ANYKA
芯 随 你 动

版本: 1.0.5 2022 年 3 月

AnyCloud37E 平台

内核板级配置手册

声　　明

本手册所涉及的知识产权归属广州安凯微电子股份有限公司所有（或经合作商授权许可使用），任何人不得侵犯。

本手册不对包括但不限于下列事项担保：适销性、特殊用途的适用性；实施该用途不会侵害第三方的知识产权等权利。

广州安凯微电子股份有限公司不对由使用本手册或执行本手册内容而带来的任何损害负责。

本手册是按当前的状态提供参考，随附产品或本书内容如有更改，恕不另行通知。

联　系　方　式

广州安凯微电子股份有限公司

地址：广州市黄埔区知识城博文路 107 号安凯微电子 H 大厦
电话: (86)-20-3221 9000

传真: (86)-20-3221 9258

邮编: 510555

销售热线:

(86)-20-3221 9499

电子邮箱:

sales@anyka.com

主页:

http: //www. anyka.com

版 本 变 更 说 明

以下表格对于本文档的版本变更做一个简要的说明。版本变更仅限于技术内容的变更，不包括版式、格式、句法等的变更。

版本	说明	完成日期
V1.0.0	正式发布	2020 年 12 月
V1.0.1	1) 添加视频采集模块章节。 2) 更新 I2C 模块、MMC 模块内容。	2021 年 2 月
V1.0.2	1) 2.2 板级配置 dts 文件章节新增 mpu 接口，TP9950 以及 mipi 核心板增加采集功能的 dts 描述。 2) 2.4.2 RTC 章节 新增 adjust-based-on-timer 属性的描述。 3) 2.4.11.1.2 cs-gpios 属性调整为匹配当前的 GPIO 的配置。	2021 年 5 月
V1.0.3	1) 增加 standby 相关内容 2) 增加 I2S 相关内容 3) 增加 PDM 相关内容	2021 年 8 月
V1.0.4	1) 增加多 GPIO 唤醒、RTC ALARM 唤醒和 AIN0 唤醒 2) 增加 UART 流控 3) 增加 SPI1/2	2021 年 10 月
V1.0.5	1)LCDC 增加驱动能力设置的说明 2)LCDC 增加 reset/power/backlight 管脚使能电平的配置说明 3)UART 增加 RXD 悬空的配置说明	2022 年 3 月

目录

1 板级外设模块资源 5

2 内核编译及板级配置说明 6

2.1 内核编译说明 6
2.2 板级配置 DTS 文件 6
2.3 DTS 使用注意事项 10
2.4 DTS 文件各模块板级配置说明 10
2.4.1 GPIO/PINCTRL 14
2.4.2 RTC 16
2.4.3 Watchdog 16
2.4.4 USB 17
2.4.5 GUI 17
2.4.6 Vdecoder 17
2.4.7 Vencoder 17
2.4.8 MMC 17
2.4.9 ADC_DAC 22
2.4.10 SAR ADC 27
2.4.11 SPI 28
2.4.12 UART 36
2.4.13 I2C 38
2.4.14 Ethernet 41
2.4.15 LEDS 45
2.4.16 GPIOKEY 47
2.4.17 ADC-KEY 49
2.4.18 LCDC 52
2.4.19 PWM 57
2.4.20 Reserved memory 和 ION 59
2.4.21 VI 65
2.4.22 I2S 67
2.4.23 standby 70
2.5 小结 75

3 新增外设说明 76

3.1 新增 LCD 屏设备 76

3.1.1 *RGB LCD 屏*.....*76*
3.1.2 *MIPI LCD 屏*.....*77*
3.1.3 *注意事项*.....*80*
3.2 新增 SPI NOR FLASH 器件参数说明.....80
3.2.1 *dts*.....*80*
3.2.2 *BurnTool*.....*83*
3.3 新增 SPI NAND FLASH 器件参数说明.....84
3.3.1 *dts*.....*84*
3.3.2 *BurnTool*.....*86*

1 板级外设模块资源

AnyCloud37E 平台是针对 AK376XE 系列芯片（支持 AK3760E 主控芯片，但目前还不支持 AK3761E 主控芯片），基于 Linux 系统的平台解决方案。板级外设模块资源如下表所示。用户可根据产品功能需求自行修改各自模块的 status 使能状态。

表 1-1 AnyCloud37E 板级外设模块资源

外设资源	模块说明
gpio	各外设模块 Share-pin 相关配置
rtc	实时时钟模块
watchdog	看门狗模块
gui	2D 图形处理模块
vdecoder	视频解码模块
vencoder	视频编码模块
usb	USB 模块
mmc0/1/2	MMC/SD/SDIO 模块
adc_dac	ADC/DAC 音频模块
saradc	saradc 模块
spi0/1/2	spi 总线外设模块
uart0/1/2/3	Uart 模块
i2c0/1/2/3	I2C 模块
ethernet/ethernet1	网卡模块
leds	LED 指示灯
gpiokeys	GPIO 按键
adkeys	ADC 按键
lcdc	LCD 驱动模块， LCD 屏幕配置模块
pwm0/1/2/3/4/5	PWM 方波模块
vi0	视频采集模块
i2s	i2s 模块

外设资源	模块说明
pdm	pdm 模块

2 内核编译及板级配置说明

2.1 内核编译说明

特别声明：用户一般情况下无需修改内核配置重新编译内核镜像，如果重新改动了内核配置，有可能导致安凯驱动 ko 加载失败，出现这种情况请及时联系安凯 FAE。正常情况下只需要编译 dtbs 即可。

AnyCloud37E 平台 提供一键编译功能，下述是针对单独编译内核的编译说明。

AnyCloud37E 内核编译命令参考如下：

```
cd kernel
mkdir ../bd
make O=../bd/ anycloud_ak37e_mini_defconfig
make O=../bd/ menuconfig（在用户自行配置内核 feature 时使用，一般情况不需要）
make O=../bd/ uImage dtbs modules -j16
```

注意：

大多数情况，用户只需按照自己的硬件，修改板级配置 dts，编译 dtb 即可。平台相关的板级配置 dtb 镜像生成目录如下：*../bd/arch/arm/boot/dts*， 用户选择对应硬件的 dtb 镜像文件烧录即可（需要注意的是烧录时将该 dtb 拷贝到 *burntool* 目录下后需要重新命名为 cloudOS.dtb）。

用户可以通过 make O=../bd/ menuconfig 配置是否打开 nfs 功能，是否打开 ftrace 或者其他内核调试选项。但是正常情况，用户不需要修改内核配置。

2.2 板级配置 dts 文件

AnyCloud37E 平台针对 AK3760E/AK3761E，目前 PDK 支持 AK3760E+RGB LCD 或者 AK3760E+MIPI LCD 或者 AK3760E+MPU LCD 的组合方式，系统编译会生成相应的 dtb 文件，用户根据具体的硬件平台选择对应的 dtb 镜像文件下载即可。

硬件核心板名称	硬件版本说明	DTS 配置文件
EVB_CBDR_AK3760E_V1.0.1	DDR2 容量：64MB 屏接口：RGB Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash	EVB_CBDR_AK3760E_V1.0.1.dts
	DDR2 容量：64MB 屏接口：RGB Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash 同 EVB_CBDR_AK3760E_V1.0.1，由于 vi0 模块的 pin 脚跟 mmc2&mac1 冲突，默认关闭 mmc2 及 mac1，打开 vi0 节点。	EVB_CBDR_AK3760E_V1.0.1_camera.dts
	DDR2 容量：64MB 屏接口：RGB Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash 同 EVB_CBDR_AK3760E_V1.0.1_camera.dts，因 TP9950 波形要求，采用模拟 IIC 的方式支持 TP9950。	EVB_CBDR_AK3760E_V1.0.1_camera_TP9950.dts
	DDR2 容量：64MB 屏接口：RGB	EVB_CBDM_AK3760E_V1.0.1_spinand.dts

硬件核心板名称	硬件版本说明	DTS 配置文件
	Flash：单 SPI NAND Flash	
	DDR2 容量：64MB 屏接口：RGB Flash：单 SPI NAND Flash 同 EVB_CBDM_AK3760E_V1.0.1_spinand，由于 vi0 模块的 pin 脚跟 mmc2&mac1, 默认关闭 mmc2 及 mac1，打开 vi0 节点	EVB_CBDR_AK3760E_V1.0.1_spinand_camera.dts
EVB_CBDR_AK3760E_V1.0.1	DDR2 容量：64MB 屏接口：MPU Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash	EVB_CBDR_AK3760E_V1.0.1_mpu.dts
EVB_CBDM_AK3760E_V1.0.1	DDR2 容量：64MB 屏接口：MIPI Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash	EVB_CBDM_AK3760E_V1.0.1.dts
	DDR2 容量：64MB 屏接口：MIPI Flash：单 SPI NAND Flash	EVB_CBDM_AK3760E_V1.0.1_spinand.dts
	DDR2 容量：64MB 屏接口：MIPI Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI	EVB_CBDM_AK3760E_V1.0.1_camera.dts

硬件核心板名称	硬件版本说明	DTS 配置文件
	NAND Flash 同 EVB_CBDM_AK3760E_V1.0.1_camera，由于 vi0 模块的 pin 脚跟 mac1 冲突，默认关闭 mac1，打开 vi0 节点。	
	DDR2 容量：64MB 屏接口：MIPI Flash：单 SPI NAND Flash 同 EVB_CBDM_AK3760E_V1.0.1_camera	EVB_CBDM_AK3760E_V1.0.1_spinand_camera.dts
	DDR2 容量：64MB 屏接口：MIPI Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash 同 EVB_CBDM_AK3760E_V1.0.1_camera.dts，因 TP9950 波形要求，采用模拟 IIC 的方式支持 TP9950。	EVB_CBDM_AK3760E_V1.0.1_camera_TP9950.dts
	DDR2 容量：64MB 屏接口：MIPI Flash：单 SPI NOR Flash 或者 SPI NOR Flash + SPI NAND Flash 同	EVB_CBDM_AK3760E_V1.0.1_I2S.dts

硬件核心板名称	硬件版本说明	DTS 配置文件
	EVB_CBDM_AK3760E_V1.0.1.dts，但支持 I2S。	

DTS 文件可以通过 include 引用 dtsi 文件，AnyCloud37E 平台完整的 DTS 文件组织结构如下所示：

```
EVB_CBDR_AK3760E_V1.0.1.dts（或者其他板级配置 dts）
    |----anycloud_ak37e.dtsi  64M/128M MEM 内核启动参数及 CMA 内存配置
    |      |----anycloud_ak37e_common.dtsi   平台基础模块相关配置，包括 clk/irq 功能模块等
    |      |----anycloud_ak37e_pinctrl.dtsi 平台管脚的 sharepin 配置， 此处包含的是客户开发过程
    |            中较少改动的硬件模块的 pin 脚配置，如有改动可在板级配置 dts 中覆盖即可。
    |----anycloud_lcd.dtsi   LCD 屏相关参数配置
    |----anycloud_norflash.dtsi   spif0 nor flash 器件列表及其参数配置
    |----anycloud_nandflash.dtsi  spif1 nand flash 器件列表及其参数配置
```

2.3 DTS 使用注意事项

- pinctrl-x 不可省略；
- pinctrl 可配置 slew rate，input enable，drive strength ，pupd config，gpio 复用等功能，从而能适配外设的电气特性。

2.4 DTS 文件各模块板级配置说明

本章针对 DTS 板级配置文件中各模块节点具体说明各模块板级配置项。

常见的板级配置操作如下表所示，方便用户快速适配。

节点	属性	配置说明
GPIO	anyka,pins	pin 脚。
	anyka,function	复用功能。

节点	属性	配置说明
	anyka,pull	驱动能力、上下拉等。
RTC	osc-select	配置 RTC 时钟源选择。
	adjust-time	内部校准时间。
watchdog	def_heartbeat	最长喂狗间隔时间。
USB	-	-
GUI	-	-
vdecoder	-	-
MMC	detect mode	配置检测模式，有 non-removable、cd-gpio/cd-gpios、 cd_clk 等。
	bus-width	配置总线的带宽
	mciX_pins(eg. mci1_pins)	配置 SD/SDIO 接口使用的 GPIO 口。
	tf_en_pins	配置 SD 卡/SDIO_WIFI 电路的电源控制 GPIO，如 TF_EN 或 WIFI_EN 等。
	power-inverted	
	power-pins	
	cap-sdio-irq	如该接口外接 SDIO 外设，需要配置该属性。
ADC_DAC	speak_pins	配置控制 speaker 的 GPIO 引脚 SPK_EN，speak-gpios-en 配置打开 speaker 时的 GPIO 输出值。
	speak-gpios	
	speak-gpios-en	
	pdm_pins	配置 PDM 引脚。
SAR ADC	vref-select	参考电压选择。

节点	属性	配置说明
	battery-divider	电池电压检测分压使能。
	sampling-num	采用次数。
SPI	spiflash_pins	配置 SPI 接口使用的 GPIO 口。
	cap-spi-highspeed	配置为高速 SPI 则需要添加该属性，目前仅 spi0 支持该属性。
UART	uartX_pins(eg. uart0_pins)	配置 UART 接口使用的 GPIO 口。
I2C	i2cX_pins(eg. i2c0_pins)	配置 I2C 接口使用的 GPIO 口。
	i2s-dev-list	配置 I2S 设备列表
	i2s-bus-id	配置 I2S 总线
	i2s-mode	配置 I2S 工作模式
ethernet	ethernet_rmiiX_pins (eg. ethernet_rmii1_pins)	配置 RMII 接口使用的 GPIO 口。
	ethernet_rmiiX_rst_pins (eg. ethernet_rmii0_rst_pins)	配置 phy 的 reset 引脚。
	reset-gpios	
	phy-address	配置 phy 的地址。
leds	led_pins	配置 LED 使用的 GPIO 引脚。
	led-gpios	
lcdc	lcd_rgb_pins	配置 RGB 使用的 GPIO 口。

节点	属性	配置说明
	lcd_reset_pins	配置 RGB 的 reset 引脚。
	reset-pins	
	lcd_power_pins	配置 RGB 的 power 引脚。
	pwr-gpio	
	lcd-logo-width lcd-logo-height	配置 Logo 的宽高参数。
	lcd-logo-fmt0 lcd-logo-fmt1 lcd-logo-rgb-seq	配置 Logo 的格式参数。
	lcd-fb-type	缓存 buffer 个数。
pwm	period-ns	PWM 周期。
	duty-ns	PWM 周期中高电平时间。
	pwm-enable	PWM 默认工作标志。
reserved memory	cma_reserved	cma 属性的 reserved 内存区域。
	dma_reserved	内核不可见的 reserved 内存区域。
ion	cma_reserved_heap	cma 内存区域的 ion 描述。
	dma_reserved_heap	内核不可见内存区域的 ion 描述。
pm_standby	wakeup-mode	内核 standby 唤醒模式

节点	属性	配置说明
	wakeup-gpio	内核 standby 唤醒 GPIO
	wakeup-gpio-edge	内核 standby 唤醒 GPIO 触发沿
	wakeup-ain0-edge	AIN0 唤醒的 GPIO 触发沿
	pm_wakeup_pins	配置唤醒使用的 GPIO 口

2.4.1 GPIO/PINCTRL

gpio 节点用于配置各模块使用到的 share-pin 配置管理。用户需要根据 PG 文档和硬件设计原理图完成相对应硬件外设模块的 share-pin 配置，DTS 中关于 share-pin 相关属性配置说明如下：

```
/*
 * "anyka,function": func index, please refer to dt-bindings/pinctrl/ak_37e_pinctrl.h
 * "anyka,pull": configuration of the pins
 * bit[7:0]: pull up/down configuration
 *    bit[0]: pull up or pull down selection, 1 means select pull down, 0 means select pull up
 *    bit[4]: pull up or pull down enable, 1 means enable pull up&down function, 0 means disable fuction
 * !!! Notice that not all pin support pull up and pull down. Please refer to the programer's guide.
 * bit[15:8]: driver strength configuration.
 *     bit[9:8] drive strength.
 * bit[23:16]: input enable configuration
 *     bit[16]: 1 means enable input, 0 means disable input.
 * bit[31:24]: pin slew rate configuration
 *     bit[24]: 1 means enable slew rate, 0 means disable slew rate.
 * !!! Notice OTHER bits which not mentioned  please set to 0.
 */
```

