
i.MX 6Dual/6Quad BSP Porting Guide

Document Number: IMX6DQBSPPG
Rev. 12.04.01, 05/2012





Contents

Section Number	Title	Page
----------------	-------	------

Chapter 1

Porting U-Boot from an i.MX 6Dual/Quad Reference Board to an i.MX 6Dual/Quad Custom Board

1.1	Overview.....	7
1.2	Obtaining the Source Code for the U-Boot.....	7
1.3	Preparing the Code.....	7
1.4	Customizing the i.MX6 Custom Board Code.....	9
1.4.1	Changing the DCD Table for i.MX6 DDR3 Initialization.....	10
1.4.2	Booting with the Modified U-Boot	10
1.4.3	Add new driver initialize code to board files.....	11
1.4.4	Further Customization at System Boot.....	11
1.4.5	Customizing the Printed Board Name.....	12
1.5	How to debug.....	12
1.5.1	Use RealView ICE for debugging.....	13
1.5.2	Use printf for debugging.....	13

Chapter 2

Configuring the IOMUX Controller

2.1	Overview.....	15
2.2	Information for Setting IOMUX Controller Registers.....	15
2.3	Setting Up the IOMUX Controller and U-Boot.....	16
2.3.1	Defining the Pads.....	16
2.3.2	Configuring IOMUX Pins for Initialization Function.....	17
2.3.3	Example-Setting a GPIO.....	17
2.4	Setting Up the IOMUX Controller in Linux.....	18
2.4.1	IOMUX Configuration Definition.....	18
2.4.2	Machine Layer File.....	19
2.4.3	Example -Setting a GPIO.....	20

Section Number	Title	Page
<p style="text-align: center;">Chapter 3 Registering a New UART Driver</p>		
3.1	Overview.....	21
3.2	Configuring UART Pads on IOMUX.....	21
3.3	Enabling UART on Kernel Menuconfig.....	22
3.4	Testing the UART.....	22
3.5	File Names and Locations.....	23
<p style="text-align: center;">Chapter 4 Adding Support for SDHC</p>		
4.1	Overview.....	25
<p style="text-align: center;">Chapter 5 Connecting an LVDS Panel to an i.MX6 Reference Board</p>		
5.1	Overview.....	27
5.2	Connecting an LVDS Panel to the i.MX6 Reference Board.....	27
5.3	Enabling a LVDS Channel.....	28
5.3.1	Locating Menu Configuration Options	28
5.4	LDB Ports.....	29
5.4.1	Input Parallel Display Ports.....	30
5.4.2	Output LVDS Ports.....	30
5.5	Further Reading.....	30
<p style="text-align: center;">Chapter 6 Supporting the i.MX6Q Camera Sensor Interface CSI0</p>		
6.1	Overview.....	33
6.2	Required Software	33
6.3	i.MX6Q CSI Interfaces Layout.....	34
6.4	Configuring the CSI Unit in Test Mode.....	34
6.5	Adding Support for a New CMOS Camera Sensor.....	35
6.5.1	Adding a Camera Sensor Entry on the Itib Catalog (Kconfig).....	35
6.5.2	Creating the Camera Sensor File.....	36
6.5.3	Adding a Compilation Flag for the New Camera.....	38

Section Number	Title	Page
6.6	Using the I2C Interface.....	39
6.7	Loading and Testing the Camera Module.....	41
6.8	Additional Reference Information.....	42
6.8.1	CMOS Interfaces Supported by the i.MX6Q.....	42
6.8.2	i.MX6Q CSI Parallel Interface.....	44
6.8.3	Timing Data Mode Protocols.....	46

Chapter 7 Porting Audio Codecs to a Custom Board

7.1	Overview.....	47
7.2	Common Porting Task.....	47
7.3	Porting the Reference BSP to a Custom Board (audio codec is the same as in the reference design).....	48
7.4	Porting the Reference BSP to a Custom Board (audio codec is different than the reference design).....	49

Chapter 8 Porting the Fast Ethernet Controller Driver

8.1	Overview.....	51
8.2	Pin Configuration.....	51
8.3	Source Code.....	53
8.4	Ethernet Configuration.....	53

Chapter 9 Porting USB Host1 and USB OTG

9.1	Overview.....	55
-----	---------------	----



Chapter 1

Porting U-Boot from an i.MX 6Dual/Quad Reference Board to an i.MX 6Dual/Quad Custom Board

1.1 Overview

This chapter provides a step-by-step guide that explains how to add i.mx6 custom board support to U-Boot. This developer's guide is based on U-Boot version rel_imx_v2009.08_12.05.01, which is present as a package on the LTIB-based Linux BSP at <http://opensource.freescale.com/git?p=imx/uboot-imx.git>.

1.2 Obtaining the Source Code for the U-Boot

The following steps explain how to obtain the source code.

1. Install LTIB as usual. Make sure you **deselect** U-Boot from compilation.
2. Manually unpack u-boot: `./ltib -m prep -p u-boot`.

The U-Boot code is now located at `rpm/BUILD/u-boot-<version number>`. The guide will now refer to the U-Boot main directory as `<UBOOT_DIR>` and assumes that your shell working directory is `<UBOOT_DIR>`.

1.3 Preparing the Code

The following steps explain how to prepare the code.

1. Make a copy of the board directory, using the following instruction:

```
$cp -R board/freescale/mx6_<reference board name> board/freescale/mx6_<custom board name>
```

2. Copy the existing `mx6_<reference board name>.h` board configuration file as `mx6_<custom board name>.h`, using the following instruction.

```
$cp include/configs/mx6_<reference board name>.h include/configs/mx6_<custom board name>.h
```

NOTE

There're some configs need to pay attention to for a new board.

- *CONFIG_LOADADDR*: Normally your uImage will be loaded to this address for boot.
- *CONFIG_SYS_MALLOC_LEN*: Heap memory size.
- *CONFIG_STACKSIZE*: Stack size.
- *CONFIG_NR_DRAM_BANKS*: Number of ddr banks.
- *PHYS_SDRAM_X*, *PHYS_SDRAM_X_SIZE*: DDR bank X start address and size.
- *CONFIG_NR_DRAM_BANKS*, *PHYS_SDRAM_X* and *PHYS_SDRAM_X_SIZE* will be passed to kernel. If these configs are wrong, kernel might be fail to boot.
- Config file is important for u-boot. Most times, It decides size, functionality and performance of u-boot.bin.

3. Create one entry in <UBOOT_DIR>/Makefile for the new i.MX 6Dual/Quad-based configuration. This file is in alphabetical order. The instruction to use is as follows:

```
mx6_<custom board name>_config      : unconfig
    @$ (MKCONFIG) $ (@:_config=) arm arm_cortexa8 mx6_<custom board name> freescale
mx6
```

NOTE

U-Boot project developers recommend adding any new board to the MAKEALL script and to run this script in order to validate that the new code has not broken any other's platform build. This is a requirement if you plan to submit a patch back to the U-Boot community. For further information, consult the U-Boot README file.

