NXP Semiconductors
User's Guide

Document identifier: IMXBSPPG
Rev. L4.14.98_2.3.0, 22 January 2020

i.MX Porting Guide

Contents

Figures

Tables

Chapter 1
Introduction

1.1 Introduction

This document provides an overview on how to develop a custom i.MX solution from an i.MX BSP release. This document describes how to customize kernel changes, U-Boot, memory, and various configurations for a custom hardware solution using an i.MX SoC.

1.2 References

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

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

This release includes the following references and additional information.

- *i.MX Linux® Release Notes* (IMXLXRN) - Provides the release information.
- *i.MX Linux® User's Guide* (IMXLUG) - Contains the information on installing U-Boot and Linux OS and using i.MX-specific features.
- *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG) - Contains the instructions for setting up and building Linux OS in the Yocto Project.
- *i.MX Reference Manual* (IMXLXRM) - Contains the information on Linux drivers for i.MX.
- *i.MX Graphics User's Guide* (IMXGRAPHICUG) - Describes the graphics features.
- *i.MX BSP Porting Guide* (IMXXBSPPG) - Contains the instructions on porting the BSP to a new board.
- *i.MX VPU Application Programming Interface Linux® Reference Manual* (IMXVPUAPI) - Provides the reference information on the VPU API on i.MX 6 VPU.

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

- SABRE Platform Quick Start Guide (IMX6QSDPQSG)
- SABRE Board Quick Start Guide (IMX6QSDBQSG)
- i.MX 6UltraLite EVK Quick Start Guide (IMX6ULTRALITEQSG)
- i.MX 6ULL EVK Quick Start Guide (IMX6ULLQSG)
- SABRE Automotive Infotainment Quick Start Guide (IMX6SABREINFOQSG)
- i.MX 6SoloLite Evaluation Kit Quick Start Guide (IMX6SLEVKQSG)
- i.MX 7Dual SABRE-SD Quick Start Guide (SABRESDBIMX7DUALQSG)
- i.MX 8M Quad Evaluation Kit Quick Start Guide (IMX8MQUADEVKQSG)
- i.MX 8M Mini Evaluation Kit Quick Start Guide (8MMINIEVKQSG)
- i.MX 8QuadXPlus Multisensory Enablement Kit Quick Start Guide (IMX8QUADXPLUSQSG)

Documentation is available online at nxp.com.

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

Chapter 2
Porting Kernel

2.1 Introduction

This chapter explains how to download, build, and load the i.MX kernel both in a standalone environment and through Yocto Project.

2.1.1 How to build and load Kernel in standalone environment

To build Kernel in a standalone environment, first, generate a development SDK, which includes the tools, toolchain, and small rootfs to compile against to put on the host machine.

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

   ```
   DISTRO=fsl-imx-fb MACHINE=Target-Machine bitbake core-image-minimal -c populate_sdk
   ```

2. From the build directory, the bitbake was run in, copy the .sh file in tmp/deploy/sdk to the host machine to build on and execute the script to install the SDK. The default location is in /opt but can be placed anywhere on the host machine.

ARM-v7A (32-bit) and ARM-v8A (64-bit) toolchain script and environment are as follows:

- i.MX 6

  ```
  Toolchain : environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi
  Linux_Config: imx_v7_defconfig
  ARCH=arm
  CROSS_COMPILE=arm-poky-linux-gnueabi-
  ```

- i.MX 7

  ```
  Toolchain : environment-setup-cortexa-7hf-neon-poky-linux-gnueabi
  Linux_Config: imx_v7_defconfig
  ARCH=arm
  CROSS_COMPILE=arm-poky-linux-gnueabi-
  ```

- i.MX 8

  ```
  Toolchain : environment-setup-aarch64-poky-linux
  Linux_Config: defconfig
  ARCH=arm64
  CROSS_COMPILE=aarch64-poky-linux-
  ```

The following are steps to build standalone Kernel sources on the host machine:

1. Set up the host terminal window toolchain environment.

   The environment variables are created in the terminal window after running the `environment-setup-<toolchain>` script. See the information above for i.MX 6, i.MX 7, and i.MX 8 toolchains.

   ```
   $ source <toolchain install directory>/environment-setup-<toolchain script>
   ```

Example for i.MX 8:

```
$ source /opt/fsl-imx-wayland/environment-setup-aarch64-poky-linux
$ echo $LDFLAGS
  -Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
$ unset LDFLAGS   // Remove env variable LDFLAGS
```

Check that new environment variables are correctly set for the target i.MX 8:

```
$ echo $ARCH
  arm64
$ echo $CROSS_COMPILE
  aarch64-poky-linux-
```

2. Get the Linux source code.

```
$ git clone https://source.codeaurora.org/external/imx/linux-imx \
  -b imx_4.14.98-2.0.0_ga
```

The git cloned repository contains all the NXP releases. To work with a different release, the `git tag` command shows available releases and `git checkout -b <tag name>` is used to move to that version.

```


rel_imx_4.14.98_2.0.0_ga
rel_imx_4.14.78_1.0.0_ga
rel_imx_4.14.62_1.0.0_beta
```

For example, choose `rel_imx_4.14.98_2.0.0_ga` snapshot:

```
$ git checkout -b  rel_imx_4.14.98-2.0.0_ga
```

3. Initialize the configuration.

```
$ cd linux-imx
$ make distclean     // delete all generated files
$ make defconfig     // configuration for i.MX 8
                     // see above for i.MX 6 and i.MX 7 configuration name


```

4. Build the kernel sources.

```
$ make -j 8        // -j: number of simulatneous jobs
                   //     use available Host CPUs for number

$ Linux kernel genearted files in directory arch/arm64/boot
  dts
  Image
  Image.gz
  install.sh
  Makefile
```

By default, i.MX U-Boot loads kernel image and device tree blob from the first FAT partition. Users can copy their images to this partition. Alternatively, users can flash images to the RAW address for U-Boot loading.

To flash the kernel generated from the build, execute the following commands:

```
$ sudo dd if=<zImageName> of=/dev/sd<partition> bs=512 seek=2048 conv=fsync && sync
```

To flash the device trees generated from the build, execute the following commands:

```
$ sudo dd if=<DevicetreeName>.dtb of=/dev/sd<partition> bs=512 seek=20480 conv=fsync
```

NOTE

For i.MX 8QuadMax and i.MX 8QuadXPlus, the kernel image and DTB need to be flashed after the first 6 MB of the SD card.

2.1.2 How to build and load Kernel in Yocto Project

To integrate kernel changes in Yocto Project, perform the following steps:

1. Set up a build environment for building the associated SoC on an i.MX reference board in Yocto Project by following the directions in the README either in the manifest branch or in the release layer. This involves using repository initialization and repository synchronization to download the Yocto Project meta data and fsl-setup-release to set up the build environment.
2. Build a reference board kernel for the associated SoC. The following is an example. For the first time, this build is longer because it builds all required tools and dependencies.

   ```
   $ MACHINE=imx6qsabresd bitbake linux-imx
   ```

3. Create a custom layer to hold custom board kernel changes. To create a custom layer, look at the existing i.MX demos for xbmc or iotg for simpler examples. A custom layer is integrated by adding it to the bblayer.conf in the `<build-dir>/conf` directory. The layer must have a `conf/layer.conf` file describing the layer name.
4. Copy an existing machine file associated with the SoC on custom board to the custom layer:

   ```
   $ cp sources/meta-freescale/conf/machine/imx6qsabresd.conf <new layer>/conf/machine/
   <custom_name>.conf
   ```

5. Edit the machine configuration file with device trees listed in the KERNEL_DEVICETREE.
6. Change the preferred version for kernel to build with linux-imx by adding this line to conf/local.conf. There are multiple providers of kernel and this forces the linux-imx version to be used.

   ```
   PREFERRED_PROVIDER_virtual/kernel_<custom_name> = "linux-imx"
   ```

7. Build the custom machine.

   ```
   $ MACHINE=<custom_name> bitbake linux-imx
   ```

   Check in `<build-dir>/tmp/work/<custom_name>-poky-linux-gnueabi/linux-imx/<version>` to find the build output. Also look in `<build-dir>/tmp/deploy/images/<custom_name>` to find the kernel binary.

8. Kernel patches and custom defconfig provided in a linux-imx_%.bbappend with these lines as an example and patch1.patch as a patch placed in

   ```
   sources/<custom_layer>/recipes-kernel/linux-imx/files
   ```

   ```
   FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
   SRC_URI_append = "file://patch1.patch  file://custom_defconfig"
   ```

Chapter 3
Porting U-Boot

3.1 Introduction

This chapter describes how to download, build, and load the i.MX U-Boot in a standalone environment and through the Yocto Project.

3.1.1 How to build U-Boot in standalone environment

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

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

    ```
    DISTRO=fsl-imx-fb MACHINE=Target-Machine bitbake core-image-minimal -c populate_sdk
    ```

    b. From the build directory, the bitbake was run in, copy the sh file in tmp/deploy/sdk to the host machine to build on and execute the script to install the SDK. The default location is in /opt but can be placed anywhere on the host machine.
2. On the host machine, perform the following steps to build U-Boot:
    a. On the host machine, set the environment with the following command before building for i.MX 8 SoC.

    ```
    $ source/opt/fsl-imx-fb/4.14.98/environment-setup-aarch64-poky-linux
    $ export ARCH=arm64
    ```

    b. On the host machine, set the environment with the following command before building for i.MX 6 or i.MX 7 SoC.

    ```
    $ export CROSS_COMPILE=/opt/fsl-imx-fb/4.14.98/environment-setup-cortexa9hf-vfp-neon-poky-
    linux-gnueabi
    $ export ARCH=arm
    ```

    c. To build the U-Boot in the standalone environment, execute the following commands.

    Download source by cloning with:

    ```
    $ git clone https://source.codeaurora.org/external/imx/uboot-imx -b
    imx_v2018.03_4.14.98-2.0.0_ga
    ```

    d. To build U-Boot in the standalone environment, find the configuration for the target boot in the `configs/` directory of the uboot-imx source code. In the following example, i.MX 6ULL is the target.

    ```
    $ cd uboot-imx
    $ make clean
    $ make mx6ull_14x14_evk_defconfig
    $ make u-boot.imx
    ```

    e. To create a custom board, copy a reference defconfig for the associated SoC to a new name and place in the configs folder and build using the new config name.

f. For i.MX 8, use the imx-mkimage tool to combine the U-Boot binary with Arm Trusted Firmware (ATF) and SCUFW to produce the final flash.bin boot image and burn to the SD card. See the imxmkimage tool for details.

g. To burn the boot image to the SD card, execute the following command:

```
dd if=<boot_image> of=/dev/sd<x> bs=1k seek=<offset> conv=fsync
```

Where:

- offset is:
  - 1 - for i.MX 6 or i.MX 7
  - 33 - for i.MX 8QuadMax A0, i.MX 8QuadXPlus A0, i.MX 8M Quad, i.MX 8M Mini
  - 32 - for i.MX 8QuadXPlus B0 and i.MX 8QuadMax B0
- sd<x> is:
  - Device node for the SD card
- boot_image is:
  - u-boot.imx - for i.MX 6 or i.MX 7
  - flash.bin - for i.MX8

3.1.2 How to build and load U-Boot in Yocto Project

To integrate U-Boot changes in Yocto Project, perform the following steps:

1. Set up a build environment for building the associated SoC on an i.MX reference board in Yocto Project by following the directions in the README either in the manifest branch or in the release layer. This involves using repository initialization and repository synchronization to download the Yocto Project meta data and fsl-setup-release to set up the build environment.
2. Build a reference board kernel for the associated SoC. The following is an example. For the first time, this build is longer because it builds all required tools and dependencies.

```
$ MACHINE=imx6qsabresd bitbake u-boot-imx
```

3. Create a custom layer to hold custom board kernel changes. To create a custom layer, look at the existing i.MX demos for xbmc or iotg for simpler examples. A custom layer is integrated by adding it to the bblayer.conf in the `<build-dir>/conf` directory. The layer must have a `conf/layer.conf` file describing the layer name.
4. Copy an existing machine file associated with the SoC on custom board to the custom layer:

```
$ cp sources/meta-freescale/conf/machine/imx6qsabresd.conf <new layer>/conf/machine/
<custom_name>.conf
```

5. Edit the machine configuration file with UBOOT_CONFIG options.
6. Change the preferred version for kernel to build with u-boot-imx by adding this line to conf/local.conf. There are multiple providers of U-Boot and this forces the u-boot-imx version to be used.

```
PREFERRED_PROVIDER_virtual/bootloader_<custom_name> = "u-boot-imx"
```

7. Build the custom machine.

```
$ MACHINE=<custom_name> bitbake u-boot-imx
```

Check in `<build-dir>/tmp/work/<custom_name>-poky-linux-gnueabi/u-boot-imx/<version>` to find the build output. Also look in `<build-dir>/tmp/deploy/images/<custom_name>` to find the boot binaries.

8. U-Boot patches for the custom machine and defconfig can be provided in a u-boot-imx_%.bbappend with these lines as an example and patch1.patch as a patch placed in

```
sources/<custom_layer>/recipes-bsp/uboot-imx/files
```

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
SRC_URI_append = "file://patch1.patch".
```

3.2 Customizing the i.MX custom board code

The new i.MX custom board is a part of the U-Boot source tree, but it is a duplicate of the i.MX reference board code and needs to be customized.

The DDR technology is a potential key difference between the two boards. If there is a difference in the DDR technology, the DDR initialization needs to be ported. DDR initialization is coded in the DCD table, inside the boot header of the U-Boot image. When porting bootloader, kernel or driver code, you must have the schematics easily accessible for reference.

If there is a difference in the DDR technology between the two boards, the DDR initialization needs to be ported. DDR initialization is coded in the DCD table, inside the boot header of the U-Boot image. When porting bootloader, kernel or driver code, you must have the schematics easily accessible for reference.

3.2.1 Changing the DCD table for i.MX DDR initialization

Before initializing the memory interface, configure the relevant I/O pins with the right mode and impedance, and then initialize the MMDC module.

For how to generate calibration parameters for DDR, see i.MX 6 Series DDR Calibration (AN4467). Users can also use the DDR script Aid and DDR stress tools in i.MX Design and Tool Lists for DDR initialization.

1. To port to the custom board, the DDR needs to be initialized properly.
2. Take an example for the i.MX 6Quad custom board. Open the file: board/freescale/mx6<customer_board_name>/imximage.cfg to mx6q.cfg.
3. Modify all the items in this file to match the memory specifications. These code blocks are read by the ROM code to initialize your DDR memory.
4. For i.MX 8QuadMax A0 and i.MX 8QuadXPlus A0, U-Boot does not contain the DCD table for DDR initialization. Users need to update the DCD table file in imx-mkimage to generate the final imx-boot image.
5. For i.MX 8QuadXPlus B0 and i.MX 8QuadMax B0, the DDR initialization codes are in SCFW. Users need to update the DCD table in SCFW and build new SCFW for imx-mkimage.
6. For i.MX 8M Quad, U-Boot does not contain DCD. It depends on SPL to initialize the DDR. SPL contains the codes for DDR PHY and DDR controller initialization and DDR PHY training, so users need to modify the codes.

3.2.2 Booting with the modified U-Boot

This section describes how to compile and write u-boot.imx to an SD card.

If the DDR configuration (board/freescale/<customer_board_name>/imximage.cfg) is modified successfully, you can compile and write u-boot.imx to an SD card. To verify this, insert the SD card into the SD card socket of the CPU board and power on the board.

The following message should be displayed on the console if the board is based on the i.MX 6Quad SABRE_SD:

```
U-Boot 2017.03-00240-gb8760a1 (March 10 2017 - 14:32:18)

CPU:   Freescale i.MX6Q rev1.2 at 792 MHz
CPU:   Temperature 36 C
Reset cause: POR
Board: MX6Q-Sabreauto revA
```

```
I2C:   ready
DRAM:  2 GiB
PMIC:  PFUZE100 ID=0x10
NAND:  0 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
No panel detected: default to Hannstar-XGA
Display: Hannstar-XGA (1024x768)
In:    serial
Out:   serial
Err:   serial
switch to partitions #0, OK
mmc1 is current device
Net:   FEC [PRIME]
Normal Boot
Hit any key to stop autoboot:  0
=>
```

The following message should be displayed on the console if the custom board is based on the i.MX 6SoloLite EVK:

```
U-Boot 2015.04-00240-gb8760a1 (Jul 10 2015 - 14:39:05)

CPU:   Freescale i.MX6SL rev1.2 at 396 MHz
CPU:   Temperature 38 C
Reset cause: POR
Board: MX6SLEVK
I2C:   ready
DRAM:  1 GiB
PMIC:  PFUZE100 ID=0x10
MMC:   FSL_SDHC: 0, FSL_SDHC: 1, FSL_SDHC: 2
In:    serial
Out:   serial
Err:   serial
switch to partitions #0, OK
mmc1 is current device
Net:   FEC [PRIME]
Error: FEC address not set.

