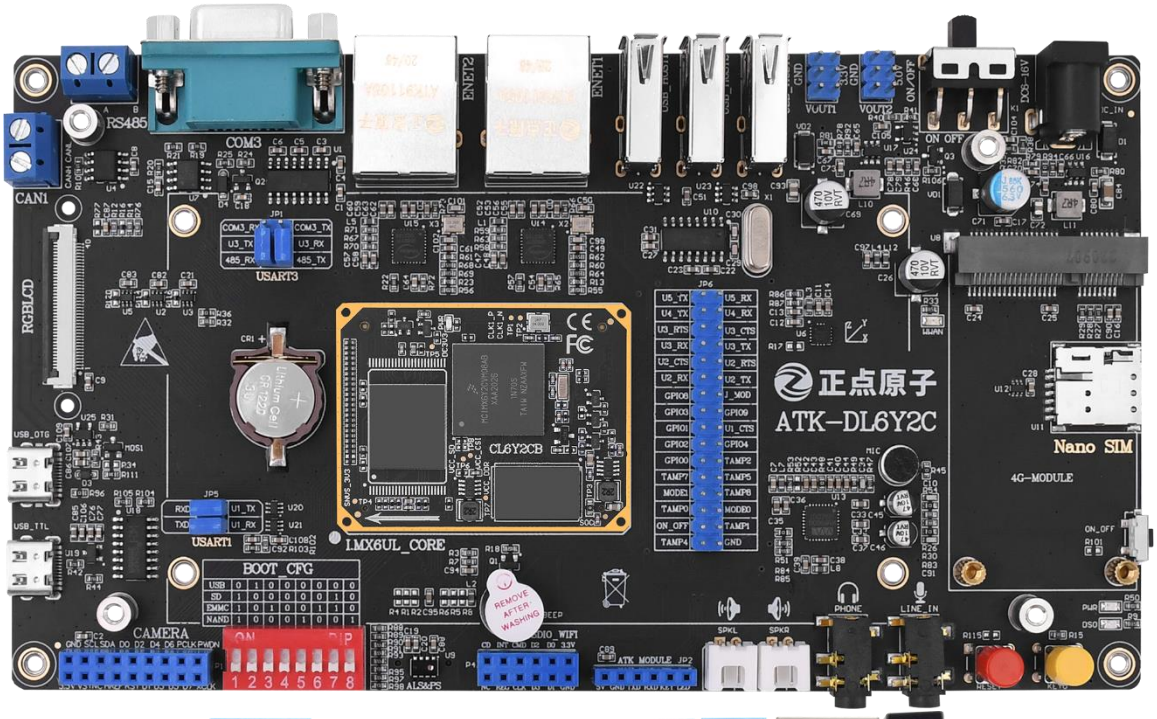
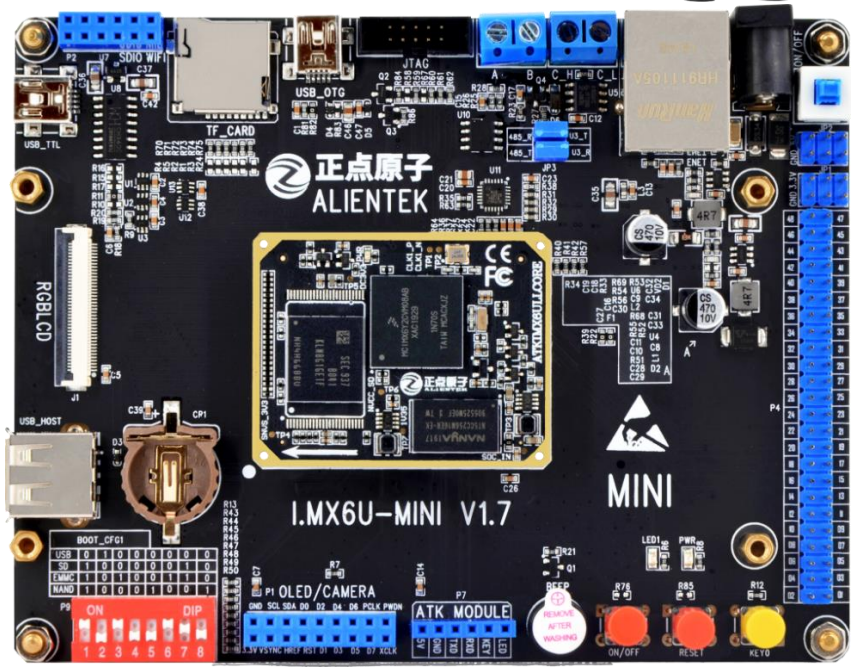
I.MX6U 出厂系统外设

复用参考文档 V1.3.1





正点原子

正点原子公司名称 ：广州市星翼电子科技有限公司

原子哥在线教学平台：www.yuanzige.com

开源电子网 / 论坛 ：http://www.openedv.com/forum.php

正点原子淘宝店铺 ：https://openedv.taobao.com

正点原子官方网站 ：www.alientek.com

正点原子 B 站视频 ：

https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请关注正点原子公众号，资料发布更新我们会通知。

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

扫码关注正点原子公众号

扫码下载“原子哥”APP

文档更新说明

版本	版本更新说明	负责人	校审	发布日期
V1.0	初稿：	正点原子 linux 团队	正点原子 linux 团队	2022.01.12
V1.1	1、 修复 5.4 小节 IO 测试中 GPIO 编号笔误 2、 添加 6.4 小节 uboot 修改 JTAG 相关管脚配置	正点原子 linux 团队	正点原子 linux 团队	2022.03.01
V1.2	1、 添加应用编程测试 demo 示例 2、 修改笔误	正点原子 linux 团队	正点原子 linux 团队	2022.08.02
V1.3	1、为适配 V2.4 ATK-DL6Y2C 底板，添加第四章、第五章、第六章里的修改说明。主要为以下内容： V2.4 底板网络 PHY 换成 SR8201F，ENET2 和 ENET1 的 PHY 地址分别 0x01,0x02，改音频 IIC 接口为 I2C1(之前是 I2C2)	正点原子 linux 团队	正点原子 linux 团队	2022.08.16
V1.3.1	1、修改 5.3.2 小节 enet1 管脚复用节点中添加 MDIO 和 MDC 管脚笔误	正点原子 linux 团队	正点原子 linux 团队	2022.11.14

目录

前言 7
第一章 出厂系统源码及编译工具 8
1.1 出厂系统源码下载 8
1.2 编译工具 8
1.3 出厂源码编译 8
1.4 设备树、uboot 文件说明 8
第二章 出厂系统外设资源 9
2.1 出厂系统资源表 9
【外设裁剪篇说明】 10
第三章 裁剪 CAMERA 11
3.1 驱动及管脚定义 11
3.2 内核裁剪 CAMERA 驱动 13
3.2.1 修改 menuconfig 配置 13
3.2.2 修改设备树文件 14
3.2.3 编译内核和设备树 17
3.3 命令测试 GPIO 输出 18
3.4 程序测试 GPIO 输出\输入\中断 19
3.4.1 GPIO 输出测试 19
3.4.2 GPIO 输入测试 19
3.4.3 GPIO 中断测试 20
第四章 裁剪 ENET1 21
4.1 驱动及管脚定义 21
4.2 U-boot 裁剪 ENET1 24
4.2.1 源码编译 24
4.2.2 修改 mx6ullevk.h 24
4.2.3 修改 mx6ullevk.c 25
4.2.4 uboot 指令 gpio 测试 26
4.3 内核裁剪 ENET1 驱动 28
4.3.1 修改内核源码 28
4.3.2 修改设备树文件 28
4.4 命令测试 GPIO 输出 29
4.5 程序测试 GPIO 输出\输入\中断 30
第五章 裁剪 ENET2 31
5.1 驱动及管脚定义 31
5.2 U-boot 裁剪 ENET2 33
5.2.1 源码编译 33
5.2.2 修改 mx6ullevk.h 33
5.2.3 修改 mx6ullevk.c 34
5.2.4 uboot 指令 gpio 测试 35
5.3 内核裁剪 ENET2 驱动 36
5.3.1 修改内核源码 36

5.3.2 修改设备树文件 37
5.4 命令测试 GPIO 输出 39
5.5 程序测试 GPIO 输出\输入\中断 39
第六章 裁剪 AUDIO/JTAG 40
6.1 驱动及管脚定义 40
6.2 内核裁剪 AUDIO 驱动 42
6.2.1 修改 menuconfig 配置 42
6.2.2 修改设备树文件 43
6.2.3 编译内核和设备树 45
6.3 IO 测试 45
6.3.1 设备树添加 GPIO 信息 45
6.3.2 测试 46
6.4 uboot 修改 JTAG 相关管脚配置 47
6.4.1 修改 mx6ullevk.c 文件 48
6.4.2 命令测试 GPIO 输出 49
6.4.3 程序测试 GPIO 输出\输入\中断 50
第七章 裁剪 LCD 51
7.1 驱动及管脚定义 51
7.2 U-boot 裁剪 LCD 55
7.2.1 源码编译 55
7.2.2 修改 mx6ullevk.c 55
7.2.3 修改 mx6ullevk.h 56
7.2.4 修改 video.c 56
7.2.5 uboot 指令 gpio 测试 58
7.3 内核裁剪 LCD 驱动 59
7.4 命令测试 GPIO 输出 62
7.5 程序测试 GPIO 输出\输入\中断 62
【外设复用篇说明】 63
第八章 多路 GPIO 复用 64
8.1 GPIO 复用修改 64
8.2 设备树添加 GPIO 64
8.3 GPIO 测试 67
8.4 程序测试 GPIO 输出\输入\中断 68
第九章 8 路 UART 复用 69
9.1 UART1（ttymxc0） 69
9.2 UART2（ttymxc1） 69
9.3 UART3（ttymxc2) 72
9.4 UART4（ttymxc3） 72
9.5 UART5（ttymxc4） 75
9.6 UART6（ttymxc5） 79
9.7 UART7（ttymxc6） 80
9.8 UART8（ttymxc7） 80
9.9 CAMERA 复用为 UART5\6 81

9.10 串口应用程序测试 86
第十章 2 路 CAN 复用 86
10.1 修改设备树文件 86
10.2 CAN 测试 88
10.3 CAN 应用程序测试 90
第十一章 多路 ADC 复用 91
11.1 GPIO_1 92
11.2 GPIO_0 94
11.2.1 USB_OTG1_ID 脚问题 95
11.3 其他 ADC 复用 96
第十二章 多路 PWM 复用 98
12.1 PWM3 99
12.1.1 修改设备树文件 99
12.1.2 PWM 测试 101
12.2 裁剪 ENET1 做 PWM1\2\5\6\7\8 103
12.2.1 修改内核和设备树 103
12.2.2 PWM 测试 106
12.3 裁剪 AUDIO\JTAG 做 PWM6\7\8 109
12.3.1 修改内核和设备树 109
12.3.2 PWM 测试 110
12.4 PWM 应用程序测试 112
附录 113
常用外设复用可选管脚 113

前言

本文档基于企业用户开发项目需要而编写，假设用户已经搭建好开发环境和掌握 Linux 操作。本文档基于正点原子 I.MX6ULL 核心板，列举了常用外设的修改方法，仅提供修改参考，每个人的修改需要不同，请读者根据自己方案和能力，选择外设进行修改。

开发环境：Ubuntu16、vscode

源码母本：正点原子 I.MX6U 开发板光盘 A-基础资料\01、例程源码\03、正点原子 Uboot 和 Linux 出厂源码

硬件：正点原子 I.MX6ULL 核心板、正点原子 I.MX6ULL 开发板（ALPHA\MINI）

第一章 出厂系统源码及编译工具

1.1 出厂系统源码下载

方式一：百度网盘下载

资料盘 开发板资料链接： https://pan.baidu.com/s/1inZtndgN-L3aVfoch2-sKA 提取码：m65i

具体路径：开发板光盘 A-基础资料\01、例程源码\03、正点原子 Uboot 和 Linux 出厂源码

方式二：coding 下载

下载链接：

https://alientek-linux.coding.net/public/imx6ull/01_Soure_Code/git/files/master/03%E3%80%81%E6%AD%A3%E7%82%B9%E5%8E%9F%E5%AD%90Uboot%E5%92%8CLinux%E5%87%BA%E5%8E%82%E6%BA%90%E7%A0%81

1.2 编译工具

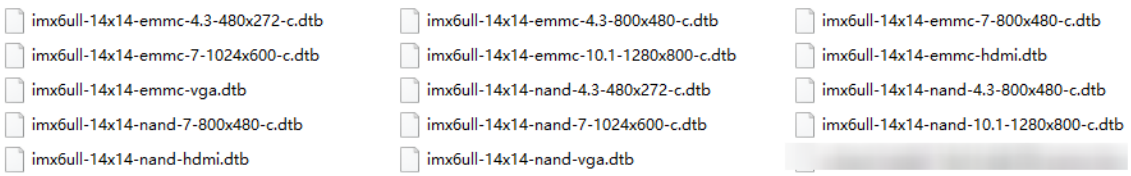
参考文档：开发板光盘 A-基础资料\10、用户手册\《【正点原子】I.MX6U 用户快速体验》
编译工具：Poky 交叉编译工具链
安装环境：Ubuntu16.04
安装方法：《【正点原子】I.MX6U 用户快速体验》第 4.2 小节 安装 Poky 交叉编译工具链

1.3 出厂源码编译

在开始修改源码前，至少要编译一次出厂源码，确保编译环境没有问题、源码完整。
编译方法：
编译出厂 uboot：《【正点原子】I.MX6U 用户快速体验》第 4.3 小节 编译出厂源码 U-boot
编译出厂内核：【正点原子】I.MX6U 用户快速体验》第 4.4 小节 编译出厂源码内核及模块

1.4 设备树、uboot 文件说明

正点原子 Linux 开发板适配多款 RGB 屏幕，设备树根据存储芯片和屏幕参数分为多种，如下图所示：



设备树文件根据核心板型号和屏幕型号选择，没有屏幕默认用 4.3-480x272 的设备树。本文档大多数编译示例以 imx6ull-14x14-emmc-4.3-480x272-c.dtb 设备树为例。如果有用到屏幕，用户可以根据自己的屏幕选择编译对应的设备树来启动。

uboot 文件，根据核心板参数分为 u-boot-imx6ull-14x14-ddr512-emmc.imx 和 u-boot-imx6ull-14x14-ddr256-nand.imx。

用 imxdownload 烧写 SD 卡启动 uboot，emmc 核心板可以使用 u-boot-imx6ull-14x14-ddr512-emmc.bin，nand flash 核心板可以使用 u-boot-imx6ull-14x14-ddr256-nand-sd.bin

第二章 出厂系统外设资源

2.1 出厂系统资源表

〇：表示提供源码
◎：表示提供源码和教程资料
□：表示提供系统和源码，出厂系统可以直接使用

外设功能	出厂内核源码驱动	教程源码驱动	应用开发	Qt 开发	裸机开发
GPIO	□	◎	◎	◎	◎
LED	□	◎	◎	◎	◎
KEY	□	◎	◎	◎	◎
LCD	□	◎	◎	◎	◎
BackLight	□	◎	◎		◎
UART	□	◎	◎	◎	◎
I2C	□	◎	◎	◎	◎
SPI	□	◎	◎	◎	◎
USB	□	◎			
FEC(NET)	□	◎	◎	◎	
PWM	□	◎	◎		◎
OV5640	□	〇	◎	◎	
OV2640	□	〇	〇	〇	
OV7725(不带FIFO)	□	〇	〇	〇	
WM8960	□	◎	◎	◎	
RTC	□	◎			
WDOG	□		◎		
CAN	□	◎	◎	◎	
ADC	□	◎	◎		
DHT11	□				
DS18B20	□				
RTL8189	□	◎			
BEEP	□	◎	◎	◎	◎
RS232	□	◎			
RS485	□	◎			
GPS	□	◎			
ME3630	□	◎			
EC20	□	◎			
HDMI	□	◎			◎
USB bluetooth	□			◎	

【外设裁剪篇说明】

出厂系统内核默认是基于 I.MX6ULL 开发板-阿尔法开发板配置的，如果用户需要某些外设功能，或者需要修改某些管脚用作别的功能，就需要进行外设裁剪。

外设裁剪，一般涉及内核裁剪，有些外设如果用在 uboot 中，也需要对 uboot 进行外设裁剪。同时，还要结合底板原理图，考虑底板电阻、外设设计的影响。

第三章 裁剪 CAMERA

3.1 驱动及管脚定义

以阿尔法开发板为例，开发板 camera 接口原理图：

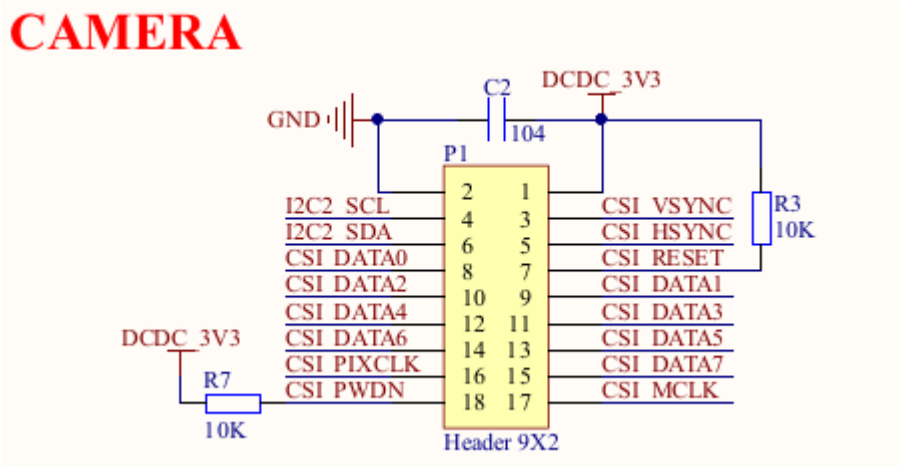


图 3.1-1 CAMERA 接口原理图

注意原理图上的上下拉电阻，会影响 IO 控制。

原理图管脚名	GPIO	可复用功能
CSI_VSYNC	GPIO4_IO19	CSI_VSYNC、USDHC2_CLK、I2C2_SDA、EIM_RW、GPIO4_IO19、PWM7_OUT、UART6_RTS_B、ESAI_TX4_RX1
I2C2_SCL	GPIO1_IO30	GPIO1_IO30、ECSPI2_MOSI、EPDC_PWRCTRL02、UART5_TX、ENET2_CRS、I2C2_SCL、CSI_DATA14、CSU_CSU_ALARM_AUTO0
CSI_HSYNC	GPIO4_IO20	CSI_HSYNC、USDHC2_CMD、I2C2_SCL、EIM_LBA_B、GPIO4_IO20、PWM8_OUT、UART6_CTS_B、ESAI_TX1
I2C2_SDA	GPIO1_IO31	UART5_RX、ENET2_COL、I2C2_SDA、CSI_DATA15、CSU_CSU_INT_DEB、GPIO1_IO31、ECSPI2_MISO、EPDC_PWRCTRL03
CSI_RESET	GPIO1_IO02	I2C1_SCL、GPT1_COMPARE2、USB_OTG2_PWR、ENET1_REF_CLK_25M、USDHC1_WP、GPIO1_IO02、SDMA_EXT_EVENT00、SRC_ANY_PU_RESET、UART1_TX
CSI_DATA0	GPIO4_IO21	CSI_DATA02、USDHC2_DATA0、ECSPI2_SCLK、EIM_AD00、GPIO4_IO21、SRC_INT_BOOT、UART5_TX、ESAI_TX_HF_CLK
CSI_DATA1	GPIO4_IO22	CSI_DATA03、USDHC2_DATA1、

		ECSPI2_SS0、EIM_AD01、 GPIO4_IO22、SAI1_MCLK、 UART5_RX、ESAI_RX_HF_CLK
CSI_DATA2	GPIO4_IO23	CSI_DATA04、USDHC2_DATA2、 ECSPI2_MOSI、EIM_AD02、 GPIO4_IO23、SAI1_RX_SYNC、 UART5_RTS_B、ESAI_RX_FS
CSI_DATA3	GPIO4_IO24	CSI_DATA05、USDHC2_DATA3、 ECSPI2_MISO、EIM_AD03、 GPIO4_IO24、SAI1_RX_BCLK、 UART5_CTS_B、ESAI_RX_CLK
CSI_DATA4	GPIO4_IO25	CSI_DATA06、USDHC2_DATA4、 ECSPI1_SCLK、EIM_AD04、 GPIO4_IO25、SAI1_TX_SYNC、 USDHC1_WP、ESAI_TX_FS
CSI_DATA5	GPIO4_IO26	CSI_DATA07、USDHC2_DATA5、 ECSPI1_SS0、EIM_AD05、 GPIO4_IO26、SAI1_TX_BCLK、 USDHC1_CD_B、ESAI_TX_CLK
CSI_DATA6	GPIO4_IO27	CSI_DATA08、USDHC2_DATA6、 ECSPI1_MOSI、EIM_AD06 、 GPIO4_IO27、SAI1_RX_DATA、 USDHC1_RESET_B、 ESAI_TX5_RX0
CSI_DATA7	GPIO4_IO28	CSI_DATA09、USDHC2_DATA7、 ECSPI1_MISO、EIM_AD07、 GPIO4_IO28、SAI1_TX_DATA、 USDHC1_VSELECT、ESAI_TX0
CSI_PIXCLK	GPIO4_IO18	CSI_PIXCLK、USDHC2_WP、 RAWNAND_CE3_B、I2C1_SCL、 EIM_OE、GPIO4_IO18、 SNVS_HP_VIO_5、UART6_RX、 ESAI_TX2_RX3
CSI_MCLK	GPIO4_IO17	CSI_MCLK、USDHC2_CD_B、 RAWNAND_CE2_B、 I2C1_SDA、EIM_CS0_B、 GPIO4_IO17、 SNVS_HP_VIO_5_CTL、 UART6_TX、ESAI_TX3_RX2

3.2 内核裁剪 CAMERA 驱动

3.2.1 修改 menuconfig 配置

进入出厂内核源码目录下，打开 menuconfig 配置。

```
source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig -j 16
make menuconfig
```

修改 menuconfig 配置，如下：

```
Device Drivers  --->
    <*> Multimedia support  --->
        [*]   V4L platform devices  --->
            MXC Camera/V4L2 PRP Features support  --->
                < > OmniVision ov5640 camera support
                < > OmniVision ov5642 camera support
                < > OmniVision ov5640 camera support using mipi
```

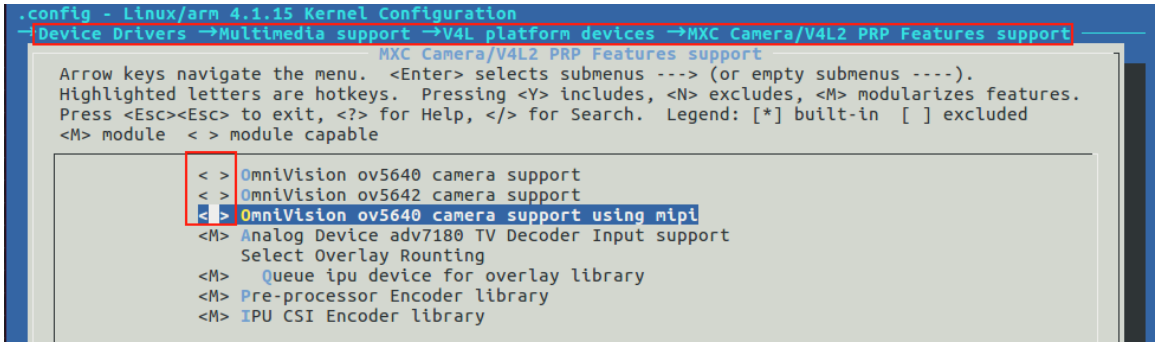


图 3.2-1 裁剪 camera 配置

save 另存为./arch/arm/configs/del_camera_imx_v7_defconfig 配置文件。

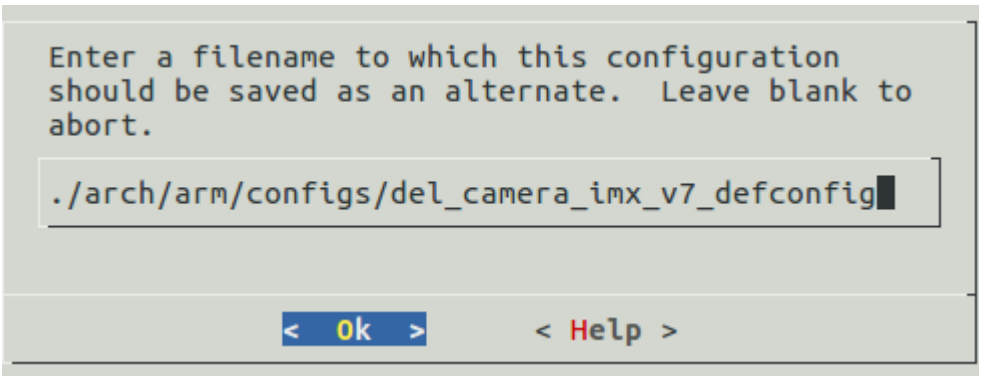


图 3.2-2 另存为配置文件

双击 Esc 键退出 menuconfig 配置。

3.2.2 修改设备树文件

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件进行修改。
搜索关键词 csi。
注释掉&csi 节点。

```
/* &csi {
    status = "okay";

    port {
        csi_ep: endpoint {
            remote-endpoint = <&camera_ep>;
        };
    };
}; */
```

注释掉 ov5640 外设信息。

```
/* #if ATK_CAMERA_ON
    ov5640: ov5640@3c {
        compatible = "ovti,ov5640";
        reg = <0x3c>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_csi1
                     &csi_pwn_rst>;
        clocks = <&clks IMX6UL_CLK_CSI>;
        clock-names = "csi_mclk";
        pwn-gpios = <&gpio1 4 1>;
        rst-gpios = <&gpio1 2 0>;
        csi_id = <0>;
        mclk = <24000000>;
        mclk_source = <0>;
        status = "okay";
        port {
            camera_ep: endpoint {
                remote-endpoint = <&csi_ep>;
            };
        };
    };
#endif */
```

注释掉 ov2640 外设信息。

```
/* #if ATK_CAMERA_OFF
        ov2640: camera@0x30 {
                compatible = "ovti,ov2640";
                reg = <0x30>;
                status = "okay";

                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_csi1
                             &csi_pwn_rst>;
                resetb = <&gpio1 2 GPIO_ACTIVE_LOW>;
                pwdn = <&gpio1 4 GPIO_ACTIVE_HIGH>;
                clocks = <&clks IMX6UL_CLK_CSI>;
                clock-names = "xvclk";

                port {
                        camera_ep: endpoint {
                                remote-endpoint = <&csi_ep>;
                bus-width = <8>;
                        };
                };
        };
#endif */
```

注释掉 ov7725 外设信息。

```
/* #if ATK_CAMERA_OFF
        ov7725: camera@0x21 {
                compatible = "ovti,ov772x","ovti,ov7725";
                reg = <0x21>;
                status = "okay";
                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_csi1
                             &csi_pwn_rst>;
                resetb = <&gpio1 2 GPIO_ACTIVE_LOW>;
                pwdn = <&gpio1 4 GPIO_ACTIVE_HIGH>;
                clocks = <&clks IMX6UL_CLK_CSI>;
                clock-frequency = <20000000>;
                clock-names = "mclk";

                port {
                        camera_ep: endpoint {
                                remote-endpoint = <&csi>;
                bus-width = <8>;
                        };
                };
        };
#endif */
```

注释掉 pinctrl_csi1 管脚配置信息。

```
/*      pinctrl_csi1: csi1grp {
            fsl,pins = <
                MX6UL_PAD_CSI_MCLK__CSI_MCLK        0x1b008
                MX6UL_PAD_CSI_PIXCLK__CSI_PIXCLK    0x1b008
                MX6UL_PAD_CSI_VSYNC__CSI_VSYNC      0x1b008
                MX6UL_PAD_CSI_HSYNC__CSI_HSYNC      0x1b008
                MX6UL_PAD_CSI_DATA00__CSI_DATA02    0x1b008
                MX6UL_PAD_CSI_DATA01__CSI_DATA03    0x1b008
                MX6UL_PAD_CSI_DATA02__CSI_DATA04    0x1b008
                MX6UL_PAD_CSI_DATA03__CSI_DATA05    0x1b008
                MX6UL_PAD_CSI_DATA04__CSI_DATA06    0x1b008
                MX6UL_PAD_CSI_DATA05__CSI_DATA07    0x1b008
                MX6UL_PAD_CSI_DATA06__CSI_DATA08    0x1b008
                MX6UL_PAD_CSI_DATA07__CSI_DATA09    0x1b008
            >;
        }; */
```

注释掉 csi_pwn_rst 管脚配置信息。

```
/*      csi_pwn_rst: csi_pwn_rstgrp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO02__GPIO1_IO02    0x10b0
                MX6UL_PAD_GPIO1_IO04__GPIO1_IO04    0x10b0
            >;
        }; */
```

检查是否有修改遗漏的地方，保存修改好的设备树文件。

3.2.3 编译内核和设备树

至此内核和设备树裁剪 camera 外设步骤完成。

之前我们保存了配置文件为 del_camera_imx_v7_defconfig，这里将用到此文件进行内核编译。设备树文件根据核心板型号和屏幕型号选择，没有屏幕默认用 4.3-480x272 的设备树。

```
source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make del_camera_imx_v7_defconfig
make zImage -j 16
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

```
alientek@ubuntu16:~/imx6ull/kernel$ source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
alientek@ubuntu16:~/imx6ull/kernel$ make del_camera_imx_v7_defconfig
#
# configuration written to .config
#
alientek@ubuntu16:~/imx6ull/kernel$ make zImage -j 16
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: 'include/generated/mach-types.h' is up to date.
  CHK     include/generated/bounds.h
  CHK     include/generated/asm-offsets.h
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  GZIP    kernel/config_data.gz
  CHK     kernel/config_data.h
  UPD     kernel/config_data.h
  CC      kernel/configs.o
  LD      kernel/built-in.o
  LINK    vmlinux
  LD      vmlinux.o
  MODPOST vmlinux.o
  GEN     .version
  CHK     include/generated/compile.h
  UPD     include/generated/compile.h
  CC      init/version.o
  LD      init/built-in.o
  KSYM    .tmp_kallsyms1.o
  KSYM    .tmp_kallsyms2.o
  LD      vmlinux
  SORTEX  vmlinux
  SYSMAP  System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  LZO     arch/arm/boot/compressed/piggy.lzo
  AS      arch/arm/boot/compressed/piggy.lzo.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
alientek@ubuntu16:~/imx6ull/kernel$ make imx6ull-14x14-emmc-4.3-480x272-c.dtb
  DTC     arch/arm/boot/dts/imx6ull-14x14-emmc-4.3-480x272-c.dtb
alientek@ubuntu16:~/imx6ull/kernel$
```

图 3.2-3 编译

将编译生成的 zImage 和设备树文件替换到开发板上启动。

3.3 命令测试 GPIO 输出

以 CSI_VSYNC 管脚为例。启动开发板，将万用表调到电压档位，黑表笔接地，红表笔接 CSI_VSYNC 接口。如下图所示：

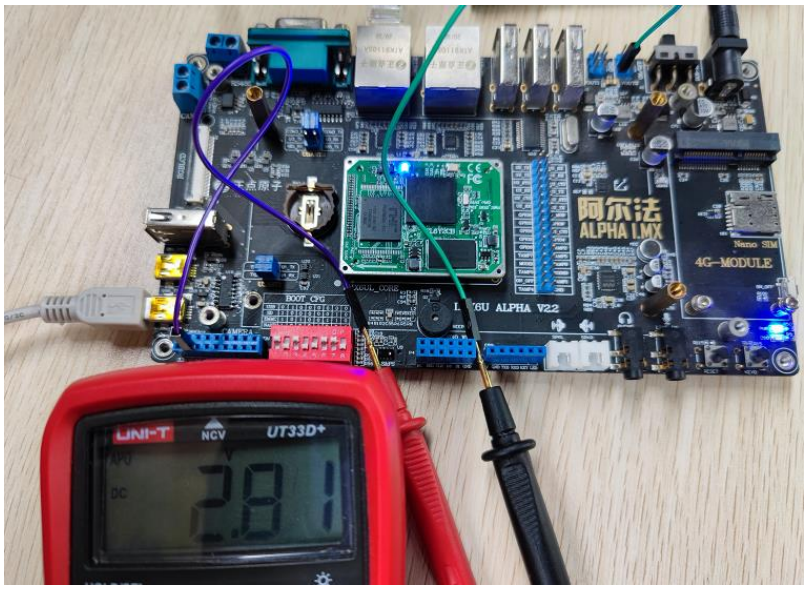


图 3.3-1 测试连接

进入开发板系统终端，进行 GPIO 操作。

```
cd /sys/class/gpio/
echo 115 > export
echo low > gpio115/direction
echo high > gpio115/direction
```

测试 CSI_VSYNC 管脚可以拉高/拉低输出。CSI 部分管脚配置到 2.8V 的电源组，具体可以看底板原理图。

依次测试其他 IO，可以直接测试的 IO 有 13 个。

原理图管脚名	GPIO 名	GPIO 编号	高/低电平	备注
CSI_VSYNC	GPIO4_IO19	115	2.81V/0V	
I2C2_SCL	GPIO1_IO30	30	-	此管脚还受其他外设控制
CSI_HSYNC	GPIO4_IO20	116	2.81V/0V	
I2C2_SDA	GPIO1_IO31	31	-	此管脚还受其他外设控制
CSI_RESET	GPIO1_IO02	2	3.3V/0V	
CSI_DATA0	GPIO4_IO21	117	2.81V/0V	
CSI_DATA1	GPIO4_IO22	118	2.81V/0V	
CSI_DATA2	GPIO4_IO23	119	2.81V/0V	
CSI_DATA3	GPIO4_IO24	120	2.81V/0V	
CSI_DATA4	GPIO4_IO25	121	2.81V/0V	
CSI_DATA5	GPIO4_IO26	122	2.81V/0V	
CSI_DATA6	GPIO4_IO27	123	2.81V/0V	
CSI_DATA7	GPIO4_IO28	124	2.81V/0V	
CSI_PIXCLK	GPIO4_IO18	114	2.81V/0V	
CSI_MCLK	GPIO4_IO17	113	2.81V/0V	

3.4 程序测试 GPIO 输出\输入\中断

测试程序参考《I.MX6U 嵌入式 Linux C 应用编程指南》里的 GPIO 应用编程。

源码路径：开发板光盘 A-基础资料\01、例程源码\11、Linux C 应用编程例程源码\16_gpio

编译方法：参考《I.MX6U 嵌入式 Linux C 应用编程指南 V1.4》519 页。

3.4.1 GPIO 输出测试

示例：执行 gpio_out 程序测试 GPIO115 输出高电平。

```
root@ATK-IMX6U:~# ls
driver  gpio_in  gpio_intr  gpio_out  shell
root@ATK-IMX6U:~# ./gpio_out
usage: ./gpio_out <gpio> <value>
root@ATK-IMX6U:~# ./gpio_out 115 1
root@ATK-IMX6U:~#
```

图 3.4-1 GPIO 输出测试

万用表电压档测试 GPIO115(即 GPIO4_IO19)，测试为高电平。

同理测试其他 GPIO 管脚，测试结果同上节测试表格。

3.4.2 GPIO 输入测试

示例：测试 GPIO115 输入电平。

用杜邦线连接 GND 和 GPIO115（即 GPIO4_IO19），给此管脚输入低电平。执行测试程序 gpio_in 测试输入电平。测试结果如下，打印 GPIO 输入电平为低电平。

```
root@ATK-IMX6U:~# ls
driver  gpio_in  gpio_intr  gpio_out  shell
root@ATK-IMX6U:~# ./gpio_in
usage: ./gpio_in <gpio>
root@ATK-IMX6U:~# ./gpio_in 115
value: 0
root@ATK-IMX6U:~#
```

图 3.4-2 GPIO 输入测试

再用杜邦线连接 3.3V 和 GPIO115（即 GPIO4_IO19），给此管脚输入高电平。执行测试程序 gpio_in 测试输入电平。测试结果如下，打印 GPIO 输入电平为高电平。

```
root@ATK-IMX6U:~# ./gpio_in 115
value: 1
root@ATK-IMX6U:~#
```

图 3.4-3 GPIO 输入测试

依次测试其他 IO，可以直接测试的 IO 有 13 个。

原理图管脚名	GPIO 名	GPIO 编号	输入高/低电平	备注
CSI_VSYNC	GPIO4_IO19	115	3.3V/0V	
I2C2_SCL	GPIO1_IO30	30	-	此管脚还受其他外设控制
CSI_HSYNC	GPIO4_IO20	116	3.3V/0V	
I2C2_SDA	GPIO1_IO31	31	-	此管脚还受其他外设控制
CSI_RESET	GPIO1_IO02	2	3.3V/0V	
CSI_DATA0	GPIO4_IO21	117	3.3V/0V	
CSI_DATA1	GPIO4_IO22	118	3.3V/0V	
CSI_DATA2	GPIO4_IO23	119	3.3V/0V	
CSI_DATA3	GPIO4_IO24	120	3.3V/0V	
CSI_DATA4	GPIO4_IO25	121	3.3V/0V	
CSI_DATA5	GPIO4_IO26	122	3.3V/0V	

CSI_DATA6	GPIO4_IO27	123	3.3V/0V	
CSI_DATA7	GPIO4_IO28	124	3.3V/0V	
CSI_PIXCLK	GPIO4_IO18	114	3.3V/0V	
CSI_MCLK	GPIO4_IO17	113	3.3V/0V	

3.4.3 GPIO 中断测试

示例：执行 gpio_intr 程序监测 GPIO115（即 GPIO4_IO19）的中断触发。

当执行命令之后，我们可以使用杜邦线将 GPIO4_IO19 引脚连接到 GND 或 3.3V 电源引脚上，来回切换，使得 GPIO4_IO19 引脚的电平状态发生由高到低或由低到高的状态变化，以验证 GPIO 中断的边沿触发情况；当发生中断时，终端将会打印相应的信息。

