i.MX 6SoloLite BSP Porting Guide

Document Number: IMX6SLBSPPG
Rev L3.0.35_4.1.0, 09/2013

Contents

Chapter 1
Porting U-Boot to an i.MX 6SoloLite Custom Board

1.1 U-Boot Overview

This chapter provides a step-by-step guide that explains how to add i.MX 6 custom board support for U-Boot.

This developer's guide is based on U-Boot v2009.08 plus LTIB-based package for the i.MX patches. Please refer to the release notes.

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>.

1.2.1 Preparing the Code

The following steps explain how to prepare the code.

1. Make a copy of the board directory, as shown below:

   ```
   $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, as shown below:

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

NOTE

You should pay attention to the following configurations when using 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 (where x denotes an index between 0 and CONFIG_NR_DRAM_BANKS-1, inclusive).
- *CONFIG_NR_DRAM_BANKS*, *PHYS_SDRAM_x* and *PHYS_SDRAM_x_SIZE* will be passed to kernel. If these configs are wrong, kernel might 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 6SoloLite-based configuration. This file is in alphabetical order. The instruction for 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 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_<reference 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 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-)
   make distclean;
   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.MX 6 custom board that you have created is an exact copy of the i.MX 6 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.

1.3 Customizing the i.MX 6 Custom Board Code

The new i.MX 6 custom board is part of the U-Boot source tree, but it is a duplicate of the i.MX 6 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.

1.3.1 Changing the DCD Table for i.MX 6 LPDDR2 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 code blocks 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 len = <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 len = <dcd items> * 8 + 4.
  e.g.

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

1.3.2 Booting with the Modified U-Boot

The content below explains how to compile and write u-boot.bin to SD card.

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 >
```

1.3.3 Add New Driver Initialize Code to Board Files

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

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

NOTE

- **board_init**() function will be called earlier before UART initialization. Please do not attempt to use printf

in this function, otherwise U-Boot will crash. Most of driver init functions are put into **board_init**() function.
- **board_late_init**() function will be called fairly later. For debugging initialization code, driver init functions may be put in it.

1.3.4 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>.

1.3.5 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.MX 6 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.

1.4 How to Debug

Normally we have two ways for debugging:

- Use Realview ICE.
- Use printf.

1.4.1 Use RealView ICE for Debugging

Normally we use RealView ICE to debug in very early stage, e.g. before uart initialization, or when it 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 0 in rules.mk which will be easier for debugging in RealView ICE.

1.4.2 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 IOMUX Overview

Before using the i.MX 6SoloLite 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.MX 6SoloLite Multimedia Applications Processor Reference Manual*. For additional information about the IOMUX controller block, see the "IOMUX Controller (IOMUXC)" chapter in the *i.MX 6SoloLite Multimedia 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 6SoloLite registers, as follows:

- 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 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 6SoloLite 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-mx6x.h file contains each pad's IOMUX definitions where the x denotes the SOC type. Use the following code to see the default definitions:

```
#define MX6x_PAD_<PIN NAME>__<FUNC NAME> (_MX6x_PAD_<PIN NAME>__<FUNC NAME> |
MUX_PAD_CTRL(MX6x_<PAD CTRL>))
```

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

```
_MX6x_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

```
MX6x_<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-mx6x_<reference board name>.c file contains the initialization functions for all peripherals (such as UART, I^{2}C, and Ethernet). Configure the relevant pins for each initializing function by using the following:

```
mxc_iomux_v3_setup_pad(iomux_v3_cfg_t pad);
mxc_iomux_v3_setup_multiple_pads(iomux_v3_cfg_t *pad_list, unsigned count)
```

Where the following applies:

<pad> < is a macro composed of mux mode, pad ctrl, and input select config. See IOMUX_PAD() definition in arch/arm/plat-mxc/include/mach/iomux-v3.h;

<pad_list> is an array of <pad>;

<count> is the size of the array of <pad>;

2.3.3 Example-Setting a GPIO

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

Add the following code to the file board-mx6x_<reference board name>.h, in the array of mx6x_<reference board name>_pads where x denotes the SOC type:

```
MX6x_PAD_SD2_DAT1__GPIO1_14;
```