Normal Boot
Hit any key to stop autoboot:  0
=>
```

The following message should be displayed on the console if the custom board is based on the i.MX 8QuadMax Validation board:

```
U-Boot 2017.03-imx_4.9.51_8qm_beta1_8qxp_alpha+gc1ec08e (Nov 22 2017 - 00:39:31 -0600)

CPU:   Freescale i.MX8QM revA A53 at 1200 MHz at 12C
Model: Freescale i.MX8QM ARM2
Board: iMX8QM LPDDR4 ARM2
Boot:  SD1
DRAM:  6 GiB
start sata init
SATA link 0 timeout.
MMC:   Actual rate for SDHC_0 is 396000000
Actual rate for SDHC_1 is 396000000
Actual rate for SDHC_2 is 396000000
FSL_SDHC: 0, FSL_SDHC: 1, FSL_SDHC: 2
Run CMD11 1.8V switch
*** Warning - bad CRC, using default environment
```

```
[pcie_ctrla_init_rc] LNK DOWN 8600000
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - SCFW 9f3fa3da, IMX-MKIMAGE 90fbac1a, ATF
  - U-Boot 2017.03-imx_4.9.51_8qm_beta1_8qxp_alpha+gc1ec08e

switch to partitions #0, OK
mmc1 is current device
SCSI:  Net:
Warning: ethernet@5b040000 using MAC address from ROM
eth0: ethernet@5b040000 [PRIME]
Error: ethernet@5b050000 address not set.

Normal Boot
Hit any key to stop autoboot:  0
```

The following message should be displayed on the console if the custom board is based on the i.MX 8M Quad EVK board:

```


U-Boot SPL 2017.03-imx_v2017.03_4.9.51_imx8m_ga+gb026428 (Mar 01 2018 - 03:15:20)
PMIC:  PFUZE100 ID=0x10
start to config phy: p0=3200mts, p1=667mts with 1D2D training
check ddr4_pmu_train_imem code
check ddr4_pmu_train_imem code pass
check ddr4_pmu_train_dmem code
check ddr4_pmu_train_dmem code pass
config to do 3200 1d training.
Training PASS
check ddr4_pmu_train_imem code
check ddr4_pmu_train_imem code pass
check ddr4_pmu_train_dmem code
check ddr4_pmu_train_dmem code pass
config to do 3200 2d training.
Training PASS
check ddr4_pmu_train_imem code
check ddr4_pmu_train_imem code pass
check ddr4_pmu_train_dmem code
check ddr4_pmu_train_dmem code pass
pstate=1: set dfi clk done done
Training PASS
Load 201711 PIE
Normal Boot
Trying to boot from MMC2


U-Boot 2017.03-imx_v2017.03_4.9.51_imx8m_ga+gb026428 (Mar 01 2018 - 03:15:20 -0600)

CPU:   Freescale i.MX8MQ rev2.0 1500 MHz (running at 1000 MHz)
CPU:   Commercial temperature grade (0C to 95C) at 21C
Reset cause: POR
Model: Freescale i.MX8MQ EVK
DRAM:  3 GiB
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110]
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
*** Warning - bad CRC, using default environment

No panel detected: default to HDMI
```

```
Display: HDMI (1280x720)
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF d2cbb20
  - U-Boot 2017.03-imx_v2017.03_4.9.51_imx8m_ga+gb026428

switch to partitions #0, OK
mmc1 is current device
Net:
Warning: ethernet@30be0000 using MAC address from ROM
eth0: ethernet@30be0000
Normal Boot
Hit any key to stop autoboot:  0
u-boot=>
```

3.2.3 Adding new driver initialization code to board files

The following steps describe how to add a new driver and how to initialize the code.

1. Find mx<customer_board>.c in board/freescale/mx<customer_board>/.
2. Edit mx<customer_board>.c and add new driver initialization code, including clock, IOMUX, and GPIO.
3. Put the driver initialization function into board_init or board_late_init.

NOTE

- The **board_early_init_f**() function is called at the very early phase if you define CONFIG_BOARD_EARLY_INIT_F. You can put the UART/SPI-NOR/NAND IOMUX setup function, which requires to be set up at the very early phase.
- The **board_init**() function is called between board_early_init_f and board_late_init. You can do some general board-level setup here. If you do not define CONFIG_BOARD_EARLY_INIT_F, do not call printf before the UART setup is finished. Otherwise, the system may be down.
- The **board_late_init**() function is called later. To debug the initialization code, put the initialization function into it.

3.2.4 Further customization at system boot

To further customize your U-Boot board project, use the first function that system boot calls on:

```
board_init_f in "common/board_f.c"
board_early_init_f()
board_init()
```

All board initialization is executed inside this function. It starts by running through the **init_sequence_f[]** array and **init_sequence_r[]** array of function pointers.

The first board dependent function inside the **init_sequence_f[]** array is **board_early_init_f()**. **board_early_init_f()** is implemented inside board/freescale/mx6<custom board name>.c.

The following line of code is very important:

```
...
setup_iomux_uart();
...
```

NOTE

If a device tree is used, the machine ID is not used. The compatible string of the DTS file is used to match the board. The device tree for file each boot variation is specified in the machine configuration files in the arch/arm/dts directory.

3.2.5 Customizing the printed board name

To customize the printed board name, use the **checkboard()** function.

This function is called from the **init_sequence_f[]** array implemented inside board/freescale/mx6<custom board name>.c. There are two ways to use **checkboard()** to customize the printed board name: the brute force way or by using a more flexible identification method if implemented on the custom board.

To customize the brute force way, delete **identify_board_id( )** inside **checkboard()** and replace printf("Board: "); with printf("Board: i.MX on <custom board>\n");.

If this replacement is not made, the custom board may use another identification method. The identification can be detected and printed by implementing the **__print_board_info()** function according to the identification method on the custom board.

3.3 Debugging

There are two ways to do debugging:

- Using a JTAG tool
- Using printf

3.3.1 Using JTAG tool for debugging

Generally, the JTAG tool is used to debug at a very early stage, for example, before UART initialization, or when it is difficult to debug with printf.

1. Make sure that the JTAG tool supports Arm® Cortex®-A9 cores on i.MX 6 and Arm Cortex-A7 cores on i.MX 7Dual and 6UltraLite, Arm Cortex-A53/A72 on i.MX 8QuadMax and Arm Cortex-A35 on i.MX 8QuadXPlus. It is recommended to use TRACE32.
2. Load U-Boot, which is an elf file, in the root directory of U-Boot fully, or just symbol (faster) to debug step by step.

NOTE

Make optimization level 0 in compiling, which is easier for debugging in the JTAG tool.

3.3.2 Using printf for debugging

This is the most common method used in debugging. You can print your value in the driver for debugging.

NOTE

To use printf in early stages, such as in board_init, put the UART initialization code earlier, such as in the board_early_init_f().

Chapter 4
Porting System Controller Firmware

4.1 Introduction

The System Controller is supported through a firmware also known as SCFW flashed into the boot image on SoC in the i.MX 8 and i.MX 8X families. Each release provides a System Controller Firmware porting kit which includes a porting guide document. For the kernel associated with a BSP, the associated porting kit must be used to ensure compatibility wtih the binaries released in the porting kit. The System Controller porting kit includes both object and source code. The source code provided is for customer enablement of boards which use SoC that have a system controller.

Chapter 5
Configuring OP-TEE

5.1 Introduction

The Trusted Execution Environment (TEE) is a set of specifications published by the GlobalPlatform association (www.globalplatform.org). The purpose of the TEE is to provide a safe environment within the application processor for developing and executing secure applications. We call an application processor a system running a Rich OS like Android or Linux. A rich environment represents a huge amount of code. It is open to third-party applications and it is an open ecosystem: it makes a Rich OS hard to audit. It is prone to bugs/vulnerability, which may compromise the security and integrity of the entire system. The TEE offers another level of protection against attacks from the rich OS. The TEE is only open to trusted partners, which makes it easier to audit. It executes only trusted and authorized software. All sensitive data are protected from the rest of the application processor and from the outside world.

The TEE relies on the Arm TrustZone technology. The TrustZone is a system-on-chip security feature available on most Arm Cortex A/M processors. It provides a strict hardware isolation between the secure world (TEE) and the normal world (REE). This technology allows each physical processor core to provide two virtual cores: one for the normal world and one for the secure world.

OP-TEE is an open source stack of the Trusted Execution Environment. This project includes:

- OP-TEE OS: Trusted side of the TEE
- OP-TEE Client: Normal world client side of the TEE
- OP-TEE Test (or xtest): OP-TEE Test Suite

The OP-TEE project is developed and maintained by Linaro under BSD 2-Clause. The source code is available at https://github.com/OP-TEE. This stack supports ARMv7 and ARMv8 architectures.

The TEE exposes its features through a tandem operation between a Client Application and a Trusted Application. The client application runs in the Rich OS and always initiates the communication with the Trusted Application that runs in the Trusted OS. The Client application interacts with the TEE through the TEE client API interface. The Secure Application interacts with the TEE Core through the TEE Internal API.

TEE GlobalPlatform specifications can be found at https://globalplatform.org/specs-library/.

5.2 Boards supported

The following boards are supported by OP-TEE:

- mx6ulevk
- mx6ul9x9evk
- mx6ullevk - mx6ulzevk (same binary)
- mx6slevk
- mx6sllevk
- mx6sxsabreauto
- mx6sxsabresd
- mx6qsabrelite
- mx6qsabresd
- mx6qsabreauto
- mx6qpsabresd
- mx6qpsabreauto

- mx6dlsabresd
- mx6dlsabreauto
- mx7dsabresd
- mx7ulpevk
- mx8mqevk
- mx8mmevk

5.3 OP-TEE booting flow

Booting flow on i.MX 6 and i.MX 7 (Arm V7):

Files and binaries required in the boot partition:

- u-boot-imx*_sd_optee.imx: U-Boot binary specific to boot OP-TEE. Only booting from the SD card is supported for OP-TEE.
- uTee-*: self-extracting image containing the OP-TEE binary.
- zImage: Kernel image.
- zImage-*.dtb: Device tree.

On i.MX 6 and i.MX 7, the bootloader is U-Boot. To boot OP-TEE, the specific version of U-Boot is required (u-boot-imx<soc>_sd-optee.imx). U-Boot loads OP-TEE OS, Linux OS, and DTB into the memory. U-Boot jumps to OP-TEE OS. OP-TEE OS initializes the secure world and modifies the DTB on the fly to add a specific node to load Linux TEE drivers. Then, it jumps to normal world to boot Linux OS.

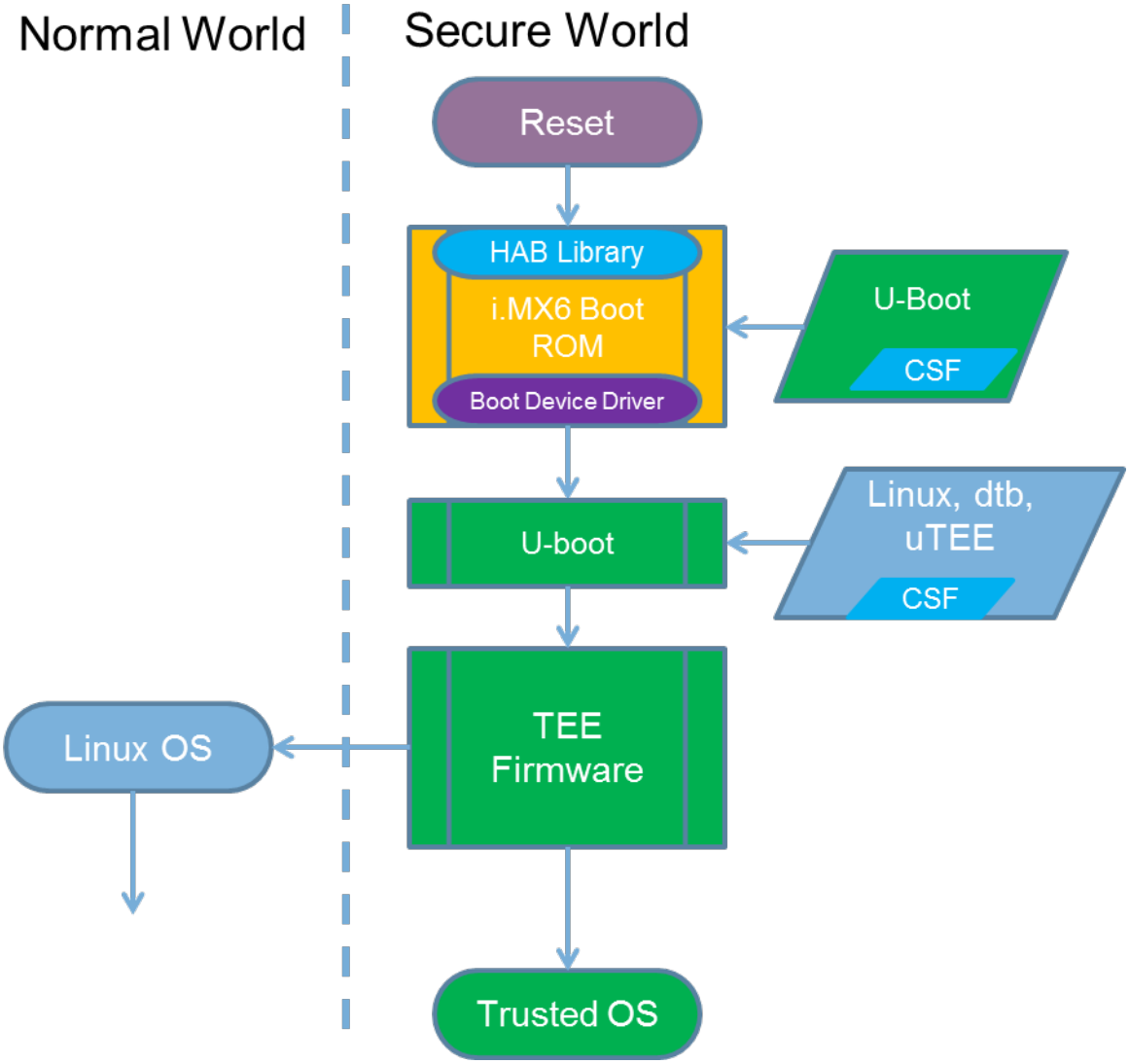

Figure 1. Booting flow on i.MX 6 and i.MX 7

Booting flow on i.MX 8 (Arm V8)

Files and binaries required in the boot partition:

- flash.bin: Fit image containing U-Boot and the ATF
- zImage: Kernel image
- zImage-*.dtb: Device tree

On Arm V8, Arm has a specified preferred way to boot Secure Component with the Arm Trusted Firmware (ATF). The ATF first loads the OP-TEE OS. The OP-TEE OS initializes the secure world. Then, the ATF loads U-Boot that modifies the DTB on the fly to add a specific node to load Linux TEE drivers. Then, the Linux OS is booted.

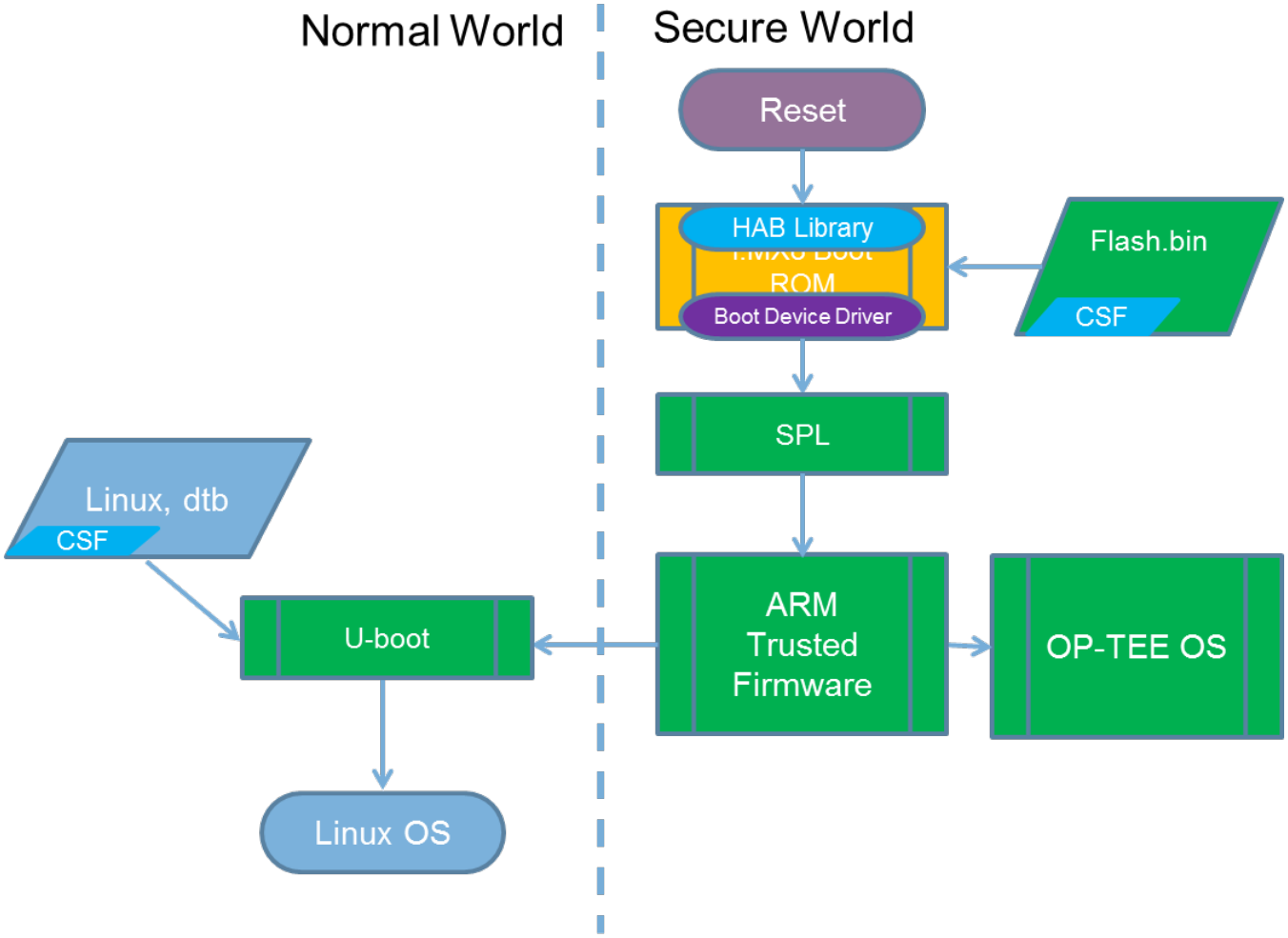

Figure 2. Booting flow on i.MX 8

5.4 OP-TEE Linux support

The Linux TEE driver defines the generic interface to a TEE. See Documentation/tee.txt for more information.

The Linux TEE driver is booted if the following node is present in the device tree:

```
    firmware {
        optee {
            compatible = "linaro, optee-tz";
            method = "smc";
            };
        };
```

This node is added by OP-TEE OS for i.MX 6, i.MX 7, and i.MX 7ULP, and added by U-Boot on i.MX 8M and i.MX 8M Mini.

5.5 Memory protection

OCRAM protection

OCRAM stands for On-Chip RAM. On i.MX 6 and i.MX 7, its size varies between 128 ko or 256 ko. Its main purpose is to hold and execute power management features, such as CPU idle, bus frequency, or suspending. When it is enabled, OP-TEE handles

power management features, such as suspending or CPU idle. Therefore, OP-TEE needs to allocate a secure area in the OCRAM to execute its own power management code. This can be done by configuring the IOMUXC_GPR registers. The lower part is set to non-secure and the upper part is set to secure.

The start address of secure OCRAM and the size are defined in the device tree, “ocram_optee” node:

```
ocram: sram@00905000 {
compatible = "mmio-sram";
reg = <0x00905000 0x3B000>;
clocks = <&clks IMX6QDL_CLK_OCRAM>;
};
ocram_optee: sram@00938000 {
compatible = "fsl,optee-lpm-sram";
reg = <0x00938000 0x8000>;
overw_reg = <&ocram 0x00905000 0x33000>;
};
```

At boot, OP-TEE modifies the “ocram” node of the device tree on the fly. To allocate and secure the OCRAM space, OP-TEE decreases the OCRAM space allocated to the kernel. This is done by modifying the “ocram” node with properties defined in “overw_reg” of the “ocram_optee” node.

NOTE

For i.MX 6SoloX and i.MX 7Dual, there are two types of OCRAM: OCRAM and OCRAM_S. The OCRAM_S is secure or non-secure: It cannot be split into two. In this case, OP-TEE always takes over OCRAM_S for power management features and leaves the OCRAM non-secure, and no OCRAM re-sizing is done.

TZASC380 – RAM protection

The TZC-380 is an IP developed by Arm designed to provide configurable protection over DRAM memory space. Its main feature is to protect security-sensitive software and data in a Trusted Execution Environment (TEE) against potentially compromised software running on the platform. The main features of TZASC are:

- Supports 16 independent address regions.
- Access controls are independently programmable for each address region.
- Sensitive registers may be locked.
- Host interrupt may be programmed to signal attempted access control violations.
- AXI master/slave interfaces for transactions.
- APB slave interface for configuration and status reporting.

Setting TZASC regions

The TZC-380 supports up to 16 independent regions that can be configured to accept or deny a transaction access to a certain DRAM address space. The number of regions that the device provides can be checked in configuration register (Offset 0x0). Except for region 0, you can program the following region parameters:

- Region enable
- Base address
- Size (The minimum address size of a region is 32 KB)
- Subregion permissions

The regions can be overlapped, and the final permission is defined according to the region priority. The priority is defined by the region number. Region 0 is the lowest priority.

32 MB of the RAM space are allocated to OP-TEE: 28 MB is mapped by the TZASC as secure (OP-TEE RAM) and the last 4 MB is mapped as non-secure (shared memory). The start address and the size of this secure memory are hardcoded:

CFG_TZDRAM_START and CFG_TZDRAM_SIZE. These values are added in the device tree by OP-TEE OS after its initialization:

```
/sys/firmware/devicetree/base/reserved-memory/
optee_core@<some_address>
optee@<some address>
```

The optee_core address belongs to the OP-TEE firmware. Any reading or writing from the normal world will result in a crash. The OP-TEE address is the shared memory between Linux OS and OP-TEE. Reading and writing are allowed from the secure and normal worlds.

Example of TZASC configuration for i.MX 6UL:

```
static int board_imx_tzasc_configure(vaddr_t addr)
{
tzc_init(addr);
tzc_configure_region(0, 0x00000000,
    TZC_ATTR_REGION_SIZE(TZC_REGION_SIZE_4G) |
    TZC_ATTR_REGION_EN_MASK | TZC_ATTR_SP_S_RW);
tzc_configure_region(1, 0x80000000,
    TZC_ATTR_REGION_SIZE(TZC_REGION_SIZE_512M) |
    TZC_ATTR_REGION_EN_MASK | TZC_ATTR_SP_NS_RW);
tzc_configure_region(2, 0x84000000,
    TZC_ATTR_REGION_SIZE(TZC_REGION_SIZE_32M) |
    TZC_ATTR_REGION_EN_MASK | TZC_ATTR_SP_S_RW);
tzc_configure_region(3, 0x9fe00000,
    TZC_ATTR_REGION_SIZE(TZC_REGION_SIZE_2M) |
    TZC_ATTR_REGION_EN_MASK | TZC_ATTR_SP_ALL);
tzc_dump_state();
return 0;
}
```

NOTE

On iMX 8QuadMax and iMX 8QuadXPlus, SCFW provides partition concept to divide resources. The 28 MB OPTEE memory (0xFE000000 -- 0xFFC00000) is assigned to secure partition by ATF. It cannot be accessed by non-secure partitions like in U-Boot and kernel. U-Boot fetches the memory regions from the current non-secure partition and sets up memory node for the kernel.

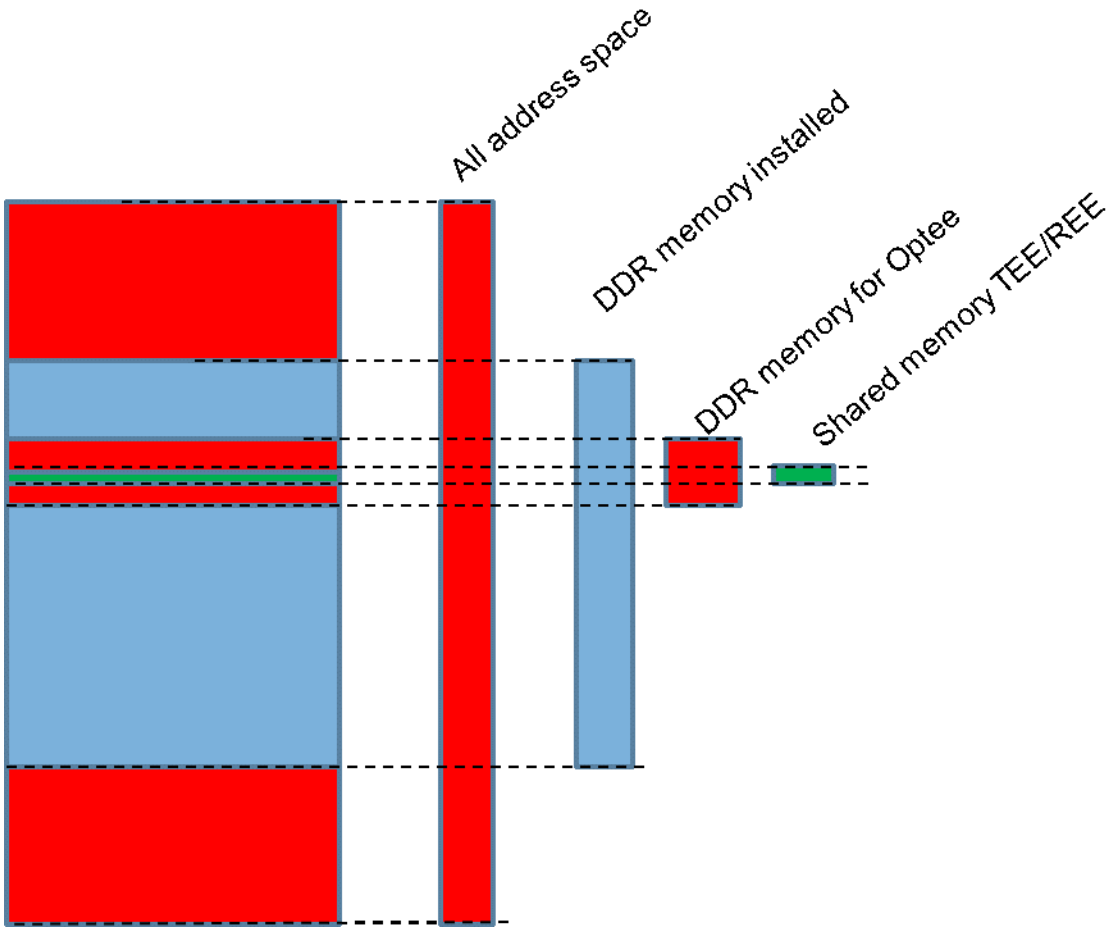

Figure 3. Example of TZASC configuration for i.MX 6UL

- On i.MX 6 and i.MX 7

  The TZASC enablement is done by the U-Boot setting the TZASC_BYPASS bit(s) in IOMUXC_GPR9 register. Once this bit is programmed, the TZASC is taken out of bypass mode and starts to perform security checks on AXI accesses to the DRAM memory. The TZASC_BYPASS bit is a "one time write" type bit. Once it is enabled, it is not possible to change until the next power-up cycle. This prevents an unauthorized disable operation.

- On i.MX 8MQuad and i.MX 8M Mini

  Similarly to i.MX 6 and i.MX 7 families, the TZASC enablement is done by setting a TZASC_EN bit in IOMUXC_GPR10. In mscale family, this bit is not a "one time write" type and TZASC_EN_LOCK must be programmed to avoid unintended disable operation. On i.MX 8M Mini, it is necessary to enable the TZASC_ID_SWAP_BYPASS in IOMUXC_GPR10[1] to avoid an AXI bus error when using GPU. The TZASC ID Swap feature is not correctly handling the AXI Users IDs leading to a GPU crash in certain conditions.

5.6 Compiling OP-TEE

To enable OP-TEE in the Yocto build:

1. Get the Yocto Project build setup completed by the instructions in the *i.MX Yocto Project User's Guide*.
2. Issue bitbakc commands such as `bitbake optee-os optee-client optee-test`.

3. Flash the SD card:

```
$ cd tmp/deploy/images/<platform>/
$ bzip2 -d tmp/deploy/images/<platform>/fsl-image-validation-imx-imx*.sdcard.bz2
$ sudo dd if=fsl-image-validation-imx-imx*.sdcard of=/dev/sd<partition> bs=1M && sync
Run the test suite to check if optee is operational:
$ root@imx: xtest
```

Another way to compile OP-TEE is to use imx-optee-os/scripts/imx-build.sh. Download and install the Linaro toolchains for cross compiling:

```
$ export CROSS_COMPILE=/<path>/arm-linux-gnueabihf-
$ export CROSS_COMPILE64=/<path>/aarch64-linux-gnu-
$ scripts/imx-build.sh <board>
```

5.7 Adding OP-TEE support for a new board

This section describes how to add the OP-TEE OS to a new board.

- U-Boot

  U-Boot must disable the TZASC bypass in registers. To do that, bit(s) must be set in the General-Purpose Registers (IOMUXC_GPR9/10). According to the Reference Manual, check bits to set to disable the TZASC bypass. Do the operation in the following U-Boot source code:

  In /uboot-imx/board/freescale/<platform>/<soc>.cfg, in the device configuration data (DCD), add the following code:

```
#ifdef CONFIG_IMX_OPTEE
DATA 4 <register addr> <value>
CHECK_BITS_SET 4 <register addr> <value>
#endif
```

  In /uboot-imx/board/freescale/<platform>/< platform>.c, the “tee” environment property must be set to “yes” by default:

```
env_set("tee", "no");
#ifdef CONFIG_IMX_OPTEE
    env_set("tee", "yes");
#endif
```

NOTE

OP-TEE can be disabled at any time by setting “env set tee no” in U-Boot environment.

- OP-TEE OS

  In plat-imx/imx-common.c, add a board function identifier, such as “bool soc_imx*(void)”.

  In plat-imx/config/imx*.h, add a board specific header file to define the constant like DRAM0_BASE, DRAM0_SIZE, CFG_UART_BASE, and CONSOLE_UART_BASE. In plat-imx/platform_config.h, add your configuration file.

  In plat-imx/registers/, eventually add board-specific registers.

  In plat-imx/conf.mk, add the new SoC to the platform flavorlist and define the SoC and the number of cores for the the new board:

```
else ifneq ($(filter $(PLATFORM_FLAVOR),$(mx*-flavorlist)))
$(call force,CFG_MX*,y)
$(call force,CFG_MX*,y)
$(call force,CFG_TEE_CORE_NB_CORE,*)
```

Specify the Linux entry address, the device tree address, and the DDR size.