```
GPIO中断触发<value=1>
GPIO中断触发<value=1>
GPIO中断触发<value=1>
GPIO中断触发<value=0>
GPIO中断触发<value=0>
GPIO中断触发<value=0>
```

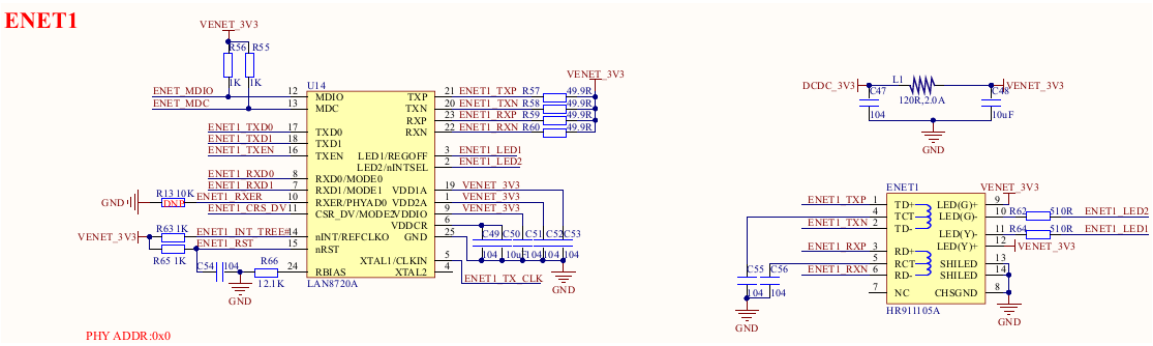
图 3.4-4 GPIO 中断测试

其他 GPIO 测试同理。

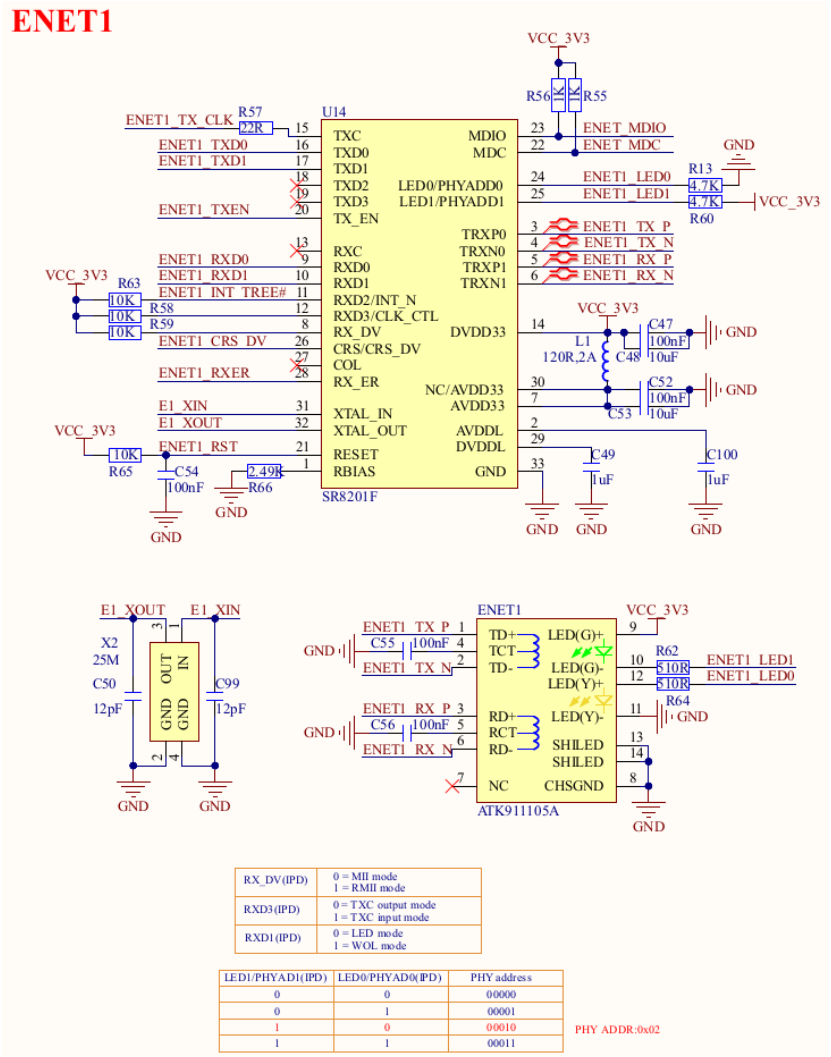
第四章 裁剪 ENET1

4.1 驱动及管脚定义

以 V2.2 版本的阿尔法开发板为例，使用的 PHY 是 LAN8720A，开发板 ENET1 接口原理图：

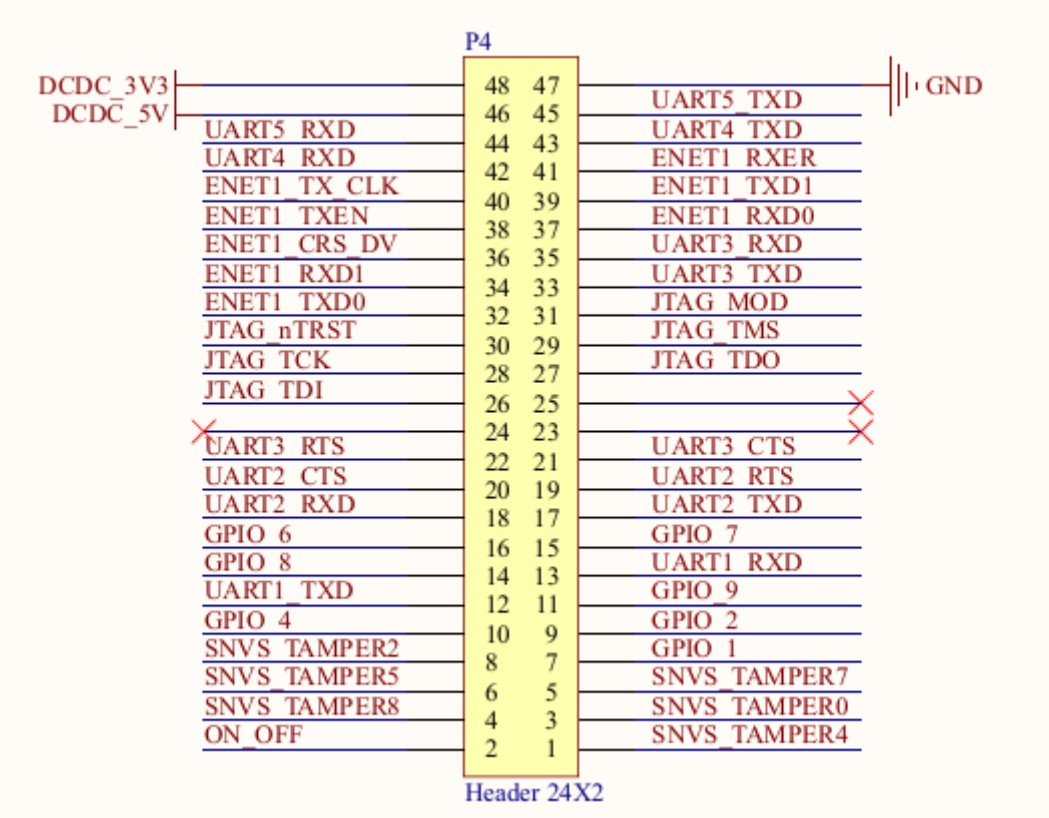


以 V2.4 版本的 ATK-DL6Y2C 开发板为例，使用的 PHY 是 SR8201F，开发板 ENET1 原理图：



MINI 板上去掉了 ENET1 相关的元器件，引出相关 IO。本小节的裁剪测试，用 MINI 板方便些，用阿尔法板和 ATK-DL6Y2C 就需要自行去除相关元器件，从核心板直接引出 IO 测试。

MINI 板相关原理图如下：



阿尔法原理图管脚	GPIO	可复用功能	MINI 板原理图管脚
ENET1_TXD0	GPIO2_IO03	ENET1_TDATA00、UART5_CTS_B、CSI_DATA19、FLEXCAN2_RX、GPIO2_IO03、KPP_COL01、USDHC2_VSELECT、EPDC_SDCE07	ENET1_TXD0
ENET1_TXD1	GPIO2_IO04	ENET1_TDATA01、UART6_CTS_B、PWM5_OUT、CSI_DATA20、ENET2_MDIO、GPIO2_IO04、KPP_ROW02、WDOG1_WDOG_RST_B_DEB、EPDC_SDCE08	ENET1_TXD1
ENET1_TXEN	GPIO2_IO05	ENET1_TX_EN、UART6_RTS_B、PWM6_OUT、CSI_DATA21、ENET2_MDC、GPIO2_IO05、KPP_COL02、WDOG2_WDOG_RST_B_DEB、EPDC_SDCE09	ENET1_TXEN

ENET1_RXD0	GPIO2_IO00	ENET1_RDATA00、UART4_RTS_B、PWM1_OUT、CSI_DATA16、FLEXCAN1_TX、GPIO2_IO00、KPP_ROW00、USDHC1_LCTL、EPDC_SDCE04	ENET1_RXD0
ENET1_RXD1	GPIO2_IO01	ENET1_RDATA01、UART4_CTS_B、PWM2_OUT、CSI_DATA17、FLEXCAN1_RX、GPIO2_IO01、KPP_COL00、USDHC2_LCTL、EPDC_SDCE05	ENET1_RXD1
ENET1_RXER	GPIO2_IO07	NET1_RX_ER、UART7_RTS_B、PWM8_OUT、CSI_DATA23、EIM_CRE、GPIO2_IO07、KPP_COL03、GPT1_CAPTURE2、EPDC_SDOEZ	ENET1_RXER
ENET1_CRS_DV（ENET1_RX_EN）	GPIO2_IO02	ENET1_RX_EN、UART5_RTS_B、CSI_DATA18、FLEXCAN2_TX、GPIO2_IO02、KPP_ROW01、USDHC1_VSELECT、EPDC_SDCE06	ENET1_CRS_DV
ENET1_TX_CLK	GPIO2_IO06	ENET1_TX_CLK、UART7_CTS_B、PWM7_OUT、CSI_DATA22、ENET1_REF_CLK1、GPIO2_IO06、KPP_ROW03、GPT1_CLK、EPDC_SDOED	ENET1_TX_CLK

4.2 U-boot 裁剪 ENET1

4.2.1 源码编译

uboot 中使能了 ENET1，因此要裁剪掉 uboot 中的 ENET1 相关配置。

解压一份出厂 uboot 源码，执行 build.sh 脚本编译一遍源码。

4.2.2 修改 mx6ullevk.h

打开 uboot 源码的配置头文件 include/configs/mx6ullevk.h

搜索 CONFIG_FEC_ENET_DEV ，此宏用于 uboot 选择哪个网口，默认如下，以 V2.2 版本的阿尔法开发板和 V1.7 版本的 MINI 板出厂源码为例，ENET1 的 PHY 地址为 0x0，ENET2 的 PHY 地址为 0x1。

```
342  #define CONFIG_FEC_ENET_DEV     1
343
344  #if (CONFIG_FEC_ENET_DEV == 0)
345  #define IMX_FEC_BASE            ENET_BASE_ADDR
346  #define CONFIG_FEC_MXC_PHYADDR          0x0
347  #define CONFIG_FEC_XCV_TYPE             RMII
348  #elif (CONFIG_FEC_ENET_DEV == 1)
349  #define IMX_FEC_BASE            ENET2_BASE_ADDR
350  #define CONFIG_FEC_MXC_PHYADDR      0x1
351  #define CONFIG_FEC_XCV_TYPE     RMII
352  #endif
353  #define CONFIG_ETHPRIME         "FEC"
```

以 V2.4 版本的 ATK-DL6Y2C 开发板和 V2.0 版本的 ATK-DL6Y2CM 开发板为例，ENET1 的 PHY 地址为 0x2，ENET2 的 PHY 地址为 0x1。

```
343  #define CONFIG_FEC_ENET_DEV     1
344
345  #if (CONFIG_FEC_ENET_DEV == 0)
346  #define IMX_FEC_BASE            ENET_BASE_ADDR
347  #define CONFIG_FEC_MXC_PHYADDR          0x2
348  /* alientek imx6ull alpha board version <= 2.2, mini board <= 1.8, CONFIG_FEC_MXC_PHYADDR = 0x0 */
349  /* #define CONFIG_FEC_MXC_PHYADDR          0x0 */
350  #define CONFIG_FEC_XCV_TYPE             RMII
351  #elif (CONFIG_FEC_ENET_DEV == 1)
352  #define IMX_FEC_BASE            ENET2_BASE_ADDR
353  #define CONFIG_FEC_MXC_PHYADDR      0x1
354  #define CONFIG_FEC_XCV_TYPE     RMII
355  #endif
356  #define CONFIG_ETHPRIME         "FEC"
```

将 ENET1 的配置注释，只保留 ENET2 的配置。CONFIG_FEC_ENET_DEV 定义为 1。

注意将#elif (CONFIG_FEC_ENET_DEV == 1)改为#if (CONFIG_FEC_ENET_DEV == 1)

修改后如下：

```
#define CONFIG_FEC_ENET_DEV     1

/* #if (CONFIG_FEC_ENET_DEV == 0)
#define IMX_FEC_BASE            ENET_BASE_ADDR
#define CONFIG_FEC_MXC_PHYADDR          0x0
#define CONFIG_FEC_XCV_TYPE             RMII */
#if (CONFIG_FEC_ENET_DEV == 1)
#define IMX_FEC_BASE            ENET2_BASE_ADDR
#define CONFIG_FEC_MXC_PHYADDR      0x1
#define CONFIG_FEC_XCV_TYPE     RMII
#endif
#define CONFIG_ETHPRIME         "FEC"
```

保存修改好的文件。

4.2.3 修改 mx6ullevk.c

打开 uboot 源码的配置文件 board/freescale/mx6ullevk/mx6ullevk.c

搜索 ENET1，将相关 IO 屏蔽。

```
95
96    /* #define ENET1_RESET_IMX_GPIO_NR(5, 7) */
```

fec1_pads 是 ENET1 的 IO 复用配置参数，注释掉（里面有注释内容的，可以先将注释内容删除或者改为//注释）

```
/* static iomux_v3_cfg_t const fec1_pads[] = {
    MX6_PAD_GPIO1_IO06__ENET1_MDIO | MUX_PAD_CTRL(MDIO_PAD_CTRL),
    MX6_PAD_GPIO1_IO07__ENET1_MDC | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_TX_DATA0__ENET1_TDATA00 | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_TX_DATA1__ENET1_TDATA01 | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_TX_EN__ENET1_TX_EN | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_TX_CLK__ENET1_REF_CLK1 | MUX_PAD_CTRL(ENET_CLK_PAD_CTRL),
    MX6_PAD_ENET1_RX_DATA0__ENET1_RDATA00 | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_RX_DATA1__ENET1_RDATA01 | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_RX_ER__ENET1_RX_ER | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_ENET1_RX_EN__ENET1_RX_EN | MUX_PAD_CTRL(ENET_PAD_CTRL),
    MX6_PAD_SNVS_TAMPER7__GPIO5_IO07 | MUX_PAD_CTRL(NO_PAD_CTRL),      // ETH1 RESET PIN
}; */
```

函数 setup_iomux_fec 就是根据 fec1_pads 和 fec2_pads 这两个网络 IO 配置数组来初始化 I.MX6ULL 的网络 IO。

注释掉 fec_id == 0 判断相关的内容。

```
667  static void setup_iomux_fec(int fec_id)
668  {
669  /*  if (fec_id == 0){
670          imx_iomux_v3_setup_multiple_pads(fec1_pads,
671                                           ARRAY_SIZE(fec1_pads));
672          gpio_direction_output(ENET1_RESET, 1);
673          gpio_set_value(ENET1_RESET, 0);
674      mdelay(20);
675          gpio_set_value(ENET1_RESET, 1);
676      } else { */
677          imx_iomux_v3_setup_multiple_pads(fec2_pads,
678                                           ARRAY_SIZE(fec2_pads));
679          gpio_direction_output(ENET2_RESET, 1);
680          gpio_set_value(ENET2_RESET, 0);
681      mdelay(20);
682          gpio_set_value(ENET2_RESET, 1);
683  //  }
684  }
```

保存修改好的文件。

4.2.4 uboot 指令 gpio 测试

执行 build.sh 脚本编译出厂 uboot 源码，生成 uboot 的 bin 文件和 imx 文件，如下图。

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls
u-boot-imx6ull-14x14-ddr256-emmc.bin  u-boot-imx6ull-14x14-ddr256-nand-sd.bin  u-boot-imx6ull-14x14-ddr512-nand.bin
u-boot-imx6ull-14x14-ddr256-emmc.imx  u-boot-imx6ull-14x14-ddr256-nand-sd.imx  u-boot-imx6ull-14x14-ddr512-nand.imx
u-boot-imx6ull-14x14-ddr256-nand.bin  u-boot-imx6ull-14x14-ddr512-emmc.bin     u-boot-imx6ull-14x14-ddr512-nand-sd.bin
u-boot-imx6ull-14x14-ddr256-nand.imx  u-boot-imx6ull-14x14-ddr512-emmc.imx     u-boot-imx6ull-14x14-ddr512-nand-sd.imx
```

根据核心板版本，选择对应的 u-boot-xxxxx.bin 文件，使用 imxdownload 软件烧写到 SD 卡。

eMMC 版本核心板选择 u-boot-imx6ull-14x14-ddr512-emmc.bin。

NAND Flash 版本核心板选择 u-boot-imx6ull-14x14-ddr256-nand-sd.bin。

烧写示范：（NAND 烧写时，烧写指令最后要加 -256m）

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sdb  /dev/sdb1
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ./imxdownload u-boot-imx6ull-14x14-ddr512-emmc.bin /dev/sdb
I.MX6ULL bin download software
Edit by:zuozhongkai
Date:2019/6/10
Version:V1.1
log:V1.0 initial version,just support 512MB DDR3
    V1.1 and support 256MB DDR3
file u-boot-imx6ull-14x14-ddr512-emmc.bin size = 355124Bytes
Board DDR SIZE: 512MB
Delete Old load.imx
Create New load.imx
Download load.imx to /dev/sdb  ......
[sudo] alientek 的密码:
记录了699+1 的读入
记录了699+1 的写出
358196 bytes (358 kB, 350 KiB) copied, 2.25007 s, 159 kB/s
```

将烧写好 uboot 的 SD 卡接到 MINI 开发板上，拨码开始 SD 模式启动，在 uboot 命令行终端进行测试。

进入 uboot 命令行，用 gpio 相关指令可以进行 IO 的上下拉测试。gpio 指令说明：

```
=> gpio
gpio - query and control gpio pins

Usage:
gpio <input|set|clear|toggle> <pin>
    - input/set/clear/toggle the specified pin
gpio status [-a] [<bank> | <pin>]   - show [all/claimed] GPIOs
```

示例：

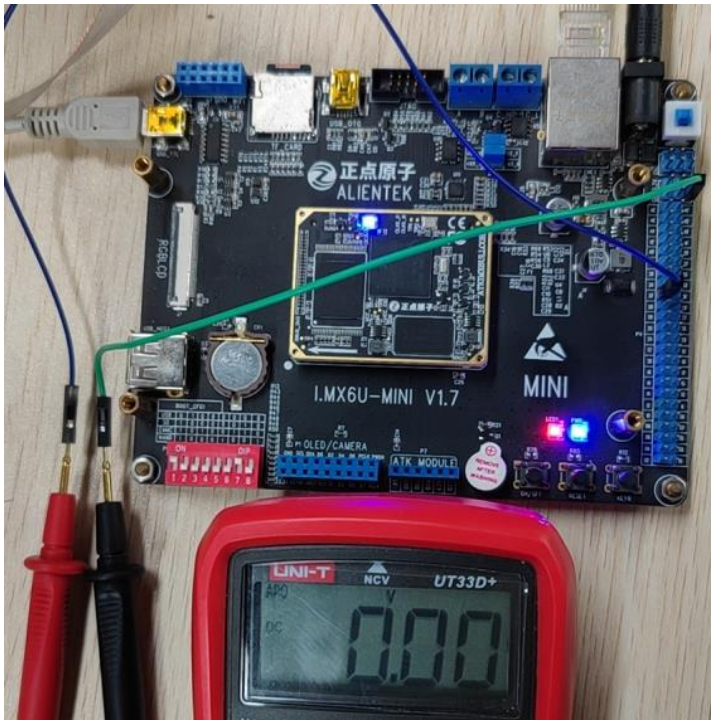
将 GPIO35 拉低。

```
gpio clear 35
```

将 GPIO35 拉高。

```
gpio set 35
```

硬件连接：以 ENET1_TXD0 管脚为例。启动 MINI 开发板，将万用表调到电压档位，黑表笔接地，红表笔接 ENET1_TXD0。如下图所示：



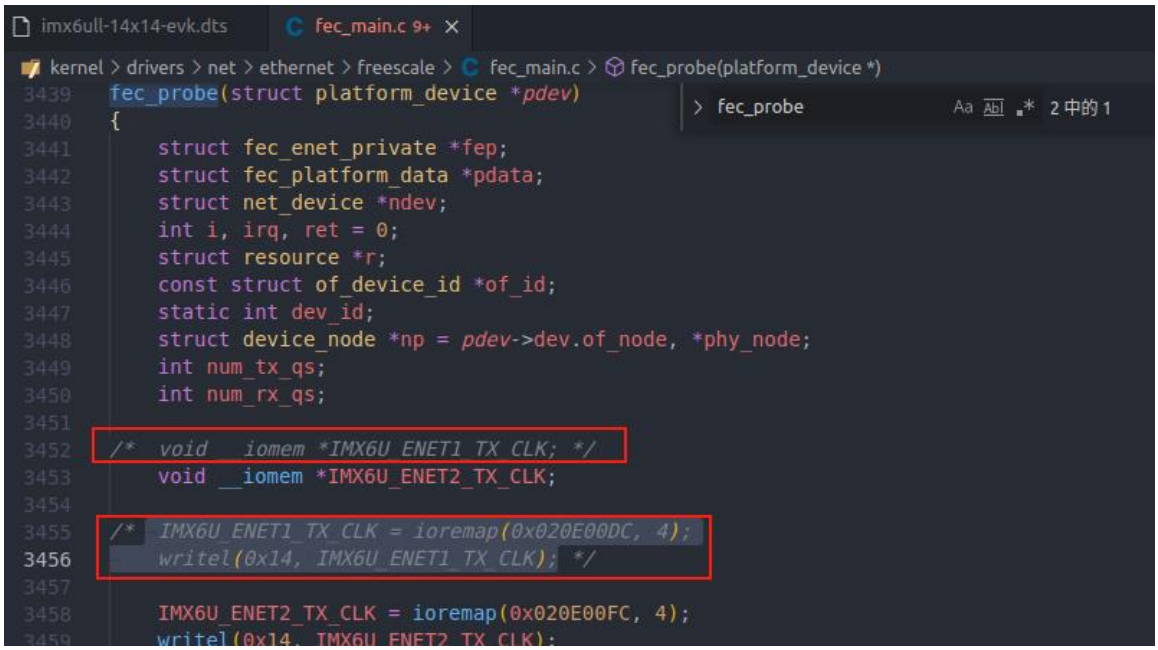
使用 gpio 指令依次对 ENET1 相关 IO 测试。结果如下：

原理图管脚	GPIO	GPIO 编号	高/低电平	MINI 板对应管脚
ENET1_TXD0	GPIO2_IO03	35	3.3V/0V	PIN32
ENET1_TXD1	GPIO2_IO04	36	3.3V/0V	PIN39
ENET1_TXEN	GPIO2_IO05	37	3.3V	PIN38
ENET1_RXD0	GPIO2_IO00	32	3.3V/0V	PIN37
ENET1_RXD1	GPIO2_IO01	33	3.3V/0V	PIN34
ENET1_RXER	GPIO2_IO07	39	3.3V/0V	PIN41
ENET1_CRS_DV	GPIO2_IO02	34	3.3V/0V	PIN36
ENET1_TX_CLK	GPIO2_IO06	38	3.3V/0V	PIN40

4.3 内核裁剪 ENET1 驱动

4.3.1 修改内核源码

打开出厂内核源码，修改内核文件 drivers/net/ethernet/freescale/fec_main.c 找到 fec_probe 函数，注释掉 ENET1_TX_CLK 相关配置。



保存修改好的文件，后续要编译内核更新一下。

4.3.2 修改设备树文件

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts，屏蔽 fec1 节点相关内容，修改完如下：

```
/* &fec1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet1
                 &pinctrl_fec1_reset>;
    phy-mode = "rmii";
    phy-handle = <&ethphy0>;
    phy-reset-gpios = <&gpio5 7 GPIO_ACTIVE_LOW>;
    phy-reset-duration = <200>;
    status = "okay";
}; */
```

搜索 pinctrl_enet1 节点，屏蔽掉 ENET1 的管脚功能，修改完如下：

```
489  /*      pinctrl_enet1: enet1grp {
490              fsl,pins = <
491                  MX6UL_PAD_ENET1_RX_EN__ENET1_RX_EN  0x1b0b0
492                  MX6UL_PAD_ENET1_RX_ER__ENET1_RX_ER  0x1b0b0
493                  MX6UL_PAD_ENET1_RX_DATA0__ENET1_RDATA00 0x1b0b0
494                  MX6UL_PAD_ENET1_RX_DATA1__ENET1_RDATA01 0x1b0b0
495                  MX6UL_PAD_ENET1_TX_EN__ENET1_TX_EN  0x1b0b0
496                  MX6UL_PAD_ENET1_TX_DATA0__ENET1_TDATA00 0x1b0b0
497                  MX6UL_PAD_ENET1_TX_DATA1__ENET1_TDATA01 0x1b0b0
498                  MX6UL_PAD_ENET1_TX_CLK__ENET1_REF_CLK1  0x4001B009
499              >;
500          }; */
```

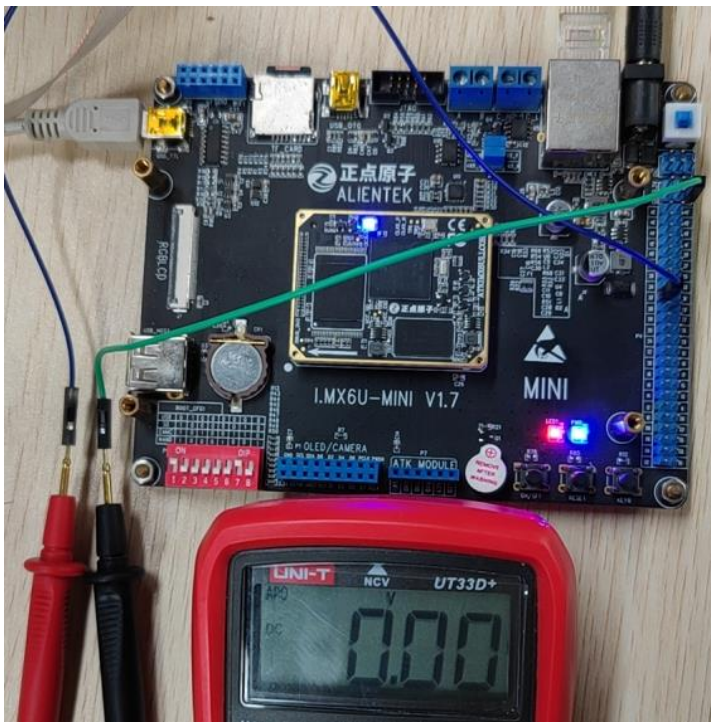
保存文件，执行编译内核和设备树文件。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make zImage -j16
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

将编译生成的内核、设备树文件替换到开发板上启动。

4.4 命令测试 GPIO 输出

以 ENET1_TXD0 管脚为例。启动 MINI 开发板，将万用表调到电压档位，黑表笔接地，红表笔接 ENET1_TXD0。如下图所示：



进入开发板系统终端，进行 GPIO 操作。示例：

```
cd /sys/class/gpio/
echo 35 > export
echo low > gpio35/direction
echo high > gpio35/direction
```

测试 ENET1_TXD0 管脚可以拉高/拉低输出。

依次测试其他 IO，可以直接测试的 IO 有 8 个

原理图管脚	GPIO	GPIO 编号	高/低电平	MINI 板对应管脚
ENET1_TXD0	GPIO2_IO03	35	3.3V/0V	PIN32
ENET1_TXD1	GPIO2_IO04	36	3.3V/0V	PIN39
ENET1_TXEN	GPIO2_IO05	37	3.3V/0V	PIN38
ENET1_RXD0	GPIO2_IO00	32	3.3V/0V	PIN37
ENET1_RXD1	GPIO2_IO01	33	3.3V/0V	PIN34
ENET1_RXER	GPIO2_IO07	39	3.3V/0V	PIN41
ENET1_CRS_DV	GPIO2_IO02	34	3.3V/0V	PIN36
ENET1_TX_CLK	GPIO2_IO06	38	3.3V/0V	PIN40

4.5 程序测试 GPIO 输出\输入\中断

参考本文 3.4 小节。

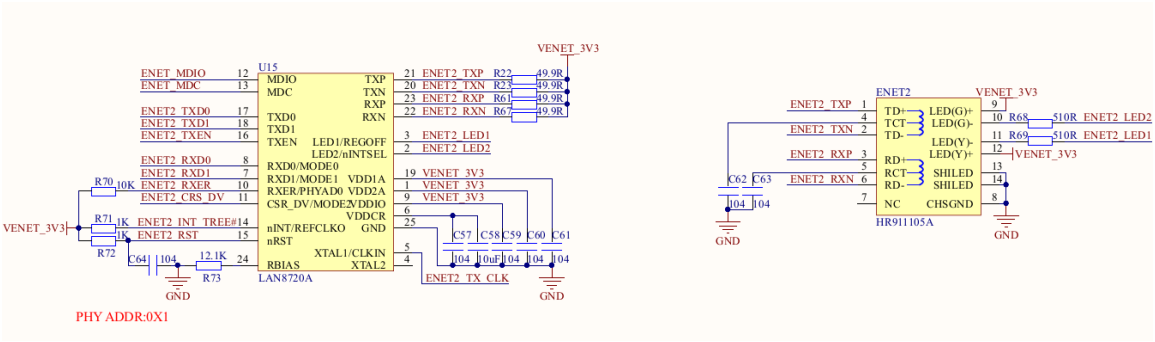
第五章 裁剪 ENET2

ENET2 的裁剪，整体流程和 ENET1 的裁剪是一样的，开发板上都保留了 ENET2，不好直接测试效果，需要使用核心板自行设计底板，从核心板引出 IO 直接测试。

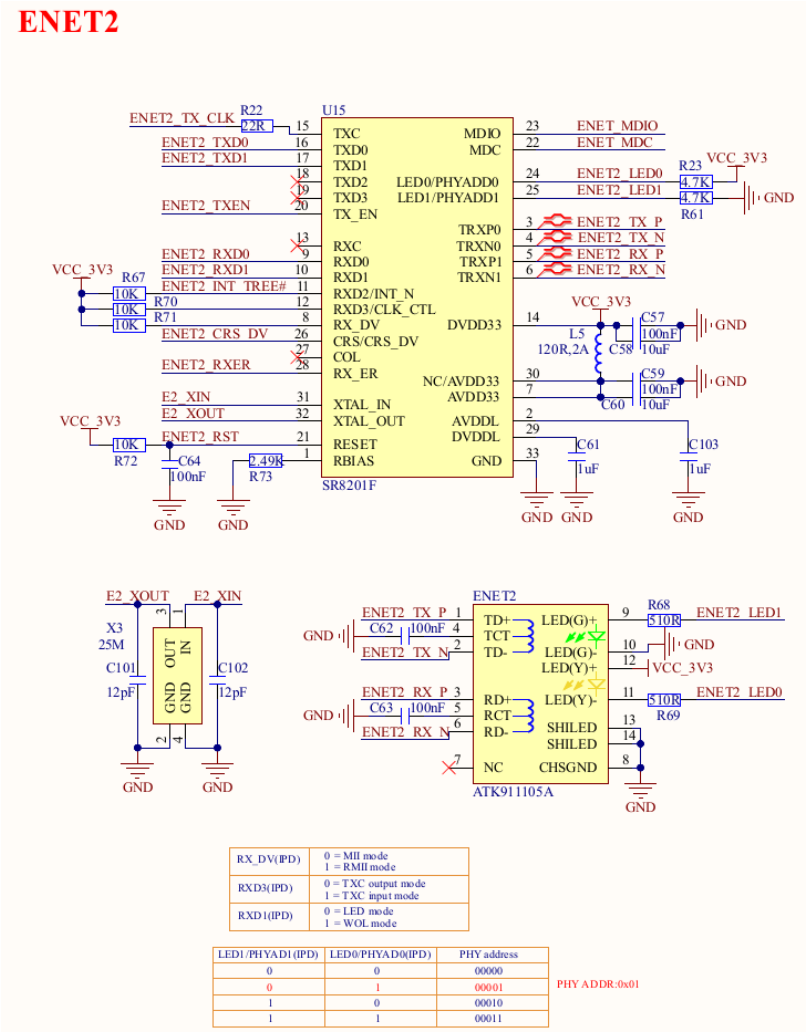
本节只单独裁剪 ENET2，保留 ENET1 配置。

5.1 驱动及管脚定义

以 V2.2 阿尔法开发板为例，使用的 PHY 芯片是 LAN8720A，开发板 ENET2 接口原理图如下：



以 V2.4 的 ATK-DL6Y2C 开发板为例，使用的 PHY 芯片是 SR820F，开发板 ENET2 原理图：



RX_DV(IPD)	0 = MII mode 1 = RMII mode
RXD3(IPD)	0 = TXC output mode 1 = TXC input mode
RXD1(IPD)	0 = LED mode 1 = WOL mode

LED1/PHYAD1(IPD)	LED0/PHYAD0(IPD)	PHY address
0	0	00000
0	1	00001
1	0	00010
1	1	00011

原理图管脚名	GPIO	可复用功能
ENET2_TXD0	GPIO2_IO11	ENET2_TDATA00、UART7_RX、I2C4_SDA、EIM_EB_B02、GPIO2_IO11、KPP_COL05、KPP_COL05
ENET2_TXD1	GPIO2_IO12	ENET2_TDATA01、UART8_TX、ECSPI4_SCLK、EIM_EB_B03、GPIO2_IO12、KPP_ROW06、USB_OTG2_PWR、EPDC_SDDO12
ENET2_TXEN	GPIO2_IO13	ENET2_TX_EN、UART8_RX、ECSPI4_MOSI、EIM_ACLK、GPIO2_IO13、KPP_COL06、USB_OTG2_OC、EPDC_SDDO13
ENET2_RXD0	GPIO2_IO08	ENET2_RDATA00、UART6_TX、I2C3_SCL、ENET1_MDIO、GPIO2_IO08、KPP_ROW04、USB_OTG1_PWR、EPDC_SDDO08
ENET2_RXD1	GPIO2_IO09	ENET2_RDATA01、UART6_RX、I2C3_SDA、ENET1_MDC、GPIO2_IO09、KPP_COL04、USB_OTG1_OC、EPDC_SDDO09
ENET2_RXER	GPIO2_IO15	ENET2_RX_ER、UART8_RTS_B、ECSPI4_SS0、EIM_ADDR25、GPIO2_IO15、KPP_COL07、WDOG1_WDOG_ANY、EPDC_SDDO15
ENET2_CRS_DV	GPIO2_IO10	ENET2_RX_EN、UART7_TX、I2C4_SCL、EIM_ADDR26、GPIO2_IO10、KPP_ROW05、ENET1_REF_CLK_25M、EPDC_SDDO10
ENET2_TX_CLK	GPIO2_IO14	ENET2_TX_CLK、UART8_CTS_B、ECSPI4_MISO、ENET2_REF_CLK2、GPIO2_IO14、KPP_ROW07、ANATOP_OTG2_ID、EPDC_SDDO14

5.2 U-boot 裁剪 ENET2

5.2.1 源码编译

uboot 中使能了 ENET2，因此要裁剪掉 uboot 中的 ENET2 相关配置。

解压一份出厂 uboot 源码，执行 build.sh 脚本编译一遍源码。

5.2.2 修改 mx6ullevk.h

打开 uboot 源码的配置头文件 inculde/configs/mx6ullevk.h

搜索 CONFIG_FEC_ENET_DEV ，此宏用于 uboot 选择哪个网口，默认如下：

对于 V2.2 版本的阿尔法开发板和 V1.7 版本的 MINI 开发板来说，ENET1 的 PHY 地址为 0x0，ENET2 的 PHY 地址为 0x1。

```
342  #define CONFIG_FEC_ENET_DEV     1
343
344  #if (CONFIG_FEC_ENET_DEV == 0)
345  #define IMX_FEC_BASE            ENET_BASE_ADDR
346  #define CONFIG_FEC_MXC_PHYADDR          0x0
347  #define CONFIG_FEC_XCV_TYPE             RMII
348  #elif (CONFIG_FEC_ENET_DEV == 1)
349  #define IMX_FEC_BASE            ENET2_BASE_ADDR
350  #define CONFIG_FEC_MXC_PHYADDR      0x1
351  #define CONFIG_FEC_XCV_TYPE     RMII
352  #endif
```

对于 V2.4 版本的 ATK-DL6Y2C 开发板和 V2.0 版本的 ATK-DL6Y2CM 开发板来说，ENET1 的 PHY 地址为 0x2，ENET2 的 PHY 地址为 0x1。

