Linux PWM
开发指南

版本号: 1.9

发布日期: 2025.7.29

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.7.27	XAA0248	初始版本
1.1	2022.11.28	XAA0309	修改文档适用范围并增加模块接口说明
1.2	2023.2.14	XAA0311	添加 dts 说明及调试接口及调试接口注意事项
1.3	2023.4.18	XAA0311	调整文档结构，并添加 capture 模式
1.4	2023.6.14	XAA0311	添加内核层的使用方法
1.5	2023.11.28	XAA0311	对 pwm 的适配命名进行修改
1.6	2024.10.22	AWA2215	添加 Linux-5.4 内核版本支持，调整文档结构
1.7	2024.11.25	AWA2215	添加 Linux-6.6 内核版本支持
1.8	2025.4.1	AWA2231	根据最新模块刷新文档，添加互补对输出调试方法
1.9	2025.7.29	AWA2351	优化文档描述和格式

目　录

插　图

1 前言

1.1 文档简介

介绍 PWM 模块的详细设计方便相关人员进行 PWM 模块的代码设计开发。

1.2 目标读者

PWM 驱动开发人员/维护人员等。

1.3 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-5.10	${SDK}/bsp/drivers/pwm/pwm-sunxi.c
Linux-5.15	${SDK}/bsp/drivers/pwm/pwm-sunxi.c
Linux-5.4	${SDK}/bsp/drivers/pwm/pwm-sunxi.c
Linux-6.6	${SDK}/bsp/drivers/pwm/pwm-sunxi.c

1.4 文档约定

1.4.1 标志说明

⚠ 注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-2: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

表 1-3: 数值单位约定

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

1.5 相关术语介绍

1.5.1 硬件术语

表 1-4: 硬件术语

术语	解释说明
PWM	脉冲宽度调制（PWM）, 是一种对模拟信号电平进行数字编码的方法。

1.5.2 软件术语

表 1-5: 软件术语

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台。
频率	PWM 的频率决定了所模拟电平的平滑度（逼真度），人耳感知的频率范围为 20Hz-16Khz，注意 PWM 的频率不要落在这个区间。
占空比	决定了一个周期内 PWM 信号高低的比例，进而决定了一个周期内的平均电压，也就是所模拟的电平的电压。
极性	决定了是高占空比的信号输出电平高，还是低占空比信号输出电平高。若信号的占空比为 100%，若为正常极性则输出电平最大，若为翻转极性则输出电平为 0。
开关	控制 PWM 信号是否输出。
PWM 对	电机等硬件需要两路脉冲信号来控制其正常运转，一般两路极性相关，频率，占空比参数相同的 PWM 构成一个 PWM 对。
PWM 死区控制时间	大功率电机，变频器等由大功率管，IGBT 等元件组成 H 桥或 3 相桥，每个桥的上半桥和下半桥是绝对不能导通的，在 PWM 信号驱动这些元件时，往往会由于没有延迟而造成未关断某路半桥，这样会造成功率元件的损坏，在 PWM 中加入死区时间的控制即是让上半桥关断后，自动插入一个事件，延迟后再打开下半桥。

2 模块介绍

- 脉冲宽度调制（PWM）是一种对模拟信号电平进行数字编码的方法。通过高分辨率计数器的使用，方波的占空比被调制用来对一个具体模拟信号的电平进行编码。
- PWM 模块属于 PWM 子系统，会调用 PWM 子系统的相关接口（详情可以查看 PWM 子系统知识）。

2.1 模块功能介绍

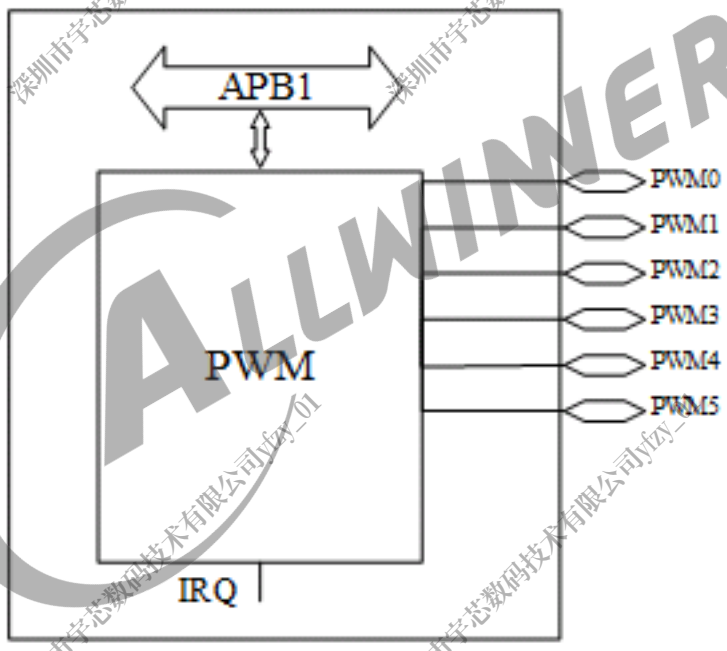

图 2-1: 模块功能

不同平台上拥有不同个数的 PWM 通道，其中两个为一个 PWM 对（平台通道数不相同，PWM 对也就不相同）。其中 PWM 具有以下特点：

- 支持脉冲，周期和互补对输出。
- 支出捕捉输入。
- 带可编程死区发生器，死区时间可控。
- 0-24M/100M 输出频率范围。0%-100% 占空比可调，最小分辨率为 1/65536。
- 支持 PWM 输出和捕捉输入产生中断。
- 组模式下各通道输出波形相对相位可配置。