4. Rename

```
board/freescale/mx6_<custom board name>/mx6_<reference board name>.c
as
```

```
board/freescale/mx6_<custom board name>/mx6_<custom board name>.c.
```

5. Adapt any fixed paths. In this case, the linker script board/freescale/mx6_<custom board name>/u-boot.lds has at least two paths that must be changed
 - Change


```
board/freescale/mx6_<reference board name>/flash_header.o
```

to

```
board/freescale/mx6_<custom board name>/flash_header.o
```

- **Change**

```
board/freescale/mx6_<reference board name>/libmx6_<reference board name>.a
```

to

```
board/freescale/mx6_<custom board name>/libmx6_<custom board name>.a
```

6. Change the line

```
COBJS := mx6_<reference board name>.o (inside board/freescale/mx6_<custom board name>/Makefile)
```

to

```
COBJS := mx6_<custom board name>.o
```

NOTE

The remaining instructions build the U-Boot manually and do not use LTIB.

7. Create a shell script under <UBOOT_DIR> named build_u-boot.sh.

The file's contents are now:

```
#!/bin/bash
export ARCH=arm
export CROSS_COMPILE=<path to cross compiler/prefix> (e.g.
PATH:/opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-gnueabi/bin/arm-
none-linux-gnueabi-
export PATH=$PATH:<path to compiler>
make mx6_<custom board name>_config
make
```

8. Compile U-Boot using \$./build_u-boot.sh

9. If everything is correct, you should now have u-boot.bin as proof that your build setup is correct and ready to be customized.

The new i.MX6 custom board that you have created is an exact copy of the i.MX6 reference board, but the boards are two independent builds. This allows you to proceed to the next step: customizing the code to suit the new hardware design.

1.4 Customizing the i.MX6 Custom Board Code

The new i.MX6 custom board is part of the U-Boot source tree, but it is a duplicate of the i.MX6 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 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.

1.4.1 Changing the DCD Table for i.MX6 DDR3 Initialization

Initializing the memory interface requires configuring the relevant I/O pins with the right mode and impedance and initializing the MMDC module.

1. To port to the custom board, the appropriate DDR initialization needs to be used. This is the same initialization as would be used in a JTAG initialization script.
2. Open the file

```
board/freescale/mx6_<custom board name>/flash_header.S
```

3. Modify all MXC_DCD_ITEM macros to match the memory specifications. These codes will be read by rom code to initialize your DDR memory.
4. Modify dcd_hdr and write_dcd_cmd value.

NOTE

If you change the number of MXC_DCD_ITEM lines in the DCD table, you must update the value of the dcd_hdr and write_dcd_cmd labels according to the number of items.

- dcd_hdr is comprised of tag(0xD2), len and version(0x40), where $\text{len} = \text{<dcd items>} * 8 + 8$.

e.g.

```
len = 128, dcd_hdr = 0x400804D2 (Tag=0xD2, Len=128*8 + 4 + 4, Ver=0x40)
```

- write_dcd_cmd is comprised of tag(0xCC), len and param(0x04), where $\text{len} = \text{<dcd items>} * 8 + 4$.

e.g.

```
len = 128, write_dcd_cmd = 0x040404CC (Tag=0xCC, Len=128*8 + 4, Param=0x04)
```

1.4.2 Booting with the Modified U-Boot

If the DCD table (board/freescale/mx6_<custom board name>/flash_header.S) was modified successfully, you can compile and write u-boot.bin to an SD card. To test this, insert the SD card into the SD card socket of the CPU board and power cycle the board.

A message like this should be printed in the console:

U-Boot 2009.08-00410-ga32bc11 (Dec 15 2011 - 13:19:05)

```
CPU: Freescale i.MX 6 family 0.0V at 792 MHz
mx6 pll1: 792MHz
mx6 pll2: 528MHz
mx6 pll3: 480MHz
mx6 pll8: 50MHz
ipg clock      : 66000000Hz
ipg per clock  : 66000000Hz
uart clock     : 80000000Hz
cspi clock     : 60000000Hz
ahb clock      : 132000000Hz
axi clock      : 264000000Hz
emi_slow clock: 29333333Hz
ddr clock      : 528000000Hz
usdhc1 clock   : 200000000Hz
usdhc2 clock   : 200000000Hz
usdhc3 clock   : 200000000Hz
usdhc4 clock   : 200000000Hz
nfc clock      : 24000000Hz
Board: MX6-<reference board name>:[ POR ]
Boot Device: SD
I2C: ready
DRAM: 2 GB
MMC: FSL_USDHC: 0,FSL_USDHC: 1,FSL_USDHC: 2,FSL_USDHC: 3
In: serial
Out: serial
Err: serial
Net: got MAC address from IIM: 00:00:00:00:00:00
FEC0 [PRIME]
Hit any key to stop autoboot: 0
<reference board name>: U-Boot >
```

1.4.3 Add new driver initialize code to board files

The following steps explain how to add new driver initialize code.

1. Find and edit `mx6_<customer_board>.c` in `board/freescale/mx6_<customer_board>/.c`.
2. Edit `mx6_<customer_board>.c` and add new module driver's initialization code, including clock, iomux, gpio. .
3. Put driver init function to `board_init` or `board_late_init`.

NOTE

- `board_init` will be called very early, before uart init, thus, don't print characters, or u-boot will crash. Most driver init function are put in `board_init`.
- `board_late_init` will be called very late. Some times for debugging init code, we can put driver init function in it.

1.4.4 Further Customization at System Boot

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

```
start_armboot in "lib_arm/board.c".
board_init()
```

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

The first board dependent function inside **init_sequence[]** array is **board_init()**. **board_init()** is implemented inside board/freescale/mx6_<custom board name>.c.

At this point the most important tip is the following line of code:

```
...
gd->bd->bi_arch_number = MACH_TYPE_MX6_<reference board name>; /* board id for Linux */
...
```

To customize your board ID, go to the registration process at <http://www.arm.linux.org.uk/developer/machines/>

This tutorial will continue to use MACH_TYPE_MX6_<reference board name>.

1.4.5 Customizing the Printed Board Name

To customize the printed board name, use the **checkboard()** function. This function is called from the **init_sequence[]** 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 the call to **identify_board_id()** inside **checkboard()** and replace `printf("Board: ");` with `printf("Board: i.MX6 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 function **__print_board_info()** according to the identification method on the custom board.

1.5 How to debug

Normally we have two ways for debugging:

- Use Realview ICE.
- Use printf.

1.5.1 Use RealView ICE for debugging

Normally we use RealView ICE to debug in very early stage, e.g. before uart initialization, or cases that is hard to debug with printf.

1. Make sure your RealView ICE can support cortex A9. If not, you need to upgrade the firmware and your RealView software.
2. Load u-boot (which is an elf file) in root directory of u-boot fully or just symbol (faster) to debug step by step.

NOTE

We can make optimization level to 0 in rules.mk which will be easier for debugging in RealView ICE.

1.5.2 Use printf for debugging

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

NOTE

If we want to use printf in early stages, e.g. in board_init, we can put uart initialization code earlier, e.g. to start_armboot() in board.c of lib_arm directory.

Chapter 2

Configuring the IOMUX Controller

2.1 Overview

Before using the i.MX 6Dual/Quad pins (or pads), users must select the desired function and correct values for characteristics such as voltage level, drive strength, and hysteresis. They do this by configuring a set of registers from the IOMUX controller.

For detailed information about each pin, see the "External Signals and Pin Multiplexing" chapter in the *i.i.MX 6Dual/Quad Applications Processor Reference Manual*. For additional information about the IOMUX controller block, see the "IOMUX Controller (IOMUXC)" chapter in the *i.i.MX 6Dual/Quad Applications Processor Reference Manual*.

