飞凌嵌入式
FORLINX EMBEDDED

技术参考手册

OKMX6UL-C 应用笔记 — linux

Rev. 1.3

2019/09/04

技术支持与定制

1、技术支持范围

1.1 本公司产品的软、硬件资源提供情况咨询；
1.2 本公司产品的软、硬件手册使用过程中遇到的问题；
1.3 本公司提供的 OEM、ODM 售后技术支持；
1.4 本公司产品的故障判断及售后维修服务；

2、技术讨论范围

2.1 源码的修改以及理解；
2.2 操作系统如何移植；
2.3 用户在自行修改以及开发中遇到的软硬件问题；
注：以上三点虽不属于技术支持范围，但我公司会尽力为用户提供帮助，如依然没能解决您的问题，敬请谅解；

3、技术支持方式

3.1 电话：0312-3119192 / 0312-3102619
3.2 论坛：bbs.witech.com.cn
3.3 邮箱：

Linux 技术支持：	linux@forlinx.com
Android 技术支持：	android@forlinx.com
硬件技术支持：	hardware@forlinx.com

3.4 知识库：bbs.witech.com.cn/kb

4、技术支持时间

周一至周五: 上午 9:00—11:30,下午 13:30—17:00；
公司按照国家法定节假日安排休息，在此期间无法提供技术支持，期间请发邮箱或论坛技术支持区，我们会在工作日尽快给您回复。

5、定制开发服务

我公司提供嵌入式操作系统底层驱动、硬件板卡的有偿定制开发服务，以缩短您的产品开发周期；
了解定制流程：http://www.forlinx.com/OEM.htm
填写需求文档：http://www.forlinx.com/docs/PR.docx
发至项目邮箱：project@forlinx.com

资料更新与获取

1、资料的更新

产品相关资料会不断的完善更新，包括本手册内容亦然如此；当您在使用这些内容时，请确保其为最新状态；

2、更新后如何通知

飞凌嵌入式产品资料更新通知采用微信公众号推送，敬请关注！

订阅号

3、资料如何获取

3.1 网络下载：

请注册并登陆“bbs.witech.com.cn”找到“开发板资料下载”选择对应平台下载；

下载前请阅读《资料下载说明》：http://bbs.witech.com.cn/thread-67932-1-1.html；

3.2 光盘：

请联系我公司销售人员购买；

版权声明

更新记录

日期	版本	更新内容
2017.7.19	V1.0	建档
2017.12.15	V1.1	支持 python2.7.9,1g nand/emmc flash 备份 dtb 与 kernel,利用 kobs-ng 更新 nand u-boot；添加关于 GPIO_IO08,GPIO_IO09 的使用注意事项说明。
2018.10.30	V1.2	增加了 JDK 的使用，将硬浮点运算及 sdio wifi 的使用添加到此手册中； 增加 SSH 登录的使用； 增加 4G-AP 的使用
2019.09.04	V1.3	增加设备树说明，修改部分设备树名称错误

目　录

第一章 软件定制

此处只是举例说明，软件版本更新之后，有些内容可能会及时更新，修改方法参考下面修改。

除 1.9 节“NAND 增加分区”以外，其他章节都以 imx6ul-c emmc 开发板为例，修改设备树为 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts，其他型号开发板，以此为参考，修改对应的设备树

emmc 设备树：

ul-c	imx6ul-14x14-evk-emmc-c-7-1024x600.dts	ul-c emmc 启机默认设备树
	imx6ul-14x14-evk-emmc-c-7-800x480.dts	800*480 7 寸屏设备树
	imx6ul-14x14-evk-emmc-c-4.3-r.dts	4.3 寸设备树
	imx6ul-14x14-evk-emmc-c-10.4-r.dts	10.4 寸屏设备树
	imx6ul-14x14-evk-emmc-c-5.6-r.dt	5.6 寸屏设备树
	imx6ul-14x14-evk-emmc-c-8-r.dts	8 寸屏设备树
ul-c2	imx6ul-14x14-evk-emmc-c2-7-1024x600.dts	ul-c2 emmc 启机默认设备树
	imx6ul-14x14-evk-emmc-c2-7-800x480.dts	800*480 7 寸屏设备树
	imx6ul-14x14-evk-emmc-c2-10.4-r.dts	10.4 寸屏设备树
	imx6ul-14x14-evk-emmc-c2-4.3-r.dts	4.3 寸设备树
	imx6ul-14x14-evk-emmc-c2-5.6-r.dts	5.6 寸屏设备树
	imx6ul-14x14-evk-emmc-c2-8-r.dts	8 寸屏设备树
ul-c3	imx6ul-14x14-evk-emmc-c3-iomuxc.dts	默认设备树
	imx6ul-14x14-evk-emmc-c3-iomuxc-1024x600.dts	1024*600 7 寸屏设备树
	imx6ul-14x14-evk-emmc-c3-gpio.dts	gpio dtb 方式采用的是\用户资料\应用笔记\GPIO 方式控制，生成/dev/gpio 设备节点
	imx6ul-14x14-evk-emmc-c3-gpio-1024x600.dts	

1g nand 设备树：

ul-c	imx6ul-14x14-evk-gpmi-c-1g-7-1024x600.dts	默认设备树
	imx6ul-14x14-evk-gpmi-c-1g-7-800x480.dts	800*480 7 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-1g-10.4-r.dts	10.4 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-1g-8-r.dts	8 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-1g-5.6-r.dts	5.6 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-1g-4.3-r.dts	4.3 寸设备树
ul-c2	imx6ul-14x14-evk-gpmi-c2-1g-7-1024x600.dts	默认设备树
	imx6ul-14x14-evk-gpmi-c2-1g-7-800x480.dts	800*480 7 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-1g-10.4-r.dts	10.4 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-1g-8-r.dts	8 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-1g-5.6-r.dts	5.6 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-1g-4.3-r.dts	4.3 寸设备树
ul-c3	imx6ul-14x14-evk-gpmi-c3-1g-iomuxc.dts	默认设备树
	imx6ul-14x14-evk-gpmi-c3-1g-iomuxc-1024x600.dts	1024*600 7 寸屏设备树

	imx6ul-14x14-evk-gpmi-c3-1g-gpio.dts	gpio dtb 方式采用的是\用户资料\应用笔记\GPIO 方式控制，生成/dev/gpio 设备节点
	imx6ul-14x14-evk-gpmi-c3-1g-gpio-1024x600.dts	1024*600 7 寸屏设备树

256m nand 设备树：

ul-c	imx6ul-14x14-evk-gpmi-c-256m-7-1024x600.dts	默认设备树
	imx6ul-14x14-evk-gpmi-c-256m-7-800x480.dts	800*480 7 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-256m-10.4-r.dts	10.4 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-256m-8-r.dts	8 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-256m-5.6-r.dts	5.6 寸屏设备树
	imx6ul-14x14-evk-gpmi-c-256m-4.3-r.dts	4.3 寸设备树
ul-c2	imx6ul-14x14-evk-gpmi-c2-256m-7-1024x600.dts	默认设备树
	imx6ul-14x14-evk-gpmi-c2-256m-7-800x480.dts	800*480 7 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-256m-10.4-r.dts	10.4 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-256m-8-r.dts	8 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-256m-5.6-r.dts	5.6 寸屏设备树
	imx6ul-14x14-evk-gpmi-c2-256m-4.3-r.dts	4.3 寸设备树
ul-c3	imx6ul-14x14-evk-gpmi-c3-256m-iomuxc.dts	默认设备树
	imx6ul-14x14-evk-gpmi-c3-256m-iomuxc-1024x600.dts	1024*600 7 寸屏设备树
	imx6ul-14x14-evk-gpmi-c3-256m-gpio.dts	gpio dtb 方式采用的是\用户资料\应用笔记\GPIO 方式控制，生成/dev/gpio 设备节点
	imx6ul-14x14-evk-gpmi-c3-256m-gpio-1024x600.dts	1024*600 7 寸屏设备树

nand 设备树包含与其对应的 emmc 设备树，大部分配置在 emmc 设备树中

```
/*
 * Copyright (C) 2015 Freescale Semiconductor, Inc.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#include "imx6ul-14x14-evk-emmc-c-7-800x480.dts"
&qspi{
            status = "disabled";
};

&usdhc2 {
            status = "disabled";
};
```

1.1 管脚复用的参数配置方法（PINMUX）

arch/arm/boot/dts/imx6ul-pinfunc.h 中有

```
/*
 * The pin function ID is a tuple of
 * <mux_reg conf_reg input_reg mux_mode input_val>
 */
```

```
#define MX6UL_PAD_LCD_DATA16__UART7_DCE_TX  0x0158 0x03E4 0x0000 1 0
```

arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts 中有

```
pinctrl_uart7: uart7grp {
  fsl,pins = <
    MX6UL_PAD_LCD_DATA17__UART7_DCE_RX  0x1b0b1
    MX6UL_PAD_LCD_DATA16__UART7_DCE_TX  0x1b0b1
  >;
};
```

将管脚的配置展开即：0x0158 0x03E4 0x0000 1 0 0x1b0b1

0x0158 | 0x03E4 | 0x0000 | 0x1 | 0x0 | 0x1b0b1

---

mux_ctrl_ofs | pad_ctrl_ofs | sel_input_ofs | mux_mode | sel_input | pad_ctrl

以上参数在参考手册怎么确定的呢？
下面以 LCD_DATA16 复用为 UART7_DCE_TX 为例说明复用管脚参数配置的方法。
注：下述参考手册为《IMX6ULRM.pdf》。

对于复用管脚的配置，应该在手册管脚复用的章节（IOMUXC）中查找。但是在确定 pad name 才方便，于是定义在 **External Signals and Pin Multiplexing** 章节，搜索 **MX6UL_PAD_LCD_DATA16__UART7_DCE_TX** 的中间部分“**LCD_DATA16**”可以直接跳转至 LCD_DATA16 引脚的寄存器章节。

其中 mux_ctrl_ofs 为 0x0158，mux_mode 为 ATL1，如图：

31.5.82 SW_MUX_CTL_PAD_LCD_DATA16 SW MUX Control Register (IOMUXC_SW_MUX_CTL_PAD_LCD_DATA16)

SW_MUX_CTL Register

Address: 20E_0000h base + 158h offset = 20E_0158h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved											SION	MUX_MODE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

IOMUXC_SW_MUX_CTL_PAD_LCD_DATA16 field descriptions

Field	Description
31–5 -	This field is reserved. Reserved
4 SION	Software Input On Field. Force the selected mux mode Input path no matter of MUX_MODE functionality. 1 **ENABLED** — Force input path of pad LCD_DATA16 0 **DISABLED** — Input Path is determined by functionality
MUX_MODE	MUX Mode Select Field. Select 1 of 9 iomux modes to be used for pad: LCD_DATA16. 0000 **ALT0** — Select mux mode: ALT0 mux port: LCDIF_DATA16 of instance: lcdif 0001 **ALT1** — Select mux mode: ALT1 mux port: UART7_TX of instance: uart7 0011 **ALT3** — Select mux mode: ALT3 mux port: CSI_DATA01 of instance: csi 0100 **ALT4** — Select mux mode: ALT4 mux port: EIM_DATA08 of instance: eim 0101 **ALT5** — Select mux mode: ALT5 mux port: GPIO3_IO21 of instance: gpio3 0110 **ALT6** — Select mux mode: ALT6 mux port: SRC_BT_CFG24 of instance: src 1000 **ALT8** — Select mux mode: ALT8 mux port: USDHC2_DATA6 of instance: usdhc2

pad_ctrl_ofs 为 0x03E4，并根据此配置 pad_ctrl 为 0x1b0b1（配置上拉电阻、频率等等），如图：

31.5.245 SW_PAD_CTL_PAD_LCD_DATA16 SW PAD Control Register (IOMUXC_SW_PAD_CTL_PAD_LCD_DATA16)

SW_PAD_CTL Register

Address: 20E_0000h base + 3E4h offset = 20E_03E4h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															HYS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PUS		PUE	PKE	ODE	Reserved			SPEED		DSE			Reserved		SRE
Reset	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0