Make sure that no any other SD2_DAT1 configuration exists in the array. Then, if done correctly, the pin SD2_DAT1 on the i.MX 6SoloLite 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 6SoloLite <reference> boards are linux/arch/arm/mach-mx6/board-mx6x_<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
linux/arch/arm/plat-mxc/include/mach/	iomux-mx6x.h(x denotes SOC type)	IOMUX configuration definitions
linux/arch/arm/mach-mx6	board-mx6x_<reference board name>.h	Machine Layer File. Contains IOMUX configuration structures

2.4.1 IOMUX Configuration Definition

The iomux-mx6x.h (x denotes SOC type) file contains definitions for all i.MX 6SoloLite 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 MX6x_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 6SoloLite Applications Processor Reference Manual*.

2.4.2 Machine Layer File

The board-mx6x_<reference board name>.h file contains structures for configuring the pads(x denotes the SOC type).

They are declared as follows:

```
static iomux_v3_cfg_t mx6x_<reference board name>_pads[] = {
…
…
…
MX6x_PAD_SD2_CLK__USDHC2_CLK,
MX6x_PAD_SD2_CMD__USDHC2_CMD,
MX6x_PAD_SD2_DAT0__USDHC2_DAT0,
MX6x_PAD_SD2_DAT1__USDHC2_DAT1,
MX6x_PAD_SD2_DAT2__USDHC2_DAT1,
MX6x_PAD_SD2_DAT3__USDHC2_DAT3,
…
…
…
};
```

Add the pad's definitions from iomux-mx6x.h to the above code.

On init function (in this example "mx6x_<reference board name>_init" function), set up the pads using the following function:

```
mxc_iomux_v3_setup_multiple_pads(mx6x_<reference board name>_pads,
ARRAY_SIZE(mx6x_<reference board name>_pads));
```

2.4.3 Example -Setting a GPIO

For 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-mx6x.h (x denotes SOC type) file as follows:

```
#define MX6x_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-mx6x_<reference board name>.h file as shown in the following code:

```
static iomux_v3_cfg_t mx6x_<reference board name>_pads[] = {
…
…
…
MX6x_PAD_NANDF_CS0__GPIO_6_11,
…
…
…
};
```

Chapter 3
Registering a New UART Driver

3.1 UART Overview

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

3.1.1 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.

Take the i.MX 6 x, where x denotes the SOC type, reference board as an example. The machine layer file used on the i.MX 6 x reference boards is linux/arch/arm/mach-mx6/board-mx6x_evk.c.

The iomux-mx6x.h file contains the definitions for all i.MX 6SoloLite pads. Configure the UART pads as follows:

```
/* UART4 */
#define MX6x_PAD_KEY_COL0_UART4_TXD
            \(_MX6x_PAD_KEY_COL0__UART4_TXD | MUX_PAD_CTRL(MX6x_UART_PAD_CTRL)

#define MX6x_PAD_KEY_ROW0_UART4_TXD
            \(_MX6x_PAD_KEY_RAW0__UART4_TXD | MUX_PAD_CTRL(MX6x_UART_PAD_CTRL)
```

The structures for configuring the pads are contained in the mx6x_<reference board name>.h 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 mx6x_brd_pads[] = {
…
…
…
          MX6x_PAD_KEY_COL0_UART4_TXD,
          MX6x_PAD_KEY_ROW0_UART4_TXD,
…
…
…
};
```

Use the following function to set up the pads on the init function mx6_evk_init() (found in the board-mx6x_evk.c file).

