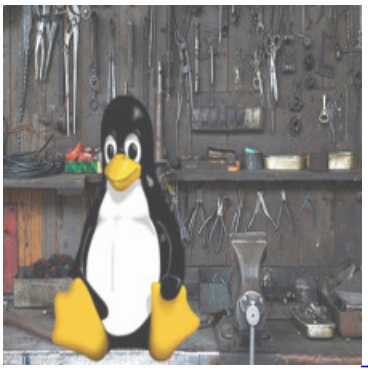
Linux misc设备（二）蜂鸣器驱动

原创

JT同学 于 2019-08-23 16:51:43 发布 868 收藏 7

分类专栏： Linux驱动 文章标签： Linux 驱动 蜂鸣器 misc

本文链接：https://blog.csdn.net/weixin_42462202/article/details/100042085



Linux misc设备驱动

Linux misc设备（二）蜂鸣器驱动

文章目录

本文将介绍如何基于misc框架写一个蜂鸣器驱动程序

一、注册misc设备

首先先基于misc驱动框架注册一个misc设备

```c
static int buzzer_open(struct inode *inode, struct file *file)
{
    return 0;
}


static int buzzer_close(struct inode *inode, struct file *file)
{
 return 0;
}

static int buzzer_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
    return 0;
}


static struct file_operations buzzer_fops = {
    .owner   =   THIS_MODULE,
    .open    =   buzzer_open,
    .release =   buzzer_close,
    .ioctl   =   buzzer_ioctl,
};

static struct miscdevice buzzer_dev = {
 .minor  = MISC_DYNAMIC_MINOR, //动态分配次设备号
 .name   = "buzzer",
 .fops   = &buzzer_fops, //文件操作集
};

static int __init buzzer_init(void)
{
    /* 注册杂项设备 */
 misc_register(&buzzer_dev);

 return 0;
}

static void __exit buzzer_exit(void)
{
    /* 注销杂项设备 */
 misc_deregister(&buzzer_dev);
}

module_init(buzzer_init);
module_exit(buzzer_exit);
MODULE_LICENSE("GPL");
```