```
343  #define CONFIG_FEC_ENET_DEV     1
344
345  #if (CONFIG_FEC_ENET_DEV == 0)
346  #define IMX_FEC_BASE            ENET_BASE_ADDR
347  #define CONFIG_FEC_MXC_PHYADDR          0x2
348  /* alientek imx6ull alpha board version <= 2.2, mini board <= 1.8, CONFIG_FEC_MXC_PHYADDR = 0x0 */
349  /* #define CONFIG_FEC_MXC_PHYADDR          0x0 */
350  #define CONFIG_FEC_XCV_TYPE             RMII
351  #elif (CONFIG_FEC_ENET_DEV == 1)
352  #define IMX_FEC_BASE            ENET2_BASE_ADDR
353  #define CONFIG_FEC_MXC_PHYADDR      0x1
354  #define CONFIG_FEC_XCV_TYPE     RMII
355  #endif
356  #define CONFIG_ETHPRIME         "FEC"
```

将 ENET2 的配置删除，只保留 ENET1 的。CONFIG_FEC_ENET_DEV 定义为 0，配置默认使用 ENET1，以 V2.2 版本的阿尔法开发板和 V1.7 版本的 MINI 开发板出厂 uboot 源码为例，修改后如下：

```
342  #define CONFIG_FEC_ENET_DEV     0
343
344  #if (CONFIG_FEC_ENET_DEV == 0)
345  #define IMX_FEC_BASE            ENET_BASE_ADDR
346  #define CONFIG_FEC_MXC_PHYADDR          0x0
347  #define CONFIG_FEC_XCV_TYPE             RMII
348  #endif
349  #define CONFIG_ETHPRIME         "FEC"
350
```

以 V2.4 版本的 ATK-DL6Y2C 开发板和 V2.0 版本的 ATK-DL6Y2CM 开发板出厂 uboot 源码为例，修改后如下：

```
342  #define CONFIG_FEC_ENET_DEV     0
343
344  #if (CONFIG_FEC_ENET_DEV == 0)
345  #define IMX_FEC_BASE            ENET_BASE_ADDR
346  #define CONFIG_FEC_MXC_PHYADDR          0x2
347  #define CONFIG_FEC_XCV_TYPE             RMII
348  #endif
349  #define CONFIG_ETHPRIME         "FEC"
```

修改完后保存文件。

5.2.3 修改 mx6ullevk.c

打开 uboot 源码的配置文件 board/freescale/mx6ullevk/mx6ullevk.c
搜索 ENET2，将相关 IO 屏蔽。

```
97   /* #define ENET2_RESET_IMX_GPIO_NR(5, 8) */
```

fec2_pads 是 ENET2 的 IO 复用配置参数。

```
651  /* static iomux_v3_cfg_t const fec2_pads[] = {
652      MX6_PAD_GPIO1_IO06__ENET2_MDIO | MUX_PAD_CTRL(MDIO_PAD_CTRL),
653      MX6_PAD_GPIO1_IO07__ENET2_MDC | MUX_PAD_CTRL(ENET_PAD_CTRL),
654
655      MX6_PAD_ENET2_TX_DATA0__ENET2_TDATA00 | MUX_PAD_CTRL(ENET_PAD_CTRL),
656      MX6_PAD_ENET2_TX_DATA1__ENET2_TDATA01 | MUX_PAD_CTRL(ENET_PAD_CTRL),
657      MX6_PAD_ENET2_TX_CLK__ENET2_REF_CLK2 | MUX_PAD_CTRL(ENET_CLK_PAD_CTRL),
658      MX6_PAD_ENET2_TX_EN__ENET2_TX_EN | MUX_PAD_CTRL(ENET_PAD_CTRL),
659
660      MX6_PAD_ENET2_RX_DATA0__ENET2_RDATA00 | MUX_PAD_CTRL(ENET_PAD_CTRL),
661      MX6_PAD_ENET2_RX_DATA1__ENET2_RDATA01 | MUX_PAD_CTRL(ENET_PAD_CTRL),
662      MX6_PAD_ENET2_RX_EN__ENET2_RX_EN | MUX_PAD_CTRL(ENET_PAD_CTRL),
663      MX6_PAD_ENET2_RX_ER__ENET2_RX_ER | MUX_PAD_CTRL(ENET_PAD_CTRL),
664      MX6_PAD_SNVS_TAMPER8__GPIO5_IO08 | MUX_PAD_CTRL(NO_PAD_CTRL),
665  };
666   */
```

函数 setup_iomux_fec 就是根据 fec1_pads 和 fec2_pads 这两个网络 IO 配置数组来初始化 I.MX6ULL 的网络 IO。

```
667  static void setup_iomux_fec(int fec_id)
668  {
669      if (fec_id == 0){
670          imx_iomux_v3_setup_multiple_pads(fec1_pads,
671                                           ARRAY_SIZE(fec1_pads));
672          gpio_direction_output(ENET1_RESET, 1);
673          gpio_set_value(ENET1_RESET, 0);
674      mdelay(20);
675          gpio_set_value(ENET1_RESET, 1);
676      }
677  /*  else {
678          imx_iomux_v3_setup_multiple_pads(fec2_pads,
679                                           ARRAY_SIZE(fec2_pads));
680          gpio_direction_output(ENET2_RESET, 1);
681          gpio_set_value(ENET2_RESET, 0);
682      mdelay(20);
683          gpio_set_value(ENET2_RESET, 1);
684      } */
685  }
```

保存修改好的文件。

5.2.4 uboot 指令 gpio 测试

执行 build.sh 脚本编译出厂 uboot 源码，生成 uboot 的 bin 文件和 imx 文件，如下图。

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls
u-boot-imx6ull-14x14-ddr256-emmc.bin  u-boot-imx6ull-14x14-ddr256-nand-sd.bin  u-boot-imx6ull-14x14-ddr512-nand.bin
u-boot-imx6ull-14x14-ddr256-emmc.imx  u-boot-imx6ull-14x14-ddr256-nand-sd.imx  u-boot-imx6ull-14x14-ddr512-nand.imx
u-boot-imx6ull-14x14-ddr256-nand.bin  u-boot-imx6ull-14x14-ddr512-emmc.bin     u-boot-imx6ull-14x14-ddr512-nand-sd.bin
u-boot-imx6ull-14x14-ddr256-nand.imx  u-boot-imx6ull-14x14-ddr512-emmc.imx     u-boot-imx6ull-14x14-ddr512-nand-sd.imx
```

根据核心板版本，选择对应的 u-boot-xxxxx.bin 文件，使用 imxdownload 软件烧写到 SD 卡。

eMMC 版本核心板选择 u-boot-imx6ull-14x14-ddr512-emmc.bin。

NAND Flash 版本核心板选择 u-boot-imx6ull-14x14-ddr256-nand-sd.bin。

烧写示范：（NAND 烧写时，烧写指令最后要加 -256m）

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sdb  /dev/sdb1
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ./imxdownload u-boot-imx6ull-14x14-ddr512-emmc.bin /dev/sdb
I.MX6ULL bin download software
Edit by:zuozhongkai
Date:2019/6/10
Version:V1.1
log:V1.0 initial version,just support 512MB DDR3
    V1.1 and support 256MB DDR3
file u-boot-imx6ull-14x14-ddr512-emmc.bin size = 355124Bytes
Board DDR SIZE: 512MB
Delete Old load.imx
Create New load.imx
Download load.imx to /dev/sdb  ......
[sudo] alientek 的密码:
记录了699+1 的读入
记录了699+1 的写出
358196 bytes (358 kB, 350 KiB) copied, 2.25007 s, 159 kB/s
```

将烧写好 uboot 的 SD 卡接到 MINI 开发板上，拨码开始 SD 模式启动，在 uboot 命令行终端进行测试。

进入 uboot 命令行，用 gpio 相关指令可以进行 IO 的上下拉测试。gpio 指令说明：

```
=> gpio
gpio - query and control gpio pins

Usage:
gpio <input|set|clear|toggle> <pin>
    - input/set/clear/toggle the specified pin
gpio status [-a] [<bank> | <pin>]   - show [all/claimed] GPIOs
```

示例：将 GPIO43 拉低。

```
gpio clear 43
```

将 GPIO43 拉高。

```
gpio set 43
```

硬件连接：请根据自制的底板硬件，用万用表对引出的 ENET2 相关 IO 进行测试。

使用 gpio 指令依次对 ENET2 相关 IO 测试。结果如下：

原理图管脚名	GPIO	GPIO 编号	高/低电平
ENET2_TXD0	GPIO2_IO11	43	3.3V/0V
ENET2_TXD1	GPIO2_IO12	44	3.3V/0V
ENET2_TXEN	GPIO2_IO13	45	3.3V/0V
ENET2_RXD0	GPIO2_IO08	40	3.3V/0V
ENET2_RXD1	GPIO2_IO09	41	3.3V/0V
ENET2_RXER	GPIO2_IO15	47	3.3V/0V
ENET2_CRS_DV	GPIO2_IO10	42	3.3V/0V
ENET2_TX_CLK	GPIO2_IO14	46	3.3V/0V

5.3 内核裁剪 ENET2 驱动

5.3.1 修改内核源码

打开出厂内核源码，修改内核文件 drivers/net/ethernet/freescale/fec_main.c

找到 fec_probe 函数，注释掉 ENET2_TX_CLK 相关配置。

```
3451
3452    void __iomem *IMX6U_ENET1_TX_CLK;
3453    /* void __iomem *IMX6U_ENET2_TX_CLK; */
3454
3455    IMX6U_ENET1_TX_CLK = ioremap(0x020E00DC, 4);
3456    writel(0x14, IMX6U_ENET1_TX_CLK);
3457
3458 /* IMX6U_ENET2_TX_CLK = ioremap(0x020E00FC, 4);
3459    writel(0x14, IMX6U_ENET2_TX_CLK); */
3460
3461    fec_enet_get_queue_num(pdev, &num_tx_qs, &num_rx_qs);
```

保存修改好的文件，后续要编译内核更新一下。

5.3.2 修改设备树文件

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts，屏蔽 fec2 节点相关内容，修改完如下：

```
/* &fec2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet2
                 &pinctrl_fec2_reset>;
    phy-mode = "rmii";
    phy-handle = <&ethphy1>;
    phy-reset-gpios = <&gpio5 8 GPIO_ACTIVE_LOW>;
    phy-reset-duration = <200>;
    status = "okay";

    mdio {
        #address-cells = <1>;
        #size-cells = <0>;

        ethphy0: ethernet-phy@2 {
            compatible = "ethernet-phy-ieee802.3-c22";
            reg = <0>;
        };

        ethphy1: ethernet-phy@1 {
            compatible = "ethernet-phy-ieee802.3-c22";
            reg = <1>;
        };
    };
}; */
```

在 fec1 节点下添加 mdio 相关配置，确保 ENET1 功能正常。

```
    mdio {
        #address-cells = <1>;
        #size-cells = <0>;

        ethphy0: ethernet-phy@0 {
            compatible = "ethernet-phy-ieee802.3-c22";
            reg = <0>;
        };
    };
```

修改完如下：

```
&fec1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet1
                 &pinctrl_fec1_reset>;
    phy-mode = "rmii";
    phy-handle = <&ethphy0>;
    phy-reset-gpios = <&gpio5 7 GPIO_ACTIVE_LOW>;
    phy-reset-duration = <200>;
    status = "okay";

    mdio {
        #address-cells = <1>;
        #size-cells = <0>;

        ethphy0: ethernet-phy@0 {
            compatible = "ethernet-phy-ieee802.3-c22";
            reg = <0>;
        };
    };
};
```

搜索 pinctrl_enet2 节点，屏蔽掉 ENET2 的管脚功能，修改完如下：

```
/*      pinctrl_enet2: enet2grp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO07__ENET2_MDC     0x1b0b0
                MX6UL_PAD_GPIO1_IO06__ENET2_MDIO    0x1b0b0
                MX6UL_PAD_ENET2_RX_EN__ENET2_RX_EN  0x1b0b0
                MX6UL_PAD_ENET2_RX_ER__ENET2_RX_ER  0x1b0b0
                MX6UL_PAD_ENET2_RX_DATA0__ENET2_RDATA00 0x1b0b0
                MX6UL_PAD_ENET2_RX_DATA1__ENET2_RDATA01 0x1b0b0
                MX6UL_PAD_ENET2_TX_EN__ENET2_TX_EN  0x1b0b0
                MX6UL_PAD_ENET2_TX_DATA0__ENET2_TDATA00 0x1b0b0
                MX6UL_PAD_ENET2_TX_DATA1__ENET2_TDATA01 0x1b0b0
                MX6UL_PAD_ENET2_TX_CLK__ENET2_REF_CLK2  0x4001B009
            >;
        }; */
```

搜索 pinctrl_enet1 节点，在此 enet1 管脚复用节点中添加 MDIO 和 MDC 的相关配置。

```
MX6UL_PAD_GPIO1_IO07__ENET1_MDC     0x1b0b0
MX6UL_PAD_GPIO1_IO06__ENET1_MDIO    0x1b0b0
```

修改完如下：

```
pinctrl_enet1: enet1grp {
    fsl,pins = <
        MX6UL_PAD_ENET1_RX_EN__ENET1_RX_EN  0x1b0b0
        MX6UL_PAD_ENET1_RX_ER__ENET1_RX_ER  0x1b0b0
        MX6UL_PAD_ENET1_RX_DATA0__ENET1_RDATA00 0x1b0b0
        MX6UL_PAD_ENET1_RX_DATA1__ENET1_RDATA01 0x1b0b0
        MX6UL_PAD_ENET1_TX_EN__ENET1_TX_EN  0x1b0b0
        MX6UL_PAD_ENET1_TX_DATA0__ENET1_TDATA00 0x1b0b0
        MX6UL_PAD_ENET1_TX_DATA1__ENET1_TDATA01 0x1b0b0
        MX6UL_PAD_ENET1_TX_CLK__ENET1_REF_CLK1  0x4001B009
        MX6UL_PAD_GPIO1_IO07__ENET1_MDC 0x1b0b0
        MX6UL_PAD_GPIO1_IO06__ENET1_MDIO 0x1b0b0
    >;
};
```

保存文件，执行编译内核和设备树文件。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make zImage -j16
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

将编译生成的内核、设备树文件替换到开发板上启动。

5.4 命令测试 GPIO 输出

硬件连接：请根据自制的底板硬件，用万用表对引出的 ENET2 相关 IO 进行测试。

进入开发板系统终端，进行 GPIO 操作。示例：

```
cd /sys/class/gpio/
echo 43 > export
echo low > gpio43/direction
echo high > gpio43/direction
```

测试 ENET2_TXD0 管脚可以拉高/拉低输出。

依次测试其他 IO，可以直接测试的 IO 有 7 个

原理图管脚名	GPIO	GPIO 编号	高/低电平
ENET2_TXD0	GPIO2_IO11	43	3.3V/0V
ENET2_TXD1	GPIO2_IO12	44	3.3V/0V
ENET2_TXEN	GPIO2_IO13	45	3.3V/0V
ENET2_RXD0	GPIO2_IO08	40	3.3V/0V
ENET2_RXD1	GPIO2_IO09	41	3.3V/0V
ENET2_RXER	GPIO2_IO15	47	3.3V/0V
ENET2_CRS_DV	GPIO2_IO10	42	3.3V/0V

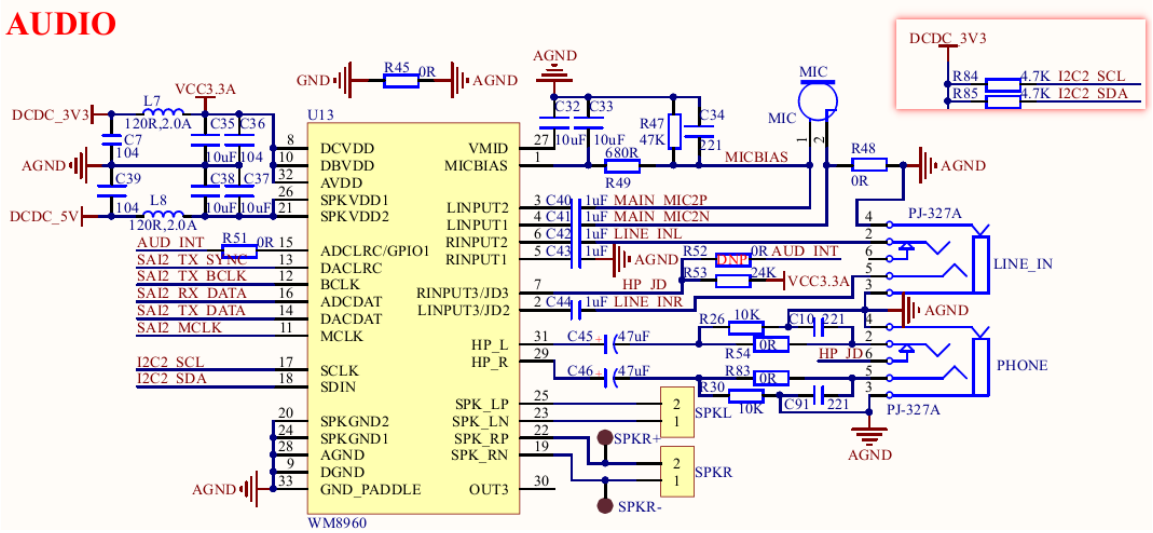
5.5 程序测试 GPIO 输出\输入\中断

参考本文 3.4 小节。

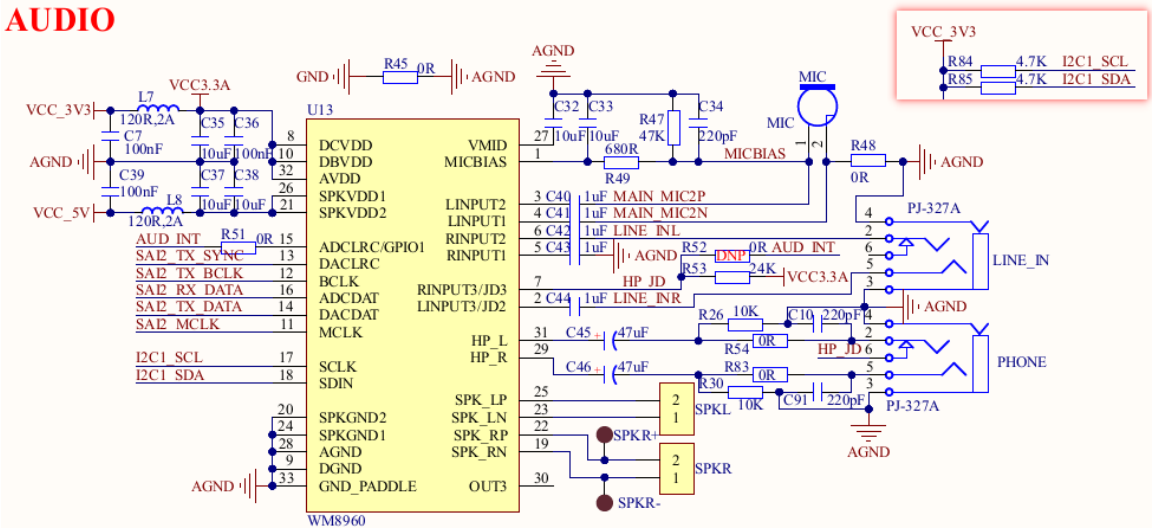
第六章 裁剪 AUDIO/JTAG

6.1 驱动及管脚定义

以 V2.2 版本阿尔法开发板为例，音频 I2C 音频接口为 I2C2，相关原理图如下：



以 V2.4 版本 ATK-DL6Y2C 开发板为例，音频 I2C 接口为 I2C1，相关原理图如下：



MINI 板上去掉了音频芯片相关的元器件，复用为 JTAG 功能。本小节的裁剪测试，用 MINI 板方便些，用阿尔法板就需要自行去除相关元器件，从核心板直接引出 IO 测试。

MINI 板相关原理图如下：

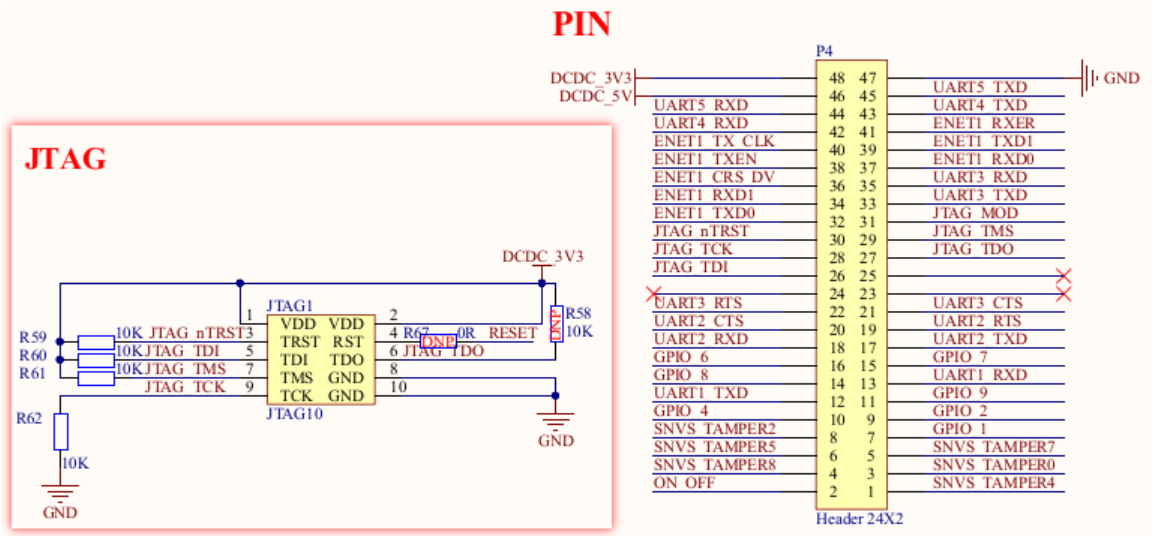


图 6.1-1 MINI 板引出 IO 和 JTAG 原理图

阿尔法原理图管脚	GPIO	可复用功能	MINI 板原理图管脚
AUD_INT	GPIO5_IO04	SNVS_TAMPER4/GPIO5_IO04	SNVS_TAMPER4
SAI2_TX_SYNC	GPIO1_IO12	SJC_TDO、GPT2_CAPTURE2、SAI2_TX_SYNC、CCM_CLKO2、CCM_STOP、GPIO1_IO12、MQS_RIGHT、EPIT2_OUT	JTAG_TDO
SAI2_TX_BCLK	GPIO1_IO13	SJC_TDI、GPT2_COMPARE1、SAI2_TX_BCLK、PWM6_OUT、GPIO1_IO13、MQS_LEFT、SIM1_POWER_FAIL	JTAG_TDI
SAI2_RX_DATA	GPIO1_IO14	SJC_TCK、GPT2_COMPARE2、SAI2_RX_DATA、PWM7_OUT、GPIO1_IO14、SIM2_POWER_FAIL	JTAG_TCK
SAI2_TX_DATA	GPIO1_IO15	SJC_TRSTB、GPT2_COMPARE3、SAI2_TX_DATA、PWM8_OUT、GPIO1_IO15、CAAM_RNG_OSC_OBS	JTAG_nTRST
SAI2_MCLK	GPIO1_IO11	SJC_TMS、GPT2_CAPTURE1、SAI2_MCLK、CCM_CLKO1、CCM_WAIT、GPIO1_IO11、SDMA_EXT_EVENT01、EPIT1_OUT	JTAG_TMS

6.2 内核裁剪 AUDIO 驱动

6.2.1 修改 menuconfig 配置

进入出厂内核源码目录下，打开 menuconfig 配置。

```
source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig -j 16
make menuconfig
```

修改 menuconfig 配置，如下：

```
Device Drivers  --->
    <*> Sound card support  --->
        <*>  Advanced Linux Sound Architecture  --->
            <*>  ALSA for SoC audio support  --->
                SoC Audio for Freescale CPUs  --->
                < > Asynchronous Sample Rate Converter (ASRC) module support
                < > Synchronous Audio Interface (SAI) module support
                < > Synchronous Serial Interface module (SSI) support
                < > Sony/Philips Digital Interface (S/PDIF) module support
                < > Enhanced Serial Audio Interface (ESAI) module support
                < > Digital Audio Mux module support
                < > SoC Audio for Freescale i.MX CPUs
```

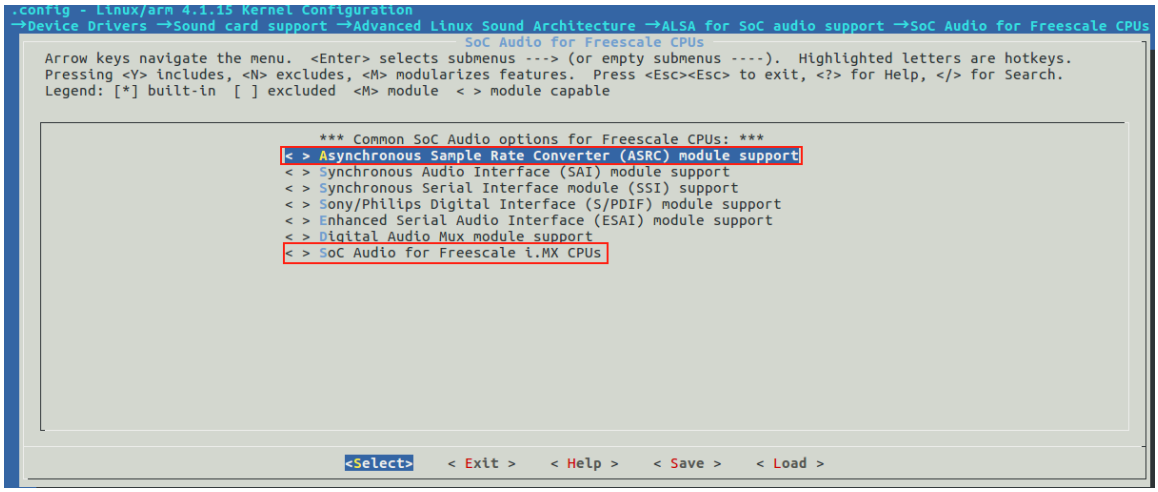


图 6.2-1 menuconfig 配置

Save 另存为./arch/arm/configs/del_audio_imx_v7_defconfig 配置文件。

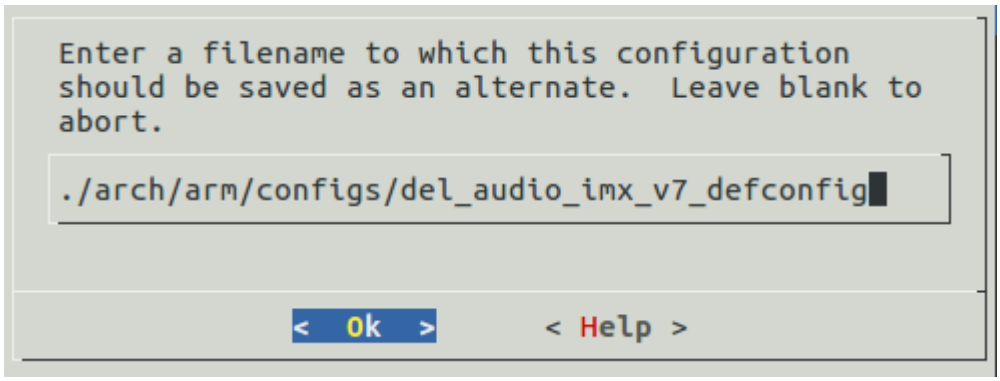


图 6.2-2 另存为配置文件

双击键盘 Esc 键退出 menuconfig 配置。

6.2.2 修改设备树文件

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件进行修改。

搜索关键词 sai。

注释掉 sound 节点（可以先删除已有的注释）。

```
/*  sound {
        compatible = "fsl,imx6ul-evk-wm8960",
                     "fsl,imx-audio-wm8960";
        model = "wm8960-audio";
        cpu-dai = <&sai2>;
        audio-codec = <&codec>;
        asrc-controller = <&asrc>;
        codec-master;
        gpr = <&gpr 4 0x100000 0x100000>;

        hp-det = <3 0>;

        audio-routing =
            "Headphone Jack", "HP_L",
            "Headphone Jack", "HP_R",
            "Ext Spk", "SPK_LP",
            "Ext Spk", "SPK_LN",
            "Ext Spk", "SPK_RP",
            "Ext Spk", "SPK_RN",
            "LINPUT2", "Mic Jack",
            "LINPUT3", "Mic Jack",
            "RINPUT1", "Main MIC",
            "RINPUT2", "Main MIC",
            "Mic Jack", "MICB",
            "Main MIC", "MICB",
            "CPU-Playback", "ASRC-Playback",
            "Playback", "CPU-Playback",
            "ASRC-Capture", "CPU-Capture",
            "CPU-Capture", "Capture";
    }; */
```

V2.2 版本阿尔法出厂内核源码注释掉&i2c2 节点下的 codec 信息。

```
/*  codec: wm8960@1a {
        compatible = "wlf,wm8960";
        reg = <0x1a>;
        clocks = <&clks IMX6UL_CLK_SAI2>;
        clock-names = "mclk";
        wlf,shared-lrclk;
    }; */
```

V2.4 版本 ATK-DL6Y2C 开发板出厂内核源码注释掉&i2c1 节点下的 codec 信息。

```
/*          codec: wm8960@1a {
                compatible = "wlf,wm8960";
                reg = <0x1a>;
                clocks = <&clks IMX6UL_CLK_SAI2>;
                clock-names = "mclk";
                wlf,shared-lrclk;
            };  */
```

注释掉 pinctrl_sai2 管脚复用功能。

```
/*      pinctrl_sai2: sai2grp {
            fsl,pins = <
                MX6UL_PAD_JTAG_TDI__SAI2_TX_BCLK    0x17088
                MX6UL_PAD_JTAG_TDO__SAI2_TX_SYNC    0x17088
                MX6UL_PAD_JTAG_TRST_B__SAI2_TX_DATA 0x11088
                MX6UL_PAD_JTAG_TCK__SAI2_RX_DATA    0x11088
                MX6UL_PAD_JTAG_TMS__SAI2_MCLK       0x17088
            >;
        }; */
```

注释掉 pinctrl_sai2_hp_det_b 管脚配置信息。

```
/*              pinctrl_sai2_hp_det_b: sai2_hp_det_grp {
                    fsl,pins = <
                        MX6ULL_PAD_SNVS_TAMPER4__GPIO5_IO04   0x17059
                    >;
                }; */
```

注释掉&sai2 节点信息。

```
/* &sai2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_sai2
                 &pinctrl_sai2_hp_det_b>;

    assigned-clocks = <&clks IMX6UL_CLK_SAI2_SEL>,
                      <&clks IMX6UL_CLK_SAI2>;
    assigned-clock-parents = <&clks IMX6UL_CLK_PLL4_AUDIO_DIV>;
    assigned-clock-rates = <0>, <11289600>;

    status = "okay";
}; */
```

检查是否有修改遗漏的地方，保存修改好的设备树文件。

6.2.3 编译内核和设备树

至此内核和设备树裁剪 audio 外设步骤完成。

之前我们保存了配置文件为 del_audio_imx_v7_defconfig，这里将用到此文件进行内核编译。设备树文件根据核心板型号和屏幕型号选择，没有屏幕默认用 4.3-480x272 的设备树。

```
source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make del_audio_imx_v7_defconfig
make zImage -j 16
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

```
alientek@ubuntu16:~/imx6ull/kernel$ source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
alientek@ubuntu16:~/imx6ull/kernel$ make del_audio_imx_v7_defconfig
#
# configuration written to .config
#
alientek@ubuntu16:~/imx6ull/kernel$ make zImage -j 16
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: 'include/generated/mach-types.h' is up to date.
  CHK     include/generated/bounds.h
  CHK     include/generated/asm-offsets.h
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  CC      arch/arm/mach-imx/tzic.o
  LD      sound/core/snd.o
  CC      arch/arm/kernel/irq.o
  LD      sound/core/built-in.o
  GZIP    kernel/config_data.gz
  CHK     kernel/config_data.h
  UPD     kernel/config_data.h
  CC      kernel/configs.o
  CC      sound/soc/soc-devres.o
  LD      sound/soc/codecs/built-in.o
  LD      sound/soc/fsl/built-in.o
  LD      kernel/built-in.o
  LD      arch/arm/mach-imx/built-in.o
  LD      sound/soc/snd-soc-core.o
  LD      arch/arm/kernel/built-in.o
  LD      sound/soc/built-in.o
  LD      sound/built-in.o
  LINK    vmlinux
  LD      vmlinux.o
  MODPOST vmlinux.o
  GEN     .version
  CHK     include/generated/compile.h
  UPD     include/generated/compile.h
  CC      init/version.o
  LD      init/built-in.o
  KSYM    .tmp_kallsyms1.o
  KSYM    .tmp_kallsyms2.o
  LD      vmlinux
  SORTEX  vmlinux
  SYSMAP  System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  LZO     arch/arm/boot/compressed/piggy.lzo
  AS      arch/arm/boot/compressed/piggy.lzo.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
alientek@ubuntu16:~/imx6ull/kernel$ make imx6ull-14x14-emmc-4.3-480x272-c.dtb
  DTC     arch/arm/boot/dts/imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

6.3 IO 测试

6.3.1 设备树添加 GPIO 信息

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件进行修改。

在设备树的 pinctrl_hog_1 节点下添加对应的 GPIO 配置。

```
MX6UL_PAD_JTAG_TMS__GPIO1_IO11      0x80000000
MX6UL_PAD_JTAG_TDO__GPIO1_IO12      0x80000000
MX6UL_PAD_JTAG_TDI__GPIO1_IO13      0x80000000
MX6UL_PAD_JTAG_TCK__GPIO1_IO14      0x80000000
```

```
    MX6UL_PAD_JTAG_TRST_B__GPIO1_IO15        0x80000000
```

```
443  &iomuxc {
444      pinctrl-names = "default";
445      pinctrl-0 = <&pinctrl_hog_1>;
446      imx6ul-evk {
447          pinctrl_hog_1: hoggrp-1 {
448              fsl,pins = <
449                  MX6UL_PAD_UART1_RTS_B__GPIO1_IO19   0x17059 /* SD1 CD */
450                  MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT    0x17059 /* SD1 VSELECT */
451                  MX6UL_PAD_GPIO1_IO00__ANATOP_OTG1_ID    0x13058 /* USB_OTG1_ID */
452                  MX6UL_PAD_JTAG_TMS__GPIO1_IO11      0x80000000
453                  MX6UL_PAD_JTAG_TDO__GPIO1_IO12      0x80000000
454                  MX6UL_PAD_JTAG_TDI__GPIO1_IO13      0x80000000
455                  MX6UL_PAD_JTAG_TCK__GPIO1_IO14      0x80000000
456                  MX6UL_PAD_JTAG_TRST_B__GPIO1_IO15       0x80000000
457              >;
458          };
```

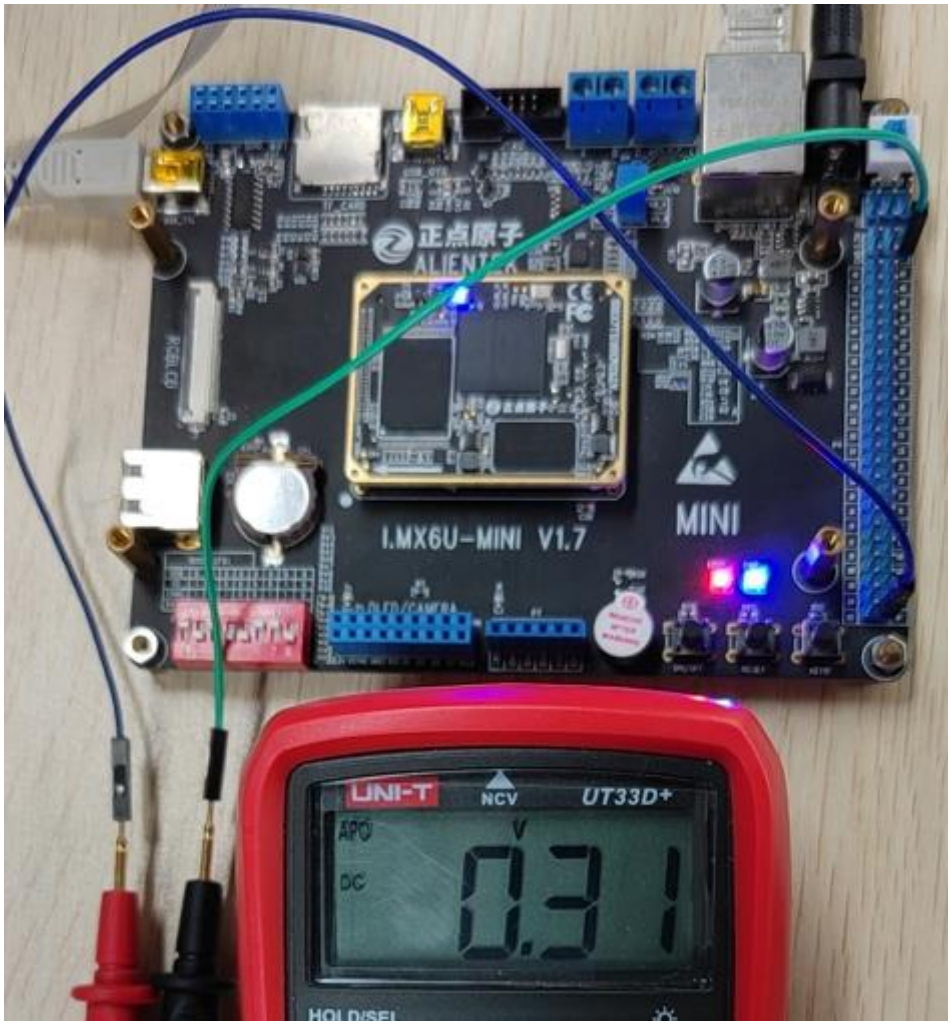
保存修改好的设备树文件，重新编译设备树。

```
    source   /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
    make del_audio_imx_v7_defconfig
    make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

