Document Number: IMXLUG
Rev L3.10.53_1.1.0-ga, 01/2015

i.MX Linux User's Guide

1 About This Book

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

The steps needed to get the i.MX platform running are covered, including board DIP switch settings, and instructions on configuring and using the U-Boot boot loader.

Using some Freescale special features are covered under the later chapters. These show how to use the specific features when running this Linux OS kernel.

Features covered in this guide may be specific to particular boards or SOCs. Please check the *i.MX Linux Release Notes* (IMX6LXRN) for the capabilities of a particular board or SOC.

Contents

1.1 Audience

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

1.2 Conventions

This document uses the following conventions:

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

1.3 Supported Hardware SoCs and Boards

- i.MX 6Quad SABRE-SD Board and Platform
- i.MX 6DualLite SABRE-SD Platform
- i.MX 6Quad SABRE-AI Platform
- i.MX 6DualLite SABRE-AI Platform
- i.MX 6SoloLite EVK
- i.MX 6SoloX SABRE-SD Platform
- i.MX 6SoloX SABRE-AI Platform

Some abbreviations are used in places in this document.

- SABRE-SD refers to both the i.MX 6Quad SABRE-SD and the i.MX 6DualLite SABRE-SD boards.
- SABRE-AI refers to both the i.MX 6Quad SABRE-AI and the i.MX 6DualLite SABRE-AI boards.
- SoloLite refers to the i.MX 6SoloLite Board.
- SoloX or SX refer to the i.MX 6SoloX SABRE-SD and SABRE-AI boards.

1.4 References

The references below are included in the release and contain additional information.

- *i.MX Linux Release Notes* (IMX6LXRN) - 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.
- *Freescale Yocto Project User's Guide* (IMXLXYOCTOUG) - Contains the instructions for setting up and building Linux in the Yocto Project.
- *i.MX 6 Linux Reference Manual* (IMX6LXRM) - Contains the information on Linux drivers for i.MX.
- *i.MX 6 Graphics User's Guide* - Describes the graphics used.
- *i.MX 6 Linux High Assurance Boot (HAB) User's Guide* (IMX6HABUG) - Contains the information on using High Assurance Boot.
- *i.MX 6 BSP Porting Guide* (IMX6XBSPPG) - Contains the instructions on porting the BSP to a new board.
- *i.MX 6 VPU Application Programming Interface Linux Reference Manual* (IMX6VPUAPI) - Provides the reference information on the VPU API.

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

- SABRE Platform Quick Start Guide (IMX6QSDPQSG)
- SABRE Board Quick Start Guide (IMX6QSDBQSG)
- SABRE Automotive Infotainment Quick Start Guide (IMX6SABREINFOQSG)
- i.MX 6SoloLite Evaluation Kit Quick Start Guide (IMX6SLEVKQSG)

Documentation is available online at freescale.com.

- i.MX 6 information is at freescale.com/iMX6series
- i.MX 6 SABRE information is at freescale.com/imxSABRE
- i.MX 6SoloLite EVK information is at freescale.com/6SLEVK

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 boot loader, 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 need on how to set up the Linux host machine, how to run and configure a Yocto Project, generate an image, and generate a rootfs, are covered in *Freescale Yocto Project User's Guide*.

Once Linux is running, this guide contains information on how to use some special features that Freescale SoCs provide. The release notes cover which features are supported on a particular board.

3 Basic Terminal Setup

The i.MX boards can communicate with a host server (Windows or Linux) 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 Windows host.

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

The i.MX 6 SABRE-SD, i.MX 6SoloLite EVK, and i.MX 6SoloX SABRE-AI boards connect the host driver using the micro USB connector. The USB to serial driver can be found under www.ftdichip.com/Drivers/VCP.htm.

1. Connect the target and the Windows® OS PC using a serial cable on i.MX 6 SABRE-AI boards or a micro-B USB cable on i.MX 6 SABRE boards.
2. Open HyperTerminal on the Windows PC and select the settings as shown in the figure below.

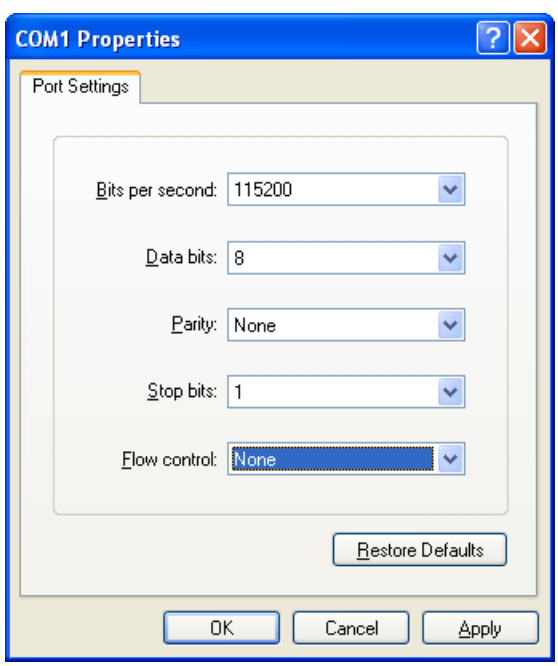

Figure 1. HyperTerminal Settings for Terminal Setup

4 Booting Linux OS

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

4.1 Software Overview

This section describes the software needed for the board to be able to boot and run Linux. To boot a Linux image, four things are needed: the bootloader (U-Boot), The Linux kernel image (zImage), a device tree file (.dtb) for the board being used, and a root file system (rootfs) for the particular Linux image.

The system can be configured for a specific graphical backend. The graphical backends are X11, Wayland, frame buffer, or direct frame buffer.

4.1.1 Boot Loader

U-Boot is the tool recommended as the bootloader. 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 boot loader names start with the name of the boot loader followed by the name of the platform and board followed by the name of the device that this image is configured to boot from: u-boot-[platform] [board]_[machine configuration].imx. If no boot device is specified, it will boot from SD/MMC.

The manufacturing tool can be used to load U-Boot onto all devices. 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.

4.1.2 The Linux Kernel Image and the Device Tree

This Freescale i.MX BSP contains a pre-built kernel image based on the 3.10.53 version of the Linux kernel and the device tree files associated with each of the platforms.

The same kernel image is used for all the i.MX boards. Device trees are kernel configuration files that allow a common kernel to boot 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. The prebuilt images are named zImage--[kernel]-[platform]-[board]-[configuration].dtb.

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, since LDO bypass mode is not supported on the CPU at 1.2 GHZ. The device tree *hdcp.dtb is used to enable the DHCP feature because of a pin conflict which requires this to be configured at build time.

4.1.3 The 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. The file system includes Freescale specific libraries and common Linux utilites. They are named with the following convention: [image recipe]-[backend]-[platform][board].[ext3|sdcard]. The ext3 extension indicates a standard file system. It can be mounted as NFS, or its contents can be stored on a boot media such as a SD/MMC card.

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

4.2 The Manufacturing Tool

The manufacturing tool, named MFGTool, is a tool that runs on a PC and is used to download images to different devices on an i.MX board. The tar.gz file can be found with the pre-built images.

4.2.1 Configuring MFGTool

Unzip Mfgtools-Rel-[version]_UPDATER.tar.gz

Instructions for MFGTool V2 can be found in the file `Profiles\[SOC] Linux Update\OS Firmware\ucl2.xml`. Read and update the `ucl2.xml` file to understand the operations before using MFGTool.

It is important to correctly configure the `cfg.ini` and `UICfg.ini` files. For example, if only one board will be supported, `PortMgrDlg=1` should be set in `UICfg.ini`; if four boards would be supported, `PortMgrDlg=4` should be set. Incorrect configuration will cause MFGTool to malfunction.

NOTE

For i.MX 6SoloX the default setting in `cfg.ini` the file needs to be changed to the following. MFGTool will look for the setting in the `ucl2.xml` file.

```
[profiles]
chip = Linux

[platform]
board = SabreSD

[LIST]
name = SDCard

[variable]
board = sabresd
mmc = 0
sxuboot=17x17arm2
sxdtb=17x17-arm2
ldo=
```

4.2.2 Using MFGTool

Follow these instructions to use the MFGTool V2:

- Connect a USB cable from a PC to the USB OTG port on the board.
- Connect a USB cable from OTG-to-UART port to PC for console output.
- Open a Terminal emulator program. Refer to "Basic Terminal Setup" in this guide.

- Set boot pin to Mfgtool mode. Refer to the section "Serial Download Mode for the Manufacturing Tool" in this guide.
- The default profile of the manufacturing tool assumes that your file system is packed and compressed using the bzip2 algorithm. An example can be found in the MFGTool release package in the folder `Profiles\[SOC] Linux Update \OS Firmware\files`. To create this file, you can run the following commands as a root user on Linux OS. You can also modify the profile to support other formats.