2.2 模块配置介绍

2.2.1 kernel menuconfig 配置说明

在根目录中执行./build.sh menuconfig，选择 Allwinner BSP 选项进入下一级配置，如下图所示：

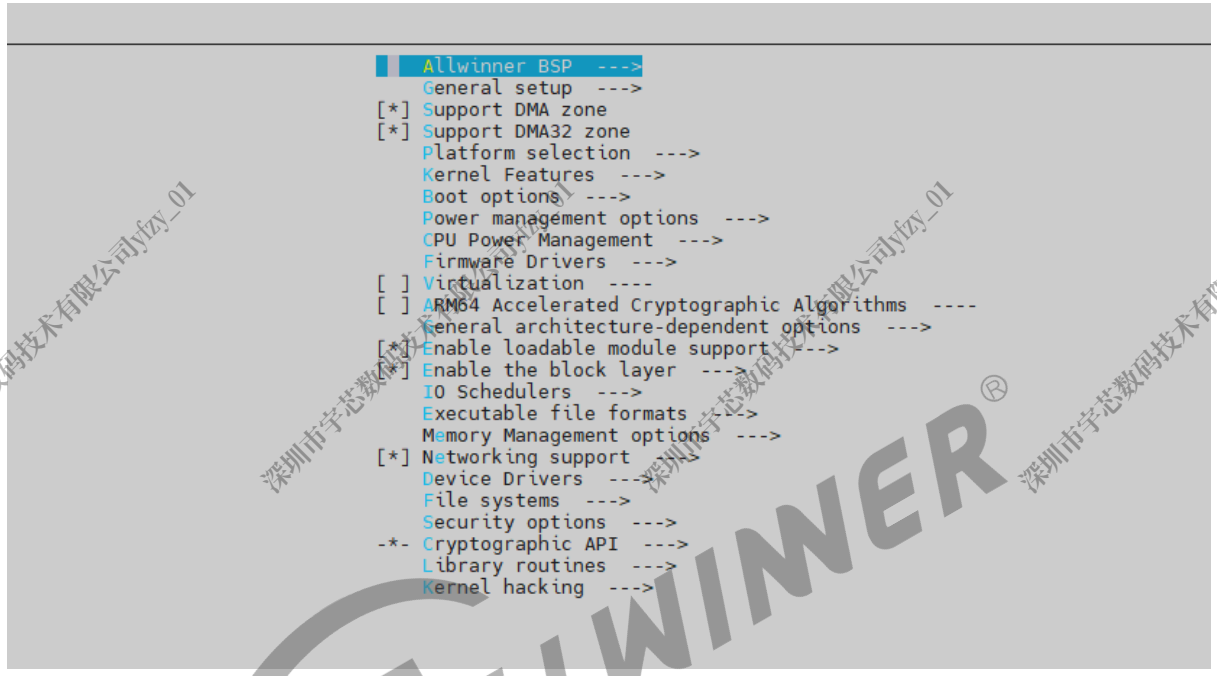

图 2-2: Allwinner BSP

选择 Device Drivers 选项进入下一级配置，如下图所示：

图 2-3: Device Drivers

选择 PWM Drivers 选项，进入下一级配置，如下图所示：

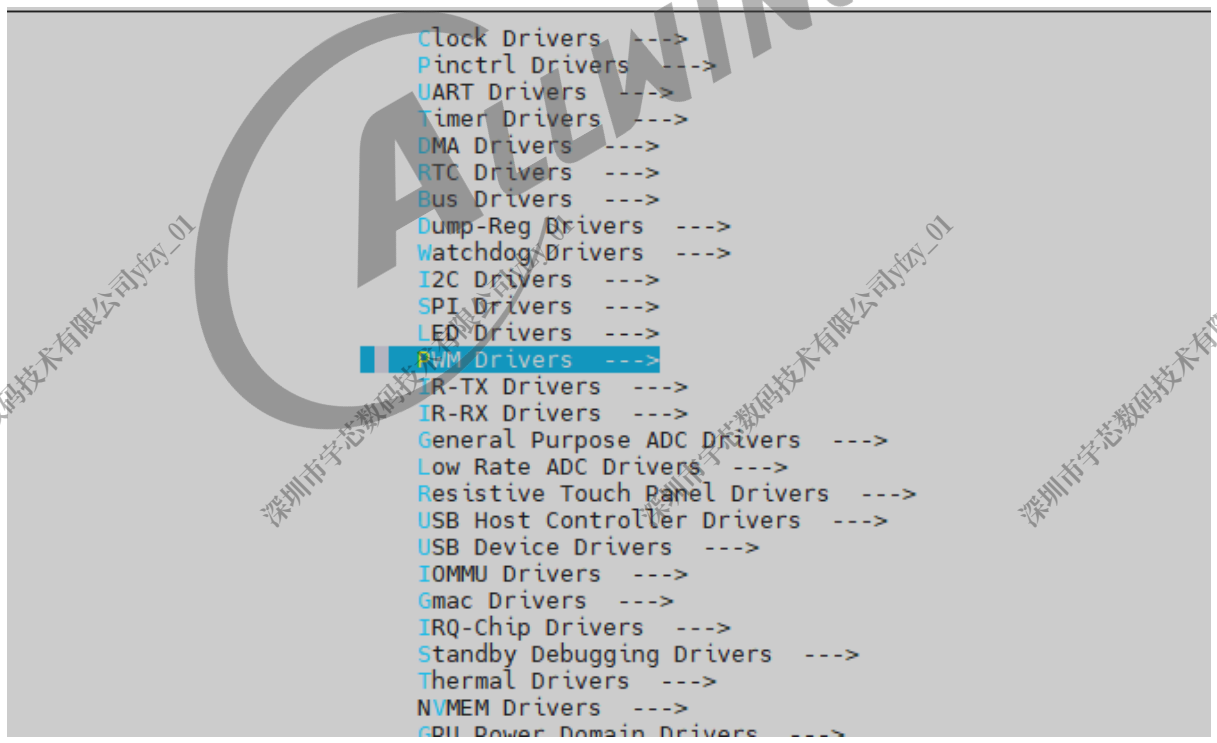

图 2-4: PWM Drivers

选择 PWM Support for Allwinner SoCs 选项，可选择直接编译进内核，也可编译成模块。如下图所示：

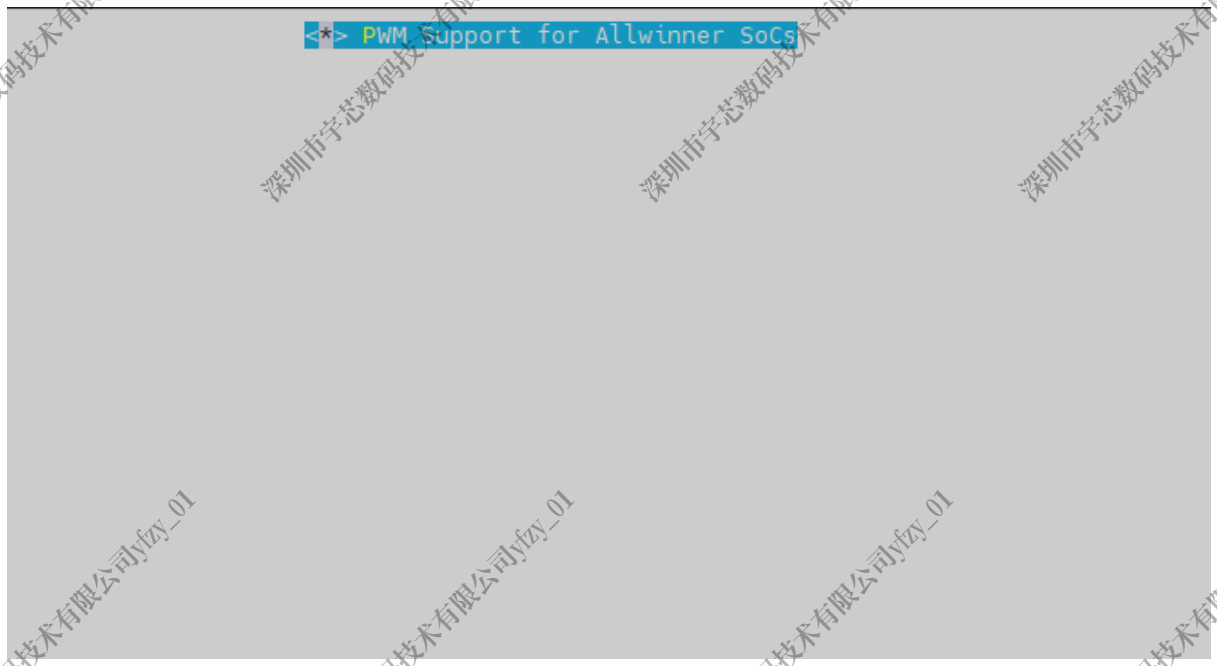

图 2-5: PWM Support for Allwinner SoCs

2.2.2 Device Tree 配置说明

- SoC 级设备树路径：

SDK/bsp/configs/{KERN_VER}/sun*.dtsi

以 pwm0 控制器为例，其配置方法如下：

```
pwm0: pwm0@2000c00 {
        #pwm-cells = <0x3>;
        compatible = "allwinner,sunxi-pwm-v201";
        reg = <0x0 0x02000c00 0x0 0x400>;
        clocks = <&ccu CLK_PWM>;
        interrupts = <GIC_SPI 19 IRQ_TYPE_LEVEL_HIGH>;
        resets = <&ccu RST_BUS_PWM>;
        pwm-number = <16>;
        pwm-base = <0x0>;
        sunxi-pwms = <&pwm0_0>, <&pwm0_1>, <&pwm0_2>, <&pwm0_3>, <&pwm0_4>,
      <&pwm0_5>, <&pwm0_6>, <&pwm0_7>, <&pwm0_8>, <&pwm0_9>,
        <&pwm0_10>, <&pwm0_11>, <&pwm0_12>, <&pwm0_13>,
        <&pwm0_14>, <&pwm0_15>;
        status = "okay";
      };

      pwm0_0: pwm0_0@2000c10 {
        compatible = "allwinner,sunxi-pwm0";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c10 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
      };
```

```
    pwm0_1: pwm0_1@2000c11 {
        compatible = "allwinner,sunxi-pwm1";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c11 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_2: pwm0_2@2000c12 {
        compatible = "allwinner,sunxi-pwm2";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c12 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_3: pwm0_3@2000c13 {
        compatible = "allwinner,sunxi-pwm3";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c13 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_4: pwm0_4@2000c14 {
        compatible = "allwinner,sunxi-pwm4";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c14 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_5: pwm0_5@2000c15 {
        compatible = "allwinner,sunxi-pwm5";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c15 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_6: pwm0_6@2000c16 {
        compatible = "allwinner,sunxi-pwm6";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c16 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_7: pwm0_7@2000c17 {
        compatible = "allwinner,sunxi-pwm7";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c17 0x0 0x4>;
        reg_base = <0x02000c00>;
        status = "disabled";
    };

    pwm0_8: pwm0_8@2000c18 {
        compatible = "allwinner,sunxi-pwm8";
        pinctrl-names = "active", "sleep";
        reg = <0x0 0x02000c18 0x0 0x4>;
```

```