IOMUXC_SW_PAD_CTL_PAD_LCD_DATA16 field descriptions

Field	Description
31–17 -	This field is reserved. Reserved
16 HYS	Hyst. Enable Field Select one out of next values for pad: LCD_DATA16 0 **HYS_0_Hysteresis_Disabled** — Hysteresis Disabled 1 **HYS_1_Hysteresis_Enabled** — Hysteresis Enabled
15–14 PUS	Pull Up / Down Config. Field Select one out of next values for pad: LCD_DATA16 00 **PUS_0_100K_Ohm_Pull_Down** — 100K Ohm Pull Down 01 **PUS_1_47K_Ohm_Pull_Up** — 47K Ohm Pull Up 10 **PUS_2_100K_Ohm_Pull_Up** — 100K Ohm Pull Up 11 **PUS_3_22K_Ohm_Pull_Up** — 22K Ohm Pull Up
13 PUE	Pull / Keep Select Field Select one out of next values for pad: LCD_DATA16 0 **PUE_0_Keeper** — Keeper 1 **PUE_1_Pull** — Pull
12 PKE	Pull / Keep Enable Field Select one out of next values for pad: LCD_DATA16

input_ofs 查找 IOMUXC 章节以 SELECT_INPUT 结尾的部分，中间选择 UART7_DCE_RTS，如果没有这里 sel_input_ofs=0x000 即可，对应的 sel_input 为 0 即可。

如果有例如 MX6UL_PAD_ENET1_RX_ER__UART7_DCE_RTS，如下图，所以 ENET1_RX_ER 的 sel_input_ofs=0x650。所以 ENET1_RX_ER（MX6UL_PAD_ENET1_RX_ER__UART7_DCE_RTS）的 sel_input=0x1。

```
#define MX6UL_PAD_ENET1_RX_ER__UART7_DCE_RTS                    0x00E0 0x036C 0x0650 1 1
```

31.5.400 UART7_RTS_B_SELECT_INPUT DAISY Register (IOMUXC_UART7_RTS_B_SELECT_INPUT)

DAISY Register

Address: 20E_0000h base + 650h offset = 20E_0650h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved														DAISY	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

i.MX 6UltraLite Applications Processor Reference Manual, Rev. 0, 08/2015

1936 Freescale Semiconductor, Inc.

Chapter 31 IOMUX Controller (IOMUXC)

IOMUXC_UART7_RTS_B_SELECT_INPUT field descriptions

Field	Description
31–2 -	This field is reserved. Reserved
DAISY	Selecting Pads Involved in Daisy Chain. Instance: uart7, In Pin: uart_rts_b 00 **ENET1_TX_CLK_ALT1** — Selecting Pad: ENET1_TX_CLK for Mode: ALT1 01 **ENET1_RX_ER_ALT1** — Selecting Pad: ENET1_RX_ER for Mode: ALT1 10 **LCD_DATA06_ALT1** — Selecting Pad: LCD_DATA06 for Mode: ALT1 11 **LCD_DATA07_ALT1** — Selecting Pad: LCD_DATA07 for Mode: ALT1

1.2 SPI 接口

R61	E4	ecspi2.SCLK	CSI_DATA00
R63	E3	ecspi2.SS0	CSI_DATA01
R65	E2	ecspi2.MOSI	CSI_DATA02
R67	E1	ecspi2.MISO	CSI_DATA03
R69	D4	ecspi1.SCLK	CSI_DATA04
R71	D3	ecspi1.SS0	CSI_DATA05
R73	D2	ecspi1.MOSI	CSI_DATA06
R75	D1	ecspi1.MISO	CSI_DATA07
	BALL	SPI功能引脚	摄像头功能引脚
R+数字代表连接器右边引脚号			

1. spi 接口采用 ecspi1。

2. 查看 IMX6ULRM.pdf 手册中 Chapter 4 External Signals and Pin Multiplexing 有

ECSPI1	MISO	CSI_DATA07	ALT3
		LCD_DATA23	ALT2
	MOSI	CSI_DATA06	ALT3
		LCD_DATA22	ALT2
	RDY	LCD_DATA12	ALT8
	SCLK	CSI_DATA04	ALT3
		LCD_DATA20	ALT2
	SS0	CSI_DATA05	ALT3
		LCD_DATA21	ALT2
	SS1	LCD_DATA05	ALT8
	SS2	LCD_DATA06	ALT8
	SS3	LCD_DATA07	ALT8

现 ecspi1 的 miso 采用 csi_data07 ，mosi 采用 csi_data06， sclk 采用 csi_data04，如果需要采用 ss0-3 根据自己需要选择。

3. 怎么查找使用哪个驱动，采用 config 中的哪个进行配置呢？

1.1 查找 ecspi 的驱动和配置选项。

设备树文件 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts 中有：

```
#include <dt-bindings/input/input.h>
#include "imx6ul.dtsi"
```

arch/arm/boot/dts/imx6ul.dtsi 为通用设备树配置文件。可以在此搜索驱动配置时的名字。比如 ecpsi1，搜索到如下：

```
ecspi1: ecspi@02008000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl,imx6ul-ecspi", "fsl,imx51-ecspi";
    reg = <0x02008000 0x4000>;
    interrupts = <GIC_SPI 31 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&clks IMX6UL_CLK_ECSPI1>,
         <&clks IMX6UL_CLK_ECSPI1>;
    clock-names = "ipg", "per";
    dmas = <&sdma 3 7 1>, <&sdma 4 7 2>;
    dma-names = "rx", "tx";
    status = "disabled";
};
```

对应驱动匹配时的名字为 fsl,imx6ul-ecspi 或者 fsl,imx51-ecspi。

一般情况下驱动多放置在 drviers 路径下。在此路径下查找 grep "fsl,imx6ul-ecspi" ./ -nr 搜索不到驱动。再用 grep "fsl,imx51-ecspi" ./ -nr 查找，如下（如果，在 drivers 下没有找到，可以在 linux3.14.38 主目录下全局搜索）：

```
匹配到二进制文件 ./built-in.o
./spi/spi-imx.c:708:    { .compatible = "fsl,imx51-ecspi", .data = &imx51_ecspi_devtype_data, },
匹配到二进制文件 ./spi/spi-imx.o
```

对应驱动路径应该为 drivers/spi/spi-imx.c。

这时再查找具体使用 config 中的哪个配置，如下步骤：

先查看 drives/spi/Makefile 文件，此文件将 spi 路径下的驱动文件和配置文件中具体哪个配置联系起来。spi-imx.c 文件编译之后为 spi-imx.o 文件。

```
obj-$(CONFIG_SPI_IMX)           += spi-imx.o
```

上图也就是说，需要 config 中需要加入 CONFIG_SPI_IMX 配置选项。

在 6ul 的 emmc 的配置文件 linux_imx6ul_emmc_defconfig 中将 CONFIG_SPI_IMX 设置为 y，编译进内核。如图：

```
#
# SPI Master Controller Drivers
#
# CONFIG_SPI_ALTERA is not set
CONFIG_SPI_BITBANG=y
CONFIG_SPI_GPIO=y
CONFIG_SPI_IMX=y
```

1.2 查找 ecspi 下挂载具体设备的驱动和配置选项。

```
&ecspi1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecspi1>;
        fsl,spi-num-chipselects = <1>;
        cs-gpios = <&gpio4 26 GPIO_ACTIVE_LOW>;
        status = "okay";

        spidev@0{
            compatible = "spidev";
            spi-max-frequency = <20000000>;
            reg = <0>;
            status = "okay";

        };
};
```

spidev 对应的驱动配置的名字为“spidev”，

一般情况下驱动多放置在 drviers 路径下。在此路径下查找 grep "spidev" ./ -nr 查找，如下：（如果，在 drivers 下没有找到，可以在 linux3.14.38 主目录下全局搜索）

```
./spi/spidev.c:655:             .name =         "spidev",
```

此处搜索结果比较多，注意区分。

先查看 drives/spi/Makefile 文件。此文件将 spi 路径下的驱动文件和配置文件中具体哪个配置联系起来，spidev.c 文件编译之后为 spidev.o 文件。

```
obj-$(CONFIG_SPI_SPIDEV)        += spidev.o
```

源码中已经有此驱动，需修改配置文件，将驱动进行编译。修改 6ul emmc 的配置文件 linux_imx6ul_emmc_defconfig，将 spi_spidev 修改为 y 编译进内核。如图：

```
#
# SPI Protocol Masters
#
CONFIG_SPI_SPIDEV=y
# CONFIG_SPI_TLE62X0 is not set
# CONFIG_HSI is not set
```

至此 defconfig 文件已经配置完。

4. 需要在设备树中，将所用引脚定义。

修改设备树文件 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts。出厂的源码中将 csi 引脚用作摄像头。将复用功能去掉或者 disabled。如下图：

```
&csi {
    status = "disabled";

    port {
        csi1_ep: endpoint {
            remote-endpoint = <&ov9650_ep>;
        };
    };
};
```

```
ov9650: ov9650@30 {
    compatible = "ovti,ov9650";
    reg = <0x30>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_csi1>;
    clocks = <&clks IMX6UL_CLK_CSI>;
    clock-names = "csi_mclk";
    pwn-gpios = <&gpio_spi 6 1>;
    rst-gpios = <&gpio_spi 5 0>;
    csi_id = <0>;
    mclk = <24000000>;
    mclk_source = <0>;
    status = "disabled";
    port {
        ov9650_ep: endpoint {
            remote-endpoint = <&csi1_ep>;
        };
    };
};
```

其中 csi 引脚也可复用为 sim2 也将其改为 disabled。

```
&sim2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_sim2_1>;
    assigned-clocks = <&clks IMX6UL_CLK_SIM_SEL>;
    assigned-clock-parents = <&clks IMX6UL_CLK_SIM_PODF>;
    assigned-clock-rates = <240000000>;
    pinctrl-assert-gpios = <&gpio4 23 GPIO_ACTIVE_LOW>;
    port = <1>;
    sven_low_active;
    status = "disabled";
};
```

添加 ecspi1 设备如下：

```
&ecspi1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecspi1>;
        fsl,spi-num-chipselects = <1>;
        cs-gpios = <&gpio4 26 GPIO_ACTIVE_LOW>;
        status = "okay";

        spidev@0{
            compatible = "spidev";
            spi-max-frequency = <20000000>;
            reg = <0>;
            status = "okay";

        };
};
```

在&iomuxc 中添加如下

```
    pinctrl_ecspi1: ecspi1grp {
        fsl,pins = <
            MX6UL_PAD_CSI_DATA07__ECSPI1_MISO    0x100b1
            MX6UL_PAD_CSI_DATA06__ECSPI1_MOSI    0x100b1
            MX6UL_PAD_CSI_DATA04__ECSPI1_SCLK    0x100b1
            MX6UL_PAD_CSI_DATA05__ECSPI1_SS0     0x1b0b1
        >;
    };
```

此处可能不知道怎么对应引脚。参考“PINMUX 说明”部分。

5. 按软件手册编译内核和设备树。注意查看编译完内核之后，是否在 drivers/spi/下生成 spi-imx.o spidev.o，如果没生成，查看配置是否出错？生成*.o 文件说明已经编译进内核。

6. 替换烧写工具中的设备树和内核，重新烧写。开机选择刚替换的设备树。

7. 此 ecspi1 驱动加载成功之后，会在 dev 下生成 spidev0.0 节点。

8. 拷贝测试程序 spi_test 到开发板，比如/forlinx 路径下。此处只是进行短接 miso 和 mosi 进行的测试，运行 spi_test –D /dev/spidev0.0

```
root@freescale ~$ ./spidev_test  -D /dev/spidev0.0
spi mode: 0
bits per word: 8
max speed: 500000 Hz (500 KHz)

AA 55 AA 55 AA 55

AA 55 AA 55 AA 55

AA 55 AA 55 AA 55

AA 55 AA 55 AA 55

AA 55 AA 55 AA 55

AA 55 AA 55 AA 55
```