举例说明如下：

```
uart0_pins: uart0_pins {
    anyka,pins = <XGPIO_022 XGPIO_023>;
    anyka,function = <XGPIO_022_FUNC_UART0_RXD XGPIO_023_FUNC_UART0_TXD>;
    anyka,pull = <0x00010000>;
};
```

anyka,pins 属性表示该模块使用的 GPIO 编号总集，对应的 GPIO 的 pin 脚的编号定义在头文件 dt-bindings/pinctrl/ak_37e_pinctrl.h 中，以 XGPIO_***格式的宏定义。

anyka,function 属性表示该模块 GPIO 功能复用总集，各个 GPIO 口的复用功能同样定义在 ak_37e_pinctrl.h 文件中，以 XGPIO_***_FUNC_***格式的宏定义。

anyka,pull 属性表示该模块 GPIO 属性配置总集，包含 slew rate，input enable，drive strength ，pupd config，如果所有 GPIO 属性配置一致，那么如上述例子所示的，用一个表示即可，否则使用 index 分别表示。

anyka,function 和 anyka,pull 属性使用 index 分别表示的示例如下：

```
icn85xx_pins: icn85xx_pins {
    anyka,pins = <XGPIO_068 XGPIO_069>;
    anyka,function = <XGPIO_068_FUNC_GPIO68 XGPIO_069_FUNC_GPIO69>;
    anyka,pull = <0x00000010 0x00010010>;
};
```

请留意，使用 index 分别表示的情况下，anyka,pins、anyka,function、anyka,pull 的个数需要保持一致。

设备驱动节点中通过如下属性引用相应的 GPIO 复用配置：

```
&uart0 {
    pinctrl-names = "default";
    pinctrl-0 = <&uart0_pins>;
    status = "okay";
};
```

特别说明：芯片提供的两个 GPI 管脚定义如下，其与 MIC 复用同一个 pin 脚：

```
#define XGPI_0                  95
#define XGPI_0_FUNC_MIC_P       0
#define XGPI_0_FUNC_GPI         1
#define XGPI_1                  96
```

```
#define XGPI_1_FUNC_MIC_N          0
#define XGPI_1_FUNC_GPI            1
```

2.4.2 RTC

RTC 模块节点配置说明如下：

```
&rtc {
    /*
     * rtc osc_source_select, 1 means INTERNAL RC OSC, 0 means EXTERNAL XTAL
     * if internal RC OSC is used to generate 32KH,time adjustment should be setting.(basic unit: ms)
     */
    osc-select = <1>;
    adjust-time = <120000>;
     adjust-based-on-timer;
    status = "okay";
};
```

osc_select 属性表示时钟源选择，1 表示内部时钟源，0 外部时钟源。

adjust-time 属性为内部校准的时间，单位为 ms。

adjust-based-on-timer 属性配置后系统时钟将来源于 24MHz 主时钟，否则系统时钟来源于 RTC 时钟计时。

status 属性表示使能状态，“okay”表示使能，“disable”表示关闭。

2.4.3 Watchdog

watchdog 模块节点配置说明如下：

```
&watchdog {
     /*max feed dog time = 357s, default setting time = 10s. */
    def_heartbeat = <10>;
    status = "okay";
};
```

def_heartbeat 属性表示 watchdog 超时时间，用户可以根据需求进行配置，单位为 s。此处配置的是默认值，应用可以通过接口配置实际使用值。

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.4 USB

```
&usb {
    status = "okay";
};
```

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.5 GUI

```
&gui {
    status = "okay";
};
```

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.6 Vdecoder

```
&vdecoder {
    status = "okay";
};
```

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.7 Vencoder

```
&vencoder{
    status = "okay";
};
```

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.8 MMC

MMC 总线可以配置为 MMC、SD 卡和 SDIO 总线功能，具体配置根据硬件设计需求来定。MMC 设备节点提供的配置选项如下：

```
&mmc1 {
    bus-width = <0x4>;
```

```
        max-frequency = <50000000>;
        /*
         *  detect mode:
         *  (1) non-removable;
         *  (2) detect-gpio/detect-gpios: card detect using gpio
         *      for example:
         *      detect-gpio = <&gpio XGPIO_076 1>;
         *  (3) cd_clk: card detect using MCLK
         */
        detect-gpio = <&gpio XGPIO_076 1>;
        /* if support highspeed, using cap-sd-highspeed */
        cap-sd-highspeed;
        pinctrl-names = "default","idle";
        pinctrl-0 = <&mci1_pins>,<&tf_en_pins>,<&tf_det_pins>;
        pinctrl-1 = <&mci1_idle_pins>;
        /*
        * support tf card circuit enable function
        */
        power-pins = <&gpio XGPIO_079 1>;      //power-pins is which tf_en_pins describes
        power-inverted;
        status = "okay";
    };
```

2.4.8.1 功能引脚配置

```
        pinctrl-names = "default","idle";
        pinctrl-0 = <&mci1_pins>,<&tf_en_pins>,<&tf_det_pins>;
        pinctrl-1 = <&mci1_idle_pins>;
```

如上属性配置 MCI 相关引脚的的功能配置。

总线引脚的 pinctrl 属性，由引用的 mci1_pins 节点描述。mci1_pins 节点的父节点为 gpio 节点，default 状态的配置如下：

```
    mci1_pins: mci1_pins {
```

```
    anyka,pins = <XGPIO_016 XGPIO_017 XGPIO_018 XGPIO_019 XGPIO_020
XGPIO_021>;
    anyka,function = <XGPIO_016_FUNC_SD1_D1
                XGPIO_017_FUNC_SD1_D0
                XGPIO_018_FUNC_SD1_MCLK
                XGPIO_019_FUNC_SD1_CMD
                XGPIO_020_FUNC_SD1_D3
                XGPIO_021_FUNC_SD1_D2>;
    anyka,pull = <0x00010010 0x00010010 0x00010001 0x00010010 0x00010010
0x00010010>;
  };
```

idle 状态为 SD/SDIO card 电路掉电时的引脚配置，pinctrl 属性由引用的 mci1_idle_pin 节点描述，配置为 gpio 功能，禁止输入，禁止上下拉，idle 状态的配置如下：

```
  mci1_idle_pins: mci1_idle_pins {
    anyka,pins = <XGPIO_016 XGPIO_017 XGPIO_018 XGPIO_019 XGPIO_020
XGPIO_021>;
    anyka,function = <XGPIO_016_FUNC_GPIO16
                XGPIO_017_FUNC_GPIO17
                XGPIO_018_FUNC_GPIO18
                XGPIO_019_FUNC_GPIO19
                XGPIO_020_FUNC_GPIO20
                XGPIO_021_FUNC_GPIO21>;
    anyka,pull = <0x00000000>;
  };
```

tf_en_pins 是供电引脚的配置，默认上拉，其配置如下：

```
  tf_en_pins: tf_en_pins {
    anyka,pins = <XGPIO_079>;
    anyka,function = <XGPIO_079_FUNC_GPIO79>;
    anyka,pull = <0x1010310>;
  };
```

tf_det_pins 是 T 卡插入的检测引脚，其默认配置如下：

```
  tf_det_pins: tf_det_pins{
```

```
    anyka,pins = <XGPIO_076>;
    anyka,function = <XGPIO_076_FUNC_GPIO76>;
    anyka,pull = <0x1010310>;
};
```

上述引脚可依据实际板子实际做变更适配。

AnyCloud37E 推荐采用 GPIO 引脚做卡的插拔检测，以获得稳定的检测效果。

2.4.8.2 bus-width 属性

bus-width 属性：总线数据线位数，可配置为 1 或 4，1 线模式配 1，4 线模式配 4。系统运行后，可通过 sysfs 查看 mci host 的 bus-width 属性，如 mci1 可通过 */sys/devices/platform/soc/20108000.mmc1/bus_width* 获取实际设置的 bus-width。

2.4.8.3 max-frequency 属性

max-frequency 属性：总线数据通信的最大频率值。

系统运行后，可通过 sysfs 查看 MMC/SDIO/SD 控制器的 max-frequency 属性，如 mci1 可通过/sys/devices/platform/soc/20108000.mmc1/f_max 获取实际设置的 max-frequency。

/sys/devices/platform/soc/20108000.mmc1/f_max 也提供写接口，通过 echo 可以动态修改控制器的最大的工作频率，最终设置的最大工作频率为“用户设置的最大工作频率”和 mmc host 可以设置的最大工作频率”的较小值。该值可以通过读取该接口获得。

而实际的总线时钟频率可以通过/sys/devices/platform/soc/20108000.mmc1/bus_clock 获取。

另外需要注意重新配置 f_max 之后，要重新插卡枚举后才生效。

2.4.8.4 detect mode 属性

SD 卡的插拔检测模式有 non-removable、cd-gpio/cd-gpios、cd_clk 等。

non-removable 为不可移除；

cd_clk 为使用 MCLK 引脚作为插拔检测的引脚；

detect-gpio/detect-gpios 为使用 GPIO 引脚作为插拔检测的引脚，需要配置具体的引脚，如使用 GPIO76，配置为 detect-gpio = <&gpio XGPIO_076 1>。

SDIO wifi 一般配置为 non-removable。

2.4.8.5 cap-sd-highspeed 属性

配置 MMC/SDIO/SD 控制器支持高速模式。

2.4.8.6 cap-sdio-irq 属性

配置 MMC/SDIO/SD 控制器支持 SDIO 中断。如使用 SDIO 外设，需要配置该属性。

2.4.8.7 power-pin 相关属性

```
power-pins = <&gpio XGPIO_079 1>;     //power-pins is which tf_en_pins describes
power-inverted;
```

2.4.8.8 power-pins 属性

配置 SD 卡/SDIO_WIFI 电路的电源控制 GPIO 引脚，如使用 GPIO81，配置方法为 power-pins= <&gpio XGPIO_081 1>（最后一列无须修改，均为 1）。

而该引脚的 pinctrl 属性，由引用的 tf_en_pins 节点描述。tf_en_pins 节点的父节点为 gpio 节点，配置如下：

```
&gpio {
        ......
        tf_en_pins: tf_en_pins {
           anyka,pins = <XGPIO_079>;
           anyka,function = <XGPIO_079_FUNC_GPIO79>;
           anyka,pull = <0x1010310>;
        };
        ......
};
```

2.4.8.8.1 power-inverted 属性

默认是低电平禁止 SD 卡/SDIO_WIFI 电路，高电平使能 SD 卡/SDIO_WIFI 电路。

如果板级硬件是高电平禁止 SD 卡/SDIO_WIFI 电路（SD 卡/SDIO_WIFI 电路掉电），低电平使能 SD 卡/SDIO_WIFI 电路，则需要配置 power-inverted 属性。

dts 按照实际板级配置后，应用层可以统一通过下述接口使能、禁止 SD 卡/SDIO_WIFI 电路。

echo 1 > /sys/mmc_en/mmcX_card_pwr_en： 使能 SD 卡/SDIO_WIFI 电路

echo 0 > /sys/mmc_en/mmcX_card_pwr_en： 禁止 SD 卡/SDIO_WIFI 电路

注意：如果是 mmc1 的话，则是路径/sys/mmc_en/mmc1_card_pwr_en，mmc2 以此类推。

2.4.8.9 status 属性

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.8.10 新增一款 SDIO WIFI 说明

新增一款新的 SDIO WIFI 驱动时，参考步骤如下：

- 使能对应的总线节点，如 mmc2，按照实际的电路原理图配置相关属性，属性配置的方法可参考章节 2.4.8.1-2.4.8.9 说明，主要的配置的内容有：
- 驱动使用到管脚配置，包括总线（如 mci2_pins）、供电管脚（如 wifi_en_pins）、重置管脚（如 wifi_rst_pins）等，可参考章节 2.4.1 中对 GPIO 管脚配置的说明；
- 数据线位数及频率配置（bus-width、max-frequency）等；
- SDIO WIFI 需增加 non-removable；
- 如果支持高速总线，则配置 cap-sd-highspeed 属性；
- 由于加入了 SDIO 外设，需配置 cap-sdio-irq；
- 适配模块供电配置，参考章节 2.4.8.8。

2.4.9 ADC_DAC

音频设备的输入输出源有：内部 ADC&DAC、I2S0 和 PDM。其中，内部 ADC&DAC 和 I2S0 支持播放、录音和环回；PDM 用于数字麦克风，只支持录音。

ADC_DAC 音频模块节点配置说明如下：

```
&adc_dac {
    pinctrl-names = "default", "sleep", "pdm_pins";
    pinctrl-0 = <&speak_pins>;
```

```
        pinctrl-1 = <&speak_pins_sleep>;
        pinctrl-2 = <&pdm_pins>;
        speak-gpios = <&gpio XGPIO_072 1>;
        /*
        * The level of speak-gpios votage for power on audio: 0=low, 1=high
        */
        speak-gpios-en = <1>;
        status = "okay";

        dev0 {
           snd-dev = <SND_CARD_DAC_PLAYBACK>;
           snd-type = <AK_PCM_DEV_PLAYBACK>;
        };
        dev1 {
           snd-dev = <SND_CARD_ADC_CAPTURE>;
           snd-type = <AK_PCM_DEV_CAPTURE>;
        };
        dev2 {
           snd-dev = <SND_CARD_DAC_LOOPBACK>;
           snd-type = <AK_PCM_DEV_LOOPBACK>;
        };
        dev3 {
           snd-dev = <SND_CARD_PDM_RECV>;
           snd-type = <AK_PCM_DEV_CAPTURE>;
        };
};
```

dev1、dev2 和 dev3 的配置请参考 2.4.9.3 PDM 及设备列表。

2.4.9.1 speak-gpios-en 相关属性

```
           pinctrl-names = "default", "sleep", "pdm_pins";
           pinctrl-0 = <&speak_pins>;
           pinctrl-1 = <&speak_pins_sleep>;
```

```
pinctrl-2 = <&pdm_pins>;
/* speak-gpios is which speak_en_pins describes */
speak-gpios = <&gpio XGPIO_072 1>;
/* the level of speak-gpios votage for power on audio: 0=low, 1=high*/
speak-gpios-en = <1>;
```

上述属性表示 SPK EN GPIO 的功能配置。default 表示正常工作模式，sleep 表示 standby 模式。

2.4.9.1.1 speak-gpios 属性

speak-gpios 属性，配置 SPEAKER EN 控制的 GPIO 引脚。如使用 GPIO72，则配置为 speak-gpios = <&gpio XGPIO_072 1>（最后一列无须修改，均为 1）。

而引脚的 pinctrl 属性，由引用的 speak_pins 节点描述。speak_pins 节点的父节点为 gpio 节点，配置如下：

```
speak_pins: speak_pins {
    anyka,pins = <XGPIO_072>;
    anyka,function = <XGPIO_072_FUNC_GPIO72>;
    anyka,pull = <0x00000010>;
};
```

speak_pins_sleep 节点描述引脚的 standby 下的引脚属性，配置如下：

```
speak_pins_sleep: speak_pins_sleep {
    anyka,pins = <XGPIO_072>;
    anyka,function = <XGPIO_072_FUNC_GPIO72>;
    anyka,pull = <0x00000000>;
};
```