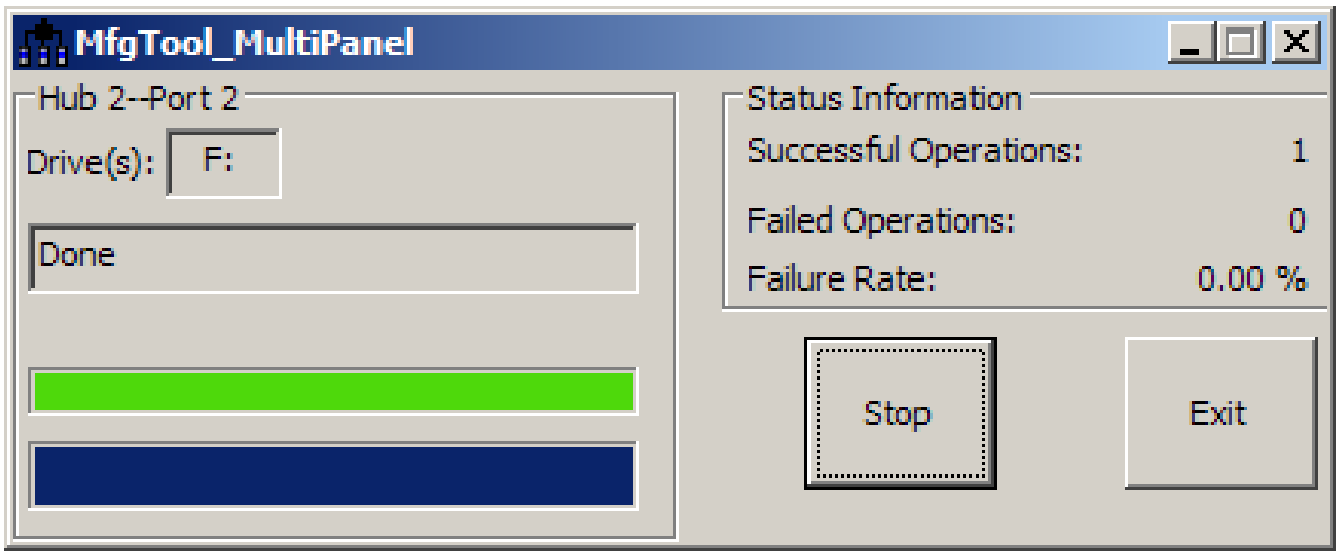

Figure 2. Programming SD with the Manufacturing Tool – Image downloading

NOTE

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

4.3 Preparing an SD/MMC Card to Boot

Information found here 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 amd MMC cards although for brevity, often only SD card is listed.

For a Linux image to be able to run, four separate pieces are needed: the Linux kernel image (zImage), the device tree file (*.dtb), the U-Boot boot loader image, and the root file system (*.ext3 or *.ext4).

A .sdcard image contains all four images properly configured for an SD card. The release contains a pre-built .sdcard image that is built specifically for the i.MX 6Quad Sabre-SD board. It runs the X11 graphical backend. It will not run on other boards unless U-Boot, the device tree and/or the rootfs are changed.

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.

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

The rootfs on the default .sdcard image is limited to a bit less than 4 GB but re-partitioning and re-loading the rootfs can increase that to the size of the card. The rootfs can also be changed to specify the graphical backend that will be 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 layout below 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	U-Boot and reserved area
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 boot loader and kernel images to initialize the partition table and copy the root file system. To simplify the instructions, it is assumed that a 4GB SD/MMC card is used.

Any Linux distribution can be used for the following procedure.

The Linux kernel running on the Linux host will assign 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, enter the 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 1 KB).

NOTE

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

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

```
$ sudo dd if=<.sdcard image> of=/dev/sdx bs=1M conv=fsync
```

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

4.3.3 Partitioning the SD/MMC Card

The full sdcard 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 many ways to partition an SD card. Essentially, the boot loader 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 a partition or not. The root file system does need 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 will be mounted automatically upon insertion. Therefore, before running fdisk, please make sure that the SD card is unmounted if it was previously mounted (via `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 but yours will differ.

```
$ 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.]

u          [switch the unit to sectors instead of cylinders]
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 boot loader 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 the Boot Loader Image

When not using the full SD card image, this section describes how to just load the boot loader image. Enter the following command to copy the U-Boot image to the SD/MMC card.

```
$ sudo dd if=<U-Boot image> of=/dev/sdx bs=512 seek=2 conv=fsync
```

The first 1 KB of the SD/MMC card, that includes the partition table, will be preserved.

4.3.5 Copying the Kernel Image and DTB File

When not using the full SD card image, this section describes how to load the kernel image and DTB. The pre-built SDcard image uses the VFAT partition for storing kernel image and DTB, which requires a VFAT partition that is mounted as a Linux drive and the files are simply copied into it. This is the preferred method.

Another method that can be used is for users to put the kernel image and DTB to the fixed raw address of the SD card by using the `dd` command. The later method needs to modify the U-Boot default environment variables for loading the kernel image and DTB.

Default: VFAT partition

1. Format partition 1 on the card as VFAT with this command:

   ```
   $ sudo mkfs.vfat /dev/sdx1
   ```
2. Mount the formatted partition:

   ```
   $ mkdir mountpoint
   $ sudo mount /dev/sdx1 mountpoint
   ```
3. Copy the zImage and *.dtb files to mountpoint by using `cp`. The device tree names should match the mount point specified by U-Boot. Be sure to un-mount the partition with:

   ```
   $ sudo umount mountpoint
   ```

Alternative: Fixed raw address

The following command can be used to copy the kernel image to the SD/MMC card:

```
$ sudo dd if=zImage_imx_v7_defconfig of=/dev/sdx bs=512 seek=2048 conv=fsync
```

This copies zImage to the media at offset 1 MB (bs x seek = 512 x 2048 = 1 MB).

The i.MX DTB image can be copied by using the copy command and copying the file to the 2nd partition or the following commands will copy an i.MX DTB image to the SD/MMC card by using `dd`. Choose the command for your board:

```
$ sudo dd if=zImage-imx6q-sabreauto.dtb of=/dev/sdx bs=512 seek=20480 conv=fsync
$ sudo dd if=zImage-imx6q-sabresd.dtb of=/dev/sdx bs=512 seek=20480 conv=fsync
$ sudo dd if=zImage-imx6sl-evk.dtb of=/dev/sdx bs=512 seek=20480 conv=fsync
```

This copies the board specific .dtb file to the media at offset 10 MB (bs x seek = 512 x 20480 = 10 MB).

4.3.6 Copying the Root File System (rootfs)

When not using the full SD card image, this section describes how to just load the rootfs image.

Copy the target file system to a partition that will only contain the rootfs. This example is using partition 2 for the rootfs. First format the partition. The file system format ext3 or ext4 is a good option for 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: extract `fsl-image-fb-imx6qdlsolo.ext3` to /home/user/rootfs for example:

```
$ sudo mount -o loop -t ext3 fsl-image-fb-imx6qdlsolo.ext3 /home/user/rootfs
```

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
```

NOTE

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

The file system content is now on the media.

4.4 Downloading Images

Images can be downloaded to a device using a U-Boot image already loaded on the boot device or by using the manufacturing tool, MFGTool. Use a terminal program to communicate with the i.MX boards.

4.4.1 Downloading Images Using U-Boot

The following section describes how to download images using the U-Boot bootloader.

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

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

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

4.4.1.1 Downloading an Image to MMC/SD

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

Insert an MMC/SD card into the SD card slot. This will be slot SD3 on i.MX 6 SABRE boards and SD1 on i.MX 6SoloLite boards.

To flash the original U-Boot, refer to Preparing an SD/MMC Card to Boot.

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

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

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

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

2. Configure the U-Boot environment for network communications. Below is an example. The lines preceded by the "#" character are comments and have no effect.

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

   The user can set a fake MAC address 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 MMC/SD card.

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

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

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

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

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

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

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

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

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

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

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

8. On i.MX 6 SABRE boards you can boot up the system through the rootfs in the SD card via the HannStar LVDS. The kernel MMC module now uses a fixed mmcblk index for the uSDHC slot. The SD3 slot uses mmcblk2 on i.MX 6 SABRE boards.

9. Boot up the i.MX 6SoloLite board.

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

4.4.1.2 eMMC4.4 on SDHC4

There is an eMMC4.4 chip on i.MX 6 SABRE Boards. It is accessed through SDHC4. The following steps describe how to use this memory device.

First, clean up environments stored on eMMC4.4, then set the correct boot pin configuration, power up the board and set the U-Boot environment variables. Finally, copy zImage to the TFTP server.

1. To clean up the environments stored on eMMC4.4, do the following in U-Boot console:

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

2. Set correct boot pin config. Power up the board and set the U-Boot environment variables as needed. For example,

```
U-Boot > setenv serverip <your tftpserver ip>
U-Boot > setenv bootfile <your kernel zImage 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 the zImage to tftp server. Then download it to RAM:

```
U-Boot > dhcp
```

4. Query the information about eMMC4.4 chip.

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

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

```
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 into eMMC4.4. For example, the command below writes the image with the size 0x800000 from ${loadaddr} to the offset 0x100000 of the eMMC4.4 chip. Here, the following equation applies: 0x800 =0x100000/0x200, 0x4000=0x800000/0x200. The block size of this card is 0x200. This example assumes the kernel image is less than 0x800000 bytes. If the kernel image exceeds 0x800000, enlarge the image length.

```
U-Boot > mmc dev 2 0
### suppose kernel zImage less than 8M
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 eMMC4.4 chip.

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

8. Boot up the system through RFS in eMMC4.4 via HannStar LVDS. The kernel MMC module now uses fixed mmcblk indices for the USDHC slots. The eMMC4.4/SD4 slot in i.MX 6 SABRE boards is mmcblk3.