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_9: pwm0_9@2000c19 {
    compatible = "allwinner,sunxi-pwm9";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c19 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_10: pwm0_10@2000c1a {
    compatible = "allwinner,sunxi-pwm10";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c1a 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_11: pwm0_11@2000c1b {
    compatible = "allwinner,sunxi-pwm11";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c1b 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_12: pwm0_12@2000c1c {
    compatible = "allwinner,sunxi-pwm12";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c1c 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_13: pwm0_13@2000c1d {
    compatible = "allwinner,sunxi-pwm13";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c1d 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_14: pwm0_14@2000c1e {
    compatible = "allwinner,sunxi-pwm14";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c1e 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  pwm0_15: pwm0_15@2000c1f {
    compatible = "allwinner,sunxi-pwm15";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c1f 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };
```

参数的具体配置如下：

- compatible = "allwinner,sunxi-pwm-v201";表示用哪套设备和驱动绑定
- reg = <0x0 0x02000c00 0x0 0x400>; 寄存器的基地址
- clocks = <&ccu CLK_PWM>;时钟配置
- interrupts = <GIC_SPI 19 IRQ_TYPE_LEVEL_HIGH>;中断号的配置
- resets = <&ccu RST_BUS_PWM>;复位配置
- pwm-number = <16>;控制器的 pwm 通道个数
- pwm-base = <0x0>;pwm 的起始基数
- sunxi-pwms = <&pwm0_0>, <&pwm0_1>, <&pwm0_2>, <&pwm0_3>, <&pwm0_4>, <&pwm0_5>, <&pwm0_6>, <&pwm0_7>, <&pwm0_8>, <&pwm0_9>, <&pwm0_10>, <&pwm0_11>, <&pwm0_12>, <&pwm0_13>, <&pwm0_14>, <&pwm0_15>; 控制器的具体通道
- 板级设备树 (board.dts) 路径：

SDK/device/config/chips/{IC}/configs/{BOARD}/{KERN_VER}/board.dts

在板级目录下的配置:

```
pwm0_3_pin_active: pwm0_3@0 {
    pins = "PB0";
    function = "pwm0_3";
};

pwm0_3_pin_sleep: pwm0_3@1 {
    pins = "PB0";
    function = "gpio_in";
    bias-pull-down;
};

pwm0_7_pin_active: pwm0_7@0 {
    pins = "PD22";
    function = "pwm0_7";
};

pwm0_7_pin_sleep: pwm0_7@1 {
    pins = "PD22";
    function = "gpio_in";
    bias-pull-down;
};

&pwm0_3 {
    pinctrl-names = "active", "sleep";
    pinctrl-0 = <&pwm0_3_pin_active>;
    pinctrl-1 = <&pwm0_3_pin_sleep>;
    status = "okay";
};

&pwm0_7 {
    pinctrl-names = "active", "sleep";
    pinctrl-0 = <&pwm0_7_pin_active>;
    pinctrl-1 = <&pwm0_7_pin_sleep>;
    status = "okay";
};
```

具体通道配置按照需求进行配置。

- pinctrl-names：分别表示 pwm 的 io 口的两种工作状态。
- pinctrl-0、pinctrl-1：pwm 两种工作状态的引脚参数设置。
- status: 模块设备的打开或关闭，当编译该模块时，status="okay";不编译时，status="disabled";。

2.3 源码模块结构

PWM 驱动的源代码位于 BSP 独立仓库的 drivers/pwm 目录下，具体的路径如下所示：

```
bsp/drivers/pwm/
├── pwm-sunxi.c  // Sunxi PWM对应的PWM驱动
├── pwm-sunxi.h
```