```
ifneq (,$(filter $(PLATFORM_FLAVOR),mx*))
CFG_DT ?= y
CFG_NS_ENTRY_ADDR ?=
CFG_DT_ADDR ?=
CFG_DDR_SIZE ?=
CFG_PSCI_ARM32 ?= y
CFG_BOOT_SYNC_CPU = *
CFG_BOOT_SECONDARY_REQUEST = *
endif
```

In plat-imx/sub.mk, define the Arm processor (Cortex A7 or A9) if the SoC is an Arm V7.

In plat-imx/tzasc.c, configure the secure memory mapping. Most of the time, four regions need to be mapped: the base region, the non-secure region for Linux, the secure space for OP-TEE, and the shared memory space.

In scripts/imx_build.sh, add the new platform flavor to the board_list.

- Linux OS:

  None.

Chapter 6
Configuring Arm Trusted Firmware

6.1 Introduction

Arm Trusted firmware (ATF) is required for all i.MX 8 boards. The Arm Trusted Firmware might need some customization on new boards. ATF currently partitions non-secure resources for the OS partition before launching. When porting to a new board, ATF must be modified for the intended partitioning of system resources with System Controller Firmware.

Chapter 7
Memory Assignment

7.1 Introduction

On i.MX 8QuadMax and i.MX 8QuadXPlus, SCFW provides partition concept to divide resources. The memory is divided into several regions and can only be accessed by particular software modules with corresponding security mode.

Generally, we have two partitions on AP cores. The secure ATF partition owns the critical resources and memory for ATF and OPTEE. The non-secure OS partition owns the resources and memory for kernel and U-Boot. When Arm Cortex-M4 is running, the Arm Cortex-M4 partition is created by SCFW, and resources and memory are assigned.

The typical DDR memory is assigned as shown in the following table on i.MX 8QuadMax MEK board. The regions in bold are accessible for Linux kernel.

Table 1. DDR memory assignment

Memory Type	Start	End	Partition	Reservation	Code
DDR	0x80000000	0x8001FFFF	Secure ATF	Reserved by ATF	`mx8_partition_resources`
	0x80020000	0x801FFFFF	Non-secure OS	Reserved by U-Boot	`dram_init_banksize`
	0x80200000	**0x87FFFFFF**	**Non-secure OS**	-	-
	0x88000000	0x887FFFFF	M4_0	Reserved by SCFW and U-Boot for Cortex-M4	SCFW: `board_system_config` U-Boot: `#define BOOTAUX_RESERVED_MEM_BASE 0x88000000` `#define BOOTAUX_RESERVED_MEM_SIZE 0x08000000`
	0x88800000	0x8FFFFFFF	M4_1	Reserved by SCFW and U-Boot for Cortex-M4	SCFW: `board_system_config` U-Boot: `#define BOOTAUX_RESERVED_MEM_BASE`

Table continues on the next page...

Table 1. DDR memory assignment (continued)

Memory Type	Start	End	Partition	Reservation	Code
					`0x88000000 #define BOOTAUX_RESERVED_MEM_SIZE 0x08000000`
	0x90000000	**0xFDFFFFFF**	**Non-secure OS**	-	-
	0xFE000000	0xFFBFFFFF	Secure ATF	Reserved by ATF for OPTEE	`mx8_partition_resources`
	0xFFC00000	**0xFFFFFFFF**	**Non-secure OS**	-	-
	0x880000000	**0x8C0000000**	**Non-secure OS**	-	-

When Arm Cortex-M4 is running, the following FlexSPI memory is assigned to the Arm Cortex-M4 partition.

Table 2. FlexSPI memory assignment

Memory Type	Start	End	Partition	Reservation	Code
FlexSPI	0x08081000	0x08180FFF	M4_0	Reserved by SCFW for M4_0	`board_system_config`
	0x08181000	0x08181FFF	M4_1	Reserved by SCFW for M4_1	`board_system_config`

In kernel, VPU/RPMSG/DSP drivers reserve DDR memory in DTB. The system cannot allocate memory from these areas. Users can find them in the reserved-memory node as follows:

```
    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        /*
         * reserved-memory layout
         * 0x8800_0000 ~ 0x8FFF_FFFF is reserved for M4
         * Shouldn't be used at A core and Linux side.
         *
         */

        decoder_boot: decoder_boot@0x84000000 {
            no-map;
            reg = <0 0x84000000 0 0x2000000>;
        };
        encoder_boot: encoder_boot@0x86000000 {
            no-map;
            reg = <0 0x86000000 0 0x400000>;
        };
        rpmsg_reserved: rpmsg@0x90000000 {
            no-map;
            reg = <0 0x90000000 0 0x400000>;
```

```
        };
        rpmsg_dma_reserved:rpmsg_dma@0x90400000 {
            compatible = "shared-dma-pool";
            no-map;
            reg = <0 0x90400000 0 0x1C00000>;
        };
        decoder_rpc: decoder_rpc@0x92000000 {
            no-map;
            reg = <0 0x92000000 0 0x200000>;
        };
        encoder_rpc: encoder_rpc@0x92200000 {
            no-map;
            reg = <0 0x92200000 0 0x200000>;
        };
        dsp_reserved: dsp@0x92400000 {
            no-map;
            reg = <0 0x92400000 0 0x2000000>;
        };
        encoder_reserved: encoder_reserved@0x94400000 {
            no-map;
            reg = <0 0x94400000 0 0x800000>;
        };

        /* global autoconfigured region for contiguous allocations */
        linux,cma {
            compatible = "shared-dma-pool";
            reusable;
            size = <0 0x3c000000>;
            alloc-ranges = <0 0x96000000 0 0x3c000000>;
            linux,cma-default;
        };

    };
```

Chapter 8
Configuring IOMUX

8.1 Introduction

Before using the i.MX pins (or pads), select the desired function and correct values for characteristics such as voltage level, drive strength, and hysteresis. You can configure a set of registers from the IOMUX controller.

For detailed information about each pin, see the "External Signals and Pin Multiplexing" chapter or for about the IOMUX controller block, see the "IOMUX Controller (IOMUXC)" in the SoC Application References Manual.

8.1.1 Information for setting IOMUX controller registers

The IOMUX controller contains four sets of registers that affect the i.MX 6Dual/6Quad/6DualLite/6Solo/6SoloLite/6SoloX/6UltraLite/7Dual registers.

- General-purpose registers (IOMUXC_GPR*x*): consist of registers that control PLL frequency, voltage, and other general purpose sets.
- "Daisy Chain" control registers (IOMUXC_<Instance_port>_SELECT_INPUT): control the input path to a module when more than one pad may drive the module's input.
- MUX control registers (changing pad modes):
  - Select which of the pad's eight different functions (also called ALT modes) is used.
  - Set the pad functions individually or by group using one of the following registers:
  - IOMUXC_SW_MUX_CTL_PAD_<PAD NAME>
  - IOMUXC_SW_MUX_CTL_GRP_<GROUP NAME>
- Pad control registers (changing pad characteristics):
  - Set pad characteristics individually or by group using one of the following registers:
  - IOMUXC_SW_PAD_CTL_PAD_<PAD_NAME>
  - IOMUXC_SW_PAD_CTL_GRP_<GROUP NAME>
  - Pad characteristics are:
    - SRE (1 bit slew rate control): Slew rate control bit; selects between FAST/SLOW slew rate output. Fast slew rate is used for high frequency designs.
    - DSE (2 bits drive strength control): Drive strength control bits; selects the drive strength (low, medium, high, or max).
    - ODE (1 bit open drain control): Open drain enable bit; selects open drain or CMOS output.
    - HYS (1 bit hysteresis control): Selects between CMOS or Schmitt Trigger when pad is an input.
    - PUS (2 bits pull up/down configuration value): Selects between pull up or down and its value.
    - PUE (1 bit pull/keep select): Selects between pull up or keeper. A keeper circuit helps assure that a pin stays in the last logic state when the pin is no longer being driven.
    - PKE (1 bit enable/disable pull up, pull down or keeper capability): Enable or disable pull up, pull down, or keeper.
    - DDR_MODE_SEL (1 bit ddr_mode control): Needed when interfacing DDR memories.
    - DDR_INPUT (1 bit ddr_input control): Needed when interfacing DDR memories.

8.1.2 Using IOMUX in the Device Tree - example

The following example shows how to use IOMUX in the Device Tree.

```
       usdhc@0219c000 { /* uSDHC4 */
        fsl,card-wired;
        vmmc-supply = <&reg_3p3v>;
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_usdhc4_1>;
};
iomuxc@020e0000 {
        compatible = "fsl,imx6q-iomuxc";
        reg = <0x020e0000 0x4000>;

        /* shared pinctrl settings */
        usdhc4 {
                pinctrl_usdhc4_1: usdhc4grp-1 {
                        fsl,pins = <
                                MX6QDL_PAD_SD4_CMD__SD4_CMD    0x17059
                                MX6QDL_PAD_SD4_CLK__SD4_CLK    0x10059
                                MX6QDL_PAD_SD4_DAT0__SD4_DATA0 0x17059
                                MX6QDL_PAD_SD4_DAT1__SD4_DATA1 0x17059
                                MX6QDL_PAD_SD4_DAT2__SD4_DATA2 0x17059
                                MX6QDL_PAD_SD4_DAT3__SD4_DATA3 0x17059
                                MX6QDL_PAD_SD4_DAT4__SD4_DATA4 0x17059
                                MX6QDL_PAD_SD4_DAT5__SD4_DATA5 0x17059
                                MX6QDL_PAD_SD4_DAT6__SD4_DATA6 0x17059
                                MX6QDL_PAD_SD4_DAT7__SD4_DATA7 0x17059
                        >;
                };
        };
        ....
};
```

For details, see:

Documentation/devicetree/bindings/pinctrl/fsl,imx-pinctrl.txt

Documentation/devicetree/bindings/pinctrl/fsl,imx6*-pinctrl.txt

Documentation/devicetree/bindings/pinctrl/fsl,imx7*-pinctrl.txt

Documentation/devicetree/bindings/pinctrl/fsl,imx8qm-pinctrl.txt

Documentation/devicetree/bindings/pinctrl/fsl,imx8qxp-pinctrl.txt

Chapter 9
UART

9.1 Introduction

UART is enabled by default. The default UART is configured as follows:

- Baud rate: 115200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None

Chapter 10
Adding SDHC

10.1 Introduction

uSDHC has 14 associated I/O signals.The following list describes the associated I/O signals.

Signal overview

- The SD_CLK is an internally generated clock used to drive the MMC, SD, and SDIO cards.
- The CMD I/O is used to send commands and receive responses to/from the card. Eight data lines (DAT7 - DAT0) are used to perform data transfers between the SDHC and the card.
- The SD_CD# and SD_WP are card detection and write protection signals directly routed from the socket. These two signals are active low (0). A low on SD_CD# means that a card is inserted and a high on SD_WP means that the write protect switch is active.
- SD_LCTL is an output signal used to drive an external LED to indicate that the SD interface is busy.
- SD_RST_N is an output signal used to reset the MMC card. This should be supported by the card.
- SD_VSELECT is an output signal used to change the voltage of the external power supplier. SD_CD#, SD_WP, SD_LCTL, SD_RST_N, and SD_VSELECT are all optional for system implementation. If the uSDHC is desired to support a 4-bit data transfer, DAT7 - DAT4 can also be optional and tied to high voltage.

Adding uSDHC support in the device tree

The following is an example for adding uSDHC support in the device tree:

```
usdhc@02194000 { /* uSDHC2 */
          compatible = "fsl,imx6q-usdhc";
          reg = <0x02194000 0x4000>;
          interrupts = <0 23 0x04>;
          clocks = <&clks 164>, <&clks 164>, <&clks 164>;
          clock-names = "ipg", "ahb", "per";
          pinctrl-names = "default";
          pinctrl-0 = <&pinctrl_usdhc2_1>;
          cd-gpios = <&gpio2 2 0>;
          wp-gpios = <&gpio2 3 0>;
          bus-width = <8>;
          no-1-8-v;
          keep-power-in-suspend;
          enable-sdio-wakeup;
          status = "okay";
};
```

```
usdhc1: usdhc@02190000 {
                                compatible = "fsl,imx6ul-usdhc", "fsl,imx6sx-usdhc";
                                reg = <0x02190000 0x4000>;
                                interrupts = <GIC_SPI 22 IRQ_TYPE_LEVEL_HIGH>;
                                clocks = <&clks IMX6UL_CLK_USDHC1>,
                                         <&clks IMX6UL_CLK_USDHC1>,
                                         <&clks IMX6UL_CLK_USDHC1>;
                                clock-names = "ipg", "ahb", "per";
                                bus-width = <4>;
```

