn
```

Prior to running the tests, the TensorFlow models need to be prepared for inference. This process is TensorFlow specific and uses TensorFlow tools. Therefore, TensorFlow must be installed on your host machine.

The following sections explain how to run each of the tests, except for TfResNext-Armnn and TfCifar10-Armnn, for which we couldn't find publicly available pretrained models.

12.4.2.1   TfInceptionV3-Armnn

Follow the steps below:

1. From host machine, generate the graph definition for the Inception model:

```
# model preparation
$: mkdir checkpoints
# clone the models repository
$: git clone https://github.com/tensorflow/models.git
$: cd models/research/slim/
# export the inference graph
$: python export_inference_graph.py --model_name=inception_v3 --output_file=../../../
checkpoints/inception_v3_inf_graph.pb
```
2. From the host machine, download the pretrained model and use TensorFlow tools to prepare it for inference. Note that "path_to_tensorflow_repo" refers to the path where you cloned or downloaded the TensorFlow repo.

```
$: cd ../../../checkpoints
# download and extract the checkpoint
$: wget http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz -qO- | tar
-xvz
# freeze the model
$: python path_to_tensorflow_repo/tensorflow/python/tools/freeze_graph.py --
input_graph=inception_v3_inf_graph.pb --input_checkpoint=inception_v3.ckpt --
input_binary=true --output_graph=inception_v3_2016_08_28_frozen.pb --
output_node_names=InceptionV3/Predictions/Reshape_1
```
3. Copy inception_v3_2016_08_28_frozen.pb to the models folder on the device
4. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
5. Find a .jpg file containing a dog. Rename it to Dog.jpg and copy it to the data folder on the device.
6. Find a .jpg file containing a cat. Rename it to Cat.jpg and copy it to the data folder on the device.
7. Run the test:

```
$: cd ArmnnTests
$: TfInceptionV3-Armnn --data-dir=data --model-dir=models
```

12.4.2.2   TfMnist-Armnn

Follow the steps below:

1. Download the model file https://raw.githubusercontent.com/ARM-software/ML-examples/master/armnn-mnist/model/simple_mnist_tf.prototxt.
2. Copy simple_mnist_tf.prototxt to the models folder on the device.
3. Download the two archives below and unpack them:
    *   http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
    *   http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
4. Copy t10k-images.idx3-ubyte and t10k-labels.idx1-ubyte to the data folder on the device.

5. Run the test:

```
$: cd ArmnnTests
$: TfMnist-Armnn --data-dir=data --model-dir=models
```

12.4.2.3 TfMobileNet-Armnn

Follow the steps below:

1. From host machine, download and unpack the model file: http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_1.0_224.tgz.
2. Copy mobilenet_v1_1.0_224_frozen.pb to the models folder on the device
3. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
4. Find a .jpg file containing a dog. Rename it to Dog.jpg and copy it to the data folder on the device.
5. Find a .jpg file containing a cat. Rename it to Cat.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: TfMobileNet-Armnn --data-dir=data --model-dir=models
```

12.4.3 TensorFlow Lite tests

Arm NN SDK provides the following test for TensorFlow Lite models:

```
/usr/bin/TfLiteInceptionV3Quantized-Armnn
/usr/bin/TfLiteInceptionV4Quantized-Armnn
/usr/bin/TfLiteMnasNet-Armnn
/usr/bin/TfLiteMobileNetSsd-Armnn
/usr/bin/TfLiteMobilenetQuantized-Armnn
/usr/bin/TfLiteMobilenetV2Quantized-Armnn
/usr/bin/TfLiteResNetV2-Armnn
/usr/bin/TfLiteVGG16Quantized-Armnn
/usr/bin/TfLiteResNetV2-50-Quantized-Armnn
```

The following sections explain how to run some of the tests. Some of the tests are excluded because either there is no public available model, or they need a huge amount of resources, more than what is available, on an i.MX 8 class embedded application processor.