2.4 驱动框架

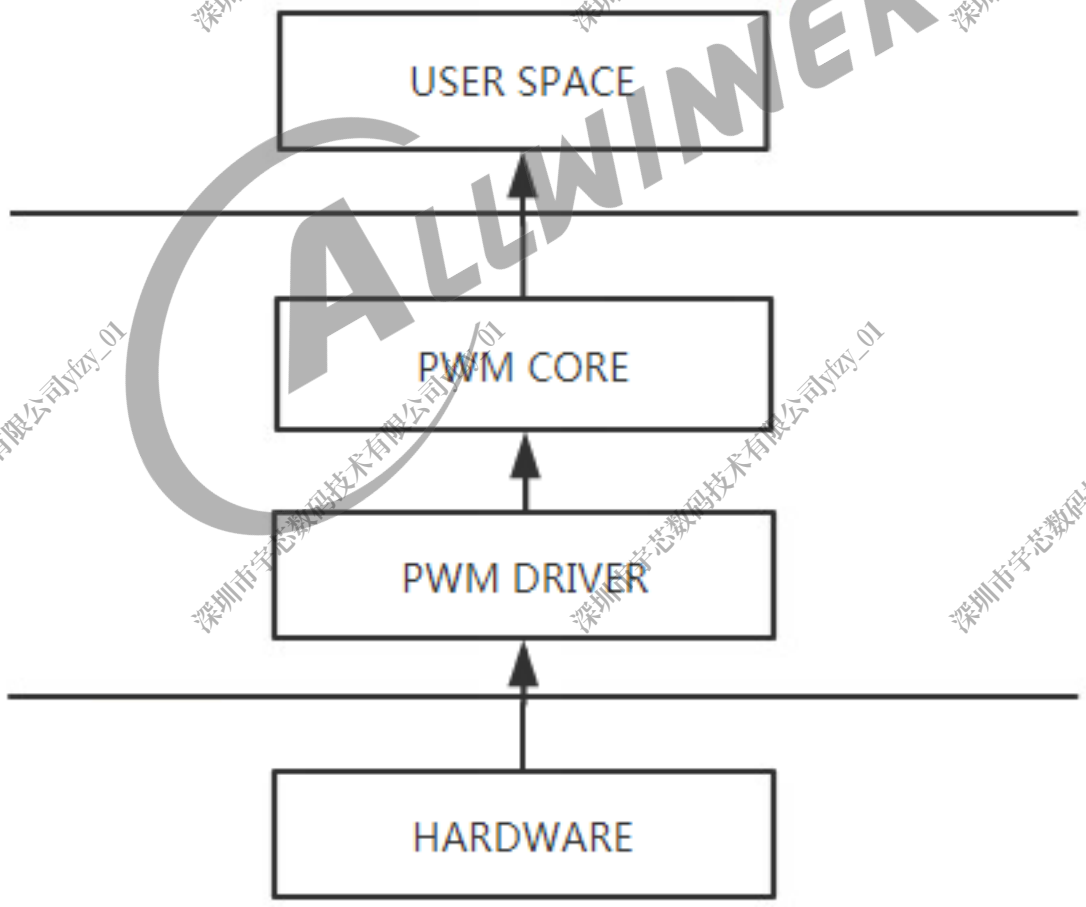

图 2-6: 软件框图

- PWM DRIVER: 负责 PWM 控制器的脉冲参数配置等，是编程人员需要实现的部分。

- PWM CORE：Linux 内核为 PWM 子系统提供的一套 API，用于完成设备和驱动的注册以及 PWM 的调用等。

3 模块接口说明

3.1 sunxi_pwm_config_channel

- 函数原型: static int sunxi_pwm_config_channel(struct pwm_chip *pwm_chip, struct pwm_device *pwm_device, int duty_ns, int period_ns)
- 作用: 配置 PWM 通道
- 参数:
  - pwm_chip: 指向 PWM 控制器
  - pwm_device: 指向当前 PWM 设备的实例
  - duty_ns: 占空比
  - period_ns: 周期
- 返回值:
  - 0: 完成配置

3.2 sunxi_pwm_enable

- 函数原型: static int sunxi_pwm_enable(struct pwm_chip *pwm_chip, struct pwm_device *pwm_device)
- 作用: 使能当前的 PWM 设备
- 参数:
  - pwm_chip: 指向 PWM 控制器
  - pwm_device: 指向当前 PWM 设备的实例
- 返回值:
  - 0: 成功
  - <0: 失败

3.3 sunxi_pwm_disable

- 函数原型: static void sunxi_pwm_disable(struct pwm_chip *pwm_chip, struct pwm_device *pwm_device)
- 作用：失能当前的 PWM 设备
- 参数:
  - pwm_chip: 指向 PWM 控制器
  - pwm_device: 指向当前 PWM 设备的实例
- 返回值:
  - void

3.4 sunxi_pwm_set_polarity

- 函数原型: static int sunxi_pwm_set_polarity(struct pwm_chip *pwm_chip, struct pwm_device *pwm, enum pwm_polarity polarity)
- 作用：设置当前 PWM 设备的极性
- 参数:
  - pwm_chip: 指向 PWM 控制器
  - pwm: 指向当前 PWM 设备的实例
  - polarity: 极性类别
- 返回值:
  - 0: 完成设置

3.5 sunxi_pwm_get_state

- 函数原型: static void sunxi_pwm_get_state(struct pwm_chip *pwm_chip, struct pwm_device *pwm_device, struct pwm_state *state)
- 作用：获取 PWM 状态
- 参数:
  - pwm_chip: 指向 PWM 控制器
  - pwm_device: 指向当前 PWM 设备的实例

  - state: 用于保存获取到的状态

- 返回值:

  - void

3.6 sunxi_pwm_capture

- 函数原型：static int sunxi_pwm_capture(struct pwm_chip pwm_chip, struct pwm_device pwm_device, struct pwm_capture *result, unsigned long timeout)
- 作用：设置捕获模式
- 参数：

  - pwm_chip：指向 PWM 控制器
  - pwm_device：指向当前 PWM 设备的实例
  - result：捕获的结果
  - timeout：超时时间

- 返回值：

  - 0：成功
  - 负值：失败

4 内核态 PWM 功能开发

4.1 功能概述

内核态 PWM 功能是在 Linux 内核空间下访问 PWM 设备，实现 PWM 的输出控制。

4.2 开发流程

步骤 1：调用 pwm_request 获取 pwm 设备。

```
struct pwm_device *pwm;
pwm = pwm_request(pwm_id, "my_pwm");
```

步骤 2：调用 pwm_config 设置 pwm 周期和占空比。

```
int duty_cycle = 50000;//占空比，单位为纳秒
int period = 100000;//周期，单位为纳秒
pwm_config(pwm, duty_cycle, period);
```

步骤 3：调用 pwm_enable/pwm_disable，使能/禁用 PWM。

使能 PWM 信号：

```
pwm_enable(pwm);
```

禁用 PWM 信号：

```
pwm_disable(pwm);
```

步骤 4：使用完毕后，释放 PWM 设备。

```
pwm_free(pwm);
```

4.3 注意事项

- 单位：PWM 的周期和占空比以纳秒为单位，配置时需要注意这些参数的值。
- 确保 PWM 设备使用完后调用 pwm_free 释放资源，避免内存泄漏。

4.4 编程示例

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/platform_device.h>
#include <linux/pwm.h>

static int sunxi_pwm_test_probe(struct platform_device *pdev)
{
   int ret;
   struct pwm_device *chip;
   struct device *dev = &pdev->dev;

   chip = devm_kzalloc(dev, sizeof(*chip), GFP_KERNEL);
   if (!chip)
      return -ENOMEM;

   chip= devm_pwm_get(dev, "pwm-test");
   if (IS_ERR(chip)) {
      dev_err(dev, "Cannot request PWM channel\n");
      return PTR_ERR(chip);
   }