```
                                status = "disabled";
                        };
```

For more information, see:

The binding document at `linux/Documentation/devicetree/bindings/mmc/fsl-imx-esdhc.txt`.

`arch/arm/boot/dts/imx6ul.dtsi`

`arch/arm/boot/dts/imx6ul-14x14-evk.dts`

`arch/arm/boot/dts/imx6qdl.dtsi`

`arch/arm/boot/dts/imx6qdl-sabresd.dtsi`

Support of SD3.0

SD3.0 requires 3.3 V and 1.8 V for signal voltage. Voltage selection needs to be implemented on your platform.

Support of SDIO

In most situations, SDIO requires more power than SD/MMC memory cards. Ensure that the power supply is in the SD slot while using SDIO, or apply an external power to SDIO instead.

Chapter 11
Configuring SPI NOR

11.1 Introduction

This chapter describes how to set up the SPI NOR Flash memory technology device (MTD) driver.

This driver uses the SPI interface to support the SPI-NOR data Flash devices. By default, the SPI NOR Flash MTD driver creates static MTD partitions.

The NOR MTD implementation provides necessary information for the upper-layer MTD driver.

11.1.1 Selecting SPI NOR on the Linux image

To enable support for SPI NOR, perform the following steps:

1. Add the pinctrl for the SPI. For example:

```
pinctrl_ecspi1: ecspi1grp {
                        fsl,pins = <
                                MX6QDL_PAD_EIM_D17__ECSPI1_MISO         0x100b1
                                MX6QDL_PAD_EIM_D18__ECSPI1_MOSI         0x100b1
                                MX6QDL_PAD_EIM_D16__ECSPI1_SCLK         0x100b1
                        >;
                };

                pinctrl_ecspi1_cs: ecspi1cs {
                        fsl,pins = <
                                MX6QDL_PAD_EIM_D19__GPIO3_IO19 0x80000000
                        >;
                };
```

2. Enable the SPI. For example:

```
&ecspi1 {
        fsl,spi-num-chipselects = <1>;
        cs-gpios = <&gpio3 19 0>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecspi1 &pinctrl_ecspi1_cs>;
        status = "okay"; /* pin conflict with WEIM NOR */

        flash: m25p80@0 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "st,m25p32";
                spi-max-frequency = <20000000>;
                reg = <0>;
        };
};
```

11.1.2 Changing the SPI interface configuration

The i.MX 6 SoC has five ECSPI interfaces. The i.MX 7Dual SoC has four ECSPI interfaces. The i.MX 8QuadMax/8QuadXPlus has four LPSPI interfaces. By default, the BSP configures ECSPI-1 interface in master mode to connect to the SPI NOR Flash.

11.1.3 Hardware operation

SPI NOR Flash is SPI compatible with frequencies up to 66 MHz.

The memory is organized in pages of 512 bytes or 528 bytes. SPI NOR Flash also contains two SRAM buffers of 512/528 bytes each, which allows data reception while a page in the main memory is being reprogrammed. It also allows the writing of a continuous data stream.

Unlike conventional Flash memories that are accessed randomly, the SPI NOR Flash accesses data sequentially. It operates from a single 2.7-3.6 V power supply for program and read operations.

SPI NOR Flashes are enabled through a chip select pin and accessed through a three-wire interface: serial input, serial output, and serial clock.

Chapter 12
Connecting LVDS Panel

12.1 Introduction

This chapter describes how to connect the LVDS panel to an i.MX reference board that supports the LVDS interface. Currently the i.MX 6 with IPU and i.MX 8QuadMax and i.MX 8QuadXPlus support the LVDS display interfaces. The implementationn of the LVDS is a DRM driver for i.MX 8 and framebuffer driver for i.MX 6. The LVDS connects to a LVDS Display bridge (LBB) which is configured as a DRM LDB driver for i.MX 8 and a framebuffer driver for i.MX 6.

The i.MX 6 with IPU has an LVDS display bridge (LDB) block that drives LVDS panels without external bridges. The LDB on i.MX with IPU supports the flow of synchronous RGB data from the IPU to external display devices through the LVDS interface.

The LDB support covers the following activities:

- Connectivity to relevant devices-display with an LVDS receiver.
- Arranging the data as required by the external display receiver and by LVDS display standards.
- Synchronization and control capabilities.

12.1.1 Connecting an LVDS panel to the i.MX 8

The LVDS interface on i.MX 8 is implemented with the DRM display framework. This LVDS interface works with the Mixel on i.MX QuadMax and the Mixel Combo on the i.MX 8QuadXPlus both using the it6263 encoder. Both support 1080p resolution. The connection to the it6263 is setup wtih a device tree such as fsl-imx8qxp-mek-it6263-lvds0-dual-channel.dts found in the kenrel repo in arch/arm64/boot/dts/freescale.

12.1.2 Connecting an LVDS panel to the i.MX 6

The kernel command line for 24-bit LVDS panel (4 pairs of LVDS data signals) displays the following line if the panel is properly connected:

```
video=mxcfb0:dev=ldb,if=RGB24
```

The kernel command line for 18-bit LVDS panel (3 pairs of LVDS data signals) displays the following line if the panel is properly connected:

```
video=mxcfb0:dev=ldb,if=RGB666
```

12.2 Enabling an LVDS channel with LDB

When the LDB device is probed by the mxc display core driver, the driver uses platform data information from DTS file to configure the LDB's reference resistor mode and also tries to match video modes for external display devices with an LVDS interface. The display signal polarities and LDB control bits are set according to the matched video modes.

The LVDS channel mapping mode and the LDB bit mapping mode of LDB are set according to the LDB device tree node set by the user.

An LVDS channel is enabled as follows:

1. Set the parent clock of ldb_di_clk and the parent clock rate.
2. Set the rate of ldb_di_clk.
3. Set the LDB in a proper mode, including display signals' polarities, LVDS channel mapping mode, bit mapping mode, and reference resistor mode.

4. Enable both ldb_di_clk and its parent clock.

12.3 LDB ports on i.MX 6

The following figure shows the LDB block.

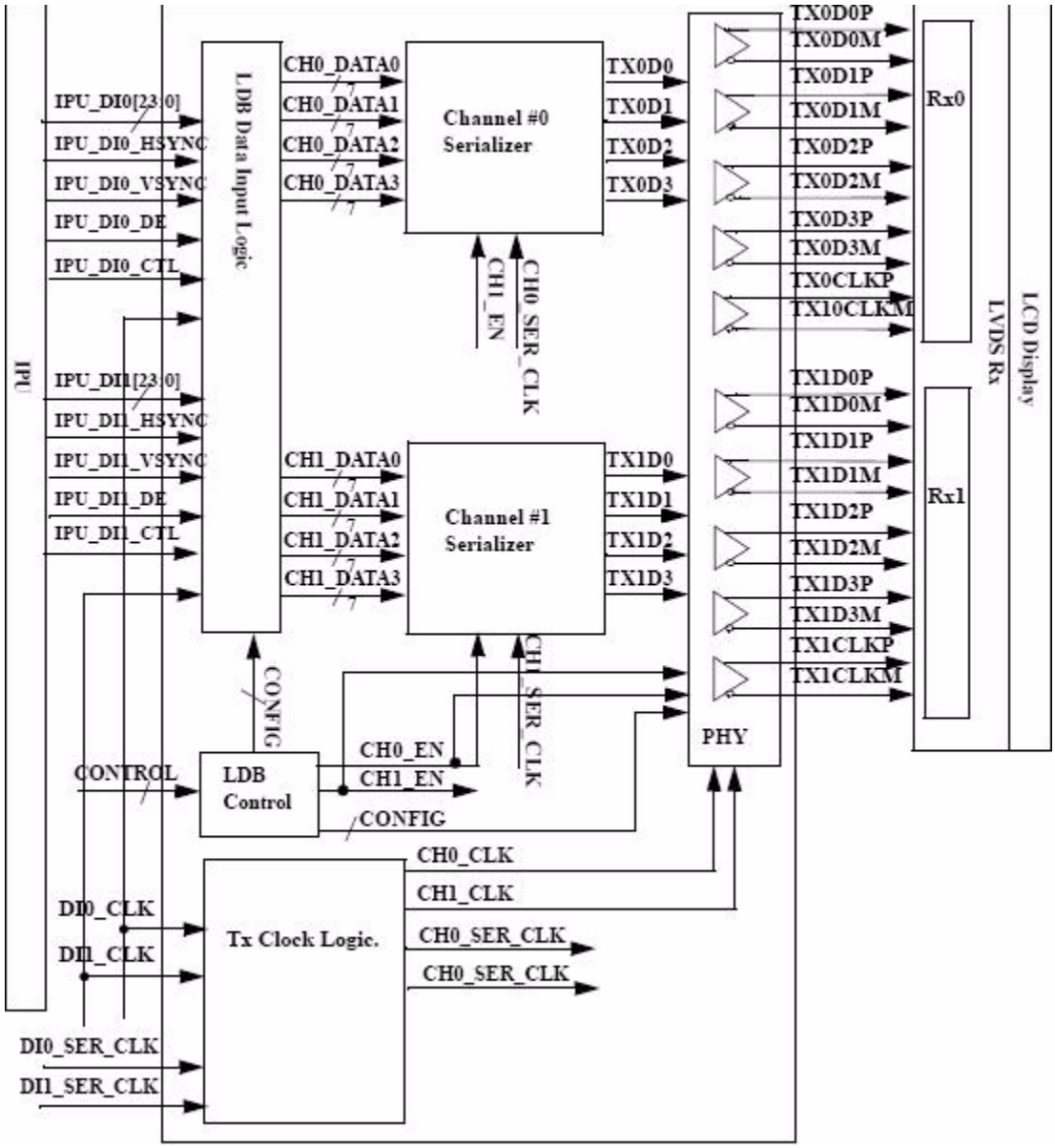

Figure 4. i.MX 6 LVDS Display Bridge (LDB) block

The LDB has the following ports:

- Two input parallel display ports.
- Two output LVDS channels
- Control signals to configure LDB parameters and operations.
- Clocks from the SoC PLLs.

12.3.1 LDB on i.MX 6 for input parallel display ports

The LDB is configurable to support either one or two (DI0, DI1) parallel RGB input ports. The LDB only supports synchronous access mode.

Each RGB data interface contains the following:

- RGB data of 18 or 24 bits
- Pixel clock
- Control signals
- HSYNC, VSYNC, DE, and one additional optional general purpose control
- Transfers a total of up to 28 bits per data interface per pixel clock cycle

The LDB supports the following data rates:

- For dual-channel output: up to 170 MHz pixel clock (such as UXGA-1600 x 1200 at 60 Hz + 35% blanking)
- For single-channel output: up to 85 MHz per interface (such as WXGA-1366 x 768 at 60 Hz + 35% blanking).

12.3.2 LDB on i.MX 6 Output LVDS ports

The LDB has two LVDS channels, which are used to communicate RGB data and controls to external LCD displays either through the LVDS interface or through LVDS receivers. Each channel consists of four data pairs and one clock pair, with a pair indicating an LVDS pad that contains PadP and PadM.

The LVDS ports may be used as follows:

- One single-channel output
- One dual channel output: single input, split to two output channels
- Two identical outputs: single input sent to both output channels
- Two independent outputs: two inputs sent, each to a distinct output channel

Chapter 13
Connecting MIPI-DSI Panel

13.1 Introduction

The MIPI DSI support on i.MX 8 is enabled through the device trees located in the kernel source in arch/arm64/boot/dts/freescale. For more information about MIPI-DSI see the MIPI-DSI section in the Video chapter in Display Interfaces. MIPI-DSI on i.MX with IPU is supported with Synopsys hardware while i.MX 8 uses the Mixel and the Advantec panel.

Chapter 14
Supporting Cameras with CSI

14.1 Introduction

This chapter provides information about how to use the expansion connector to include support for a new camera sensor.

The camera sensor is support on all i.MX but configured using different capture controllers. For more information see the Capture Overview section in the Video chapter in the i.MX Linux Reference Manual. For i.MX 6 with IPU, the CSI interface is thruogh the IPU but on other parts the Parallel CSI driver available to support the CSI interface. For i.MX QuadXPlus it uses an ISI controller and a custom Parallel CSI interface driver.

This chapter will describe the following operations:

- Configuring the CSI unit in test mode (Configuring the CSI unit in test mode)
- Adding support for a new CMOS sensor in the i.MX 6Dual/6Quad/6Solo/6DualLite BSP (Adding support for a new CMOS camera sensor)
- Setting up and using the I^2C interface to handle your camera bus (Using the I2C interface)
- Loading and testing the camera module (Loading and testing the camera module)

It also provides reference information about configuring the CSI interface on i.MX with IPU.:

- Required software and hardware
- Reference IPU-CSI interfaces layout (i.MX 6Dual/6Quad/6Solo/6DualLite CSI interfaces layout)
- CMOS sensor interfaces (IPU-CSI) (CMOS interfaces supported by the i.MX 6Dual/6Quad/6Solo/6DualLite)
- Parallel interface (i.MX 6Dual/6Quad/6Solo/6DualLite CSI parallel interface)
- CSI test mode (Timing data mode protocols)

14.1.1 Required software

In i.MX BSPs, all capture devices support the V4L2 standard. Therefore, only the CMOS-dependent layer needs to be modified to include a new CMOS sensor. All other layers are developed to work with V4L2.

14.1.2 i.MX 6Dual/6Quad/6Solo/6DualLite CSI interfaces layout

The following figure shows the camera interface layout on an i.MX 6Solo/6DualLite SABRE-SD board.

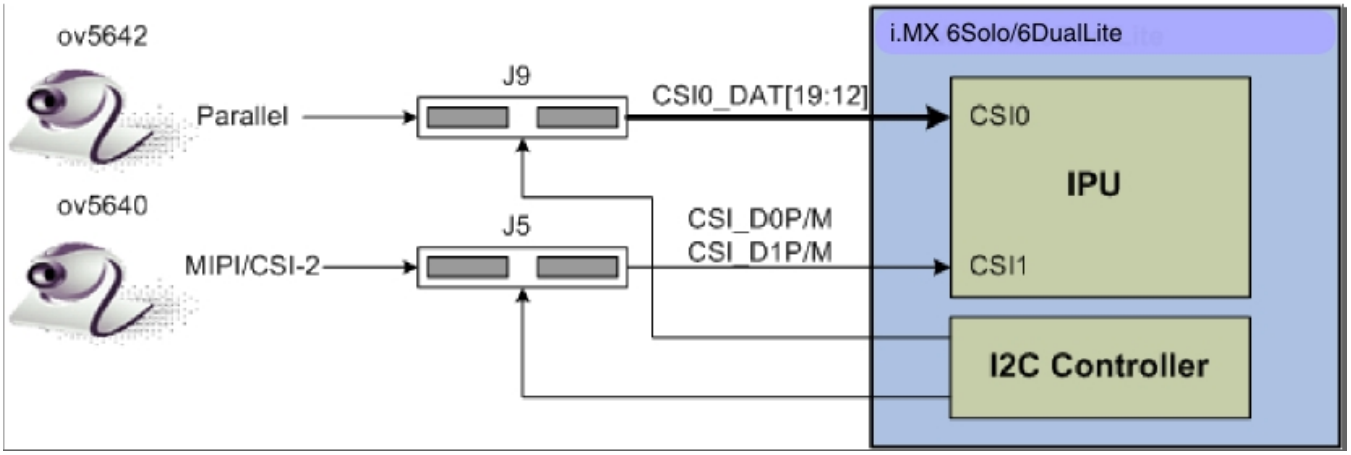

Figure 5. Camera Interface Layout

CSI0 is used as a parallel sensor input interface. CSI1 is used as a MIPI sensor input interface.

14.1.3 Configuring the CSI unit in test mode

This chapter uses the test mode for its example scenario of a new camera driver that generates a chess board.

When you set the TEST_GEN_MODE register, the device is in test mode, which is used for debugging. The CSI generates a frame automatically and sends it to one of the destination units. The sent frame is a chess board composed of black and configured color squares. The configured color is set with the registers PG_B_VALUE, PG_G_VALUE, and PG_R_VALUE. The data can be sent in different frequencies according to the configuration of DIV_RATIO register.

When CSI is in test mode, configure the CSI unit with a similar configuration to the described settings in the following table. Call ipu_csi_init_interface() to configure the CSI interface protocol, formats, and features.

Table 3. Settings for Test Mode

Bit Field	Value	Description
CSI0_DATA_DEST	0x4	Destination is IDMAC through SMFC.
CSI0_DIV_RATIO	0x0	SENSB_MCLK rate = HSP_CLK rate.
CSI0_EXT_VSYNC	0x1	External VSYNC mode.
CSI0_DATA_WIDTH	0x1	8 bits per color.
CSI0_SENS_DATA_FORMAT	0x0	Full RGB or YUV444.
CSI0_PACK_TIGHT	0x0	Each component is written as a 16 bit word where the MSB is written to bit #15. Color extension is done for the remaining least significant bits.
CSI0_SENS_PRTCL	0x1	Non-gated clock sensor timing/data mode.
CSI0_SENS_PIX_CLK_POL	0x1	Pixel clock is inverted before applied to internal circuitry.
CSI0_DATA_POL	0x0	Data lines are directly applied to internal circuitry.
CSI0_HSYNC_POL	0x0	HSYNC is directly applied to internal circuitry.
CSI0_VSYNC_POL	0x0	VSYNC is directly applied to internal circuitry.

14.2 Adding support for a new CMOS camera sensor

To add support for a new CMOS camera sensor to your BSP, create a device driver to support it.

This device driver is the optimal location for implementing initialization routines, the power up sequence, power supply settings, the reset signal, and other desired features for your CMOS sensor. It is also the optimal location to set the parallel protocol used between the camera and the i.MX 6Dual/6Quad/6Solo/6DualLite.

Perform the following three steps on the i.MX 6Dual/6Quad/6Solo/6DualLite BSP to create a device driver:

1. Add a camera sensor entry in Kconfig.
2. Create the camera file.
3. Add compilation flag for the new camera sensor.

These steps are described in detail in the following subsections.

14.2.1 Adding a camera sensor entry in Kconfig

Select specific camera drivers in the following location (as shown in the following figure):