2.2 Information for Setting IOMUX Controller Registers

The IOMUX controller contains four sets of registers that affect the i.MX 6Dual/Quad registers, as follows:

- General-purpose registers (IOMUXC_GPRx)-consist of three 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 8 different functions (also called ALT modes) is used.
 - Can set pad's 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; select 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 help 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.

2.3 Setting Up the IOMUX Controller and U-Boot

The IOMUX controller contains four sets of registers that affect the i.MX 6Dual/Quad registers, as follows:

Table 2-1. Configuration Files

Path	Filename	Description
cpu/arm_cortexa8/mx6/	iomux-v3.c	IOMUX functions (no need to change)
include/asm-arm/arch-mx6/	iomux-v3.h	IOMUX definitions (no need to change)
include/asm-arm/arch-mx6/	mx6_pins.h	Definition of all processor's pads
board/freescale/mx6_<reference board name>/	mx6_<reference board name>.c	Board initialization file

2.3.1 Defining the Pads

The iomux-mx6q.h file contains each pad's IOMUX definitions. Use the following code to see the default definitions:

```
#define MX6Q_PAD_<PIN NAME>__<FUNC NAME> (_MX6Q_PAD_<PIN NAME>__<FUNC NAME> |  
MUX_PAD_CTRL(MX6Q_<PAD_CTRL>))
```


To change the values for each pad according to your hardware configuration, use the following:

```
_MX6Q_PAD_<PIN NAME>_<FUNC NAME> = IOMUX_PAD(_pad_ctrl_ofs, _mux_ctrl_ofs, _mux_mode,
_sel_input_ofs, _sel_input, _pad_ctrl)
```

Where:

- `_pad_ctrl_ofs` - PAD Control Offset
- `_mux_ctrl_ofs` - MUX Control Offset
- `_mux_mode` - MUX Mode
- `_sel_input_ofs` - Select Input Offset
- `_sel_input` - Select Input
- `_pad_ctrl` - PAD Control

```
MX6Q_<PAD CTRL> = (pull_keep_en | pull_keep_sel | pull_config | speed | drive_strength |
slew_rate | hyst_en | open_drain_en )
```

2.3.2 Configuring IOMUX Pins for Initialization Function

The `board-mx6q_<reference board name>.c` file contains the initialization functions for all peripherals (such as UART, I²C, and Ethernet). Configure the relevant pins for each initializing function, using the following:

```
mxc_request_iomux(<pin name>, <iomux config>);
mxc_iomux_set_input(<mux input select>, <mux input config>);
mxc_iomux_set_pad(<pin name>, <iomux pad config>);
```

Where the following applies:

`<pin name>` See all pins definitions on file `iomux-mx6q.h`

`<iomux config>` See parameters defined at `iomux_config` enumeration on file `iomux.h`

`<iomux input select>` See parameters defined at `iomux_input_select` enumeration on file `iomux.h`

`<iomux input config>` See parameters defined at `iomux_input_config` enumeration on file `iomux.h`

`<iomux pad config>` See parameters defined at `iomux_pad_config` enumeration on file `iomux.h`

2.3.3 Example-Setting a GPIO

For an example, configure and use pin `SD2_DAT1` as a general GPIO and toggle its signal.

Add the following code to the file `board-mx6q_<reference board name>.c`, in the array of `mx6_<reference board name>_pads`:

```
MX6Q_PAD_SD2_DAT1__GPIO1_14;
```

Make sure, no any other `SD2_DAT1` configuration exists in the array. Then if done correctly, the pin `SD2_DAT1` on the i.MX 6Dual/Quad toggles when booting.

2.4 Setting Up the IOMUX Controller in Linux

The folder `linux/arch/arm/mach-<platform name>` contains the specific machine layer file for your custom board.

For example, the machine layer file used on the i.MX 6Dual/Quad <reference> boards are `linux/arch/arm/mach-mx6/board-mx6q_<reference board name>.c`. This platform is used in the examples in this section. The machine layer files include the IOMUX configuration information for peripherals used on a specific board.

To set up the IOMUX controller and configure the pads, change the two files described in table below:

Table 2-2. IOMUX Configuration Files

Path	File name	Description
<code>linux/arch/arm/plat-mxc/include/mach/</code>	<code>iomux-mx6q.h</code>	IOMUX configuration definitions
<code>linux/arch/arm/mach-mx6</code>	<code>board-mx6q_<reference board name>.c</code>	Machine Layer File. Contains IOMUX configuration structures

2.4.1 IOMUX Configuration Definition

The `iomux-mx6q.h` file contains definitions for all i.MX 6Dual/Quad pins. Pin names are formed according to the formula `<SoC>_PAD_<Pad Name>_GPIO_<Instance name>_<Port name>`. Definitions are created with the following line code.

```
IOMUX_PAD(PAD Control Offset, MUX Control Offset, MUX Mode, Select Input Offset, Select Input, Pad Control)
```

The variables are defined as follows:

PAD Control Offset Address offset to pad control register
(`IOMUXC_SW_PAD_CTL_PAD_<PAD_NAME>`)

MUX Control Offset Address offset to MUX control register
(IOMUXC_SW_MUX_CTL_PAD_<PAD NAME>)

MUX Mode MUX mode data, defined on MUX control registers

Select Input Offset Address offset to MUX control register
(IOMUXC_<Instance_port>_SELECT_INPUT)

Select Input Select Input data, defined on select input registers

Pad Control Pad Control data, defined on Pad control registers

Definitions can be added or changed, as shown in the following example code:

```
#define MX6Q_PAD_SD2_DAT1__USDHC2_DAT1      IOMUX_PAD (0x0360, 0x004C, 0, 0x0000, 0, 0)
```

The variables are as follows:

0x0360 - PAD Control Offset

0x004C - MUX Control Offset

0 - MUX Mode

0x0000 - Select Input Offset

0 - Select Input

0 - Pad Control

For all addresses and register values, check the IOMUX chapter in the *i.MX 6Dual/Quad Applications Processor Reference Manual*.

2.4.2 Machine Layer File

The board-mx6q_<reference board name>.c file contains structures for configuring the pads. They are declared as follows:

```
static iomux_v3_cfg_t mx6q_<reference board name>_pads[] = {
...
...
...
MX6Q_PAD_SD2_CLK__USDHC2_CLK,
MX6Q_PAD_SD2_CMD__USDHC2_CMD,
MX6Q_PAD_SD2_DAT0__USDHC2_DAT0,
MX6Q_PAD_SD2_DAT1__USDHC2_DAT1,
MX6Q_PAD_SD2_DAT2__USDHC2_DAT1,
MX6Q_PAD_SD2_DAT3__USDHC2_DAT3,
...
...
...
};
```

Add the pad's definitions from iomux-mx6q.h to the above code.

On init function (in this example "mx6q_<reference board name>_init" function), set up the pads using the following function:

```
mx6q_iomux_v3_setup_multiple_pads(mx6q_<reference board name>_pads,  
ARRAY_SIZE(mx6q_<reference board name>_pads));
```

2.4.3 Example -Setting a GPIO

For an example, configure the pin PATA_DA_1 (PIN L3) as a general GPIO and toggle its signal.