另外 imx6ul 共有 ecspi1、ecspi2、ecspi3、cspi4 其他接口方法类似，此处不再说明。

1.3 SPI 转 CAN 接口

1. 其中 SPI 部分驱动参考“SPI 接口”部分。
2. 首先搜索一下 6UL 是否自带 mcp2515 驱动。

#cd drivers

#find -name "mcp25*"

```
./net/can/mcp251x.c
```

#vi ./net/can/Makefile

```
obj-$(CONFIG_CAN_MCP251X)    += mcp251x.o
```

3. 在配置文件 linux_imx6ul_emmc_defconfig 中设置 CONFIG_CAN_MCP251X=y.
4. 同时需要配置设备树。

```
clocks {
        mcp251x_clock: mcp251x_clock{
         compatible = "fixed-clock";
         #clock-cells = <0>;
         clock-frequency = <8000000>;
        };
};
```

```
regulators {
    compatible = "simple-bus";
    #address-cells = <1>;
    #size-cells = <0>;

    reg_can_3v3: regulator@0 {
        compatible = "regulator-fixed";
        reg = <0>;
        regulator-name = "can-3v3";
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
        gpios = <&gpio_spi 3 GPIO_ACTIVE_LOW>;
        startup-active-us = <20000>;
        enable-active-high;
    };
```

```
};
&ecspi2 {
        compatible = "fsl,imx51-ecspi";
        fsl,spi-num-chipselects = <1>;
        cs-gpios = <&gpio4 22 0>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecspi2>,<&pinctrl_ecspi2_cs>;
        status = "okay";

        can0: mcp2515@0 {
                pinctrl-names = "default";
                compatible = "microchip,mcp2515";
                pinctrl-0 = <&pinctrl_can>;
                cs-gpios = <&gpio4 22 0>;
                reg = <0>;
                status = "okay";
                spi-max-frequency = <1000000>;
                clocks = <&mcp251x_clock>;
                interrupt-parent = <&gpio4>;
                interrupts = <28 0x2>;
                vdd-supply = <&reg_can_3v3>;
                xceiver-supply = <&reg_can_3v3>;

            };
};
```

```
pinctrl_ecspi2: ecspi2grp {
        fsl,pins = <
           MX6UL_PAD_CSI_DATA03__ECSPI2_MISO    0x100b1
           MX6UL_PAD_CSI_DATA02__ECSPI2_MOSI    0x100b1
           MX6UL_PAD_CSI_DATA00__ECSPI2_SCLK    0x100b1
        >;
};
pinctrl_ecspi2_cs: ecspi2_csgrp {
        fsl,pins = <
           MX6UL_PAD_CSI_DATA01__GPIO4_IO22     0x80000000
        >;
};
pinctrl_can: can {
        fsl,pins = <
           MX6UL_PAD_CSI_DATA07__GPIO4_IO28     0x100b1
        >;
};
```

步骤 1

按软件手册编译内核和设备树。注意查看编译完内核之后，是否在 drivers/spi/下生成 spi-imx.o，是否在 drivers/net/can/下生成 mcp251x.o，如果没生成，查看配置是否出错？生成*.o 文件说明已经编译进内核。

步骤 2

替换烧写工具中的设备树和内核，重新烧写。开机选择刚替换的设备树。

步骤 3

此 ecspi2 驱动加载成功之后，cat /sys/bus/spi/devices/spi1.0/modalias 会出现 spi:mcp2515。

步骤 4

查看打印信息是否生成 can0 节点。

1.4 ADC 接口

以将电阻触摸的 4 路触摸用作 ADC 为例。

1. 查看 IMX6ULRM.pdf 手册中 Chapter 4 External Signals and Pin Multiplexing 有:

ADC1_IN0	GPIO1_IO00
ADC1_IN1	GPIO1_IO01
ADC1_IN2	GPIO1_IO02
ADC1_IN3	GPIO1_IO03
ADC1_IN4	GPIO1_IO04
ADC1_IN5	GPIO1_IO05
ADC1_IN6	GPIO1_IO06
ADC1_IN7	GPIO1_IO07
ADC1_IN8	GPIO1_IO08
ADC1_IN9	GPIO1_IO09

采用 gpio1_io01 gpio1_io02 gpio1_io03 gpio1_io04 作为四路 adc。

2. 怎么查找用哪个驱动，采用 config 中的哪个进行配置呢？
3. 查找 adc 的驱动和配置选项。

设备树文件 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts 中有

```
#include <dt-bindings/input/input.h>
#include "imx6ul.dtsi"
```

arch/arm/boot/dts/imx6ul.dtsi 为通用设备树配置文件。可以在此搜索驱动配置时的名字。比如 adc，搜索到如下：

```
adc1: adc@02198000 {
    compatible = "fsl,imx6ul-adc", "fsl,vf610-adc";
    reg = <0x02198000 0x4000>;
    interrupts = <GIC_SPI 100 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&clks IMX6UL_CLK_ADC1>;
    num-channels = <2>;
    clock-names = "adc";
    status = "disabled";
        };
```

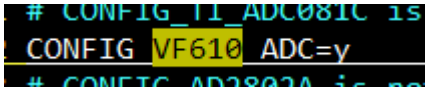
一般情况下驱动多放置在 drviers 路径下。在此路径下查找 grep "fsl,vf610-adc" ./ -nr 查找，如下：（如果，在 drivers 下没有找到，可以在 linux3.14.38 主目录下全局搜索）

```
匹配到二进制文件 ./built-in.o
匹配到二进制文件 ./iio/built-in.o
./iio/adc/vf610_adc.c:544:      { .compatible = "fsl,vf610-adc", },
匹配到二进制文件 ./iio/adc/built-in.o
```

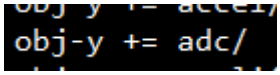
先查看 drives/spi/Makefile 文件。此文件将 adc 路径下的驱动文件和配置文件中具体哪个配置联系起来。vf610_adc.c 文件编译之后为 vf610_adc.o 文件。

```
obj-$(CONFIG_VF610_ADC) += vf610_adc.o
```

查看 6ul emmc 的配置文件 linux_imx6ul_emmc_defconfig 中 CONFIG_VF610_ADC=y。如图：



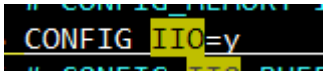
而 adc 在 drivers/iio 下，查看 drivers/iio/Makefile 中，要编译 adc 下的文件，需要有



默认都编译 adc 下文件。

而 iio 在 drivers 下，查看 drivers/Makefile，要编译 iio 下的文件，需要有：

查看 6ul emmc 的配置文件 linux_imx6ul_emmc_defconfig 中 CONFIG_IIO=y。如图：



至此驱动配置完成。

4. 修改设备树文件 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts，添加 adc1.

```
&adc1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_adc1>;
    vref-supply = <&reg_vref_3v3>;
    status = "okay";
};
```

需要用到参考电压，添加 reg_vref_3v3，如图：

```
    reg_usb_otg1_vbus: regulator@2 {
        compatible = "regulator-fixed";
        reg = <2>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_usb_otg1>;
        regulator-name = "usb_otg1_vbus";
        regulator-min-microvolt = <5000000>;
        regulator-max-microvolt = <5000000>;
        gpio = <&gpio3 2 GPIO_ACTIVE_HIGH>;
        enable-active-high;
    };
    reg_vref_3v3: regulator@3 {
        compatible = "regulator-fixed";
        regulator-name = "vref-3v3";
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
    };
};

sound {
```

在&iomuxc 中添加所用到的具体引脚。此处关于上下拉电阻配置部分，参考“PINMUX 说明”部分进行设置。如图：

```
pinctrl_adc1: adc1grp {
        fsl,pins = <
            MX6UL_PAD_GPIO1_IO01__GPIO1_IO01    0xb0
            MX6UL_PAD_GPIO1_IO02__GPIO1_IO02    0xb0
            MX6UL_PAD_GPIO1_IO03__GPIO1_IO03    0xb0
            MX6UL_PAD_GPIO1_IO04__GPIO1_IO04    0xb0
        >;
};
```

并将其他复用 gpio1_io01-4 的地方去掉或者 disabled，如下

```
&tsc {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_tsc>;
    status = "disabled";
    xnur-gpio = <&gpio1 3 0>;
    measure_delay_time = <0xffff>;
    pre_charge_time = <0xfff>;
};
```

5. 编译生成 dtb zImage，编译内核，查看 drivers/iio/adc/是否生成 vf610_adc.o，如果生成，已编译进内核。如果未生成，查看是否配置出错？
6. 替换 dtb zImage，并烧写，启动。
7. 查看开发板/dev 下有节点 iio:device0，则驱动加载成功。

或者进入 cd /sys/bus/iio/devices/iio\:device0/路径查看。

1.5 串口

1.5.1 增加串口

1. 此处以 uart4 配置进行说明。查看 IMX6ULRM.pdf 手册中 Chapter 4 External Signals and Pin Multiplexing 有

UART4	CTS_B	ENET1_RX_DATA1	ALT1
		LCD_HSYNC	ALT2
	RTS_B	ENET1_RX_DATA0	ALT1
		LCD_VSYNC	ALT2
	RX_DATA	LCD_ENABLE	ALT2
		UART4_RX_DATA	ALT0
	TX_DATA	LCD_CLK	ALT2
		UART4_TX_DATA	ALT0

2. 因有调试串口，驱动为同一个，此处不再修改配置 configs 文件。
3. 修改设备树文件 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts。

```
&uart4 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart4>;
    status = "okay";
};
```

```
pinctrl_uart4: uart4grp {
    fsl,pins = <
        MX6UL_PAD_UART4_RX_DATA__UART4_DCE_RX   0x1b0b1
        MX6UL_PAD_UART4_TX_DATA__UART4_DCE_TX   0x1b0b1
    >;
};
```

因 uart4_rx tx 复用为 i2c1，此处将 i2c1 设置为 disabled。

```
&i2c1 {
    clock-frequency = <100000>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_i2c1>;
    status = "disabled";
```

4. 编译生成 dtb，替换并烧写。
5. 查看开发板/dev 下有节点 ttymxc3，则驱动加载成功。

测试同其他串口的测试方法。此处不再说明。

1.5.2 串口去掉 DMA

i.MX6UL 源码中，默认除了 debug 串口 uart1 之外，其它的都是默认打开的 DMA 的，如果串口只是接了 TXD/RXD，而没有硬件流控 RTS/CTS，则使用 DMA 传输大量数据有可能报 DMA 错误，所以如果只接 TXD/RXD 可以只使用 PIO 模式，参考 uart1 设置如下：

arch\arm\boot\dts\imx6ul.dtsi

以 uart7 为例：

```
uart7: serial@02018000 {
    compatible = "fsl,imx6ul-uart",
    "fsl,imx6q-uart", "fsl,imx21-uart";
…
    //去掉 dma 支持   dmas = <&sdma 43 4 0>, <&sdma 44 4 0>;
```