```
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 RFS in eMMC4.4 via CLAA WVGA panel:

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

10. Boot up the system through RFS in eMMC4.4 via HDMI:

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

11. To program the rootfs to MMC/SD, see Using an i.MX Board as Host Server to Create a rootfs, or Preparing an SD/MMC Card to Boot.

4.4.1.3 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`.

```
----------------
tftpboot ${loadaddr} u-boot.imx
-----------------
```

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

```
-----------------
sf probe
sf erase 0 0x80000
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 target board.

4.4.1.4 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. Boot from SD card.
2. TFTP the U-Boot image.

```
tftpboot ${loadaddr} u-boot.imx
```

3. Flash U-Boot image.

```
cp.b ${loadaddr} 0x1000 ${filesize}
```

4. Change boot switches and reboot.

```
S2 all 0
S1-6 1 others 0
```

5. By default, rootfs is mounted on NFS.

4.4.1.5 Flashing an ARM Cortex-M4 Image on QuadSPI

i.MX 6SoloX SABRE-SD and SABRE-AI boards have an ARM Cortex-M4 processor and QuadSPI memory can be used to flash an image to it.

U-boot has a default script to flash ARM Cortex-M4 image from SD card VFAT partition. 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 SD card.
3. Flash the ARM Cortex-M4 image from SD card to the NOR flash on QuadSPI2 PortB CS0 on i.MX 6SoloX SABRE-SD board or QuadSPI1 PortB CS0 on i.MX 6SoloX SABRE-AI board..

```
----------------
run update_m4_from_sd
-----------------
```

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

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

```
-----------------
tftp ${loadaddr} m4_qspi.bin
----------------
```

3. Select the NOR flash on QuadSPI2 PortB CS0 on i.MX 6SoloX SABRE-SD board or QuadSPI1 PortB CS0 on i.MX 6SoloX SABRE-AI board..

```
-----------------
sf probe 1:0
----------------
```

4. Flash the ARM Cortex-M4 image to the selected NOR flash. The erase size is the ${filesize}, around 64 Kbytes. This example assumes that it is 128 Kbytes.

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

4.4.2 Using an i.MX Board as Host Server to Create a rootfs

Linux 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 SATA device. The example below is for an SD card. The device file node name will need 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
   ```

   Replace $SD below with the name of your device.
2. To create a partition on the MMC/SD card, use the fdisk command (requires root privileges) in the Linux console:

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

   If this is a new SD card, you may get the following message:

   ```
   The device contains neither a valid DOS partition table, nor Sun, SGI or OSF disk label
   Building a new DOS disklabel. Changes will remain in memory only,
   until you decide to write them. After that the previous content
   won't be recoverable.
   The number of cylinders for this disk is set to 124368.
   There is nothing wrong with that, but this is larger than 1024,
   and could in certain setups cause problems with:
   1) software that runs at boot time (e.g., old versions of LILO)
   2) booting and partitioning software from other OSs
      (e.g., DOS FDISK, OS/2 FDISK)
   ```

   The usual prompt and commands to partition the card follow. Bold 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
   ```

3. As described in Flash Memory Maps, the rootfs partition should be located after kernel image; the first 0x800000 bytes can be reserved for MBR, bootloader, and kernel sections. From the log shown above, the Units of current MMC/SD card is 32768 bytes. The beginning cylinder of the first partition can be set as "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
   ```

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

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

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

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

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

6. Type sync to write the contents to MMC/SD.

7. Type poweroff to power down the system. Follow the instructions in Running Linux on the Target to boot the image from MMC/SD card.

NOTE

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

1. Format the sd/mmc card first partition(for example 32M) with vfat filesystem.
2. Copy zImage and the DTB file into the VFAT partition after you mount the VFAT partition into your host PC.
3. Make sure that the zImage and DTB file name sync with the file name pointed to by the U-Boot environment variables: fdt_file and image. Use print command under U-Boot to display these two environment variables. For example:

   ```
   print fdt_file image
   ```

4. U-Boot will load the kernel image and the DTB file from your VFAT partition automatically when you boot from the SD/MMC card.

4.5 How to Boot the i.MX Boards

Once U-Boot has been 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 SD2

The SD card slot that is labeled SD2 indicates that this slot is connected to the uSDHC pin SD2 on the processor. Most boards label this slot as SD2. This slot will be referred to as SD2 here.

i.MX 6 SABRE-SD Boards

The following table shows the DIP switch settings for booting from the SD card slot labeled SD2 and J500 on the i.MX 6 SABRE-SD boards. The SD2 card slot is located beside the LVDS1 connection on the back of the board.

Table 2. Booting from SD2 (J500) on i.MX 6 SABRE-SD

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

i.MX 6SoloLite Boards

The i.MX 6SoloLite boards have three SD card slots on the main board. The one on the end is labeled as the SD2 slot. The following table shows the DIP switch settings for booting from SD2.

Table 3. Booting from SD2 on i.MX 6SoloLite

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW3	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
SW4	OFF	OFF	ON	OFF	ON	ON	OFF	OFF
SW5	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
S1	OFF	ON						

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 will be referred to as SD3 in this documentation.

i.MX 6 SABRE-AI Board

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

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

Table continues on the next page...

Table 4. Booting from an SD card in slot SD3 on i.MX 6 SABRE-AI boards (continued)

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

i.MX 6SoloX SABRE-AI Board

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

Table 5. Booting from an MMC card in Slot SD3 on i.MX 6SoloX SABRE-AI board

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 6. Booting from an SD Card in Slot SD3 on i.MX 6 SABRE-SD boards

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 shows the dip switch settings for booting from an SD card in slot SD4.

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

Table 7. 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 8. 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 eMMC4.4

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, refer to the table "MMC/eMMC Boot Fusemap" in the IC reference manual.

The i.MX 6SoloLite EVK requires a daughter card with the eMMC chip mounted on it. The table below shows the boot switch settings to boot from eMMC4.4 on i.MX 6SoloLite EVK.

Table 9. Booting From eMMC4.4 Boot with 4 bit SDR Mode on i.MX 6SoloLite EVK

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

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

Table 10. Booting From eMMC on i.MX 6 SABRE-SD boards

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

4.5.5 Booting From SATA

The switch settings below enable booting from SATA.

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

Table 11. Booting From SATA on i.MX 6 SABRE-SD boards.

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 12. 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 6SoloX SABRE-AI board.

Table 13. Booting from NAND on i.MX 6 SoloX SABRE-AI

Switch	D1	D2	D3	D4	D5	D6	D7	D8
S4	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON
S3	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
S1	OFF	OFF	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 14. Booting from SPI-NOR on i.MX 6 SABRE-AI boards

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						

Table 15. Booting from SPI-NOR on i.MX 6SoloLite EVK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW3	OFF	OFF	ON	ON	OFF	OFF	OFF	OFF
SW4	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW5	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF

4.5.8 Booting From EIM (Parallel) NOR

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

Table 16. Booting From EIM NOR on i.MX 6 SABRE-AI boards

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. Both cannot be used for the same configuration. The default U-Boot configuration is set to SPI NOR.

4.5.9 Booting From QuadSPI

The following table shows the DIP switch settings for booting from QuadSPI.

Table 17. 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 18. 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				

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 and i.MX 6SoloLite boards. There are various ways to enter serial download mode, however. One way is to set the boot mode to boot from SD slot SD3 (set SW6 DIP switches 2 and 7 **on** the rest are **off**). Do not insert the SD card into the SD3 slot, and power on the board. After the message "HID Compliant device" is displayed, the board will enter serial download mode. Now 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.

Table 19. Setup for the Manufacturing Tool on i.MX 6SoloLite EVK

Switch	D1 (BOOT_MODE0)	D2 (BOOT_MODE1)
S1	ON	OFF

The table below 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 will also enter the serial download mode.

Table 20. Setup for the Manufacturing Tool on i.MX 6 SABRE-AI boards

Switch	D1	D2	D3	D4
S3	OFF	ON	OFF	OFF

4.6 Flash Memory Maps

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

This information may be 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 example below briefly describes how to use this. Memory is allocated in the order 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) in order 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. The default is the the U-Boot boot arguments use 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 would be 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 will load 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. Below 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 1k offset of the SPI-NOR memory, so that the device can boot from it. The default configuration is that on boot up, U-Boot will load 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. Below 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 QuadSPI memory so that the device can boot from it. The default configuration is that on boot up, U-Boot will load 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. Below 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 21. U-Boot Mapping for QuadSPI

Device on HW	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	0x60000000	
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	

4.7 Running Linux on the Target

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

These instructions assume that you have downloaded the kernel image using the instructions in Downloading Images , or Preparing an SD/MMC Card to Boot. If you have not setup your Serial Terminal yet, please refer to Basic Terminal Setup.