On Kernel menuconfig, add sysfs interface support for GPIO with the following code:

```
Device Drivers --->  
    [*] GPIO Support --->  
        [*] /sys/class/gpio/... (sysfs interface)
```

Define the pad on iomux-mx6q.h file as follows:

```
#define MX6Q_PAD_SD2_DAT1__GPIO_1_14 IOMUX_PAD(0x0360, 0x004C, 5, 0x0000, 0, 0)
```

Parameters:

0x0360 - PAD Control Offset

0x004C - MUX Control Offset

5 - MUX Mode

0x0000 - Select Input Offset

0 - Select Input

0 - Pad Control

To register the pad, add the previously defined pin to the pad description structure in the board-mx6q_<reference board name>.c file, as shown in the following code.

```
static iomux_v3_cfg_t mx6q_<reference board name>_pads[] = {  
...  
...  
...  
MX6Q_PAD_NANDF_CS0__GPIO_6_11,  
...  
...  
...  
};
```

Chapter 3

Registering a New UART Driver

3.1 Overview

Since Linux already has a UART driver for the i.MX6Q configure the UART pads on the IOMUX registers.

This chapter explains how to configure the UART pads, enable the UART driver, and test that the UART was set up correctly.

3.2 Configuring UART Pads on IOMUX

The IOMUX register must be set up correctly before the UART function can be used. This section provides example code to show how to set up the IOMUX register.

Pads are configured using the file `linux/arch/arm/mach-mx6/<platform>.c`, with `<platform>` replaced by the appropriate platform file name. For example, the machine layer file used on the i.MX6Q reference boards are `linux/arch/arm/mach-mx6/board-mx6q_arm2.c`.

The `iomux-mx6q.h` file contains the definitions for all i.MX6Q pads. Configure the UART pads as follows:

```
/* UART4 */
#define MX6Q_PAD_KEY_COLO_UART4_TXD
    \(_MX6Q_PAD_KEY_COLO__UART4_TXD | MUX_PAD_CTRL(MX6Q_UART_PAD_CTRL)

#define MX6Q_PAD_KEY_ROW0_UART4_TXD
    \(_MX6Q_PAD_KEY_ROW0__UART4_TXD | MUX_PAD_CTRL(MX6Q_UART_PAD_CTRL)
```

The structures for configuring the pads are contained in the `mx6q_<reference board name>.c` file. Update them so that they match the configured pads' definition as shown above. The code below shows the non-updated structures:

```
static iomux_v3_cfg_t mx6q_arm2_pads[] = {
...
...
...
}
```

Enabling UART on Kernel Menuconfig

```
MX6Q_PAD_KEY_COL0_UART4_TXD,  
MX6Q_PAD_KEY_ROW0_UART4_TXD,  
...  
...  
...  
};
```

Use the following function to set up the pads on the init function `mx6_arm2_init()` (found in the `board-mx6q_arm2.c` file).

3.3 Enabling UART on Kernel Menuconfig

Enable the UART driver on Linux menuconfig. This option is located at:

```
-> Device Drivers  
    -> Character devices  
        -> Serial drivers  
            <*> IMX serial port support  
            [*] Sonsole on IMX serial port
```

After enabling the UART driver, build the Linux kernel and boot the board.

3.4 Testing the UART

By default, the UART is configured as follows:

- Baud Rate: 9600
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None

If the user employed a different UART configuration for a device that needs to connect to the i.MX6Q processor, connection and communication will fail. There is a simple way to test whether the UART is properly configured and enabled.

On the i.MX6Q Linux command line, type the following:

```
echo "test" > /dev/ttyMXC1
```

UART4 (on the i.MX6Q expansion board) sends the string "test".

3.5 File Names and Locations

There are three Linux source code directories that contain relevant UART files.

Table below lists the UART files that are available on the directory <linux source code directory>/drivers/tty/serial/

Table 3-1. Available Files-First Set

File	Description
imx.c	uart driver

Table below lists the UART files that are available on the directory <linux source code directory>/arch/arm/plat-mxc/include/mach/

Table 3-2. Available Files-Second Set

File	Description
mxc_uart.h	UART header containing UART configuration and data structures
iomux-<platform>.h	IOMUX pads definitions

Chapter 4

Adding Support for SDHC

4.1 Overview

Signal Overview

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

- 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 MMC card. This should be supported by 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.

Pin IOMUX

Make modification to IOMUX according to your platform in the following file:

- arch/arm/plat-mxc/include/mach/iomux-mx6q.h

Support of SD3.0

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

Support of SDIO

In most cases, SDIO requires more power than SD/MMC memory cards. Just make sure the power supply on SD slot while using SDIO, or please apply an external power to SDIO instead.

Chapter 5

Connecting an LVDS Panel to an i.MX6 Reference Board

5.1 Overview

This chapter explains how to connect the LVDS panel to an i.MX6 reference board. The i.MX6 processor has a LVDS display bridge (LDB) block that drives LVDS panels without external bridges. The LDB supports the flow of synchronous RGB data from the IPU to external display devices through the LVDS interface. This 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.

5.2 Connecting an LVDS Panel to the i.MX6 Reference Board

The following LVDS panels were tested on the i.MX6 reference boards:

- HannStar display (model number: HSD100PXN1)

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

```
LVDS0 and LVDS1 on the board: video=mxcfb0:dev=ldb,LDB-XGA,if=RGB24 video=mxcfb1:dev=ldb,LDB-XGA,if=RGB24 ldb=sep0
```

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

```
LVDS0 and LVDS1 on the board: video=mxcfb0:dev=ldb,LDB-XGA,if=RGB666 video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666 ldb=sep0
```

5.3 Enabling a LVDS Channel

The LDB driver source code is available at <ltib_dir>/rpm/BUILD/linux/drivers/video/mxc/ldb.c.

To make a built-in LDB driver functional, add the 'ldb' option to the kernel command line.

The driver configures the LDB when the device is probed.

When the LDB device is probed properly, the driver uses platform data information to configure the LDB's reference resistor mode and regulator. The LDB driver probe function 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 boot up LDB option chosen by the user. If the user has not specified an option but the video mode can be found in the local video mode database, the driver chooses an appropriate LDB setting. If no video mode is matched, nothing is done in probe function. Users can set up the LDB later by using ioctrls. The LDB will be fully enabled in probe function if the driver finds that the primary display device is a single display device with an LVDS interface.

The steps the driver takes to enable a LVDS channel are as follows:

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

5.3.1 Locating Menu Configuration Options

Linux kernel configuration options are provided for the build-in status to enable this module. To locate these options, use the following procedure.

1. Go to <ltib dir>.
2. Use the ./ltib -c command.
3. Select **Configure the Kernel** on the screen displayed and exit.

- When the next screen appears, follow this sequence: Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer > MXC LDB

5.4 LDB Ports

Figure below shows the LDB block.