3.1.2 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
               [*] Console on IMX serial port
```

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

3.1.3 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 processor, connection and communication will fail. There is a simple way to test whether the UART is properly configured and enabled.

In Linux command line, type the following:

```
echo "test" > /dev/ttymxc1
```

3.1.4 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
imx_uart.h	UART header containing UART configuration and data structures
iomux-<platform>.h	IOMUX pads definitions

Chapter 4
Adding Support for SDHC

4.1 SDHC Overview

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 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-mx6sl.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 then SD/MMC memory cards. Make sure that the power supply is on SD slot while using SDIO, or please apply an external power to SDIO instead.

Chapter 5
Configuring the SPI NOR Flash Memory Technology Device (MTD) Driver

5.1 SPI NOR Overview

This chapter explains 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.

5.2 Source code structure

The SPI NOR MTD driver is implemented in the following file:

```
<ltib_dir>/rpm/BUILD/linux/drivers/mtd/devices/m25p80.c
```

The SPI NOR MTD partitions are implemented in the following file:

```
<ltib_dir>/rpm/BUILD/linux/arch/arm/mach-mx6/board-mx6sl_<boardname>.c
```

5.2.1 Configuration options

Freescale's BSP supports the following SPI NOR Flash models.

- "SST 25VF016B" "sst25vf016b"
- "M25P32-VMW3TGB" "m25p32"

Those models are defined in the structure

```
static const struct spi_device_id m25p_ids[],
```

located at

```
<ltib_dir>/rpm/BUILD/linux/drivers/mtd/devices/m25p80.c
```

5.2.2 Selecting SPI NOR on the Linux image

Follow these steps to enable support for SPI NOR:

1. Open the file (located at arch/arm/mach-mx6) and modify the structure called static struct flash_platform_data xxxx_spi_flash_data[]
2. Write the name of the data flash desired on the .type variable of this structure. This name must be exactly the same as it appears on the m25p_ids[] structure.
3. Set the number of partitions you want to use on the SPI NOR Flash. On the <board name>.c file, go to the structure called static struct mtd_partition xxxx_spi_nor_partitions[]
4. Each partition has three elements: the name of the partition, the offset, and the size. By default, these elements are partitioned into a bootloader section and a kernel section, and defined as:

   ```
   .name = "bootloader",
   .offset = 0,
   .size = 0x00100000,