编译该驱动程序后，加载模块，就会生成`/dev/buzzer`设备节点

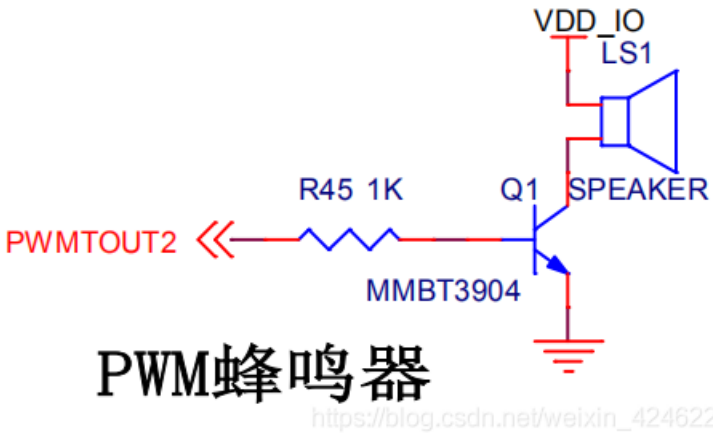
我们要实现这个驱动程序的功能是可以通过ioctl来控制蜂鸣器的频率

那么就需要在`buzzer_ioctl`里去操作硬件，下面来讲一讲如何去操控蜂鸣器

二、硬件操作

此驱动程序的芯片是三星`S5PV210`，硬件平台是`X210`

首先打开原理图，找到蜂鸣器



从原理图中可以看到蜂鸣器接在`PWMTOUT2`引脚处，表明该引脚可以用于PWM输出

只要该引脚为高电平，蜂鸣器就会响，那么通过PWM就可以控制蜂鸣器的频率

再看一看`PWMTOUT2`接到芯片的哪一个引脚

该GPIO为GPD0_2

要使该GPIO输出PWM，第一步就是设置该GPIO的功能，第二步就是设置PWM相关的寄存器

我们先看第一部分，打开datasheet，查找该GPIO相关的寄存器

2.1 GPIO设置

2.2.7.1 Port Group GPD0 Control Register (GPD0CON, R/W, Address = 0xE020_00A0)

GPD0CON	Bit	Description	Initial State
GPD0CON[3]	[15:12]	0000 = Input 0001 = Output 0010 = TOUT_3 0011 ~ 1110 = Reserved 1111 = GPD0_INT[3]	0000
GPD0CON[2]	[11:8]	0000 = Input 0001 = Output 0010 = TOUT_2 0011 ~ 1110 = Reserved 1111 = GPD0_INT[2]	0000
GPD0CON[1]	[7:4]	0000 = Input 0001 = Output 0010 = TOUT_1 0011 ~ 1110 = Reserved 1111 = GPD0_INT[1]	0000
GPD0CON[0]	[3:0]	0000 = Input 0001 = Output 0010 = TOUT_0 0011 ~ 1110 = Reserved 1111 = GPD0_INT[0]	0000