12.4.3.1 TfLiteInceptionV3Quantized-Armnn

Follow the steps below:

1. From the host machine, download and unpack the model file: http://download.tensorflow.org/models/tflite_11_05_08/inception_v3_quant.tgz.
2. Copy inception_v3_quant.tflite to the models folder on the device
3. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
4. Find a .jpg file containing a dog. Rename it to Dog.jpg and copy it to the data folder on the device.
5. Find a .jpg file containing a cat. Rename it to Cat.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: TfLiteInceptionV3Quantized-Armnn --data-dir=data --model-dir=models
```

12.4.3.2 TfLiteMnasNet-Armnn

Follow the steps below:

1. From host machine, download and unpack the model file: http://download.tensorflow.org/models/tflite/mnasnet_1.3_224_09_07_2018.tgz.
2. Copy mnasnet_1.3_224/mnasnet_1.3_224.tflite to the models folder on the device.
3. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
4. Find a .jpg file containing a dog. Rename it to Dog.jpg and copy it to the data folder on the device.
5. Find a .jpg file containing a cat. Rename it to Cat.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: TfLiteMnasNet-Armnn --data-dir=data --model-dir=models
```

12.4.3.3   TfLiteMobilenetQuantized-Armnn

Follow the steps below:

1. From host machine, download the model file: http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_1.0_224_quant.tgz.
2. Copy mobilenet_v1_1.0_224_quant.tflite to the models folder on the device
3. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
4. Find a .jpg file containing a dog. Rename it to Dog.jpg and copy it to the data folder on the device.
5. Find a .jpg file containing a cat. Rename it to Cat.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: TfLiteMobilenetQuantized-Armnn --data-dir=data --model-dir=models
```

12.4.3.4   TfLiteMobilenetV2Quantized-Armnn

Follow the steps below:

1. From the host machine, download the model file: http://download.tensorflow.org/models/tflite_11_05_08/mobilenet_v2_1.0_224_quant.tgz.
2. Copy mobilenet_v2_1.0_224_quant.tflite to the models folder on the device.
3. Find a .jpg file containing a shark. Rename it to shark.jpg and copy it to the data folder on the device.
4. Find a .jpg file containing a dog. Rename it to Dog.jpg and copy it to the data folder on the device.
5. Find a .jpg file containing a cat. Rename it to Cat.jpg and copy it to the data folder on the device.
6. Run the test:

```
$: cd ArmnnTests
$: TfLiteMobilenetV2Quantized-Armnn --data-dir=data --model-dir=models
```

12.5   ONNX

ONNX Runtime provides a tool that can run the collection of standard tests provided in the ONNX model Zoo. The tool named onnx_test_runner is installed in /usr/bin.

ONNX models are available at https://github.com/onnx/models and consist mostly of scripts and test data for training and validation. Because some models require a lot of disk space, it is advised to store the ONNX test files on a larger partition, as described in the SD card image flashing section.

Here is an example with the steps required to run the squeezenet test, where the data is already pretrained using the script in the github repository:

- Download and unpack the squeezenet test archive located at https://s3.amazonaws.com/download.onnx/models/opset_8/squeezenet.tar.gz.

- Copy the squeezenet folder containing the model and test data on the device in case the previous step was not executed directly on the board, for example, to the /home/root folder.
- Run onnx_test_runner tool providing squeezenet folder path as a command line parameter:

```
$: ls /home/root/squeezenet/
model.onnx          test_data_set_11  test_data_set_5  test_data_set_9
test_data_set_0     test_data_set_2   test_data_set_6
test_data_set_1     test_data_set_3   test_data_set_7
test_data_set_10    test_data_set_4   test_data_set_8
$: onnx_test_runner /home/root/squeezenet/
result:
        Models: 1
        Total test cases: 12
                Succeeded: 12
                Not implemented: 0
                Failed: 0
        Stats by Operator type:
                Not implemented(0):
                Failed:
Failed Test Cases:
$:
```

13 Revision History

This table provides the revision history.

Table 46. 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 8M Quad
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 8M Quad GA
L4.9.88_2.0.0-ga	05/2018	i.MX 7ULP and i.MX 8M Quad GA release
L4.9.88_2.1.0_8mm-alpha	06/2018	i.MX 8M Mini Alpha release
L4.9.88_2.2.0_8qxp-beta2	07/2018	i.MX 8QuadXPlus Beta2 release
L4.9.123_2.3.0_8mm	09/2018	i.MX 8M Mini GA release
L4.14.62_1.0.0_beta	11/2018	i.MX 4.14 Kernel Upgrade, Yocto Project Sumo upgrade
L4.14.78_1.0.0_ga	01/2019	i.MX6, i.MX7, i.MX8 family GA release
L4.14.98_2.0.0_ga	04/2019	i.MX 4.14 Kernel upgrade and board updates
L4.19.35_1.0.0	07/2019	i.MX 4.19 Beta Kernel and Yocto Project Upgrades
L4.19.35_1.1.0	10/2019	i.MX 4.19 Kernel and Yocto Project Upgrades
L5.4.3_1.0.0	03/2020	i.MX 5.4 Kernel and Yocto Project Upgrades