```
Device Drivers  > Multimedia support  > Video capture adapters V4L platform devices > MXC Video For
Linux Camera  >  MXC Camera/V4L2 PRP Features
support
```

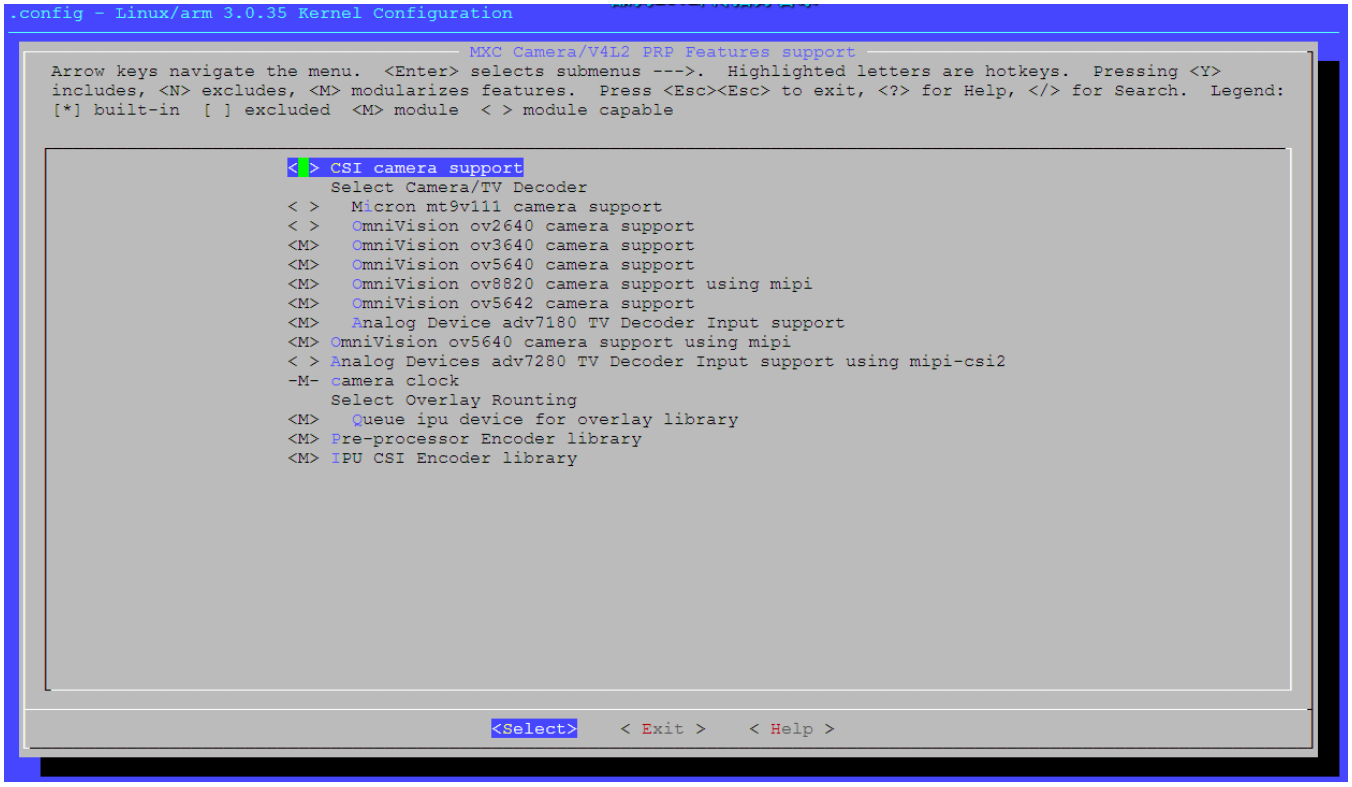

Figure 6. MXC camera/V4L2 PRP features support window

To add a new camera sensor entry on the Kconfig camera file, perform the following steps:

1. Enter the following command into the display-specific folder:

```
$ cd linux/drivers/media/video/mxc/capture
```

2. Open the Kconfig file:

```
$ gedit Kconfig &
```

3. Add the entry where you want it to appear:

```
config MXC_IPUV3_CSI0_TEST_MODE
                    tristate "IPUv3 CSI0 test mode camera support"
                    depends on !VIDEO_MXC_EMMA_CAMERA
                    ---help---
                    If you plan to use the IPUv3 CSI0 in test mode with your MXC system, say Y
here.
```

14.2.2 Creating the camera sensor file

The camera sensor file enables camera initialization, reset signal generation, power settings, and all sensor-specific code.

NOTE

Before connecting a camera sensor to the i.MX 6Dual/6Quad/6Solo/6DualLite board, check whether the sensor is powered with the proper supply voltages and whether the sensor data interface has the correct VIO value. Power supply mismatches can damage either the CMOS or the i.MX 6Dual/6Quad/6Solo/6DualLite.

Create a file with the required panel-specific functions in the following path:

```
linux/drivers/media/video/mxc/capture/
```

The camera file-ipuv3_csi0_chess.c-must include the functions described in the following table and may include additional functions and macros required for your driver.

Table 4. Required functions

Function name	Function declaration	Description
ioctl_g_ifparm	static int ioctl_g_ifparm(struct v4l2_int_device *s, struct v4l2_ifparm *p)	V4L2 sensor interface handler for VIDIOC_G_PARM ioctl.
ioctl_s_power	static int ioctl_s_power(struct v4l2_int_device *s, int on)	V4L2 sensor interface handler for VIDIOC_S_POWER ioctl. Sets sensor module power mode (on or off).
ioctl_g_parm	static int ioctl_g_parm(struct v4l2_int_device *s, struct v4l2_streamparm *a)	V4L2 sensor interface handler for VIDIOC_G_PARM ioctl. Get streaming parameters.
ioctl_s_parm	static int ioctl_s_parm(struct v4l2_int_device *s, struct v4l2_streamparm *a)	V4L2 sensor interface handler for VIDIOC_S_PARM ioctl. Set streaming parameters.
ioctl_g_fmt_cap	static int ioctl_g_fmt_cap(struct v4l2_int_device *s, struct v4l2_format *f)	Returns the sensor's current pixel format in the v4l2_format parameter.
ioctl_g_ctrl	static int ioctl_g_ctrl(struct v4l2_int_device *s, struct v4l2_control *vc)	V4L2 sensor interface handler for VIDIOC_G_CTRL. If the requested control is supported, returns the control's current value from the video_control[] array. Otherwise, it returns -EINVAL if the control is not supported.
ioctl_s_ctrl	static int ioctl_s_ctrl(struct v4l2_int_device *s, struct v4l2_control *vc)	V4L2 sensor interface handler for VIDIOC_S_CTRL. If the requested control is supported, it sets the control's current value in HW (and updates the video_control[] array). Otherwise, it returns -EINVAL if the control is not supported.
ioctl_init	static int ioctl_init(struct v4l2_int_device *s)	V4L2 sensor interface handler for VIDIOC_INT_INIT. Initialize sensor interface.
ioctl_dev_init	static int ioctl_dev_init(struct v4l2_int_device *s)	Initializes the device when slave attaches to the master.
ioctl_dev_exit	static int ioctl_dev_exit(struct v4l2_int_device *s)	De-initializes the device when slave detaches to the master.

After the functions are created, add additional information to ipuv3_csi0_chess_slave and ipuv3_csi0_chess_int_device. The device uses this information to register as a V4L2 device.

The following ioctl function references are included:

```
static struct v4l2_int_slave ipuv3_csi0_chess_slave = {
      .ioctls = ipuv3_csi0_chess_ioctl_desc,
      .num_ioctls = ARRAY_SIZE(ipuv3_csi0_chess_ioctl_desc),
};


static struct v4l2_int_device ipuv3_csi0_chess_int_device = {
   ...
      .type = v4l2_int_type_slave,
   ...
};

static int ipuv3_csi0_chess_probe(struct i2c_client *client,const struct i2c_device_id *id)
{
   ...
      retval = v4l2_int_device_register(&ipuv3_csi0_chess_int_device);
   ...
}
```

It is also necessary to modify other files to prepare the BSP for CSI test mode. Change the sensor pixel format from YUV to RGB565 in the ipu_bg_overlay_sdc.c file so that the image converter does not perform color space conversion and the input received from the CSI test mode generator is sent directly to the memory. Additionally, modify mxc_v4l2_capture.c to preserve CSI test mode settings, which are set by the ipuv3_csi0_chess_init_mode() function in the ipuv3_csi0_chess.c file.

14.2.3 Adding a compilation flag for the new camera

After camera files are created and the Kconfig file has the entry for your new camera, modify the Makefile to create the new camera module during compilation.

The Makefile is located in the same folder as your new camera file and Kconfig: linux/drivers/media/video/mxc/capture.

1. Enter the following into the i.MX 6Dual/6Quad/6Solo/6DualLite camera support folder:

   ```
   $ cd linux/drivers/media/video/mxc/capture
   ```

2. Open the i.MX 6Dual/6Quad/6Solo/6DualLite camera support Makefile.

   ```
   $ gedit Makefile &
   ```

3. Add the CMOS driver compilation entry to the end of the Makefile.

   ```
   ipuv3_csi0_chess_camera-objs := ipuv3_csi0_chess.o

   obj-$(CONFIG_MXC_IPUV3_CSI0_TEST_MODE) += ipuv3_csi0_chess_camera.o
   ```

The kernel object is created by using the ipuv3_csi0_chess.c file. You should have the following files as output:

- ipuv3_csi0_chess_camera.mod.c
- ipuv3_csi0_chess.o
- ipuv3_csi0_chess_camera.o
- ipuv3_csi0_chess_camera.mod.o

- ipuv3_csi0_chess_camera.ko

14.3 Using the I2C interface

Many camera sensor modules require a synchronous serial interface for initialization and configuration.

This section uses the linux/drivers/media/video/mxc/capture/ov5642.c file as its example code. This file contains a driver that uses the I2C interface for sensor configuration.

After the I2C interface is running, create a new I2C device to handle your camera bus. If the camera sensor file (called mycamera.c in the following example code) is located in the same folder as ov5642.c, the code is as follows:

```
struct i2c_client * mycamera_i2c_client;

static s32 mycamera_read_reg(u16 reg, u8 *val);
static s32 mycamera_write_reg(u16 reg, u8 val);

static const struct i2c_device_id mycamera_id[] = {
   {"mycamera", 0},
   {},
};

MODULE_DEVICE_TABLE(i2c, mycamera_id);

static struct i2c_driver mycamera_i2c_driver = {
      .driver = {
              .owner = THIS_MODULE,
        .name  = "mycamera",
              },
      .probe  = mycamera_probe,
      .remove = mycamera_remove,
      .id_table = mycamera_id,
};


static s32 my_camera_write_reg(u16 reg, u8 val)
{
   u8 au8Buf[3] = {0};
   au8Buf[0] = reg >> 8;
   au8Buf[1] = reg & 0xff;
   au8Buf[2] = val;
   if (i2c_master_send(my_camera_i2c_client, au8Buf, 3) < 0) {
      pr_err("%s:write reg error:reg=%x,val=%x\n",__func__, reg, val);
      return -1;
   }
   return 0;
}

static s32 my_camera_read_reg(u16 reg, u8 *val)
{
   u8 au8RegBuf[2] = {0};
   u8 u8RdVal = 0;
   au8RegBuf[0] = reg >> 8;
   au8RegBuf[1] = reg & 0xff;

   if (2 != i2c_master_send(my_camera_i2c_client, au8RegBuf, 2)) {
      pr_err("%s:write reg error:reg=%x\n",__func__, reg);
      return -1;
   }
```

```
   if (1 != i2c_master_recv(my_camera_i2c_client, &u8RdVal, 1)) {//  @ECA
      pr_err("%s:read reg error:reg=%x,val=%x\n",__func__, reg, u8RdVal);
   return -1;
   }

   *val = u8RdVal;
   return u8RdVal;
}

static int my_camera_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
   ...
   my_camera_i2c_client = client;
   ...
}


static __init int mycamera_init(void)
{
   u8 err;
   err = i2c_add_driver(&mycamera_i2c_driver);
   if (err != 0)
      pr_err("%s:driver registration failed, error=%d \n",__func__, err);
   return err;
}


static void __exit mycamera_clean(void)
{
   i2c_del_driver(&mycamera_i2c_driver);
}

module_init(mycamera_init);
module_exit(mycamera_clean);
```

Check ov5642.c for the complete example code.

After creating the new I^2C device driver, add a new I2C node to your platform dts file.

You may modify the dts file at this point to specify features about your camera such as the CSI interface used (CSI0 or CSI1), the MCLK frequency, and some power supply settings related to the module.

You can now read and write from/to the sensor in the camera sensor file by using the following:

```
retval = mycamera_write_reg(RegAddr, Val);
retval = mycamera_read_reg(RegAddr, &RegVal);
```

14.3.1 Loading and testing the camera module

If your camera driver is created as a kernel module, as in the example in this chapter, the module must be loaded prior to any camera request attempt.

According to the Makefile information, the camera module is named ipuv3_csi0_chess_camera.ko.

To load the V4L2 camera interface and CSI in test mode, execute the following commands:

```
root@freescale /unit_tests$ modprobe ipuv3_csi0_chess_camera
root@freescale /unit_tests$ modprobe mxc_v4l2_capture
```

To test the video0 input (camera), an mxc_v4l2_overlay test is included in the BSP. If the imx-test package has also been included, open the unit test folder and execute the test.