2.4.9.1.2 speak-gpios-en 属性

speak-gpios-en 配置使能 speaker 时的电平。如果是高电平使能，则配置为:

```
speak-gpios-en = <1>;
```

如果为低电平使能，则配置为：

```
speak-gpios-en = <0>;
```

2.4.9.2 status 属性

表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.9.3 PDM 及设备列表

PDM 用于数字麦克风，只支持录音。PDM 的引脚通过 pdm_pins 配置：

```
pdm_pins: pdm_pins {
    anyka,pins = <XGPIO_054 XGPIO_055>;
    anyka,function = <XGPIO_054_FUNC_PDM_CLK
XGPIO_055_FUNC_PDM_DATA>;
    anyka,pull = <0x01010300>;
};
```

考虑到音频驱动会同时支持多种不同类型的设备，为方便功能配置，devX 子节点用于声明支持的设备列表：

```
dev0 {
    snd-dev = <SND_CARD_DAC_PLAYBACK>;
    snd-type = <AK_PCM_DEV_PLAYBACK>;
};
dev1 {
    snd-dev = <SND_CARD_ADC_CAPTURE>;
    snd-type = <AK_PCM_DEV_CAPTURE>;
};
dev2 {
    snd-dev = <SND_CARD_DAC_LOOPBACK>;
    snd-type = <AK_PCM_DEV_LOOPBACK>;
};
dev3 {
    snd-dev = <SND_CARD_PDM_RECV>;
    snd-type = <AK_PCM_DEV_CAPTURE>;
};
```

其中 snd-dev 表明该设备的硬件模块，目前支持的硬件模块如下：

- SND_CARD_DAC_PLAYBACK 内部的 DAC 模块

- SND_CARD_ADC_CAPTURE 内部的 ADC 模块
- SND_CARD_DAC_LOOPBACK 内部 DAC 的环回模块
- SND_CARD_I2S0_SEND 外部 I2S0 连接的发送模块
- ND_CARD_I2S0_RECV 外部的 I2S0 连接的接收模块
- SND_CARD_I2S0_LOOPBACK 外部 I2S0 连接的发送模块的环回模块
- SND_CARD_I2S1_RECV 保留给 I2S1，但未启用
- SND_CARD_PDM_RECV 外部的 PDM 连接的 ADC 模块

snd-type 表示该设备所属的类型，其定义如下：

- AK_PCM_DEV_PLAYBACK 播放设备
- AK_PCM_DEV_CAPTURE 采集设备
- AK_PCM_DEV_LOOPBACK 环回设备

如果同时支持多个设备，子设备节点请按照 dev0，dev1，dev2，dev3 依次命名的规则添加，考虑到不同平台上的硬件模块的复用情况可能不同，驱动未做设备的互斥检测。

注意，I2S0 与内部 ADC&DAC 存在硬件复用，两者只能互斥使用。PDM 与 I2S0 或者内部 ADC&DAC 可同时并存。另外，AK376XE 上 I2S1 由于和 PDM 存在硬件复用，两者只能互斥使用，所以 I2S1 未启用。

如上声明了 4 个设备节点，分别是内部 DAC 用于播放，内部 ADC 用于录音，内部 DAC 用于环回和 PDM 用于录音。需要注意的是，I2S 不在此 adc_dac 节中配置。

所有的设备节点均位于/dev 目录下，设备节点的命名规则为 pcmCxDy[c/p/l]，其中

Cx：C 代表的是不同的声卡，X 按照不同的声卡给予不同的定义，目前音频框架支持三类不同的输入输出源，分别是

x	代表的声卡
0	内部的 ADC&DAC 控制器
1	I2S 设备
2	PDM 设备

Dy：D 代表的是该类声卡下不同的设备号，例如 AK3760E 支持两路的 I2S（I2S0/I2S1），就会有 D0 代表 I2S0，D1 代表 I2S1。

[c/p/l]：代表的是不同的设备功能，其中 c 表示 capture，p 表示 playback，l 表示 loopback。

综上，音频设备节点有：

输入输出源	设备节点名称
内部 ADC&DAC	pcmC0D0p（播放设备）pcmC0D0c（录音设备）pcmC0D0l（环回设备）
I2S0	pcmC1D0p（播放设备）pcmC1D0c（录音设备）pcmC1D0l（环回设备）
PDM	pcmC2D0c（录音设备）

2.4.10 SAR ADC

```
&saradc{
        /* SAR ADC VREF selection
         * 0:To select AVCC 3.3V as VREF for SAR ADC.
        * 1:To select 3.0V generated from BGR as VREF for SAR ADC.
        */
        vref-select = <0>;
        /*
        * 0:To disable BAT voltage divider.
        * 1:To enable BAT voltage divider divided by 2.
        */
        battery-divider = <1>;
       /*
        * Sampling number
        * Read the sampling data for N times and then calculate the average value.
        * N ranges from 1 to 7.
        */
         sampling-num = <3>;
         status = "okay";
 };
```

2.4.10.1 vref-select 属性

vref-select =1 表示参考电压为 3.0V。

vref-select =0 表示参考电压为 3.3V。

2.4.10.2 battery-divider 属性

配置电池电压检测的分压使能。如果使能，则采用 2 分压，检测的电压范围则扩大到 2 倍。

```
/*
 * battery-divider
*  0:To disable BAT voltage divider.
*  1:To enable BAT voltage divider divided by 2.
*/
```

2.4.10.3 sampling-num 属性

表示 sar adc 采样的次数，根据多次采样的值求取其平均值，即为本次采样的值。

```
/*
 * Sampling number
 * Read the sampling data for N times and then calculate the average value.
 * N ranges from 1 to 7.
 */
```

2.4.10.4 status 属性

表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.11 SPI

每一条 SPI 总线可以挂载多个 SPI 总线设备，设备通过 CS 片选信号控制访问。目前 37E 开发板的硬件设计中 SPI0 总线挂载了 spi nor flash。用户根据实际硬件设计进行配置即可。

2.4.11.1 SPI 总线控制器节点

Spi0 的配置如下：

```
&spi0 {
    pinctrl-names = "default","sleep";
    pinctrl-0 = <&spiflash_pins>;
    pinctrl-1 = <&spiflash_pins_sleep>;
```

```
        spi-bus-frequency = <80000000>;
        /*
         * cap-spi-highspeed;
         * if defined, means high speed controller(Maximum 80M/s);
         * if not, means normal speed controller(below 60M/s)
         */
        cap-spi-highspeed;
        status = "okay";
    };
```

2.4.11.1.1 功能引脚配置

```
        pinctrl-names = "default","sleep";
        pinctrl-0 = <&spiflash_pins>;
        pinctrl-1 = <&spiflash_pins_sleep>;
```

如上属性配置 SPI 相关引脚的的功能配置。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 spiflash_pins 节点描述。引脚进入 standby 模式时的 pinctrl 属性，由引用的 spiflash_pins_sleep 节点描述。spiflash_pins 和 spiflash_pins_sleep 节点的父节点为 gpio 节点，默认配置定义在 anycloud_ak37e_pinctrl.dtsi 中，客户如果更改了对应的 pin 脚的组合，可在顶级的板级 dts 中重新赋值，默认配置如下：

```
        spiflash_pins: spiflash_pins {
            anyka,pins = <XGPIO_015
                        XGPIO_014
                        XGPIO_013
                        XGPIO_012
                        XGPIO_011
                        XGPIO_010
                        >;
            anyka,function = <XGPIO_015_FUNC_SPI0_DOUT
                            XGPIO_014_FUNC_SPI0_WP
                            XGPIO_013_FUNC_SPI0_SCLK
                            XGPIO_012_FUNC_SPI0_DIN
```

```
                    XGPIO_011_FUNC_SPI0_HOLD
                    XGPIO_010_FUNC_SPI0_CS0
                >;
        anyka,pull = <0x00010000>;
    };
    spiflash_pins_sleep: spiflash_pins_sleep {
        anyka,pins = <XGPIO_015
                XGPIO_014
                XGPIO_013
                XGPIO_012
                XGPIO_011
                XGPIO_010
                >;
        anyka,function = <XGPIO_015_FUNC_GPIO15
                    XGPIO_014_FUNC_GPIO14
                    XGPIO_013_FUNC_GPIO13
                    XGPIO_012_FUNC_GPIO12
                    XGPIO_011_FUNC_GPIO11
                    XGPIO_010_FUNC_GPIO10
                >;
        anyka,pull = <0x00000000>;
    };
```

2.4.11.1.2 cs-gpios 属性

cs-gpios 属性配置多 spi 子设备的 cs 控制的 GPIO 引脚。

如果使用一个子设备，cs 引脚使用 GPIO10，则配置为

```
cs-gpios = <&gpio XGPIO_010 1>;
```

如果使用两个子设备，cs 引脚分别使用 GPIO10 和 GPIO82，则配置为

```
cs-gpios = <&gpio XGPIO_010 1>, <&gpio XGPIO_082 1>;
```

注意：

1) **配置 cs-gpios 属性，驱动会将 cs 引脚配置成 GPIO 引脚来控制，而不是 spi 的功能引脚 cs。 由于驱动内部已经做了自适应，所以用户无需修改 spiflash_pins 节点中 cs 的功能引脚配置。**
2) **默认 cs 低电平有效。若需要设置为 cs 高电平有效，除了配置 cs-gpios 属性还需要增加 spi-cs-high 属性。**

2.4.11.1.3 spi-bus-frequency 属性

表示 SPI 总线支持的最大数据通信频率。

2.4.11.1.4 status 属性

表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.11.2 SPI 子设备节点

在 SPI0 总线下，根据片选 CS，预留了两个子设备接口，spif0 和 spif1。

2.4.11.2.1 spif0

spif0 子设备接口挂载了 SPI NOR Flash。

在 anycloud_ak37e_common.dtsi 文件中，有以下配置：

```
spif0:spi-flash@0 {
    compatible = "anyka,ak-spiflash";
    reg = <0>; /* Chip select 0 */
    spi-max-frequency = <80000000>;
    status = "disable";
    #address-cells = <1>;
    #size-cells = <0>;
};
```

compatible = "anyka,ak-spiflash";属性表示设备驱动匹配的名称。

reg = <0>; /* Chip select 0 */属性表示 CS 片选 ID 号。

spi-max-frequency = <30000000>;属性表示 SPI 设备数据通信的频率设置。

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

而在板级配置 dts 中，如 EVB_CBDR_AK3760E_V1.0.1.dts，有以下配置：

```
&spif0 {
    /* spi flash data bus width:
    ** 1 means 1 wire mode.
    ** 2 means 2 wire mode
    ** 4 means 4 wire mode.
    ** default: 4 wire mode.
    **/
    bus-width = <4>;
    status = "okay";
};
```

bus-width 配置 spi 总线的线宽，根据实际电路配置。

2.4.11.2.2 Spif1

Spif1 子设备接口挂载了 SPI NAND Flash。

在 anycloud_ak37e_common.dtsi 文件中，有以下配置：

```
spif1:spi-flash@1 {
    compatible = "anyka,ak-spinand";
    reg = <1>; /* Chip select 1 */
    spi-max-frequency = <80000000>;
    status = "disable";
};
```

compatible = "anyka,ak-spifnand";属性表示设备驱动匹配的名称。

reg = <1>; /* Chip select 1 */属性表示 CS 片选 ID 号。

spi-max-frequency = <30000000>;属性表示 SPI 设备数据通信的频率设置。

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

而在板级配置 dts 中，可按以下配置打开使能该节点：

```
&spif1 {
```

```
    status = "okay";
};
```

2.4.11.3 SPI NOR Flash 器件激活

SPI NOR Flash 有很多厂家的器件类型，我们将 SPI NOR Flash 器件列表放在 anycloud_norflash.dtsi 配置文件中。

添加或者删除一款 flash 均在上述配置文件中操作，但是激活 flash 器件的操作如下：

```
&gd25q64{
    status = "okay";
};
```

激活 8MByte GD SPI NOR Flash 器件，那么设备接口在识别设备的时候，会根据读取到的 flash jedec id，自动找到 flash 的 dts 配置参数信息；

2.4.11.4 SPI NAND Flash 器件激活

SPI NAND Flash 有很多厂家的器件类型，我们计划将 SPI NAND Flash 器件列表放在 anycloud_nandflash1.dtsi 配置文件中，目前 DTS 暂时未增加对 SPI NAND Flash 的支持。

添加或者删除一款 flash 均在上述配置文件中操作，但是激活 flash 器件的操作如下：

```
&GD5F1GQ4UB{
    status = "okay";
};
```

激活 1Gbit GD SPI NAND Flash 器件，那么设备接口在识别设备的时候，会根据读取到的 flash jedec id，自动找到 flash 的 dts 配置参数信息；

2.4.11.5 Flash 配置的注意事项

注意：驱动默认 cs 低电平有效，如果需要高电平有效则增加 cs-gpios 属性将 cs 引脚配置成 GPIO 引脚并增加 spi-cs-high 属性，而 spiflash_pins 无需修改 cs 引脚为 gpio 模式。

2.4.11.6 SPI1/2

AK3760E 支持 SPI1 和 SPI2 用于通用目的，每一条 SPI 总线可以挂载多个 SPI 总线设备，设备通过 CS 片选信号控制访问，用户根据实际硬件设计进行配置即可。由于管脚冲突，默认并没有打开 SPI1 和 SPI2。如果需要打开 SPI1 或 SPI2，下面以 spi2 为例进行配置：

```
&spi2 {
    pinctrl-names = "default","sleep";
    pinctrl-0 = <&spi2_pins>;
    pinctrl-1 = <&spi2_pins_sleep>;
    spi-bus-frequency = <50000000>;
    status = "okay";
};
```

其中，

pinctrl-names = "default","sleep";

pinctrl-0 = <&spi2_pins>;

pinctrl-1 = <&spi2_pins_sleep>;

如上属性配置 SPI 相关引脚的的功能配置。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 spi2_pins 节点描述。引脚进入 standby 模式时的 pinctrl 属性，由引用的 spi2_pins_sleep 节点描述。spi2_pins 和 spi2_pins_sleep 节点的父节点为 gpio 节点，spi2_pins 和 spi2_pins_sleep 中的管脚需根据实际硬件进行配置，示例配置如下：

```
spi2_pins: spi2_pins {
        anyka,pins = <XGPIO_024 XGPIO_025 XGPIO_026 XGPIO_027>;
        anyka,function = <XGPIO_024_FUNC_SPI2_CS0
                        XGPIO_025_FUNC_SPI2_SCLK
                        XGPIO_026_FUNC_SPI2_DIN
                        XGPIO_027_FUNC_SPI2_DOUT>;
        anyka,pull = <0x00010000>;
    };
    spi2_pins_sleep: spi2_pins_sleep {
        anyka,pins = <XGPIO_024 XGPIO_025 XGPIO_026 XGPIO_027>;
        anyka,function = <XGPIO_024_FUNC_GPIO24
                        XGPIO_025_FUNC_GPIO25
```

```
                    XGPIO_026_FUNC_GPIO26
                    XGPIO_027_FUNC_GPIO27>;
        anyka,pull = <0x00000000>;
    };
```

SPI 总线频率通过 spi-bus-frequency 配置：

spi-bus-frequency = <50000000>;

这里 SPI 总线频率配置为 50MHz。

status = "okay";

表示激活该 SPI 总线。

2.4.11.7 spidev

spidev 用于应用程序直接控制 SPI 总线上的数据收发，通过设备文件/dev/spidevX.X，SPI1 和 SPI2 都支持 spidev，可在 spi 节点下的 spidev 节点进行配置：

```
    spidev0: spidev@0 {
        compatible = "rohm,dh2228fv";
        reg = <0>; /* Chip select */
        spi-max-frequency = <50000000>;
        status = "okay";
    };
```

其中

spidev 的 compatible 属性固定使用 ohm,dh2228fv：

compatible = "rohm,dh2228fv"; reg 表示片选：

reg = <0>; /* Chip select */ 这里表示片选 0。

spi-max-frequency 表示器件最大 SPI 时钟频率：

spi-max-frequency = <50000000>;