将编译生成的 zImage 和设备树文件替换到开发板上启动。

6.3.2 测试

以 SNVS_TAMPER4 管脚为例。启动 MINI 开发板，将万用表调到电压档位，黑表笔接地，红表笔接 SNVS_TAMPER4 接口。如下图所示：



进入开发板系统终端，进行 GPIO 操作。

```
cd /sys/class/gpio/
echo 132 > export
echo low > gpio132/direction
echo high > gpio132/direction
```

测试 SNVS_TAMPER4 管脚可以拉高/拉低输出。

依次测试其他 IO，可以直接测试的 IO 有 6 个。

MINI 原理图管脚名	GPIO	GPIO 编号	高/低电平	备注
SNVS_TAMPER4	GPIO5_IO04	132	3.3V/0V	MINI 板 PIN1
TAG_TDO	GPIO1_IO12	12	3.3V/0V	MINI 板 PIN27
JTAG_TDI	GPIO1_IO13	13	3.3V/0V	MINI 板 PIN26
JTAG_TCK	GPIO1_IO14	14	3.3V/0V	MINI 板 PIN28
JTAG_nTRST	GPIO1_IO15	15	3.3V/0V	MINI 板 PIN30
JTAG_TMS	GPIO1_IO11	11	3.3V/0V	MINI 板 PIN29

6.4 uboot 修改 JTAG 相关管脚配置

目的：把 JTAG 相关管脚配置为 GPIO 复用，控制电平。

涉及管脚：

MINI 原理图管脚名	GPIO	GPIO 编号
TAG_TDO	GPIO1_IO12	12
JTAG_TDI	GPIO1_IO13	13
JTAG_TCK	GPIO1_IO14	14
JTAG_nTRST	GPIO1_IO15	15
JTAG_TMS	GPIO1_IO11	11
JTAG_MOD	GPIO1_IO10	10

6.4.1 修改 mx6ullevk.c 文件

打开出厂 uboot 源码的 board/freescale/mx6ullevk/mx6ullevk.c 文件，添加 jtag_pads 配置信息，可以参考源码中的 leds_pads 格式。

```
static iomux_v3_cfg_t const jtag_pads[] = {
MX6_PAD_JTAG_MOD__GPIO1_IO10 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_MOD */
MX6_PAD_JTAG_TMS__GPIO1_IO11 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TMS */
MX6_PAD_JTAG_TDO__GPIO1_IO12 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TDO */
MX6_PAD_JTAG_TDI__GPIO1_IO13 | MUX_PAD_CTRL(NO_PAD_CTRL), /* TAG_TDI */
MX6_PAD_JTAG_TCK__GPIO1_IO14 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TCK */
MX6_PAD_JTAG_TRST_B__GPIO1_IO15 | MUX_PAD_CTRL(NO_PAD_CTRL),
                                                    /* JTAG_TRST_B */
};
```

```
1089  static iomux_v3_cfg_t const leds_pads[] = {
1090      CONFIG_LED1_IOMUXC | MUX_PAD_CTRL(NO_PAD_CTRL),
1091      CONFIG_LED2_IOMUXC | MUX_PAD_CTRL(NO_PAD_CTRL),
1092  };
1093
1094  static iomux_v3_cfg_t const jtag_pads[] = {
1095      MX6_PAD_JTAG_MOD__GPIO1_IO10 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_MOD */
1096      MX6_PAD_JTAG_TMS__GPIO1_IO11 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TMS */
1097      MX6_PAD_JTAG_TDO__GPIO1_IO12 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TDO */
1098      MX6_PAD_JTAG_TDI__GPIO1_IO13 | MUX_PAD_CTRL(NO_PAD_CTRL), /* TAG_TDI */
1099      MX6_PAD_JTAG_TCK__GPIO1_IO14 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TCK */
1100      MX6_PAD_JTAG_TRST_B__GPIO1_IO15 | MUX_PAD_CTRL(NO_PAD_CTRL), /* JTAG_TRST_B */
1101  };
```

MX6_PAD_JTAG_MOD__GPIO1_IO10 等管脚配置信息可以在 uboot 源码的 arch/arm/include/asm/arch-mx6/mx6ul_pins.h 文件中获取。

在 board_init 函数中添加管脚默认初始化电平，如下所示：

```
    imx_iomux_v3_setup_multiple_pads(jtag_pads, ARRAY_SIZE(jtag_pads));
    gpio_direction_output(IMX_GPIO_NR(1,10),0);
    gpio_direction_output(IMX_GPIO_NR(1,11),0);
    gpio_direction_output(IMX_GPIO_NR(1,12),0);
    gpio_direction_output(IMX_GPIO_NR(1,13),0);
    gpio_direction_output(IMX_GPIO_NR(1,14),0);
    gpio_direction_output(IMX_GPIO_NR(1,15),0);
```

```
int board_init(void)
{
    /* Address of boot parameters */
    gd->bd->bi_boot_params = PHYS_SDRAM + 0x100;

    imx_iomux_v3_setup_multiple_pads(leds_pads, ARRAY_SIZE(leds_pads));

    imx_iomux_v3_setup_multiple_pads(iox_pads, ARRAY_SIZE(iox_pads));

    imx_iomux_v3_setup_multiple_pads(jtag_pads, ARRAY_SIZE(jtag_pads));
    gpio_direction_output(IMX_GPIO_NR(1,10),0);
    gpio_direction_output(IMX_GPIO_NR(1,11),0);
    gpio_direction_output(IMX_GPIO_NR(1,12),0);
    gpio_direction_output(IMX_GPIO_NR(1,13),0);
    gpio_direction_output(IMX_GPIO_NR(1,14),0);
    gpio_direction_output(IMX_GPIO_NR(1,15),0);

    iox74lv_init();
```

imx_iomux_v3_setup_multiple_pads 参考源码自带的 imx_iomux_v3_setup_multiple_pads 格式，将刚刚写好的 jtag_pad 配置添加进来。

gpio_direction_output 函数由源码中的 arch/blackfin/cpu/gpio.c 文件提供，原型为：

```
int gpio_direction_output(unsigned gpio, int value)
```

IMX_GPIO_NR 宏由 arch/arm/include/asm/imx-common/gpio.h 文件提供，原型为：

```
#define IMX_GPIO_NR(port, index)        ((((port)-1)*32)+((index)&31))
```

综上可得，gpio_direction_output(IMX_GPIO_NR(1,10),0)的作用是是将 GPIO1_IO10 设置为低电平输出，这样编译出来的 uboot 就是默认 GPIO1_IO10 为低电平。

6.4.2 命令测试 GPIO 输出

执行 build.sh 脚本编译出厂 uboot 源码，生成 uboot 的 bin 文件和 imx 文件，如下图。

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls
u-boot-imx6ull-14x14-ddr256-emmc.bin  u-boot-imx6ull-14x14-ddr256-nand-sd.bin  u-boot-imx6ull-14x14-ddr512-nand.bin
u-boot-imx6ull-14x14-ddr256-emmc.imx  u-boot-imx6ull-14x14-ddr256-nand-sd.imx  u-boot-imx6ull-14x14-ddr512-nand.imx
u-boot-imx6ull-14x14-ddr256-nand.bin  u-boot-imx6ull-14x14-ddr512-emmc.bin     u-boot-imx6ull-14x14-ddr512-nand-sd.bin
u-boot-imx6ull-14x14-ddr256-nand.imx  u-boot-imx6ull-14x14-ddr512-emmc.imx     u-boot-imx6ull-14x14-ddr512-nand-sd.imx
```

根据核心板版本，选择对应的 u-boot-xxxxx.bin 文件，使用 imxdownload 软件烧写到 SD 卡。

eMMC 版本核心板选择 u-boot-imx6ull-14x14-ddr512-emmc.bin。

NAND Flash 版本核心板选择 u-boot-imx6ull-14x14-ddr256-nand-sd.bin。

烧写示范：（NAND 烧写时，烧写指令最后要加 -256m）

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sdb  /dev/sdb1
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ./imxdownload u-boot-imx6ull-14x14-ddr512-emmc.bin /dev/sdb
I.MX6ULL bin download software
Edit by:zuozhongkai
Date:2019/6/10
Version:V1.1
log:V1.0 initial version,just support 512MB DDR3
    V1.1 and support 256MB DDR3
file u-boot-imx6ull-14x14-ddr512-emmc.bin size = 355124Bytes
Board DDR SIZE: 512MB
Delete Old load.imx
Create New load.imx
Download load.imx to /dev/sdb  ......
[sudo] alientek 的密码:
记录了699+1 的读入
记录了699+1 的写出
358196 bytes (358 kB, 350 KiB) copied, 2.25007 s, 159 kB/s
```

将烧写好 uboot 的 SD 卡接到 MINI 开发板上，拨码开始 SD 模式启动，在 uboot 命令行终端进行测试。

进入 uboot 命令行，用 gpio 相关指令可以进行 IO 的上下拉测试。gpio 指令说明：

```
=> gpio
gpio - query and control gpio pins

Usage:
gpio <input|set|clear|toggle> <pin>
    - input/set/clear/toggle the specified pin
gpio status [-a] [<bank> | <pin>]  - show [all/claimed] GPIOs
```

示例：将 GPIO1_IO10 拉低。

```
gpio clear 10
```

将将 GPIO1_IO10 拉高。

```
gpio set 10
```

硬件连接：请根据自制的底板硬件或 Mini 板，用万用表对引出的 JTAG 相关 IO 进行测试。

使用 gpio 指令依次对 JTAG 相关 IO 测试。结果如下：

MINI 原理图管脚名	GPIO	GPIO 编号	高/低电平
TAG_TDO	GPIO1_IO12	12	3.3V/0V
JTAG_TDI	GPIO1_IO13	13	3.3V/0V
JTAG_TCK	GPIO1_IO14	14	3.3V/0V
JTAG_nTRST	GPIO1_IO15	15	3.3V/0V
JTAG_TMS	GPIO1_IO11	11	3.3V/0V
JTAG_MOD	GPIO1_IO10	10	3.3V/0V

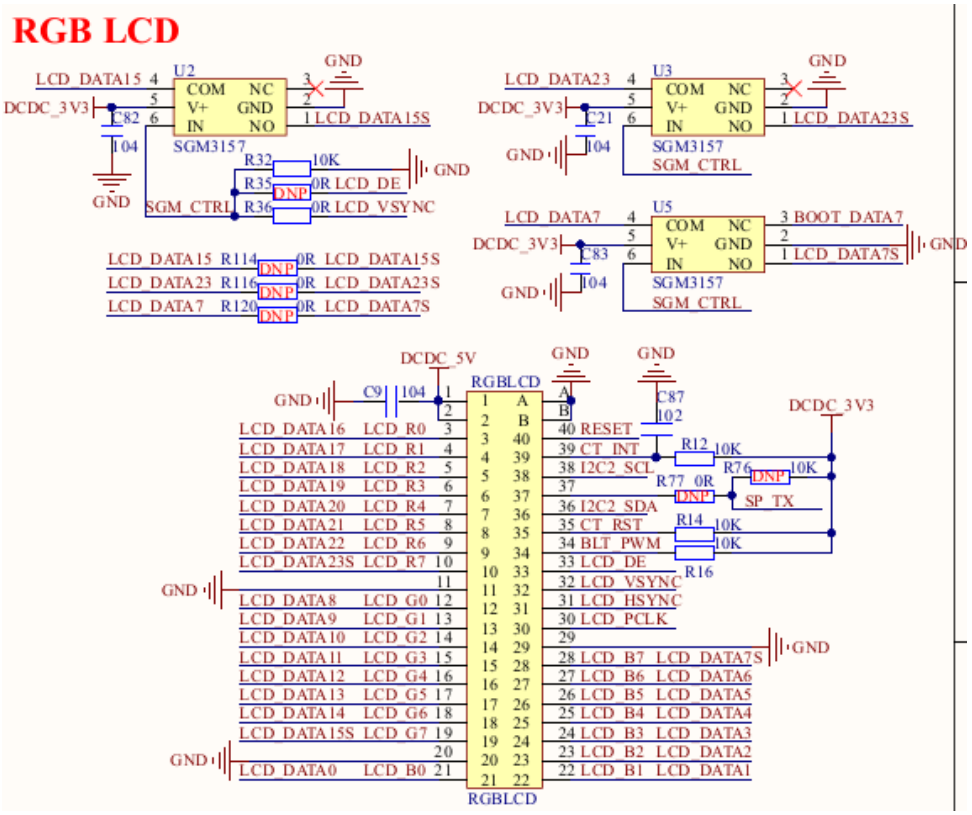
6.4.3 程序测试 GPIO 输出\输入\中断

参考本文 3.4 小节。

第七章 裁剪 LCD

7.1 驱动及管脚定义

以阿尔法开发板为例，LCD 接口相关原理图如下：



可以看到 LCD 涉及到的 IO 种类特别多，有 LCD 数据传输的 IO，也有 I2C、PWM、PCLK、HSYNC 等功能 IO，裁剪起来考虑的因素比较多。

LCD_DATA7\15\23 这几个 IO 可能影响上电设计，不建议修改这几路 IO 做别的功能。

原理图管脚名	GPIO 名	可复用功能
LCD_DATA16	GPIO3_IO21	LCDIF_DATA16、UART7_TX、CSI_DATA01、EIM_DATA08、GPIO3_IO21、SRC_BT_CFG24、USDHC2_DATA6、EPDC_GDCLK
LCD_DATA17	GPIO3_IO22	LCDIF_DATA17、UART7_RX、CSI_DATA00、EIM_DATA09、GPIO3_IO22、SRC_BT_CFG25、USDHC2_DATA7、EPDC_GDSP
LCD_DATA18	GPIO3_IO23	LCDIF_DATA18、PWM5_OUT、CA7_MX6ULL_EVENTO、CSI_DATA10、EIM_DATA10、GPIO3_IO23、SRC_BT_CFG26、USDHC2_CMD、EPDC_BDR01
LCD_DATA19	GPIO3_IO24	EIM_DATA11、GPIO3_IO24、

		SRC_BT_CFG27、USDHC2_CLK、 EPDC_VCOM00、LCDIF_DATA19、 PWM6_OUT、 WDOG1_WDOG_ANY、 CSI_DATA11
LCD_DATA20	GPIO3_IO25	EIM_DATA12、GPIO3_IO25、 SRC_BT_CFG28、USDHC2_DATA0、 EPDC_VCOM01、LCDIF_DATA20、 UART8_TX、ECSPI1_SCLK、 CSI_DATA12
LCD_DATA21	GPIO3_IO26	LCDIF_DATA21、UART8_RX、 ECSPI1_SS0、CSI_DATA13、 EIM_DATA13、GPIO3_IO26、 SRC_BT_CFG29、USDHC2_DATA1、 EPDC_SDCE01
LCD_DATA22	GPIO3_IO27	LCDIF_DATA22、MQS_RIGHT、 ECSPI1_MOSI、CSI_DATA14、 EIM_DATA14、GPIO3_IO27、 SRC_BT_CFG30、USDHC2_DATA2、 EPDC_SDCE02
LCD_DATA23	GPIO3_IO28	EPDC_SDCE03、LCDIF_DATA23 MQS_LEFT、MQS_LEFT、 CSI_DATA15、EIM_DATA15、 GPIO3_IO28、SRC_BT_CFG31、 USDHC2_DATA3
LCD_DATA8	GPIO3_IO13	LCDIF_DATA08、SPDIF_IN、 CSI_DATA16、EIM_DATA00、 GPIO3_IO13、SRC_BT_CFG08、 FLEXCAN1_TX、EPDC_PWRIRQ
LCD_DATA9	GPIO3_IO14	LCDIF_DATA09、SAI3_MCLK、 CSI_DATA17、EIM_DATA01、 GPIO3_IO14、SRC_BT_CFG09、 FLEXCAN1_RX、EPDC_PWRWAKE
LCD_DATA10	GPIO3_IO15	LCDIF_DATA10、SAI3_RX_SYNC、 CSI_DATA18、EIM_DATA02、 GPIO3_IO15、SRC_BT_CFG10、 FLEXCAN2_TX、EPDC_PWRCOM
LCD_DATA11	GPIO3_IO16	LCDIF_DATA11、SAI3_RX_BCLK、 CSI_DATA19、EIM_DATA03、 GPIO3_IO16、SRC_BT_CFG11、 FLEXCAN2_RX、EPDC_PWRSTAT
LCD_DATA12	GPIO3_IO17	LCDIF_DATA12、SAI3_TX_SYNC、 CSI_DATA20、EIM_DATA04、 GPIO3_IO17、SRC_BT_CFG12、 ECSPI1_RDY、EPDC_PWRCTRL00
LCD_DATA13	GPIO3_IO18	LCDIF_DATA13、SAI3_TX_BCLK、 CSI_DATA21、EIM_DATA05、 GPIO3_IO18、SRC_BT_CFG13、

		USDHC2_RESET_B、EPDC_BDR00
LCD_DATA14	GPIO3_IO19	LCDIF_DATA14、SAI3_RX_DATA、 CSI_DATA2、EIM_DATA0、 GPIO3_IO19、SRC_BT_CFG14、 USDHC2_DATA4、EPDC_SDSHR
LCD_DATA15	GPIO3_IO20	LCDIF_DATA15、SAI3_TX_DATA、 CSI_DATA23、EIM_DATA07、 GPIO3_IO20、SRC_BT_CFG15、 USDHC2_DATA5、EPDC_GDRL
LCD_DATA0	GPIO3_IO05	LCDIF_DATA00、PWM1_OUT、 ENET1_1588_EVENT2_IN、 I2C3_SDA、GPIO3_IO05、 SRC_BT_CFG00、SAI1_MCLK、 EPDC_SDDO00
LCD_DATA1	GPIO3_IO06	LCDIF_DATA01、PWM2_OUT、 ENET1_1588_EVENT2_OUT、 I2C3_SCL、GPIO3_IO06、 SRC_BT_CFG01、SAI1_TX_SYNC 、EPDC_SDDO01
LCD_DATA2	GPIO3_IO07	LCDIF_DATA02、PWM3_OUT、 ENET1_1588_EVENT3_IN、 I2C4_SDA、GPIO3_IO07、 SRC_BT_CFG02、 SAI1_TX_BCLK EPDC_SDDO02
LCD_DATA3	GPIO3_IO08	LCDIF_DATA03、PWM4_OUT、 ENET1_1588_EVENT3_OUT、 I2C4_SCL、GPIO3_IO08、 SRC_BT_CFG03、 SAI1_RX_DATA、 EPDC_SDDO03
LCD_DATA4	GPIO3_IO09	LCDIF_DATA04、UART8_CTS_B、 ENET2_1588_EVENT2_IN、 SPDIF_SR_CLK、GPIO3_IO09、 SRC_BT_CFG04、SAI1_TX_DATA、 EPDC_SDDO04
LCD_DATA5	GPIO3_IO10	LCDIF_DATA05、UART8_RTS_B、 ENET2_1588_EVENT2_OUT、 SPDIF_OUT、GPIO3_IO10、 SRC_BT_CFG05、ECSPI1_SS1、 EPDC_SDDO05
LCD_DATA6	GPIO3_IO11	LCDIF_DATA06、UART7_CTS_B、 ENET2_1588_EVENT3_IN、 SPDIF_LOCK、GPIO3_IO11、 SRC_BT_CFG06、ECSPI1_SS2、 EPDC_SDDO06
LCD_DATA7	GPIO3_IO12	LCDIF_DATA07、UART7_RTS_B、 ENET2_1588_EVENT3_OUT、 SPDIF_EXT_CLK、GPIO3_IO12、

		SRC_BT_CFG07、ECSPI1_SS3、 EPDC_SDDO07
LCD_PCLK	GPIO3_IO00	LCDIF_CLK、LCDIF_WR_RWN、 UART4_TX、SAI3_MCLK、 EIM_CS2_B、GPIO3_IO00、 WDOG1_WDOG_RST_B_DEB、 EPDC_SDCLK
LCD_HSYNC	GPIO3_IO02	LCDIF_HSYNC、LCDIF_RS、 UART4_CTS_B、SAI3_TX_BCLK、 WDOG3_WDOG_RST_B_DEB、 GPIO3_IO02、ECSPI2_SS1、 EPDC_SDOE
LCD_VSYNC	GPIO3_IO03	LCDIF_VSYNC、LCDIF_BUSY、 UART4_RTS_B、SAI3_RX_DATA、 WDOG2_WDOG_B、GPIO3_IO03、 ECSPI2_SS2、EPDC_SDCE00
LCD_DE	GPIO3_IO01	LCDIF_ENABLE、LCDIF_RD_E、 UART4_RX、SAI3_TX_SYNC、 EIM_CS3_B、GPIO3_IO01、 ECSPI2_RDY、EPDC_SDLE
BLT_PWM	GPIO1_IO08	PWM1_OUT、 WDOG1_WDOG_B、SPDIF_OUT、 CSI_VSYNC、USDHC2_VSELECT、 GPIO1_IO08、CCM_PMIC_RDY、 UART5_RTS_B
CT_RST	GPIO5_IO09	GPIO5_IO09
I2C2_SDA	GPIO1_IO31	UART5_RX、ENET2_COL、 I2C2_SDA、CSI_DATA15、 CSU_CSU_INT_DEB、GPIO1_IO31、 ECSPI2_MISO、EPDC_PWRCTRL03
SP_TX	GPIO1_IO10	SJC_MOD、GPT2_CLK、 SPDIF_OUT、 ENET1_REF_CLK_25M、 CCM_PMIC_RDY、 GPIO1_IO10、 SDMA_EXT_EVENT00
I2C2_SCL	GPIO1_IO30	GPIO1_IO30、ECSPI2_MOSI、 EPDC_PWRCTRL02、UART5_TX ENET2_CRS、I2C2_SCL、 CSI_DATA14、CSU_CSU_ALARM_ AUT00
CT_INT	GPIO1_IO09	PWM2_OUT、 WDOG1_WDOG_ANY、SPDIF_IN、 CSI_HSYNC、 USDHC2_RESET_B、 GPIO1_IO09、USDHC1_RESET_B、 UART5_CTS_B

7.2 U-boot 裁剪 LCD

uboot 里有 LCD 相关配置，因此要裁剪掉 uboot 中的 LCD 相关配置。

7.2.1 源码编译

解压出厂 uboot 源码并编译一遍。

7.2.2 修改 mx6ullevk.c

打开 uboot 源码，修改 board/freescale/mx6ullevk/mx6ullevk.c 文件。
搜索 LCD，注释掉 uboot 中 LCD 相关管脚功能配置。

```
/*
#ifdef CONFIG_VIDEO_MXS
static iomux_v3_cfg_t const lcd_pads[] = {
    MX6_PAD_LCD_CLK__LCDIF_CLK | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_ENABLE__LCDIF_ENABLE | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_HSYNC__LCDIF_HSYNC | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_VSYNC__LCDIF_VSYNC | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA00__LCDIF_DATA00 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA01__LCDIF_DATA01 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA02__LCDIF_DATA02 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA03__LCDIF_DATA03 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA04__LCDIF_DATA04 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA05__LCDIF_DATA05 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA06__LCDIF_DATA06 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA07__LCDIF_DATA07 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA08__LCDIF_DATA08 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA09__LCDIF_DATA09 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA10__LCDIF_DATA10 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA11__LCDIF_DATA11 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA12__LCDIF_DATA12 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA13__LCDIF_DATA13 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA14__LCDIF_DATA14 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA15__LCDIF_DATA15 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA16__LCDIF_DATA16 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA17__LCDIF_DATA17 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA18__LCDIF_DATA18 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA19__LCDIF_DATA19 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA20__LCDIF_DATA20 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA21__LCDIF_DATA21 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA22__LCDIF_DATA22 | MUX_PAD_CTRL(LCD_PAD_CTRL),
    MX6_PAD_LCD_DATA23__LCDIF_DATA23 | MUX_PAD_CTRL(LCD_PAD_CTRL),

    //LCD_RST
    MX6_PAD_SNVS_TAMPER9__GPIO5_IO09 | MUX_PAD_CTRL(NO_PAD_CTRL),

    // Use GPIO for Brightness adjustment, duty cycle = period.
    MX6_PAD_GPIO1_IO08__GPIO1_IO08 | MUX_PAD_CTRL(NO_PAD_CTRL),
};
*/
```

```
/* static iomux_v3_cfg_t const lcd_id_pads[] = {
    MX6_PAD_LCD_DATA23__GPIO3_IO28 | MUX_PAD_CTRL(NO_PAD_CTRL),  // lcd_r7
    MX6_PAD_LCD_DATA15__GPIO3_IO20 | MUX_PAD_CTRL(NO_PAD_CTRL),  // lcd_g7
    MX6_PAD_LCD_DATA07__GPIO3_IO12 | MUX_PAD_CTRL(NO_PAD_CTRL),  // lcd_b7
    MX6_PAD_LCD_VSYNC__GPIO3_IO03 | MUX_PAD_CTRL(NO_PAD_CTRL),  // bootcfg signal isolation
}; */
```

```
/* void do_enable_parallel_lcd(struct display_info_t const *dev)
{
    enable_lcdif_clock(dev->bus);

    imx_iomux_v3_setup_multiple_pads(lcd_pads, ARRAY_SIZE(lcd_pads));

    gpio_direction_output(IMX_GPIO_NR(1, 8) , 0);
} */
```

注册掉 void select_display_dev (void)函数，这里图片放不下。
保存修改好的文件。

7.2.3 修改 mx6ullevk.h

对 include/configs/mx6ullevk.h 文件进行修改。
屏蔽掉#define CONFIG_VIDEO

```
#ifndef CONFIG_SPL_BUILD
/*#define CONFIG_VIDEO*/
#ifdef CONFIG_VIDEO
#define CONFIG_CFB_CONSOLE
```

保存修改好的文件。

7.2.4 修改 video.c

对 arch/arm/imx-common/video.c 文件进行修改。
注释掉 int board_video_skip(void)函数相关内容。

```
{
    int i;
    int ret;
    char const *panel; */

    /* Select LCD configuration based on current hardware ID */
/*  select_display_dev(); */

/*  panel = getenv("panel");
    if (!panel) {
        for (i = 0; i < display_count; i++) {
            struct display_info_t const *dev = displays+i;
            if (dev->detect && dev->detect(dev)) {
                panel = dev->mode.name;
                printf("auto-detected panel %s\n", panel);
                break;
            }
        }
        if (!panel) {
            panel = displays[0].mode.name;
            printf("No panel detected: default to %s\n", panel);
            i = 0;
        }
    } else {
        for (i = 0; i < display_count; i++) {
            if (!strcmp(panel, displays[i].mode.name))
                break;
        }
    }

    if (i < display_count) {
#if defined(CONFIG_VIDEO_IPUV3)
        ret = ipuv3_fb_init(&displays[i].mode, 0,
                    displays[i].pixfmt);
#elif defined(CONFIG_VIDEO_MXS)
        ret = mxs_lcd_panel_setup(displays[i].mode,
                    displays[i].pixfmt,
                    displays[i].bus);
#endif
        if (!ret) {
            if (displays[i].enable)
                displays[i].enable(displays + i);

            printf("Display: %s (%ux%u)\n",
                    displays[i].mode.name,
                    displays[i].mode.xres,
                    displays[i].mode.yres);
        } else
            printf("LCD %s cannot be configured: %d\n",
                    displays[i].mode.name, ret);
    } else {
        printf("unsupported panel %s\n", panel);
        return -EINVAL;
    }

    return 0;
} */
```

保存修改好的文件。

7.2.5 uboot 指令 gpio 测试

执行 build.sh 脚本编译出厂 uboot 源码，生成 uboot 的 bin 文件和 imx 文件，如下图。

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls
u-boot-imx6ull-14x14-ddr256-emmc.bin  u-boot-imx6ull-14x14-ddr256-nand-sd.bin  u-boot-imx6ull-14x14-ddr512-nand.bin
u-boot-imx6ull-14x14-ddr256-emmc.imx  u-boot-imx6ull-14x14-ddr256-nand-sd.imx  u-boot-imx6ull-14x14-ddr512-nand.imx
u-boot-imx6ull-14x14-ddr256-nand.bin  u-boot-imx6ull-14x14-ddr512-emmc.bin     u-boot-imx6ull-14x14-ddr512-nand-sd.bin
u-boot-imx6ull-14x14-ddr256-nand.imx  u-boot-imx6ull-14x14-ddr512-emmc.imx     u-boot-imx6ull-14x14-ddr512-nand-sd.imx
```

根据核心板版本，选择对应的 bin 文件，使用 imxdownload 软件烧写到 SD 卡。

eMMC 版本核心板选择 u-boot-imx6ull-14x14-ddr512-emmc.bin。

NAND Flash 版本核心板选择 u-boot-imx6ull-14x14-ddr256-nand-sd.bin。

烧写示范：（NAND 烧写时，烧写指令最后要加 -256m）

```
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sdb  /dev/sdb1
alientek@ubuntu16:~/imx6ull/uboot/tmp$ ./imxdownload u-boot-imx6ull-14x14-ddr512-emmc.bin /dev/sdb
I.MX6ULL bin download software
Edit by:zuozhongkai
Date:2019/6/10
Version:V1.1
log:V1.0 initial version,just support 512MB DDR3
    V1.1 and support 256MB DDR3
file u-boot-imx6ull-14x14-ddr512-emmc.bin size = 355124Bytes
Board DDR SIZE: 512MB
Delete Old load.imx
Create New load.imx
Download load.imx to /dev/sdb  ......
[sudo] alientek 的密码:
记录了699+1 的读入
记录了699+1 的写出
358196 bytes (358 kB, 350 KiB) copied, 2.25007 s, 159 kB/s
```

将烧写好 uboot 的 SD 卡接到 MINI 开发板上，拨码开始 SD 模式启动，在 uboot 命令行终端进行测试。

进入 uboot 命令行，用 gpio 相关指令可以进行 IO 的上下拉测试。gpio 指令说明：

```
=> gpio
gpio - query and control gpio pins

Usage:
gpio <input|set|clear|toggle> <pin>
    - input/set/clear/toggle the specified pin
gpio status [-a] [<bank> | <pin>]  - show [all/claimed] GPIOs
```

示例：将 GPIO85 拉低。

```
gpio clear 85
```

将 GPIO85 拉高。

```
gpio set 85
```

硬件连接：请根据自制的底板硬件，用万用表对引出的 LCD 相关 IO 进行测试。

使用 gpio 指令依次对 LCD 相关 IO 测试。结果如下：

原理图管脚	GPIO	GPIO 编号	高/低电平
LCD_DATA16	GPIO3_IO21	85	3.3V/0V
LCD_DATA17	GPIO3_IO22	86	3.3V/0V
LCD_DATA18	GPIO3_IO23	87	3.3V/0V
LCD_DATA19	GPIO3_IO24	88	3.3V/0V
LCD_DATA20	GPIO3_IO25	89	3.3V/0V
LCD_DATA21	GPIO3_IO26	90	3.3V/0V
LCD_DATA22	GPIO3_IO27	91	3.3V/0V
LCD_DATA23	GPIO3_IO28	92	3.3V/0V

LCD_DATA8	GPIO3_IO13	77	3.3V/0V
LCD_DATA9	GPIO3_IO14	78	3.3V/0V
LCD_DATA10	GPIO3_IO15	79	3.3V/0V
LCD_DATA11	GPIO3_IO16	80	3.3V/0V
LCD_DATA12	GPIO3_IO17	81	3.3V/0V
LCD_DATA13	GPIO3_IO18	82	3.3V/0V
LCD_DATA14	GPIO3_IO19	83	3.3V/0V
LCD_DATA15	GPIO3_IO20	84	3.3V/0V
LCD_DATA0	GPIO3_IO05	69	3.3V/0V
LCD_DATA1	GPIO3_IO06	70	3.3V/0V
LCD_DATA2	GPIO3_IO07	71	3.3V/0V
LCD_DATA3	GPIO3_IO08	72	3.3V/0V
LCD_DATA4	GPIO3_IO09	73	3.3V/0V
LCD_DATA5	GPIO3_IO10	74	3.3V/0V
LCD_DATA6	GPIO3_IO11	75	3.3V/0V
LCD_DATA7	GPIO3_IO12	76	3.3V/0V
LCD_PCLK	GPIO3_IO00	64	3.3V/0V
LCD_HSYNC	GPIO3_IO02	66	3.3V/0V
LCD_VSYNC	GPIO3_IO03	67	3.3V/0V
LCD_DE	GPIO3_IO01	65	3.3V/0V
BLT_PWM	GPIO1_IO08	8	3.3V/0V
CT_RST	GPIO5_IO009	137	3.3V/0V
I2C2_SDA	GPIO1_IO31	31	3.3V/0V
SP_TX	GPIO1_IO10	10	3.3V/0V
I2C2_SCL	GPIO1_IO30	30	3.3V/0V
CT_INT	GPIO1_IO09	9	3.3V/0V

7.3 内核裁剪 LCD 驱动

打开内核源码 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件进行修改。

依次注释掉&iomuxc 节点下的 pinctrl_lcdif_dat、pinctrl_lcdif_ctrl、pinctrl_pwm1、pinctrl_sii902x、pinctrl_wdog、pinctrl_lcdif_reset、&lcdif 节点配置信息。屏蔽掉&pwm1、&wdog1、sii902x: sii902x@39 相关配置。

pinctrl_lcdif_dat

```
/*      pinctrl_lcdif_dat: lcdifdatgrp {
            fsl,pins = <
                MX6UL_PAD_LCD_DATA00__LCDIF_DATA00  0x49
                MX6UL_PAD_LCD_DATA01__LCDIF_DATA01  0x49
                MX6UL_PAD_LCD_DATA02__LCDIF_DATA02  0x49
                MX6UL_PAD_LCD_DATA03__LCDIF_DATA03  0x49
                MX6UL_PAD_LCD_DATA04__LCDIF_DATA04  0x49
                MX6UL_PAD_LCD_DATA05__LCDIF_DATA05  0x49
                MX6UL_PAD_LCD_DATA06__LCDIF_DATA06  0x49
                MX6UL_PAD_LCD_DATA07__LCDIF_DATA07  0x51
                MX6UL_PAD_LCD_DATA08__LCDIF_DATA08  0x49
                MX6UL_PAD_LCD_DATA09__LCDIF_DATA09  0x49
                MX6UL_PAD_LCD_DATA10__LCDIF_DATA10  0x49
                MX6UL_PAD_LCD_DATA11__LCDIF_DATA11  0x49
                MX6UL_PAD_LCD_DATA12__LCDIF_DATA12  0x49
                MX6UL_PAD_LCD_DATA13__LCDIF_DATA13  0x49
                MX6UL_PAD_LCD_DATA14__LCDIF_DATA14  0x49
                MX6UL_PAD_LCD_DATA15__LCDIF_DATA15  0x51
                MX6UL_PAD_LCD_DATA16__LCDIF_DATA16  0x49
                MX6UL_PAD_LCD_DATA17__LCDIF_DATA17  0x49
                MX6UL_PAD_LCD_DATA18__LCDIF_DATA18  0x49
                MX6UL_PAD_LCD_DATA19__LCDIF_DATA19  0x49
                MX6UL_PAD_LCD_DATA20__LCDIF_DATA20  0x49
                MX6UL_PAD_LCD_DATA21__LCDIF_DATA21  0x49
                MX6UL_PAD_LCD_DATA22__LCDIF_DATA22  0x49
                MX6UL_PAD_LCD_DATA23__LCDIF_DATA23  0x51
            >;
        }; */
```

pinctrl_lcdif_ctrl

```
/*      pinctrl_lcdif_ctrl: lcdifctrlgrp {
            fsl,pins = <
                MX6UL_PAD_LCD_CLK__LCDIF_CLK        0x49
                MX6UL_PAD_LCD_ENABLE__LCDIF_ENABLE  0x49
                MX6UL_PAD_LCD_HSYNC__LCDIF_HSYNC    0x49
                MX6UL_PAD_LCD_VSYNC__LCDIF_VSYNC    0x49
            >;
        }; */
```

pinctrl_pwm1

```
/*      pinctrl_pwm1: pwm1grp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO08__PWM1_OUT   0x110b0
            >;
        }; */

```

pinctrl_sii902x

```
/*              pinctrl_sii902x: hdmigrp-1 {
                        fsl,pins = <
                                MX6UL_PAD_GPIO1_IO09__GPIO1_IO09        0x11
                        >;
                }; */
```

pinctrl_wdog

```
/*      pinctrl_wdog: wdoggrp {
            fsl,pins = <
                MX6UL_PAD_LCD_RESET__WDOG1_WDOG_ANY    0x30b0
            >;
        }; */
```

pinctrl_lcdif_reset

```
/*      pinctrl_lcdif_reset: lcdifresetgrp {
                        fsl,pins = <
                                // used for lcd reset
                                MX6ULL_PAD_SNVS_TAMPER9__GPIO5_IO09  0x49
                        >;
                }; */
```

&lcdif

```
/* &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 = <35500000>;
                        hactive = <800>;
                        vactive = <480>;
                        hfront-porch = <210>;
                        hback-porch = <46>;
                        hsync-len = <20>;
                        vback-porch = <23>;
                        vfront-porch = <22>;
                        vsync-len = <3>;

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

&pwm1

```
/* &pwm1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_pwm1>;
    status = "okay";
}; */
```

&wdog1

```
1083    /* &wdog1 {
1084        pinctrl-names = "default";
1085        pinctrl-0 = <&pinctrl_wdog>;
1086        fsl,wdog_b;
1087    }; */
1088
```

sii902x: sii902x@39

```
439    /*          sii902x: sii902x@39 {
440                    compatible = "SiI,sii902x";
441                    pinctrl-names = "default";
442                    pinctrl-0 = <&pinctrl_sii902x>;
443                    interrupt-parent = <&gpio1>;
444                    interrupts = <9 IRQ_TYPE_EDGE_FALLING>;
445            irq-gpios = <&gpio1 9 GPIO_ACTIVE_LOW>;
446                    mode_str = "1280x720M@60";
447                    bits-per-pixel = <16>;
448                    resets = <&sii902x_reset>;
449                    reg = <0x39>;
450                    status = "disabled";
451            }; */
452    };
```