1.6 调试 LCD 的分辨率

1.6.1 内核调试 lcd 参数

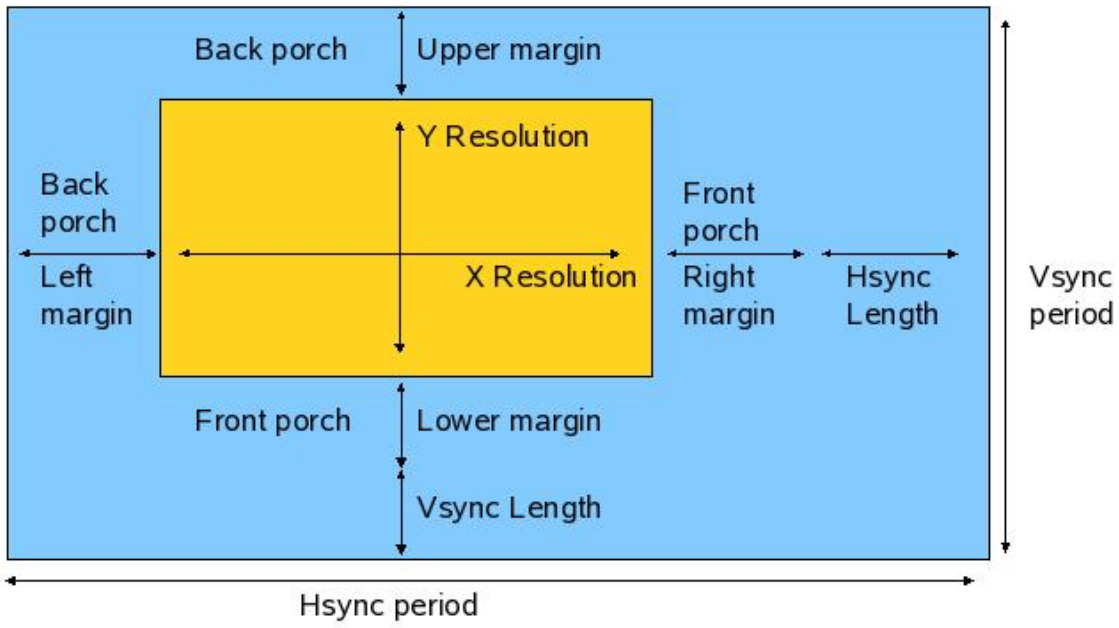
1. 内核设备树。以修改 4.3 吋为 3.5 吋屏。打开内核 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts 找到&lcdif。

```
timing0: timing0 {
    clock-frequency = <9200000>;
    hactive = <480>;
    vactive = <272>;
    hfront-porch = <8>;
    hback-porch = <4>;
    hsync-len = <41>;
    vback-porch = <2>;
    vfront-porch = <4>;
    vsync-len = <10>;

    hsync-active = <0>;
    vsync-active = <0>;
    de-active = <1>;
    pixelclk-active = <0>;
    };*/                                    /*4.3*/
```

2. 参考屏体手册中有：

Signal	Item	Symbol	Min	Typ	Max	Unit
Dclk	Frequency	Tosc	-	156	-	ns
	High Time	Tch	-	78	-	ns
	Low Time	Tcl	-	78	-	ns
Data	Setup Time	Tsu	12	-	-	ns
	Hold Time	Thd	12	-	-	ns
Hsync	Period	TH	-	408	-	Tosc
	Pulse Width	THS	5	30	-	Tosc
	Back-Porch	Thb		38		Tosc
	Display Period	TEP	-	320	-	Tosc
	Hsync-den time	THE	36	68	88	Tsoc
	Front-Porch	Thf	-	20	-	Tosc
Vsync	Period	Tv	-	262	-	TH
	Pulse Width	Tvs	1	3	5	TH
	Back-Porch	Tvb	-	15	-	TH
	Display Period	Tvd	-	240	-	TH
	Front-Porch	Tvf	2	4	-	TH



3. 修改设备树中

其中 clock-frequency= fframe*(hfront+hback+hsync+xres)*(vfront+vback+vsync+yres) 其中 fframe=60

```
&lcdif {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_lcdif_dat
             &pinctrl_lcdif_ctrl>;
    display = <&display0>;
    status = "okay";

    display0: display {
        bits-per-pixel = <16>;
        bus-width = <24>;

        display-timings {
            native-mode = <&timing0>;
        timing0: timing0 {
            clock-frequency = <6410256>;
            hactive = <320>;
            vactive = <240>;
            hfront-porch = <20>;
            hback-porch = <38>;
            hsync-len = <30>;
            vback-porch = <15>;
            vfront-porch = <4>;
            vsync-len = <3>;

            hsync-active = <0>;
            vsync-active = <0>;
            de-active = <1>;
            pixelclk-active = <0>;
            };                                  /*4.3*/
        };
    };
};
```

4. 编译 dtb 文件。#make dtbs。生成 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts. 替换烧写工具中的 dtb 中文件。烧写。在 uboot 选择 5-4.3 吋屏。重启。发现 uboot 显示不正常，内核显示正常。

5. 如果发现屏幕闪烁，调整时钟。或者查看硬件。

```
clock-frequency = <6413760>;
```

或未在中心位置。微调下面 6 个参数。

```
hfront-porch = <20>;
hback-porch = <38>;
hsync-len = <30>;
vback-porch = <15>;
vfront-porch = <4>;
vsync-len = <3>;
```

1.6.2 uboot 调整屏幕参数

注：uboot 不开源，此处可跳过

1. 修改 board/freescale/mx6ul_14x14_evk/mx6ul_14x14_evk.c 中

其中 pixclock = 10^{12} / clock_frequency

```
static struct lcd_panel_info_t const displays[] = {
        {
                .lcdif_base_addr = LCDIF1_BASE_ADDR,
                .depth = 24,
                .enable = do_enable_parallel_lcd,
                .mode   = {
                        .name           = "TFT43AB",
                        .xres           = 320,
                        .yres           = 240,
                        .pixclock       = 156000,
                        .left_margin    = 38,
                        .right_margin   = 20,
                        .upper_margin   = 15,
                        .lower_margin   = 4,
                        .hsync_len      = 30,
                        .vsync_len      = 3,
                        .sync           = 0,
                        .vmode          = FB_VMODE_NONINTERLACED
                }
        },
```

2. 编译生成 u-boot.imx。替换烧写工具中对应核心板 uboot 烧写。选择 5-4.3 吋屏。
3. 如果偏离中心位置，调整参数。闪烁查看时钟或者硬件。

   如果不调试 uboot，只修改内核，uboot 阶段，显示不正确。

1.6.3 文件系统的修改

修改/etc/rc.d/qt_env.sh，根据实际需求调整 QWS_SIZE 的大小。

```
32 #export QWS_MOUSE_PROTO=mouseman:/dev/input/mice
33 export QWS_MOUSE_PROTO=tslib:/dev/input/event0
34 export QWS_DISPLAY="linuxfb:mmWidth50:mmHeight130:0"
35 #export QWS_SIZE=800x480
36 if [ $FB_SIZE = "800,1200" ]; then
37 export QWS_SIZE=800x600
38 elif [ $FB_SIZE = "1920,1440" ]; then
39 export QWS_SIZE=1280x720
40 elif [ $FB_SIZE = "1024,1536" ]; then
41 export QWS_SIZE=1024x768
42 elif [ $FB_SIZE = "1280,1600" ]; then
43 export QWS_SIZE=1280x800
44 else
45 export QWS_SIZE=800x480
46 fi
```

1.7 复用 GPIO

1.7.1 采用 iomux 的方法

1. 现采用 CSI_DATA01 用作 GPIO。
2. 首先在 arch/arm/boot/dts/imx6ul-pinfunc.h 查找

```
#define MX6UL_PAD_CSI_DATA01__EIM_AD01                  0x01E8 0x0474 0x0000 4 0
#define MX6UL_PAD_CSI_DATA01__GPIO4_IO22                0x01E8 0x0474 0x0000 5 0
#define MX6UL_PAD_CSI_DATA01__SAI1_MCLK                 0x01E8 0x0474 0x0000 6 0
```

此处的具体参数不再说明，查看 IMX6ULRM.pdf 手册中对应寄存器。

3. 在设备树中 iomux 中添加复用 gpio 引脚如下：

```
&iomuxc {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_hog_1>;
    imx6ul-evk {
        pinctrl_hog_1: hoggrp-1 {
            fsl,pins = <
                MX6UL_PAD_LCD_RESET__WDOG1_WDOG_ANY     0x30b0
                MX6UL_PAD_UART1_RTS_B__GPIO1_IO19    0x17059 /* SD1 CD */
                MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT     0x17059 /* SD1 VSELECT */
                MX6UL_PAD_SNVS_TAMPER1__GPIO5_IO01  0x80000000 /*ACC INT*/
                MX6UL_PAD_CSI_DATA01__GPIO4_IO22     0x1f0b1
            >;
        };
```

同时修改设备树文件中，出厂的源码中将 csi 引脚用作摄像头。将复用功能去掉或者 disabled。如下图：

```
&csi {
    status = "disabled";

    port {
        csi1_ep: endpoint {
            remote-endpoint = <&ov9650_ep>;
        };
    };
};
```

```
ov9650: ov9650@30 {
    compatible = "ovti,ov9650";
    reg = <0x30>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_csi1>;
    clocks = <&clks IMX6UL_CLK_CSI>;
    clock-names = "csi_mclk";
    pwn-gpios = <&gpio_spi 6 1>;
    rst-gpios = <&gpio_spi 5 0>;
    csi_id = <0>;
    mclk = <24000000>;
    mclk_source = <0>;
    status = "disabled";
    port {
        ov9650_ep: endpoint {
            remote-endpoint = <&csi1_ep>;
        };
    };
};
```

其中 csi 引脚也可复用为 sim2.也将其改为 disabled。

```
&sim2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_sim2_1>;
    assigned-clocks = <&clks IMX6UL_CLK_SIM_SEL>;
    assigned-clock-parents = <&clks IMX6UL_CLK_SIM_PODF>;
    assigned-clock-rates = <240000000>;
    pinctrl-assert-gpios = <&gpio4 23 GPIO_ACTIVE_LOW>;
    port = <1>;
    sven_low_active;
    status = "disabled";
};
```

编译设备树。替换设备树，并重新烧写。

4. 开机选择刚替换的设备树。

此时可以使用 echo 命令进行控制：

以 GPIO4_IO22 为例进行命令说明。

4.1 GPIO 设置，步骤如下：

a 计算对应 sys/class/gpio 的值 GPIOn_IOx = (n-1)*32 + x

GPIO4_IO22=（4-1）*32+22=118

b 将 GPIO4_IO22 设置为输出。

echo 118 > /sys/class/gpio/export 用于通知系统需要导出控制的 GPIO 引脚编号

echo "out" > /sys/class/gpio/gpio118/direction 控制为输出

echo "1" > /sys/class/gpio/gpio118/value 输出为高电平

或者 echo "0" > /sys/class/gpio/gpio118/value 输出为低电平

echo 118 > /sys/class/gpio/unexport 通知系统取消导出

c 将 GPIO4_IO22 设置为输入。

echo 118 > /sys/class/gpio/export 用于通知系统需要导出控制的 GPIO 引脚编号

echo "in" > /sys/class/gpio/gpio118/direction 控制为输入

这时给该引脚接高电平，输入即为高电平，反之为低电平

echo 118 > /sys/class/gpio/unexport 通知系统取消导出

d 另外客户可以自己通过 shell 文件来控制多个 gpio 做为输入或者输出。

4.2 GPIO 输出测试

编写测试脚本 vi gpiotest_o.sh

```
#!/bin/bash
# gpio list gpio (bank-1)*32 + nr
for test in 118 119 120 137 136 12
do
echo Exporting pin $test.
echo $test> /sys/class/gpio/export
echo Setting pin $1.
echo  out > /sys/class/gpio/gpio$test/direction
echo  $1 > /sys/class/gpio/gpio$test/value
echo $test> /sys/class/gpio/unexport
done
echo complete
```

修改脚本执行权限：chmod u+x gpiotest_o.sh
测试 gpio 输出为低。进入到脚本所在路径：./gpiotest_o.sh 0
所有 GPIO 输出低电平 0V。
测试 gpio 输出为高电平。进入到脚本所在路径：./gpiotest_o.sh 1
所有 GPIO 输出高电平。输出的高电平，根据引脚所在的电源域不同，可能会有区别。

另外有些客户发现
echo 118 > /sys/class/gpio/export 用于通知系统需要导出控制的 GPIO 引脚编号
echo "out" > /sys/class/gpio/gpio118/direction 控制为输出
echo "1" > /sys/class/gpio/gpio118/value 输出为高电平
cat /sys/class/gpio/gpio118/value 仍旧为 0
原因如下图所示，客户可以从 CPU 手册中查找到相关内容：

27.4.3.1 GPIO Read Mode

The programming sequence for reading input signals should be as follows:

1. Configure IOMUX to select GPIO mode (Via IOMUX Controller (IOMUXC)).
2. Configure GPIO direction register to input (GPIO_GDIR[GDIR] set to 0b).
3. Read value from data reg*ister*/pad status register.