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

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

```
U-Boot > saveenv
```

4. Run the boot command:

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

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

Specifying the Console

The console for debug and command-line control can be specified on the Linux boot command line. The SABRE-Auto board uses ttymxc3, so it is not same for all boards. It is usually specified as listed below but the baudrate and the port can be modified, so for NFS it might be `ttymxc3`.

```
U-Boot > setenv consoleinfo 'console=ttymxc0,115200'
```

Specifying Displays

Display information can be specified on the Linux boot command line. It is not dependant on the source of the Linux image. If nothing is specified for the display, the settings in the device tree will be used. Add `${displayinfo}` to the environment macro containing `bootargs`. The specific parameters can be found in the *i.MX Linux Release Notes*. Below are some examples of what these might look like.

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

Specifying Memory Addresses

The address in memory where the kernel and device tree will be loaded to do not change based on the device Linux is running from. The instructions in this chapter use the environment variables `loadaddr` and `ftd_addr` to indicate these values. The table below shows what addresses are used on the different i.MX boards.

Table 22. Board-Specific Default Values

Variable	6Quad and 6DualLite SABRE (AI and SD)	SoloLite	SoloX	Description
loadaddr	0x12000000	0x80800000	0x80800000	the address in memory the kernel will be loaded to
fdt_addr	0x18000000	0x83000000	0x83000000	the address in memory to copy the device tree code to

In addition, `fdt_file` is used to specify the filename of the device tree file. Setting the U-Boot environment variables would look like this:

```
U-Boot > setenv loadaddr 0x12000000
U-Boot > setenv fdt_addr 0x18000000
U-Boot > setenv fdt_file 'imx6q-sabresd.dtb'
```

Specifying the Location of the Root File System

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

- `U-Boot > setenv rootfsinfo 'root=/dev/nfs ip=dhcp nfsroot=${serverip}:${nfsroot},v3,tcp'`
- `U-Boot > setenv rootfsinfo 'root=/dev/nfs ip=dhcp weim-nor nfsroot=${serverip}:${nfsroot},v3,tcp'`
- `U-Boot > setenv rootfsinfo 'ubi.mtd=3 root=ubi0:rootfs rootfstype=ubifs rootwait rw mtdparts=gpmi-nand:16m(boot),16m(kernel),16m(dtb),-(rootfs)'`
- `U-Boot > setenv rootfsinfo 'root=/dev/mmcblk0p2 rootwait rw'`

Special Settings

SoloLite can specify `uart_at_4M` and SoloX can specify `uart_at_osc` on the command line to specify that the OSC clock and not PLL3 should be used. This allows the system to enter low power mode.

For SoloLite

```
U-Boot > setenv special 'uart_at_4M'
```

For SoloX

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

Building the Command Line

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

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

The executed boot command then looks something like the one below. Arguments to this will vary by device.

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

4.7.1 Running Linux From MMC/SD

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

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

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

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

Table 23. Linux uSDHC Relationships

uSDHC	mmcroot
uSDHC 1	mmcblk0*
uSDHC 2	mmcblk1*

Table continues on the next page...

Table 23. Linux uSDHC Relationships (continued)

uSDHC	mmcroot
uSDHC 3	mmcblk2*
uSDHC 4	mmcblk3*

To boot the system follow the steps below. In the default configuration of the SD card, and the example here, U-Boot is at the 1024 byte offset before the first partition, partition 1 is the partition with the Linux kernel and device trees, and partition 2 is the rootfs.

Setting up the environment variables

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

The settings below are one way of setting up the items needed to boot linux.

```
U-Boot > setenv mmcpart 1
U-Boot > setenv loadfdt 'fatload mmc ${mmcdev}:${mmcpart} ${fdt_addr} ${fdt_file}'
U-Boot > setenv loadkernel 'fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} zImage'

U-Boot > setenv bootcmd 'mmc dev ${mmcdev}; run loadkernel; run mmcargs; run loadfdt; bootz $
{loadaddr} - ${fdt_addr};'
```

Here are some descriptions of the variables used above:

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

NOTE

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

Reading the Kernel Image From eMMC4.4

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

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

4.7.2 Running the Image from NAND

NAND can be found on i.MX 6 SABRE-AI Boards.

Power up the board, then enter the commands provided. The following settings may be used to boot the Linux system from NAND:

Assume the kernel image starts from the address 0x1400000 byte (the block start address is 0x800). The kernel image size is less than 0x400000 byte. The rootfs is located in `/dev/mtd2`.

```
U-Boot > setenv bootcmd 'run  bootargsset; nand read ${loadaddr} 0x1000000 0x800000; nand
read ${fdt_addr} 0x2000000 0x100000; bootz ${loadaddr} - ${fdt_addr}'
```

4.7.3 Running Linux from Parallel NOR

Parallel NOR is available on i.MX 6 SABRE-AI boards. The following steps may be used to boot the system from Parallel NOR:

Assume the kernel image starts at address 0xc0000 bytes. At the U-Boot prompt setup these variables:

```
U-Boot > setenv bootcmd 'run bootargsset; cp.b 0x80c0000 ${loadaddr} 0x800000;cp.b 0x80a0000
${fdt_addr} 0x20000;bootz ${loadaddr} - ${fdt_addr} '
```

4.7.4 Running the Linux Image From QuadSPI

QuadSPI is available on i.MX 6SoloX SABRE-SD Boards. The following steps may be used to boot the Linux system from QuadSPI NOR:

Assume the kernel image starts from the address 0xA00000 byte. The DTB file starts from address 0x800000. At the U-Boot prompt set the following environment variables:

```
U-Boot > setenv bootcmd 'run bootargsset; sf probe; sf read ${loadaddr} 0xA00000 0x2000; sf
read ${fdt_addr} 0x800000 0x800; bootz ${loadaddr} - ${fdt_addr} '
```

4.7.5 Running the Linux Image From NFS

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

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

NOTE

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

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

4.7.6 Running the M4 Image

On i.MX 6SoloX boards, there are two ways to boot M4 Images in U-Boot:

1. M4 Normal Up (Supported on i.MX 6SoloX SABRE-AI and SABRE-SD boards). Performed by running the U-Boot command. Requires:
   a. U-Boot normal SD image if A9 booting from SD card. U-Boot normal qspi image if A9 booting from QSPI NOR flash.

   b. Kernel DTB: imx6sx-sdb-m4.dtb for i.MX 6SoloX SABRE-SD board. imx6sx-sabreauto-m4.dtb for i.MX 6SoloX SABRE-AI board.
   c. Have the M4 image burned. (NOR flash of QuadSPI2 PortB CS0 for i.MX 6SoloX SABRE-SD board. NOR flash of QuadSPI1 PortB CS0 for i.MX 6SoloX SABRE-AI board.)
2. M4 Fast Up (Only supported on i.MX 6SoloX SABRE-SD board). Initiated by U-Boot at a very early boot phase to meet the requirement of M4 booting in 50 ms. No U-Boot command is involved. Requires:
   a. U-Boot M4 fast up image and A9 must boot from QSPI2 NOR flash.
   b. Kernel DTB: imx6sx-sdb-m4.dtb
   c. Have the M4 image burned. (NOR flash of QuadSPI2 PortB CS0)

To facilitate the M4 Normal Up, a script has been added to the default U-Boot. The following steps may help users who need to run the M4 Normal Up script.

1. Power up the board.
2. On i.MX6 SoloX SABRE-SD board, assumed the M4 image is at address 0x78000000 (NOR flash of QuadSPI2 PortB CS0). On i.MX6 SoloX SABRE-AI board, assumed the M4 image is at address 0x68000000 (NOR flash of QuadSPI1 PortB CS0)

At the U-Boot prompt:

```
U-Boot > run m4boot
```

Or users can perform the commands without depending on the script:

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

For i.MX 6SoloX SABRE-SD board:

```
U-Boot >  bootaux 0x78000000
```

For i.MX 6SoloX SABRE-AI board:

```
U-Boot >  bootaux 0x68000000
```

NOTE

For how to add the MCC demo to the kernel and limit RAM available to kernel to use it, see Chapter 53 "i.MX 6 SoloX Multi-Core Communication (MCC)" of the *i.MX Linux Reference Manual*.

4.7.7 Linux Login

The default login for the Freescale Linux is `root` with no password.

5 Enabling Solo Emulation

Solo emulation can be enabled on the i.MX 6 SABRE-SD and the i.MX 6 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 6 SABRE platforms, the capabilities of the i.MX 6DualLite change to the following:

- One CPU is enabled
- 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, 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, 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 6 Linux Reference Manual* contains information on the power modes that are available and other Freescale-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 setting, `/sys/power/state`. The command 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 Linux Reference Manual.

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

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

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

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

  ```
  /unit_tests/rtcwakeup.out -m mem -s 10
  ```

  (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 `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`.