   /* set pwm period and duty_cycle */
   ret = pwm_config(chip, 5000, 10000);
   if (ret) {
      dev_err(dev, "Cannot configure PWM\n");
      return ret;
   }

   /* Enable PWM */
   ret = pwm_enable(chip);
   if (ret) {
      dev_err(dev, "Cannot enable PWM\n");
      return ret;
   }

   platform_set_drvdata(pdev, chip);

   return 0;
}

static int sunxi_pwm_test_remove(struct platform_device *pdev)
{
   struct pwm_device *chip = platform_get_drvdata(pdev);

   pwm_disable(chip);
   pwm_free(chip);

   return 0;
}

static const struct of_device_id sunxi_pwm_test_of_match[] = {
   { .compatible = "sunxi-pwm-test", },
   {},
};
MODULE_DEVICE_TABLE(of, sunxi_pwm_test_of_match);
```

```
static struct platform_driver sunxi_pwm_test_driver = {
    .driver = {
        .name = "sunxi-pwm-test",
        .owner = THIS_MODULE,
        .of_match_table = sunxi_pwm_test_of_match,
    },
    .probe = sunxi_pwm_test_probe,
    .remove = sunxi_pwm_test_remove,
};

module_platform_driver(sunxi_pwm_test_driver);

MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("yourname <yourname@allwinnertech.com>");
MODULE_DESCRIPTION("sunxi-pwm test driver");
MODULE_VERSION("1.0.0");
```

5 调试方法

5.1 用户层调试接口

可以直接在 linux 内核中调试 pwm 模块，具体如下：

- 进入/sys/class/pwm 目录。该目录是 linux 内核为 pwm 子系统提供的类目录，遍历该目录。可看到很多的 pwmchipX，其中 X 表示很多能够被控制的通道，下面以 pwmchip0 为例进行说明。

```
/sys/class/pwm # ls
pwmchip0
```

- 可以看到，上述 pwmchip0 就是我们注册的 pwm 控制器，进入该目录，然后遍历该目录。

```
/sys/class/pwm # cd pwmchip0/
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # ls
device    export    npwm      subsystem  uevent    unexport
```

- 其中 npwm 文件储存了该 pwm 控制器的 pwm 个数，而 export 和 unexport 是导出和删除某个 pwm 设备的文件，下面演示导出 pwm1。

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # cat npwm
2
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # echo 1 > export
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # ls
device    export    npwm      pwm1      subsystem  uevent    unexport
```

可以看到目录中多出 pwm1 目录，进入该目录显示出不同的节点。不同节点的参数如下：

- enable：使能 pwm，其中 1 代表使能，0 代表不使能。
- duty_cycle：pwm 信号的占空比，单位为（ns）。
- period：pwm 信号的频率，单位为（ns）。
- polarity：是否翻转极性，其中 1 表示翻转极性，0 表示不翻转极性。

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # cd pwm1/
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm1 # ls
capture    duty_cycle enable     period     polarity   uevent
```

可通过以上节点来对 pwm 的状态进行改变：

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm1 # echo 1000000000 > period
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm1 # echo 500000000 > duty_cycle
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm1 # echo normal > polarity
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm1 # echo 1 > enable
```

如果相关引脚接上了示波器等，可以看到波形。最后返回上层目录，删除该 pwm 设备。

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm1 # cd ..
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # ls
device    export    npwm     pwm1     subsystem  uevent    unexport
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # echo 1 > unexport
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # ls
device    export    npwm     subsystem  uevent    unexport
```

5.2 内核层调试接口

内核调用的接口如下

```
devm_pwm_get(struct device *dev, const char *con_id)  //请求pwm设备
pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns)  //设置频率和占空比
pwm_enable(struct pwm_device *pwm)  //启动PWM输出
```

5.2.1 devm_pwm_get

- 函数原型: struct pwm_device devm_pwm_get(struct device dev, const char *con_id)
- 作用: 请求 pwm 设备
- 参数:
  - dev: 指向当前 PWM 设备的实例
  - con_id: 使用方绑定的名称
- 返回值:
  - 成功：返回请求成功的 pwm 设备
  - 失败：返回错误码

5.2.2 pwm_config

- 函数原型：static inline int pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns)
- 作用: 更改 PWM 设备配置
- 参数:
  - dev: 指向当前 PWM 设备的实例
  - duty_ns: 占空比

- period_ns：周期

- 返回值:
  - 成功：0
  - 失败：负值错误码

5.2.3 pwm_enable

- 函数原型：static inline int pwm_enable(struct pwm_device *pwm)
- 作用: 启动 PWM 输出
- 参数:
  - pwm: 指向当前 PWM 设备的实例
- 返回值:
  - 成功：0
  - 失败：负值错误码

5.2.4 SoC 配置

```
soc {
    pwm0: pwm0@2000c00 {
    #pwm-cells = <0x3>;
    compatible = "allwinner,sunxi-pwm-v201";
    reg = <0x0 0x02000c00 0x0 0x400>;
    clocks = <&ccu CLK_PWM>;
    resets = <&ccu RST_BUS_PWM>;
    pwm-number = <16>;
    pwm-base = <0x0>;
    sunxi-pwms = <&pwm0_0>, <&pwm0_1>, <&pwm0_2>, <&pwm0_3>, <&pwm0_4>,
    <&pwm0_5>, <&pwm0_6>, <&pwm0_7>, <&pwm0_8>, <&pwm0_9>,
    <&pwm0_10>, <&pwm0_11>, <&pwm0_12>, <&pwm0_13>,
    <&pwm0_14>, <&pwm0_15>;
    status = "disabled";
  };

  pwm0_0: pwm0_0@2000c10 {
    compatible = "allwinner,sunxi-pwm0";
    pinctrl-names = "active", "sleep";
    reg = <0x0 0x02000c10 0x0 0x4>;
    reg_base = <0x02000c00>;
    status = "disabled";
  };

  sunxi_pwm_test: sunxi-pwm-test {
      compatible = "sunxi-pwm-test";
      #pwm-cells = <3>;
```

```
        pwms = <&pwm0 0 500000 1000000 0>;
        pwm-names = "pwm-test";
        status = "disabled";
};
```

- pwms = <&pwm0 0 500000 1000000 0>：**&a_pwm 表示要引用的 PWM 的控制器（根据需求可引用对应的引脚）；**0 表示对应 pwm0_0 下的具体通道；500000 表示占空比；1000000 表示周期；0 表示极性。
- pwm-names = “pwm-test”；表示索引具体的 pwm 通道名称。其名称与 devm_pwm_get(dev, “pwm-test”) 的第二个参数保持一致。

5.2.5 board 配置