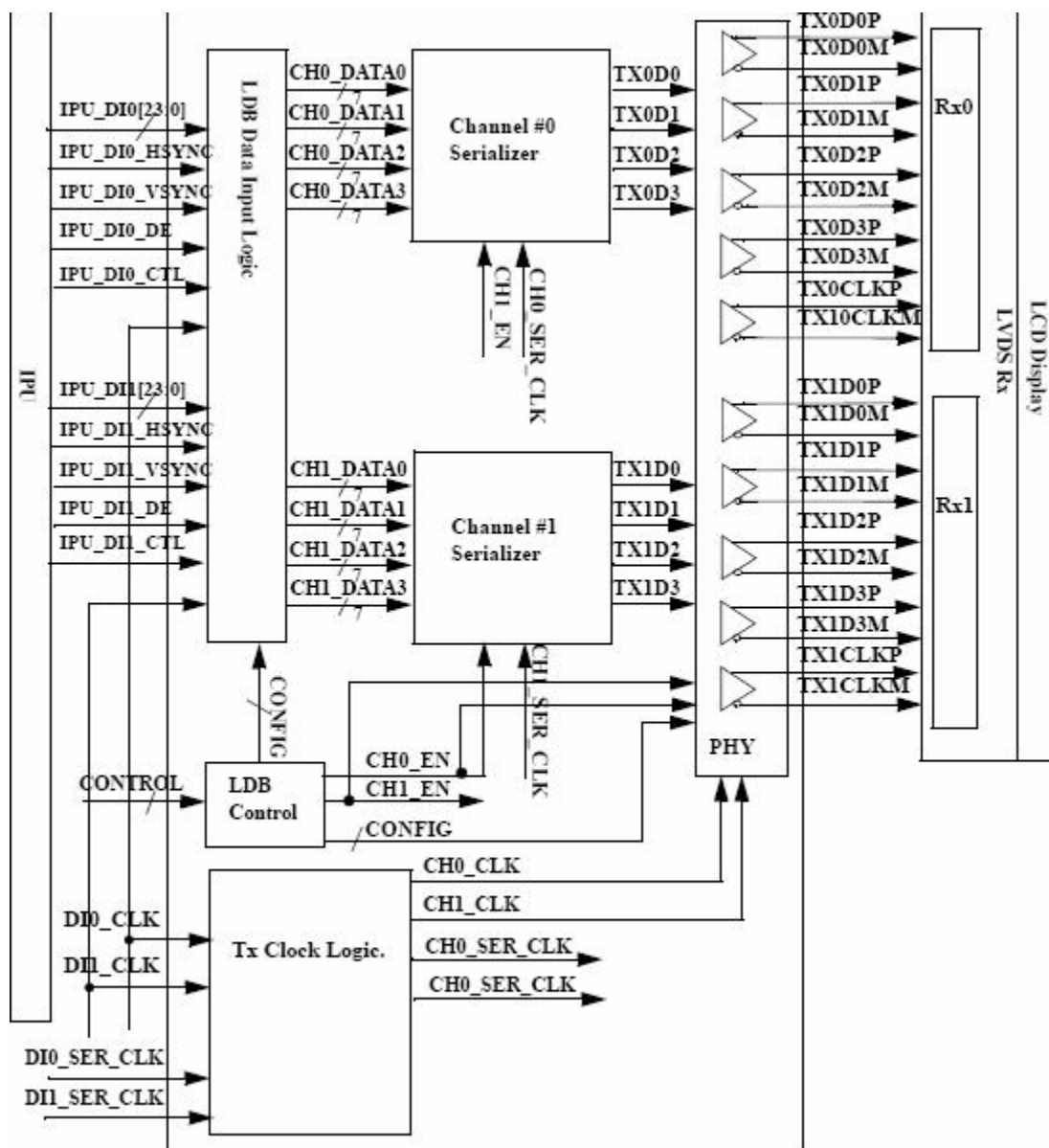


Figure 5-1. i.MX6 LVDS Display Bridge (LDB) Block

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.

5.4.1 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 (e.g. UXGA-1600 x 1200 at 60 Hz + 35% blanking)
- For single-channel output: up to 85 MHz per interface. (e.g. WXGA-1366 x 768 at 60 Hz + 35% blanking).

5.4.2 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 pair and one clock pair, with a pair meaning 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 different output channel

5.5 Further Reading

Please consult the following reference materials for additional information:

- i.MX6 Multimedia Applications Processor Reference Manual
- i.MX6 Linux Reference Manual, included as part of the Linux BSP

Chapter 6

Supporting the i.MX6Q Camera Sensor Interface CSI0

6.1 Overview

This chapter provides information about how to use the expansion connector to include support for a new camera sensor on an i.MX6Q reference board. It explains how to do the following:

- Configure the CSI unit in test mode ([Configuring the CSI Unit in Test Mode](#))
- Add support for a new CMOS sensor in the i.MX6Q BSP ([Adding Support for a New CMOS Camera Sensor](#))
- Set up and use the I²C interface to handle your camera bus ([Using the I²C Interface](#))
- Load and test the camera module ([Loading and Testing the Camera Module](#))

It also provides reference information about the following:

- Required software and hardware
- i.MX6Q reference CSI interfaces layout ([i.MX6Q CSI Interfaces Layout](#))
- CMOS sensor interfaces (CSI) supported by the i.MX6Q (IPU) ([CMOS Interfaces Supported by the i.MX6Q](#))
- i.MX6Q Sabre SD CSI parallel interface ([i.MX6Q CSI Parallel Interface](#))
- i.MX6Q CSI test mode ([Timing Data Mode Protocols](#))

6.2 Required Software

In Freescale BSPs, all capture devices are based on the V4L2 standard. Therefore, only the CMOS-dependent layer needs to be modified to include a new CMOS sensor; all other layers have been developed to work with V4L2.

Required development tools are as follows:

- Linux host with i.MX6Q Linux L3.0.15_12.04.01 or newer
- Serial port terminal (such as Hyperterminal, TeraTerm, Minicom).

6.3 i.MX6Q CSI Interfaces Layout

Figure below shows the camera interface layout on an i.MX6Q-based board.

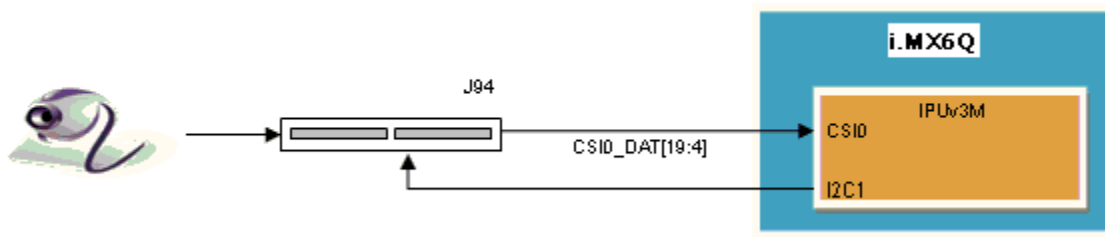


Figure 6-1. Camera Interface Layout

Only CSI0 is used for the purpose of this document. The usage of CSI1 depends on its availability on the board being used.

6.4 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. Setting the TEST_GEN_MODE register places the device in test mode, which is used for debugging. The CSI generates a frame by itself 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 table below. Call `ipu_csi_init_interface()` to configure the CSI interface protocol, formats and features.

Table 6-1. Settings for Test Mode

Bit Field	Value	Description
CSI0_DATA_DEST	0x4	Destination is IDMAC via 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

Table continues on the next page...

Table 6-1. Settings for Test Mode (continued)

Bit Field	Value	Description
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	VSYSN is directly applied to internal circuitry

6.5 Adding Support for a New CMOS Camera Sensor

To add support for a new CMOS camera sensor to your BSP, first create a device driver for supporting 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 implement the CSI configuration for the parallel protocol used between the camera and the i.MX6Q.

Perform the following three steps on the i.MX6Q BSP to create the device driver:

1. Add a camera sensor entry on the Itib catalog.
2. Create the camera file.
3. Add compilation flag for the new camera sensor.

These steps are discussed in detail in the following subsections.