`GPD0CON`为GPD0组的GPIO的功能配置寄存器，其地址为`0xE020_00A0`，其中GPD0CON[2]表示GPD0_2的配置

配置中的`TOUT_2`表示PWM输出功能，那么要配置GPD0_2为PWM输出功能，可以这样做

```
GPD0CON |= 0x2<<4;
```

下面继续来介绍PWM相关的设置

从datasheet中可以得知，S5PV210有5个PWM定时器，其中定时器0、1、2、3有PWM输出引脚，定时器4没有PWM输出引脚

从我们上述的原理图可以看出，我们使用的是PWM定时器2，其对应输出引脚为GPD0_2

2.2 PWM定时器的时钟设置

这些定时器的时钟源为APB-PCLK，定时器0和1共享一个8位分频器，定时器2、3、4共享一个8位分频器，每个定时器还有二级的时钟分频器，如下图所示

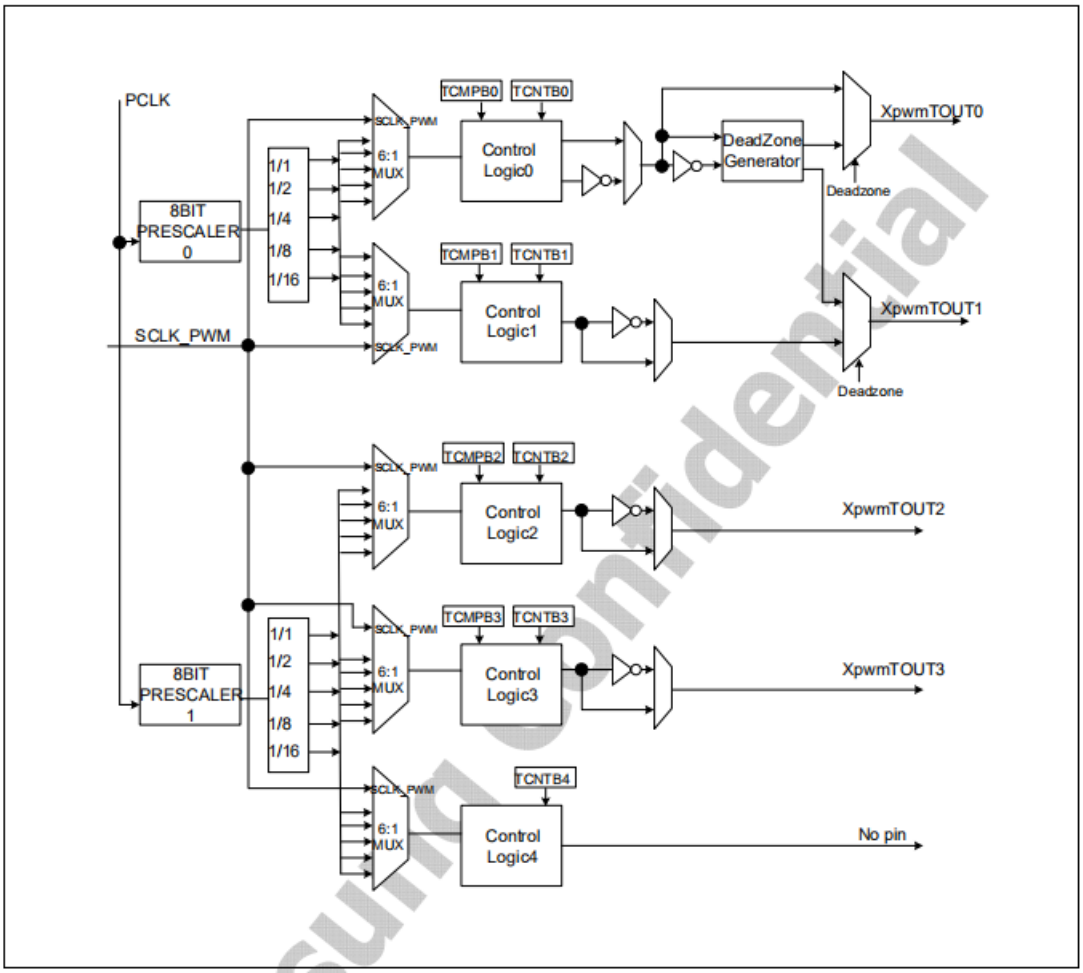


Figure 1-2 PWM TIMER Clock Tree Diagram

也就是说，PWM的定时器的时钟源是`APB-PCLK`经过1级8位分频器分频，再经过2级复用器分频得来

查看相应的寄存器

1.5.1.1 Timer Configuration Register (TCFG0, R/W, Address = 0xE250_0000)

Timer Input Clock Frequency = PCLK / ({prescaler value + 1}) / {divider value}
{prescaler value} = 1~255
{divider value} = 1, 2, 4, 8, 16, TCLK
Dead zone length = 0~254

TCFG0	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Dead zone length	[23:16]	Dead zone length	0x00
Prescaler 1	[15:8]	Prescaler 1 value for Timer 2, 3 and 4	0x01
Prescaler 0	[7:0]	Prescaler 0 value for timer 0 and 1	0x01

1.5.1.2 Timer Configuration Register (TCFG1, R/W, Address = 0xE250_0004)

TCFG1	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Divider MUX4	[19:16]	Selects Mux input for PWM Timer 4 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX3	[15:12]	Selects Mux input for PWM Timer 3 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX2	[11:8]	Selects Mux input for PWM Timer 2 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX1	[7:4]	Selects Mux input for PWM Timer 1 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX0	[3:0]	Selects Mux input for PWM Timer 0 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00

其中`TCFG0`用于设置1级分频器倍数，`TCFG1`用于设置2级复用器

PWM定时器的频率可以这样计算

```
Frequency = PCLK / (一级分频 + 1) / (二级分频)
```

如果要设置PWM定时器2的频率，可以这样做

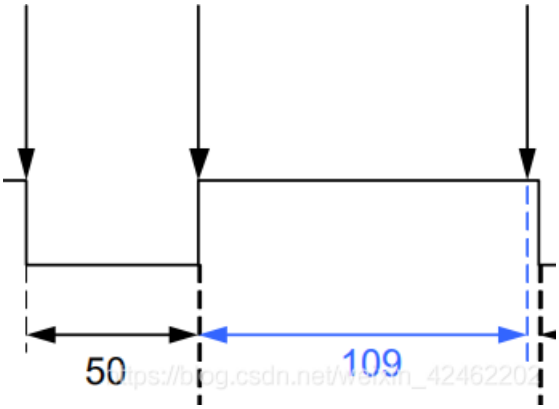
```
TCFG0 |= prescaler<<8;
TCFG1 |= mux<<8;
```