A pseudocode description to read [input3:input0] values is as follows:

```
// SET INPUTS TO GPIO MODE.
write sw_mux_ctl_<input0>_<input1>_<input2>_<input3>, 32'h00000000
// SET GDIR TO INPUT.
write GDIR[31:4,input3_bit, input2_bit, input1_bit, input0_bit,] 32'hxxxxxxx0
// READ INPUT VALUE FROM DR.
read DR
// READ INPUT VALUE FROM PSR.
read PSR
```

NOTE

While the GPIO direction is set to input (GPIO_GDIR = 0), a read access to GPIO_DR does not return GPIO_DR data. Instead, it returns the GPIO_PSR data, which is the corresponding input signal value.

输入模式读取的是 psr 的值。

27.4.3.2 GPIO Write Mode

The programming sequence for driving output signals should be as follows:

1. Configure IOMUX to select GPIO mode (Via IOMUXC), also enable SION if need to read loopback pad value through PSR
2. Configure GPIO direction register to output (GPIO_GDIR[GDIR] set to 1b).
3. Write value to data register (GPIO_DR).

A pseudocode description to drive 4'b0101 on [output3:output0] is as follows:

```
// SET PADS TO GPIO MODE VIA IOMUX.
write sw_mux_ctl_pad_<output[0-3]>.mux_mode, <GPIO_MUX_MODE>
// Enable loopback so we can capture pad value into PSR in output mode
write sw_mux_ctl_pad_<output[0-3]>.sion, 1
// SET GDIR=1 TO OUTPUT BITS.
write GDIR[31:4,output3_bit,output2_bit, output1_bit, output0_bit,] 32'hxxxxxxxF
// WRITE OUTPUT VALUE=4'b0101 TO DR.
write DR, 32'hxxxxxxx5
// READ OUTPUT VALUE FROM PSR ONLY.
read_cmp PSR, 32'hxxxxxxx5
```

读取 output 的 value 值是从 PSR 中读取的。而写入 output 值是写入到 DR 中的。可以通过设置 SION 位回环。

4.3 GPIO 输入测试

编写测试脚本 vi gpiotest_i.sh

```
#!/bin/bash
# gpio list gpio (bank-1)*32 + nr
for test in 118 119 120 137 136 12
do
echo Exporting pin $test.
echo $test> /sys/class/gpio/export
echo    in > /sys/class/gpio/gpio$test/direction
gpioval=`cat    /sys/class/gpio/gpio$test/value`
echo GPIO $test = $gpioval
echo
echo $test> /sys/class/gpio/unexport
done
echo complete
```

修改脚本执行权限：chmod u+x gpiotest_i.sh
测试 gpio 输入为低。进入到脚本所在路径：./gpiotest_i.sh
所有 GPIO 输入为 0。
测试 gpio 输入为高电平，比如 5v。进入到脚本所在路径：./gpiotest_i.sh
所有 GPIO 输入为 1。
\iomuxc\shell\di\in-test.sh，复制到 forlinx（比如）目录下./in-test.sh 118
\iomuxc\shell\do\close.sh，复制到 forlinx（比如）目录下./close.sh 118
\iomuxc\shell\do\open.sh，复制到 forlinx（比如）目录下./open.sh 118
或者采用\iomuxc\write-117-out-high\test，复制到 forlinx（比如）目录下./test 将 gpio 117 输出为高。

1.7.2 创建 dev/gpio 节点。

1. 在设备树文件中添加设备节点定义以及其引脚定义：

```
gpios {
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_user>;
compatible = "gpio-user";
gpio0{
        label = "DO1";
        gpios = <&gpio5 9 1>;
        default-direction = "out";
};
gpio1{
        label = "DO2";
        gpios = <&gpio1 9 1>;
        default-direction = "out";
};
};
```

```
&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog_1>;
        imx6ul-evk {
                pinctrl_hog_1: hoggrp-1 {
                        fsl,pins = <
                                MX6UL_PAD_LCD_RESET__WDOG1_WDOG_ANY     0x30b0
                                MX6UL_PAD_UART1_RTS_B__GPIO1_IO19       0x17059 /* SD1 CD */
                                MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT    0x17059 /* SD1 VSELECT */
                                MX6UL_PAD_SNVS_TAMPER1__GPIO5_IO01      0x80000000 /*ACC INT*/
                        >;
                };
                pinctrl_user: usergrp {
                        fsl,pins = <
                                MX6UL_PAD_SNVS_TAMPER9__GPIO5_IO09      0x3008
                                MX6UL_PAD_GPIO1_IO09__GPIO1_IO09        0x3008
                        >;
                };
```

并将其他复用引脚对应的功能 disabed，保证这些引脚没被重复定义使用。引脚的 pinmux 可以查看 imx6ul-pinfunc.h 文件。

2. driver/misc/gpio 目录下添加 gpio 驱动 gpio-user.c，名字需要与节点定义里的驱动名字保相同，客户也可以自己写驱动。同时添加 Kconfig 和 Makefile 文件。

修改 driver/misc 下 Kconfig 和 Makefile 文件：

在 driver/misc/Makefile 中添加：

obj-y += gpio/

编辑 driver/misc/Kconfig，添加一行：

source "drivers/misc/gpio/Kconfig"，如图：

```
source "drivers/misc/c2port/Kconfig"
source "drivers/misc/eeprom/Kconfig"
source "drivers/misc/cb710/Kconfig"
source "drivers/misc/ti-st/Kconfig"
source "drivers/misc/lis3lv02d/Kconfig"
source "drivers/misc/carma/Kconfig"
source "drivers/misc/altera-stapl/Kconfig"
source "drivers/misc/mei/Kconfig"
source "drivers/misc/vmw_vmci/Kconfig"
source "drivers/misc/mic/Kconfig"
source "drivers/misc/genwqe/Kconfig"
source "drivers/misc/gpio/Kconfig"
endmenu
```

在根目录下修改 linux_imx6ul_config 文件，添加：

CONFIG_GPIO_USER_INTF=y

3. 编译。

make zImage

make ARCH=arm CROSS_COMPILE=arm-fsl-linux-gnueabi- dtbs

查看 driver/misc/gpio 下生成 gpio-user.o，说明 gpio-user.c 已编译进内核。

4. 把前面生成的 zImage, imx6ul-14x14-evk.dtb 替换 SD 卡 system 目录中相应文件，SD 卡方式烧录。

烧录完后，启动开发板，在 dev 下有 gpio 节点。

5. 用 gpio-test.c 为用户测试程序。编译为 gpio-test。

使用 gpio-test in 2 测试 DI。此处假设 gpios 中的 gpio2 为 di 输入。

使用 gpio-test out 0 1 测试 gpios 中的 gpio0 也就是 DO1 输出为高电平。

使用 gpio-test out 0 0 测试 gpios 中的 gpio0 也就是 DO1 输出为低电平。

1.8 USB 转串口

内核自带了 PL2303 的驱动，需要将配置文件 linux_imx6ul_emmc_defconfig 或者 linux_imx6ul_nand_config 中 CONFIG_USB_SERIAL_PL2303 设置为 y。

```
CONFIG_USB_SERIAL_PL2303=y
```

编译内核，烧写并替换内核。

启动文件系统，插入 usb 转串口，如下：

```
root@freescale ~$ [   17.334618] usb 1-1.2: new full-speed USB device number 4 using ci_hdrc
[   17.477775] pl2303 1-1.2:1.0: pl2303 converter detected
[   17.503409] usb 1-1.2: pl2303 converter now attached to ttyUSB0
```

在 dev 下产生 ttyUSB0 节点。

```
root@freescale ~$ ls -la /dev/ttyUSB0
crw-rw----    1 root     uucp      188,   0 Jan 13 09:22 /dev/ttyUSB0
```

另外有些 USB 设备需要将配置文件中 CONFIG_USB_PRINTER=y，需要注意。

测试方法同 UART 串口测试章节。

1.9 NAND 增加分区

以 256m nand 核心板为例,修改 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts 这个文件中，

```
&gpmi {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_nand>;
    status = "disabled";
    partition@0{
            label = "boot";
            reg = <0x0000 0x400000>; /*4M*/
    };
    partition@1{
            label = "logo";
            reg = <0x400000 0x200000>; /*2M*/
    };
    partition@2{
            label = "ENV";
            reg = <0x600000 0x100000>; /*1M*/
    };
    partition@3{
            label = "DTB";
            reg = <0x700000 0x300000>; /*3M*/
    };
    partition@4{
            label = "kernel";
            reg = <0xa00000 0x800000>; /*8M*/
    };
    partition@5{
            label = "rootfs";
            reg = <0x1200000 0xce00000>;
    };
    partition@6{
            label = "nand2";
            reg = <0xe000000 0x1000000>; /* 16M */
    };

    partition@7{
            label = "nand3";
            reg = <0xf000000 0x1000000>; /* 16M */
    };
};
```

编译 make dtbs

```
  DTC     arch/arm/boot/dts/imx6ul-14x14-evk.dtb
  DTC     arch/arm/boot/dts/imx6ul-14x14-evk-7-r.dtb
  DTC     arch/arm/boot/dts/imx6ul-14x14-evk-csi.dtb
  DTC     arch/arm/boot/dts/imx6ul-14x14-evk-gpmi.dtb
  DTC     arch/arm/boot/dts/imx6ul-14x14-evk-gpmi-csi.dtb
  DTC     arch/arm/boot/dts/imx6ul-14x14-evk-gpmi-7-r.dtb
```

arch/arm/boot/dts/imx6ul-14x14-evk-gpmi-c-256m-*.dtb 替换烧写工具中设备树，进行烧写。启动选择对应设备树。

```
freescale login: root
root@freescale ~$ ls -la /dev/mtdblock*
brw-r-----    1 root     disk       31,   0 Jan 15 05:14 /dev/mtdblock0
brw-r-----    1 root     disk       31,   1 Jan 15 05:14 /dev/mtdblock1
brw-r-----    1 root     disk       31,   2 Jan 15 05:14 /dev/mtdblock2
brw-r-----    1 root     disk       31,   3 Jan 15 05:14 /dev/mtdblock3
brw-r-----    1 root     disk       31,   4 Jan 15 05:14 /dev/mtdblock4
brw-r-----    1 root     disk       31,   5 Jan 15 05:14 /dev/mtdblock5
brw-r-----    1 root     disk       31,   6 Jan 15 05:14 /dev/mtdblock6
brw-r-----    1 root     disk       31,   7 Jan 15 05:14 /dev/mtdblock7
root@freescale ~$ cat /proc/m
meminfo  misc     modules  mounts   mtd
root@freescale ~$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00400000 00020000 "boot"
mtd1: 00200000 00020000 "logo"
mtd2: 00100000 00020000 "ENV"
mtd3: 00300000 00020000 "DTB"
mtd4: 00800000 00020000 "kernel"
mtd5: 0ce00000 00020000 "rootfs"
mtd6: 01000000 00020000 "nand2"
mtd7: 01000000 00020000 "nand3"
```

在 mnt 下创建文件夹 nand2 nand3

挂载：

mount /dev/mtdblock6 /mnt/nand2

mount /dev/mtdblock7 /mnt/nand3

```
root@freescale /$ mount /dev/mtdblock6  /mnt/nand2
[  484.399723] yaffs: dev is 32505862 name is "mtdblock6" rw
[  484.405741] yaffs: passed flags ""
root@freescale /$
```

```
root@freescale /mnt$ cd ../
root@freescale /$ mount /dev/mtdblock7   /mnt/nand3
[  536.559731] yaffs: dev is 32505863 name is "mtdblock7" rw
[  536.566610] yaffs: passed flags ""
root@freescale /$
```

1.10 增加 PWM3

CPU 默认可输出 8 路 PWM，默认背光采用的为 PWM1，增加 GPIO1_IO04 复用为 PWM3，并输出 2.4K 的方波。

驱动默认已经加载，驱动路径为 drivers/pwm/pwm-imx.c。

修改设备树，修改 imx6ul.dtsi 中

```
pwm3: pwm@02088000 {
        compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
                reg = <0x02088000 0x4000>;
                interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks IMX6UL_CLK_DUMMY>,
                         <&clks IMX6UL_CLK_DUMMY>;
                clock-names = "ipg", "per";
                #pwm-cells = <2>;
        };
```

为

```
pwm3: pwm@02088000 {
                compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
                reg = <0x02088000 0x4000>;
```

```
                interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks IMX6UL_CLK_PWM3>,
                         <&clks IMX6UL_CLK_PWM3>;
            clock-names = "ipg", "per";
            #pwm-cells = <2>;
        };
```