```
&pio {
    pwm0_0_pin_active: pwm0_0@0 {
      pins = "PD23";
    };
    pwm0_0_pin_sleep: pwm0_0@1 {
      pins = "PD23";
      function = "gpio_in";
      bias-pull-down
    };
};
&pwm0_0 {
      pinctrl-names = "active", "sleep";
      pinctrl-0 = <&pwm0_0_pin_active>;
      pinctrl-1 = <&pwm0_0_pin_sleep>;
      status = "okay";
    };
```

测试方法：将该驱动编译进内核后，通过示波器测量对应引脚的波形。

测试结果：

如 demo 中所示，输入占空比为：5000；周期为：10000，其波形如下所示：

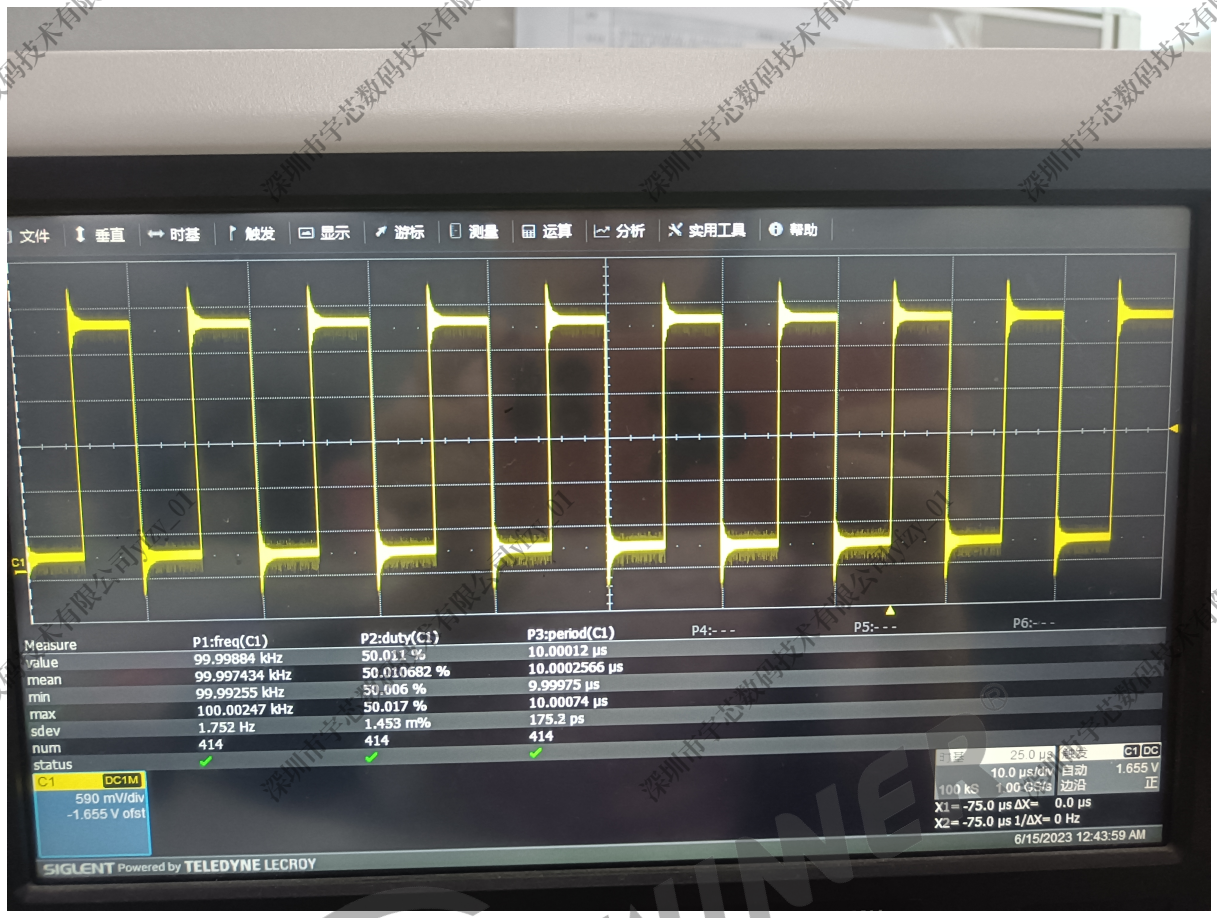

图 5-1: 内核调用的测试结果

5.3 捕获调试接口

捕获模式原理：将两个引脚连接起来，对 A 引脚设置参数，用 B 引脚去捕获 A 的参数。

测试方法：用杜邦线将两个具有 PWM 功能的引脚连接起来，在一个引脚上设置需要的频率及占空比, 在另一个引脚的下读取 capture。

- 根据普通调试接口的方法, 对其中一个引脚设置频率及占空比。

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # echo 0 > export
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # cd pwm0/
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo 20000 > period
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo 5000 > duty_cycle
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo normal > polarity
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo 1 > enable
```

- 采用 capture 模式在另一个引脚上捕获频率及占空比。

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # echo 3 > export
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # cd pwm3/
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm3 # cat capture
```

测试结果：

- capture 的 log 如下