这里表示该 spidev 最大 SPI 时钟频率是 50MHz。

status = "okay";

表示激活这个 spidev。

2.4.12 UART

37E 平台支持 4 组 UART，对应设备节点分别为 uart0，uart1，uart2，uart3，用户根据实际硬件设计情况配置使用相应的串口。uart 相关节点配置说明如下：

```
&uart0 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&uart0_pins>;
    pinctrl-1 = <&uart0_pins_sleep>;
    status = "okay";
};
```

```
pinctrl-names = "default", "sleep";
pinctrl-0 = <&uart0_pins>;
pinctrl-1 = <&uart0_pins_sleep>;
```

如上属性表示 UART 端口相关 GPIO 的功能配置，具体配置在 GPIO 节点中完成。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 uart0_pins 节点描述。引脚进入 standby 模式时的 pinctrl 属性，由引用的 uart0_pins_sleep 节点描述。

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

uart0_pins 和 uart0_pins_sleep 节点的父节点为 gpio 节点，默认配置定义在 anycloud_ak37e_pinctrl.dtsi 中，客户如果更改了对应的 pin 脚的组合，可在顶级的板级 dts 中重新赋值，默认配置如下：

```
uart0_pins: uart0_pins {
    anyka,pins = <XGPIO_022 XGPIO_023>;
    anyka,function = <XGPIO_022_FUNC_UART0_RXD
XGPIO_023_FUNC_UART0_TXD>;
    anyka,pull = <0x00010010 0x00000010>;
};
uart0_pins_sleep: uart0_pins_sleep {
    anyka,pins = <XGPIO_022 XGPIO_023>;
    anyka,function = <XGPIO_022_FUNC_GPIO22 XGPIO_023_FUNC_GPIO23>;
    anyka,pull = <0x00000000>;
};
```

注意：如果 UART 的 RXD 管脚悬空，建议打开 RXD 管脚的内置上拉，避免串口打印乱码。

2.4.12.1 硬件流控

UART1 支持硬件流控，如需要可如下配置：

```
&uart1 {
    cap_hw_flow_control;
    pinctrl-names = "default";
    pinctrl-0 = <&uart1_pins>;
    status = "okay";
};
```

cap_hw_flow_control 表示该 UART 支持硬件流控，37E 平台支持硬件流控的只有 UART1。默认 CTS 和 RTS 都是低电平有效。

```
    pinctrl-names = "default";
    pinctrl-0 = <&uart1_pins>;
```

如上属性表示 UART 端口相关 GPIO 的功能配置，具体配置在 GPIO 节点中完成。default 表示正常工作模式。引脚正常工作时的 pinctrl 属性，由引用的 uart1_pins 节点描述。如下配置：

```
uart1_pins: uart1_pins {
    anyka,pins = <XGPIO_070 XGPIO_071 XGPIO_072 XGPIO_073>;
    anyka,function = <XGPIO_070_FUNC_UART1_RXD
                      XGPIO_071_FUNC_UART1_TXD
                      XGPIO_072_FUNC_UART1_CTS
                      XGPIO_073_FUNC_UART1_RTS>;
    anyka,pull = <0x00010010 0x00010010 0x00010010 0x00010010>;
};
```

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

注意，如果要配置 CTS 和 RTS 为高电平有效，需要在上面配置上增加：

rts_active_low = <0>;

cts_active_low = <0>;

2.4.13 I2C

37E 平台支持 4 组 i2c 总线，对应设备节点分别为 i2c0，i2c1，i2c2，i2c3，用户根据实际硬件设计情况配置使用相应的 i2c 总线。i2c 相关节点配置说明如下：

```
&i2c2 {
    pinctrl-names = "default","sleep";
    pinctrl-0 = <&i2c2_pins>;
    pinctrl-1 = <&i2c2_sleep_pins>;
    clock-frequency = <312000>;
    sda-delay = <100>;
    status = "okay";
    #address-cells = <1>;
    #size-cells = <0>;
    touch-screen@40 {
        compatible = "anyka,ak37e-i2c-touchscreen-gslX680";
        reg = <0x40>;
        /* 10 inch MIPI touch-pane irq and reset gpio setting */
        pinctrl-names = "default";
        pinctrl-0 = <&gslX680_pins>;
        irq-gpio = <&gpio XGPIO_045 1>;
        reset-gpio = <&gpio XGPIO_044 1>;
        TP_MAX_X = <1280>;
        TP_MAX_Y = <800>;
        status = "disable";
      };

      touch-screen@48 {
        compatible = "anyka,ak37e-i2c-touchscreen-icn85xx";
        reg = <0x48>;
        /* 7 inch RGB touch-pane irq and reset gpio setting */
        pinctrl-names = "default";
```

```
        pinctrl-0 = <&icn85xx_pins>;
        irq-gpio = <&gpio XGPIO_069 1>;
        reset-gpio = <&gpio XGPIO_068 1>;
        TP_MAX_X = <1024>;
        TP_MAX_Y = <600>;
        status = "okay";
      };
  };
```

2.4.13.1 功能引脚配置

```
    pinctrl-names = "default","sleep";
    pinctrl-0 = <&i2c2_pins>;
    pinctrl-1 = <&i2c2_sleep_pins>;
```

如上属性配置 I2C 相关引脚的的功能配置。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 i2c2_pins 节点描述。引脚进入 standby 模式时的 pinctrl 属性，由引用的 i2c2_sleep_pins 节点描述。i2c2_pins 节点的父节点为 gpio 节点，配置如下：

```
  i2c2_pins: i2c2_pins {
    anyka,pins = <XGPIO_084 XGPIO_083>;
    anyka,function = <XGPIO_084_FUNC_TWI2_SDA XGPIO_083_FUNC_TWI2_SCL>;
    anyka,pull = <0x1010010>;
  };
```

i2c2_sleep_pins 节点的父节点为 gpio 节点，配置如下：

```
  &gpio {
    ......
    i2c2_sleep_pins: i2c2_sleep_pins {
      anyka,pins = <XGPIO_084 XGPIO_083>;
      anyka,function = <XGPIO_084_FUNC_GPIO84 XGPIO_083_FUNC_GPIO83>;
      anyka,pull = <0x0000000>;
    };
    ......
  };
```

2.4.13.2 clock-frequency 属性

配置 I2C 的时钟，最大为 400KHz。

2.4.13.3 sda-delay 属性

配置 I2C 的时序 sda_delay(data hold up time)时间，单位为 ns，一般为默认值，不用修改。

2.4.13.4 status 属性

表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.13.5 I2C 子设备节点——触摸屏

以“touch-screen@40”节点为例。表示触摸屏驱动的配置信息。

2.4.13.5.1 compatible

该属性是驱动的配置信息，不同触摸屏驱动有唯一标识。

2.4.13.5.2 reg

指触摸屏驱动芯片的 i2c 地址（7bit 地址）。

2.4.13.5.3 pinctrl-names

一般指定“default”，让系统默认就配置好 pinctrl。

2.4.13.5.4 pinctrl-0

pinctrl 的配置信息。

2.4.13.5.5 irq-gpio

触摸屏驱动芯片输出中断信号的引脚信息。

2.4.13.5.6 reset-gpio

触摸屏驱动芯片 reset 信号的引脚信息。

2.4.13.5.7 TP_MAX_X 和 TP_MAX_Y

触摸屏所表示的屏幕尺寸大小。

2.4.13.5.8 status 属性

表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.13.6 新增 I2C 设备说明

当新增一款新的 I2C 设备驱动时，dts 需要有设备的相关配置，实际操作中可参考如下：

1) 使能相关的 I2C 总线设备节点，配置相关属性，参考章节 2.4.13.1-2.4.13.4 对各个属性的描述，如上述的 i2c2 总线节点；
2) 添加 I2C 子设备节点，将子设备的 dts 节点配置写入到对应的总线节点中作为其子节点，如上述的 touch-screen@40 子节点与 i2c2 总线节点，如果该设备其他的管脚使用，重点留意的是管脚（中断管脚、供电控制管脚等）的配置，如下描述：

```
pinctrl-names = "default";
pinctrl-0 = <&gslX680_pins>;
```

将对应的管脚配置成 GPIO 功能，管脚的功能配置方式可参考章节 2.4.1 描述。

2.4.14 Ethernet

Ethernet 对应的设备节点说明如下：

```
&ethernet {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&ethernet_rmii0_pins &ethernet_rmii0_rst_pins>;
    pinctrl-1 = <&ethernet_rmii0_pins_sleep &ethernet_rmii0_rst_pins_sleep>;
    reset-gpios = <&gpio 85 1>;
    phy-address = <1>;
    status = "okay";
};

&ethernet1 {
    pinctrl-names = "default", "sleep";
```

```
        pinctrl-0 = <&ethernet_rmii1_pins &ethernet_rmii1_rst_pins>;
        pinctrl-1 = <&ethernet_rmii1_pins_sleep &ethernet_rmii1_rst_pins_sleep>;
        reset-gpios = <&gpio 86 1>;
        phy-address = <1>;
        status = "okay";
    };
```

AK37E 支持双网卡，支持的属性配置是一样的，下面以其中一个网卡的配置为例：

2.4.14.1 功能引脚配置

```
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&ethernet_rmii0_pins &ethernet_rmii0_rst_pins>;
    pinctrl-1 = <&ethernet_rmii0_pins_sleep &ethernet_rmii0_rst_pins_sleep>;
```

如上属性配置 Ethernet 相关引脚的的功能配置。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 ethernet_rmii0_pins 和 ethernet_rmii0_rst_pins 节点描述。引脚进入 standby 模式时的 pinctrl 属性，由引用的 ethernet_rmii0_pins_sleep 和 ethernet_rmii0_rst_pins_sleep 节点描述。ethernet_pins_rmii、ethernet_rmii0_rst_pins、ethernet_rmii0_pins_sleep 和 ethernet_rmii0_rst_pins_sleep 节点的父节点为 gpio 节点，配置定义在 anycloud_ak37e_pinctrl.dtsi 中，如引脚的组合发生变化，在顶级文件覆盖定义即可，其默认定义如下：

```
    ethernet_rmii0_pins: ethernet_rmii0_pins {
        anyka,pins = <XGPIO_000
                    XGPIO_001
                    XGPIO_002
                    XGPIO_003
                    XGPIO_004
                    XGPIO_005
                    XGPIO_006
                    XGPIO_007
                    XGPIO_008
                    XGPIO_009>;
        anyka,function = <XGPIO_000_FUNC_RMII0_MDIO
                        XGPIO_001_FUNC_RMII0_MDC
```

```
                    XGPIO_002_FUNC_RMII0_RXER
                    XGPIO_003_FUNC_RMII0_RXDV
                    XGPIO_004_FUNC_RMII0_RXD0
                    XGPIO_005_FUNC_RMII0_RXD1
                    XGPIO_006_FUNC_OPCLK0
                    XGPIO_007_FUNC_RMII0_TXD0
                    XGPIO_008_FUNC_RMII0_TXD1
                    XGPIO_009_FUNC_RMII0_TXEN
                    >;
        anyka,pull = <0x00010010 0x00010011 0x00010011 0x00010011 0x00010011
0x00010011 0x00010300 0x00010011 0x00010011 0x00010011>;
    };

    ethernet_rmii0_pins_sleep: ethernet_rmii0_pins_sleep {
     anyka,pins = <XGPIO_000
                XGPIO_001
                XGPIO_002
                XGPIO_003
                XGPIO_004
                XGPIO_005
                XGPIO_006
                XGPIO_007
                XGPIO_008
                XGPIO_009>;
     anyka,function = <XGPIO_000_FUNC_GPIO0
                    XGPIO_001_FUNC_GPIO1
                    XGPIO_002_FUNC_GPIO2
                    XGPIO_003_FUNC_GPIO3
                    XGPIO_004_FUNC_GPIO4
                    XGPIO_005_FUNC_GPIO5
                    XGPIO_006_FUNC_GPIO6
                    XGPIO_007_FUNC_GPIO7
```

```
                XGPIO_008_FUNC_GPIO8
                XGPIO_009_FUNC_GPIO9
                >;
    anyka,pull = <0x00000000>;
};

ethernet_rmii0_rst_pins: ethernet_rmii0_rst_pins {
    anyka,pins = <XGPIO_085>;
    anyka,function = <XGPIO_085_FUNC_GPIO85>;
    anyka,pull = <0x00010010>;
};

ethernet_rmii0_rst_pins_sleep: ethernet_rmii0_rst_pins_sleep {
    anyka,pins = <XGPIO_085>;
    anyka,function = <XGPIO_085_FUNC_GPIO85>;
    anyka,pull = <0x00000000>;
};
```

2.4.14.2 reset-gpios 属性

配置网卡设备 reset 功能的 GPIO 引脚。如使用 GPIO80，则配置为：

```
pinctrl-0 = <&ethernet_rmii0_pins &ethernet_rmii0_rst_pins>;
reset-gpios = <&gpio XGPIO_085 1>;
```

引脚的 pinctrl 属性，由引用的 ethernet_rmii0_rst_pins 节点描述。ethernet_rmii0_rst_pins 节点的父节点为 gpio 节点，配置如下：

```
ethernet_rmii0_rst_pins: ethernet_rmii0_rst_pins {
    anyka,pins = <XGPIO_085>;
    anyka,function = <XGPIO_085_FUNC_GPIO85>;
    anyka,pull = <0x00010010>;
};
```

2.4.14.3 phy-address 属性

```
phy-address = <1>;
```

该属性用于配置 PHY 地址，如果没有配置该属性，则默认 phy 的地址为 1。

2.4.14.4 mac-address 属性

```
mac-address = [00 00 00 00 00 00];
```

该属性用于配置 MAC 地址，如果没有配置该属性或者该地址无效，则随机产生 MAC 地址。

2.4.14.5 status 属性

表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.15 LEDS

LED 相关设备节点配置如下：

```
&leds {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&led_pins>;
    pinctrl-1 = <&led_pins_sleep>;
    status = "okay";
    state_led {
        label = "state_led";
        gpios = <&gpio XGPIO_078 GPIO_ACTIVE_HIGH>;
    };
};
```

2.4.15.1 引脚配置

```
pinctrl-names = "default", "sleep";
pinctrl-0 = <&led_pins>;
pinctrl-1 = <&led_pins_sleep>;
```

如上属性配置 LED 相关引脚的配置。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 led_pins 节点描述。引脚进入 standby 模式时

的 pinctrl 属性，由引用的 led_pins_sleep 节点描述。led_pins 和 led_pins_sleep 节点的父节点为 gpio 节点，配置如下：

```
led_pins:led_pins {
    anyka,pins = <XGPIO_078>;
    anyka,function = <XGPIO_078_FUNC_GPIO78>;
    anyka,pull = <0x00000010>;
};
led_pins_sleep:led_pins_sleep {
    anyka,pins = <XGPIO_078>;
    anyka,function = <XGPIO_078_FUNC_GPIO78>;
    anyka,pull = <0x00000000>;
};
```

2.4.15.2 led 子节点

如上所示，leds 节点中有 state_led 子节点。每个子节点包括 label 和 gpios 两个属性。如 state_led：

```
state_led {
    label = "state_led";
    gpios = <&gpio XGPIO_078 GPIO_ACTIVE_HIGH>;
};
```

label 为 led 的标签，会生成节点 /sys/devices/platform/leds/leds/state_led/brightness。

gpios 属性描述使用的 gpio 引脚，如上所示，使用 gpio78。

如果属性为 gpios = <&gpio XGPIO_078 GPIO_ACTIVE_HIGH>, 则当

echo 1 > /sys/devices/platform/leds/leds/state_led/brightness ， state_led 对应引脚输出高电平

如果属性为 gpios = <&gpio XGPIO_078 GPIO_ACTIVE_LOW>, 则当

echo 1 > /sys/devices/platform/leds/leds/state_led/brightness ， state_led 对应引脚输出低电平。