在 imx6ul-14x14-evk-emmc-c-7-800x480.dts 中添加：

```
&pwm3 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_pwm3>;
        status = "okay";
};
添加 pinctl_pwm3
  pinctrl_pwm3: pwm3grp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO04__PWM3_OUT     0x110b0
                >;
            };
```

并将GPIO1_IO04其他复用的地方取消或者注释掉。

```
tsc中status = “disabled”;
```

编译设备树，替换开发板中设备树，并选择此设备树。

在开发板启动之后，命令行输入：

```
root@freescale ~$ echo 0 > /sys/class/pwm/pwmchip2/export
root@freescale ~$ echo 1 >/sys/class/pwm/pwmchip2/pwm0/enable
root@freescale ~$ echo 416667 > /sys/class/pwm/pwmchip2/pwm0/period
root@freescale ~$ echo 208333 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle
```

此时 GPIO1_IO04 输出 2.4K 方波。

如果输出 1KHz 方波，命令如下：

```
root@freescale ~$echo 1000000 > /sys/class/pwm/pwmchip2/pwm0/period
root@freescale ~$echo 500000 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle
```

1.11 裁剪启动时间

1. uboot

   在 bootargs 中加入 lpj quiet 参数，去掉 uboot 3s 延时。

2. 内核和文件系统裁剪

   总的原则是将不用的驱动和文件系统中不用的库文件去掉。参考裁剪启动时间文件夹。

1.12 LCD 转 LVDS 模块

关闭电源，将 LCD 屏幕接至 LVDS 接口，现阶段支持深圳拓普微的 LMT070DICFWD-AKA 液晶显示器。

上电即可正常显示，触摸可用，如有需要可联系销售人员。

1.13 LCD 转 VGA 模块

关闭电源，将 LCD 屏幕接至 LCD 转 VGA 模块。需要更改显示的各参数，客户根据实际使用的设备树修改参数即可。如有需要可联系销售人员。

1.14 QT 显示汉字

要在 Qt 的应用程序中显示汉字，

1.汉字库文件，如“宋体”simsun.ttc，如果 Qt 找不到汉字库，或者要显示的汉字不再 Qt 能找到的汉字库中，可以下载字库文件，直接拷贝到 Qt 的 lib/fonts 目录下；

2.QT 程序支持汉字显示。

1.15 命令行显示汉字

有些客户需要在命令行，查看带汉字的文件或者文件夹，参考命令行显示中文文件夹中修改方法。

1.16 wm8960 line-in

有些客户对 line-in 接口有需要，需要硬件支持，如下修改测试：

1.修改设备树，添加所用接口。比如采用 LINPUT3 和 RINPUT3。

```
              "Headset Jack", "HP_R",
@@ -120,16 +120,14 @@
              "Ext Spk", "SPK_LN",
              "Ext Spk", "SPK_RP",
              "Ext Spk", "SPK_RN",
-             "LINPUT2", "Hp MIC",
-             "LINPUT3", "Hp MIC",
-             "RINPUT1", "Main MIC",
-             "RINPUT2", "Main MIC",
+             "LINPUT3", "Main MIC",
+             "RINPUT3", "Main MIC",
              "Hp MIC", "MICB",
              "Main MIC", "MICB",
```

2.编译并选择所用设备树。系统启动之后在输入命令测试。

设置参数：

```
amixer cset name='Left Output Mixer PCM Playback Switch',1
amixer cset name='Right Output Mixer PCM Playback Switch',1
amixer cset name='Left Boost Mixer LINPUT3 Switch' on
amixer cset name='Left Input Mixer Boost Switch' on
```

```
amixer cset name='Right Boost Mixer RINPUT3 Switch' on
amixer cset name='Right Input Mixer Boost Switch' on
amixer cset name='Capture Volume' 23,23
amixer cset name='Capture Volume ZC Switch' off
amixer cset name='Capture Switch' off
amixer cset name='ADC PCM Capture Volume' 195,195
amixer cset name="Playback Volume" 255,255
amixer cset name="Speaker Playback Volume" 127,127
amixer cset name="Headphone Playback Volume" 127,127
```

录音：

```
arecord -r 44100 -f S16_LE -c 2 -d 10 record.wav
```

可以将录音文件放到电脑播放。

1.17 串口蓝牙

串口蓝牙可采用 CC2540 模块，采用透传模式。

1.18 FTP 限定用户访问特定路径

需要移植 vsftpd，移植方法参考用户资料\应用\ftp 限定在特定路径文件夹。

另外需要注意将默认的 ftp 关闭，修改/etc/inetd.conf 文件

```
ftp     stream  tcp     nowait  root    /usr/sbin/ftpd  ftpd –wS
```

修改为

```
# ftp     stream  tcp     nowait  root    /usr/sbin/ftpd  ftpd –wS
```

可如下查看端口：

```
root@freescale /etc$ netstat -ntpl | grep vsftpd
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      889/vsftpd
netstat: /proc/net/tcp6: No such file or directory
root@freescale /etc$
```

1.19 制作开机 LOGO 图片

各个屏幕的分辨率分别如下：

4——480*272

5.6——600*480

7——800*480

8——800*600

10.4——800*600

现在以 7 寸屏为例。

1、先制作出 logo.jpg，注意大小要和 7 寸屏大小（800*480）相同，否则图片位置和效果可能不佳。

2、ubuntu 系统下载图形转换工具

#sudo apt-get install netpbm

制作适用的图片

创建工作目录

```
#mkdir uboot-logo
#cd uboot-logo
```

将图片拷贝到 uboot-logo 目录下
#cp logo.jpg .
编写图片转换脚本
#vi mkbmp.sh
内容如下：
#!/bin/sh
jpegtopnm $1 | ppmquant 31 | ppmtobmp -bpp 8 > $2
#chmod +x mkbmp.sh
利用脚本转换成适合 uboot 的图片
#./mkbmp.sh logo.jpg logo.bmp
可以看到生成了 logo.bmp 图片。
至此，可以用的图片制作完成。
按照烧写过程，可将图片重新命名为 logo.bmp 烧写到 emmc 或者 nand 指定地址上。

1.20 支持 Python2.7.9

光盘资料 linux/镜像/filesystem 中的 rootfs-console.tar.bz2 的文件系统镜像默认部署了 Python2.7.9。
关于 Python2.7.9 的使用示例简介如下：
1.进入 Linux OS，输入 python 回车启动 python

```
root@freescale ~$ python
Python 2.7.9 (default, Dec  4 2017, 15:59:28)
[GCC 4.6.2 20110630 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2.输入 python 命令如下

```
>>> import sys
>>> sys.version
'2.7.9 (default, Dec  4 2017, 15:59:28) \n[GCC 4.6.2 20110630 (prerelease)]'
>>> sys.stdout.write('hello python!\n')
hello python!
>>>
```

按 ctrl+d 退出 python。
3.通过 vi 编辑器编写一个简单的 python 脚本
vi test.py

```
#! /usr/local/python2.7.9/bin python
import sys
sys.stdout.write('hello world\n')
~
```

保存退出 vi 编辑器，修改 test.py 为可执行权限，通过命令行执行：
root@freescale ~$ python test.py
输出如下：

```
root@freescale ~$ python test.py
hello world
```

1.21 kernel 与 dtb 备份

Okmx6ul-c/c2 平台的 1g nand flash、emmc flash 对 kernel 和 dtb 进行了备份。

1.针对 emmc:

```
root@freescale ~$ ls -l /media/mmcblk1p1
total 14344
-rwxr-xr-x    1 root     root         38367 Dec  9 15:07 imx6ul-14x14-evk-10.4-r.dtb
-rwxr-xr-x    1 root     root         37610 Dec  9 15:07 imx6ul-14x14-evk-4.3-r.dtb
-rwxr-xr-x    1 root     root         38460 Dec  9 15:07 imx6ul-14x14-evk-5.6-r.dtb
-rwxr-xr-x    1 root     root         38342 Dec  9 15:07 imx6ul-14x14-evk-7.dtb
-rwxr-xr-x    1 root     root         38363 Dec  9 15:07 imx6ul-14x14-evk-8-r.dtb
-rwxr-xr-x    1 root     root         38342 Dec  9 15:07 imx6ul-14x14-evk.dtb
-rwxr-xr-x    1 root     root        391736 Dec  9 15:07 logo-4.3.bmp
-rwxr-xr-x    1 root     root        308278 Dec  9 15:07 logo-5.6.bmp
-rwxr-xr-x    1 root     root        385078 Dec  9 15:07 logo-7.bmp
-rwxr-xr-x    1 root     root        481078 Dec  9 15:07 logo-8.bmp
-rwxr-xr-x    1 root     root       6424664 Dec  9 15:07 zImage
-rwxr-xr-x    1 root     root       6424664 Dec  9 15:07 zImagebak
```

其中的 zImagebak 与 imx6ul-14x14-evk.dtb 为备份的 kernel 和 dtb，备份的 dtb 支持 7 吋 LCD 显示。

删除第一映像 zImage

```
root@freescale ~$ rm -rf /media/mmcblk1p1/zImage
root@freescale ~$ sync
```

重启系统，u-boot 阶段读备份的 kernel 到 ram 启动 Linux os 的部分 log 信息如下

```
Normal Boot
Hit any key to stop autoboot:  0
reading boot.scr
** Unable to read file boot.scr **
mmc boot.........
reading zImage
** Unable to read file zImage **
reading zImagebak
6424664 bytes read in 162 ms (37.8 MiB/s)
Booting from mmc ...
reading imx6ul-14x14-evk-7.dtb
38342 bytes read in 18 ms (2 MiB/s)
Kernel image @ 0x80800000 [ 0x000000 - 0x620858 ]
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Using Device Tree in place at 83000000, end 8300c5c5

Starting kernel ...
```

删除系统启动时默认加载的 imx6ul-14x14-evk-emmc-c-7-800x480.dtb

```
root@freescale ~$ rm -rf /media/mmcblk1p1/imx6ul-14x14-evk-emmc-c-7-800x480.dtb
root@freescale ~$ sync
```

重启系统，u-boot 阶段读备份的 dtb: imx6ul-14x14-evk-emmc-c-bak.dtb 到 ram 启动 Linux os 部分 log 信息如下

```
Normal Boot
Hit any key to stop autoboot:  0
reading boot.scr
** Unable to read file boot.scr **
mmc boot.........
reading zImage
** Unable to read file zImage **
reading zImagebak
6424664 bytes read in 162 ms (37.8 MiB/s)
Booting from mmc ...
reading imx6ul-14x14-evk-7.dtb
** Unable to read file imx6ul-14x14-evk-7.dtb **
reading imx6ul-14x14-evk.dtb
38342 bytes read in 18 ms (2 MiB/s)
Kernel image @ 0x80800000 [ 0x000000 - 0x620858 ]
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Using Device Tree in place at 83000000, end 8300c5c5

Starting kernel ...
```

针对 1g nand flash

```
root@freescale ~$ cat /proc/mtd*
dev:    size   erasesize  name
mtd0: 00800000 00080000 "boot"
mtd1: 00200000 00080000 "logo"
mtd2: 00100000 00080000 "ENV"
mtd3: 00300000 00080000 "DTB"
mtd4: 00800000 00080000 "kernel"
mtd5: 00100000 00080000 "DTBbak"
mtd6: 00800000 00080000 "kernelbak"
mtd7: 3e100000 00080000 "rootfs"
```

其中 mtd5 和 mtd6 分区为备份的 dtb 和 kernel

擦除分区 mtd4 的起始 512K

```
root@freescale ~$ flash_erase /dev/mtd4 0 0
Erasing 512 Kibyte @ 780000 -- 100 % complete
root@freescale ~$ sync
```

重启系统，u-boot 阶段读备份的 kernel 到 ram，启动 linux os 的部分 log 信息如下

```
Normal Boot
Hit any key to stop autoboot:  0
nand boot.........

NAND read: device 0 offset 0xe00000, size 0x800000
 8388608 bytes read: OK

NAND read: device 0 offset 0xb00000, size 0x40000
 262144 bytes read: OK
Bad Linux ARM zImage magic!