```
root@freescale ~$ cd /unit_tests/
root@freescale /unit_tests$ ./mxc_v4l2_overlay.out
```

14.4 Additional reference information

14.4.1 CMOS interfaces supported by the i.MX 6Dual/6Quad/6Solo/6DualLite

The camera sensor interface, which is a part of the image processing unit (IPU) module on the i.MX 6Dual/6Quad/6Solo/6DualLite, handles CMOS sensor interfaces. The i.MX 6Dual/6Quad/6Solo/6DualLite IPU is able to handle two camera devices through its CSI ports: one connected to the CSI0 port and the other to the CSI1 port. Both CSI ports are identical and provide glueless connectivity to a wide variety of raw/smart sensors and TV decoders.

Each of the camera ports includes the following features:

- Parallel interface
  - Up to 20-bit input data bus
  - A single value in each cycle
  - Programmable polarity
- Multiple data formats
  - Interleaved color components, up to 16 bits per value (component)
  - Input Bayer RGB, Full RGB, or YUV 4:4:4, YUV 4:2:2 Component order:UY1VY2 or Y1UY2V, grayscale and generic data
- Scan order: progressive or interlaced
- Frame size: up to 8192 x 4096 pixels
- Synchronization-video mode
  - The sensor is the master of the pixel clock (PIXCLK) and synchronization signals.
  - Synchronization signals are received by using either of the following methods:
    - Dedicated control signals-VSYNC, HSYNC-with programmable pulse width and polarity.
    - Controls embedded in the data stream following loosely the BT.656 protocol with flexibility in code values and location.
  - The image capture is triggered by the MCU or by an external signal (such as a mechanical shutter).
  - Synchronized strobes are generated for up to six outputs-the sensor and camera peripherals (flash, mechanical shutter...).
- Frame rate reduction by periodic skipping of frames.

For details, see the "Image Processing Unit (IPU)" chapter in the *i.MX 6Dual/6Quad Applications Processor Reference Manual* (IMX6DQRM) or *i.MX 6Solo/6DualLite Applications Processor Reference Manual* (IMX6SDLRM). The following figure shows the block diagram.

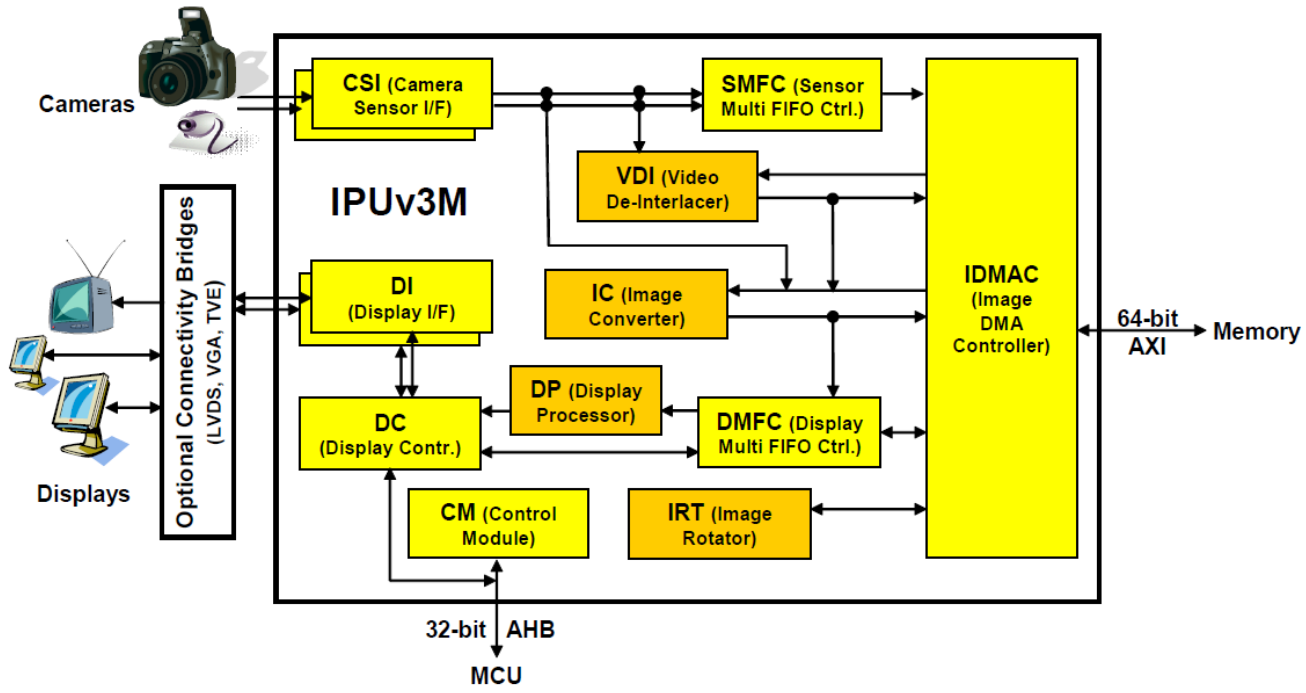

Figure 7. IPU block diagram

Several sensors can be connected to each of the CSIs. Simultaneous functionality (for sending data) is supported as follows:

- Two sensors can send data independently, each through a different port.
- One stream can be transferred to the VDI or IC for on-the-fly processing while the other one is sent directly to system memory.

The input rate supported by the camera port is as follows:

- Peak: up to 180 MHz (values/sec).
- Average (assuming 35% blanking overhead) for YUV 4:2:2.
  - Pixel in one cycle (BT.1120): up to 135 MP/sec, such as 9 Mpixels at 15 fps.
  - Pixel on two cycles (BT.656): up to 67 MP/sec, such as 4.5 Mpixels at 15 fps.
- On-the-fly processing may be restricted to a lower input rate.

If required, additional cameras can be connected through the USB port.

14.4.2 i.MX 6Dual/6Quad/6Solo/6DualLite CSI parallel interface

The CSI obtains data from the sensor, synchronizes the data and the control signals to the IPU clock (HSP_CLK), and transfers the data to the IC and/or SMFC.

The CSI parallel interface,as shown in the following figure, provides a clock output (MCLK), which is used by the sensor as a clock input reference. The i.MX 6Dual/6Quad/6Solo/6DualLite requests either video or still images through a different interface between the processor and the camera module. In most situations, the interface is a synchronous serial interface such as the I^2C. After the frame has been requested, the camera module takes control of the CSI bus, and uses synchronization signals VSYNC, HSYNC, DATA_EN and PIXCLK to send the image frame to the i.MX 6Dual/6Quad/6Solo/6DualLite. The camera sensor creates PIXCLK based on MCLK input.

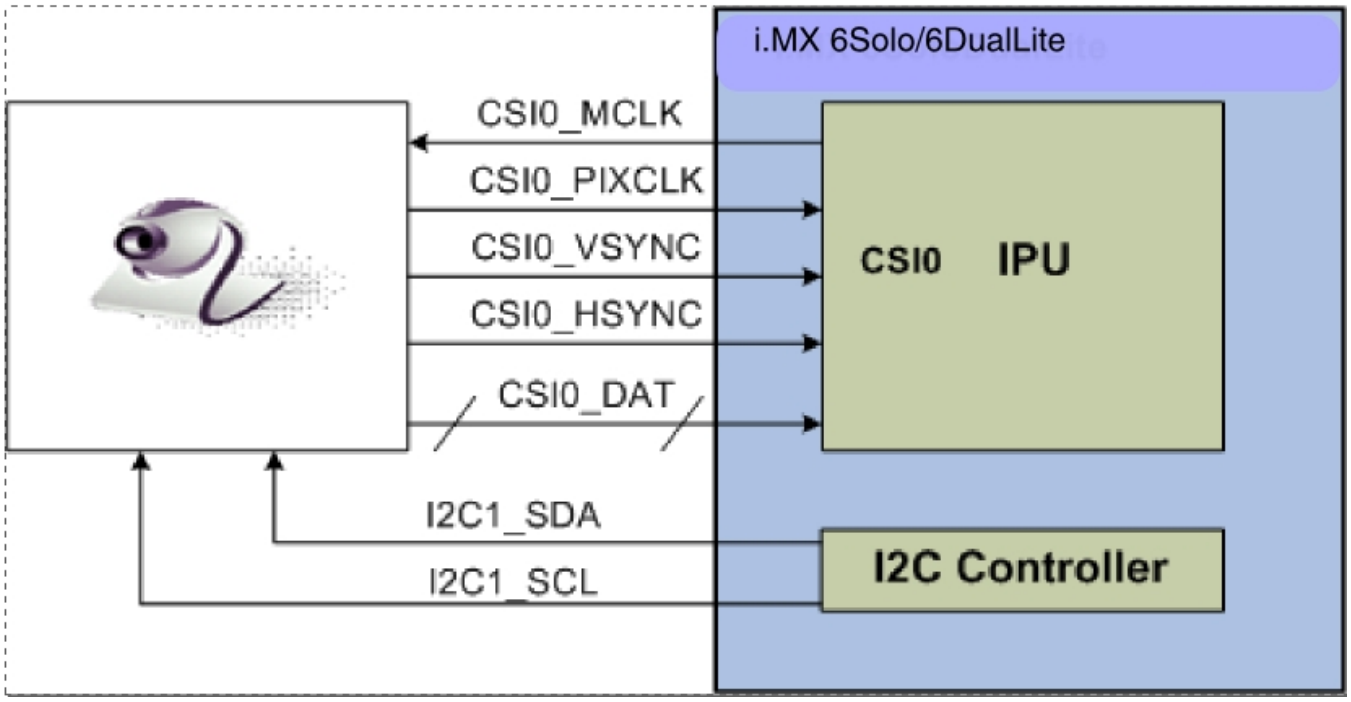

Figure 8. Parallel interface layout

In parallel interface, a single value arrives in each clock, except in BT.1120 mode when two values arrive per cycle. Each value can be 8-16 bits wide according to the configuration of DATA_WIDTH. If DATA_WIDTH is configured to N, then 20-N LSB bits are ignored.

Therefore, you never need CSI0_DAT[3:0], unless you are using BT.1120 mode, because the maximum pixel width is 16 (CSI0_DAT[19:4]). The expansion port 2 includes CSI0_DAT[19:4], but only CSI0_DAT[19:10] are used for the CSI data bus (10-bit wide data). CSI0_DAT[9:4] are shared with other interfaces and are used for audio and I^2C.

CSI can support several data formats according to SENS_DATA_FORMAT configuration. When the data format is YUV, the output of the CSI is always YUV444-even if the data arrives in YUV422 format.

The polarity of the inputs can be configured using the following registers:

- SENS_PIX_CLK_POL
- DATA_POL
- HSYNC_POL
- VSYNC_POL

The following table describes the camera parallel interface provided by the i.MX 6Dual/6Quad/6Solo/6DualLite:

Table 5. CSI0 parallel interface signals

Signal	IPU Pin	Description
MCLK	CSI0_MCLK	Master clock (Output)
PIXCLK	CSI0_PIXCLK	Pixel clock
VSYNC	CSI0_VSYNC	Vertical synchronization signal
HSYNC	CSI0_HSYNC	Horizontal synchronization signal

Table continues on the next page...

Table 5. CSI0 parallel interface signals (continued)

Signal	IPU Pin	Description
DATA_EN	CSI0_DATA_EN	Data enable or data ready
DATA[19:10]	CSI0_DAT [19:10]	Pixel data bus, optional to [19:4]

The following section explains how the timing data mode protocols use these signals. Not all signals are used in each timing data mode protocol.

14.4.3 Timing data mode protocols

The CSI interface supports the following four timing/data protocols:

- Gated mode
- Non-gated mode
- BT.656 (Progressive and interlaced)
- BT.1120 (Progressive and interlaced)

In gated mode, VSYNC is used to indicate the beginning of a frame, and HSYNC is used to indicate the beginning of a raw. The sensor clock is always ticking.

In non-gated mode, VSYNC is used to indicate the beginning of a frame, and HSYNC is not used. The sensor clock only ticks when data is valid.

In BT.656 mode, the CSI works according to recommendation ITU-R BT.656. The timing reference signals (frame start, frame end, line start, line end) are embedded in the data bus input.

In BT1120 mode, the CSI works according to recommendation ITU-R BT.1120. The timing reference signals (frame start, frame end, line start, line end) are embedded in the data bus input.

For details, see the *i.MX 6Dual/6Quad Applications Processor Reference Manual* (IMX6DQRM) or *i.MX 6Solo/6DualLite Applications Processor Reference Manual* (IMX6SDLRM).

Chapter 15
Supporting Cameras with MIPI-CSI

15.1 Introduction

This chapter describes how to configure the MIPI-CSI cameras on the i.MX 7 and i.MX8. For more information on MIPI-CSI see the Capture Overview section in the Video chapter in the i.MX Linux Reference Manual.

The i.MX 7 family uses the Samsung hardware while i.MX 8 uses the Mixel hardware.

Chapter 16
Porting Audio Codecs

16.1 Introduction

This chapter describes how to port audio drivers from the i.MX reference board to a custom board.

This procedure varies depending on whether the audio codec on the custom board is the same as, or different than the audio codec on the NXP reference design. This chapter first describes the common porting task and then various other porting tasks.

Common porting tasks for configuring audio codecs requireing ALSA customizations. To use the ALSA Audio function, CPU DAI driver, CODEC DAI driver, and DAI LINK driver machine driver) should be registered in the device tree, and accordingly there must be three nodes in the board specified dts file. Device trees are located in arch/arm/boot/dts for i.MX 6 and i.MX 7 and arch/arm64/boot/dts for all i.MX 8. An example of detailed nodes can be found in arch/arm/boot/dts/imx6qdl-sabresd.dtsi:

```
/* DT binding for CPU DAI driver */
ssi2: ssi@0202c000 {
    fsl,mode = "i2s-slave";
    status = "okay";
};

/* DT binding for CODEC DAI driver */
codec: wm8962@1a {
    compatible = "wlf,wm8962";
    reg = <0x1a>;
    clocks = <&clks 169>;
    DCVDD-supply = <&reg_audio>;    /* 1.8v */
    DBVDD-supply = <&reg_audio>;    /* 1.8v */
    AVDD-supply = <&reg_audio>;     /* 1.8v */
    CPVDD-supply = <&reg_audio>;    /* 1.8v */
    MICVDD-supply = <&reg_audio>;   /* 3.3v */
    PLLVDD-supply = <&reg_audio>;   /* 1.8v */
    SPKVDD1-supply = <&reg_audio>;  /* 4.2v */
    SPKVDD2-supply = <&reg_audio>;  /* 4.2v */
    gpio-cfg = <
        0x0000 /* 0:Default */
        0x0000 /* 1:Default */
        0x0013 /* 2:FN_DMICCLK */
        0x0000 /* 3:Default */
        0x8014 /* 4:FN_DMICCDAT */
        0x0000 /* 5:Default */
    >;
};