2.4.15.3 status 属性

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.15.4 使用说明

系统运行，加载驱动后，驱动会提供以下 sysfs 接口：

```
/sys/devices/platform/leds/leds/state_led/brightness
```

2.4.16 GPIOKEY

目前开发板上没提供 gpiokey 相关硬件，故开发板对应的 dts 中并没有使能对应的设备节点，gpiokey 驱动设备的节点参考配置如下：

```
&gpiokeys {
        #address-cells = <1>;
        #size-cells = <0>;
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&gpiokey_pins>;
        pinctrl-1 = <&gpiokey_pins_sleep>;
        status = "okay";

        key_detect {
                label = "key_detect";
                gpios = <&gpio XGPIO_074 1>;
                linux,code = <116>;
                debounce-interval = <100>;
                wakeup-source;
        };
};
```

2.4.16.1 引脚配置

```
pinctrl-names = "default", "sleep";
pinctrl-0 = <&gpiokey_pins>;
pinctrl-1 = <&gpiokey_pins_sleep>;
```

如上属性配置 GPIO-KEY 相关引脚。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 gpiokey_pins 节点描述。引脚进入 standby 模

式时的 pinctrl 属性，由引用的 gpiokey_pins_sleep 节点描述。gpiokey_pins 和 gpiokey_pins_sleep 节点的父节点为 gpio 节点，配置如下：

```
&gpio {
    ......
    gpiokey_pins:gpiokey_pins {
        anyka,pins = <XGPIO_074>;
        anyka,function = <XGPIO_074_FUNC_GPIO74>;
        anyka,pull = <0x01010010>;
    };
    gpiokey_pins_sleep:gpiokey_pins_sleep {
        anyka,pins = <XGPIO_074>;
        anyka,function = <XGPIO_074_FUNC_GPIO74>;
        anyka,pull = <0x00000000>;
    };
    ......
};
```

注意，当 gpiokey 同时用作唤醒 gpio 时，要删除 standby 节点 pm_standby 里面的 pinctrl 的状态。请参考 2.4.23 standby 章节。

2.4.16.2 key_detect 子节点

```
key_detect {
        label = "key_detect";
        gpios = <&gpio XGPIO_074 1>;
        linux,code = <116>;
        debounce-interval = <100>;
        wakeup-source;
};
```

- label 属性为按键的标签。
- gpios 属性为该按键使用的 GPIO 引脚。

  如 gpios = <&gpio XGPIO_074 1> 代表使用 GPIO74。
- linux,code 配置按键 event 的 code 值。

- debounce-interval 属性为防抖时间，单位 ms。

2.4.17 ADC-KEY

ADC-KEY 驱动设备的节点配置如下：

```
&adkeys {
    /*
    * poll interval: ms
    * debounce interval: ms
    */
    interval = <120>;
    debounce_interval = <20>;
    /*
    * if pressed-input = 0, input 0 when key is pressed, else input 1.
    */
    pressed-input = <0>;
    status = "okay";

    addet0 {
        unpress_min = <0>;
        unpress_max = <3200>;
        adc_index = <1>;
        adc_channel = <0>;

        key1 {
            linux,code = <102>; /*HOME*/
            min = <0>;
            max = <50>;
        };

        key2 {
            linux,code = <103>; /*UP*/
```

```
        min = <478>;
        max = <708>;
    };

    key3 {
        linux,code = <108>; /*DOWN*/
        min = <1020>;
        max = <1320>;
    };

    key4 {
        linux,code = <105>; /*LEFT*/
        min = <1510>;
        max = <1810>;
    };

    key5 {
        linux,code = <106>; /*RIGHT*/
        min = <1980>;
        max = <2280>;
    };

    key6 {
        linux,code = <28>; /*OK*/
        min = <2460>;
        max = <2760>;
    };

    key7 {
        linux,code = <158>; /*RETURN*/
        min = <2850>;
        max = <3100>;
```

```
    };
  };
};
```

2.4.17.1 interval 属性

配置 ADC-KEY 驱动内部检测按键的轮询时间，如配置 120ms，则为：

```
    interval = <120>;            /* poll interval ms */
```

2.4.17.2 debounce_interval 属性

配置 ADC-KEY 按键的反抖时间，如配置 20ms，则为

```
    debounce_interval = <20>;  /* debounce interval ms */
```

2.4.17.3 pressed-input 属性

配置 ADC-KEY 按键按下时，上报 0 还会上报 1。如果设置上报 0，则为：

```
    pressed-input = <0>;     /*if pressed-input == 0, input 0 when key is pressed,else input 1.*/
```

2.4.17.4 按键组子节点

```
  addet0 {
    unpress_min = <0>;
    unpress_max = <3200>;
    adc_index = <1>;
    adc_channel = <0>;

    key1 {
      linux,code = <102>; /*HOME*/
      min = <0>;
      max = <50>;
    };
  ....
  }
```

unpress_min 和 unpress_max 配置这组按键的按下时的电压范围（单位为 mV）。

而 key1/key2/key3 等节点中的 min 和 max 则是某个按键按下时的电压范围（单位为 mV）。如电压为 0~50mV，表示 key1 被按下。依次类推。

linux,code 配置按键 event 的 code 值，如 code 值为 102，则为：

```
linux,code = <102>;    //key value
```

2.4.17.5 status 属性

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.18 LCDC

LCD 控制器驱动设备节点配置说明如下：

```
&lcdc {
    pinctrl-names = "default","sleep";
    pinctrl-0 = <&lcd_rgb_pins &lcd_reset_pins &lcd_power_pins>;
    pinctrl-1 = <&lcd_rgb_pins_sleep &lcd_reset_pins &lcd_power_pins>;
    reset-pins = <&gpio 28 GPIO_ACTIVE_LOW>;
    pwr-gpio = <&gpio 43 GPIO_ACTIVE_LOW>;
    backlight-gpio = <&gpio 29 GPIO_ACTIVE_HIGH>;

    lcd-logo-width = <600>;
    lcd-logo-height = <165>;
    //[fmt1:fmt0] 00 16bits input(RGB565 or BGR565)
    //[fmt1:fmt0] 01 24bits input(RGB888 or BGR888)
    //[fmt1:fmt0] 10 or 11 32bits input(ARGB888 ABGR888 RGBA888 BGRA888)
    lcd-logo-fmt0 = <1>;
    lcd-logo-fmt1 = <0>;
    lcd-logo-rgb-seq = <1>; //0 for BGR, 1 for RGB
    lcd-fb-type = <1>; //0:single buffer; 1:double buffer
    status = "okay";
};
```

2.4.18.1 引脚配置

```
pinctrl-names = "default","sleep";
pinctrl-0 = <&lcd_rgb_pins &lcd_reset_pins &lcd_power_pins>;
pinctrl-1 = <&lcd_rgb_pins_sleep &lcd_reset_pins &lcd_power_pins>;
```

上述属性配置 lcd 的总线引脚、reset 引脚以及供电引脚。

default 表示正常工作模式，sleep 表示 standby 模式。引脚的 pinctrl 属性，由引用的 lcd_rgb_pins/lcd_reset_pins/lcd_power_pins/lcd_rgb_pins_sleep 节点描述，父节点为 gpio 节点，配置如下：

```
&gpio {
 ......
lcd_reset_pins: lcd_reset_pins {
   anyka,pins = <XGPIO_026>;
   anyka,function = <XGPIO_026_FUNC_GPIO26>;
   anyka,pull = <0x01000010>;
};

lcd_rgb_pins: lcd_rgb_pins {
   anyka,pins = <XGPIO_074  XGPIO_075  XGPIO_025 XGPIO_024 XGPIO_028
                 XGPIO_029 XGPIO_055 XGPIO_054  XGPIO_053 XGPIO_052
                 XGPIO_051 XGPIO_050 XGPIO_049  XGPIO_048 XGPIO_047
                 XGPIO_046 XGPIO_045 XGPIO_044 XGPIO_043 XGPIO_042
                 XGPIO_041 XGPIO_040  XGPIO_039 XGPIO_038 XGPIO_037
                 XGPIO_036  XGPIO_035 XGPIO_034>;
   anyka,function = <XGPIO_074_FUNC_RGB_D23 XGPIO_075_FUNC_RGB_D22
                     XGPIO_025_FUNC_RGB_D21 XGPIO_024_FUNC_RGB_D20
                     XGPIO_028_FUNC_RGB_D19 XGPIO_029_FUNC_RGB_D18
                     XGPIO_055_FUNC_RGB_D17 XGPIO_054_FUNC_RGB_D16
                     XGPIO_053_FUNC_RGB_D15  XGPIO_052_FUNC_RGB_D14
                     XGPIO_051_FUNC_RGB_D13 XGPIO_050_FUNC_RGB_D12
                     XGPIO_049_FUNC_RGB_D11 XGPIO_048_FUNC_RGB_D10
                     XGPIO_047_FUNC_RGB_D9 XGPIO_046_FUNC_RGB_D8
```

```
                XGPIO_045_FUNC_RGB_D7 XGPIO_044_FUNC_RGB_D6
                XGPIO_043_FUNC_RGB_D5  XGPIO_042_FUNC_RGB_D4
                XGPIO_041_FUNC_RGB_D3 XGPIO_040_FUNC_RGB_D2
                XGPIO_039_FUNC_RGB_D1 XGPIO_038_FUNC_RGB_D0
                XGPIO_037_FUNC_RGB_VOPCLK
                XGPIO_036_FUNC_RGB_VOHSYNC
                XGPIO_035_FUNC_RGB_VOVSYNC
                XGPIO_034_FUNC_RGB_VOGATE >;
    anyka,pull = < 0x00000010>;
};
    lcd_rgb_pins_sleep: lcd_rgb_pins_sleep {
        anyka,pins = <XGPIO_074 XGPIO_075 XGPIO_025 XGPIO_024 XGPIO_028
                      XGPIO_029 XGPIO_055 XGPIO_054 XGPIO_053 XGPIO_052
                      XGPIO_051 XGPIO_050 XGPIO_049 XGPIO_048 XGPIO_047
                       XGPIO_046 XGPIO_045 XGPIO_044 XGPIO_043 XGPIO_042
                       XGPIO_041 XGPIO_040 XGPIO_039 XGPIO_038 XGPIO_037
                       XGPIO_036 XGPIO_035 XGPIO_034>;
        anyka,function = <XGPIO_074_FUNC_GPIO74
                      XGPIO_075_FUNC_GPIO75
                      XGPIO_025_FUNC_GPIO25
                      XGPIO_024_FUNC_GPIO24
                      XGPIO_028_FUNC_GPIO28
                      XGPIO_029_FUNC_GPIO29
                      XGPIO_055_FUNC_GPIO55
                      XGPIO_054_FUNC_GPIO54
                      XGPIO_053_FUNC_GPIO53
                      XGPIO_052_FUNC_GPIO52
                      XGPIO_051_FUNC_GPIO51
                      XGPIO_050_FUNC_GPIO50
                      XGPIO_049_FUNC_GPIO49
                      XGPIO_048_FUNC_GPIO48
                      XGPIO_047_FUNC_GPIO47
```

```
                        XGPIO_046_FUNC_GPIO46
                        XGPIO_045_FUNC_GPIO45
                        XGPIO_044_FUNC_GPIO44
                        XGPIO_043_FUNC_GPIO43
                        XGPIO_042_FUNC_GPIO42
                        XGPIO_041_FUNC_GPIO41
                        XGPIO_040_FUNC_GPIO40
                        XGPIO_039_FUNC_GPIO39
                        XGPIO_038_FUNC_GPIO38
                        XGPIO_037_FUNC_GPIO37
                        XGPIO_036_FUNC_GPIO36
                        XGPIO_035_FUNC_GPIO35
                        XGPIO_034_FUNC_GPIO34
                        >;
            anyka,pull = <0x00000000>;
    };
    lcd_power_pins: lcd_power_pins {
        anyka,pins = <XGPIO_073>;
        anyka,function = <XGPIO_073_FUNC_GPIO73>;
        anyka,pull = <0x00000010>;
    };
    ......
};
```

注意：在不影响正常工作的前提下，RGB IO 的驱动能力须尽量使用满足要求的最小档（例如，若档位 1 可以满足要求，则设为档位 1，请勿提高至档位 2 或者档位 3），slew rate 须尽可能设为 slow 模式（即非必要时请勿设置为 fast 模式），以便减少同步开关噪音，降低电磁干扰。

2.4.18.2 reset-pins 属性

```
        reset-pins = <&gpio 28 GPIO_ACTIVE_LOW>;
```

reset-pins 对应 lcd_reset_pins 配置的 pin 脚。GPIO_ACTIVE_LOW 表示该管脚低电平使能。

2.4.18.3 pwr-gpio 属性

```
pwr-gpio = <&gpio 43 GPIO_ACTIVE_LOW>;
```

pwr-gpio 对应着 lcd_power_pins 配置的 pin 脚。GPIO_ACTIVE_LOW 表示该管脚低电平使能。

2.4.18.4 backlight-gpio 属性

```
backlight-gpio = <&gpio 29 GPIO_ACTIVE_HIGH>;
```

backlight-gpio 配置背光的 pin 脚。GPIO_ACTIVE_HIGH 表示该管脚高电平使能。

2.4.18.5 logo 属性

logo 属性如下：

```
lcd-logo-width = <600>;
lcd-logo-height = <165>;
//[fmt1:fmt0] 00 16bits input(RGB565 or BGR565)
//[fmt1:fmt0] 01 24bits input(RGB888 or BGR888)
//[fmt1:fmt0] 10 or 11 32bits input(ARGB888 ABGR888 RGBA888 BGRA888)
lcd-logo-fmt0 = <1>;
lcd-logo-fmt1 = <0>;
lcd-logo-rgb-seq = <1>; //0 for BGR, 1 for RGB
```

2.4.18.6 buffer 属性

```
lcd-fb-type = <1>; //0:single buffer; 1:double buffer
```

Lcd-fb-type=0 表示开辟一个视频 buffer 缓存

Lcd-fb-type=1 表示开辟两个视频 buffer 缓存

……

以此类推 lcd-fb-type 可以开辟任意个视频 buffer 缓存。但不建议配置 3 个以上的适配 buffer 缓存，这样会占用过多内存。

2.4.18.7 status 属性

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.18.8 lcd 子节点

LCD 子节点配置说明如下：

```
&sat101cp40 {
    status = "okay";
};

&am1080wq05 {
    status = "disable";
};
```

LCD panel 相关配置已经在 anycloud_lcd.dtsi 中进行配置，在 dts 文件中只需要选择相应的 panel 屏节点使能 status 状态即可，如上 sat101cp40 节点。

注意：LCD panel 节点仅能使能一个，否则会驱动设备加载重复导致异常。

2.4.19 PWM

PWM 驱动设备节点配置说明如下：

```
&pwm5 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&pwm5_pins>;
    pinctrl-1 = <&pwm5_pins_sleep>;
    /*
    * frequency of PWM ranges from 92/256Hz(0.36Hz) to 6MHz.
    * duty of PWM ranges from 1 to 65535.
    * Minimum duty cycle:1/65536
    * Maximum duty cyscle:65535/65536.
    * period-ns: period of the PWM signal (in nanoseconds)
    * duty-ns: duty cycle of the PWM signal (in nanoseconds)
    * pwm-enable = 0 means disable pwm function.
    * pwm-enable = 1 means enable pwm function.
    */
    period-ns = <1000000>;
    duty-ns = <500000>;
```

```
        pwm-enable = <1>;
        status = "okay";
    };
```

2.4.19.1 引脚配置

```
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&pwm5_pins>;
        pinctrl-1 = <&pwm5_pins_sleep>;
```