NAND read: device 0 offset 0x1700000, size 0x800000
 8388608 bytes read: OK

NAND read: device 0 offset 0x1600000, size 0x40000
 262144 bytes read: OK
Kernel image @ 0x80800000 [ 0x000000 - 0x62a830 ]
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Using Device Tree in place at 83000000, end 8300c645

Starting kernel ...
```

擦除分区 mtd3 的起始 512K

root@freescale ~$ flash_erase /dev/mtd3 0 0

Erasing 512 Kibyte @ 280000 -- 100 % complete

root@freescale ~$ sync

重启系统，u-boot 阶段读备份的 dtb 到 ram，启动 linux os 的部分 log 信息如下

```
Normal Boot
Hit any key to stop autoboot:  0
nand boot.........

NAND read: device 0 offset 0xe00000, size 0x800000
 8388608 bytes read: OK

NAND read: device 0 offset 0xb00000, size 0x40000
 262144 bytes read: OK
Kernel image @ 0x80800000 [ 0x000000 - 0x62a830 ]
ERROR: Did not find a cmdline Flattened Device Tree
Could not find a valid device tree

NAND read: device 0 offset 0x1700000, size 0x800000
 8388608 bytes read: OK

NAND read: device 0 offset 0x1600000, size 0x40000
 262144 bytes read: OK
Kernel image @ 0x80800000 [ 0x000000 - 0x62a830 ]
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Using Device Tree in place at 83000000, end 8300c645

Starting kernel ...
```

1.22 kobs-ng update nand u-boot

Okmx6ul-c/c2 平台新发布的文件系统支持 kobs-ng 更新 nand u-boot，更新步骤介绍如下：
在 Linux OS 中，擦除原 nand u-boot 分区的 u-boot 镜像
flash_erase /dev/mtd0 0 0
2.将待更新的 u-boot 的镜像写步骤 1 擦除的分区
kobs-ng init -x /root/u-boot.imx
忽略终端打印的警告：
WARNING: Parameter 'chip_count' is no longer used, ignoring
sync
3.重启系统，新的 u-boot 镜像生效。

1.23 GPIO1_IO08 GPIO1_IO09 使用注意事项

GPIO1_IO08 在 u-boot 阶段使能 lcd 背光/屏蔽 lcd 背光函数中有相应的电平设置，在 kernel 阶段再次使用该 GPIO 复用为 ADC 时需要注意。
GPIO1_IO09 在 u-boot 阶段的 mmc 初始化函数中有相应的电平设置，在 kernel 阶段再使用该 GPIO 复用为 ADC 时需要注意。

1.24 硬浮点运算

i.MX6UL 的 CPU 本身有 FPU，支持 VFPv4-D32。假设测试程序为 test.c。
硬浮点交叉编译:

```
arm-linux-gcc -march=armv7-a -mfpu=neon -mfloat-abi=hard -o test test.c
```

加入编译参数 -mfloat-abi=hard，并且使用 arm-linux-readelf -A test 查看，如下：

```
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "7-A"
  Tag_CPU_arch: v7
  Tag_CPU_arch_profile: Application
  Tag_ARM_ISA_use: Yes
  Tag_THUMB_ISA_use: Thumb-2
  Tag_FP_arch: VFPv3
  Tag_Advanced_SIMD_arch: NEONv1
  Tag_ABI_PCS_wchar_t: 4
  Tag_ABI_FP_denormal: Needed
  Tag_ABI_FP_exceptions: Needed
  Tag_ABI_FP_number_model: IEEE 754
  Tag_ABI_align_needed: 8-byte
  Tag_ABI_align_preserved: 8-byte, except leaf SP
  Tag_ABI_enum_size: int
  Tag_ABI_HardFP_use: SP and DP
  Tag_ABI_VFP_args: VFP registers
  Tag_DIV_use: Not allowed
```

采用的是 Tag_ABI_VFP_args: VFP registers ， 已经采用硬浮点了。
而软浮点交叉编译：

arm-linux-gcc -o teset_soft test.c 并且使用 arm-linux-readelf -A test_soft 查看，如下：

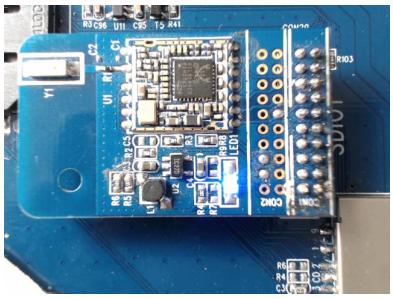
```
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "ARM10TDMI"
  Tag_CPU_arch: v5T
  Tag_ARM_ISA_use: Yes
  Tag_THUMB_ISA_use: Thumb-1
  Tag_ABI_PCS_wchar_t: 4
  Tag_ABI_FP_denormal: Needed
  Tag_ABI_FP_exceptions: Needed
  Tag_ABI_FP_number_model: IEEE 754
  Tag_ABI_align_needed: 8-byte
  Tag_ABI_align_preserved: 8-byte, except leaf SP
  Tag_ABI_enum_size: int
  Tag_DIV_use: Not allowed
```

未使用 VFP.

进行 10 亿次加减乘除运算，硬浮点时间为 1 分 34 秒 8 软浮点时间为 4 分 19 秒 7。

1.25 SDIO WIFI 使用及测试

SDIO WIFI 无线局域模组是选配模块。如若有需求，请联系飞凌嵌入式销售人员。i.MX6UL 支持飞凌提供的 8189es 模块。连接方法如图：



使用之前需要修改替换设备树。采用的是 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts 由：

```
480 &usdhc1 {
……
485              cd-gpios = <&gpio1 19 0>;
……
490 };
```

修改为：

```
480 &usdhc1 {
 ……
 485    /*        cd-gpios = <&gpio1 19 0>;     */
 486               nonremovable;
……
491 };
```

编译并替换为此设备树。

SDIO WIFI 功能测试步骤：

步骤 1：开发板断电，连接好飞凌的 SDIO WIFI 到飞凌开发板的 SDIO 接口，正确安装如上图。

步骤 2：开发板上电，启动 Linux 系统，默认插入之后，模块会自动加载，如果模块没自动加载，请确保已经卸载之后，手动加载。

```
root@freescale ~$ insmod /lib/modules/$(uname -r)/kernel/drivers/net/wireless/realtek/rtl8189ES/8189es.ko
```

查看加载驱动：

```
root@freescale ~$lsmod
```

出现如下信息，表示模块和驱动匹配成功

```
root@freescale ~$lsmod
8189es 886788 0 - Live 0x7f000000
```

步骤 3：执行下面的命令，检测开发板 wifi 网卡状况，路由器使用 wpa 加密。

```
root@freescale ~$ifconfig wlan0
```

串口信息：

```
root@freescale ~$ ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr E8:4E:06:13:0F:E8
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

步骤 4：关闭以太网卡，命令如下。

```
root@freescale ~$ifconfig eth0 down
root@freescale ~$ifconfig eth1 down
```

步骤 5：启动 SDIO WIFI，命令如下。

```
root@freescale ~$ifconfig wlan0 up
```

步骤 6：使用 SDIO WIFI 扫描无线网络设备，命令如下。

```
root@freescale ~$ iwlist wlan0 scan
          Cell 04 - Address: 00:21:27:65:77:5E
                    ESSID:"devnet"
                    Protocol:IEEE 802.11bg
                    Mode:Master
                    Frequency:2.437 GHz (Channel 6)
                    Encryption key:on
                    Bit Rates:54 Mb/s
                    Quality=20/100  Signal level=87/100
```

步骤 7：设置 SDIO WIFI 的 ESSID。（此步骤可以省略）

```
#iwconfig wlan0 essid devnet
```

步骤 8：生成 wpa 密码，wpa_passphrase 命令从标准输入读取明文，执行命令后占用终端等待明文输入。

```
# wpa_passphrase "devnet"  > wpa.conf
1234567890
```

输入明文密码，回车结束后自动保存到 wpa.conf。

步骤 9：连接路由器，命令如下。

#wpa_supplicant -Dwext -cwpa.conf -iwlan0 &

```
[  245.994674] wlan0: send auth to [redacted] (try 1/3)
wlan0: Association request to the driver failed
[  246.004576] wlan0: authenticated
[  246.009335] wlan0: associate with [redacted] (try 1/3)
[  246.025729] wlan0: RX AssocResp from [redacted] (capab=0x431 status=0
aid=4)
[  246.034222] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[  246.040847] wlan0: associated
wlan0: Associated with 0c:82:68:9d:53:78
wlan0: WPA: Key negotiation completed with [redacted] [PTK=CCMP GTK=TKIP]
wlan0: CTRL-EVENT-CONNECTED - Connection to [redacted] completed (auth) [
id=0 id_str=]
```

如果出现 wifi 模块连接失败，重连路由器前，需要采用 ps 查看是否存在 wpa_supplicant -Dwext -cwpa.conf -iwlan0 进程。如果存在，将此进程 kill 掉之后，再连接路由器。

步骤 10：自动 ip 地址分配 dhcp，命令如下。

udhcpc -iwlan0

```
root@freescale /media/mmcblk0p1$ udhcpc -iwlan0
udhcpc (v1.20.2) started
Sending discover...
Sending select for 192.168.1.108...
Lease of 192.168.1.108 obtained, lease time 7200
Deleting routers
adding dns 202.106.0.20
adding dns 10.198.1.1
root@freescale /media/mmcblk0p1$
```

步骤 11：ping ip 或者域名，命令如下。

#ping www.forlinx.com

```
root@freescale /$ ping www.forlinx.com
PING www.forlinx.com (223.4.217.169): 56 data bytes
64 bytes from 223.4.217.169: seq=0 ttl=116 time=34.545 ms
64 bytes from 223.4.217.169: seq=1 ttl=116 time=33.062 ms
64 bytes from 223.4.217.169: seq=2 ttl=116 time=33.745 ms
```

步骤 12：卸载已经加入内核的模块。

root@freescale ~$ rmmod 8189es

[725.946669] RTL871X: module exit start

[725.993183] RTL871X: indicate disassoc

[726.000795] RTL871X: rtw_ndev_uninit(wlan0) if1

[726.068585] RTL871X: rtw_cmd_thread: DriverStopped(True) SurpriseRemoved(False) break at line 564

[726.088291] RTL871X: rtw_dev_unload: driver not in IPS

[726.099148] RTL871X: module exit success

如果采用 wep 加密方式路由器连接时，采用如下命令：

设置 essid：

root@freescale ~$ iwconfig wlan0 essid “devnet”

设置路由器访问密码：

root@freescale ~$ iwconfig wlan0 key “1234567890”

之后动态分配 IP 或静态设置 IP 与网关均可。

注：本小节中是一个 SDIO WIFI 连接路由的示例。由于网络环境的不同，所以在您做本实验时，请根据实际情况进行设置。

1.26 JDK 的支持

1. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
   在此网址下载的
   jdk-8u151-linux-arm32-vfp-hflt.tar.gz
   jdk-8u151-linux-arm32-vfp-hflt-demos.tar.gz
2. 解压上述 2 个压缩包，比如解压到/home/root/ jdk1.8.0_151 下
3. 在/etc/profile 最后添加
   JAVA_HOME=/home/root/jdk1.8.0_151
   CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
   PATH=$JAVA_HOME/bin:$PATH
   export JAVA_HOME CLASSPATH PATH
4. source /etc/profile
   理论上应该出现：

```
root@imx6ulevk:~# java -version
random: nonblocking pool is initialized
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) Client VM (build 25.151-b12, mixed mode)
root@imx6ulevk:~# file jdk1.8.0_151/bin/java
jdk1.8.0_151/bin/java: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.26, BuildID[sha1]=120db07177774249eac5925cd
51bc42cf3033ccd, not stripped
```