2.3 PWM定时器设置

每个PWM定时器都有一个32位的向下计数器

需要设置计数周期，PWM电平触发时间，然后手动更新计数周期（使其加载到定时器的计数器中）

需要设置自动装载，在每次定时器减到0后，会自动装载计数周期



图中所示，计数周期为159(50+109)，PWM电平触发时间时间为109，表示每次加载到定时器的计数器的起始值为159，当计数器减到109时，触发电平装换

查看datasheet中描述寄存器

TCNTBn为计数周期的缓存寄存器

1.5.1.10 Timer2 Counter Register (TCNTB2, R/W, Address = 0xE250_0024)

TCNTB2	Bit	Description	Initial State
Timer 2 Count Buffer	[31:0]	Timer 2 Count Buffer Register	0x0000_0000

TCMPBn为比较寄存器，描述PWM电平转换时刻

1.5.1.11 Timer2 Compare Register (TCMPB2, R/W, Address = 0xE250_0028)

TCMPB2	Bit	Description	Initial State
Timer 2 Compare Buffer	[31:0]	Timer 2 Compare Buffer Register	0x0000_0000

TCON控制PWM定时器，设置自动重装载，手动更新计数周期，启动定时器

1.5.1.3 Timer Control Register (CON, R/W, Address = 0xE250_0008)

TCON	Bit	Description	Initial State
Reserved	[31:23]	Reserved Bits	0x000
Timer 4 Auto Reload on/off	[22]	0 = One-Shot 1 = Interval Mode(Auto-Reload)	0x0
Timer 4 Manual Update	[21]	0 = No Operation 1 = Update TCNTB4	0x0
Timer 4 Start/Stop	[20]	0 = Stop 1 = Start Timer 4	0x0
Timer 3 Auto Reload on/off	[19]	0 = One-Shot 1 = Interval Mode(Auto-Reload)	0x0
Timer 3 Output Inverter on/off	[18]	0 = Inverter Off	0x0

设置PWM定时器的步骤

- 设置计数周期缓存寄存器TCNTBn
- 设置比较器TCMPBn
- 通过置位TCON中的手动更新TCMPBn的位，来加载TCNBn的值到PWM定时器的计数器中
- 通过置位TCON中相应的位来开启自动重装载
- 复位TCON中的手动更新TCMPBn的位
- 通过设置TCON寄存器打开定时器

总结一下：pwm的设置总共三部分，第一设置GPIO功能，第二设置PWM定时器的时钟源，第三设置PWM定时器

三、源码

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <linux/clk.h>
#include <linux/miscdevice.h>
#include <linux/gpio.h>
#include <plat/gpio-cfg.h>

#define TCFG0   0xE2500000
#define TCFG1   0xE2500004
#define TCON    0xE2500008
#define TCNTB2  0xE2500024
#define TCMPB2  0xE2500028

#define PWM_IOCTL_SET_FREQ  1
#define PWM_IOCTL_STOP   0

static volatile unsigned int *tcfg0;
static volatile unsigned int *tcfg1;
static volatile unsigned int *tcon;
static volatile unsigned int *tcntb2;
static volatile unsigned int *tcmpb2;

static void pwm_set_freq( unsigned long freq)
{
    unsigned long cfg;
 struct clk *clk_p;
 unsigned long pclk;

 /* 设置GPD0_2为PWM输出 */
 s3c_gpio_cfgpin(S5PV210_GPD0(2), S3C_GPIO_SFN(2));

 clk_p = clk_get(NULL, "pclk");
 pclk  = clk_get_rate(clk_p);

    /* 计算周期 */
 cfg  = (pclk/16/16)/freq;
    writel(cfg, tcntb2);
    writel(cfg/2, tcmpb2); //占空比50%
```