/* DT binding for DAI LINK driver */
sound {
    compatible = "fsl,imx6q-sabresd-wm8962",
        "sl,imx-audio-wm8962";
    model = "wm8962-audio";
    si-controller = <&ssi2>;
    udio-codec = <&codec>;

    audio-routing =
        "Headphone Jack", "HPOUTL",
        "Headphone Jack", "HPOUTR",
        "Ext Spk", "SPKOUTL",
```

```
        "Ext Spk", "SPKOUTR",
        "MICBIAS", "AMIC",
        "IN3R", "MICBIAS",
        "DMIC", "MICBIAS",
        "DMICDAT", "DMIC";
    mux-int-port = <2>;
    mux-ext-port = <3>;
    hp-det-gpios = <&gpio7 8 1>;    /*active low*/
    mic-det-gpios = <&gpio1 9 1>;   /*active low*/
};
```

NOTE

The specific meaning of the device tree binding can be checked up in binding doc located in Documentation/devicetree/bindings/sound/.

16.1.1 Porting the reference BSP to a custom board (audio codec is the same as in the reference design)

When the audio codec is the same in the reference design and the custom board, ensure that the I/O signals and the power supplies to the codec are properly initialized to port the reference BSP to the custom board.

Devicetree uses pin control group for I/O signals' configuration. There are some examples in arch/arm/boot/dts/imx6qdl-sabresd.dtsi and the definitions of those pin control groups can be found in arch/arm/boot/dts/imx6qdl.dtsi.

The essential signals for wm8962 codec are as follows:

- I^2C interface signals
- I^2S interface signals
- SSI external clock input to wm8962

The following table shows the required power supplies for the wm8962 codec.

Table 6. Required power supplies

Power Supply Name	Definition	Value
PLLVDD	PLL supply	1.8 V
SPKVDD1	Supply for left speaker drivers	4.2 V
SPKVDD2	Supply for right speaker drivers	4.2 V
DCVDD	Digital core supply	1.8 V
DBVDD	Digital supply	1.8 V
AVDD	Analog supply	1.8 V
CPVDD	Charge pump power supply	1.8 V
MICVDD	Microphone bias amp supply	3.3 V

16.1.2 Porting the reference BSP to a custom board (audio codec is different from the reference design)

When adding support for an audio codec that is different from the one on the reference design, create new ALSA drivers to port the reference BSP to a custom board. The ALSA drivers plug into the ALSA sound framework, which enables the standard ALSA interface to be used to control the codec.

The source code for the ALSA driver is located in the Linux kernel source tree at linux/sound/soc. The following table shows the files used for the wm8962 codec support:

Table 7. Files for wm8962 codec support

File name	Definition
imx-pcm-dma.c	• Shared by the stereo ALSA SoC driver, the esai driver, and the spdif driver. • Responsible for preallocating DMA buffers and managing DMA channels.
fsl_ssi.c	• Register the CPU DAI driver for the stereo ALSA SoC. • Configures the on-chip SSI interfaces.
wm8962.c	• Register the stereo codec and Hi-Fi DAI drivers. • Responsible for all direct hardware operations on the stereo codec.
imx-wm8962.c	• Machine layer code. • Create the driver device. • Register the stereo sound card.

NOTE

If using a different codec, adapt the driver architecture shown in the table above accordingly. The exact adaptation depends on the codec chosen. Obtain the codec-specific software from the codec vendor.

Chapter 17
Porting HiFi4

17.1 Introduction

The HiFi-4 DSP framework is provided on specific i.MX 8QuadXPlus SoC and i.MX 8QuadMax SoC. Supporting the HiFi4 on a custom board is documented in the i.MX DSP Porting Guide available with the i.MX DSP Redistibution package available to customers who have a HiFi4 license with Cadence.

Chapter 18
Porting Ethernet

18.1 Introduction

This chapter explains how to port the Ethernet controller driver to the i.MX 6 or i.MX 7 processor.

Using i.MX FEC standard driver makes porting simple. Porting needs to address the following three areas:

- Pin configuration
- Source code
- Ethernet connection configuration

18.1.1 Pin configuration

The Ethernet Controller supports three different standard physical media interfaces: a reduced media independent interface (RMII), a media independent interface (MII), and a 4-bit reduced RGMII.

In addition, the Ethernet Controller includes support for different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The i.MX Ethernet Controller supports the 10/100 Mbps MII, and 10/100 Mbps RMII. The i.MX 6Dual/6Quad/6Solo/6DualLite/6SoloX FEC also supports 1000 Mbps RGMII, which uses 4-bit reduced GMII operating at 125 MHz.

A brief overview of the device functionality is provided here. For details, see the Ethernet chapter of the related Applications Processor Reference Manual.

In MII mode, there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. MII, RMII, and RGMII modes use a subset of the 18 signals. These signals are listed in the following table.

Table 8. Pin usage in MII RMII and RGMII modes

Direction	EMAC pin name	MII usage	RMII usage	RGMII usage (not supported by i.MX 6SoloLite)
In/Out	FEC_MDIO	Management Data Input/Output	Management Data Input/output	Management Data Input/Output
Out	FEC_MDC	Management Data Clock	General output	Management Data Clock
Out	FEC_TXD[0]	Data out, bit 0	Data out, bit 0	Data out, bit 0
Out	FEC_TXD[1]	Data out, bit 1	Data out, bit 1	Data out, bit 1
Out	FEC_TXD[2]	Data out, bit 2	Not Used	Data out, bit 2
Out	FEC_TXD[3]	Data out, bit 3	Not Used	Data out, bit 3
Out	FEC_TX_EN	Transmit Enable	Transmit Enable	Transmit Enable
Out	FEC_TX_ER	Transmit Error	Not Used	Not Used
In	FEC_CRS	Carrier Sense	Not Used	Not Used
In	FEC_COL	Collision	Not Used	Not Used

Table continues on the next page...

Table 8. Pin usage in MII RMII and RGMII modes (continued)

Direction	EMAC pin name	MII usage	RMII usage	RGMII usage (not supported by i.MX 6SoloLite)
In	FEC_TX_CLK	Transmit Clock	Not Used	Synchronous clock reference (REF_CLK, can connect from PHY)
In	FEC_RX_ER	Receive Error	Receive Error	Not Used
In	FEC_RX_CLK	Receive Clock	Not Used	Synchronous clock reference (REF_CLK, can connect from PHY)
In	FEC_RX_DV	Receive Data Valid	Receive Data Valid and generate CRS	RXDV XOR RXERR on the falling edge of FEC_RX_CLK.
In	FEC_RXD[0]	Data in, bit 0	Data in, bit 0	Data in, bit 0
In	FEC_RXD[1]	Data in, bit 1	Data in, bit 1	Data in, bit 1
In	FEC_RXD[2]	Data in, bit 2	Not Used	Data in, bit 2
In	FEC_RXD[3]	Data in, bit 3	Not Used	Data in, bit 3

Because i.MX 6 has more functionality than it has physical I/O pins, it uses I/O pin multiplexing.

Every module requires specific pad settings. For each pad, there are up to eight muxing options called ALT modes. For further explanation, see IOMUX chapter in the SoC Application Processor Reference Manual.

NOTE

Designs with an external Ethernet PHY may require an external pin configured as a simple GPIO to reset the Ethernet PHY before enabling physical clock. Otherwise, some PHYs fail to work correctly.

18.1.2 Ethernet configuration

This section describes the Ethernet driver bring up issues. For more information about Ethernet MAC configuration and using flow control in full duplex and more, check the ethernet chapter in the SoC Applications Processor References Manual.

Note the following during Ethernet driver bring up:

- Configure all I/O pins used by MAC correctly in dts files.
- Check physical input clock and power, physical led1 and led2 lighten on if clock and power input are ok.
- Make sure that MAC tx_clk has the right clock input. Otherwise, MAC cannot work.
- Make sure that the MAC address is set and valid.

By default, the Ethernet driver gets the MAC address from the Ethernet node property "local-mac-address" in dts file. If dts does not have the property, the driver get the MAC address from fuse. If the fuse does not burn the MAC address, the driver gets the MAC address from the Ethernet registers set by the bootloader. If no legal MAC address exists, MAC malfunctions. In this example, add the MAC address in the U-Boot command line for kernel, such as "fec.macaddr=0x00,0x01,0x02,0x03,0x04,0x05" in bootargs.

The Ethernet driver and hardware are designed to comply with the IEEE standards for Ethernet auto-negotiation.

Chapter 19
Porting USB

19.1 Introduction

The USB supports USB 2.0 on i.MX 6 and i.MX 7 families using the Chip IDEA hardware. On all i.MX 8 families the USB supports USB 2.0 and USB 3.0. This chapter will explain how to configure USB.

The number of USB ports vary on differnt boards but are listed below.

- 4 USB ports supporting USB 2.0 on 6Dual/6Quad/6Solo/6DualLite/6UltraLite/7Dual.
- 3 USB ports supporting USB 2.0 on 6SoloLit/6SLL/6SoloX.
- 2 USB ports supporting USB 2.0 and 3.0 on 8M Quad/8M Mini/8QuadMax/8QuadXPlus

There are up to four USB ports on i.MX 6Dual/6Quad/6Solo/6DualLite/6UltraLite/7Dual serial application processors:

- USB OTG port
- USB H1 port
- USB HSIC1 port
- USB HSIC2 port

There are three USB ports on i.MX 8QuadMax:

- USB OTG port
- USB HSIC port
- USB 3.0 port

NOTE

There is no HSIC2 port on i.MX 6SoloLite.

The following power supplies must be provided:

- 5 V power supply for USB OTG VBUS
- 5 V power supply for USB H1 VBUS
- 3.3 V power supply for HSIC1/2 port
- 3.15 V +/- 5% power supply for USB OTG/H1 PHY. Because this power can be routed from USB OTG/H1 VBUS, it indicates that if either of the power supplies is powered up, the USB PHY is powered as well. However, if neither can be powered up, an external power supply is needed.

For the USB OTG port, the following signals are used:

- USB_OTG_CHD_B
- USB_OTG_VBUS
- USB_OTG_DN
- USB_OTG_DP
- USBOTG_ID
- USBOTG_OC_B
- One pin is used to control the USB_OTG_VBUS signal.

The following signals, needed to set with proper IOMUX, are multiplexed with other pins.

NOTE

For the USBOTG_ID pin, a pin that has an alternate USBOTG_ID function must be used.

- USBOTG_ID
- USBOTG_OC_B
- One pin used to control the USB_OTG_VBUS signal.

For USB H1 port, the following signals are used:

- USB_H1_VBUS
- USB_H1_DN
- USB_H1_DP
- USBH_OC_B

The following signals are multiplexed with other pins, and need to set with proper IOMUX:

- USBH_OC_B

For USB HSIC 1/2 port, the following signals are used:

- H2_STROBE
- H3_STROBE
- H2_DATA
- H3_DATA

The following signals are multiplexed with other pins, and need to be set with proper IOMUX:

- H2_STROBE
- H3_STROBE
- H2_DATA
- H3_DATA

To secure HSIC connection, the USB HSIC port must be powered up before the USB HSIC device.

For i. MX 6SoloLite, there is only one HSIC port, so only H2_xxx signals are used.

19.2 USB overview for i.MX 6SoloLite/6SLL/6SoloX

There are up to three USB ports on i.MX 6 SoloLite/6SLL/6SoloX serial application processors:

- USB OTG1 port
- USB OTG2 port
- USB HSIC1 port

The following power supplies must be provided:

- 5V power supply for USB OTG1 VBUS
- 5V power supply for USB OTG2 VBUS
- 3.3V power supply for HSIC1 port
- 3.15 +/- 5%V power supply for USB OTG1/OTG2 PHY. Since this power can be routed from USB OTG1/OTG2 VBUS, it indicates that if either of the power supplies is powered up, the USB PHY is powered as well. However, if neither can be powered up, an external power supply is needed.

For the USB OTG1 port, the following signals are used:

- USB_OTG1_CHD_B

- USB_OTG1_VBUS
- USB_OTG1_DN
- USB_OTG1_DP
- USBOTG1_ID
- USBOTG1_OC_B
- One pin is used to control the USB_OTG1_VBUS signal.

The following signals, needed to set with proper IOMUX, are multiplexed with other pins.

NOTE

For the USBOTG_ID pin, a pin that has an alternate USBOTG_ID function must be used.

- USBOTG_ID
- USBOTG_OC_B
- One pin used to control the USB_OTG_VBUS signal.

For USB OTG2 port, the following signals are used:

- USB_OTG2_VBUS
- USB_OTG2_DN
- USB_OTG2_DP
- USBOTG2_OC_B

The following signals are multiplexed with other pins, and need to set with proper IOMUX:

- USBOTG2_OC_B

For USB HSIC 1 port, the following signals are used:

- H2_STROBE
- H2_DATA

The following signals are multiplexed with other pins, and need to set with proper IOMUX:

- H2_STROBE
- H2_DATA

To secure HSIC connection, the USB HSIC port must be powered up before the USB HSIC device.

19.3 USB overview for i.MX 8

There are two identical USB 3.0 ports on i.MX 8. Each USB 3.0 port supports both host mode and device mode with USB 2.0 and USB 3.0 device/host.

The USB PHY power supply must be configured as the following.

Take the first port (USB1) as an example, the 3.3 V power supply must be provided for:

- USB1_VDD33
- USB1_VPH

The 0.9 V power supply must be provided for:

- USB1_VPTX
- USB1_VP
- USB1_DVDD

The following signals are used:

- USB1_DN
- USB1_DP
- USB2_ID
- USB1_RESREF
- USB1_RX_N
- USB1_RX_P
- USB1_TX_N
- USB1_TX_P
- USB1_VBUS

Chapter 20
Revision History

20.1 Revision History

This table provides the revision history.

Table 9. Revision history

Revision number	Date	Substantive changes
L4.9.51_imx8qxp-alpha	11/2017	Initial release
L4.9.51_imx8qm-beta1	12/2017	Added i.MX 8QuadMax
L4.9.51_imx8mq-beta	12/2017	Added i.MX 8M Quad
L4.9.51_8qm-beta2/8qxp-beta	02/2018	Added i.MX 8QuadMax Beta2 and i.MX 8QuadXPlus Beta
L4.9.51_imx8mq-ga	03/2018	Added i.MX 8M Quad GA
L4.9.88_2.0.0-ga	05/2018	i.MX 7ULP and i.MX 8M Quad GA release
L4.9.88_2.1.0_8mm-alpha	06/2018	i.MX 8M Mini Alpha release
L4.9.88_2.2.0_8qxp-beta2	07/2018	i.MX 8QuadXPlus Beta2 release
L4.9.123_2.3.0_8mm	09/2018	i.MX 8M Mini GA release
L4.14.62_1.0.0_beta	11/2018	i.MX 4.14 Kernel Upgrade, Yocto Project Sumo upgrade
L4.14.78_1.0.0_ga	01/2019	i.MX6, i.MX7, i.MX8 family GA release
L4.14.98_2.0.0_ga	04/2019	i.MX 4.14 Kernel upgrade and board updates
L4.14.98_2.1.0	07/2019	i.MX 8M Nano Alpha release
L4.14.98_2.2.0	11/2019	i.MX 8M Nano GA release
L4.14.98_2.3.0	01/2020	i.MX 8M Nano and 8QuadXPlus C0 post GA release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 January 2020

Document identifier: IMXBSPPG

arm