保存修改好的设备树文件，编译内核和设备树启动。

7.4 命令测试 GPIO 输出

硬件连接：请根据自制的底板硬件，用万用表对引出的 LCD 相关 IO 进行测试。
进入开发板系统终端，进行 GPIO 操作。示例：

```
cd /sys/class/gpio/
echo 85 > export
echo low > gpio85/direction
echo high > gpio85/direction
```

测试 LCD_DATA16 管脚可以拉高/拉低输出。
其他 IO 测试方法一样，具体测试结果可以参考前面 uboot 裁剪 LCD 测试结果的表格。
注意：SP_TX 管脚（即 GPIO1_IO10）的修改还需要初始化 uboot 电平，可以参考 6.4 小节。

7.5 程序测试 GPIO 输出\输入\中断

参考本文 3.4 小节。

【外设复用篇说明】

在上一篇中讲解了 Linux 开发板上常用的外设裁剪，包括摄像头接口、双网口、音频接口、JTAG 接口的裁剪，通过对这些外设的裁剪，可以释放出很多的管脚。用户可以根据芯片参考手册，将这些管脚复用成自己需要的功能，达到项目需求。

本篇基于 Linux 开发板释放的管脚，进行常见的外设复用配置和实现。本文档仅供参考，具体实现效果还取决于实际硬件是否正常。

第八章 多路 GPIO 复用

8.1 GPIO 复用修改

I.MX6ULL 核心板上绝大多数管脚都可以复用成 GPIO 功能使用，这部分在前面外设裁剪篇有具体的修改说明。GPIO 复用修改，首先要先判断当前管脚在内核、设备树、系统、硬件上是否已经用做其他外设功能，如果有的话需要依次对其进行裁剪释放。完成释放的管脚，一般情况下是可以通过系统的 GPIO 指令进行操作的，可以通过设置高低电平，结合万用表来测试效果。

如果需要对 GPIO 进行更细致的管理，可以将 GPIO 管脚相关配置添加到设备树里面。

如果开发板启动或者复位过程中，产生 GPIO 电平差异，比如设置某 GPIO 为低电平，复位过程中发生此 GPIO 为高电平或者电平上下起伏的现象，就要检查下 uboot、内核中是否还有配置功能影响到此 GPIO、底板硬件电路上是否有干扰到此 GPIO。

8.2 设备树添加 GPIO

这部分可以结合【正点原子】I.MX6U 嵌入式 Linux 驱动开发指南 V1.6.pdf 文档的第四十五章 pinctrl 和 gpio 子系统实验，里面有具体的教程说明。

以释放的 JTAG 管脚为例，在设备树中添加 GPIO 配置信息。

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件进行修改。

在设备树的 pinctrl_hog_1 节点下添加对应的 PIN 配置信息。

```
MX6UL_PAD_JTAG_TMS__GPIO1_IO11      0x80000000
MX6UL_PAD_JTAG_TDO__GPIO1_IO12      0x80000000
MX6UL_PAD_JTAG_TDI__GPIO1_IO13      0x80000000
MX6UL_PAD_JTAG_TCK__GPIO1_IO14      0x80000000
MX6UL_PAD_JTAG_TRST_B__GPIO1_IO15       0x80000000
```

```
443  &iomuxc {
444      pinctrl-names = "default";
445      pinctrl-0 = <&pinctrl_hog_1>;
446      imx6ul-evk {
447          pinctrl_hog_1: hoggrp-1 {
448              fsl,pins = <
449                  MX6UL_PAD_UART1_RTS_B__GPIO1_IO19   0x17059 /* SD1 CD */
450                  MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT    0x17059 /* SD1 VSELECT */
451                  MX6UL_PAD_GPIO1_IO00__ANATOP_OTG1_ID    0x13058 /* USB_OTG1_ID */
452                  MX6UL_PAD_JTAG_TMS__GPIO1_IO11      0x80000000
453                  MX6UL_PAD_JTAG_TDO__GPIO1_IO12      0x80000000
454                  MX6UL_PAD_JTAG_TDI__GPIO1_IO13      0x80000000
455                  MX6UL_PAD_JTAG_TCK__GPIO1_IO14      0x80000000
456                  MX6UL_PAD_JTAG_TRST_B__GPIO1_IO15       0x80000000
457              >;
458          };
```

保存修改好的设备树文件，重新编译设备树。

pinctrl_hog_1 子节点就是和热插拔有关的 PIN 集合，比如 USB OTG 的 ID 引脚。我们可以直接将释放完的管脚 GPIO 配置信息添加到这个节点下直接使用。对于一个 PIN 的配置主要包括两方面，一个是设置这个 PIN 的复用功能，另一个就是设置这个 PIN 的电气特性。

格式说明：

管脚复用定义名 电气特性

如果需要在 iomuxc 中添加我们自定义外设的 PIN，那么需要新建一个子节点，然后将这个自定义外设的所有 PIN 配置信息都放到这个子节点中。

管脚复用定义名的说明，以 MX6UL_PAD_JTAG_TMS__GPIO1_IO11 为例：

之前我们释放了 JTAG_TMS 管脚，现在就可以根据芯片参考手册或者出厂内核源码的 arch/arm/boot/dts/imx6ul-pinfunc.h 文件，对这个管脚做复用功能选型。imx6ul-pinfunc.h 中对 JTAG_TMS 管脚的信息如下：

```
36  #define MX6UL_PAD_JTAG_TMS__SJC_TMS                  0x0048 0x02D4 0x0000 0x0 0x0
37  #define MX6UL_PAD_JTAG_TMS__GPT2_CAPTURE1            0x0048 0x02D4 0x0598 0x1 0x0
38  #define MX6UL_PAD_JTAG_TMS__SAI2_MCLK                0x0048 0x02D4 0x05F0 0x2 0x0
39  #define MX6UL_PAD_JTAG_TMS__CCM_CLKO1                0x0048 0x02D4 0x0000 0x3 0x0
40  #define MX6UL_PAD_JTAG_TMS__CCM_WAIT                 0x0048 0x02D4 0x0000 0x4 0x0
41  #define MX6UL_PAD_JTAG_TMS__GPIO1_IO11               0x0048 0x02D4 0x0000 0x5 0x0
42  #define MX6UL_PAD_JTAG_TMS__SDMA_EXT_EVENT01         0x0048 0x02D4 0x0614 0x6 0x0
43  #define MX6UL_PAD_JTAG_TMS__EPIT1_OUT                0x0048 0x02D4 0x0000 0x8 0x0
```

其中，MX6UL_PAD_JTAG_TMS__GPIO1_IO11 就是系统中 JTAG_TMS 用做 GPIO 功能时的管脚复用定义名。

电气特性说明：

电气特性，也就是 conf_reg 寄存器值。此值由用户自行设置，通过此值来设置一个 IO 的上/下拉、驱动能力和速度等，这部分可以结合芯片参考手册来设置。之前 JTAG 管脚设置电气特性为 0X80000000 仅为测试用，无特性。

以 MX6UL_PAD_CSI_DATA03__GPIO4_IO24 0XB0B1 为例：

0XB0B1 = 1011 0000 1011 0001

bit [16]:0 HYS 关闭

bit [15:14]: 10 100K Ohm 上拉

bit [13]: 1 Pull 功能

bit [12]: 1 pull/keeper 使能

bit [11]: 0 关闭开路输出

bit [7:6]: 10 速度 100Mhz

bit [5:3]: 110 R0/6 驱动能力

bit [0]: 1 快速转换率

芯片参考手册里的具体信息：

32.6.254 SW_PAD_CTL_PAD_CSI_DATA03 SW PAD Control Register (IOMUXC_SW_PAD_CTL_PAD_CSI_DATA03)

SW_PAD_CTL Register

Address: 20E_0000h base + 47Ch offset = 20E_047Ch

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_CSI_DATA03 field descriptions

Field	Description
31–17 -	This field is reserved. Reserved
16 HYS	Hyst. Enable Field Select one out of next values for pad: CSI_DATA03 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: CSI_DATA03 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: CSI_DATA03 0 **PUE_0_Keeper** — Keeper 1 **PUE_1_Pull** — Pull
12 PKE	Pull / Keep Enable Field Select one out of next values for pad: CSI_DATA03 0 **PKE_0_Pull_Keeper_Disabled** — Pull/Keeper Disabled 1 **PKE_1_Pull_Keeper_Enabled** — Pull/Keeper Enabled
11 ODE	Open Drain Enable Field Select one out of next values for pad: CSI_DATA03

IOMUXC_SW_PAD_CTL_PAD_CSI_DATA03 field descriptions (continued)

Field	Description
	0 **ODE_0_Open_Drain_Disabled** — Open Drain Disabled 1 **ODE_1_Open_Drain_Enabled** — Open Drain Enabled
10–8 -	This field is reserved. Reserved
7–6 SPEED	Speed Field Select one out of next values for pad: CSI_DATA03 00 **SPEED_0_low_50MHz_** — low(50MHz) 01 **SPEED_1_medium_100MHz_** — medium(100MHz) 10 **SPEED_2_medium_100MHz_** — medium(100MHz) 11 **SPEED_3_max_200MHz_** — max(200MHz)
5–3 DSE	Drive Strength Field Select one out of next values for pad: CSI_DATA03 000 **DSE_0_output_driver_disabled_** — output driver disabled; 001 **DSE_1_R0_260_Ohm___3_3V__150_Ohm_1_8V__240_Ohm_for_DDR_** — R0(260 Ohm @ 3.3V, 150 Ohm@1.8V, 240 Ohm for DDR) 010 **DSE_2_R0_2** — R0/2 011 **DSE_3_R0_3** — R0/3 100 **DSE_4_R0_4** — R0/4 101 **DSE_5_R0_5** — R0/5 110 **DSE_6_R0_6** — R0/6 111 **DSE_7_R0_7** — R0/7
2–1 -	This field is reserved. Reserved
0 SRE	Slew Rate Field Select one out of next values for pad: CSI_DATA03 0 **SRE_0_Slow_Slew_Rate** — Slow Slew Rate 1 **SRE_1_Fast_Slew_Rate** — Fast Slew Rate

8.3 GPIO 测试

修改完 GPIO 功能配置后，就可以进行 GPIO 测试。前面的裁剪篇中有介绍到 uboot 和系统中的 GPIO 测试，这里不再重复。

这里介绍一下如何查看开发板出厂文件系统 IO 的使用情况。可以在开发板出厂文件系统中执行下面指令：

```
cat /sys/kernel/debug/pinctrl/20e0000.iomuxc/pinmux-pins
```

打印示例：

```
Pinmux settings per pin
Format: pin (name): mux_owner gpio_owner hog?
pin 0 (MX6UL_PAD_RESERVE0): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 1 (MX6UL_PAD_RESERVE1): (MUX UNCLAIMED) (GPIO UNCLAIMED)
……
pin 17 (MX6UL_PAD_JTAG_MOD): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 18 (MX6UL_PAD_JTAG_TMS): 202c000.sai (GPIO UNCLAIMED) function imx6ul-evk group sai2grp
```

```
pin 19 (MX6UL_PAD_JTAG_TDO): 202c000.sai (GPIO UNCLAIMED) function imx6ul-evk group sai2grp
pin 20 (MX6UL_PAD_JTAG_TDI): 202c000.sai (GPIO UNCLAIMED) function imx6ul-evk group sai2grp
pin 21 (MX6UL_PAD_JTAG_TCK): 202c000.sai (GPIO UNCLAIMED) function imx6ul-evk group sai2grp
......
pin 24 (MX6UL_PAD_GPIO1_IO01): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 25 (MX6UL_PAD_GPIO1_IO02): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 26 (MX6UL_PAD_GPIO1_IO03): leds (GPIO UNCLAIMED) function imx6ul-evk group gpio-leds
......
pin 128 (MX6UL_PAD_CSI_DATA07): (MUX UNCLAIMED) (GPIO UNCLAIMED)
```

示例中，(MUX UNCLAIMED) (GPIO UNCLAIMED)表示当前管脚没有配置功能，例如 pin 24 (MX6UL_PAD_GPIO1_IO01)： (MUX UNCLAIMED) (GPIO UNCLAIMED)，表明 MX6UL_PAD_GPIO1_IO01 这个管脚可以直接做 GPIO 使用。

pin 18 (MX6UL_PAD_JTAG_TMS): 202c000.sai (GPIO UNCLAIMED) function imx6ul-evk group sai2grp，表明 MX6UL_PAD_JTAG_TMS 这个管脚，在系统中已经配置成 sai 音频相关的功能，可以结合设备树来分析。

8.4 程序测试 GPIO 输出\输入\中断

参考本文 3.4 小节。

第九章 8 路 UART 复用

I.MX6ULL 常用于工业数据采集，硬件上最多支持 8 路 UART（其中 1 路为 UART1，用于调试使用，其他 7 路 UART 可以做数据采集），但由于存在复用冲突的问题，配置为 8 路 UART 后需要裁剪掉某些外设功能。常见的 8 路 UART 方案如下：

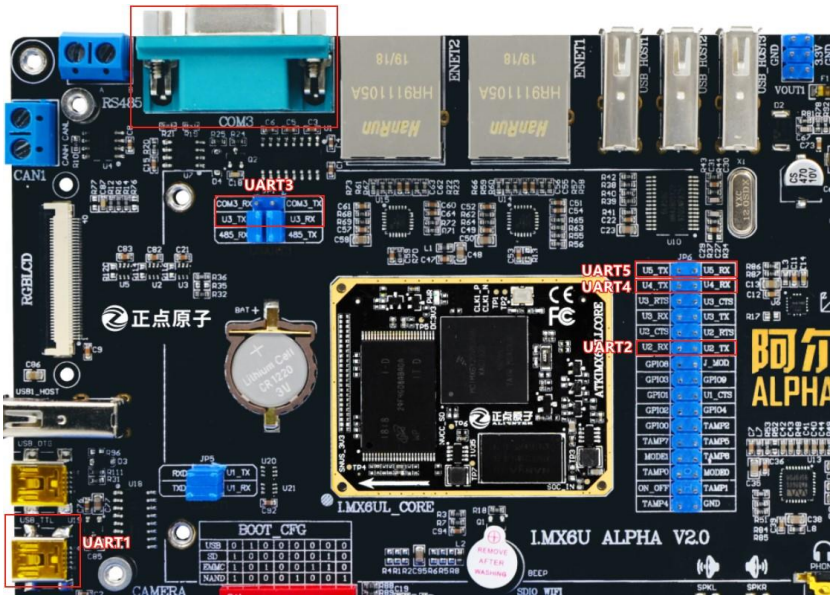
方案	需求	备注
方案一	单独 8 路 UART+LCD	裁剪 ENET2、CAMERA
方案二	8 路 UART+1 路百兆网口+LCD	裁剪 ENET2、CAMERA
方案三	8 路 UART+2 路百兆网口	裁剪 LCD、CAMREA

下面以方案一、二为例，配置 8 路 UART 功能。

使用到的源码：正点原子出厂系统内核源码。

硬件：正点原子 I.MX6ULL 开发板或者基于 I.MX6ULL 核心板做的底板。I.MX6ULL 开发板上能直接复用 5 路串口使用（UART1\2\3\4\5），其他 3 路串口（UART6\7\8）需要自行做底板设计时引出。这部分的 UART6\7\8，笔者是通过裁剪 ENET2 得到的。

开发板板载了串口 1 和串口 3 的接口，以及一排的管脚排针。出厂系统默认配置了串口 1（ttymxc0）和串口 3（ttymxc2），排针上有引出串口 2（ttymxc1）、串口 4（ttymxc3）和串口 5（ttymxc4）



9.1 UART1（ttymxc0）

系统默认配置，做调试串口使用，不需要修改。

9.2 UART2（ttymxc1）

打开 imx6ull-14x14-evk.dts，屏蔽掉 MX6UL_PAD_UART2_TX_DATA 和 MX6UL_PAD_UART2_RX_DATA 其他复用功能，只保留 TX_DATA 和 RX_DATA 功能。

```
531                 pinctrl_ecspi3: ecspi3grp {
532                         fsl,pins = <
533                                 MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO       0x100b1
534                                 MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI       0x100b1
535 /*                                  MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK     0x100b1
536                                 MX6UL_PAD_UART2_TX_DATA__GPIO1_IO20      0x100b0  */
537                         >;
538                 };
```

```
675
676         pinctrl_uart2dte: uart2dtegrp {
677             fsl,pins = <
678 /*              MX6UL_PAD_UART2_TX_DATA__UART2_DTE_RX   0x1b0b1
679                 MX6UL_PAD_UART2_RX_DATA__UART2_DTE_TX   0x1b0b1 */
680                 MX6UL_PAD_UART3_RX_DATA__UART2_DTE_CTS  0x1b0b1
681                 MX6UL_PAD_UART3_TX_DATA__UART2_DTE_RTS  0x1b0b1
682             >;
683         };
```

添加/修改节点标签&uart2，只使用 pinctrl_uart2，注释掉 fsl,uart-has-rtscts; status 修改为 "okay"，修改完如下

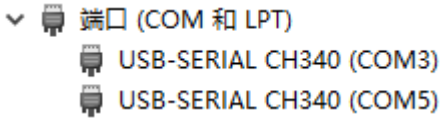
```
&uart2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart2>;
    /*fsl,uart-has-rtscts;  */
    /* for DTE mode, add below change */
    /* fsl,dte-mode; */
    /* pinctrl-0 = <&pinctrl_uart2dte>; */
    status = "okay";
};
```

保存文件，执行 build.sh 脚本编译内核源码生成设备树文件，将生成的设备树文件更新到开发板上，启动开发板，查看是否存在 ttymxc1。

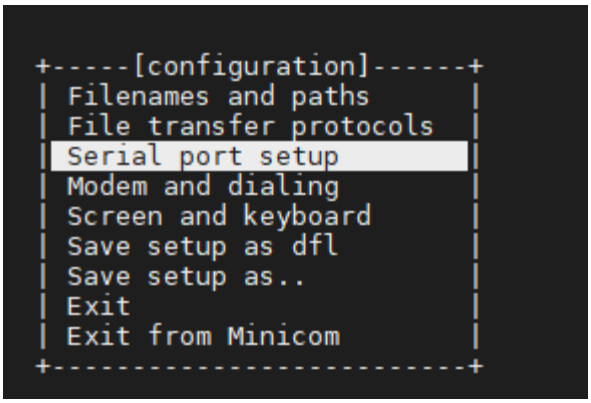
```
root@ATK-IMX6U:~# ls /dev/ttymxc*
/dev/ttymxc0  /dev/ttymxc1  /dev/ttymxc2
root@ATK-IMX6U:~#
```

测试

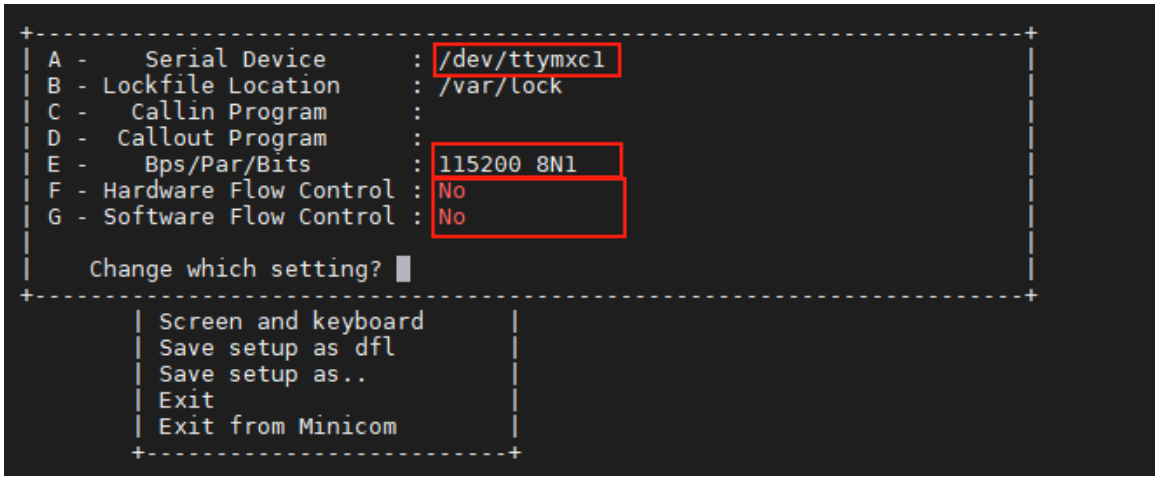
接线，将开发板引出的 U2_RX 接到串口的 TX，U2_TX 接到串口的 RX，查看设备管理器中串口端口号，这里为 COM3。开发板 UART1 用于查看信息，为 COM5



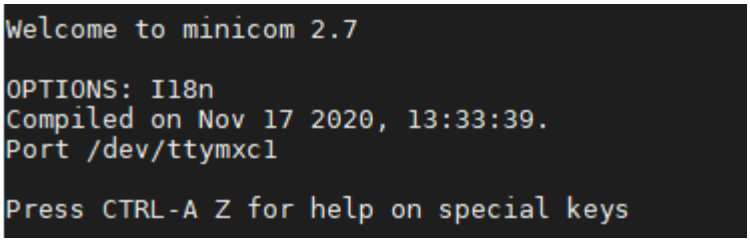
开发板输入 `minicom -s`，打开 minicom 配置界面，选择 Serial port setup



按回车键进入，设置菜单如下，输入 A\B\C\D\E\F\G 可以依次进入对应的设置选项：

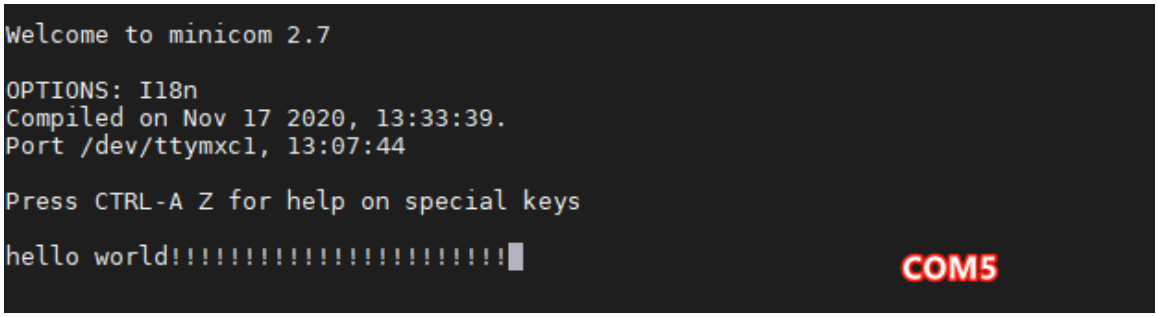


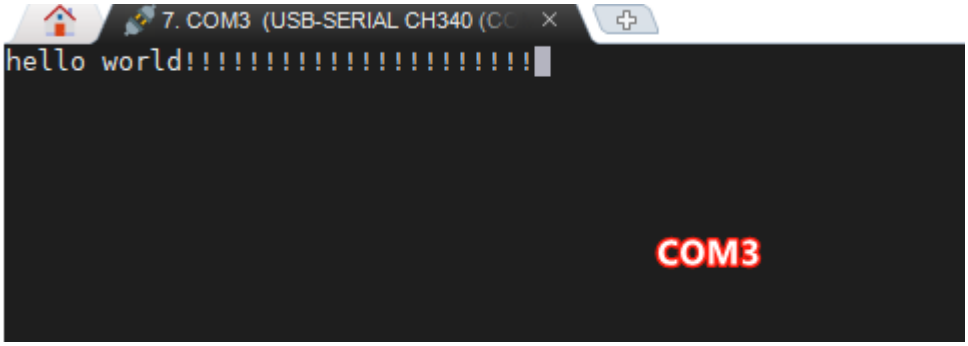
设置完保存后退出菜单，回到如下界面：



按下 CTRL 键和 A 键，再按 Z 键，得到如图所示的选项栏，按下 E 键打开回显功能。

打开 COM3，在串口 1（COM5，开发板文件系统）输入，在 COM3 会有显示，表示 UART2 已正常使用。





9.3 UART3（ttymxc2）

出厂系统上默认配置 UART3 了，我们这边需要对出厂源码稍微修改下，防止 UART2 和 UART3 功能出现冲突。

打开 imx6ull-14x14-evk.dts ，屏蔽掉 MX6UL_PAD_UART3_TX_DATA 和 MX6UL_PAD_UART3_RX_DATA 其他复用功能，只保留 TX_DATA 和 RX_DATA 功能。

```
667         pinctrl_uart2: uart2grp {
668             fsl,pins = <
669                 MX6UL_PAD_UART2_TX_DATA__UART2_DCE_TX   0x1b0b1
670                 MX6UL_PAD_UART2_RX_DATA__UART2_DCE_RX   0x1b0b1
671 /*              MX6UL_PAD_UART3_RX_DATA__UART2_DCE_RTS  0x1b0b1
672                 MX6UL_PAD_UART3_TX_DATA__UART2_DCE_CTS  0x1b0b1 */
673             >;
674         };
675
676         pinctrl_uart2dte: uart2dtegrp {
677             fsl,pins = <
678 /*              MX6UL_PAD_UART2_TX_DATA__UART2_DTE_RX   0x1b0b1
679                 MX6UL_PAD_UART2_RX_DATA__UART2_DTE_TX   0x1b0b1 */
680 /*              MX6UL_PAD_UART3_RX_DATA__UART2_DTE_CTS  0x1b0b1
681                 MX6UL_PAD_UART3_TX_DATA__UART2_DTE_RTS  0x1b0b1 */
682             >;
683         };
```

保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc2，然后对其测试。

9.4 UART4（ttymxc3）

在 imx6ul-pinfunc.h 文件中搜索 UART4_TX_DATA 和 UART4_RX_DATA 得到 UART4 收发引脚的定义，即：

MX6UL_PAD_UART4_TX_DATA__UART4_DCE_TX
MX6UL_PAD_UART4_RX_DATA__UART4_DCE_RX

修改设备树 imx6ull-14x14-evk.dts，搜索 uart4，注释掉相关复用管脚功能。

```
525         pinctrl_i2c1: i2c1grp {
526             fsl,pins = <
527 /*              MX6UL_PAD_UART4_TX_DATA__I2C1_SCL 0x4001b8b0
528             |   MX6UL_PAD_UART4_RX_DATA__I2C1_SDA 0x4001b8b0      */
529             >;
```

disabled 掉相关的复用节点。

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

添加 uart4 节点（ps：可以拷贝 uart3 的来修改）

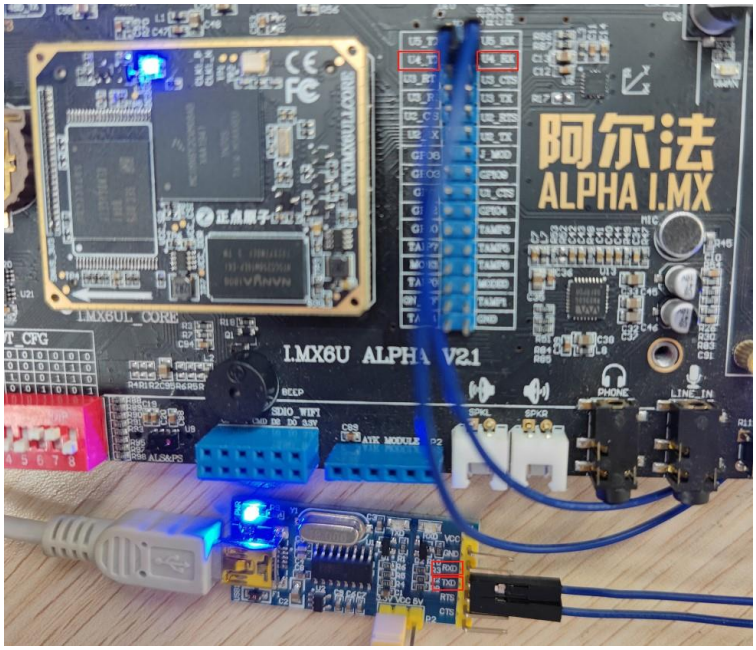
```
644         pinctrl_uart3: uart3grp {
645             fsl,pins = <
646                 MX6UL_PAD_UART3_RX_DATA__UART3_DCE_RX   0x1b0b1
647                 MX6UL_PAD_UART3_TX_DATA__UART3_DCE_TX   0x1b0b1
648             >;
649         };
650
651         pinctrl_uart4: uart4grp {
652         fsl,pins = <
653             MX6UL_PAD_UART4_RX_DATA__UART4_DCE_RX   0x1b0b1
654             MX6UL_PAD_UART4_TX_DATA__UART4_DCE_TX   0x1b0b1
655         >;
656     };
```

添加/修改节点标签&uart4（ps：可以拷贝 uart3 的来修改）

```
930  &uart3 {
931      pinctrl-names = "default";
932      pinctrl-0 = <&pinctrl_uart3>;
933      status = "okay";
934  };
935
936  &uart4 {
937      pinctrl-names = "default";
938      pinctrl-0 = <&pinctrl_uart4>;
939      status = "okay";
940  };
```

保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc3。

接线，这里笔者使用 USB 转串口模块，开发板 U4_TX 引脚接模块的 RXD 引脚，U4_RX 引脚接模块的 TXD 引脚。



软件配置：minicom 配置如下：

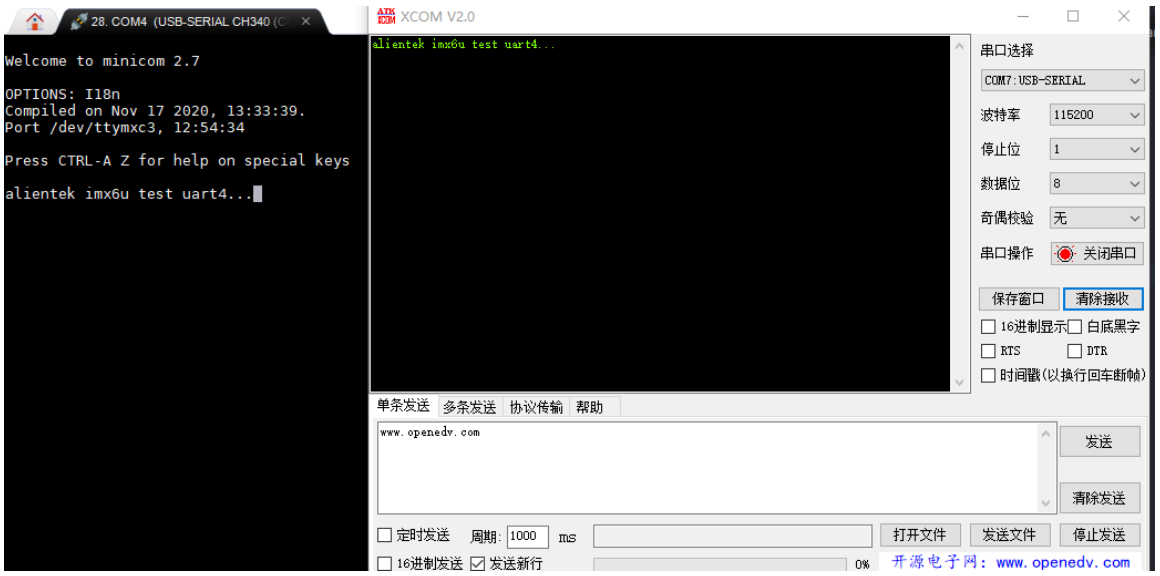
```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttymxc3                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

在 minicom 界面按下 crtl+a 后，依次按下 z 和 e，打开回显。

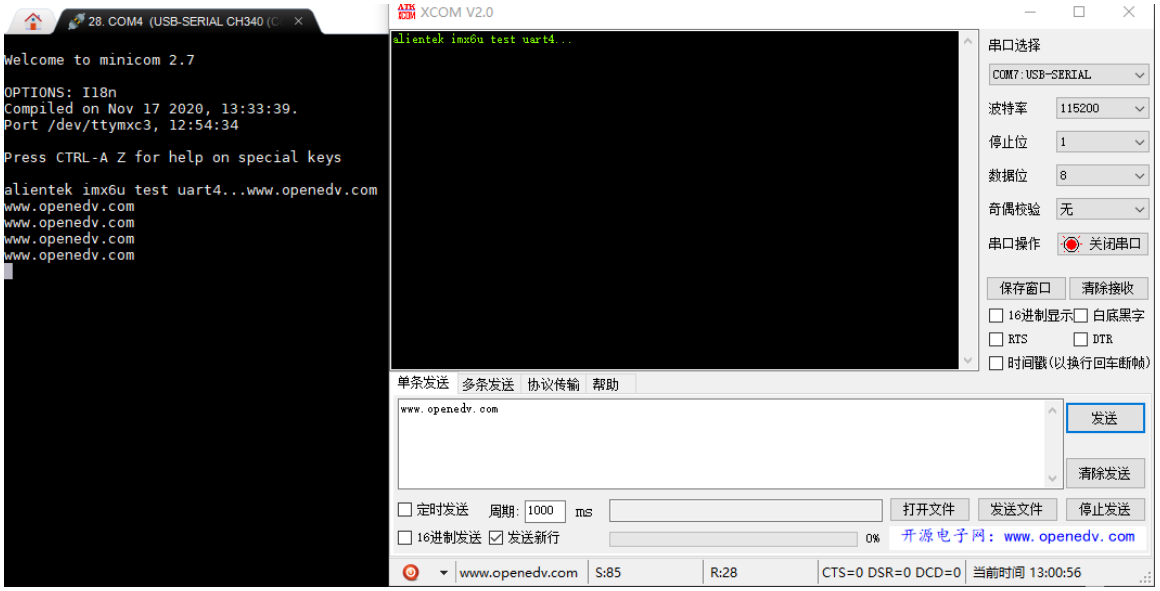
打开另一个串口终端，选择 USB 转串口模块对应的 COM 口，配置和上面 minicom 配置一样，波特率 115200，8N1，无控制流

在开发板终端 minicom 输入字符后，在模块 COM 口处可以接收。

发送测试：



接收测试：



在 minicom 界面按下 crtl+a 后，依次按下 z 和 x，退出 minicom 测试。

9.5 UART5（ttymxc4）

在 imx6ul-pinfunc.h 文件中搜索 UART5_TX_DATA 和 UART5_RX_DATA 得到 UART5 收发引脚的定义，即：

```
MX6UL_PAD_UART5_TX_DATA__UART5_DCE_TX
MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX
```

这里值得注意的是，NXP 官方源码这里寄存器配置错误了！

```
#define MX6UL_PAD_UART5_TX_DATA__I2C2_SCL                0x00BC 0x0348 0x05AC 0x2 0x2
#define MX6UL_PAD_UART5_TX_DATA__CSI_DATA14              0x00BC 0x0348 0x04FC 0x3 0x0
#define MX6UL_PAD_UART5_TX_DATA__CSU_CSU_ALARM_AUT00     0x00BC 0x0348 0x0000 0x4 0x0
#define MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX            0x00C0 0x034C 0x0644 0x0 0x5
#define MX6UL_PAD_UART5_RX_DATA__UART5_DTE_TX            0x00C0 0x034C 0x0000 0x0 0x0
#define MX6UL_PAD_UART5_RX_DATA__ENET2_COL               0x00C0 0x034C 0x0000 0x1 0x0
```

需要进行以下修改，将：

```
#define MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX       0x00C0 0x034C 0x0644 0x0 0x5
```

修改为：

```
#define MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX       0x00C0 0x034C 0x0644 0x0 0x7
```

修改完的配置：

```
#define MX6UL_PAD_UART5_TX_DATA__I2C2_SCL                0x00BC 0x0348 0x05AC 0x2 0x2
#define MX6UL_PAD_UART5_TX_DATA__CSI_DATA14              0x00BC 0x0348 0x04FC 0x3 0x0
#define MX6UL_PAD_UART5_TX_DATA__CSU_CSU_ALARM_AUT00     0x00BC 0x0348 0x0000 0x4 0x0
#define MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX            0x00C0 0x034C 0x0644 0x0 0x7
#define MX6UL_PAD_UART5_RX_DATA__UART5_DTE_TX            0x00C0 0x034C 0x0000 0x0 0x0
#define MX6UL_PAD_UART5_RX_DATA__ENET2_COL               0x00C0 0x034C 0x0000 0x1 0x0
```

修改完保存文件。没有修改此文件的话，UART5 只能发送，不能接收。

修改设备树 imx6ull-14x14-evk.dts，搜索 uart5，注释掉相关复用管脚功能。

```
        pinctrl_i2c2: i2c2grp {
            fsl,pins = <
/*              MX6UL_PAD_UART5_TX_DATA__I2C2_SCL 0x4001b8b0        */
/*              MX6UL_PAD_UART5_RX_DATA__I2C2_SDA 0x4001b8b0        */
            >;
        };
```

注意：这里 I2C2 配置为 LCD 的触摸功能，屏蔽掉后屏幕无法触摸。如果不想冲突，可以用后面 CAMERA 复用为 UART5 小节的修改方法。

disabled 掉相关复用节点。