```
    /* 设置并启动定时器 */
    cfg = readl(tcon);
 cfg &= ~(0xf<<12);
 cfg |= (0xb<<12); //取消死区，使能自动装载，置位手动更新，开启定时器
 writel(cfg, tcon);

    /* 复位手动更新位 */
 cfg &= ~(2<<12);
 writel(cfg, tcon);
}

void pwm_stop( void )
{
 s3c_gpio_cfgpin(S5PV210_GPD0(2), S3C_GPIO_INPUT);
}

static int buzzer_open(struct inode *inode, struct file *file)
{
    return 0;
}


static int buzzer_close(struct inode *inode, struct file *file)
{
 return 0;
}

static int buzzer_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
 switch (cmd)
 {
  case PWM_IOCTL_SET_FREQ:
            pwm_set_freq(arg);
   break;

  case PWM_IOCTL_STOP:
  default:
            pwm_stop();
   break;
 }

 return 0;
}


static struct file_operations buzzer_fops = {
    .owner   =   THIS_MODULE,
    .open    =   buzzer_open,
    .release =   buzzer_close,
    .ioctl   =   buzzer_ioctl,
};

static struct miscdevice buzzer_dev = {
 .minor   = MISC_DYNAMIC_MINOR,
 .name    = "buzzer",
 .fops    = &buzzer_fops,
};

static int __init buzzer_init(void)
{
```

```
{
 int ret;
    unsigned long cfg;

    /* 注册杂项设备 */
 ret = misc_register(&buzzer_dev);

 /* 申请GPIO */
 ret = gpio_request(S5PV210_GPD0(2), "GPD0");

    /* 设置GPIO为输入，不让蜂鸣器响 */
 s3c_gpio_cfgpin(S5PV210_GPD0(2), S3C_GPIO_INPUT);

    /* 映射地址 */
    tcfg0  = ioremap(TCFG0, 4);
    tcfg1  = ioremap(TCFG1, 4);
    tcon   = ioremap(TCON, 4);
    tcntb2 = ioremap(TCNTB2, 4);
    tcmpb2 = ioremap(TCMPB2, 4);

    /* 配置定时器时钟 */
    /* prescaler1+1 */
    cfg = readl(tcfg0);
    cfg &= ~(0xFF<<8);
    cfg |= (0x0F<<8);
    writel(cfg, tcfg0);

    /* MUX1 */
    cfg = readl(tcfg1);
    cfg &= ~(0xF<<8);
    cfg |= (0x4<<8);
    writel(cfg, tcfg1);

 return ret;
}

static void __exit buzzer_exit(void)
{
    /* 注销杂项设备 */
 misc_deregister(&buzzer_dev);

    /* 取消地址映射 */
    iounmap(tcfg0);
    iounmap(tcfg1);
    iounmap(tcon);
    iounmap(tcntb2);
    iounmap(tcmpb2);
}

module_init(buzzer_init);
module_exit(buzzer_exit);
MODULE_LICENSE("GPL");
```

四、测试

编译驱动程序，加载模块，会生成`/dev/buzzer`设备节点

测试应用程序

```c
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>


#define DEVNAME         "/dev/buzzer"

#define PWM_IOCTL_SET_FREQ              1
#define PWM_IOCTL_STOP                  0


int main(void)
{
        int fd = -1;

        fd = open(DEVNAME, O_RDWR);
        if (fd < 0)
        {
                perror("open");
                return -1;
        }

        ioctl(fd, PWM_IOCTL_SET_FREQ, 10000);
        sleep(1);
        ioctl(fd, PWM_IOCTL_STOP);
        sleep(1);
        ioctl(fd, PWM_IOCTL_SET_FREQ, 3000);
        sleep(1);
        ioctl(fd, PWM_IOCTL_STOP);
        sleep(1);


        close(fd);

        return 0;
}
```

运行测试程序，可以听到不同频率的蜂鸣器响声