   实际出现
   $java –version
   -sh java： no find
5. 考虑 3.14.38 内核版本中 file jdk1.8.0.151/bin/java 时出现了/lib/ld-linux-armhf.so.3 这个库。
   root@imx6ulevk:/lib# ls -la ld-linux-armhf.so.3
   lrwxrwxrwx 1 root root 10 Jan 1 1970 ld-linux-armhf.so.3 -> ld-2.23.so
   那就查看一下 3.14.38 中这个库是否存在。
   发现只有 ld-2.13.so，那就做个软链接试试。
   结果
   root@freescale /$ java -version
   java version "1.8.0_151"
   Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
   Java HotSpot(TM) Client VM (build 25.151-b12, mixed mode)
   验证 java 应该是可用的。
6 验证
   本人能力有限，不会写 java 程序，写了一个 helloworld 的简单 java 程序。测试一下。
   root@freescale /$ java ArgsTest
   0
   hello zhangdongguang,java ok
   结果正确。
   初步判断应该是可用了。
7. 进一步复杂程序测试

jdk-8u151-linux-arm32-vfp-hflt-demos.tar.gz 这里面有很多测试程序。

解压为 sample demo 两个文件夹。

```
$javac   sample/forkjoin/mergesort/ MergeDemo.java
```

编译完之后有：

```
root@freescale ~/jdk1.8.0_151/sample/forkjoin/mergesort$ ls
MergeDemo$1.class                    MergeDemo.java
MergeDemo$Configuration.class    MergeSort$MergeSortTask.class
MergeDemo$Range.class                MergeSort.class
MergeDemo.class                      MergeSort.java
```

再运行测试：

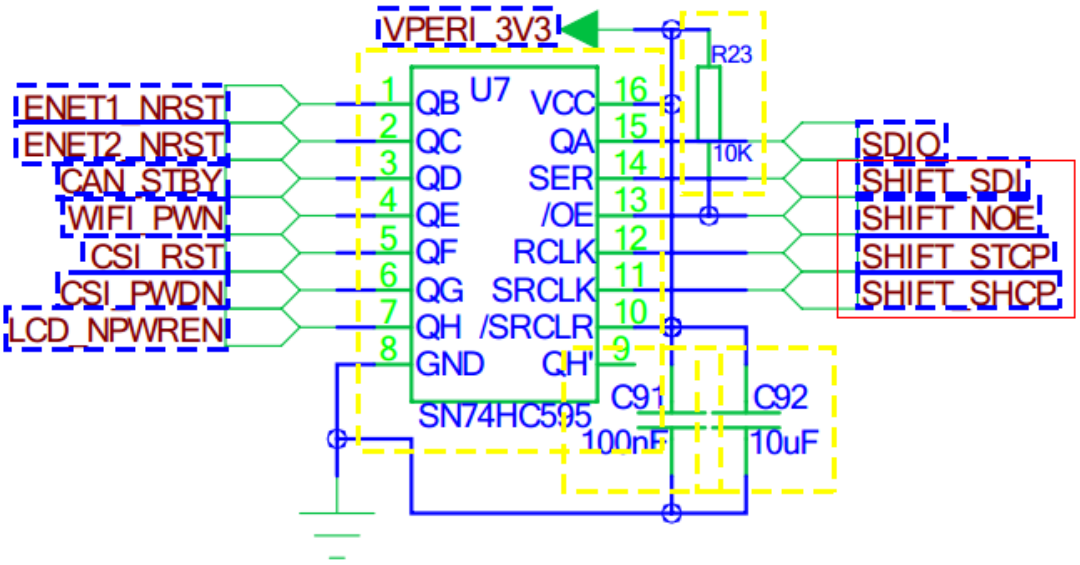
```
root@freescale ~/jdk1.8.0_151/sample/forkjoin/mergesort$ java MergeDemo
Default configuration. Running with parameters: 20000 20000 10 2 2 10
```

测试程序可用。

其他例程也可这样测试。

1.27 去掉 SN74HC595 芯片

有客户反馈说不使用 SN74HC595 芯片，uboot 与内核中均有设置，对应的 CPU 侧的这个几个引脚，如果悬空，不影响使用。如果用作输入，可能会影响启动。



另外这个 SPI 转 GPIO 对应出来的 GPIO 控制一些使能引脚，去掉此模块的话，其他的功能也需要修改。

比如采用 CSI_DATA0-ENET1_NRST　GPIO4_IO21
CSI_DATA1-ENET2_NRST　GPIO4_IO22
CSI_DATA2-CAN_STBY　GPIO4_IO23
CSI_DATA3-WIFI_PWN　GPIO4_IO24
CSI_DATA4-CSI_RST　GPIO4_IO25
CSI_DATA5-CSI_PWDN　GPIO4_IO26
CSI_DATA6-LCD_NPWREN　GPIO4_IO27

查看设备树 arch/arm/boot/dts/imx6ul-14x14-evk-emmc-c-7-800x480.dts

```
spi4 {
                compatible = "spi-gpio";
                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_spi4>;
                pinctrl-assert-gpios = <&gpio5 8 GPIO_ACTIVE_LOW>;
                status = "okay";
                gpio-sck = <&gpio5 11 0>;
                gpio-mosi = <&gpio5 10 0>;
                cs-gpios = <&gpio5 7 0>;
                num-chipselects = <1>;
                #address-cells = <1>;
                #size-cells = <0>;

                gpio_spi: gpio_spi@0 {
                        compatible = "fairchild,74hc595";
                        gpio-controller;
                        #gpio-cells = <2>;
                        reg = <0>;
                        registers-number = <1>;
                        registers-default = /bits/ 8 <0xa7>;
                        spi-max-frequency = <100000>;
                };
        };
```

将 status 修改为 disabled

搜索 gpio_spi，有

CAN_STBY 设置：

```
reg_can_3v3: regulator@0 {
                        compatible = "regulator-fixed";
                        reg = <0>;
                        regulator-name = "can-3v3";
                        regulator-min-microvolt = <3300000>;
                        regulator-max-microvolt = <3300000>;
                        gpios = <&gpio_spi 3 GPIO_ACTIVE_LOW>;
                        startup-active-us = <20000>;
                        enable-active-high;
                };
```

修改为

```
gpios = <&gpio4 23 GPIO_ACTIVE_LOW>;
```

```
CSI_RST/CSI_PWDN
ov9650: ov9650@30 {
                compatible = "ovti,ov9650";
                reg = <0x30>;
                pinctrl-names = "default";
```

```
                pinctrl-0 = <&pinctrl_csi1>;
                clocks = <&clks IMX6UL_CLK_CSI>;
                clock-names = "csi_mclk";
                pwn-gpios = <&gpio_spi 6 1>;
                rst-gpios = <&gpio_spi 5 0>;
                csi_id = <0>;
                mclk = <24000000>;
                mclk_source = <0>;
                status = "okay";
                port {
                        ov9650_ep: endpoint {
                                remote-endpoint = <&csi1_ep>;
                        };
                };
        };
```

修改为

```
pwn-gpios = <&gpio4 26 1>;
rst-gpios = <&gpio4 25 0>;
```

WIFI_PWN 在 pinctrl_hog_1 中加入 MX6UL_PAD_CSI_DATA03__GPIO4_IO24

```
&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog_1>;
        imx6ul-evk {
                pinctrl_hog_1: hoggrp-1 {
                        fsl,pins = <
                                MX6UL_PAD_LCD_RESET__WDOG1_WDOG_ANY     0x30b0
                                MX6UL_PAD_UART1_RTS_B__GPIO1_IO19       0x17059 /*
SD1 CD */
                                MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT    0x17059
/* SD1 VSELECT */
                                MX6UL_PAD_SNVS_TAMPER5__GPIO5_IO05      0x130b1
                                MX6UL_PAD_CSI_DATA03__GPIO4_IO24        0x1b0b1

                        >;
                };
```

并在文件系统开机自启动脚本/etc/rc.d/rc.local 中加入

```
echo 120 >/sys/class/gpio/export
echo out >/sys/class/gpio/gpio120/direction
echo 0 >/sys/class/gpio/gpio120/value
```

ENET1_NRST/ENET2_NRST

修改

```
&fec1 {
        pinctrl-names = "default";
```

```
        pinctrl-0 = <&pinctrl_enet1>;
        phy-mode = "rmii";
        phy-handle = <&ethphy0>;
        status = "okay";
};

&fec2 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_enet2>;
        phy-mode = "rmii";
        phy-handle = <&ethphy1>;
        status = "okay";
……
```

为

```
&fec1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_enet1>;
        phy-mode = "rmii";
        phy-handle = <&ethphy0>;
        phy-reset-duration = <100>;
        phy-reset-gpios = <&gpio4 21 0>;
        status = "okay";
};

&fec2 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_enet2>;
        phy-mode = "rmii";
        phy-handle = <&ethphy1>;
        phy-reset-duration = <100>;
        phy-reset-gpios = <&gpio4 22 0>;
        status = "okay";
……
```

LCD_NPWREN 如果硬件默认设置死的话，软件不用修改。如果软件修改的话，uboot 阶段设置 GPIO 控制。在内核 arch/arm/mach-imx/mach-imx6ul.c 的 imx6ul_init_machine 中添加 GPIO 控制高低电平。或者在设备树中加入对应的 gpio，方法参考“**创建 dev/gpio 节点**”部分。

1.28 OTG 修改模式

设备树中设置为 device 模式：

```
&usbotg1 {
    dr_mode = "peripheral";//默认设置为 device 模式，所以 MFG 下载可以用。
    status = "okay";
};
```

设备树中设置为 host 模式：

```
&usbotg1 {
vbus-supply = <&reg_usb_otg1_vbus>;
dr_mode = "host";
status = "okay";
};
```

1.29 SSH 登录

1.)生成密钥

```
mkdir /etc/dropbear
cd /etc/dropbear/
dropbearkey -t rsa -f dropbear_rsa_host_key
dropbearkey -t dss -f dropbear_dss_host_key
```

2.)开启 dropbear 服务

```
vi /etc/rc.d/rc.local   合适的位置添加如下行：
/usr/sbin/dropbear
```

3.）修改 root 用户密码（否则无法通过 ssh 登陆开发板）

```
 passwd root
```

按提示输入即可

4.）配置开发板 IP 和网关 如：

```
ifconfig eth1 192.168.x.xxx
route add gw 192.168.x.x dev eth1
```

5.)测试 dropbear

1.) 测试由 linux 主机通过 ssh 访问开发板

```
test@forlinx:~$ ssh root@192.168.0.232
root@192.168.0.232's password:
root@freescale ~$
```

2.）测试由 Windows 主机通过 putty 访问开发板

```
 login as: root
 root@192.168.0.232's password:
 root@freescale ~$
```

3.）由开发板访问 linux 主机

```
root@freescale ~$ dbclient    test@192.168.0.6
test@192.168.0.6's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.4.0-31-generic x86_64)
 * Documentation:   https://help.ubuntu.com/
New release '16.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
Last login: Tue Mar 13 13:32:24 2018 from 192.168.0.6
test@forlinx:~$
```

1.30 4G-AP

1. EC20 4G 模块拨号成功并分配 IP，可连接外网。设置转发规则：

```
root@freescale /$ ./quectel-CM    &     /*拨号，如果文件系统中无此应用程序，请参考应用笔记
```

```
中源码，交叉编译之后，拷贝到文件系统中*/
echo 1 > /proc/sys/net/ipv4/ip_forward /* 打开 IP 转发 */
iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE   /*eth2 为 4G 模块识别出的网卡，设置转发规则 */
```

2. 设置 WiFi 的模式与 IP

```
确保模块 8723bu 已经加载。
ifconfig wlan0 up        /*打开 WiFi*/
ifconfig wlan0 192.168.0.10 netmask 255.255.255.0   /*设置 IP 与子网掩码*/
ifconfig wlan0 promisc          /*设置 wlan0 为混杂模式 */
```

3. 开启 AP

```
udhcpd /etc/udhcpd.conf &      /*WiFi 地址、网关等配置信息*/
/home/hostapd -d /etc/hostapd.conf & /* 加密方式、用户名、密码等设置，此时用户名为 FORLINX，密码为 12345678 */
```

4. 手机等移动终端可以通过 WiFi 连接到 FCU1101 的 AP 热点，访问外网。

5. 如果使用的华为的 ME-909s 模块，按软件手册中先进行拨号，再配置 iptables 转发规则，即可实现通过 4G 模块实现热点功能。