```
&i2c2 {
    clock-frequency = <100000>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_i2c2>;
    status = "disabled";
```

添加 uart5 节点。

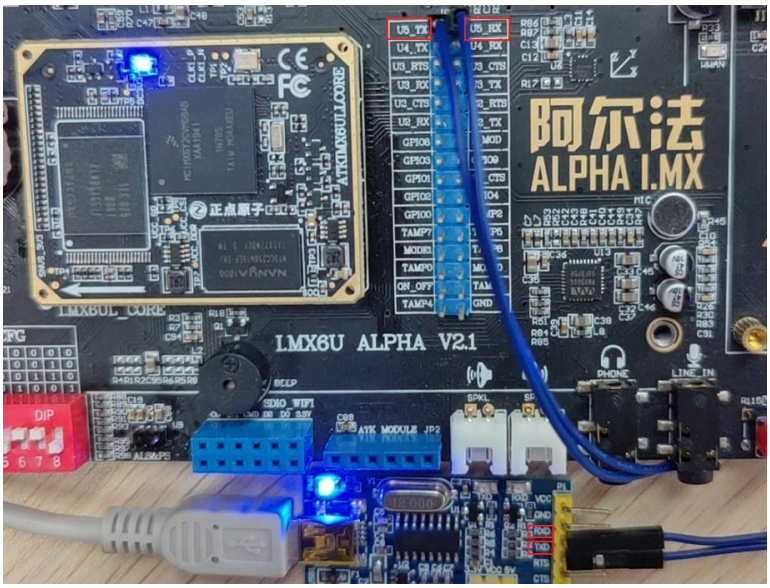
```
643
644         pinctrl_uart3: uart3grp {
645             fsl,pins = <
646                 MX6UL_PAD_UART3_RX_DATA__UART3_DCE_RX   0x1b0b1
647                 MX6UL_PAD_UART3_TX_DATA__UART3_DCE_TX   0x1b0b1
648             >;
649         };
650
651         pinctrl_uart4: uart4grp {
652         fsl,pins = <
653             MX6UL_PAD_UART4_RX_DATA__UART4_DCE_RX   0x1b0b1
654             MX6UL_PAD_UART4_TX_DATA__UART4_DCE_TX   0x1b0b1
655         >;
656     };
657
658         pinctrl_uart5: uart5grp {
659         fsl,pins = <
660             MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX   0x1b0b1
661             MX6UL_PAD_UART5_TX_DATA__UART5_DCE_TX   0x1b0b1
662         >;
663     };
```

添加节点标签。

```
937     &uart3 {
938         pinctrl-names = "default";
939         pinctrl-0 = <&pinctrl_uart3>;
940         status = "okay";
941     };
942
943     &uart4 {
944         pinctrl-names = "default";
945         pinctrl-0 = <&pinctrl_uart4>;
946         status = "okay";
947     };
948
949     &uart5 {
950         pinctrl-names = "default";
951         pinctrl-0 = <&pinctrl_uart5>;
952         status = "okay";
953     };
```

保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc4。

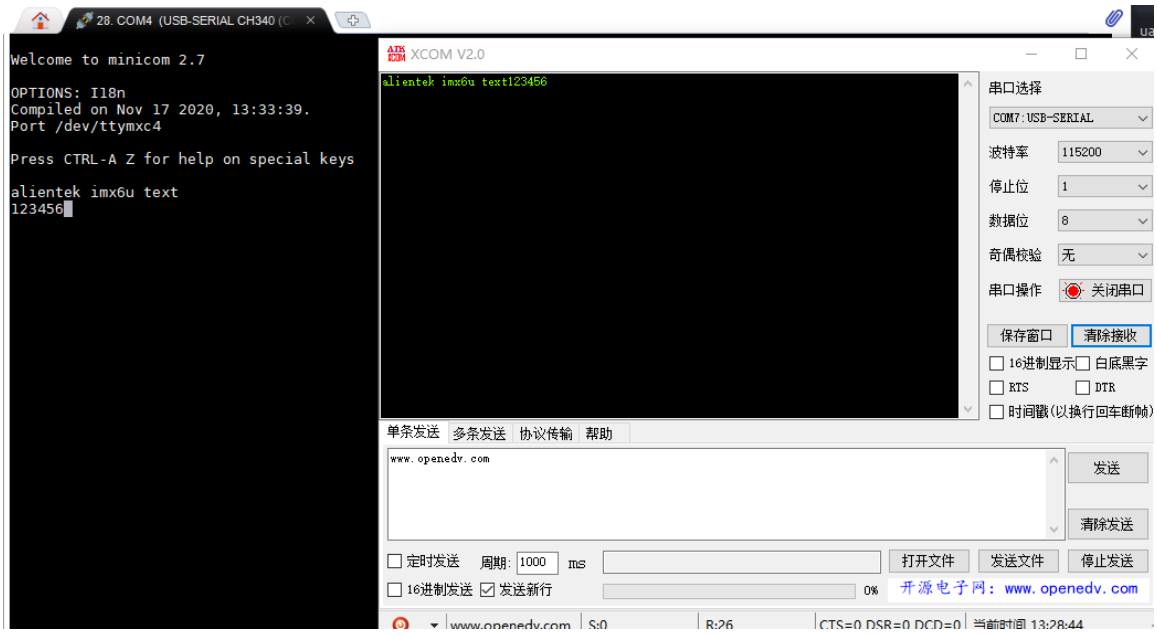
接线，这里笔者使用 USB 转串口模块，开发板 U5_TX 引脚接模块的 RXD 引脚，U5_RX 引脚接模块的 TXD 引脚。



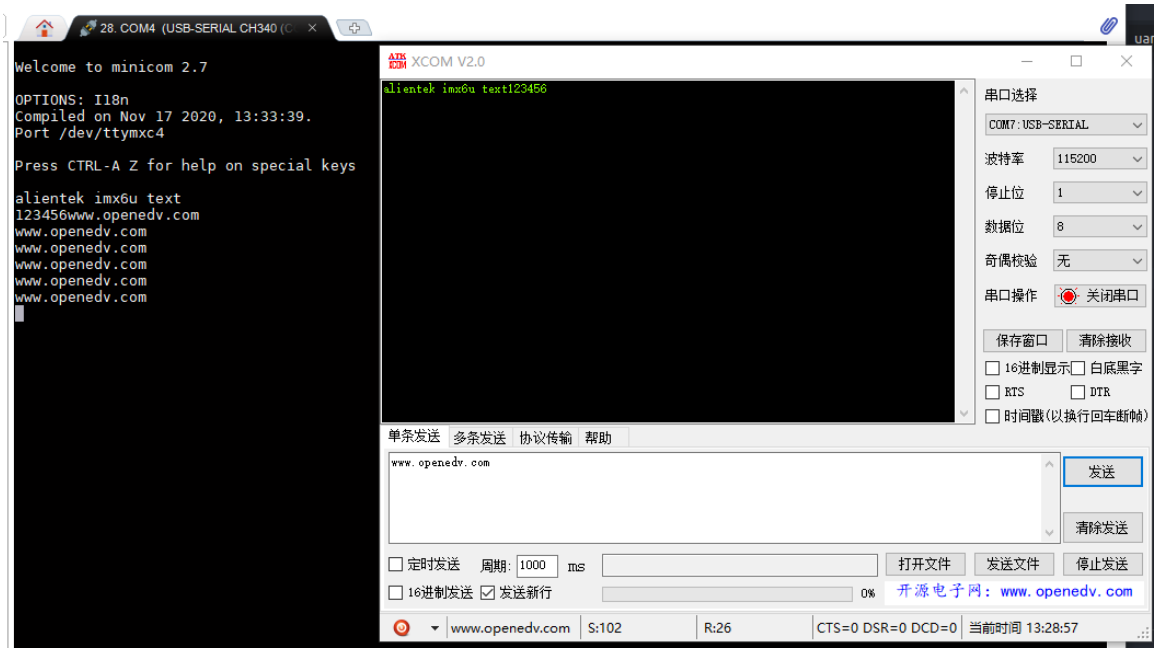
minicom 配置如下：

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttymxc4                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

发送测试：



接收测试：



至此，基于 I.MX6ULL 开发板的五路串口配置完成实现。

9.6 UART6（ttymxc5）

这部分的 UART6\7\8，笔者是通过裁剪 ENET2 得到的。ENET2 的裁剪可以参考前面对应的裁剪章节。这里假设已经完成裁剪并测试管脚的步骤。

打开 imx6ul-pinfunc.h，搜索 UART6，找到板子上对应的管脚及 UART6 功能，这里我是 GPIO2_IO08 做 UART6_DCE_TX，GPIO2_IO09 做 UART6_DCE_ RX。

在&iomuxc 节点下添加 pinctrl_uart6 节点信息。

```
706         pinctrl_uart6: uart6grp {
707             fsl,pins = <
708                 MX6UL_PAD_ENET2_RX_DATA1__UART6_DCE_RX  0x1b0b1
709                 MX6UL_PAD_ENET2_RX_DATA0__UART6_DCE_TX  0x1b0b1
710             >;
711         };
```

屏蔽掉 MX6UL_PAD_ENET2_RX_DATA1 和 MX6UL_PAD_ENET2_RX_DATA0 的其他复用功能和节点。根节点下添加&uart6 节点。

```
1009    &uart6 {
1010        pinctrl-names = "default";
1011        pinctrl-0 = <&pinctrl_uart6>;
1012        status = "okay";
1013    };
```

保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc5 并测试串口收发功能。

9.7 UART7（ttymxc6）

这部分的 UART6\7\8，笔者是通过裁剪 ENET2 得到的。ENET2 的裁剪可以参考前面对应的裁剪章节。这里假设已经完成裁剪并测试管脚的步骤。

打开 imx6ul-pinfunc.h，搜索 UART7，找到板子上对应的管脚及 UART7 功能，这里我是 GPIO2_IO10 做 UART7_DCE_TX，GPIO2_IO11 做 UART7_DCE_ RX。

在&iomuxc 节点下添加 pinctrl_uart7 节点信息。

```
pinctrl_uart7: uart7grp {
    fsl,pins = <
        MX6UL_PAD_ENET2_TX_DATA0__UART7_DCE_RX  0x1b0b1
        MX6UL_PAD_ENET2_RX_EN__UART7_DCE_TX 0x1b0b1
    >;
};
```

屏蔽掉 MX6UL_PAD_ENET2_TX_DATA0 和 MX6UL_PAD_ENET2_RX_EN 的其他复用功能和节点。

根节点下添加&uart7 节点。

```
&uart7 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart7>;
    status = "okay";
};
```

保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc6 并测试串口收发功能。

9.8 UART8（ttymxc7）

这部分的 UART6\7\8，笔者是通过裁剪 ENET2 得到的。ENET2 的裁剪可以参考前面对应的裁剪章节。这里假设已经完成裁剪并测试管脚的步骤。

打开 imx6ul-pinfunc.h，搜索 UART8，找到板子上对应的管脚及 UART8 功能，这里我是 GPIO2_IO12 做 UART8_DCE_TX，GPIO2_IO13 做 UART8_DCE_ RX。

```
pinctrl_uart8: uart8grp {
    fsl,pins = <
        MX6UL_PAD_ENET2_TX_EN__UART8_DCE_RX 0x1b0b1
        MX6UL_PAD_ENET2_TX_DATA1__UART8_DCE_TX  0x1b0b1
    >;
};
```

屏蔽掉 MX6UL_PAD_ENET2_TX_DATA1 和 MX6UL_PAD_ENET2_TX_EN 的其他复用功能和节点。

根节点下添加&uart8 节点。可以参考前面&uart7 的示例。

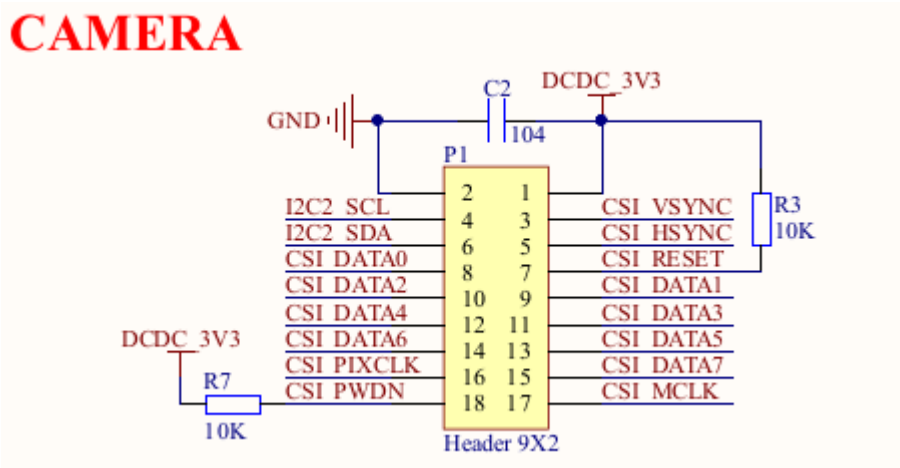
保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc7 并测试串口收发功能。

9.9 CAMERA 复用为 UART5\6

前面我们说过，直接用开发板上引出的 UART5 的话，会和出厂系统上 LCD 的触摸功能冲突，因此需要修改别的外设来复用成 UART5 功能。恰好开发板上的 CAMERA 接口上有管脚可以复用成 UART5，还能复用成 UART6，如下表所示：

CAMERA 管脚	复用功能	GPIO
CSI_DATA0	UART5_TX	GPIO4_IO21
CSI_DATA1	UART5_RX	GPIO4_IO22
CSI_MCLK	UART6_TX	GPIO4_IO17
CSI_PIXCLK	UART6_RX	GPIO4_IO18

以阿尔法开发板为例，开发板 camera 接口原理图：



这里假设已经完成第三章 CAMERA 裁剪的内核裁剪和设备树裁剪的步骤。

首先编译下已经裁剪完摄像头接口的内核。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make del_camera_imx_v7_defconfig
make zImage -j 16
```

打开出厂内核源码，摄像头相关的内容在之前的裁剪章节已经注释掉了，这里我们要继续添加 UART 相关的配置。

打开 arch/arm/boot/dts/imx6ul-pinfunc.h，搜索 UART5，找到板子上对应的管脚及 UART6 功能，这里我是 CSI_DATA00 做 UART5_DCE_TX，CSI_DATA01 做 UART5_DCE_ RX。

搜索 UART6，找到板子上对应的管脚及 UART6 功能，这里我是 CSI_MCLK 做 UART6_DCE_TX，CSI_PIXCLK 做 UART6_DCE_ RX。

打开 arch/arm/boot/dts/imx6ull-14x14-evk.dts，在 &iomuxc 节点下添加 pinctrl_uart5 和 pinctrl_uart6 节点信息。

```
678         pinctrl_uart5: uart5grp {
679             fsl,pins = <
680                 MX6UL_PAD_CSI_DATA00__UART5_DCE_TX  0x1b0b1
681                 MX6UL_PAD_CSI_DATA01__UART5_DCE_RX  0x1b0b1
682             >;
683         };
684
685         pinctrl_uart6: uart6grp {
686             fsl,pins = <
687                 MX6UL_PAD_CSI_MCLK__UART6_DCE_TX    0x1b0b1
688                 MX6UL_PAD_CSI_PIXCLK__UART6_DCE_RX  0x1b0b1
689             >;
690         };
```

根节点下添加&uart5 和&uart6 节点。

```
974 &uart5 {
975     pinctrl-names = "default";
976     pinctrl-0 = <&pinctrl_uart5>;
977     status = "okay";
978 };
979
980 &uart6 {
981     pinctrl-names = "default";
982     pinctrl-0 = <&pinctrl_uart6>;
983     status = "okay";
984 };
```

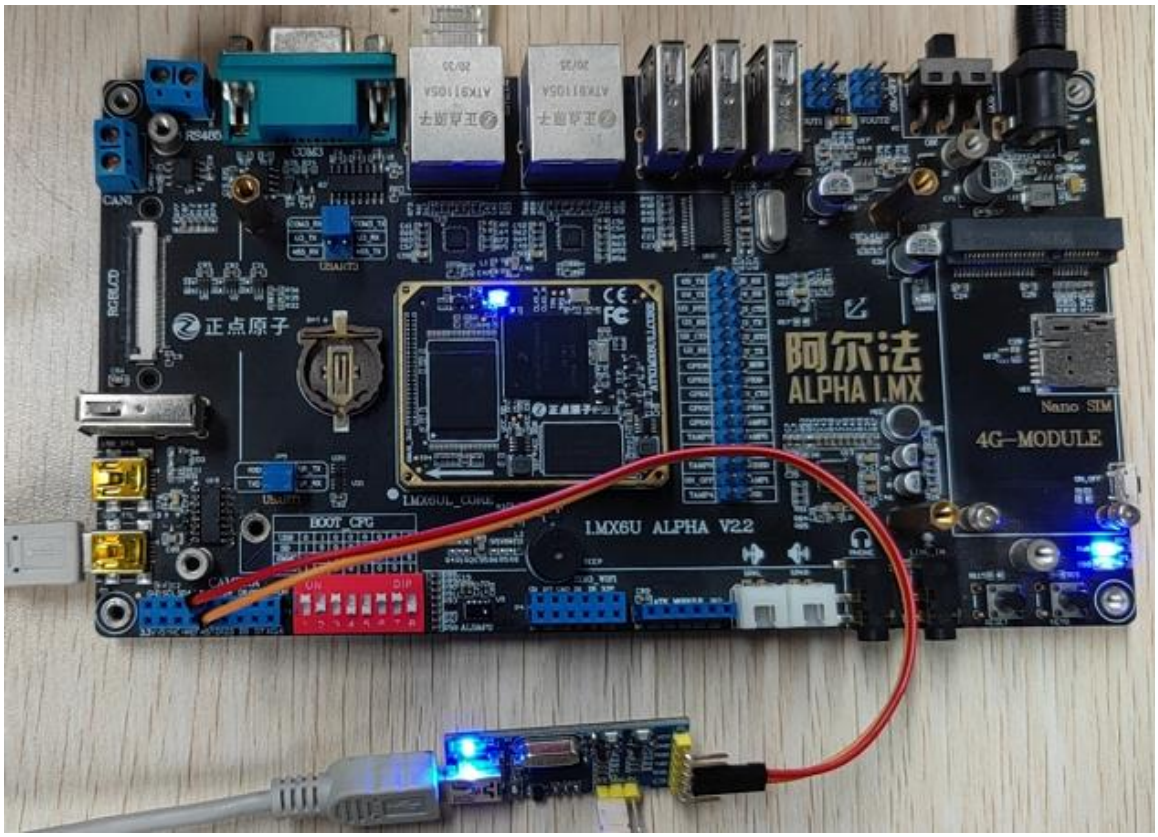
保存文件，编译内核源码生成设备树文件，用此文件启动开发板，查看是否存在 ttymxc4 和 ymxc5 并测试串口收发功能。具体设备树可以根据核心板型号和屏幕选择，没屏幕默认用-4.3-480x272 的配置。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make del_camera_imx_v7_defconfig
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

```
root@ATK-IMX6U:~# ls /dev/ttymxc*
/dev/ttymxc0  /dev/ttymxc2  /dev/ttymxc4  /dev/ttymxc5
```

UART5 测试：

接线，这里笔者使用 USB 转串口模块，开发板 CSI_DATA0 引脚接模块的 RXD 引脚，CSI_DATA1 脚接模块的 TXD 引脚。



minicom 配置如下：

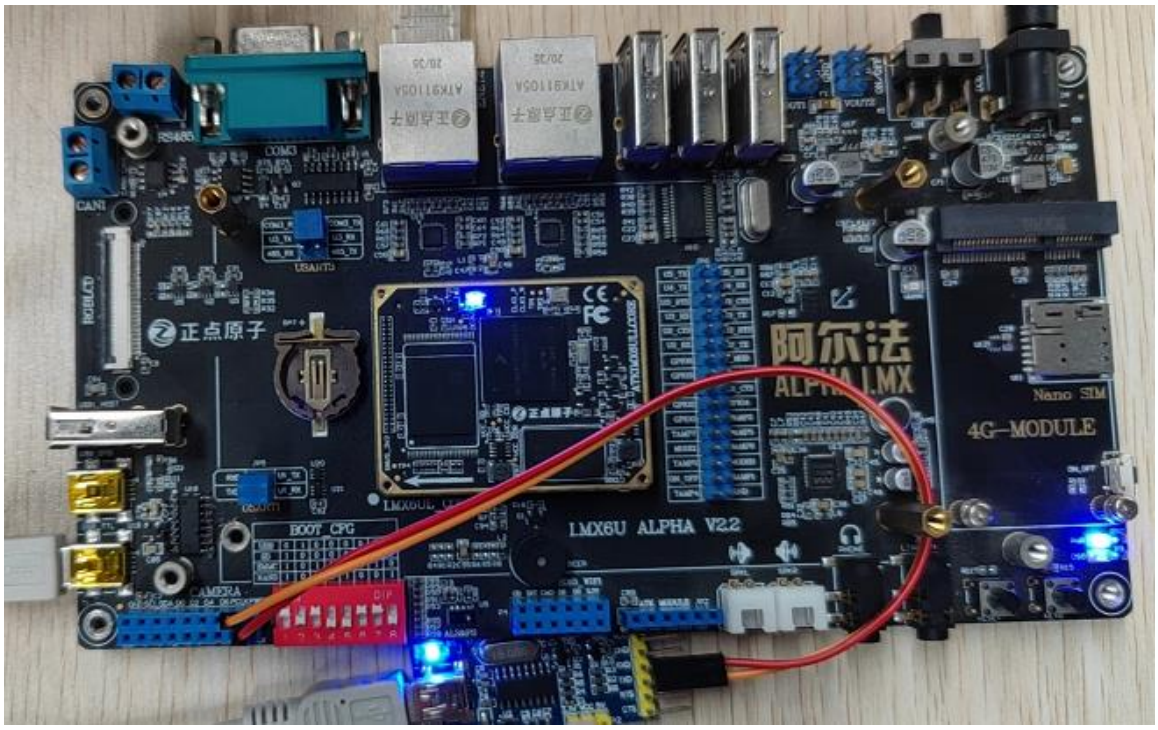
```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttymxc4                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

发送测试：

接收测试：

UART6 测试：

接线，这里笔者使用 USB 转串口模块，开发板 CSI_MCLK 引脚接模块的 RXD 引脚，CSI_PIXCLK 脚接模块的 TXD 引脚。



minicom 配置如下：

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttymxc5                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

发送测试：

接收测试：

至此，CAMERA 复用 UART5\6 功能修改和测试结束。

9.10 串口应用程序测试

测试程序可以参考《I.MX6U 嵌入式 Linux C 应用编程指南 V1.4》26 章串口应用编程。

第十章 2 路 CAN 复用

I.MX6ULL 最多可以使用两路 CAN，在阿尔法板和 Linux MINI 开发板上默认配置了一路 CAN1 可以给我们参考，我们只需要添加一路 CAN2。

核心板上有多路管脚可以复用成 CAN2，具体可以看资料里的 imx6ull 核心板引脚分配图.pdf。这里以阿尔法和 MINI 开发板为例，将开发板上的 U2_CTS 和 U2_RTS 复用为 CAN2 功能。

10.1 修改设备树文件

打开出厂内核源码 arch/arm/boot/dts/imx6ul-pinfunc.h 文件。

搜索 MX6UL_PAD_UART2_CTS_B 和 MX6UL_PAD_UART2_RTS_B，找到对应的复用成 CAN 的管脚定义。

```
#define MX6UL_PAD_UART2_CTS_B__UART2_DCE_CTS
#define MX6UL_PAD_UART2_CTS_B__UART2_DTE_RTS
#define MX6UL_PAD_UART2_CTS_B__ENET1_CRS
#define MX6UL_PAD_UART2_CTS_B__FLEXCAN2_TX
#define MX6UL_PAD_UART2_CTS_B__CSI_DATA08
#define MX6UL_PAD_UART2_CTS_B__GPT1_COMPARE2
#define MX6UL_PAD_UART2_CTS_B__GPIO1_IO22
#define MX6UL_PAD_UART2_CTS_B__SJC_DE_B
#define MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI
#define MX6UL_PAD_UART2_RTS_B__UART2_DCE_RTS
#define MX6UL_PAD_UART2_RTS_B__UART2_DTE_CTS
#define MX6UL_PAD_UART2_RTS_B__ENET1_COL
#define MX6UL_PAD_UART2_RTS_B__FLEXCAN2_RX
#define MX6UL_PAD_UART2_RTS_B__CSI_DATA09
#define MX6UL_PAD_UART2_RTS_B__GPT1_COMPARE3
#define MX6UL_PAD_UART2_RTS_B__GPIO1_IO23
#define MX6UL_PAD_UART2_RTS_B__SJC_FAIL
#define MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO
```

打开出厂内核源码 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件。

搜索 MX6UL_PAD_UART2_CTS_B 和 MX6UL_PAD_UART2_RTS_B，屏蔽掉这两个管脚除 CAN 功能外的的复用功能。

```
pinctrl_ecspi3: ecspi3grp {
        fsl,pins = <
/*                      MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO      0x100b1
                      MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI      0x100b1    */
                      MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK    0x100b1  /* CLK*/
                      MX6UL_PAD_UART2_TX_DATA__GPIO1_IO20     0x100b0  /* CS*/
        >;
};
```

出厂系统内核源码上保留了 CAN2 的配置，这里不需要修改。

```
pinctrl_flexcan2: flexcan2grp{
    fsl,pins = <
        MX6UL_PAD_UART2_RTS_B__FLEXCAN2_RX  0x1b020
        MX6UL_PAD_UART2_CTS_B__FLEXCAN2_TX  0x1b020
    >;
};
```

搜索 pinctrl_flexcan2，找到&flexcan2，将 status 修改为 okay，如下：

```
&flexcan2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_flexcan2>;
    xceiver-supply = <&reg_can_3v3>;
    status = "okay";
};
```

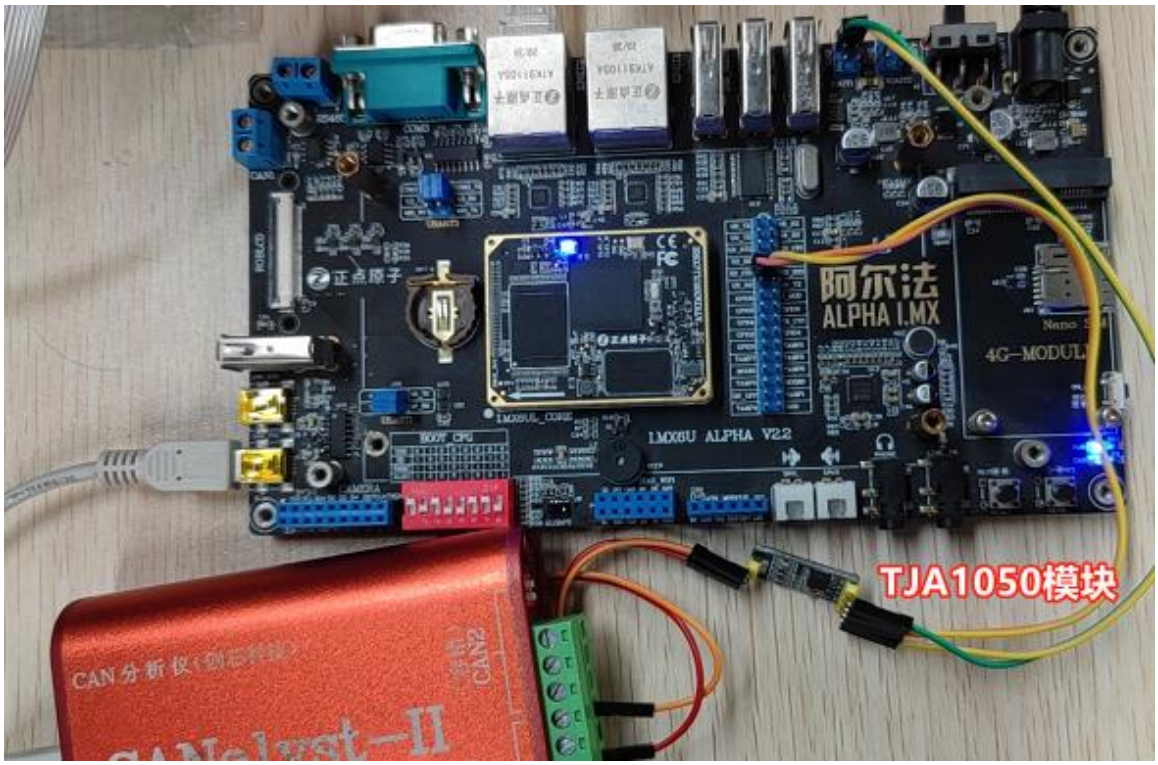
保存修改好的设备树文件，编译设备树。

```
    source   /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
    make imx_v7_defconfig
    make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

用编译好的设备树文件启动开发板。

10.2 CAN 测试

开发板上只有一路 CAN 芯片，需要在 U2_CTS 和 U2_RTS 外接一个 CAN 芯片模块做测试。
U2_CTS 接 CAN 模块的 TX，U2_RTS 接 CAN 模块的 RX。
CAN 模块的 CANH 接 CAN 分析仪的 CANH，CAN 模块的 CANL 接 CAN 分析仪的 CANL。
CAN 模块接 3.3V 供电，接地。如下图所示。



设置 can1 的 can 设备通信波特率为 1000000，也就是最大通信波特率 1MBit/s。

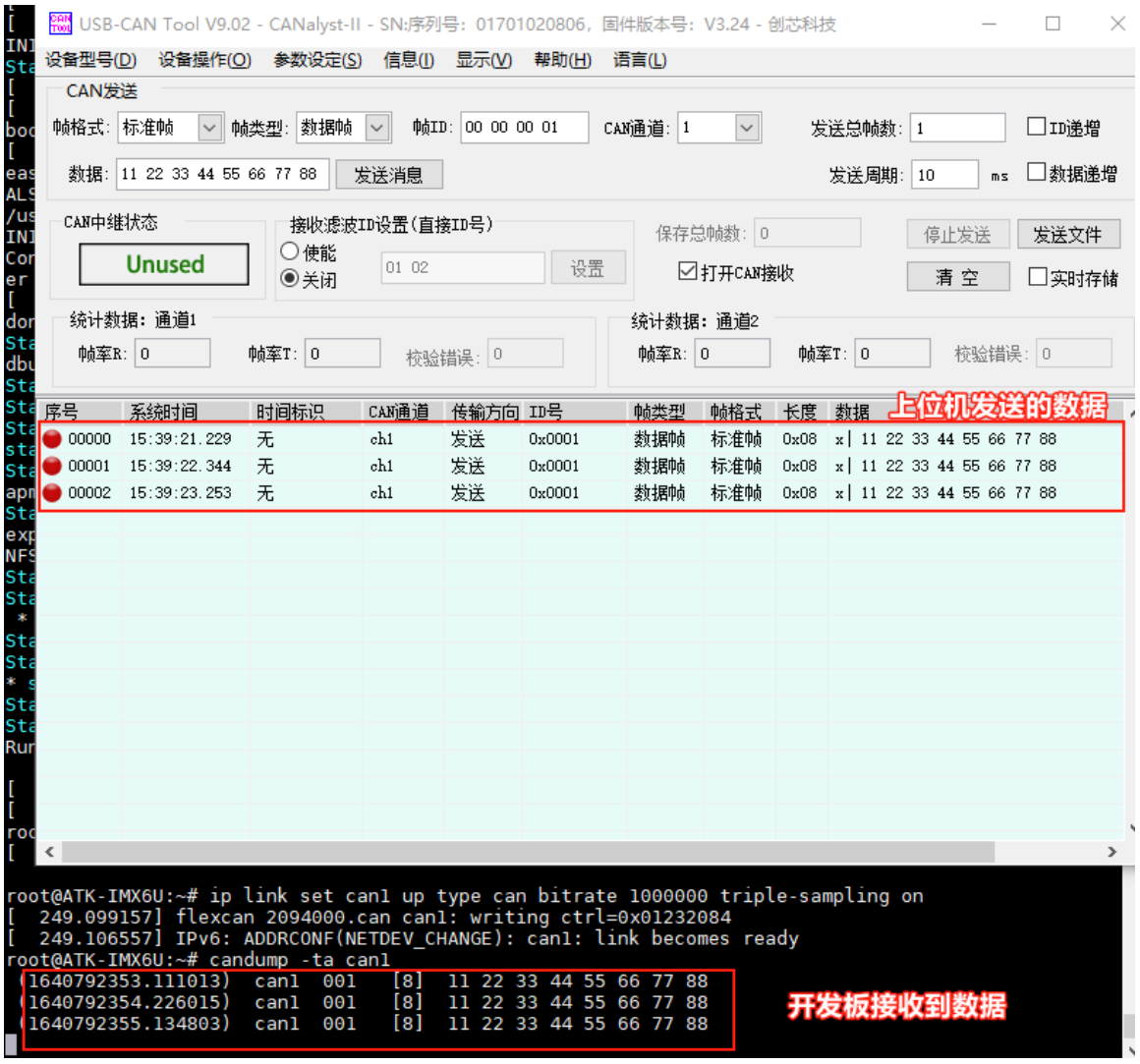
```
    ip link set can1 up type can bitrate 1000000 triple-sampling on
```

```
root@ATK-IMX6U:~# ip link set can1 up type can bitrate 1000000 triple-sampling on
[  249.099157] flexcan 2094000.can can1: writing ctrl=0x01232084
[  249.106557] IPv6: ADDRCONF(NETDEV_CHANGE): can1: link becomes ready
```

使用 candump 指令接收来自 can1 的数据

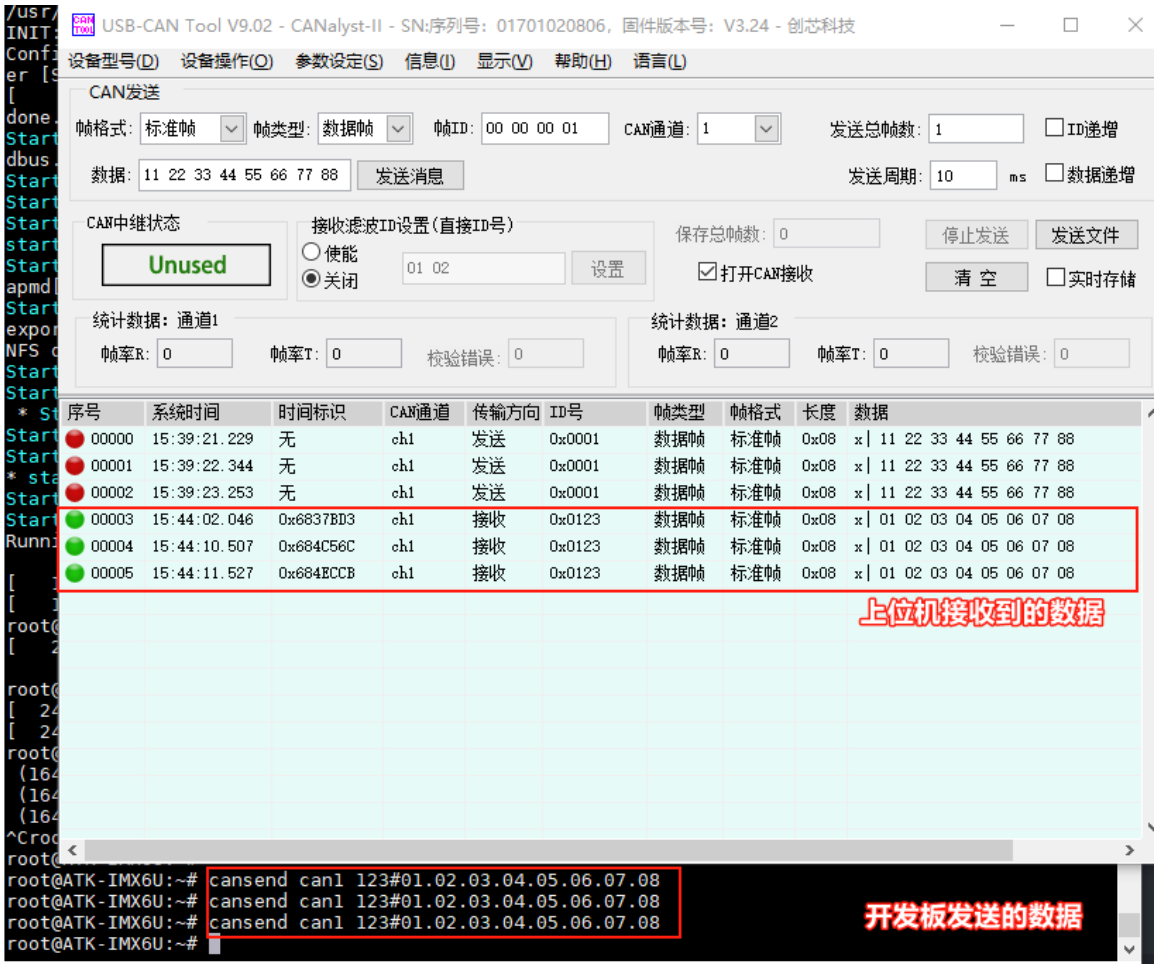
```
    candump -ta can1
```

-ta：t 代表打印时间，a 代表开启 ASCII 输出。
如下图，使用创芯科技的 CAN 分析仪上面机界面与开发板收发数据如下。



开发板使用 cansend 指令发送数据。

```
cansend can1 123#01.02.03.04.05.06.07.08
```



CAN1 的测试方法也是类似的，可以直接参考资料里的《【正点原子】I.MX6U 用户快速体验》对应的 CAN 测试章节测试，这里不再重复。至此，I.MX6ULL 开发板 2 路 CAN 配置和测试完成。

10.3 CAN 应用程序测试

测试程序可以参考《I.MX6U 嵌入式 Linux C 应用编程指南 V1.4》31 章 CAN 应用编程。

第十一章 多路 ADC 复用

根据芯片手册和数据手册说明，I.MX6ULL 核心板最多可以复用 10 路 ADC，分别为 GPIO1_00 到 GPIO1_09。芯片参考手册对 ADC 信号的说明：

13.2 External Signals

The following table describes the external signals of ADC:

Table 13-1. ADC External Signals

Signal	Description	Pad	Mode	Direction
ADC_VREFH	Voltage reference high	ADC_VREFH	-	I
ADC1_IN0	Analog channel 1 input 0	GPIO1_IO00	-	I
ADC1_IN1	Analog channel 1 input 1	GPIO1_IO01	-	I
ADC1_IN2	Analog channel 1 input 2	GPIO1_IO02	-	I
ADC1_IN3	Analog channel 1 input 3	GPIO1_IO03	-	I
ADC1_IN4	Analog channel 1 input 4	GPIO1_IO04	-	I
ADC1_IN5	Analog channel 1 input 5	GPIO1_IO05	-	I
ADC1_IN6	Analog channel 1 input 6	GPIO1_IO06	-	I
ADC1_IN7	Analog channel 1 input 7	GPIO1_IO07	-	I
ADC1_IN8	Analog channel 1 input 8	GPIO1_IO08	-	I
ADC1_IN9	Analog channel 1 input 9	GPIO1_IO09	-	I
ADC2_IN0	Analog channel 2 input 0	GPIO1_IO00	-	I
ADC2_IN1	Analog channel 2 input 1	GPIO1_IO01	-	I
ADC2_IN2	Analog channel 2 input 2	GPIO1_IO02	-	I
ADC2_IN3	Analog channel 2 input 3	GPIO1_IO03	-	I
ADC2_IN4	Analog channel 2 input 4	GPIO1_IO04	-	I
ADC2_IN5	Analog channel 2 input 5	GPIO1_IO05	-	I
ADC2_IN6	Analog channel 2 input 6	GPIO1_IO06	-	I
ADC2_IN7	Analog channel 2 input 7	GPIO1_IO07	-	I
ADC2_IN8	Analog channel 2 input 8	GPIO1_IO08	-	I
ADC2_IN9	Analog channel 2 input 9	GPIO1_IO09	-	I

在正点原子 I.MX6ULL 开发板上可以直接复用 GPIO1_IO01 做 ADC 功能，其他的管脚已经用于其他功能，需要对这部分进行裁剪后才能使用。裁剪后可用做 ADC 的管脚如下表：

核心板管脚	GPIO	ADC	备注
GPIO_0	GPIO1_IO00	ADC_IN0	开发板上做 USB_OTG1_ID，需软硬件裁剪
GPIO_1	GPIO1_IO01	ADC_IN1	开发板上可以直接添加 ADC 复用
GPIO_2	GPIO1_IO02	ADC_IN2	开发板上做 CSI_RESET，裁剪 CAMERA 来复用
GPIO_3	GPIO1_IO03	ADC_IN3	开发板上做 LED，需软硬件裁剪
GPIO_4	GPIO1_IO04	ADC_IN4	开发板上做 CSI_PWDN，裁剪 CAMERA 来复用
GPIO_5	GPIO1_IO05	ADC_IN5	开发板上做 SD1_VSELECT，不建议修改
GPIO_6	GPIO1_IO06	ADC_IN6	开发板上做 ENET_MDIO，不建议修改
GPIO_7	GPIO1_IO07	ADC_IN7	开发板上做 ENET_MDC，不建议修改
GPIO_8	GPIO1_IO08	ADC_IN8	开发板上做 BLT_PWM，通过裁剪 LCD 复用
GPIO_9	GPIO1_IO09	ADC_IN9	开发板上做 CT_INT，通过裁剪 LCD 复用

根据开发板软硬件裁剪的难度，多路 ADC 复用建议如下：

简单：

ADC_IN1：添加 ADC 复用。

麻烦：

ADC_IN0：修改烧录文件，以防烧写时 ID 脚不识别；修改硬件，去掉电阻；添加 ADC 复用。

ADC_IN2：裁剪 CAMREA，去掉电阻，添加 ADC 复用。

ADC_IN3：裁剪系统心跳灯 LED，去掉电阻，添加 ADC 复用。

ADC_IN4：裁剪 CAMREA，添加 ADC 复用。

ADC_IN8：裁剪 LCD，去掉电阻，添加 ADC 复用。

ADC_IN9：裁剪 LCD，去掉电阻，添加 ADC 复用。

困难：

ADC_IN5：设计底板，修改 SD 配置，释放 GPIO1_IO05 管脚。

ADC_IN6：设计底板，修改 ENET 配置，释放 GPIO1_IO06 管脚。

ADC_IN7：设计底板，修改 ENET 配置，释放 GPIO1_IO07 管脚。

本章节以 GPIO1_IO00、GPIO1_IO01 为例，复用 ADC_IN0、ADC_IN1 这 2 路 ADC，其他的 ADC 配置方法也是类似的。

11.1 GPIO_1

打开出厂内核源码 arch/arm/boot/dts/imx6ull-14x14-evk.dts 设备树文件。

在根节点里找到 regulators 节点，添加 reg_vref_3v3 节点信息，如下：

```
        reg_vref_3v3: regulator@2 {
                    compatible = "regulator-fixed";
                    regulator-name = "vref-3v3";
                    regulator-min-microvolt = <3300000>;
                    regulator-max-microvolt = <3300000>;
                };
    };
```

在根节点下添加&adc1

```
&adc1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_adc1>;
    vref-supply = <&reg_vref_3v3>;
    num-channels = <2>; //此参数为 ADC 的通道，扫描 2 个通道，0-1
    status = "okay";
};
```

在&iomuxc 节点下添加 pinctrl_adc1 节点信息，添加 GPIO1_IO01 的 ADC 定义，如下：

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

搜索 MX6UL_PAD_GPIO1_IO01，屏蔽掉除 adc 外的其他外设复用功能。

```
                pinctrl_ds18b20: 18b20 {
                        fsl,pins = <
/*                                      MX6UL_PAD_GPIO1_IO01__GPIO1_IO01        0x17059 */
                        >;
                };

                pinctrl_dht11: dht11 {
                        fsl,pins = <
/*                                      MX6UL_PAD_GPIO1_IO01__GPIO1_IO01        0x17059 */
                        >;
                };
```

```
        pinctrl_tsc: tscgrp {
            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
            >;
        };
```

disabled 掉这些外设复用的节点标签。

```
&tsc {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_tsc>;
    xnur-gpio = <&gpio1 3 GPIO_ACTIVE_LOW>;
    measure-delay-time = <0xffff>;
    pre-charge-time = <0xfff>;
    status = "disabled";
};
```

```
    dht11 {
        compatible = "alientek, dht11";
        pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_dht11>;
        dht11-gpio = <&gpio1 1 GPIO_ACTIVE_LOW>;
        status = "disabled";
    };

    ds18b20 {
        compatible = "alientek, ds18b20";
        pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_ds18b20>;
        ds18b20-gpio = <&gpio1 1 GPIO_ACTIVE_LOW>;
        status = "disabled";
    };
```

修改完成后保存文件，编译设备树。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

用生成的设备树文件启动开发板，进入开发板文件系统测试 ADC。

执行下面指令，获取 ADC 驱动设备值。

```
cat /sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0/in_voltage1_raw
```

```
root@ATK-IMX6U:~# cat /sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0/in_voltage1_raw
1044
root@ATK-IMX6U:~#
```

开发板接 3.3V 到 GPIO_1 管脚，再获取电压值。（注意不能接 5V，会烧坏板子）

```
root@ATK-IMX6U:~# cat /sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0/in_voltage1_raw
4040
```

11.2 GPIO_0

以阿尔法底板为例，GPIO_0 原理图如下：

可以看到 GPIO_0 这个管脚没有接电阻和其他外设，我们可以通过修改设备树来直接使用。

打开内核源码设备树文件 arch/arm/boot/dts/imx6ull-14x14-evk.dts

搜索 MX6UL_PAD_GPIO1_IO00，查看系统中已用的管脚配置并屏蔽。

```
458   &iomuxc {
459       pinctrl-names = "default";
460       pinctrl-0 = <&pinctrl_hog_1>;
461       imx6ul-evk {
462           pinctrl_hog_1: hoggrp-1 {
463               fsl,pins = <
464                   MX6UL_PAD_UART1_RTS_B__GPIO1_IO19   0x17059 /* SD1 CD */
465                   MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT    0x17059 /* SD1 VSELECT */
466   /*                MX6UL_PAD_GPIO1_IO00__ANATOP_OTG1_ID    0x13058    */
```

在根节点里找到 regulators 节点，添加 reg_vref_3v3 节点信息（已有则不用改），如下：

```
        reg_vref_3v3: regulator@2 {
                    compatible = "regulator-fixed";
                    regulator-name = "vref-3v3";
                    regulator-min-microvolt = <3300000>;
                    regulator-max-microvolt = <3300000>;
                };
    };
```

在根节点下添加&adc1（已有则不用改）

```
&adc1 {
    pinctrl-names = "default";
```

```
        pinctrl-0 = <&pinctrl_adc1>;
        vref-supply = <&reg_vref_3v3>;
        num-channels = <2>; //此参数为 ADC 的通道，扫描 2 个通道，0-1
        status = "okay";
    };
```

在&iomuxc 节点下添加 pinctrl_adc1 节点信息，添加 GPIO1_IO00 的 ADC 定义，如下：

```
        pinctrl_adc1: adc1grp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO00__GPIO1_IO00            0xb0
            >;
        };
```

可以基于上一小节添加，如下：

```
815         pinctrl_adc1: adc1grp {
816             fsl,pins = <
817                 MX6UL_PAD_GPIO1_IO01__GPIO1_IO01            0xb0
818                 MX6UL_PAD_GPIO1_IO00__GPIO1_IO00            0xb0
819             >;
820         };
821     };
```

修改完成后保存文件，编译设备树。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

用生成的设备树文件启动开发板，进入开发板文件系统测试 ADC。

执行下面指令，获取 GPIO1_IO00 的 ADC 驱动设备值。

```
cat /sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0/in_voltage0_raw
```

```
root@ATK-IMX6U:~# cat /sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0/in_voltage0_raw
90
```

开发板接 3.3V 到 GPIO_0 管脚，再获取电压值。（注意不能接 5V，会烧坏板子）

```
root@ATK-IMX6U:~# cat /sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0/in_voltage0_raw
4023
```

11.2.1 USB_OTG1_ID 脚问题

如果用户在设计底板时，将 USB_OTG1_ID 管脚（也就是 GPIO1_IO00）配置为其他外设功能或者没有引出，那么在使用正点原子出厂烧录工具烧写系统时，可能在串口显示的烧写信息里会看到卡在 cpu_id is 0 这里。因为出厂系统烧录工具默认配置了识别到 USB _OTB1_ID 拉高时进行烧写。

我们可以通过修改烧录工具里的设备树文件来解决这个问题，让烧录工具不识别 ID 脚，直接进行烧写。

打开出厂内核源码设备树文件 arch/arm/boot/dts/imx6ull-14x14-evk.dts

找到&usbotg1 节点信息，将 dr_mode = "otg";修改为 dr_mode = "peripheral";如下图所示：

```
1008  &usbotg1 {
1009  /*  dr_mode = "otg"; */
1010      dr_mode = "peripheral";
1011      srp-disable;
1012      hnp-disable;
1013      adp-disable;
1014      status = "okay";
1015  };
```

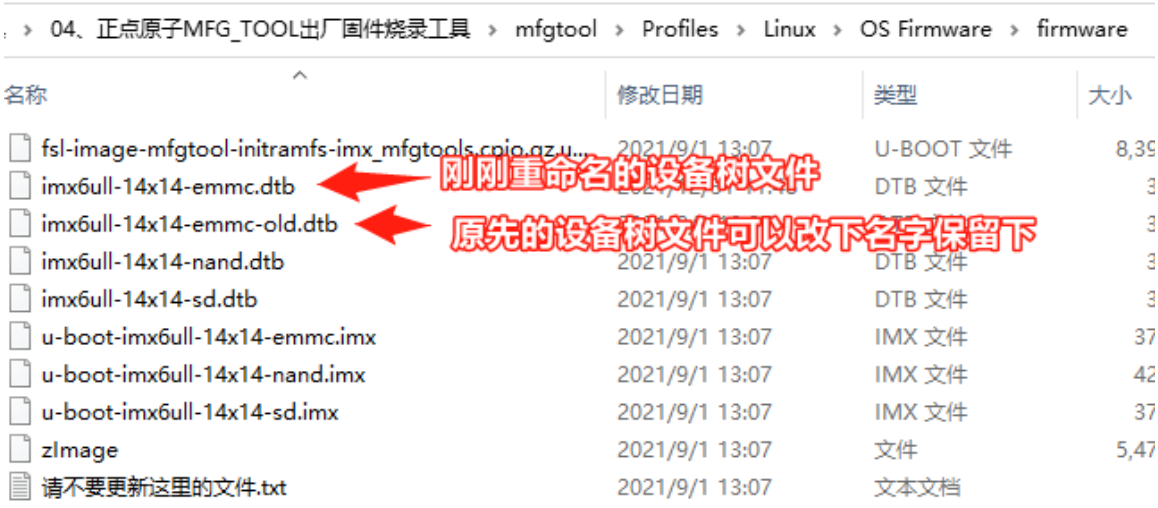
保存修改好的文件，编译设备树文件。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

将生成的设备树文件重命名为 imx6ull-14x14-emmc.dtb

```
alientek@ubuntu16:~/imx6ull/kernel$ cd arch/arm/boot/dts
alientek@ubuntu16:~/imx6ull/kernel/arch/arm/boot/dts$ mv imx6ull-14x14-emmc-4.3-480x272-c.dtb imx6ull-14x14-emmc.dtb
```

将 imx6ull-14x14-emmc.dtb 设备树替换到 04、正点原子 MFG_TOOL 出厂固件烧录工具\mfgtool\Profiles\Linux\OS Firmware\firmware 目录下，如下图所示：



然后就能正常启动烧录工具进行烧写，无需识别 ID 脚。

11.3 其他 ADC 复用

GPIO1_IO00~GPIO1_IO09 里的更多 ADC 复用，可以参考前面对应的裁剪章节和 ADC 复用章节来修改，这里不再复述。

第十二章 多路 PWM 复用

根据芯片手册和数据手册说明，I.MX6ULL 核心板最多可以复用 8 路 PWM，芯片参考手册对 PWM 信号的说明：

PWM1	OUT	ENET1_RX_DATA0	ALT2
		GPIO1_IO08	ALT0
		LCD_DATA00	ALT1
PWM2	OUT	ENET1_RX_DATA1	ALT2
		GPIO1_IO09	ALT0
		LCD_DATA01	ALT1
PWM3	OUT	GPIO1_IO04	ALT1
		LCD_DATA02	ALT1
		NAND_ALE	ALT3
PWM4	OUT	GPIO1_IO05	ALT1
		LCD_DATA03	ALT1
		NAND_WP_B	ALT3
PWM5	OUT	ENET1_TX_DATA1	ALT2
		LCD_DATA18	ALT1
		NAND_DQS	ALT3
PWM6	OUT	ENET1_TX_EN	ALT2
		JTAG_TDI	ALT4
		LCD_DATA19	ALT1
PWM7	OUT	CSI_VSYNC	ALT6
		ENET1_TX_CLK	ALT2
		JTAG_TCK	ALT4
PWM8	OUT	CSI_HSYNC	ALT6
		ENET1_RX_ER	ALT2
		JTAG_TRST_B	ALT4

可以看到，ENET1 系列相关的引脚大部分都可以复用做 PWM 功能，刚好我们 I.MX6ULL 开发板的 MINI 开发板硬件上裁剪了 ENTE1 网口，引出了部分 ENET1 的管脚。本章就以 MINI 开发板为例进行多路 PWM 配置。

MINI 开发板上管脚复用 PWM 表格：

核心板管脚	开发板管脚	MINI 板排针	PWM	备注
ENET1_RX_DATA0	ENET1_RXD0	PIN37	PWM1	需裁剪 ENET1
ENET1_RX_DATA1	ENET1_RXD1	PIN34	PWM2	需裁剪 ENET1
GPIO1_IO04	GPIO1_IO04	PIN10	PWM3	需裁剪 CAREMA
GPIO1_IO05	USDHC1_VSELECT		PWM4	做 SD，不建议修改
ENET1_TX_DATA1	ENET1_TXD1	PIN39	PWM5	需裁剪 ENET1
ENET1_TX_EN	ENET1_TXEN	PIN38	PWM6	需裁剪 ENET1
ENET1_TX_CLK	ENET1_TX_CLK	PIN40	PWM7	需裁剪 ENET1
ENET1_RX_ER	ENET1_RXER	PIN41	PWM8	需裁剪 ENET1

PWM3 不需要进行 ENET1 裁剪，比较方便，这里先配置 PWM3。

12.1 PWM3

12.1.1 修改设备树文件

打开出厂系统内核源码设备树文件 arch/arm/boot/dts/imx6ull-14x14-evk.dts
出厂系统默认配置了一路 pwm1，我们可以参考这个来添加其他 pwm 配置。
在根节点目录下添加&pwm3 节点信息，如下：

```
922  &pwm1 {
923      pinctrl-names = "default";
924      pinctrl-0 = <&pinctrl_pwm1>;
925      status = "okay";
926  };
927
928  &pwm3 {
929      pinctrl-names = "default";
930      pinctrl-0 = <&pinctrl_pwm3>;
931      status = "okay";
932  };
```

```
&pwm3 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_pwm3>;
    status = "okay";
};
```

在&iomuxc 节点下添加 pinctrl_pwm3 节点，添加 pwm3 管脚配置信息，如下：

```
619          pinctrl_pwm1: pwm1grp {
620              fsl,pins = <
621                  MX6UL_PAD_GPIO1_IO08__PWM1_OUT   0x110b0
622              >;
623          };
624
625          pinctrl_pwm3: pwm3grp {
626              fsl,pins = <
627                  MX6UL_PAD_GPIO1_IO04__PWM3_OUT   0x110b0
628              >;
629          };
```

```
        pinctrl_pwm3: pwm3grp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO04__PWM3_OUT   0x110b0
            >;
        };
```

管脚名 MX6UL_PAD_GPIO1_IO04__PWM3_OUT 是在 arch/arm/boot/dts/imx6ul-pinfunc.h 文件中找到的定义。

屏蔽 GPIO1_IO04 的其他复用。由于 GPIO1_IO04 已被出厂系统默认复用为摄像头 ov5640\2640\7725 功能引脚，故需屏蔽使用 ov5640\2640\7725 摄像头功能，将属性 status 由默认“okay”改为“disabled”，如下：

```
#if ATK_CAMERA_ON
    ov5640: ov5640@3c {
        compatible = "ovti,ov5640";
        reg = <0x3c>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_csi1
                     &csi_pwn_rst>;
        clocks = <&clks IMX6UL_CLK_CSI>;
        clock-names = "csi_mclk";
        pwn-gpios = <&gpio1 4 1>;
        rst-gpios = <&gpio1 2 0>;
        csi_id = <0>;
        mclk = <24000000>;
        mclk_source = <0>;
        status = "disabled";
        port {
            camera_ep: endpoint {
                remote-endpoint = <&csi_ep>;
            };
        };
    };
#endif
```

```
#if ATK_CAMERA_OFF
        ov2640: camera@0x30 {
                compatible = "ovti,ov2640";
                reg = <0x30>;
                status = "disabled";

                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_csi1
                             &csi_pwn_rst>;
                resetb = <&gpio1 2 GPIO_ACTIVE_LOW>;
                pwdn = <&gpio1 4 GPIO_ACTIVE_HIGH>;
                clocks = <&clks IMX6UL_CLK_CSI>;
                clock-names = "xvclk";

                port {
                        camera_ep: endpoint {
                                remote-endpoint = <&csi_ep>;
                bus-width = <8>;
                        };
                };
        };
#endif
```

```
#if ATK_CAMERA_OFF
        ov7725: camera@0x21 {
                compatible = "ovti,ov772x","ovti,ov7725";
                reg = <0x21>;
                status = "disabled";
                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_csi1
                             &csi_pwn_rst>;
                resetb = <&gpio1 2 GPIO_ACTIVE_LOW>;
                pwdn = <&gpio1 4 GPIO_ACTIVE_HIGH>;
                clocks = <&clks IMX6UL_CLK_CSI>;
                clock-frequency = <20000000>;
                clock-names = "mclk";

                port {
                        camera_ep: endpoint {
                                remote-endpoint = <&csi>;
                bus-width = <8>;
                        };
                };
        };
#endif
```

保存修改好的设备树文件，编译设备树。

```
source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

用生成的设备树文件启动开发板系统。

12.1.2 PWM 测试

进行开发板文件系统/sys/class/pwm。

```
cd /sys/class/pwm
```

```
root@ATK-IMX6U:~# cd /sys/class/pwm/
root@ATK-IMX6U:/sys/class/pwm# ls
pwmchip0  pwmchip1  pwmchip2  pwmchip3  pwmchip4  pwmchip5  pwmchip6  pwmchip7
```

pwmchip0~7 分别对应 pwm1~8，这里测试 pwm3，即 pwmchip2。

执行指令调出 pwmchip2 的 pwm0 目录：

```
echo 0 > /sys/class/pwm/pwmchip2/export
```

```
root@ATK-IMX6U:/sys/class/pwm# cd pwmchip2
root@ATK-IMX6U:/sys/class/pwm/pwmchip2# ls
device  export  npwm  power  subsystem  uevent  unexport
root@ATK-IMX6U:/sys/class/pwm/pwmchip2# echo 0 > /sys/class/pwm/pwmchip2/export
root@ATK-IMX6U:/sys/class/pwm/pwmchip2# ls
device  export  npwm  power  pwm0  subsystem  uevent  unexport
root@ATK-IMX6U:/sys/class/pwm/pwmchip2#
```

使能 PWM3。

```
echo 1 > /sys/class/pwm/pwmchip2/pwm0/enable
```

设置 PWM3 频率。

注意，这里设置的是周期值，单位为 ns，比如 20KHz 频率的周期就是 50000ns，输入如下命令：

```
echo 50000 > /sys/class/pwm/pwmchip2/pwm0/period
```

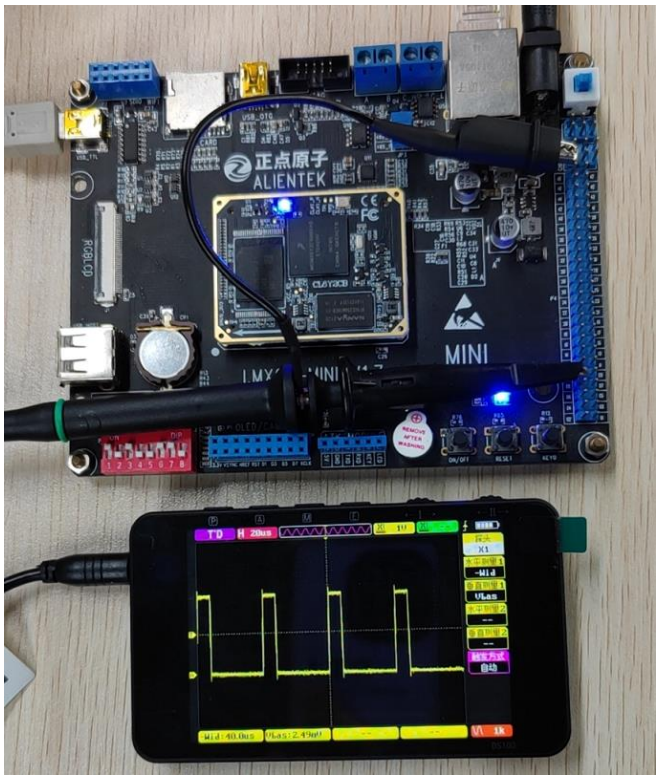
设置 PWM3 的占空比。

这里不能直接设置占空比，而是设置的一个周期的 ON 时间，也就是高电平时间，比如 20KHz 频率下 20%占空比的 ON 时间就是 10000，输入如下命令：

```
echo 10000 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle
```

```
root@ATK-IMX6U:~# cd /sys/class/pwm/
root@ATK-IMX6U:/sys/class/pwm# echo 0 > /sys/class/pwm/pwmchip2/export
root@ATK-IMX6U:/sys/class/pwm# echo 1 > /sys/class/pwm/pwmchip2/pwm0/enable
root@ATK-IMX6U:/sys/class/pwm# echo 50000 > /sys/class/pwm/pwmchip2/pwm0/period
root@ATK-IMX6U:/sys/class/pwm# echo 10000 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle
root@ATK-IMX6U:/sys/class/pwm# cat /sys/class/pwm/pwmchip2/pwm0/period 查看周期
50000
root@ATK-IMX6U:/sys/class/pwm# cat /sys/class/pwm/pwmchip2/pwm0/duty_cycle 查看占空比
10000
root@ATK-IMX6U:/sys/class/pwm#
```

使用正点原子手持示波器进行 PWM 波形捕获测试：


正点原子
ALIENTEK
MINI

12.2 裁剪 ENET1 做 PWM1\2\5\6\7\8

按照第四章裁剪 ENET1 的步骤裁剪完后就可以进行下面的 PWM 配置，这里假设用户已经完成对 ENET1 的裁剪和管脚测试。

12.2.1 修改内核和设备树

PWM1：出厂系统上用于配置 LCD 背光，这里不再重复配置，要用 PWM1 的话需要先处理 LCD 背光相关管脚配置。

PWM2：将 ENET1_RXD1 管脚配置成 PWM 功能。

在&iomuxc 节点下添加 pinctrl_pwm2、pinctrl_pwm5、pinctrl_pwm6、pinctrl_pwm7、pinctrl_pwm8 配置信息：

```
614         pinctrl_pwm1: pwm1grp {
615             fsl,pins = <
616                 MX6UL_PAD_GPIO1_IO08__PWM1_OUT   0x110b0
617             >;
618         };
619
620         pinctrl_pwm2: pwm2grp {
621             fsl,pins = <
622                 MX6UL_PAD_ENET1_RX_DATA1__PWM2_OUT   0x110b0
623             >;
624         };
625
626         pinctrl_pwm3: pwm3grp {
627             fsl,pins = <
628                 MX6UL_PAD_GPIO1_IO04__PWM3_OUT   0x110b0
629             >;
630         };
631
632         pinctrl_pwm5: pwm5grp {
633             fsl,pins = <
634                 MX6UL_PAD_ENET1_TX_DATA1__PWM5_OUT   0x110b0
635             >;
636         };
637
638         pinctrl_pwm6: pwm6grp {
639             fsl,pins = <
640                 MX6UL_PAD_ENET1_TX_EN__PWM6_OUT   0x110b0
641             >;
642         };
643
644         pinctrl_pwm7: pwm7grp {
645             fsl,pins = <
646                 MX6UL_PAD_ENET1_TX_CLK__PWM7_OUT   0x110b0
647             >;
648         };
649
650         pinctrl_pwm8: pwm8grp {
651             fsl,pins = <
652                 MX6UL_PAD_ENET1_RX_ER__PWM8_OUT   0x110b0
653             >;
654         };
```

在根节点下添加&pwm2、&pwm5、&pwm6、&pwm7、&pwm8 节点信息：

```
954  &pwm1 {
955      pinctrl-names = "default";
956      pinctrl-0 = <&pinctrl_pwm1>;
957      status = "okay";
958  };
959
960  &pwm2 {
961      pinctrl-names = "default";
962      pinctrl-0 = <&pinctrl_pwm2>;
963      status = "okay";
964  };
965
966  &pwm3 {
967      pinctrl-names = "default";
968      pinctrl-0 = <&pinctrl_pwm3>;
969      status = "okay";
970  };
971
972
973  &pwm5 {
974      pinctrl-names = "default";
975      pinctrl-0 = <&pinctrl_pwm5>;
976      status = "okay";
977  };
978
979  &pwm6 {
980      pinctrl-names = "default";
981      pinctrl-0 = <&pinctrl_pwm6>;
982      status = "okay";
983  };
984
985  &pwm7 {
986      pinctrl-names = "default";
987      pinctrl-0 = <&pinctrl_pwm7>;
988      status = "okay";
989  };
990
991  &pwm8 {
992      pinctrl-names = "default";
993      pinctrl-0 = <&pinctrl_pwm8>;
994      status = "okay";
995  };
```

修改 arch/arm/boot/dts/imx6ull.dtsi 文件的 PWM 时钟配置。将 pwm2\5\6\7\8 的的时钟 IMX6UL_CLK_DUMMY 改为对应的 IMX6UL_CLK_PWM2、IMX6UL_CLK_PWM5、IMX6UL_CLK_PWM6、IMX6UL_CLK_PWM7、IMX6UL_CLK_PWM8。

```
433             pwm1: pwm@02080000 {
434                 compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
435                 reg = <0x02080000 0x4000>;
436                 interrupts = <GIC_SPI 83 IRQ_TYPE_LEVEL_HIGH>;
437                 clocks = <&clks IMX6UL_CLK_PWM1>,
438                      <&clks IMX6UL_CLK_PWM1>;
439                 clock-names = "ipg", "per";
440                 #pwm-cells = <2>;
441             };
442
443             pwm2: pwm@02084000 {
444                 compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
445                 reg = <0x02084000 0x4000>;
446                 interrupts = <GIC_SPI 84 IRQ_TYPE_LEVEL_HIGH>;
447 /*              clocks = <&clks IMX6UL_CLK_DUMMY>,
448                      <&clks IMX6UL_CLK_DUMMY>; */
449                 clocks = <&clks IMX6UL_CLK_PWM2>,
450                      <&clks IMX6UL_CLK_PWM2>;
451                 clock-names = "ipg", "per";
452                 #pwm-cells = <2>;
453             };
```

```
rch > arm > boot > dts > imx6ull.dtsi
796             pwm5: pwm@020f0000 {
797                 compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
798                 reg = <0x020f0000 0x4000>;
799                 interrupts = <GIC_SPI 114 IRQ_TYPE_LEVEL_HIGH>;
800 /*              clocks = <&clks IMX6UL_CLK_DUMMY>,
801                      <&clks IMX6UL_CLK_DUMMY>; */
802                 clocks = <&clks IMX6UL_CLK_PWM5>,
803                      <&clks IMX6UL_CLK_PWM5>;
804                 clock-names = "ipg", "per";
805                 #pwm-cells = <2>;
806             };
807
808             pwm6: pwm@020f4000 {
809                 compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
810                 reg = <0x020f4000 0x4000>;
811                 interrupts = <GIC_SPI 115 IRQ_TYPE_LEVEL_HIGH>;
812 /*              clocks = <&clks IMX6UL_CLK_DUMMY>,
813                      <&clks IMX6UL_CLK_DUMMY>; */
814                 clocks = <&clks IMX6UL_CLK_PWM6>,
815                      <&clks IMX6UL_CLK_PWM6>;
816                 clock-names = "ipg", "per";
817                 #pwm-cells = <2>;
818             };
819
820             pwm7: pwm@020f8000 {
821                 compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
822                 reg = <0x020f8000 0x4000>;
823                 interrupts = <GIC_SPI 116 IRQ_TYPE_LEVEL_HIGH>;
824 /*              clocks = <&clks IMX6UL_CLK_DUMMY>,
825                      <&clks IMX6UL_CLK_DUMMY>; */
826                 clocks = <&clks IMX6UL_CLK_PWM7>,
827                      <&clks IMX6UL_CLK_PWM7>;
828                 clock-names = "ipg", "per";
829                 #pwm-cells = <2>;
830             };
831
832             pwm8: pwm@020fc000 {
833                 compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
834                 reg = <0x020fc000 0x4000>;
835                 interrupts = <GIC_SPI 117 IRQ_TYPE_LEVEL_HIGH>;
836 /*              clocks = <&clks IMX6UL_CLK_DUMMY>,
837                      <&clks IMX6UL_CLK_DUMMY>; */
838                 clocks = <&clks IMX6UL_CLK_PWM8>,
839                      <&clks IMX6UL_CLK_PWM8>;
840                 clock-names = "ipg", "per";
841                 #pwm-cells = <2>;
842             };
```

保存修改好的设备树各个文件，编译设备树和内核。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make imx_v7_defconfig
make zImage -j 16
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

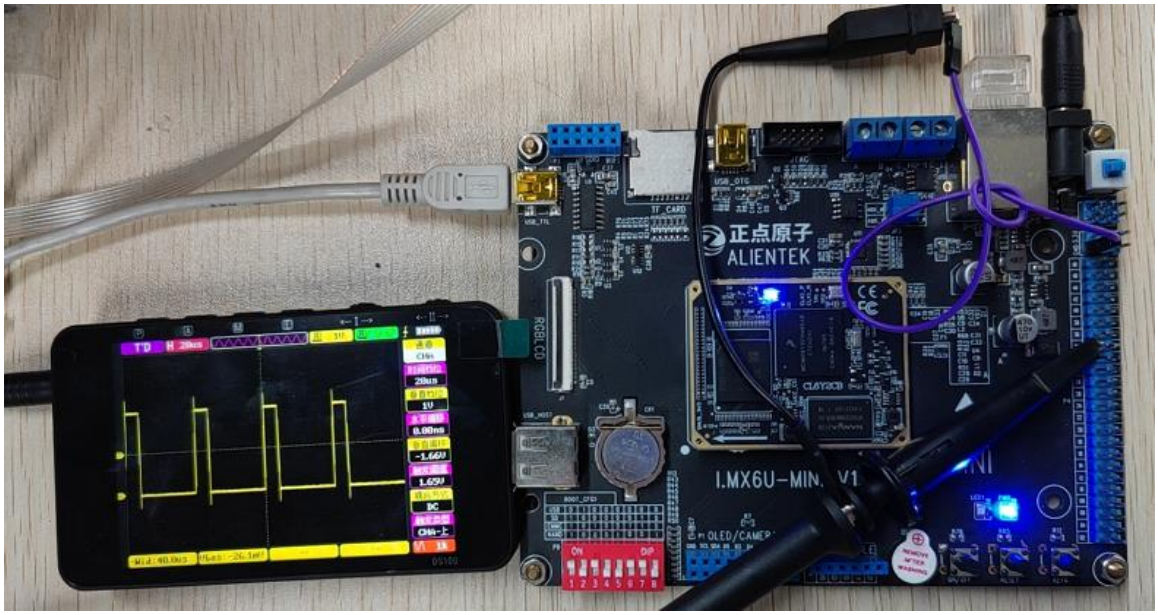
用编译好的设备树和内核启动开发板，进入系统。

12.2.2 PWM 测试

PWM2 测试，开发板系统配置指令：

```
echo 0 > /sys/class/pwm/pwmchip1/export
echo 1 > /sys/class/pwm/pwmchip1/pwm0/enable
echo 50000 > /sys/class/pwm/pwmchip1/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip1/pwm0/duty_cycle
```

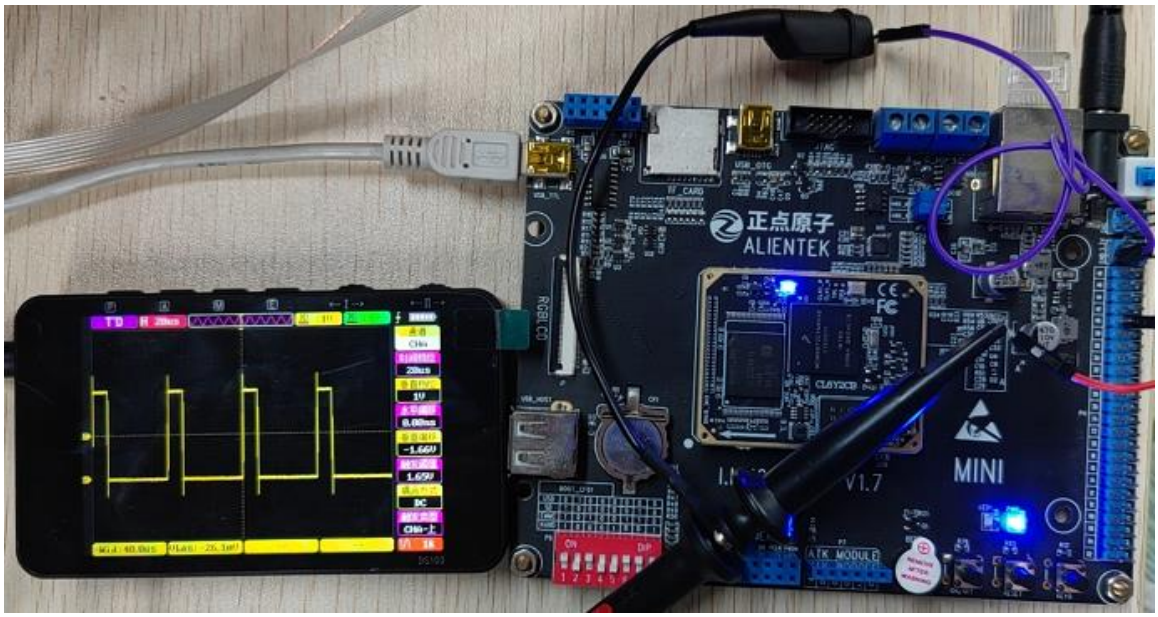
硬件连接：示波器接 MINI 开发板 ENET1_RXD1，即 PIN34 排针。



PWM5 测试，开发板系统配置指令：

```
echo 0 > /sys/class/pwm/pwmchip4/export
echo 1 > /sys/class/pwm/pwmchip4/pwm0/enable
echo 50000 > /sys/class/pwm/pwmchip4/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip4/pwm0/duty_cycle
```

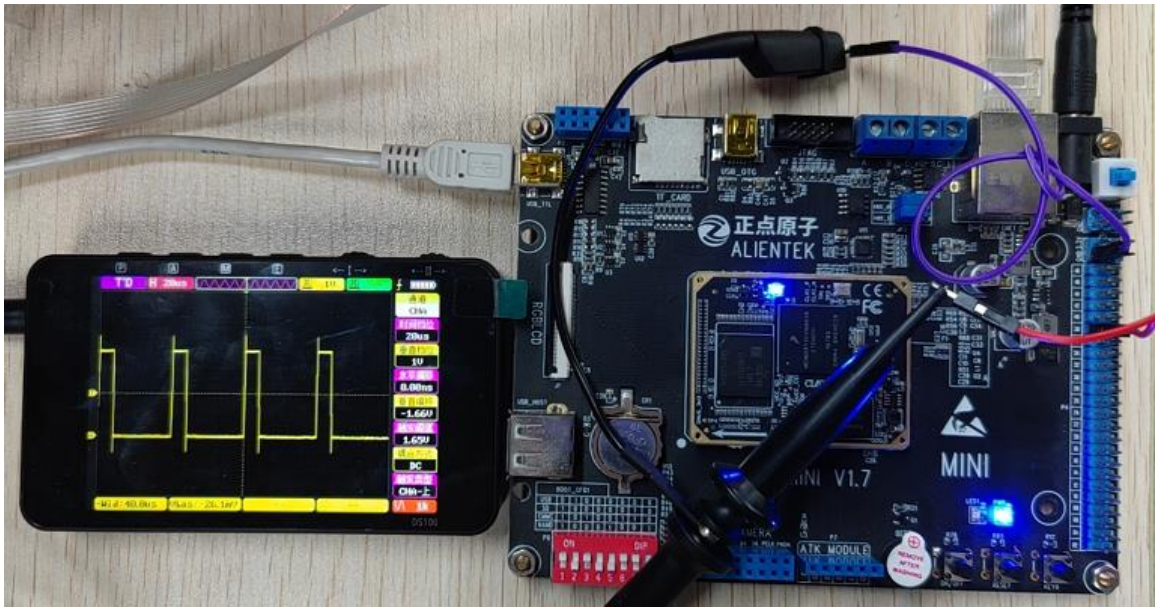
硬件连接：示波器接 MINI 开发板 ENET1_TXD1，即 PIN39 排针。



PWM6 测试，开发板系统配置指令：