```
/sys/devices/platform/soc@3000000/2000c00.a_pwm/pwm/pwmchip0 # echo 3 > export
/sys/devices/platform/soc@3000000/2000c00.a_pwm/pwm/pwmchip0 # cd pwm3/
/sys/devices/platform/soc@3000000/2000c00.a_pwm/pwm/pwmchip0/pwm3 # ls
capture     enable      polarity    uevent
duty_cycle  period      power
/sys/devices/platform/soc@3000000/2000c00.a_pwm/pwm/pwmchip0/pwm3 # cat capture
19916 4958
```

图 5-2: capture 的 log

- 在另一个引脚测试的波形如下

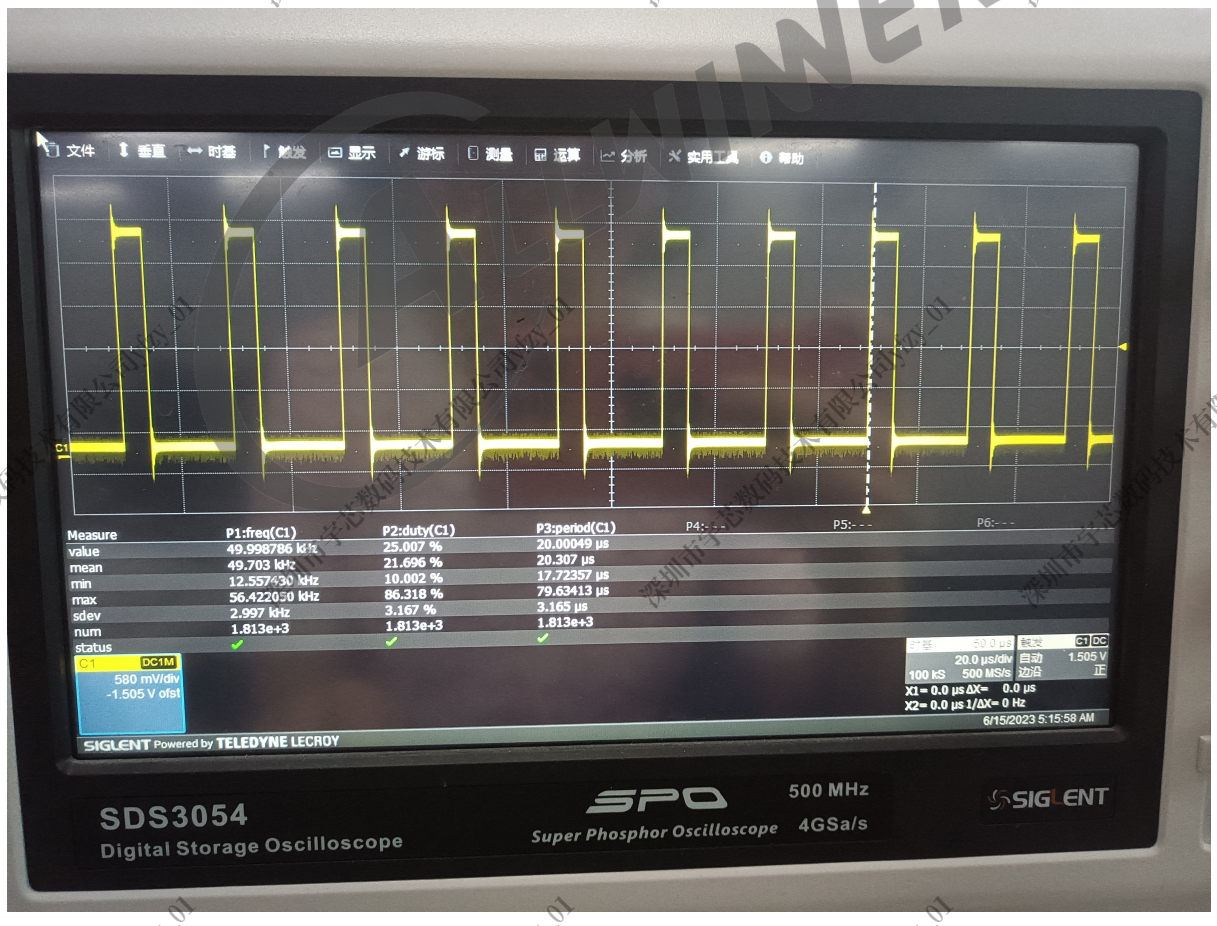

图 5-3: capture 的波形

5.4 互补对输出调试接口

5.4.1 board 配置

```
&pio {
    pwm0_0_pin_active: pwm0_0@0 {
        pins = "PB0";
        function = "pwm0_0";
    };

    pwm0_0_pin_sleep: pwm0_0@1 {
        pins = "PB0";
        function = "gpio_in";
        bias-pull-down;
    };

    pwm0_1_pin_active: pwm0_1@0 {
        pins = "PB1";
        function = "pwm0_1";
    };

    pwm0_1_pin_sleep: pwm0_1@1 {
        pins = "PB1";
        function = "gpio_in";
        bias-pull-down;
    };
};

&pwm0_0 {
        pinctrl-names = "active", "sleep";
        pinctrl-0 = <&pwm0_0_pin_active>;
        pinctrl-1 = <&pwm0_0_pin_sleep>;
        bind_pwm = <1>;    //设置互补模式pwm通道
        status = "okay";
};

&pwm0_1 {
        pinctrl-names = "active", "sleep";
        pinctrl-0 = <&pwm0_1_pin_active>;
        pinctrl-1 = <&pwm0_1_pin_sleep>;
        bind_pwm = <0>;       //设置互补模式pwm通道
        dead_time = <2500>;   //设置死区时间,其中死区时间不得超过设置的pwm占空比
        status = "okay";
};
```

5.4.2 用户层调试接口

互补对输出通过配置 pwm0_0 和 pwm0_1 为互补对，只需操作 pwm0_0，即可输出互补对波形

测试方法：根据普通调试接口的方法, 对其中一个引脚设置频率及占空比。

```
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # echo 0 > export
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0 # cd pwm0/
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo 20000 > period
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo 5000 > duty_cycle
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo normal > polarity
/sys/devices/platform/soc/1c23400.pwm/pwm/pwmchip0/pwm0 # echo 1 > enable
```

每一个 pwm 都支持 pwm 对波形输出，pwm 对波形输出包括互补输出合插入死区时间的 PWM 对输出。

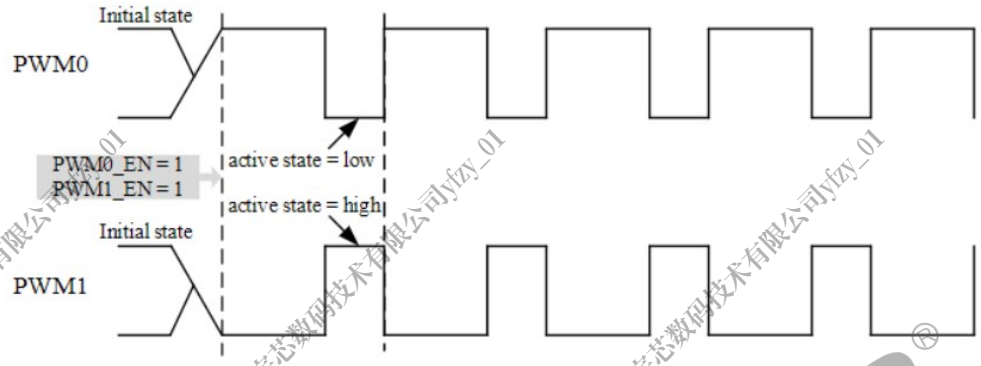

Figure 2-6 PWM01 对互补对输出

图 5-4: 互补输出

PWM01 实现互补输出需要满足的条件:

- PWM0 和 PWM1 通道的时钟分频、频率、占空比、相位一致。
- PWM0 和 PWM1 通道的极性相反。
- 同时使能 PWM0 和 PWM1 的时钟 gating，然后同时使能通道输出。

5.4.3 死区控制输出

PWM01_DZ_EN =1 后，PWM01 对的死区功能使能，PWM01 对输出一对插入死区时间的 PWM 波形，PWM01 对输出的波形由 PWM0 定时器逻辑模块和 Dead Zone Generator 01 决定。

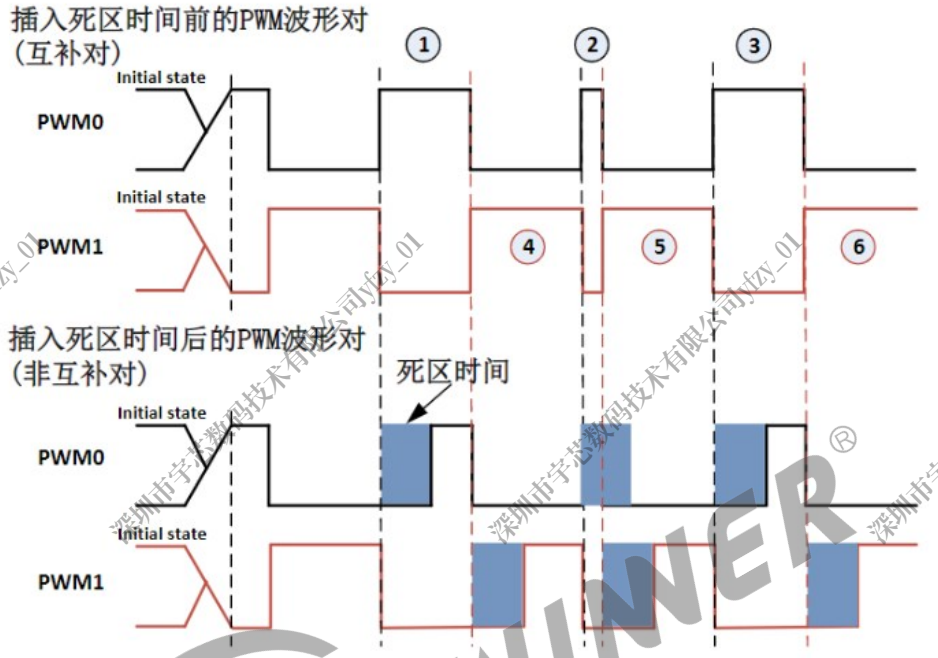

图 5-5: 死区控制

插入死区的原则是：**凡遇到上升沿就插入一段死区时间**

6 FAQ

6.1 AndroidT_A523_PWM_ 概率性出现息屏背光

6.1.1 问题现象

在 A523 平板方案验证中，概率性出现灭屏后，背光依旧存在的现象。

6.1.2 问题分析

- 执行/sys/class/pwm/pwmchip0 和 echo 0 > export 会得到 pwm0 设备的，则说明 **disp** 已经 disable 掉 pwm0，因为如果显示没有把 pwm0 disable，echo 0 > export 是不会得到 pwm0 设备的；但是背光依然存在，查看 PD23 发现 pin 脚还是 pwm，由此可推测问题出在 pwm 设置 pin 时出现了问题。
- 对比情况和异常情况下 PWM 引脚状态的 log，发现异常情况下 pin 先于 disp 加载，pwm 使用出错, 由此可证明步骤 1 结论的正确性。正常情况下 disp 是在 pinctrl 之后加载的。