   .name = "kernel",
   .offset = MTDPART_OFS_APPEND,
   .size = MTDPART_SIZ_FULL,
   ```

   Bootloader starts from address 0 and has a size of 1M byte. Kernel starts from address 1M byte.

   NOTE

   You may create more partitions or modify the size and names of these ones.

5. To get to the SPI NOR MTD driver, use the command ./ltib -c when located in the <ltib dir>.
6. On the screen displayed, select **Configure the kernel** and exit.
7. When the next screen appears, enable the SPI NOR MTD driver. This option is available under Device Drivers > Memory Technology Device (MTD) support > Self-contained MTD device drivers > Support most SPI Flash chips (AT26DF, M25P, ...). The configuration is called CONFIG_MTD_M25P80. This configuration enables access to the SPI-NOR chips.

5.3 Changing the SPI interface configuration

The i.i.MX 6SoloLite chip has five ECSPI interfaces. By default, the i. i.MX 6SoloLiteBSP configures ECSPI-1 interface in the master mode to connect to the SPI NOR Flash.

5.3.1 Changing the ECSPI Interface

To change the ECSPI interface used, use the following procedure:

1. Locate the file at arch/arm/mach-mx6/<board name>.c
2. Look for the structure spi_board_info. The field bus_num decides which ECSPI module to use. This starts from 0 which indicates ECSPI1 and so on. The field chip_select decides which chip select to use within the ECSPI module.
3. Use the function spi_board_register_info() to register the ECSPI interface.

5.3.2 Changing the Chip Select

To change the chip select used, locate the file at arch/arm/mach-mx6/board-mx6q<board name>.c and use the static struct spi_board_info structure.

Replace the value of ".chip_select" variable with the desired chip select value. For example, .chip_select = 3 sets the chip select to number 3 on the ECSPI interface.

5.3.3 Changing the external signals

The iomux-mx6q.h/iomux-mx6sl.h file contains the definitions for all pads. Find the configuration for the ESCPI pins needed for the SPI-NOR and add it to the <boardname>_pads[] structure found in arch/arm/mach-mx6/board-<board name>.h.

5.4 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.

5.4.1 Software Operation

In a Flash-based embedded Linux system, a number of Linux technologies work together to implement a file system.

Figure below illustrates the relationships between standard components.

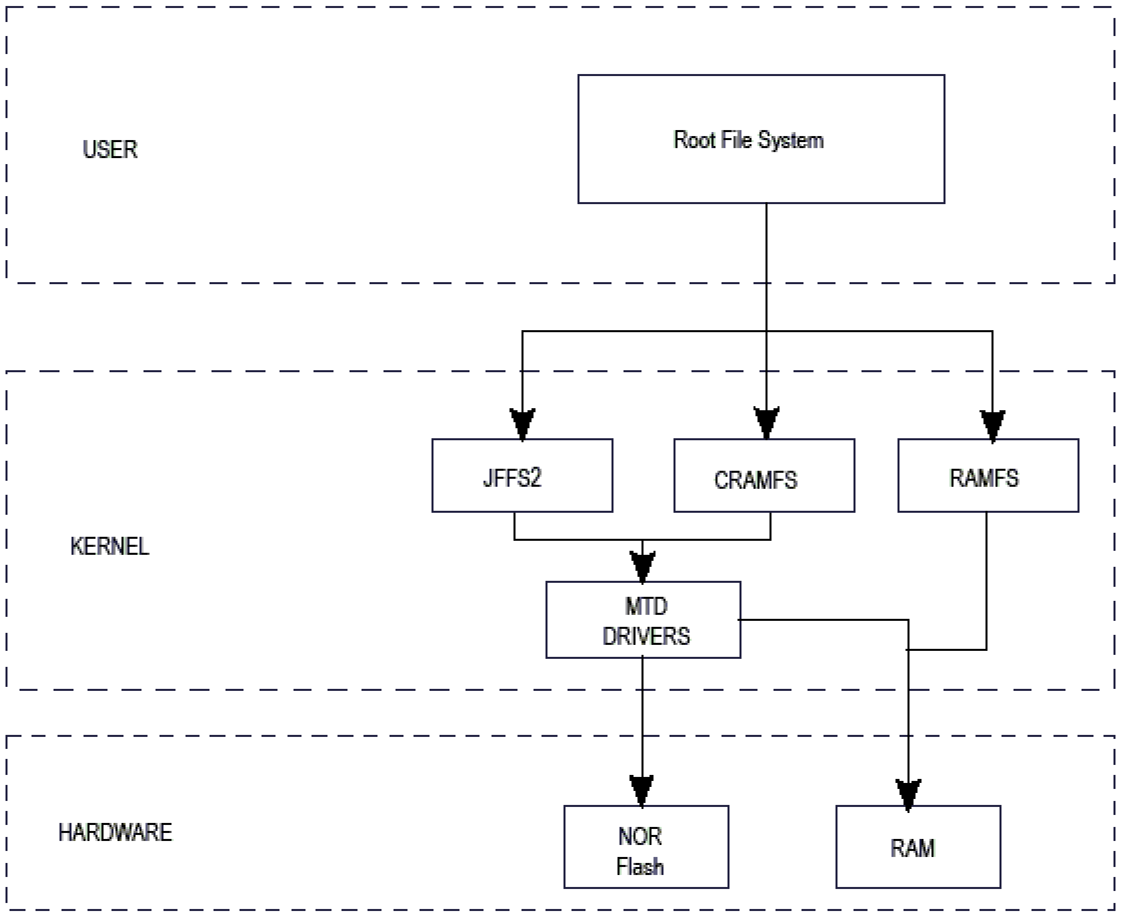

Figure 5-1. Components of a Flash-based file system

The MTD subsystem for Linux is a generic interface to memory devices such as Flash and RAM which provides simple read, write, and erase access to physical memory devices. Devices called mtdblock devices can be mounted by JFFS, JFFS2, and CRAMFS file systems. The SPI NOR MTD driver is based on the MTD data Flash driver in the kernel by adding SPI accesses.

In the initialization phase, the SPI NOR MTD driver detects a data Flash by reading the JEDEC ID. The driver then adds the MTD device. The SPI NOR MTD driver also provides the interfaces to read, write, erase NOR Flash.

Chapter 6
Porting Audio Codecs to a Custom Board

6.1 Audio 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.

6.1.1 Common Porting Task

The mxc_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-mx6sl_<board name>.c.