如上属性配置 PWM 相关引脚的配置。default 表示正常工作模式，sleep 表示 standby 模式。引脚正常工作时的 pinctrl 属性，由引用的 pwm5_pins 节点描述。引脚进入 standby 模式时的 pinctrl 属性，由引用的 pwm5_pins_sleep 节点描述。pwm5_pins 节点的父节点为 gpio 节点，配置如下：

```
    &gpio {
      ......
    pwm5_pins: pwm5_pins {
        anyka,pins = <XGPIO_027>;
        anyka,function = <XGPIO_027_FUNC_PWM5>;
        anyka,pull = <0x1000190>;
    };
      ......
  };
```

pwm5_pins_sleep 节点的父节点为 gpio 节点，配置如下：

```
    &gpio {
      ......
      pwm5_pins_sleep: pwm5_pins_sleep {
          anyka,pins = <XGPIO_027>;
          anyka,function = <XGPIO_027_FUNC_GPIO27>;
          anyka,pull = <0x00000000>;
      };
      ......
    };
```

2.4.19.2 周期属性

```
period-ns = <1000000>;
```

表示 pwm 的周期，单位为纳秒 ns

PWM 的频率范围在 92H to 6MHz。

2.4.19.3 占空比属性

```
duty-ns = <500000>;
```

duty-ns/period-ns 为占空比。

PWM 的占空比范围在 1/65536 to 65535/65536。

2.4.19.4 使能属性

```
pwm-enable = <1>;
```

此属性为 1 表示驱动加载后 pwm 马上工作，为 0 表示驱动加载后默认 pwm 不工作。

2.4.19.5 status 属性

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。

2.4.20 Reserved memory 和 ION

Reserved memory 和 ION 节点一般不需要修改。如果用户需要修改，可以参考下述。如有疑问，请及时联系安凯 FAE。

2.4.20.1 reserved memory 节点

```
reserved-memory {
    #address-cells = <1>;
    #size-cells = <1>;
    ranges;

    cma_reserved: cma_reserved@CMA_RESERVED_MEMORY_BASE {
        compatible = "shared-dma-pool";
        linux,cma-default;
```

```
        reusable;
        size = <CMA_RESERVED_MEMORY_SIZE>;
        reg = <CMA_RESERVED_MEMORY_BASE CMA_RESERVED_MEMORY_SIZE>;
        status = "okay";
    };

    dma_reserved: dma_reserved@DMA_RESERVED_MEMORY_SIZE {
        /*compatible = "shared-dma-pool";*/
        /*linux,cma-default;*/
        /*reusable;*/
        size = <DMA_RESERVED_MEMORY_SIZE>;
        reg = <DMA_RESERVED_MEMORY_BASE DMA_RESERVED_MEMORY_SIZE>;
        status = "okay";
    };
};
```

cma_reserved 是 cma 属性的 reserved 内存区域，主要用于驱动调用 dma_alloc_writecombine 或者 dma_alloc_coherent 分配内存的场合。

dma_reserved 是内核不可见的 reserved 内存区域，主要用于编码库和解码库等需要连续内存的场合。

size 属性，主要描述 reserved 内存区域的大小。如 size = <0x800000> 表示，内存区域大小为 8M。

reg 属性有两个值，前者表示内存区域的起始地址，后者表示内存区域的大小。 如 reg = <0x83400000 0x800000>表示， 内存区域的起始地址为 0x83400000， 内存区域的大小为 8M。

请注意，为了方便修改，对应的起始地址跟 size 属性，实际在 dts 中采用宏定义的方式。

2.4.20.2 ION 节点

ION 节点主要用于设置 ION 内存管理模块使用的属性。ION 的基本节点属性如下：

```
ion {
    compatible = "anyka,ak37e-ion-reserve";
```

```
        memory-region = <&dma_reserved>;
        #address-cells = <1>;
        #size-cells = <1>;
        heaps-nr = <1>;
        ranges;

        dma_reserved_heap: dma_reserved_heap {
            compatible = "anyka,dma-reserve";
            ion-id = <1>;
            /* ion-type: 0 = cma ; 1 = prereserved memory which kernel do not  aware of */
            ion-type = <1>;
            base-address = <DMA_RESERVED_MEMORY_BASE>;
            size = <DMA_RESERVED_MEMORY_SIZE>;
            align = <0x4>;
            status = "okay";
        };
    };
```

compatible 字段为 ion 的匹配字段。

memory-region 字段，描述 ion 驱动使用的 reserved 内存区域。该例子中，使用 2.4.20.1 中的 dma_reserved 区域。该例子中，该例子中有 1 个 heap，dma_reserved_heap 与 dma_reserved 区域对应。

heaps-nr 字段，描述 ion 驱动使用的堆的个数。

子节点 dma_reserved_heap 描述堆的属性。

ion-id 字段为堆的 id。注意目前的设计中， ion-id 为 <0>， 为 CMA 区域的预留内存；ion-id 为 <1>， 为内核不可见的预留内存。

base-address 字段，描述堆的起始地址。和 reserved memory 节点对应的内存区域一致。

size 字段，描述堆的大小。 和 reserved memory 节点对应的内存区域一致。

2.4.20.3 实例一

reserved-memory 只有 cma 属性的 reserved 内存区域(40M)。ION 只有一个堆，使用 cma 属性的 reserved 内存区域，为更加直观，下面例子的 base 地址跟 size 长度直接用数字表示：

```
reserved-memory {
    #address-cells = <1>;
    #size-cells = <1>;
    ranges;

    cma_reserved: cma_reserved@81400000 {
        compatible = "shared-dma-pool";
        linux,cma-default;
        reusable;
        size = <0x2800000>;
        reg = <0x81400000 0x2800000>;
        status = "okay";
    };
};

ion {
    compatible = "anyka,ak37e-ion-reserve";
    memory-region = <&cma_reserved>;
    #address-cells = <1>;
    #size-cells = <1>;
    heaps-nr = <1>;
    ranges;

    cma_reserved_heap: cma_reserved_heap {
        compatible = "anyka,cma-reserve";
        ion-id = <0>;
        base-address = <0x81400000>;
        size = <0x2800000>;
```

```
            align = <0x4>;
            status = "okay";
        };
    }
```

2.4.20.4 实例二

reserved-memory 有两个区域：cma 属性的 reserved 内存区域(8M)和内核不可见的 reserved 内存区域(32M)。ION 只有一个堆，使用内核不可见的 reserved 内存。

```
    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;
        cma_reserved: cma_reserved@83400000 {
            compatible = "shared-dma-pool";
            linux,cma-default;
            reusable;
            size = <0x800000>;
            reg = <0x83400000 0x800000>;
            status = "okay";
        };

        dma_reserved: dma_reserved@81400000 {
            size = <0x2000000>;
            reg = <0x81400000 0x2000000>;
            status = "okay";
        };
    };

    ion {
        compatible = "anyka,ak37e-ion-reserve";
        memory-region = <&dma_reserved>;
        #address-cells = <1>;
```

```
        #size-cells = <1>;
        heaps-nr = <1>;
        ranges;

        dma_reserved_heap: dma_reserved_heap {
            compatible = "anyka,dma-reserve";
            ion-id = <1>;
            base-address = <0x81400000>;
            size = <0x2000000>;
            align = <0x4>;
            status = "okay";
        };
    };
```

实际开发过程中为方便修改，使用到的内存 base 跟 size 参数，anycloud_ak37e.dtsi 采用宏定义的方式：

```
#define CMA_RESERVED_MEMORY_BASE  0x83400000
#define CMA_RESERVED_MEMORY_SIZE  0x800000
#define DMA_RESERVED_MEMORY_BASE  0x81400000
#define DMA_RESERVED_MEMORY_SIZE  0x2000000
/// 对应的节点引用：
cma_reserved: cma_reserved@CMA_RESERVED_MEMORY_BASE {
    compatible = "shared-dma-pool";
    linux,cma-default;
    reusable;
    size = <CMA_RESERVED_MEMORY_SIZE>;
    reg = <CMA_RESERVED_MEMORY_BASE CMA_RESERVED_MEMORY_SIZE>;
    status = "okay";
};
```

2.4.21 VI

VI 视频采集模块由控制器端驱动和 sensor 两个驱动组成。控制器端驱动负责系统资源的申请及图像的获取，sensor 驱动负责初始化和配置 sensor 硬件参数。

VI 视频采集模块的 dts 涉及两个节点：一是 I2C 节点，关联 sensor 驱动；二是 VI 节点，关联控制器端驱动。一般仅需要修改 reset-gpio 和 pwdn-gpio 属性，这两个节点的其他属性一般无需修改。

2.4.21.1 Sensor 相关的 i2c 配置

```
&i2c0 {
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_pins>;
    pinctrl-1 = <&i2c0_sleep_pins>;

    clock-frequency = <312000>;
    sda-delay = <100>;
    status = "okay";
    #address-cells = <1>;
    #size-cells = <0>;

    sensor0: camera@0x42{
        compatible = "anyka,sensor0";
        reg = <0x42>;
        pinctrl-names = "default";
        pinctrl-0 = <&sensor0_reset_pins &sensor0_pwdn_pins>;
        /*
        * 0xFFFF means the gpio is invalid
        */
        reset-gpio = <&gpio XGPIO_032 1>;
        pwdn-gpio = <&gpio XGPIO_071 1>;

        port {
```

```
        sensor0_0: endpoint@0 {
            remote-endpoint = <&vi_0_0>;
        };
    };
  };
};
```

pinctrl-names = "default";

pinctrl-0 = <&i2c0_pins>;

pinctrl-1 = <&i2c0_sleep_pins>;

如上属性表示 i2c 相关 GPIO 的功能配置，具体配置在 GPIO 节点中完成。

clock-frequency = <312000>;属性表示 I2C 的时钟，最大为 400KHz。

sda-delay = <100>;属性表示 I2C 开始发送数据时的延时，一般为默认值，不用修改。

#address-cells = <1>;

#size-cells = <0>;

如上两个属性固定。

compatible = "anyka,sensor0";属性匹配 sensor 驱动用，固定。

reg = <0x42>;属性表示 sensor 的 slave i2c 地址。

reset-gpio = <&gpio XGPIO_032 1>;属性表示 sensor 的 reset 控制脚。具体的电平要求在 sensor 驱动。

pwdn-gpio = <&gpio XGPIO_071 1>;属性表示 sensor 的 power down 控制脚。配置为 0xffff 表示该控制脚不可用。

remote-endpoint = <&vi_0_0>;属性为关联的节点，这里表示 sensor 输出至采集的节点。

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭。决定该驱动是否在内核中加载。

2.4.21.2 控制器端的 vi 配置

```
&vi0 {
    pinctrl-names = "default", "sleep", "dvp0_8bits", "csi0_sclk";
    pinctrl-0 = <&dvp0_8bits_pins &csi0_sclk_pin>;
    pinctrl-1 = <&dvp0_8bits_pins_sleep &csi0_sclk_pin_sleep>;
```

```
    pinctrl-2 = <&dvp0_8bits_pins>;
    pinctrl-3 = <&csi0_sclk_pin>;
    status = "okay";

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

        vi_0_0: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&sensor0_0>;
        };
    };
};
```

pinctrl-names = "default", "sleep", "dvp0_8bits", "csi0_sclk";

pinctrl-0 = <&dvp0_8bits_pins &csi0_sclk_pin>;

pinctrl-names、pinctrl-X 属性表示 vi 视频采集相关 GPIO 的功能配置，具体配置在 GPIO 节点中完成。

目前 AnyCloud37E 仅支持 8 线的 DVP 接口，因此配置 default 属性在驱动加载时就将对应的 GPIO 的管脚的功能引脚配置为对应的 DVP 管脚及时钟管脚。

#address-cells = <1>;

#size-cells = <0>;

如上两个属性固定。

remote-endpoint = <&sensor0_0>;属性为关联的节点，这里表示采集数据来源的节点。

status 属性表示使能状态，"okay"表示使能，"disable"表示关闭，决定该驱动是否在内核中加载。

2.4.22 I2S

同在 ADC_DAC 中介绍的那样，音频设备的输入输出源有：内部 ADC&DAC、I2S0 和 PDM。内部 ADC&DAC 和 PDM 在 adc_dac 节中配置。由于 I2S0 与内部 ADC&DAC 存在硬件复用，两者只能互斥使用，因此 I2S0 不在 adc_dac 节中配置。另外，AK376XE 上

I2S1 由于和 PDM 存在硬件复用，两者只能互斥使用，所以 I2S1 未启用。这里以 cs42l52 音频编解码器为例说明 I2S0 的配置。

cs42l52 是通过 I2C 进行控制的，因此要在 I2C 节点中增加如下配置：

```
&i2c3 {
.....
cs42l52@4A {
   compatible = "anyka,cs42l52";
   reg = <0x4A>;
   pinctrl-names = "default";
   pinctrl-0 = <&cs42l52_pins &i2s0_pins>;
   irq-gpio = <&gpio XGPIO_040 GPIO_ACTIVE_HIGH>;
   reset-gpio = <&gpio XGPIO_035 GPIO_ACTIVE_HIGH>;
   i2s-dev-list = <SND_CARD_I2S0_SEND SND_CARD_I2S0_RECV>;
   i2s-bus-id = <0>;
   clocks = <&audio_pll>,    //source_pll
         <&gclk_asic ASIC_GCLK_SD_ADC>,    //sdadc_gclk
         <&gclk_asic ASIC_GCLK_SD_DAC>,     //sddac_gclk
         <&factor_audio_clk AUDIO_FACTOR_I2S0_MCLK>,    //i2s0_mclk
         <&factor_audio_clk AUDIO_FACTOR_I2S0_B_LR_CLK>;    //i2s0_b_lr_clk
   clock-names =
         "source_pll",
         "sdadc_gclk",
         "sddac_gclk",
         "i2s0_mclk",
         "i2s0_b_lr_clk";
   status = "okay";
};
}
```

compatible = "anyka,cs42l52";

reg = <0x4A>;

reg 配置 cs42l52 的 I2C 地址。

pinctrl-names = "default";

pinctrl-0 = <&cs42l52_pins &i2s0_pins>;

cs42l52_pins 和 i2s0_pins 用于配置 cs42l52 的引脚，在 gpio 节点中，配置如下：

```
&gpio {
......
    cs42l52_pins: cs42l52_pins {
        anyka,pins = <XGPIO_035 XGPIO_040>;
        anyka,function = <XGPIO_035_FUNC_GPIO35 XGPIO_040_FUNC_GPIO40>;
        anyka,pull = <0x00000010 0x00010010>;
    };
     i2s0_pins: i2s0_pins{
        anyka,pins = <XGPIO_075 XGPIO_076 XGPIO_077 XGPIO_078 XGPIO_079>;
        anyka,function = <XGPIO_075_FUNC_I2S0_DIN XGPIO_076_FUNC_I2S0_LRCLK
XGPIO_077_FUNC_I2S0_BCLK XGPIO_078_FUNC_I2S0_MCLK
XGPIO_079_FUNC_I2S0_DOUT>;
        anyka,pull = <0x01010300>;
};
......
}
```

irq-gpio = <&gpio XGPIO_040 GPIO_ACTIVE_HIGH>;

reset-gpio = <&gpio XGPIO_035 GPIO_ACTIVE_HIGH>;

irq-gpio 和 reset-gpio 用于配置复位引脚和 IRQ 引脚，同 cs42l52_pins 中的配置。

i2s-dev-list = <SND_CARD_I2S0_SEND SND_CARD_I2S0_RECV>;

用于配置 I2S 设备列表，SND_CARD_I2S0_SEND 和 SND_CARD_I2S0_RECV 的定义参考 2.4.9 ADC_DAC 章节。这里定义了两个设备节点，分别是 I2S0 录音和 I2S0 播放。

i2s-bus-id = <0>;