Below 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/stats/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
```

  The frequency will be 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 will 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 will set the scaling governor to a specified frequency, if that frequency is supported. If the frequency is not supported, the closest supported frequency will be 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

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

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

- Normal DDR frequency – 528 MHz or 400 MHz
- Audio DDR frequency – 50 MHz on i.MX 6Quad, i.MX 6DualLite, and i.MX 6SoloX; and 100 Mhz on i.MX 6SoloLite
- Low power idle DDR frequency – 24 MHz
- Ultra Low power IDLE DDR frequency – 1 MHz (only on i.MX 6SoloLite)

In order 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, etc if more than one display is active).

`ifconfig eth0 down` -> will disable the Ethernet module. On i.MX 6SoloX, should also disable ethernet 1(eth1).

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

To manipulate busfreq use the following commands to achieve the results desired:

`cat /sys/bus/platform/drivers/imx6_busfreq/busfreq.[id]/enable` -> displays the status of busfreq

`echo 0 > /sys/bus/platform/drivers/imx6_busfreq/busfreq.[id]/enable` -> disables busfreq

`echo 1 > /sys/bus/platform/drivers/imx6_busfreq/busfreq.[id]/enable` -> enables busfreq.

The Linux Reference Manual has more information on busfreq in the DVFS chapter.

7 Multimedia

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

Examples are provided for both Gstreamer 1.x and Gstreamer 0.10. Freescale Gstreamer 1.x will be version of future development while Gstreamer 0.10 is in maintenance mode and will eventually be deprecated when a future Yocto Project versions removes the core Gstreamer 0.10 components.

7.1 Freescale Multimedia Packages

Due to license limitations, Freescale multimedia packages can be found in two locations:

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

For details, see the *i.MX Release Notes*.

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 `fslcodec-microsoft-$version.tar.gz`.

7.3 Multimedia Use Cases

Gstreamer is the default multimedia framework on Linux. The sections below give examples of Gstreamer commands to perform the specific functions indicated. The names of the commands vary between different versions of Gstreamer. The table below shows how this document will refer to common functions and what the actual command is in the different versions.

Table 24. Command Mapping

Variable	$GSTL	$PLAYBIN	$GPLAY	$GSTINSPECT
Gstreamer 0.10	gst-launch	playbin2	gplay	gst-inspect
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 in the examples below. You may want to use the full path to the command on your system.

For Gstreamer 1.x:

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

For Gstreamer 0.10:

```
export GSTL=gst-launch
export PLAYBIN=playbin2
export GPLAY=gplay
export GSTINSPECT=gst-inspect
```

In this document, variables are often used to describe the command parameters that have multiple options. These variables are of the format $description where description describes the type of values that can be used. The possible options can be found in the *i.MX Linux Release Notes* in the Multimedia section, for Freescale 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 a !. Each plugin can have arguments of its own which appear on the command line after the plugin name and before the next !. 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

Playbacks 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 will have no effect.

This example plays an MP3 file. The command is the same for Gstreamer 1.x and 0.10:

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

7.3.1.2 Video Only Playback

A Gstreamer 1.x command would include a capsfilter so it might look like this:

```
$GSTL filesrc location=test.video typefind=true
```

```
     ! $capsfilter ! $demuxer_plugin ! queue max-size-time=0
     ! $video_decoder_plugin ! $video_sink_plugin
```

An MP4 video file playback might look like this:

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

Gstreamer 0.10 commands for video playback might look like this:

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

AVI video file playback might look like this:

```
$GSTL filesrc location=test.avi typefind=true
     ! aiurdemux ! queue max-size-time=0
     ! vpudec ! imxv4l2sink
```

7.3.1.3 Audio/Video File Playback

A Gstreamer 1.x command to play a video file with audio might look like this:

```
$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
```

An example of this for Gstreamer 1.x for an AVI file follows:

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

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

A Gstreamer 0.10 command to play a video file with audio might look like this:

```
$GSTL filesrc location=test_file typefind=true
     ! $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
```

A Gstreamer 0.10 AVI file playback example follows:

```
$GSTL filesrc location=test.avi typefind=true
```

```
! aiurdemux name=demux demux.
! queue max-size-buffers=0 max-size-time=0 ! vpudec
! imxv4l2sink demux.
! queue max-size-buffers=0 max-size-time=0 ! beepdec
! pulsesink
```

7.3.1.4 Multi-Channel Audio Playback

Multi-channel audio playback settings to use when pulseaudio is enabled can be found in pulseaudio Input/Output Settings.

7.3.1.5 Other Methods for Playback

You can use the $PLAYBIN plugin or the Freescale $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. For Gstreamer 1.x`overlaysink` can be used as the video sink plugin. For Gstreamer 0.10 both `imxv4l2sink` and `mfw_isink` can do this.

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). To see how to configure the system to boot in this mode, see the *i.MX 6 BSP Porting Guide* (IMX6BSPPG).

To configure the video sink plugin for multi-display mode, refer to overlaysink Usage and mfw_isink Usage later in this document.

7.3.1.6.1 Playing Different Videos On Different Displays

The command line to play 2 videos on different displays in Gstreamer 1.x might look like this.

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

Here are some examples for the two video sink plugins for Gstreamer 0.10.

- imxv4l2sink

  ```
  $GSTL $PLAYBIN uri=file:///$file1 video-sink="imxv4l2sink device=$VIDEO_DEVICE1
       disp-width=$width disp-height=$height" &
       $GSTL $PLAYBIN uri=file:///$file2 video-sink="imxv4l2sink device=$VIDEO_DEVICE2
       disp-width=$width disp-height=$height"
  ```

  Example on i.MX 6Dual/6Quad SD board, LVDS (primary) + HDMI:

  ```
  $GSTL $PLAYBIN uri=file:///$file1 video-sink="imxv4l2sink device=/dev/video17" &
  $GSTL $PLAYBIN uri=file:///$file2 video-sink="imxv4l2sink device=/dev/video18
       disp-width=1920 disp-height=1080"
  ```

- mfw_isink

  Example on i.MX 6Dual/6Quad SD board, LVDS (primary) + HDMI:

```
export VSALPHA=1
$GSTL $PLAYBIN uri=file:///$file1 video-sink="mfw_isink display=LVDS"
$PLAYBIN uri=file:///$file2 video-sink="mfw_isink display=HDMI"
```

Note: LVDS and HDMI in the command above are the display names defined in the vssconfig file.

7.3.1.6.2 Routing The Same Video To Different Displays

Video can be displayed on multiple displays in Gstreamer 1.x with a command like the following:

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

The examples below show how the same video can be displayed on different displays simultaneously using Gstreamer 0.10.

- imxv4l2sink

```
$GSTL $PLAYBIN uri=file:///$filename video-sink="tee name=tee
     ! queue ! imxv4l2sink device=$VIDEO_DEVICE1 disp-width=$width disp-height=$height
     tee.
     ! queue ! imxv4l2sinkdevice=$VIDEO_DEVICE2 disp-width=$width disp-height=$height"
```

- mfw_isink

```
export VSALPHA=1
$GSTL $PLAYBIN uri=file:///$file video-sink="mfw_isink display=LVDS display-1=HDMI"
```

7.3.1.6.3 Multiple Video Overlay

The `overlaysink` plugin in Gstreamer 1.0 and the `mfw_isink` plugin in Gstreamer 0.10 provide support for compositing multiple videos together and rendering them to the same display. The result might look something like the image below.

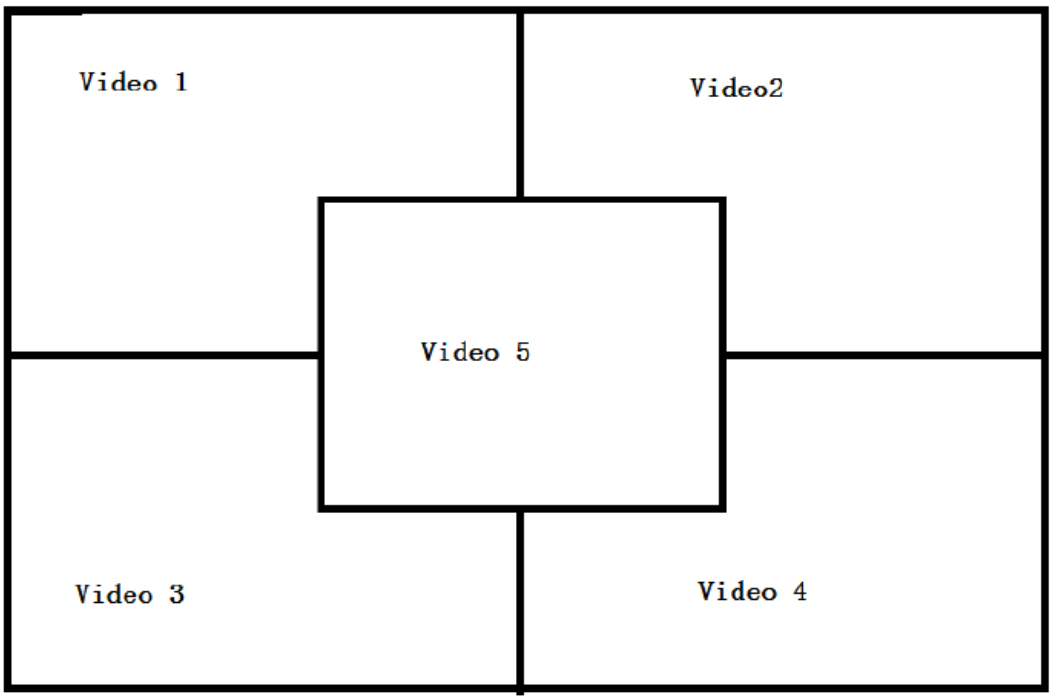

Figure 3. Multiple video overlay

A Gstreamer 1.x command to do this would look like this:

```
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"
```

With Gstreamer 0.10, if you want to play three videos to different windows of one display, 320x240 at (0,0), 640x480 at (400,0), and 320x240 at (400, 400), you can do it with the following commands:

```
export VSALPHA=1
$GSTL $PLAYBIN uri=file:///$file1
    video-sink="mfw_isink axis-left=0 axis-top=0 disp-width=320 disp-height=240” &
    $GSTL $PLAYBIN  uri=file:///$file2
    video-sink="mfw_isink axis-left=400 axis-top=0 disp-width=640 disp-height=480” &
    $GSTL $PLAYBIN uri=file:///$file3
    video-sink="mfw_isink axis-left=400 axis-top=500 disp-width=320 disp-height=240”
```

7.3.2 Audio Encoding

Here are some examples for MP3 encoding.

For Gstreamer 1.x the command might look like this.

```
$GSTL filesrc location=test.wav ! wavparse ! imxmp3enc bitrate=48 quality=1
     ! filesink location=output.mp3
```

For Gstreamer 0.10 the command might look like this.

```
$GSTL filesrc location=test.wav ! wavparse ! mfw_mp3encoder bitrate=48 optmode=0
     ! filesink location=output.mp3
```

WMA encoding for Gstreamer 0.10 might look like this.

```
$GSTL filesrc location=test.wav ! wavparse ! mfw_wma8encoder
     ! filesink location=output.wma
```

7.3.3 Video Encoding

Video encoding requires VPU hardware, so this command will only work on a SOC with a VPU.

The command in Gstreamer 1.x might look like this.

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

- The target encoder codec type could be MPEG4, H263, H264, MJPEG.
- It can be set in place of $capsfilter in above command line.
- By default, if $capsfilter is null, the target codec type is H264.
- If $capsfilter is set as ”video/mpeg, mpegversion=(int)4, systemstream=(boolean)false”, the target codec type is MPEG4.
- If $capsfilter is set as ”video/x-h263”, the target codec type is H263.
- If $capsfilter is set as ”image/jpeg”, the target codec type is MJPG.
- The $MUXER can be set as qtmux, matroskamux, mp4mux, avimux, or flvmux.
- Different muxers support different encoded codec types. Use $GSTINSPECT $MUXER to see the capabilities of the muxer to be used.

A Gstreamer 0.10 command might look like the one below.

```
$GSTL filesrc location=test.yuv blocksize=$blocksize
     ! 'video/x-raw-yuv, format=(fourcc)I420, width=$WIDTH, height=$HEIGHT,
framerate=(fraction)30/1'
     ! vpuenc codec=0 ! matroskamux ! filesink location=output.mkv sync=false
```

- The blocksize property of the filesrc plug-in depends on the resolution of the input image. For I420 YUV files, Blocksize = inputwidth * inputheight * 1.5
- The codec type property of the $video_encoder_plugin plug-in controls the target encode codec type. It could be 0 (MPEG4), 5 (H263), 6 (H264), or 12 (MJPG).

7.3.4 Transcoding

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

For Gstreamer 1.x, the command might look like this:

```
$GSTL filesrc location=$filename typefind=true ! $capsfilter ! aiurdemux
! vpudec ! imxvideoconvert_ipu ! $CAPS1 ! vpuenc ! $CAPS2 ! matroskamux ! filesink
location=720p.mkv
```

capsfilter is the container's mime type. CAPS1 is the target video resolution, CAPS2 is the target encoder codec type.

For example:

```
gst-launch-1.0 filesrc location=$FILE.mp4 typefind=true ! video/quicktime ! aiurdemux !
vpudec ! imxvideoconvert_ipu ! video/x-raw,format=NV12,width=1280,height=720 ! vpuenc !
video/x-h263 ! matroskamux ! filesink location=$FILE.mkv
```

A command line example for gstreamer 0.10 follows. Essentially, this command converts the file to raw YUV and then to the requested output format.

```
$GSTL filesrc location=$filename typefind=true ! aiurdemux ! vpudec ! mfw_ipucsc
      ! 'video/x-raw-yuv, format=(fourcc)NV12, width=1280, height=720' ! vpuenc
      ! matroskamux ! filesink location=720p.mkv
```

7.3.5 Audio Recording

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

- MP3 recording

  The Gstreamer 1.x command might look like this:

  ```
  $GSTL pulsesrc num-buffers=$NUMBER blocksize=$SIZE ! imxmp3enc
       ! filesink location=output.mp3
  ```

  The Gstreamer 0.10 command would look like this:

  ```
  $GSTL alsasrc num-buffers=$NUMBER blocksize=$SIZE ! mfw_mp3encoder
       ! filesink location=output.mp3
  ```

- WMA recording

  The Gstreamer 0.10 command would look like this:

  ```
  $GSTL alsasrc num-buffers=$NUMBER blocksize=$SIZE ! mfw_wma8encoder
       ! filesink location=output.wma
  ```

NOTE

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

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

For Gstreamer 1.x:

```
$GSTL alsasrc num-buffers=430 blocksize=4096 ! imxmp3enc
     ! filesink location=output.mp3
```

For Gstreamer 0.10:

```
$GSTL alsasrc num-buffers=430 blocksize=4096 ! mfw_mp3encoder
     ! 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 the BSP document for camera).

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

An example of recording with Gstreamer 1.x might look like this:

```
$GSTL imxv4l2src device=$DEVICE num-buffers=300 ! $INPUT_CAPS ! queue ! vpuenc
    ! $ $MUXER ! filesink location=output.$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 as 'video/x-raw,format=(string)NV12,width=1920,height=1080,framerate=(fraction)30/1'.
- $MUXER can be set as as qtmux, matroskamux, mp4mux, avimux, or flvmux.
- $EXTENSION is filename extension according to muxer type.

A Gstreamer 0.10 example of recording follows:

```
$GSTL imxv4l2src fps-n=15 capture-mode=X !  queue ! $video_encoder_plugin codec=0
     ! matroskamux ! filesink location=output.mkv sync=false
```

The fps-n property of the imxv4l2src plug-in controls the camera capture frame rate. The codec property of the $video_encoder_plugin plug-in controls the target encode codec type.

7.3.7 Audio/Video Recording

A Gstreamer 1.x command to record audio and video together might look like this:

```
$GSTL –e imxv4l2src device=$DEVICE ! $INPUT_CAPS ! queue ! vpuenc ! 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 as 'video/x-raw, format=(string)NV12, width=1920, height=1080, framerate=(fraction)30/1’
- $MUXER can be set as as qtmux, matroskamux, mp4mux, avimux, or flvmux

A Gstreamer 0.10 command to record audio and video together might look like this:

```
$GSTL –e imxv4lsrc capture-mode=X fps-n=30 ! $video_encoder_plugin  codec=0  ! queue
     ! mux. alsasrc ! 'audio/x-raw-int,rate=44100,channels=2' ! mfw_mp3encoder ! queue
     ! mux. $MUXER name=mux ! filesink location= output.$EXTENSION sync=false
```

- `$MUXER` can be matroskamux, mp4mux, avimux, flvmux, qtmux, or mpegtsmux.
- If multiplexing the MPEG4 video to mpegtsmux, vpuenc needs to set the property seqheader-method=2. The FSL MPG parser cannot support the MPEG4 format at the same time.

Common parameters are:

- `-e` indicates 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 Recording 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 follows:

For gstreamer 1.x use:

```
gst-launch-1.0 imxv4l2src ! imxv4l2sink
```

For Gstreamer 010 use:

```
gst-launch tvsrc ! imxv4l2sink

$GSTL tvsrc num-buffers=100 ! vpuenc ! matroskamux
     ! filesink location=./output.mkv sync=false
```

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 imxv4l2sink deinterface is True.