6.5.1 Adding a Camera Sensor Entry on the Itib Catalog (Kconfig)

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

```
Device Drivers > Multimedia support > Video capture adapters > MXC Camera/V4L2 PRP
Features
support
```

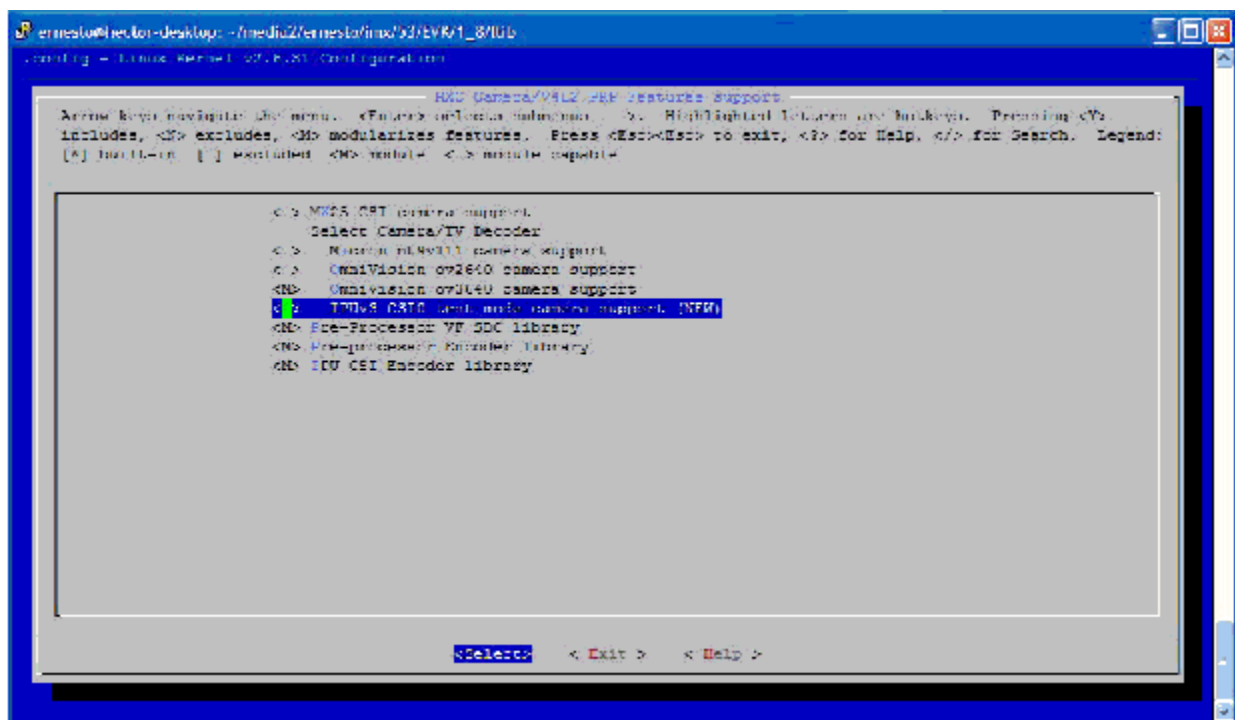


Figure 6-2. MXC Camera/V4L2 PRP Features Support Window

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

1. Enter the following into the i.MX6Q display specific folder:

```
$ cd <ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture
```

2. Open 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.
```

6.5.2 Creating the Camera Sensor File

The camera sensor file enables camera initialization, reset signal generation, power settings, CSI configuration and all sensor-specific code. Because power settings are handled by the PMIC among other voltage regulators, camera drivers must configure the PMIC during its initialization in order to set up the power voltage configuration unless this has already been done. The reset waveform and the initialization routine must also be included somewhere, usually on the probe function.

NOTE

Before connecting a camera sensor to the i.MX6Q board, you must check whether the sensor is powered with the proper supply voltages and also whether the sensor data interface has the correct VIO value. Power supply mismatches can damage either the CMOS or the i.MX6Q.

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

```
<ltib_dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture/
```

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

Table 6-2. Required Functions

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

After the functions have been created, you need to 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_prp_vf_sdc_bg.c` file so that the image converter will not perform color space conversion and the input received from the CSI test mode generator will be sent directly to the memory. Also, 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.

6.5.3 Adding a Compilation Flag for the New Camera

After camera files have been 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: `<ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture`.

1. Enter the following into the i.MX6Q camera support folder

```
$ cd <ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture
```

2. Open the i.MX6Q 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 sensor_clock.o
obj-$(CONFIG_MXC_IPUV3_CSI0_TEST_MODE) += ipuv3_csi0_chess_camera.o
```

The kernel object is created 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

6.6 Using the I²C Interface

Many camera sensor modules require a synchronous serial interface for initialization and configuration. This section uses the <ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture/ov3640.c file for 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 ov3640.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 ipuv3_csi0_chess_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;
```

Using the I2C Interface

```
    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 ov3640.c for the complete example code.

After creating the new I²C device, add the following lines to your platform file (located at <ltib dir>/rpm/BUILD/linux/arch/arm/mach-mx6/mx6q-<board name>.c).

```
static struct mxc_camera_platform_data camera_data = {
    .analog_regulator = "VSD",
    .mclk = 24000000,
    .csi = 0,
    .pwn = camera_pwn,
};

static struct i2c_board_info mxc_i2c0_board_info[] __initdata = {
    {
        .type = "ov3640",
        .addr = 0x3C,
        .platform_data = (void *)&camera_data,
    },
    {
        .type = "adv7180",          .addr = 0x21,          .platform_data = (void
*)&adv7180_data,    },
    {
        .type = "cs42888",
        .addr = 0x48,
    },
};
```



```

    },
    {
        .type = "mycamera",
        .addr = 0x2E,
        .platform_data = (void *)&camera_data,
    },
};

static void __init mxc_board_init(void)
{
    ...
    i2c_register_board_info(0, mxc_i2c0_board_info, ARRAY_SIZE(mxc_i2c0_board_info));
    ...
}

```

You may modify the platform 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. Notice that ".addr" is used to specify the I²C address of the camera sensor module.

NOTE

It is mandatory that ".type" be equal to the i2c_device_id on your camera sensor file (mycamera.c).

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);

```

6.7 Loading and Testing the Camera Module

If your camera driver has been 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@freescall /unit_tests$ modprobe ipuv3_csi0_chess_camera
root@freescall /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@freescall ~$ cd /unit_tests/
root@freescall /unit_tests$ ./mxc_v4l2_overlay.out

```

If the imx-test package has not been built, select it from the LTIB package menu:

```

Package List > imx-test

```

The chessboard appears in a rectangle located on the left top side of the WVGA panel, as shown in figure below. The colors of the chessboard toggle between red, green, and blue every time you run the test.