配置 I2S 总线号。I2S1 未启用，不支持。

```
  clocks = <&audio_pll>,  //source_pll
        <&gclk_asic ASIC_GCLK_SD_ADC>,  //sdadc_gclk
        <&gclk_asic ASIC_GCLK_SD_DAC>,  //sddac_gclk
        <&factor_audio_clk AUDIO_FACTOR_I2S0_MCLK>,  //i2s0_mclk
        <&factor_audio_clk AUDIO_FACTOR_I2S0_B_LR_CLK>;  //i2s0_b_lr_clk
```

```
clock-names =
        "source_pll",
        "sdadc_gclk",
        "sddac_gclk",
        "i2s0_mclk",
        "i2s0_b_lr_clk";
```

clocks 和 clock-names 是配置 I2S 的 ADC、DAC、MCLK 和 LRCLK 的时钟。

status = "okay";

status 表示该设备是否启动。okay 表示启用，disable 表示不启用。

2.4.23 standby

standby 模块负责实现系统的休眠和唤醒功能，达到节省功耗的目的。目前支持 GPIO、RTC ALARM 和 AIN0 唤醒。GPIO/AIN0 唤醒可以选择上升沿或者下降沿唤醒。

pm_standby 节点描述了 standby 的配置，示例如下：

```
&pm_standby {
    pinctrl-names = "default";
    pinctrl-0 = <&pm_wakeup_pins>;
    /* linux kernel standby wakeup mode
    * wakeup mode:0x0:no_src, 0x1:GPIO, 0x2:rtc alarm, 0x3:ain0 */
    wakeup-mode = <0x1 0x2 0x3>;
    wakeup-gpio = <&gpio XGPIO_031 1>;
    /*
    * wakeup gpio trigger: 0x0:rising-edge,0x1:falling-edge,.
    */
    wakeup-gpio-edge = <0x1>;
   /*
    * wakeup ain0 trigger: 0x0:falling-edge, 0x1:rising-edge
    */
    wakeup-ain0-edge = <0x0>;
    status = "okay";
 };
```

其中，

pinctrl-names = "default";

pinctrl-0 = <&pm_wakeup_pins>;

pinctrl-0 属性表示 standby 唤醒的 GPIO 的 pinctrl 配置，具体配置在 pinctrl 节点的 pm_wakeup_pins 中完成，示例如下：

```
pm_wakeup_pins: pm_wakeup_pins {
    anyka,pins = <XGPIO_031>;
    anyka,function = <XGPIO_031_FUNC_GPIO31>;
    anyka,pull = <0x01010210>;
};
```

anyka,pins = <XGPIO_031>;

anyka,function = <XGPIO_031_FUNC_GPIO31>;

anyka,pins 和 anyka,function 配置唤醒源是 GPIO31。

anyka,pull = <0x01010210>;

anyka,pull 配置唤醒源 GPIO31 的属性。如果是下降沿触发唤醒的 GPIO，需要默认配置 pinctrl 为上拉、上拉使能和输入使能，驱动能力需根据硬件的实际电平进行选择。如果是上升沿触发唤醒的 GPIO，需要默认配置下拉、下拉使能和输入使能，驱动能力需根据硬件的实际电平进行选择，例如 0x00010011。

继续看 pm_standby 中的属性：

wakeup-mode = <0x1 0x2 0x3>;

wakeup-mode 这里表示支持 GPIO、RTC ALARM 和 AIN0 唤醒。

wakeup-gpio = <&gpio XGPIO_031 1>;

wakeup-gpio 设置唤醒源 GPIO 号，需要同上述的 pm_wakeup_pins 中的 GPIO 设置保持一致。最后一项保留，目前并没有使用。

wakeup-gpio-edge = <0x1>;

wakeup-gpio-edge 设置唤醒源的 GPIO 边沿触发方式，0 表示上升沿触发，1 表示下降沿触发。

wakeup-ain0-edge = <0x0>;

wakeup-ain0-edge 设置 ain0 唤醒边沿触发方式，0 表示下降沿触发，1 表示上升沿触发。

status = "okay";

status 表示是否使能 pm_standby，okay 表示打开，disable 表示关闭。

注意，要使能 standby 功能，内核需要打开以下配置选项：

CONFIG_PM=y

CONFIG_PM_SLEEP=y

CONFIG_FREEZER=y

CONFIG_SUSPEND=y

CONFIG_VT_CONSOLE_SLEEP=y

应用操作让系统进入 standby 模式，可在终端输入：

echo standby > /sys/power/state

唤醒系统则根据配置的 GPIO 口和上升沿/下降沿方式进行相应的操作。

支持唤醒源 GPIO 有：

GPIO2, GPIO8, GPIO9, GPIO16, GPIO20, GPIO22, GPIO24, GPIO28,
GPIO31, GPIO46, GPIO49, GPIO51, GPIO53, GPIO55, GPIO56, GPIO61,
GPIO67, GPIO68, GPIO70, GPIO72, GPIO75, GPIO78, GPIO84,
GPIO85, GPIO86

如果要配置多 GPIO 唤醒，需修改 pm_wakeup_pins 节点，pm_standby 中的 wakeup-gpio 和 wakeup-gpio-edge 等属性。假设要配置成 GPIO31 和 GPIO70 下降沿唤醒的话，如下修改：

```
pm_wakeup_pins: pm_wakeup_pins {
   anyka,pins = <XGPIO_031 XGPIO_070>;
   anyka,function = <XGPIO_031_FUNC_GPIO31 XGPIO_070_FUNC_GPIO70>;
   anyka,pull = <0x01010210 0x01010210>;
};
&pm_standby {
    pinctrl-names = "default";
    pinctrl-0 = <&pm_wakeup_pins>;
    /* linux kernel standby wakeup mode
```

```
    * wakeup mode:0x0:no_src, 0x1:GPIO, 0x2:rtc alarm, 0x3:ain0
    */
    wakeup-mode = <0x1 0x2 0x3>;

    wakeup-gpio = <&gpio XGPIO_031 1>, <&gpio XGPIO_070 1>;

    /*
    * wakeup gpio trigger: 0x0:rising-edge,0x1:falling-edge,.
    */
    wakeup-gpio-edge = <0x1>, <0x1>;

    /*
    * wakeup ain0 trigger: 0x0:falling-edge, 0x1:rising-edge
    */
    wakeup-ain0-edge = <0x0>;

    status = "okay";
};
```

AIN0 唤醒默认已打开，如果未打开，可如下修改 pm_standby 中的属性：

wakeup-mode = <0x1 0x2 0x3>; // 将 0x3 加入到其中

wakeup-ain0-edge = <0x0>; // 设置 AIN0 边沿触发方式，0 表示下降沿，1 表示上升沿

RTC ALARM 默认已打开，如果未打开，可如下修改 pm_standby 中的属性：

wakeup-mode = <0x1 0x2 0x3>; // 将 0x2 加入到其中

RTC ALARM 唤醒的使用方式如下：

1）如果 RTC 时间未设置或不准确，首先设置 RTC 时间，例如：

date -s "2021-1-1 12:30:10" && hwclock -w

2）如果要在 2021 年 1 月 2 日的 12:30:40 唤醒，则使用如下命令：

rtcwake -m standby -t `date -d "2021-1-2 12:30:40" +%s`

注意，如果设定时间已过，则系统将不会被唤醒。

注意，当唤醒 gpio 同时用作 gpiokey 时，要删除 pm_standby 节点里面的 pinctrl 设置和 pm_wakeup_pins 节点，否则会导致系统报错。修改方法是将 standby 节点中配置的 pinctrl 配置搬移到 gpiokey_pins_sleep 中，也就是让 gpiokey 驱动在系统 standby 前配置好这个 gpio。

以下是 gpio31 作为唤醒源并同时作为 gpiokey 的例子：

```
    gpiokey_pins:gpiokey_pins {
        anyka,pins = <XGPIO_031>;
        anyka,function = <XGPIO_031_FUNC_GPIO31>;
        anyka,pull = <0x01010010>;
    };
    gpiokey_pins_sleep:gpiokey_pins_sleep {
        anyka,pins = <XGPIO_031>;
        anyka,function = <XGPIO_031_FUNC_GPIO31>;
        anyka,pull = <0x01010210>;
};

// 删除 pm_wakeup_pins 节点
//pm_wakeup_pins: pm_wakeup_pins {
//    anyka,pins = <XGPIO_031>;
//    anyka,function = <XGPIO_031_FUNC_GPIO31>;
//    anyka,pull = <0x01010210>;
//};

&gpiokeys {
    #address-cells = <1>;
    #size-cells = <0>;
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&gpiokey_pins>;
    pinctrl-1 = <&gpiokey_pins_sleep>;
    status = "okay";
```

```
    key_detect {
        label = "key_detect";
        gpios = <&gpio 31 GPIO_ACTIVE_LOW>;
        linux,code = <116>;
        debounce-interval = <100>;
        wakeup-source;
    };
};

&pm_standby {
    // 删除 pinctrl-names 和 pinctrl-0 这两行
    //pinctrl-names = "default";
    //pinctrl-0 = <&pm_wakeup_pins>;
    wakeup-mode = <0x1>;
    wakeup-gpio = <&gpio XGPIO_031 1>;
    wakeup-gpio-edge = <0x1>;
    status = "okay";
};
```

2.5 小结

在 AnyCloud 37E 平台所有外设驱动均已经实现模块化开发，用户可以灵活的选择相应产品中需要用的模块驱动加载使用。用户使用到的相关模块需要根据对应的硬件设计在 DTS 中配置相关的 GPIO 功能配置，具体模块的配置方法参见上一章节相关内容。

模块加载和卸载指令如下：*insmod / rmmod*

模块驱动版本号查看方法：*cat /sys/modules/模块名/version*

3 新增外设说明

3.1 新增 LCD 屏设备

3.1.1 RGB LCD 屏

Step 1，在 anycloud_lcd.dtsi 中添加 RGB 屏 dts 节点，示例如下：

```
sat070cp50: lcd-panel@0 { //for RGB panel setting
    compatible = "sat,sat070cp50";
    panel-if = <0>; //0 for RGB, 1 for MIPI, 2 for MPU
    panel-rgb-seq = <1>; //0 for BGR, 1 for RGB
    panel-width = <1024>;
    panel-height = <600>;
    panel-pclk-div = <10>; //pixel clock = Peri_pll/(pclk_div+1)
    panel-thpw = <60>;
    panel-thb = <100>;
    panel-thf = <120>;
    panel-tvpw = <2>;
    panel-tvb = <21>;
    panel-tvf = <12>;
    panel-pclk-pol = <0>; //0:pos, 1:neg
    panel-hsync-pol = <1>;
    panel-vsync-pol = <1>;
    panel-vogate-pol = <0>;
    panel-bus-width = <3>; //0:8bits,1:16bits,2:18bits,3:24bits
    panel-sync-mode = <1>;

    status = "disable";
};
```

Step 2，调整 dts 节点中的带下划线部分参数。

Step 3，在板级配置 dts 文件中使能 lcd 节点，然后编译烧录 dtb 文件即可

参考 EVB_CBDR_AK3760E_V1.0.1.dts 中的节点：

```
&adt07016BR50 {
```

```
    status = "okay";
};
```

3.1.2 MIPI LCD 屏

Step 1, 在 anyka_lcd.dtsi 中添加 MIPI 屏 dts 节点，示例如下：

```
sat101cp40: lcd-panel@2 { //for JD9365 MIPI panel setting
        compatible = "sat,sat101cp40";
        panel-if = <1>; //0 for RGB, 1 for MIPI, 2 for MPU
        panel-rgb-seq = <1>; //0 for BGR, 1 for RGB
        panel-width = <800>;
        panel-height = <1280>;
        panel-pclk-div = <8>; //pixel clock = Peri_pll/(pclk_div+1)
        panel-dsi-num-lane = <3>; //0-3: 1lane-4lane
        //0-3: data0-data3, 4: clk
        panel-dsi-txd0 = <0>;
        panel-dsi-txd1 = <1>;
        panel-dsi-txd2 = <4>;
        panel-dsi-txd3 = <2>;
        panel-dsi-txd4 = <3>;
        panel-dsi-noncontinuous-clk = <0>; //0:continuous,1:noncontinuous
        panel-dsi-t-pre = <1>; //t-pre, >=1
        panel-dsi-t-post = <1>; //t-post=tlpx+t-prepare+t-zero+t-pre, >=1
        panel-dsi-tx-gap = <1>; //ths-exit
        panel-dsi-autoinsert-eotp = <0>; //0:not insert,1:auto-insert
        panel-dsi-htx-to-count = <0xFFFFFF>; //high speed TX timeout count
        panel-dsi-lrx-to-count = <0xFFFFFF>; //Low power RX timeout count
        panel-dsi-bta-to-count = <0xFFFFFF>; //Bus turn around timeout count
        panel-dsi-t-wakeup = <0xc8>; //DPHY T-wakeup time
        panel-dsi-pix-fifo-send-level = <512>; //buffer send level <= 512
        //0-2:16-bit config 1-3; 3-4:18-bit config 1-2; 5:24-bit
        panel-dsi-if-color-coding = <0x05>;
```

```
//0:16-bit;1:18-bit;2:18-bit loosely packed;3:24-bit
panel-dsi-pix-format = <0x03>;
panel-dsi-vsync-pol = <0>; //0:active low; 1:active high
panel-dsi-hsync-pol = <0>; //0:active low; 1:active high
//0:Non-burst mode with sync pulses;
//1:Noe-burst mode with sync events;
//2:Burst mode; 3:reserved
panel-dsi-video-mode = <2>;
panel-dsi-hfp = <40>; //H front porch blanking packet payload size in bytes
panel-dsi-hbp = <20>; //H back porch blanking packet payload size in bytes
panel-dsi-hsa = <20>; //H sync width blanking packet payload size in bytes
panel-dsi-mult-pkts-en = <0>; //0: single packet;1: two packets
panel-dsi-vbp = <8>; //V back porch lines
panel-dsi-vfp = <8>; //V front porch lines
panel-dsi-vsa = <4>;
//0:blanking packets send during BLLP;1:LP mode used for BLLP
panel-dsi-bllp-mode = <1>;
panel-dsi-use-null-pkt-bllp = <0>; //0:blanking packet; 1:Null packet
panel-dsi-vc = <0x00>; //Virtual channel
panel-dsi-cmd-type = <0x0>; //0:send in LP mode, 1:send in HS mode
panel-dsi-clk = <600>;
//add special config list here
panel-init-list = <0x00 0x15 0xE0 0x00 0xFFF   //delay, type, data, parameters, end symbol
                  0x00 0x15 0xE1 0x93 0xFFF
                  0x00 0x15 0xE2 0x65 0xFFF
                  .......
                  0x12C 0x15 0xE7 0x02 0xFFF
                  0x0 0x15 0x35 0x00 0xFFF
                  0xC8 0x05 0x11 0x00 0xFFF
                  0x96 0x05 0x29 0x00 0xFFF>;
```

```
        status = "disable";
    };
```

Step 2，调整 dts 节点中的带下划线部分参数。

panel-pclk-div = <8>; //pixel clock = Peri_pll/(pclk_div+1)

----调整 pclk 分频系数

panel-dsi-txd0 = <0>;

panel-dsi-txd1 = <1>;

panel-dsi-txd2 = <4>;

panel-dsi-txd3 = <2>;

panel-dsi-txd4 = <3>;

----根据硬件设计调整 mipi 4lane 线序

panel-dsi-hfp = <40>; //H front porch blanking packet payload size in bytes

panel-dsi-hbp = <20>; //H back porch blanking packet payload size in bytes

panel-dsi-hsa = <20>; //H sync width blanking packet payload size in bytes

panel-dsi-vbp = <8>; //V back porch lines

panel-dsi-vfp = <8>; //V front porch lines

panel-dsi-vsa = <4>;

----根据屏给出来的 RGB 时序资料设置相关时序参数

panel-dsi-vc = <0x00>; //Virtual channel

----MIPI 通信中数据包中的通道号，根据屏资料确定，一般为 0.

panel-dsi-clk = <600>;