```
static struct mxc_audio_platform_data wm8962_data = {
          .ssi_num = 1,
          .src_port = 2,
          .ext_port = 3,
          .hp_gpio = MX6_BRD_HEADPHONE_DET,
          .hp_active_low = 1,
          .mic_gpio = -1,
          .mic_active_low = 1,
          .init = mxc_wm8962_init,
          .clock_enable = wm8962_clk_enable,
};
```

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 micphone detection

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

init: initialize wm8962 resource relevant to board, such as mclk source

clock_enable: a callback for enable/disable mclk

6.1.2 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-mx6sl_<board name>.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 EVK BOARD, are as follows:

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

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

Table 6-1. 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

6.1.3 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 6-2. Files for wm8962 Codec Support

File Name	Definition
imx-pcm-dma-mx2.c	• Shared by the stereo ALSA SoC driver, the esai driver, and the spdif driver. • Responsible for preallocating DMA buffers and managing DMA channels.
imx-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 table above accordingly. The exact adaptation will depend on the codec chosen. Obtain the codec-specific software from the codec vendor.

Chapter 7
Porting the Fast Ethernet Controller Driver

7.1 FEC Overview

This chapter explains how to port the fast Ethernet controller (FEC) driver to the i.MX 6 processor.

Using Freescale's standard (FEC) driver makes porting to the i.MX 6 simple. Porting needs to address the following three areas:

- Pin configuration
- Source code
- Ethernet connection configuration

7.1.1 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.

A brief overview of the device functionality is provided here. For details see the FEC chapter of the *i.MX 6SoloLite 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 modes uses a subset of the 18 signals. These signals are listed in table below.

Table 7-1. Pin Usage in MII, RMIIModes

Direction	EMAC Pin Name	MII Usage	RMII Usage	RGMII Usage (i.MX 6SL don't support)
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
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.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 8 muxing options called ALT modes. For further explanation refer to IOMUX chapter in the *i.MX 6SoloLite Multimedia Applications Processor Reference Manual*.

Please note that the pin FEC_PHY_RESET_B is used as a simple GPIO to reset the FEC PHY before enabling phy clock, otherwise some phys will not work correctly.

7.1.2 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 7-2. Source Code Files

File Names	Descriptions
• fec.h • fec.c	FEC low-level Ethernet driver:

The driver uses the following compile definitions:

CONFIG_FEC: enable FEC driver.

CONFIG_FEC_NAPI: enable NAPI polling method for ethernet RX path.

7.1.3 Ethernet Configuration

This section mainly covers FEC bring up issues. Please refer to *i.MX 6SoloLite Multimedia Applications Processor Reference Manual* FEC chapter if you want to know more about FEC MAC configuration.

Please note the following during FEC bring up:

- Configure all I/O pins used by MAC correctly for related function.
- Check phy input clock and power, phy led1 and led2 lighten on if clock and power input are ok.
- Make sure MAC tx_clk has right clock input, otherwise MAC cannot work.
- Make sure MAC address is set and valid.

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 reads all zeros, which makes MAC malfunction. If this occurs, please add the MAC address in the U-Boot command line for kernel, such as add early parameter "fec_mac=00:01:02:03:04:05" in bootargs.

The FEC driver and hardware are designed to comply with the IEEE standards for Ethernet auto-negotiation. See the FEC chapter in the *i.MX 6SoloLite Multimedia Applications Processor Reference Manual* for a description of using flow control in full duplex and more.

Chapter 8
Porting USB Host1 and USB OTG

8.1 USB Overview

There are up to four USB ports on i.MX 6 serial application processors:

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

NOTE

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

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 an 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

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

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: IMX6SLBSPPG
Rev. L3.0.35_4.1.0
09/2013