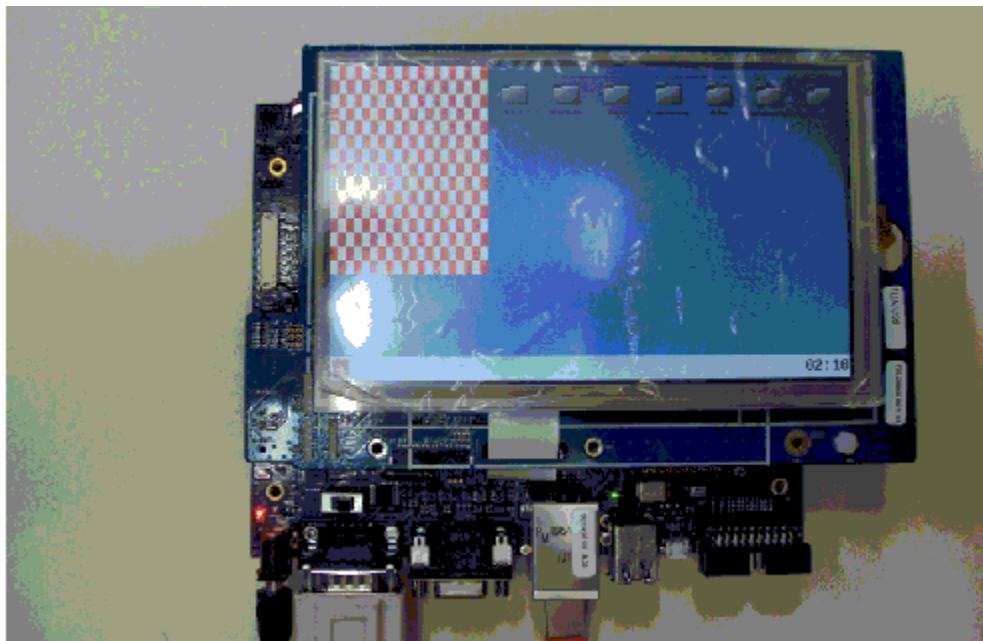


Figure 6-3. Chessboard Test

6.8 Additional Reference Information

This section provides reference information about the following:

- CMOS interfaces supported by the i.MX6Q
- i.MX6Q CSI parallel interface
- Timing data mode protocols

6.8.1 CMOS Interfaces Supported by the i.MX6Q

The camera sensor interface, which is part of the image processing unit (IPU) module on the i.MX6Q, handles CMOS sensor interfaces. The i.MX6Q 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 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 (e.g. a mechanical shutter)
 - Synchronized strobes are generated for up to 6 outputs-the sensor and camera peripherals (flash, mechanical shutter...)
- Frame rate reduction by periodic skipping of frames
- Window-of-interest selection
- Pre-flash for red-eye reduction and for measurements (e.g. focus) in low-light conditions.

For details, refer to the "Image Processing Unit (IPU)" chapter in the *i.MX6Q Applications Processor Reference Manual*. Figure below shows the block diagram.

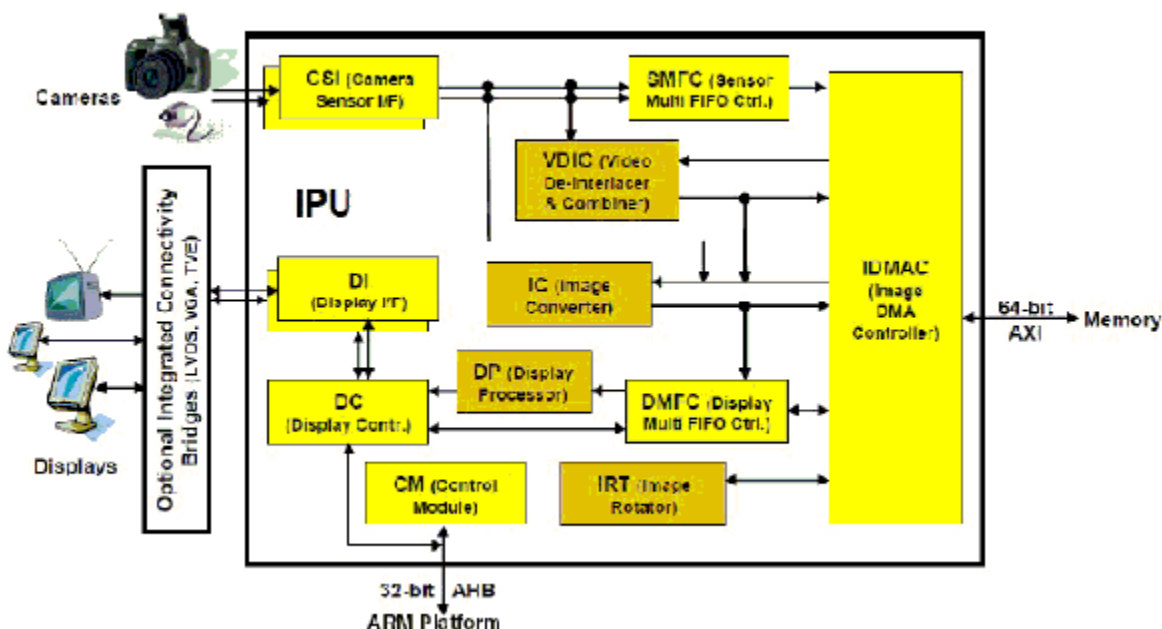


Figure 6-4. IPU Block Diagram

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, e.g. 9 Mpixels at 15 fps
 - Pixel on two cycles (BT.656): up to 67 MP/sec, e.g. 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.

6.8.2 i.MX6Q 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 (shown in figure below) provides a clock output (MCLK), which is used by the sensor as a clock input reference. The i.MX6Q requests either video or still images through a different interface between the processor and the camera

module. In most cases, the interface is a synchronous serial interface such as the I²C. 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.MX6Q. The camera sensor creates PIXCLK based on MCLK input.

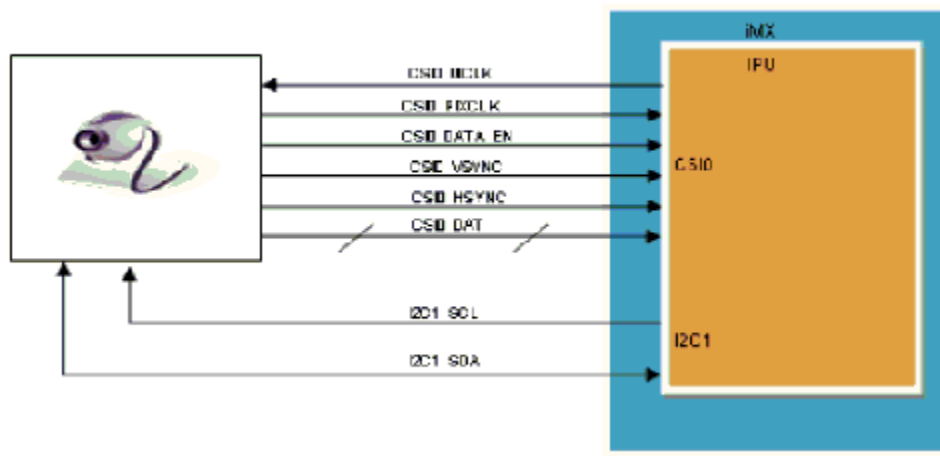


Figure 6-5. Parallel Interface Layout

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 I²C.

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 camera parallel interface provided by the i.MX6Q is a 15 line interface, as described in table below:

Table 6-3. 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
DATA_EN	CSI0_DATA_EN	Data Enable or Data ready
DATA[19:10]	CSI0_DAT [19:10]	Pixel data bus, optional to [19:4]

[Timing Data Mode Protocols](#), explains how the timing data mode protocols use these signals. Not all signals are used in each timing data mode protocol.

6.8.3 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 beginning of a frame, and HSYNC is used to indicate the beginning of a row. The sensor clock is always ticking.

In non-gated mode, VSYNC is used to indicate 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, refer to the *i.MX6Q Applications Processor Reference Manual*.