```
[    5.158248][    T7] cpufreq: cpufreq_online: CPU0: Running at unlisted initial frequency: 1008000 KHz, changing to: 1104000 KHz
[    5.174870][    T7] cpufreq: cpufreq_online: CPU4: Running at unlisted initial frequency: 1008000 KHz, changing to: 1440000 KHz
[    5.195176][    T7] sun55iw3-pinctrl 2000000.pinctrl: initialized sunXi PIO driver
[    5.205230][    T7] disp 5000000.disp: Adding to iommu group 0
[    5.211563][    T7] clk: failed to reparent bus tcontv to pll_video0_4x:  -22
[    5.219669][    T7] [DISP WRN]: [disp_init:2523]:                  smooth display screen:0 type:1 mode:4
[    5.230437][    T7] [DISP WRN]: [gamma_init:1677]:                 alloc gamma[0] mm size=0x1100
[    5.240428][    T7] [DISP WRN]: [gamma_init:1677]:                 alloc gamma[1] mm size=0x1100
[    5.250521][    T7] [DISP WRN]: [de_wb_init:0775]:                 not support more than 1 wb at present
[    5.261273][    T7] [DISP WRN]: [de_gamma_get_reg_blocks:1738]:    priv->reg_blk_num =4
[    5.271629][    T7] [DISP WRN]: [de_smbl_get_reg_blocks:0224]:     priv->reg_blk_num=6
[    5.281807][    T7] [DISP WRN]: [de_gamma_get_reg_blocks:1751]:    blk_num =4
[    5.291242][    T7] [DISP WRN]: [de_smbl_get_reg_blocks:0238]:     get blk_num=6, 0
[    5.301150][    T7] [DISP WRN]: [de_gamma_get_reg_blocks:1738]:    priv->reg_blk_num =4
[    5.311502][    T7] [DISP WRN]: [de_smbl_get_reg_blocks:0224]:     priv->reg_blk_num=6
[    5.321665][    T7] [DISP WRN]: [de_wb_get_reg_blocks:0868]:       not support more than 1 wb at present
[    5.333346][    T7] [DISP WRN]: [de_gamma_get_reg_blocks:1751]:    blk_num =4
[    5.343054][    T7] [DISP WRN]: [de_smbl_get_reg_blocks:0238]:     get blk_num=6, 1
```

图 6-1: disp 加载在 pinctrl 之后

异常的情况下 disp 会加载在 pinctrl 之前, 然后 pwm 使能引脚的时候会报错, 之后就导致异常了。

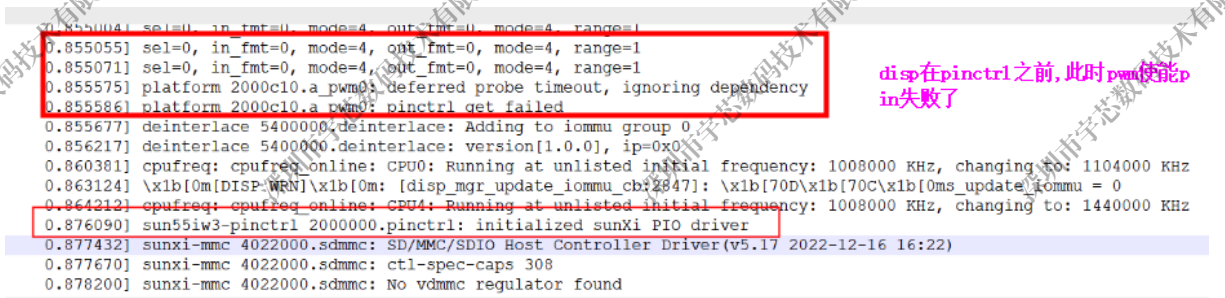

图 6-2: disp 加载在 pinctrl 之前

6.1.3 根本原因

android 各个 ko 并行化加载，pwm、pin 和 disp 之间并没有严格的依赖关系，对比正常与异常的 log，可看出会极小概率出现 pin 加载未完成，disp 却使用 pwm 导致出错问题。

6.1.4 解决办法

可以通过 board.dts 约束依赖关系，在 disp 根节点下添加对 pin 驱动的引用。

```
&disp {
    ..........................
    pwms = <&a_pwm 0 5000000 1>;
    pinctrl-names = "active", "sleep";
    pinctrl-0 = <&a_pwm0_pin_active>;
    pinctrl-1 = <&a_pwm0_pin_sleep>;
};
```

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。