7.3.9 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
```

7.3.10 Real Time Streaming Protocol (RTSP) Playback

You can use the following command to see the Gstreamer RTP depacketize plugins:

```
$GSTINSPECT | grep depay
```

RTSP streams can be played with a manual pipline or by using playbin. The format of the commands is below.

- 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, you can 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.

If you need to pause or resume the RTSP streaming playback, you need to 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 will cause buffers to be dropped after resuming.

Gstreamer 1.x

Manual pipeline example for Gstreamer 1.x:

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

Playback will not exit automatically in Gstreamer 1.x if `buffer-mode` is set to `buffer` in the `rtpsrc` plugin.

Gstreamer 0.10

For the H.264 high bit rate playback with Gstreamer 0.10, set the access-unit property in the `rtph264depay` plugin to True to output H.264 in complete frames.

When using Gstreamer 0.10, set the video and audio decoder to low-latency mode for RTSP streaming playback. For the `vpudec` playback, if the low-latency property is set to True, it can work in low-latency mode. For the `beepdec` playback, if an audio parser is connected before the `beepdec` input audio is framed, the `beepdec` playback can work in low-latency mode.

- Manual pipeline example for Gstreamer 0.10

  For example with H.264 + AAC on Gstreamer 1.x:

```
gst-launch rtspsrc location=rtsp://10.192.241.11:8554/test name=source latency=200
buffer-mode=slave
     ! queue ! rtph264depay access-unit=true ! vpudec low-latency=true ! imxv4l2sink
source.
     ! queue ! rtpmp4gdepay ! aacparse ! beepdec ! pulsesink
```

  The audio parse plugin is required before the `beepdec` plugin enables `beepdec` to work in low-latency mode.

- PLAYBIN

  The `vpudec` low latency in Gstreamer 0.10 needs to be set to True if playing with $PLAYBIN.

```
gst-launch playbin2 uri=$RTSP_URI
```

7.3.11 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.
- One property of `aiurdemux` that is useful for UDP/RTP TS streaming is:

  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 case, you should cancel the current command and restart the pipeline. If the value is too large, you will need to wait for a long time to see the video after starting the server.

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
```

Gstreamer 1.x might use the following command:

```
$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
```

Gstreamer 0.10 commands should use low-latency so they would look like these:

```
$GSTL udpsrc do-timestamp=false uri=udp://10.192.241.255:10000 caps="video/mpegts"
     ! aiurdemux streaming_latency=400 name=d d. ! queue ! vpudec low-latency=true
     ! queue ! imxv4l2sink sync=true d. ! queue ! beepdec ! pulsesink sync=true
```

The format for a RTP MPEGTS streaming command is covered below:

```
$GSTL udpsrc do-timestamp=false uri=$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
```

A Gstreamer 1.x example:

```
$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
```

A Gstreamer 0.10 RTP MPEGTS command follows:

```
$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 low-latency=true ! queue ! imxv4l2sink sync=true d. ! queue
```

```
! beepdec ! $audio_sink_plugin sync=true
```

7.3.12 RTSP Streaming Server

The RTSP streaming server use case is based on the open source gst-rtsp-server package. It uses the Freescale aiurdemux plugin to demultiplex the file to audio or video elementary streams and to send them out through RTP. You can 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. You can 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 += "gst-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:

   $yocto_root/build/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/gst-rtsp-server/$version/gst-rtsp-server-$version/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 follows:

- Commands:

  ```
  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

7.4 pulseaudio Input/Output Settings

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

Audio output settings

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

```
$ pactl list sinks
```

A list of available audio sinks will be 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 available audio sources:

```
$ pactl list sources
```

A list of available audio sources will be 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 monitor mode.

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

```
$ pactl stat
```

Multi-channel output support settings

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

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

```
$ pacmd list-cards
```

The available sound cards and the profiles they support will be 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>
        ...
        ...
```

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

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

`$CARD` is the card name listed by `pacmd list-cards` (e.g. alsa_card.platform-sound-cs42888.34 in the example above), `$PROFILE` is the profile name. These are also listed by `pamcd list-cards`. (e.g. output:analog-surround-51 in the example above).

Then after setting the card profile, use `$ pactl list sinks` and `$pacmd set-default-sink $sink-number` to set the default sink.

7.5 New Multimedia Items In Gstreamer 1.x

The sections below contain new use cases, plugin and other information for Gstreamer 1.x.

7.5.1 Camera Preview

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

```
$GSTL imxv4l2src ! 'video/x-raw, format=(string)$FORMAT,
     width=$WIDTH, height=$HEIGHT, framerate=(fraction)30/1'
     ! imxv4l2sink
```

Camera preview example:

```
$GSTL imxv4l2src device=/dev/video1 ! 'video/x-raw,
     format=(string)UYVY,width=640,height=480,framerate=(fraction)30/1'
     ! imxv4l2sink
```

Parameter comments:

- Get the camera support format and resolution using gst-inspect-1.0 imxv4l2src.
- Set caps filter according to the camera's supported capabilities if the user needs other format or resolution.
- Ensure the right caps filter has been set which also needs to be supported by imxv4l2sink.

7.5.2 Web Camera

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

```
$GSTL imxv4l2src device=/dev/video1 ! vpuenc ! rtph264pay ! udpsink host=$HOST_IP
```

Where 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 ! application/x-rtp ! rtph264depay ! vpudec ! imxv4l2sink
```

7.5.3 AVB Streaming

The AVB streaming uses Freescale AVB plugins to send a stream over Ethernet. A typical use case is to stream local media files from the talker, with one listener for audio and another listener for video, keeping the two listener's audio and video synchronized.

According to IEC61883-4, compressed video needs to be packetized into its container, so h264parse, mpegtsmux and tsdemux in gst-plugin-bad are needed.

The gst-plugin-bad package is not installed by default in the Yocto Project release, follow the steps below to build and install it.

1. After setting up the Yocto Project build, run the command `bitbake -c compile -f gstreamer1.0-plugins-bad` to build the plugins.
2. Look for the libraries in $yocto_project_root/build/tmp/work/cortexa9hf-vfp-neon-mx6-poky-linux-gnueabi/gstreamer1.0-plugins-bad/1.2.3-r0/gst-plugins-bad-1.2.3
3. Find and copy these libraries to `/usr/lib/gstreamer-1.0`:
   - gst/videoparsers/.libs/libgstvideoparsersbad.so
   - gst/mpegtsmux/.libs/libgstmpegtsmux.so
   - gst/mpegtsdemux/.libs/libgstmpegtsdemux.so
4. Find and copy these libraries to `/usr/lib`:
   - gst-libs/gst/mpegts/.libs/libgstmpegts-1.0.so
   - gst-libs/gst/mpegts/.libs/libgstmpegts-1.0.so.0
   - gst-libs/gst/codecparsers/.libs/libgstcodecparsers-1.0.so
   - gst-libs/gst/codecparsers/.libs/libgstcodecparsers-1.0.so.0

The plugins use IXXAT to get the PTP time. Use the binary file `ptp-1.5.03-yacto` to support this. The command `./ptp-1.5.03-yacto -i 0:eth0 &` will enable this in both the talker and listener sides. IXXAT is not included in the release package. Contact your Freescale marketing representative to get it.

Another way to support PTP clock is to use the provided Linuxptp, launch `ptp4l -A -4 -H -m -i eth0`, and modify `gstavbclock.c` to use it.

Before running the pipeline, input the config command in both the talker and listener sides.

Listener configuration commands:

```
echo 98886080 > /proc/sys/net/core/wmem_max
echo 98886080 > /proc/sys/net/core/rmem_max
echo 98886080 > /proc/sys/net/core/wmem_default
echo 98886080 > /proc/sys/net/core/rmem_default
echo 32767 > /proc/sys/net/core/optmem_max
echo 256 > /proc/sys/net/core/somaxconn
echo 10000 > /proc/sys/net/core/netdev_max_backlog
```

Talker configuration commands:

```
ethtool -C eth0 rx-frames 1
ethtool -C eth0 tx-frames 1
```

Next run this configuration command for both Listener and Talker:

```
./ptp-1.5.03-yacto -i 0:eth0 &
```

Talker commands

```
 $GSTL filesrc location=$FILE_URI ! $CAPS_FILTER ! aiurdemux name=d d.
    ! queue ! h264parse config-interval=0 ! queue ! mpegtsmux ! queue
    ! avbmpegtssink latency=800000000 sync=true d. ! queue ! beepdec ! queue
    ! avbpcmsink latency=1500000000 sync=true
```

For example:

```
$GSTL filesrc location=/home/root/temp/1.mp4 ! video/quicktime
    ! aiurdemux name=d d. ! queue ! h264parse config-interval=0 ! queue
    ! mpegtsmux ! queue ! avbmpegtssink latency=800000000 sync=true d.
    ! queue ! beepdec ! queue ! avbpcmsink latency=1500000000 sync=true
```

Listener commands:

Audio:

```
$GSTL avbpcmsrc ! queue ! audioconvert ! pulsesink sync=true slave-method=2
```

Video:

```
$GSTL avbmpegtssrc ! queue max-size-buffers=0 max-size-bytes=0
    ! tsdemux name=d d. ! queue ! vpudec ! queue ! overlaysink sync=true
```

The properties avbpcmsink and avbmpegtssink are useful for AVB streaming. Latency is the pipeline latency on the listener side when playing an AVB stream. The value should be writen on talker side. As tsdemux on the listener side, it has a latency of 700ms. The latency of avbpcmsink should be 700ms more than that of avbmpegtssink. You may adjust the value to affect the synchronization of audio and video between multi listeners.

7.5.4 Video Conversion

There are two video conversion plugins, imxvideoconvert_ipu and imxvideoconvert_g2d. Both of them can be used to perform video color space conversion, resize, rotate. imxvideoconvert_ipu can be used to perform video deinterlacing also. 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.

Note that imxvideoconvert_g2d can only performance color space converting to RGB space.

CSC

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

gst-launch-1.0 videotestsrc ! video/x-raw,format=YV12 ! imxvideoconvert_g2d ! video/x-
raw,format=RGB16 ! ximagesink display=:0
```