```
echo 0 > /sys/class/pwm/pwmchip5/export
echo 1 > /sys/class/pwm/pwmchip5/pwm0/enable
echo 50000 > /sys/class/pwm/pwmchip5/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip5/pwm0/duty_cycle
```

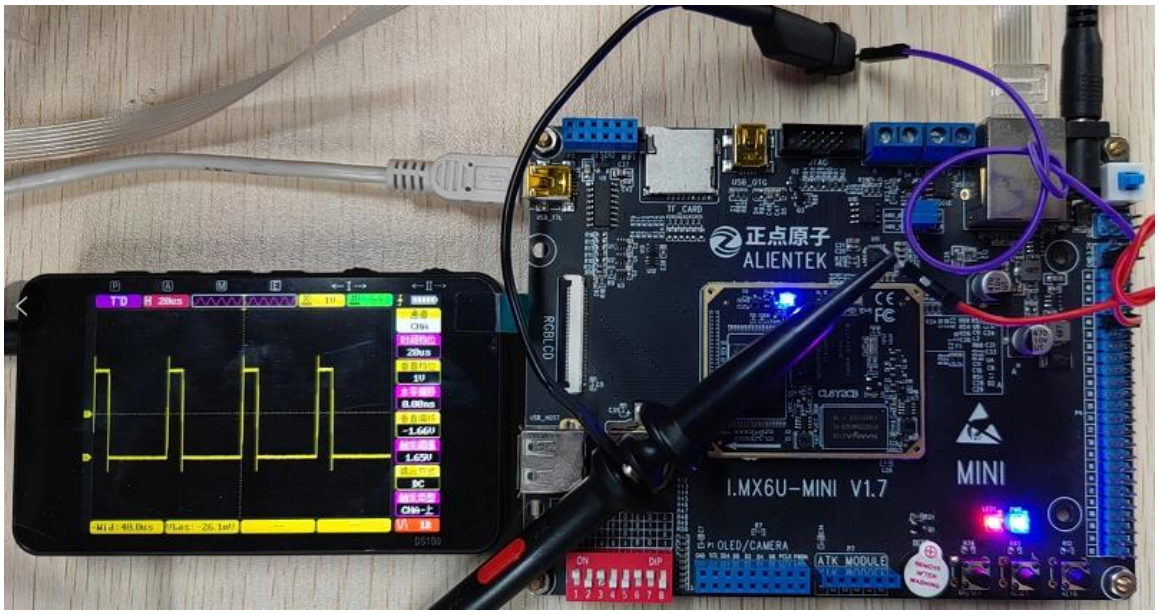
硬件连接：示波器接 MINI 开发板 ENET1_TXEN，即 PIN38 排针。



PWM7 测试，开发板系统配置指令：

```
echo 0 > /sys/class/pwm/pwmchip6/export
echo 1 > /sys/class/pwm/pwmchip6/pwm0/enable
echo 50000 > /sys/class/pwm/pwmchip6/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip6/pwm0/duty_cycle
```

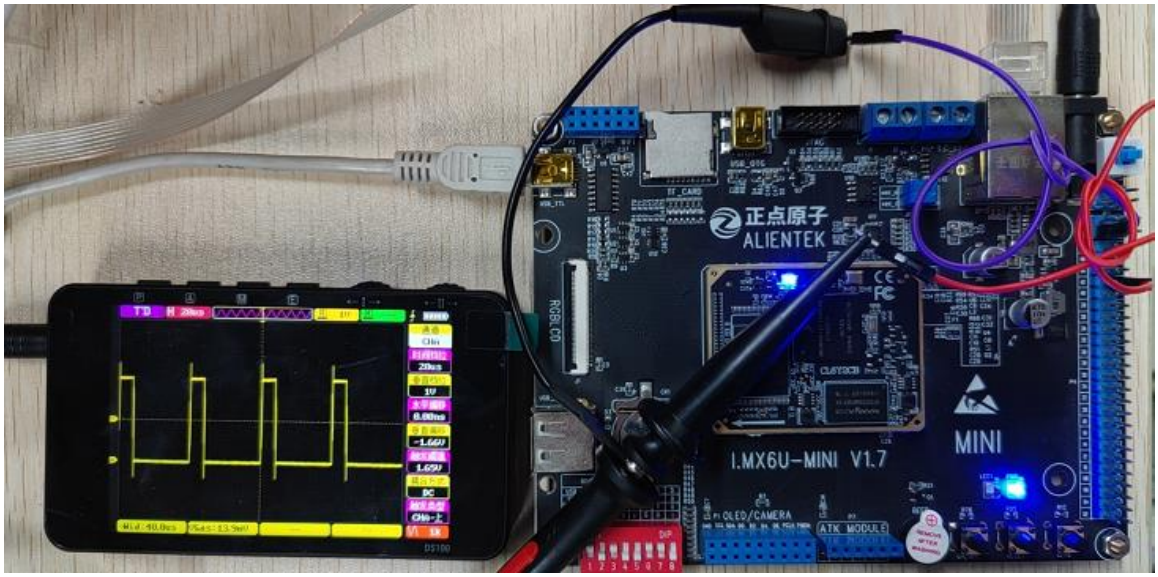
硬件连接：示波器接 MINI 开发板 ENET1_TX_CLK，即 PIN40 排针。



PWM8 测试，开发板系统配置指令：

```
echo 0 > /sys/class/pwm/pwmchip7/export
echo 1 > /sys/class/pwm/pwmchip7/pwm0/enable
echo 50000 > /sys/class/pwm/pwmchip7/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip7/pwm0/duty_cycle
```

硬件连接：示波器接 MINI 开发板 ENET1_RXER，即 PIN41 排针。



12.3 裁剪 AUDIO\JTAG 做 PWM6\7\8

12.3.1 修改内核和设备树

这里使用 I.MX6ULL MINI 板，假设已经完成第六章的 AUDIO\JTAG 裁剪和 GPIO 测试。
在裁剪完 AUDIO\JTAG 的设备树里，在&iomuxc 添加 pwm6\7\8 配置信息：

```
        pinctrl_pwm6: pwm6grp {
            fsl,pins = <
                MX6UL_PAD_JTAG_TDI__PWM6_OUT    0x110b0
            >;
        };

        pinctrl_pwm7: pwm7grp {
            fsl,pins = <
                MX6UL_PAD_JTAG_TCK__PWM7_OUT    0x110b0
            >;
        };

        pinctrl_pwm8: pwm8grp {
            fsl,pins = <
                MX6UL_PAD_JTAG_TRST_B__PWM8_OUT    0x110b0
            >;
        };
```

在根节点下添加&pwm6\7\8 节点追加信息：

```
&pwm6 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_pwm6>;
    status = "okay";
};

&pwm7 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_pwm7>;
    status = "okay";
};

&pwm8 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_pwm8>;
    status = "okay";
};
```

修改 arch/arm/boot/dts/imx6ull.dtsi 文件的 PWM 时钟配置。

```
            pwm6: pwm@020f4000 {
                compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
                reg = <0x020f4000 0x4000>;
                interrupts = <GIC_SPI 115 IRQ_TYPE_LEVEL_HIGH>;
/*              clocks = <&clks IMX6UL_CLK_DUMMY>,
                     <&clks IMX6UL_CLK_DUMMY>; */
                clocks = <&clks IMX6UL_CLK_PWM6>,
                     <&clks IMX6UL_CLK_PWM6>;
                clock-names = "ipg", "per";
                #pwm-cells = <2>;
            };

            pwm7: pwm@020f8000 {
                compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
                reg = <0x020f8000 0x4000>;
                interrupts = <GIC_SPI 116 IRQ_TYPE_LEVEL_HIGH>;
/*              clocks = <&clks IMX6UL_CLK_DUMMY>,
                     <&clks IMX6UL_CLK_DUMMY>; */
                clocks = <&clks IMX6UL_CLK_PWM7>,
                     <&clks IMX6UL_CLK_PWM7>;
                clock-names = "ipg", "per";
                #pwm-cells = <2>;
            };

            pwm8: pwm@020fc000 {
                compatible = "fsl,imx6ul-pwm", "fsl,imx27-pwm";
                reg = <0x020fc000 0x4000>;
                interrupts = <GIC_SPI 117 IRQ_TYPE_LEVEL_HIGH>;
/*              clocks = <&clks IMX6UL_CLK_DUMMY>,
                     <&clks IMX6UL_CLK_DUMMY>; */
                clocks = <&clks IMX6UL_CLK_PWM8>,
                     <&clks IMX6UL_CLK_PWM8>;
                clock-names = "ipg", "per";
                #pwm-cells = <2>;
            };
```

保存修改好的设备树各个文件，编译设备树和内核。

```
source  /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
make del_audio_imx_v7_defconfig
make zImage -j 16
make imx6ull-14x14-emmc-4.3-480x272-c.dtb
```

用编译好的设备树和内核启动开发板，进入系统。

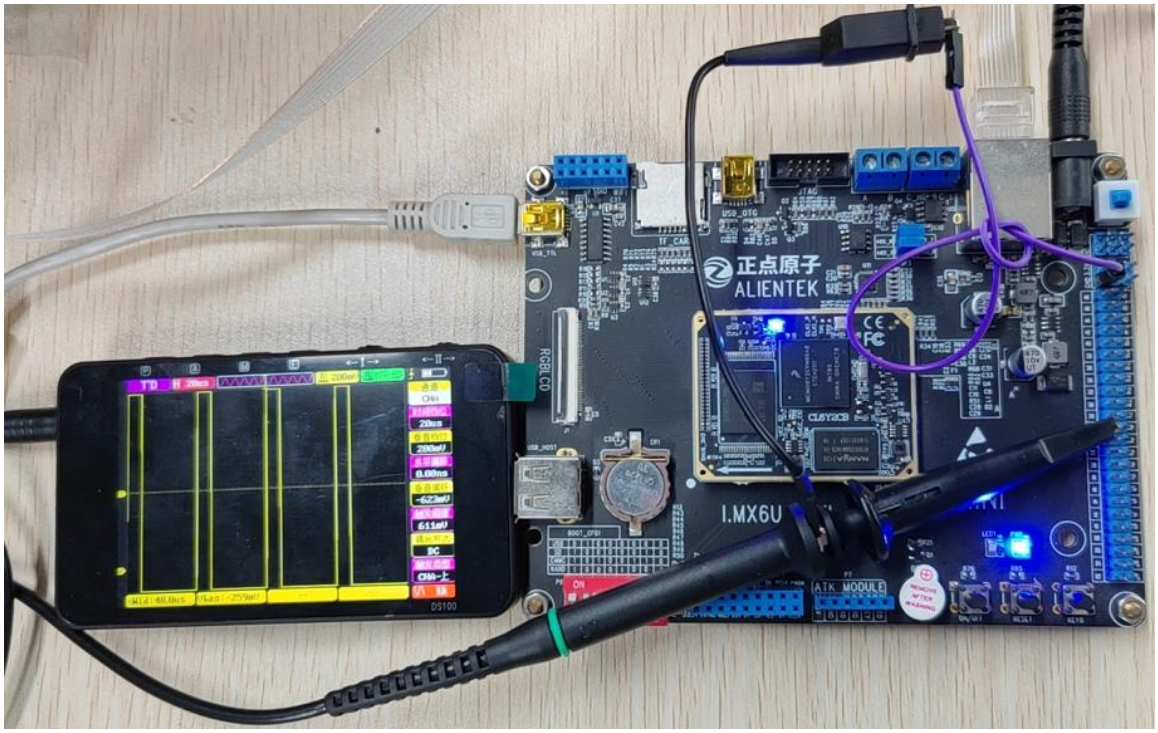
12.3.2 PWM 测试

PWM6 测试，MINI 板对应的管脚是 JTAG_TDI，即引出排针的 PIN26。

进入开发板系统。

```
echo 0 > /sys/class/pwm/pwmchip5/export
echo 50000 > /sys/class/pwm/pwmchip5/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip5/pwm0/duty_cycle
echo 1 > /sys/class/pwm/pwmchip5/pwm0/enable
```

示波器查看 PWM6 波形。

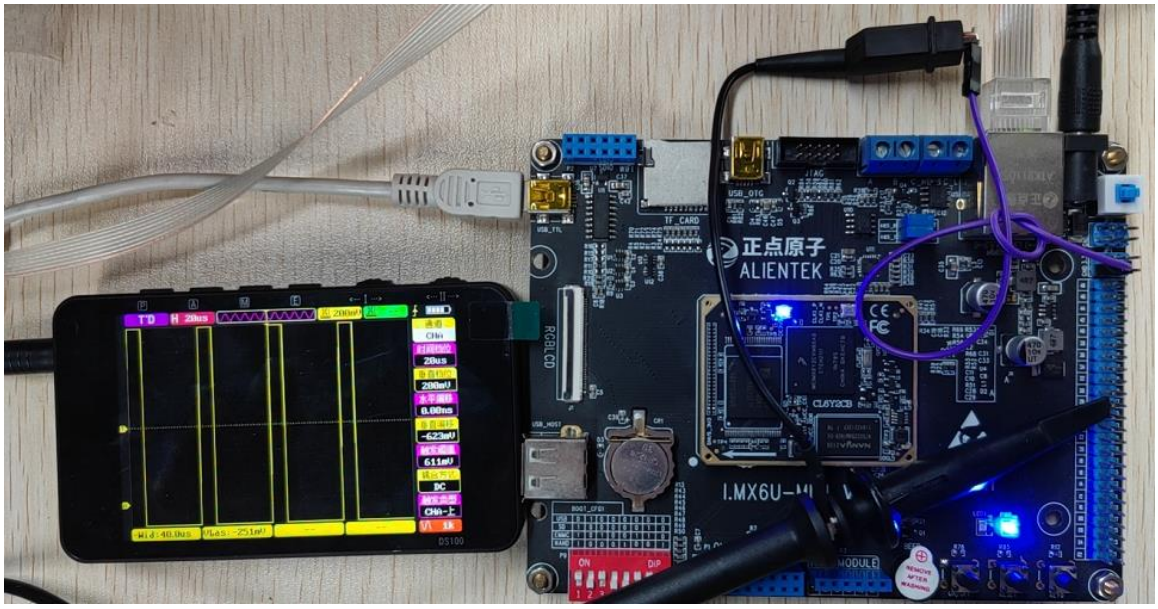


PWM7 测试，MINI 板对应的管脚是 JTAG_TCK，即引出排针的 PIN28。

进入开发板系统。

```
echo 0 > /sys/class/pwm/pwmchip6/export
echo 50000 > /sys/class/pwm/pwmchip6/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip6/pwm0/duty_cycle
echo 1 > /sys/class/pwm/pwmchip6/pwm0/enable
```

示波器查看 PWM7 波形。

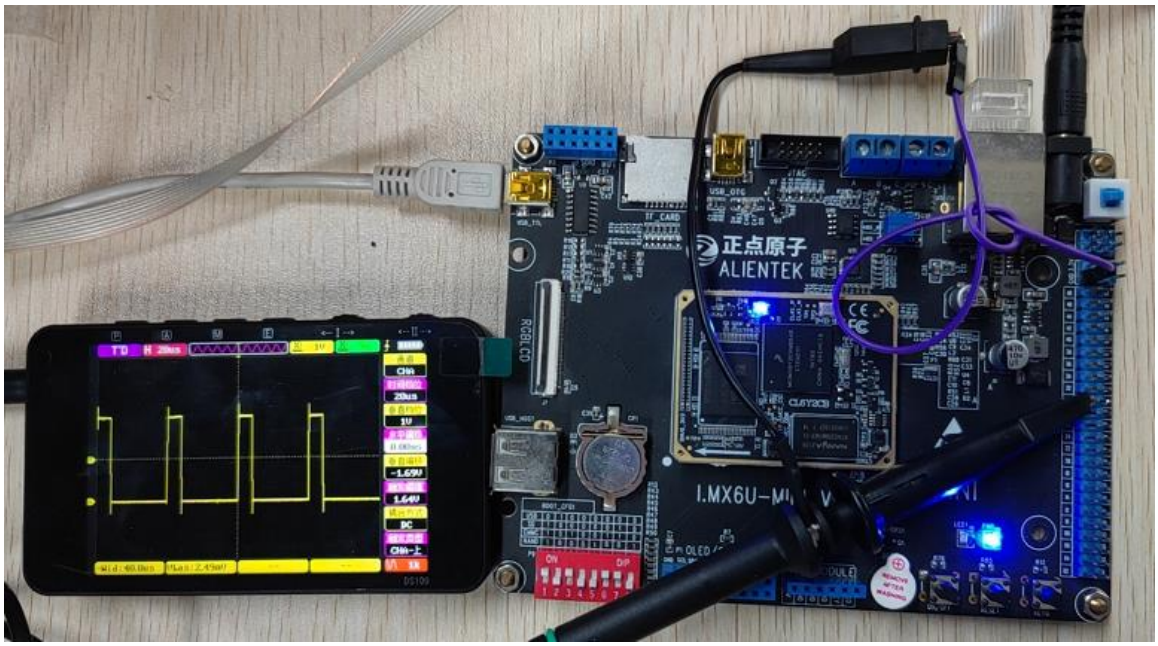


PWM8 测试，MINI 板对应的管脚是 JTAG_nTRST，即引出排针的 PIN30。

进入开发板系统。

```
echo 0 > /sys/class/pwm/pwmchip7/export
echo 50000 > /sys/class/pwm/pwmchip7/pwm0/period
echo 10000 > /sys/class/pwm/pwmchip7/pwm0/duty_cycle
echo 1 > /sys/class/pwm/pwmchip7/pwm0/enable
```

示波器查看 PWM8 波形。



12.4 PWM 应用程序测试

测试程序可以参考《I.MX6U 嵌入式 Linux C 应用编程指南 V1.4》24 章 PWM 应用编程。

附录

常用外设复用可选管脚

摘自《IMX6ULL 参考手册》第四章。

Instance	Port	Pad	Mode
CSI	DATA0	LCD_DATA17	ALT3
		UART3_RX_DATA	ALT3
	DATA1	LCD_DATA16	ALT3
		UART3_TX_DATA	ALT3
	DATA2	CSI_DATA00	ALT0
		UART1_TX_DATA	ALT3
	DATA3	CSI_DATA01	ALT0
		UART1_RX_DATA	ALT3
	DATA4	CSI_DATA02	ALT0
		UART1_CTS_B	ALT3
	DATA5	CSI_DATA03	ALT0
		UART1_RTS_B	ALT3
	DATA6	CSI_DATA04	ALT0
		UART2_TX_DATA	ALT3
	DATA7	CSI_DATA05	ALT0
		UART2_RX_DATA	ALT3
	DATA8	CSI_DATA06	ALT0
		UART2_CTS_B	ALT3
	DATA9	CSI_DATA07	ALT0
		UART2_RTS_B	ALT3
	DATA10	LCD_DATA18	ALT3
		UART3_CTS_B	ALT3
	DATA11	LCD_DATA19	ALT3
		UART3_RTS_B	ALT3
	DATA12	LCD_DATA20	ALT3
		UART4_TX_DATA	ALT3
	DATA13	LCD_DATA21	ALT3
		UART4_RX_DATA	ALT3

	DATA17	ENET1_RX_DATA1	ALT3
		LCD_DATA09	ALT3
	DATA18	ENET1_RX_EN	ALT3
		LCD_DATA10	ALT3
	DATA19	ENET1_TX_DATA0	ALT3
		LCD_DATA11	ALT3
	DATA20	ENET1_TX_DATA1	ALT3
		LCD_DATA12	ALT3
	DATA21	ENET1_TX_EN	ALT3
		LCD_DATA13	ALT3
	DATA22	ENET1_TX_CLK	ALT3
		LCD_DATA14	ALT3
	DATA23	ENET1_RX_ER	ALT3
		LCD_DATA15	ALT3
	FIELD	GPIO1_IO05	ALT3
		NAND_DQS	ALT1
	HSYNC	CSI_HSYNC	ALT0
		GPIO1_IO09	ALT3
	MCLK	CSI_MCLK	ALT0
		GPIO1_IO06	ALT3
	PIXCLK	CSI_PIXCLK	ALT0
		GPIO1_IO07	ALT3
	VSYNC	CSI_VSYNC	ALT0
		GPIO1_IO08	ALT3

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
ECSPI2	MISO	CSI_DATA03	ALT3
		UART5_RX_DATA	ALT8
	MOSI	CSI_DATA02	ALT3
		UART5_TX_DATA	ALT8
	RDY	LCD_ENABLE	ALT8
	SCLK	CSI_DATA00	ALT3
		UART4_TX_DATA	ALT8
	SS0	CSI_DATA01	ALT3
		UART4_RX_DATA	ALT8
	SS1	LCD_HSYNC	ALT8
	SS2	LCD_VSYNC	ALT8
	SS3	LCD_RESET	ALT8

ECSPI3	MISO	NAND_CLE	ALT3
		UART2_RTS_B	ALT8
	MOSI	NAND_CE1_B	ALT3
		UART2_CTS_B	ALT8
	RDY	NAND_WP_B	ALT8
	SCLK	NAND_CE0_B	ALT3
		UART2_RX_DATA	ALT8
	SS0	NAND_READY_B	ALT3
		UART2_TX_DATA	ALT8
	SS1	NAND_ALE	ALT8
	SS2	NAND_RE_B	ALT8
	SS3	NAND_WE_B	ALT8
ECSPI4	MISO	ENET2_TX_CLK	ALT3
		NAND_DATA06	ALT3
	MOSI	ENET2_TX_EN	ALT3
		NAND_DATA05	ALT3
	RDY	NAND_DATA00	ALT8
	SCLK	ENET2_TX_DATA1	ALT3
		NAND_DATA04	ALT3
	SS0	ENET2_RX_ER	ALT3
		NAND_DATA07	ALT3
	SS1	NAND_DATA01	ALT8
	SS2	NAND_DATA02	ALT8
	SS3	NAND_DATA03	ALT8

PWM1	OUT	ENET1_RX_DATA0	ALT2
		GPIO1_IO08	ALT0
		LCD_DATA00	ALT1
PWM2	OUT	ENET1_RX_DATA1	ALT2
		GPIO1_IO09	ALT0
		LCD_DATA01	ALT1
PWM3	OUT	GPIO1_IO04	ALT1
		LCD_DATA02	ALT1
		NAND_ALE	ALT3
PWM4	OUT	GPIO1_IO05	ALT1
		LCD_DATA03	ALT1
		NAND_WP_B	ALT3
PWM5	OUT	ENET1_TX_DATA1	ALT2
		LCD_DATA18	ALT1
		NAND_DQS	ALT3
PWM6	OUT	ENET1_TX_EN	ALT2
		JTAG_TDI	ALT4
		LCD_DATA19	ALT1
PWM7	OUT	CSI_VSYNC	ALT6
		ENET1_TX_CLK	ALT2
		JTAG_TCK	ALT4
PWM8	OUT	CSI_HSYNC	ALT6
		ENET1_RX_ER	ALT2
		JTAG_TRST_B	ALT4

UART1	CTS_B	GPIO1_IO06	ALT8
		UART1_CTS_B	ALT0
	RTS_B	GPIO1_IO07	ALT8
		UART1_RTS_B	ALT0
	RX_DATA	GPIO1_IO03	ALT8
		UART1_RX_DATA	ALT0
	TX_DATA	GPIO1_IO02	ALT8
		UART1_TX_DATA	ALT0
UART2	CTS_B	NAND_DATA06	ALT8
		UART2_CTS_B	ALT0
		UART3_TX_DATA	ALT4
	RTS_B	NAND_DATA07	ALT8
		UART2_RTS_B	ALT0
		UART3_RX_DATA	ALT4
	RX_DATA	NAND_DATA05	ALT8
		UART2_RX_DATA	ALT0
	TX_DATA	NAND_DATA04	ALT8
		UART2_TX_DATA	ALT0
UART3	CTS_B	NAND_CE1_B	ALT8
		UART3_CTS_B	ALT0
	RTS_B	NAND_CLE	ALT8
		UART3_RTS_B	ALT0
	RX_DATA	NAND_CE0_B	ALT8
		UART3_RX_DATA	ALT0
	TX_DATA	NAND_READY_B	ALT8
		UART3_TX_DATA	ALT0
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

UART5	CTS_B	CSI_DATA03	ALT8
		ENET1_TX_DATA0	ALT1
		GPIO1_IO09	ALT8
	RTS_B	CSI_DATA02	ALT8
		ENET1_RX_EN	ALT1
		GPIO1_IO08	ALT8
	RX_DATA	CSI_DATA01	ALT8
		GPIO1_IO05	ALT8
		UART5_RX_DATA	ALT0
	TX_DATA	CSI_DATA00	ALT8
		GPIO1_IO04	ALT8
		UART5_TX_DATA	ALT0
UART6	CTS_B	CSI_HSYNC	ALT8
		ENET1_TX_DATA1	ALT1
	RTS_B	CSI_VSYNC	ALT8
		ENET1_TX_EN	ALT1
	RX_DATA	CSI_PIXCLK	ALT8
		ENET2_RX_DATA1	ALT1
	TX_DATA	CSI_MCLK	ALT8
		ENET2_RX_DATA0	ALT1
UART7	CTS_B	ENET1_TX_CLK	ALT1
		LCD_DATA06	ALT1
	RTS_B	ENET1_RX_ER	ALT1
		LCD_DATA07	ALT1
	RX_DATA	ENET2_TX_DATA0	ALT1
		LCD_DATA17	ALT1
	TX_DATA	ENET2_RX_EN	ALT1
		LCD_DATA16	ALT1
UART8	CTS_B	ENET2_TX_CLK	ALT1
		LCD_DATA04	ALT1
	RTS_B	ENET2_RX_ER	ALT1
		LCD_DATA05	ALT1
	RX_DATA	ENET2_TX_EN	ALT1
		LCD_DATA21	ALT1
	TX_DATA	ENET2_TX_DATA1	ALT1
		LCD_DATA20	ALT1

ENET1	1588_EVENT0_IN	GPIO1_IO00	ALT6
	1588_EVENT0_OUT	GPIO1_IO01	ALT6
	1588_EVENT1_IN	UART3_CTS_B	ALT4
	1588_EVENT1_OUT	UART3_RTS_B	ALT4
	1588_EVENT2_IN	LCD_DATA00	ALT3
	1588_EVENT2_OUT	LCD_DATA01	ALT3
	1588_EVENT3_IN	LCD_DATA02	ALT3
	1588_EVENT3_OUT	LCD_DATA03	ALT3
	COL	UART2_RTS_B	ALT1
	CRS	UART2_CTS_B	ALT1
	MDC	ENET2_RX_DATA1	ALT4
		GPIO1_IO07	ALT0
	MDIO	ENET2_RX_DATA0	ALT4
		GPIO1_IO06	ALT0
	RDATA0	ENET1_RX_DATA0	ALT0
	RDATA1	ENET1_RX_DATA1	ALT0
	RDATA2	UART1_TX_DATA	ALT1
	RDATA3	UART1_RX_DATA	ALT1
	RX_CLK	UART1_CTS_B	ALT1
	RX_EN	ENET1_RX_EN	ALT0
	RX_ER	ENET1_RX_ER	ALT0
	TDATA0	ENET1_TX_DATA0	ALT0
	TDATA1	ENET1_TX_DATA1	ALT0
	TDATA2	UART2_TX_DATA	ALT1
	TDATA3	UART2_RX_DATA	ALT1
	TX_CLK	ENET1_TX_CLK	ALT0
	TX_EN	ENET1_TX_EN	ALT0
	TX_ER	UART1_RTS_B	ALT1

ENET2	1588_EVENT0_IN	GPIO1_IO04	ALT6
	1588_EVENT0_OUT	GPIO1_IO05	ALT6
	1588_EVENT1_IN	UART1_CTS_B	ALT4
	1588_EVENT1_OUT	UART1_RTS_B	ALT4
	1588_EVENT2_IN	LCD_DATA04	ALT3
	1588_EVENT2_OUT	LCD_DATA05	ALT3
	1588_EVENT3_IN	LCD_DATA06	ALT3
	1588_EVENT3_OUT	LCD_DATA07	ALT3
	COL	UART5_RX_DATA	ALT1
	CRS	UART5_TX_DATA	ALT1
	MDC	ENET1_TX_EN	ALT4
		GPIO1_IO07	ALT1
	MDIO	ENET1_TX_DATA1	ALT4
		GPIO1_IO06	ALT1
	RDATA0	ENET2_RX_DATA0	ALT0
	RDATA1	ENET2_RX_DATA1	ALT0
	RDATA2	UART3_TX_DATA	ALT1
	RDATA3	UART3_RX_DATA	ALT1
	RX_CLK	UART3_CTS_B	ALT1
	RX_EN	ENET2_RX_EN	ALT0
	RX_ER	ENET2_RX_ER	ALT0
	TDATA0	ENET2_TX_DATA0	ALT0
	TDATA1	ENET2_TX_DATA1	ALT0
	TDATA2	UART4_TX_DATA	ALT1
	TDATA3	UART4_RX_DATA	ALT1
	TX_CLK	ENET2_TX_CLK	ALT0
	TX_EN	ENET2_TX_EN	ALT0
	TX_ER	UART3_RTS_B	ALT1

FLEXCAN1	RX	ENET1_RX_DATA1	ALT4
		LCD_DATA09	ALT8
		SD1_DATA1	ALT3
		UART3_RTS_B	ALT2
	TX	ENET1_RX_DATA0	ALT4
		LCD_DATA08	ALT8
		SD1_DATA0	ALT3
		UART3_CTS_B	ALT2
FLEXCAN2	RX	ENET1_TX_DATA0	ALT4
		LCD_DATA11	ALT8
		SD1_DATA3	ALT3
		UART2_RTS_B	ALT2
	TX	ENET1_RX_EN	ALT4
		LCD_DATA10	ALT8
		SD1_DATA2	ALT3
		UART2_CTS_B	ALT2

GPIO1	IO0	GPIO1_IO00	ALT5
	IO1	GPIO1_IO01	ALT5
	IO2	GPIO1_IO02	ALT5
	IO3	GPIO1_IO03	ALT5
	IO4	GPIO1_IO04	ALT5
	IO5	GPIO1_IO05	ALT5
	IO6	GPIO1_IO06	ALT5
	IO7	GPIO1_IO07	ALT5
	IO8	GPIO1_IO08	ALT5
	IO9	GPIO1_IO09	ALT5
	IO10	JTAG_MOD	ALT5
	IO11	JTAG_TMS	ALT5
	IO12	JTAG_TDO	ALT5
	IO13	JTAG_TDI	ALT5
	IO14	JTAG_TCK	ALT5
	IO15	JTAG_TRST_B	ALT5
	IO16	UART1_TX_DATA	ALT5
	IO17	UART1_RX_DATA	ALT5
	IO18	UART1_CTS_B	ALT5
	IO19	UART1_RTS_B	ALT5
	IO20	UART2_TX_DATA	ALT5
	IO21	UART2_RX_DATA	ALT5
	IO22	UART2_CTS_B	ALT5
	IO23	UART2_RTS_B	ALT5
	IO24	UART3_TX_DATA	ALT5
	IO25	UART3_RX_DATA	ALT5
	IO26	UART3_CTS_B	ALT5
	IO27	UART3_RTS_B	ALT5
	IO28	UART4_TX_DATA	ALT5
	IO29	UART4_RX_DATA	ALT5
	IO30	UART5_TX_DATA	ALT5
	IO31	UART5_RX_DATA	ALT5

GPIO2	IO0	ENET1_RX_DATA0	ALT5
	IO1	ENET1_RX_DATA1	ALT5
	IO2	ENET1_RX_EN	ALT5
	IO3	ENET1_TX_DATA0	ALT5
	IO4	ENET1_TX_DATA1	ALT5
	IO5	ENET1_TX_EN	ALT5
	IO6	ENET1_TX_CLK	ALT5
	IO7	ENET1_RX_ER	ALT5
	IO8	ENET2_RX_DATA0	ALT5
	IO9	ENET2_RX_DATA1	ALT5
	IO10	ENET2_RX_EN	ALT5
	IO11	ENET2_TX_DATA0	ALT5
	IO12	ENET2_TX_DATA1	ALT5
	IO13	ENET2_TX_EN	ALT5
	IO14	ENET2_TX_CLK	ALT5
	IO15	ENET2_RX_ER	ALT5
	IO16	SD1_CMD	ALT5
	IO17	SD1_CLK	ALT5
	IO18	SD1_DATA0	ALT5
	IO19	SD1_DATA1	ALT5
	IO20	SD1_DATA2	ALT5
	IO21	SD1_DATA3	ALT5

GPIO3	IO0	LCD_CLK	ALT5
	IO1	LCD_ENABLE	ALT5
	IO2	LCD_HSYNC	ALT5
	IO3	LCD_VSYNC	ALT5
	IO4	LCD_RESET	ALT5
	IO5	LCD_DATA00	ALT5
	IO6	LCD_DATA01	ALT5
	IO7	LCD_DATA02	ALT5
	IO8	LCD_DATA03	ALT5
	IO9	LCD_DATA04	ALT5
	IO10	LCD_DATA05	ALT5
	IO11	LCD_DATA06	ALT5
	IO12	LCD_DATA07	ALT5
	IO13	LCD_DATA08	ALT5
	IO14	LCD_DATA09	ALT5
	IO15	LCD_DATA10	ALT5
	IO16	LCD_DATA11	ALT5
	IO17	LCD_DATA12	ALT5
	IO18	LCD_DATA13	ALT5
	IO19	LCD_DATA14	ALT5
	IO20	LCD_DATA15	ALT5
	IO21	LCD_DATA16	ALT5
	IO22	LCD_DATA17	ALT5
	IO23	LCD_DATA18	ALT5
	IO24	LCD_DATA19	ALT5
	IO25	LCD_DATA20	ALT5
	IO26	LCD_DATA21	ALT5
	IO27	LCD_DATA22	ALT5
	IO28	LCD_DATA23	ALT5

GPIO4	IO0	NAND_RE_B	ALT5
	IO1	NAND_WE_B	ALT5
	IO2	NAND_DATA00	ALT5
	IO3	NAND_DATA01	ALT5
	IO4	NAND_DATA02	ALT5
	IO5	NAND_DATA03	ALT5
	IO6	NAND_DATA04	ALT5
	IO7	NAND_DATA05	ALT5
	IO8	NAND_DATA06	ALT5
	IO9	NAND_DATA07	ALT5
	IO10	NAND_ALE	ALT5
	IO11	NAND_WP_B	ALT5
	IO12	NAND_READY_B	ALT5
	IO13	NAND_CE0_B	ALT5
	IO14	NAND_CE1_B	ALT5
	IO15	NAND_CLE	ALT5
	IO16	NAND_DQS	ALT5
	IO17	CSI_MCLK	ALT5
	IO18	CSI_PIXCLK	ALT5
	IO19	CSI_VSYNC	ALT5
	IO20	CSI_HSYNC	ALT5
	IO21	CSI_DATA00	ALT5
	IO22	CSI_DATA01	ALT5
	IO23	CSI_DATA02	ALT5
	IO24	CSI_DATA03	ALT5
	IO25	CSI_DATA04	ALT5
	IO26	CSI_DATA05	ALT5
	IO27	CSI_DATA06	ALT5
	IO28	CSI_DATA07	ALT5

GPIO5	IO0	SNVS_TAMPER0	No Muxing (ALT5)
	IO1	SNVS_TAMPER1	No Muxing (ALT5)
	IO2	SNVS_TAMPER2	No Muxing (ALT5)
	IO3	SNVS_TAMPER3	No Muxing (ALT5)
	IO4	SNVS_TAMPER4	No Muxing (ALT5)
	IO5	SNVS_TAMPER5	No Muxing (ALT5)
	IO6	SNVS_TAMPER6	No Muxing (ALT5)
	IO7	SNVS_TAMPER7	No Muxing (ALT5)
	IO8	SNVS_TAMPER8	No Muxing (ALT5)
	IO9	SNVS_TAMPER9	No Muxing (ALT5)
	IO10	BOOT_MODE0	No Muxing (ALT5)
	IO11	BOOT_MODE1	No Muxing (ALT5)

I2C1	SCL	CSI_PIXCLK	ALT3
		GPIO1_IO02	ALT0
		UART4_TX_DATA	ALT2
	SDA	CSI_MCLK	ALT3
		GPIO1_IO03	ALT0
		UART4_RX_DATA	ALT2
I2C2	SCL	CSI_HSYNC	ALT3
		GPIO1_IO00	ALT0
		UART5_TX_DATA	ALT2
	SDA	CSI_VSYNC	ALT3
		GPIO1_IO01	ALT0
		UART5_RX_DATA	ALT2
I2C3	SCL	ENET2_RX_DATA0	ALT3
		LCD_DATA01	ALT4
		UART1_TX_DATA	ALT2
	SDA	ENET2_RX_DATA1	ALT3
		LCD_DATA00	ALT4
		UART1_RX_DATA	ALT2
I2C4	SCL	ENET2_RX_EN	ALT3
		LCD_DATA03	ALT4
		UART2_TX_DATA	ALT2
	SDA	ENET2_TX_DATA0	ALT3
		LCD_DATA02	ALT4
		UART2_RX_DATA	ALT2