----MIPI 通信线上频率（Mbps），根据我们芯片设计规格设置为 5 的整数倍

panel-init-list = < 0x00 0x15 0xE0 0x00 0xFFF

//delay, type, data, parameters, end symbol

----按照既定格式整理屏的初始化参数

Delay 表示该条指令发送完之后的延时时间

Type 表示 MIPI DSI 协议中定义的指令类型，常用的类型如下：

- 0x03 Generic Short WRITE, no parameters Short
- 0x13 Generic Short WRITE, 1 parameter Short
- 0x23 Generic Short WRITE, 2 parameters Short

- 0x05 DCS Short WRITE, no parameters Short
- 0x15 DCS Short WRITE, 1 parameter Short
- 0x29 Generic Long Write Long
- 0x39 DCS Long Write/write_LUT Command Packet Long

Data，parameters 表示该条指令中需要发送的数据（屏厂给出来的初始化指令参数序列）

End symbol 表示该指令发送结束符

Step 3，在板级配置 dts 文件中使能 lcd 节点，然后编译烧录 dtb 文件即可

参考 EVB_CBDR_AK3760E_V1.0.1.dts 中的节点：

```
&sat101cp40 {
    status = "okay";
};
```

3.1.3 注意事项

在 Step 3 中增加 LCD 节点时候需要注意 dts 文件中只能有一个 LCD 节点的 status 是 okay 的状态，如果配置多个 MIPI LCD 节点使能，可能会导致 LCD 驱动加载初始化屏幕异常。

后续调试后的 LCD 参数增加节点到 anyka_lcd.dtsi 即可，在客户开发使用中进行板级配置的时候选择具体型号节点，添加状态使能属性即可使用。

3.2 新增 SPI NOR Flash 器件参数说明

3.2.1 dts

在 anycloud_norflash.dtsi 中新增描述节点：

```
gd25q64: spi-norflash@46 {
    compatible = "gd,gd25q64";
    #clock-cells = <1>;
    norflash-name = "gd25q64";
    reg = <46>;
```

```
/* spiflash gd25q64&gd25q64c */
norflash-jedec-id = <0xc84017>;
norflash-ext-id = <0>;
norflash-sector-size = <0x10000>;
norflash-n-sectors = <128>;

/**
* flags:chip character bits:
* bit 0: under_protect flag; bit 1: fast read flag; bit 2: AAI flag; bit 3: support status2; bit 4: support dual io read; bit 5: support dual read
* bit 6: support quad io read; bit 7: support quad read; bit 8: support dual io write; bit 9: support dual write; bit 10: support quad io write
* bit 11: support quad write; bit 12: support 4k erase sector; bit 13: support status3; bit 14: support no QE ; bit 15:write status mode
*/

/*
* flags = SFLAG_SECT_4K|SFLAG_COM_STATUS2|
*
SFLAG_DUAL_READ|SFLAG_QUAD_READ|SFLAG_DUAL_IO_READ|SFLAG_QUAD_IO_READ|SFLAG_QUAD_WRITE
* wr_mode:
*    0:means one write status cmd can wirte all status regs; 1:means one write status cmd can only wirte one status reg
* norflash-wr_flags: write wr startus regs count
*/
norflash-flags = <0x18F8>;
rd_status_cmd = <0x05>, <0x35>, <0x15>;
wr_status_cmd = <0x01>, <0x31>;
wr_mode = <1>;
norflash-wr_flags = <2>;

/* status reg bit map */
```

```
            norflash-b-wip = <0>;
            norflash-b-wel = <1>;
            norflash-b-bp0 = <2>;
            norflash-b-bp1 = <3>;
            norflash-b-bp2 = <4>;
            norflash-b-bp3 = <5>;
            norflash-b-bp4 = <6>;
            norflash-b-srp0 = <7>;
            norflash-b-srp1 = <8>;
            norflash-b-qe = <9>;
            norflash-b-cmp = <14>;
            norflash-b-sus = <15>;

            status = "disable";
        };
```

相关参数说明如下：

compatible = "gd,gd25q64"; 描述器件的厂家属性类型；

norflash-name = "gd25q64"; 描述器件的型号；

reg = <46>; 描述器件列表中，该款器件在列表中的序列号；

norflash-jedec-id = <0xc84017>; 描述该 flash 的 jedec-id 的 id 值，用于识别 flash 的型号；

norflash-ext-id = <0>; 描述该 flash 的扩展 id 值；

norflash-sector-size = <0x10000>;描述该 flash 的块大小；

norflash-n-sectors = <128>; 描述该 flash 的块数量；

norflash-flags = <0x18F8>; 描述该 flash 的 flags 参数，包括：擦除块大小，状态寄存器个数和读写的一线或者四线操作；

rd_status_cmd = <0x05>, <0x35>, <0x15>; 描述该 flash 的读状态寄存器命令；

wr_status_cmd = <0x01>, <0x31>; 描述该 flash 的写状态寄存器命令；

wr_mode = <1>; 描述该 flash 的写状态寄存器模式，为 1 则一个指令写一个状态寄存器，为 0 则一个指令写多个状态寄存器；

norflash-wr_flags = <2>; 描述该 flash 的写状态寄存器个数；

/* status reg bit map */

norflash-b-wip = <0>; 描述该 flash 的状态寄存器中的 wip 为对应的 Bit 序号是第 0 bit；其它状态寄存器属性描述类似；

status = "disable"; 默认 flash 器件的激活状态时关闭的；

如需使用该器件，在对应的板级配置 dts 中使能配置即可：

```
&gd25q64{
    status = "okay";
};
```

3.2.2 BurnTool

查看平台烧录工具 BurnTool 的配置，在菜单栏【配置文件】选择“config.txt”，此为 SPI NOR Flash 的配置文件，并在**设置->研发者->SpiFlash** 中根据 flash 芯片的数据手册填写列表中相关的信息，配置界面如下图所示：

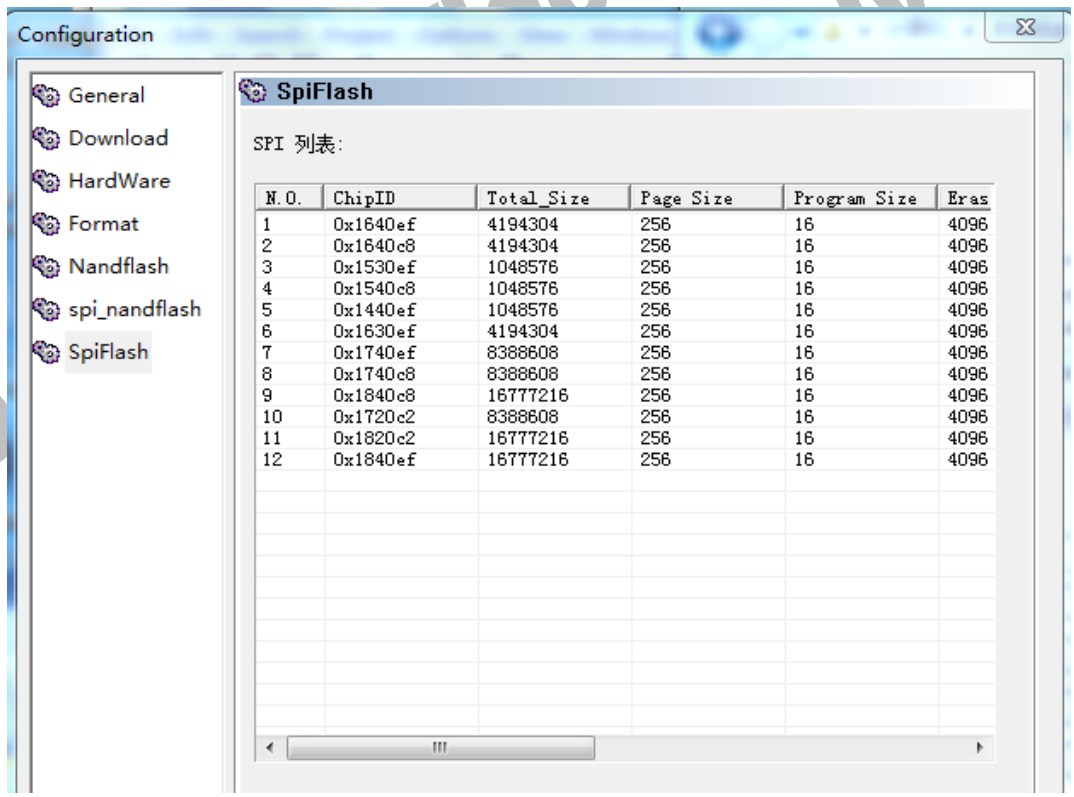

图 3-1 SPI NOR Flash 配置界面

注意：烧录工具中的 ID 和驱动中 ID 相反。如驱动读取的 ID 为 0xef4016，烧录工具 ID 则为 0x1640ef。

3.3 新增 SPI NAND Flash 器件参数说明

3.3.1 dts

```
GD5F1GQ4UB: spi-nandflash@0 {
        compatible = "gd,GD5F1GQ4UB";
        nandflash-name = "GD5F1GQ4UB";
        /* spiflash GD5F1GQ4UB */
        nandflash-jedec-id = <0xc8d1c8d1>;
        nandflash-ext-id = <0>;
        nandflash-page_size = <2048>;
        nandflash-page_per_block = <64>;
        nandflash-block_size = <0x20000>;
        nandflash-n_blocks = <1024>;
        nandflash-oob_size = <64>;
        nandflash-oob_up_skiplen = <0>;
        nandflash-oob_seglen = <12>;
        nandflash-oob_down_skiplen = <4>;
        nandflash-oob_seg_perpage = <4>;
        nandflash-oob_vail_data_offset = <4>;
    /*
    * flags = SFLAG_DUAL_READ|SFLAG_QUAD_READ|SFLAG_QUAD_WRITE
    * = 0x8A0
    */
        nandflash-flags = <0x8A0>;

        /* status reg bit map */
        nandflash-b_wip = <0>;
        nandflash-b_wel = <1>;
        nandflash-b_efail = <2>;
        nandflash-b_pfail = <3>;
        nandflash-b_bp2 = <4>;
        nandflash-b_bp3 = <5>;
```

```
            nandflash-b_bp4 = <6>;
            nandflash-b_srp0 = <7>;

            nandflash-b_qe = <8>;
            nandflash-b_srp1 = <9>;
            nandflash-b_lb = <10>;

            nandflash-b_cmp = <14>;
            nandflash-b_sus = <15>;

            status = "disable";
        };
```

相关参数说明如下：

compatible = "gd,GD5F1GQ4UB"; 描述器件的厂家属性类型；

nandflash-name = "GD5F1GQ4UB"; 描述器件的型号；

nandflash-jedec-id = <0xc8d1c8d1>; 描述该 nandflash 的 jedec-id 的 id 值，用于识别 flash 的型号；

nandflash-ext-id = <0>; 描述该 nandflash 的扩展 id 值；

nandflash-page_size = <2048>; 描述该 nandflash 的页大小；

nandflash-page_per_block = <64>; 描述该 nandflash 的每块包含的页数量；

nandflash-block_size = <0x20000>; 描述该 nandflash 的每块大小；

nandflash-n_blocks = <1024>; 描述该 nandflash 包含的块数量；

nandflash-oob_size = <64>; 描述该 nandflash 的 oob 区域大小；

nandflash-oob_up_skiplen = <0>; 描述该 nandflash 的 oob up 区域长度；

nandflash-oob_seglen = <12>; 描述该 nandflash 的 oob 段长度；

nandflash-oob_down_skiplen = <4>; 描述该 nandflash 的 oob down 区域长度；

nandflash-oob_seg_perpage = <4>; 描述该 nandflash 每页包含的 oob 段数量；

nandflash-oob_vail_data_offset = <4>; 描述该 nandflash oob vail 数据段偏移；

nandflash-flags = <0x8A0>; 描述该 nandflash 的 flags 参数，包括：读写的一线或者四线操作；

/* status reg bit map */

nandflash-b_wip = <0>; 描述该 flash 的状态寄存器中的 wip 为对应的 Bit 序号是第 0 bit；其它状态寄存器属性描述类似；

status = "disable"; 默认 flash 器件的激活状态时关闭的；

如需使用该器件，在对应的板级配置 dts 中使能配置即可：

```
&GD5F1GQ4UB{
   status = "okay";
};
```

3.3.2 BurnTool

看平台烧录工具 BurnTool 的配置，在菜单栏【配置文件】选择“spinand_config.txt”，此为 SPI NAND Flash 的配置文件，在**设置->研发者->spi_nandflash** 中根据 flash 芯片的数据手册填写列表中相关的信息，如下图所示：

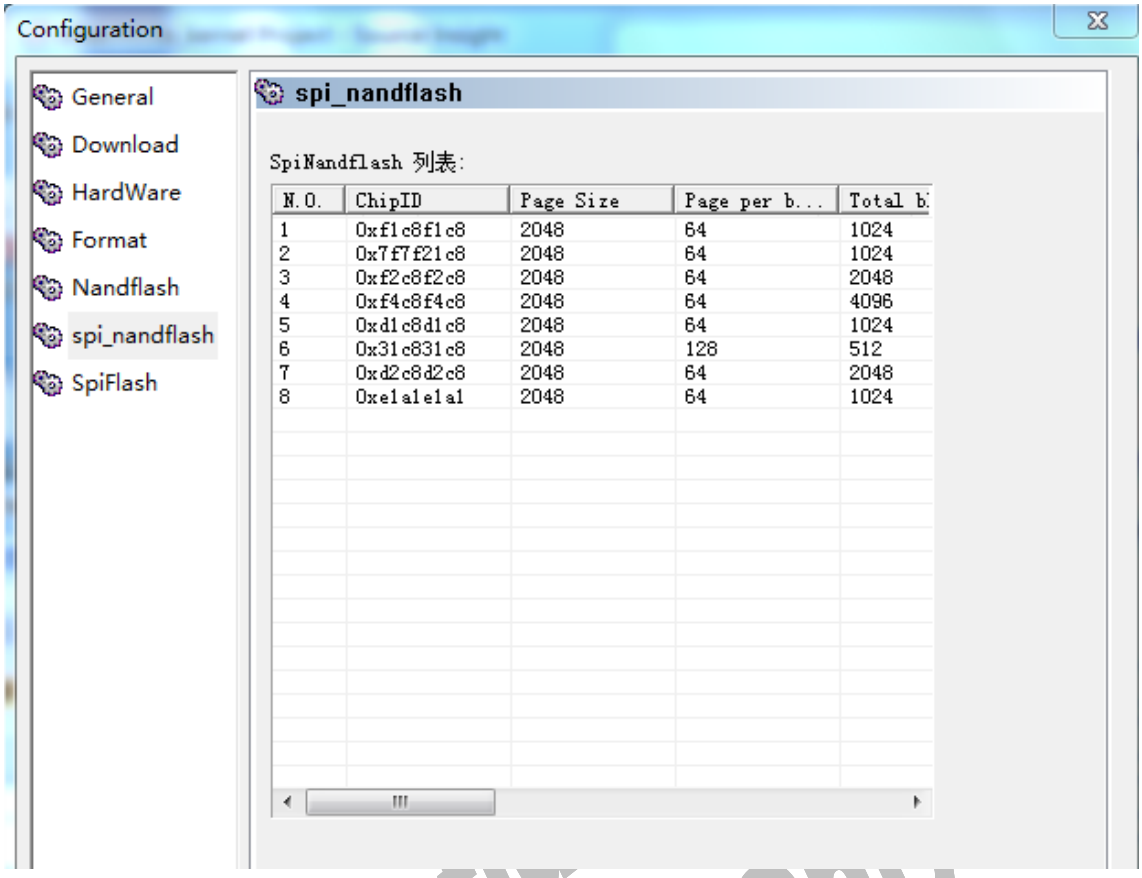

图 3-2 SPI NAND Flash 配置界面

注意：烧录工具中的 ID 与驱动中的 ID 相反，如驱动读取的 ID 为 0xc8f1c8f1 Burntool 里面填写的 ID 即为 0xf1c8f1c8。