Resize

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

gst-launch-1.0 videotestsrc ! video/x-raw,format=RGB16,width=800,height=600 !
imxvideoconvert_g2d ! video/x-raw, width=640, height=480 ! ximagesink display=:0
```

Rotate

```
gst-launch-1.0 videotestsrc ! imxvideoconvert_ipu rotation=2 ! ximagesink display=:0

gst-launch-1.0 videotestsrc ! imxvideoconvert_g2d rotation=2 ! ximagesink display=:0
```

Deinterlacing

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

Transcoding

```
gst-launch-1.0 filesrc location=$FILE.mp4 typefind=true ! video/quicktime ! aiurdemux !
vpudec ! imxvideoconvert_ipu ! video/x-raw,format=NV12,width=1280,height=720 ! vpuenc !
video/x-h263 ! avimux ! filesink location=$FILE.avi
```

Combination

It is possible to combine CSC, resize, rotate and deinterlace at once time, also both of imxvideoconvert_ipu and imxvideoconvert_g2d can be used at the same time in a pipeline. following is a 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 ! vpudec ! imxvideoconvert_g2d rotation=3 !
video/x-raw,format=RGB16,width=640,height=480 ! ximagesink sync=false display=:0
```

7.5.5 overlaysink Usage

The overlaysink plugin is based on the GPU. It provides two main functions for video rendering:

- Video Overlay: composite multiple video playbacks into same display.
- Multiple Display: show videos to multiple display, up to 4 displays.

overlaysink uses a configuration file to set the display device parameters, located in `/usr/share`. For i.MX 6Q/6DL the configuration file name is `imx_6q_display_config`. For i.MX 6SX/6SL, the configuration file name is `imx_6sx_display_config`.

The configuration syntax is:

[Display Name]

Mandatory. Specify the display name, used in overlaysink property display-x to enable the x display.

device

Mandatory. Specify the display device (v4l2 output) name, eg `/dev/video17`.

fmt

Optional. Specify the display device color format, set "RGBP" for RGB565 16bit display or "RGBx" for RGB8888 32bit display. The default is "RGBP".

width

Optional. Specify the width of the display, default is screen width.

height

Optional. Specify the height of the display, default is screen height.

alpha

Optional. Specify the global alpha value, default is 0.

color_key

Optional. Specify the color key value, default is disabled.

Below is an example of display configuration for dual display mode, LVDS(master) + HDMI(slave).

```
# LVDS display
[master]
device = /dev/video17
fmt = RGBP
width = 1024
height = 768
alpha = 0

# HDMI display
[slave]
device = /dev/video18
fmt = RGBP
width = 1920
height = 1080
```

7.5.6 Installing gstreamer1.0-libav into rootfs

The steps below describe how to install `gstreamer1.0-libav` into a rootfs image.

Step1: Add the following lines into the configuration file `conf/local.conf`

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

Step2: Build gstreamer1.0-libav

```
$ bitbake gstreamer1.0-libav
```

Step3: Build the rootfs image

```
$ bitbake $image_name
```

7.6 Multimedia Information Specific to Gstreamer 0.10

The sections below are specific to Gstreamer 0.10.

7.6.1 Audio Equalizer

```
$GSTL filesrc location=test.mp3 typefind=true ! beepdec ! mfw_audio_pp enable=1 eqmode=2
     ! $audio_sink_plugin

$GSTL $PLAYBIN uri=file:///test.mp3 audio-sink="mfw_audio_pp enable=true eqmode=2
     ! $audio_sink_plugin"
```

The `eqmode` value **2** indicates the “bass booster” scenario.

7.6.2 mfw_isink Usage

The mfw_isink plugin is based on IPU. It provides two main functions for video rendering:

- Video overlay: composites multiple video playbacks into the same display.
- Multiple displays: shows videos to multiple displays, up to four displays.

isink defines an environment variable VSALPHA to control the video visibility:

- VSALPHA = 1: The video is visible.
- VSALHPA=0 or undefined: The video is invisible.

isink uses a configuration file `vssconfig` to set parameters for each display device. The file is located in the `/usr/share` folder, and the configuration syntax is as follows:

[Display Name]

Specify the display name, used in the mfw_isink property display-x to enable the x display.

type

Currently specified to framebuffer.

format

The framebuffer color format.

fb_num

The frame buffer number for this display to show.

vsmax

The maximum videos can be showed for this display, with default value of 4.

main_fb_num

UI framebufer number. Usually, it is 0.

The following is an example of `vssconfig` for dual-display mode, LVDS(master) + HDMI.

```
# master display
[LVDS]
type = framebuffer
format = RGBP
fb_num = 1
main_fb_num = 0
vsmax=4

# slave display

[HDMI]
type = framebuffer
format = RGBP
fb_num = 2
vs_max = 4
```

We have two examples of `vssconfig` installed into the image, vssconfig.dual.lvds_hdmi, and vssconfig.triple.2lvds_hdmi for dual displays (LVDS + HDMI) and triple displays (LVDS + HDMI + LVDS). You can refer to them respectively.

7.6.3 Installing gst-ffmpeg Into the rootfs Image

This section contains brief instructions on adding the `gst-ffmpeg` package to the Yocto Project image. This package is only useful on a SOC without a VPU.

1. To add the package to the image, add the following lines to the configuration file `conf/local.conf` in your build directory.

   ```
   IMAGE_INSTALL_append = " gst-ffmpeg"
   LICENSE_FLAGS_WHITELIST = "commercial"
   ```

2. Build gst-ffmpeg.

   ```
   $ bitbake gst-ffmpeg
   ```

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 will be some variation on what is included based on the build and packages selected, the board, and the backend specified. This section will briefly describe 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 fsl-gpu-sdk

This graphics package contains source for several graphics examples for OpenGLES 2.0 and OpenGLES 3.0 apis for X11, Framebuffer, DirectFB and Wayland 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 6 family of graphic cores. More details about this SDK are in the *i.MX 6 Graphics User's Guide*. This SDK is only supported for hardware that has OpenGLES hardware acceleration.

8.2 G2D - fsl-samples

G2D API (Application Programming Interface) 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.

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 sync
- Contiguous memory allocator
- Supports the 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.

- `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, OpenVG 1.1, and HAL tests.

Below are listed the basic sanity tests which could 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 and i.MX 6DualLite boards.
- es20 - This contains tests for Open GLES 2.0.
  - vv_launcher
    - coverflow.sh
    - vv_launcher

- hal - This contains a variety of 2D unit tests.
  - tvui
  - unit_test
- 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. This can be used as sanity tests.

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 Qt5

Qt5 is built into the Linux image in the Yocto Project environment with the command `bitkake fsl-image-qt5`.

To run the Qt5 examples on the board, the platform and input plugin need to be specified. Different backends require different graphics plugins. The table below shows these.

Table 25. Graphics plugins for backends

Backend	GRAPHICS
FB	eglfs
Wayland	wayland-egl
X11	xcb

- DirectFB is not yet supported.
- Qt5 is only supported on X11 on i.MX 6SoloLite

The command lines for some of the Qt5 sample application follows. For Wayland, it sometimes helps to add --fullscreen to the command line. The Qt5 desktop may contain links to these examples.

```
Qt5_CinematicExperience -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0
/usr/share/qt5nmapcarousedemo-1.0/Qt5_NMap_CarouselDemo -platform ${GRAPHICS} -plugin
evdevtouch:/dev/input/event0
/usr/bin/qt5/qmlscene -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0 /usr/share/
qt5ledscreen-1.0/example_hello.qml
/usr/bin/qt5/qmlscene -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0 /usr/share/
qt5ledscreen-1.0/example_billboard.qml
/usr/bin/qt5/qmlscene -platform ${GRAPHICS} -plugin evdevtouch:/dev/input/event0 /usr/share/
qt5ledscreen-1.0/example_combo.qml
```

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

Table 26. Example Directories and Command Lines

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

8.5 Other Graphics

- `glmark2`

9 Security

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

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

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

An example of the key portion of the output 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 connectivity for Bluetooth and Wi-Fi.

- Bluetooth support works best with a USB dongle with either BlueZ4 or BlueZ5. For this release BlueZ4 is default. To switch between BlueZ4 and BlueZ5 requires a clean build and changes in local.conf. BlueZ4 and BlueZ5 both use different tools. More information about these tools is in the readme-bluez.txt in the meta-fsl-bsp-release/imx/meta-fsl-bluez layer. Bluetooth is enabled in the default kernel config. To disable Bluetooth, run the following command:

  ```
  bitbake linux-imx -c menuconfig
  ```

  Then, disable Bluetooth. Bluetooth works with a USB dongle.
- Wi-Fi is supported with the Atheros hardware using community firmware.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: IMXLUG
Rev. L3.10.53_1.1.0-ga
01/2015