Chapter 7

Porting Audio Codecs to a Custom Board

7.1 Overview

This chapter explains how to port audio drivers from the Freescale reference BSP 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 Freescale reference design. This chapter first explains the common porting task and then various other porting tasks.

7.2 Common Porting Task

The `mxs_audio_platform_data` structure must be defined and filled appropriately for the custom board before doing any other porting tasks. An example of a filled structure can be found in the file located at `linux/arch/arm/mach-mx6/board-mx6q_sabresd.c`.

```
static struct mxs_audio_platform_data wm8962_data = {
    .ssi_num = 1,
    .src_port = 2,
    .ext_port = 3,
    .hp_gpio = SABRESD_HEADPHONE_DET,
    .hp_active_low = 1,
    .mic_gpio = SABRESD_MICROPHONE_DET,
    .mic_active_low = 1,
};
```

Customize the structure according to the following definitions:

ssi_num: The ssi used for this codec

src_port: The digital audio mux (DAM) port used for the internal SSI interface

ext_port: The digital audio mux (DAM) port used for the external device audio interface

hp_gpio: The IRQ line used for headphone detection

hp_active_low: When headphone is inserted, the detection pin status, if pin voltage level is low, the value should be 1.

mic_gpio: The IRQ line used for microphone detection

mic_active_low: When microphone is inserted, the detection pin status, if pin voltage level is low, the value should be 1.

7.3 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, users must ensure that the I/O signals and the power supplies to the codec are properly initialized in order to port the reference BSP to the custom board.

The board-mx6q_sabresd.h file contains the pads definitions. Add entries to this file to define the configuration for the audio codec signals.

The necessary signals for the wm8962 codec, which is used on the mx6 sabresd board, are as follows:

- I²C interface signals
- I²S interface signals
- SSI external clock input to wm8962

Table below shows the required power supplies for the wm8962 codec.

Table 7-1. 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

7.4 Porting the Reference BSP to a Custom Board (audio codec is different than the reference design)

When adding support for an audio codec that is different than the one on the Freescale reference design, users must create new ALSA drivers in order to port the reference BSP to a custom board. The ALSA drivers plug into the ALSA sound framework, which allows 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`. Table below shows the files used for the wm8962 codec support:

Table 7-2. Files for wm8962 Codec Support

File Name	Definition
<code>imx-pcm-dma-mx2.c</code>	<ul style="list-style-type: none"> • Shared by the stereo ALSA SoC driver, the esai driver, and the spdif driver. • Responsible for preallocating DMA buffers and managing DMA channels.
<code>imx-ssi.c</code>	<ul style="list-style-type: none"> • Register the CPU DAI driver for the stereo ALSA SoC • Configures the on-chip SSI interfaces
<code>wm8962.c</code>	<ul style="list-style-type: none"> • Register the stereo codec and Hi-Fi DAI drivers. • Responsible for all direct hardware operations on the stereo codec.
<code>imx-wm8962.c</code>	<ul style="list-style-type: none"> • Machine layer code • Create the driver device • Register the stereo sound card.

NOTE

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

Chapter 8

Porting the Fast Ethernet Controller Driver

8.1 Overview

This chapter explains how to port the fast Ethernet controller (FEC) driver to the i.MX6 processor. Using Freescale's standard (FEC) driver makes porting to the i.MX6 simple. Porting needs to address the following three areas:

- Pin configuration
- Source code
- Ethernet connection configuration

8.2 Pin Configuration

The FEC supports three different standard physical media interfaces: a reduced media independent interface (RMII), a media independent interface (MII), and a 7-wire serial interface.

In addition, the FEC includes support for different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbps MII, and 10/100 Mbps RMII. The 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 FEC chapter of the *i.MX6 Multimedia 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 table below.

Table 8-1. Pin Usage in MII and SNI Modes

Direction	EMAC Pin Name	MIU Usage	RMII Usage	RGMII Usage
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_CRD	Carrier Sense	Not Used	Not Used
In	FEC_COL	Collision	Not Used	Not Used
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

Since i.MX6 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 8 muxing options called ALT modes. For further explanation refer to IOMUX chapter in the Reference Manual.

RMII signals are assigned on gpio1 (GPIO[22-31]). By default, those signals are in mode ALT0 and need to be changed to the proper ALT mode (in general, for all RMII signals the mode should be ALT1). For further explanation about which mode should be set for each signal, refer to the IOMUX chapter in the Reference Manual.

The FEC_PHY_RESET_B signal comes up by default as gpio2 (pin #0), which is ALT function 1. This particular signal/pin is used as a simple GPIO to reset the FEC PHY.

8.3 Source Code

The source code for the Freescale FEC Linux environment is located under the `../ltib/rpm/BUILD/linux/drivers/net` directory. It contains the following files:

Table 8-2. Source Code Files

	File Names
FEC low-level Ethernet driver:	<ul style="list-style-type: none"> • <code>fec.h</code> • <code>fec.c</code>
MAC Switch software	<ul style="list-style-type: none"> • <code>fec_switch.h</code> • <code>fec_switch.c</code>
IEEE 1588 PTP (network time sync)	<ul style="list-style-type: none"> • <code>fec_1588.h</code> • <code>fec_1588.c</code>
MPC52xx PowerPC Ethernet Driver	<ul style="list-style-type: none"> • <code>fec_mpc52xx.h</code> • <code>fec_mpc52xx.c</code> • <code>fec_mpc52xx_phy.c</code>

Of those files, only the FEC low-level Ethernet driver code (`fec.[ch]`) constitutes the Linux i.MX6 FEC driver.

The driver uses the following compile definitions:

`CONFIG_FEC_1588` Set for IEEE 1588 network time synchronization.

`CONFIG_M5272` PowerPC information. Can be safely ignored and should not be set.

`CONFIG_MXC` IMXxx parts. Should be defined.

`CONFIG_MXS` Legacy MXS part. Should generally not be defined.

8.4 Ethernet Configuration

This section covers aspects such as duplex and speed configurations.

The two most common issues are as follows:

- MAC address is missing or invalid
- Ethernet connection (duplex, speed)

By default, the Ethernet driver reads the burned-in MAC address, which is found in code from the `fec.c` file located in the function `fec_get_mac()`. If no MAC address exists in the hardware, the MAC is read as all zeros, which creates problems. If this occurs, modify the code to read the MAC address from Flash or elsewhere.

The FEC driver and hardware are designed to comply to the IEEE standards for Ethernet auto-negotiation. See the FEC chapter in the *i.MX6 Applications Processor Reference Manual* for a description of using flow control in full duplex and more.

Chapter 9

Porting USB Host1 and USB OTG

9.1 Overview

There are up to four USB ports on i.MX6Q platform

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

The following power supplies must be provided:

- 5V power supply for USB OTG VBUS
- 5V power supply for USB H1 VBUS
- 3.3V power supply for HSIC1/2 port
- 3.15 +/- 5%V power supply for USB OTG/H1 PHY. Since this power can be routed from USB OTG/H1 VBUS, that means 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 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 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 which has a alternate USBOTG_ID function must be used.

- USBOTG_ID
- USBOTG_OC_B
- one pin used to control 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, needed 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, needed to set with proper IOMUX:

- H2_STROBE
- H3_STROBE
- H2_DATA
- H3_DATA

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

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductors products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor 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. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claims alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-complaint and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2012 Freescale Semiconductor, Inc.