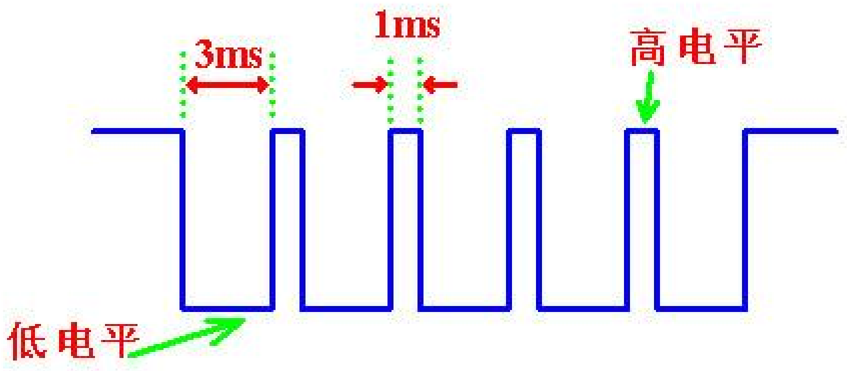
第11章 PWM编程

PWM概述

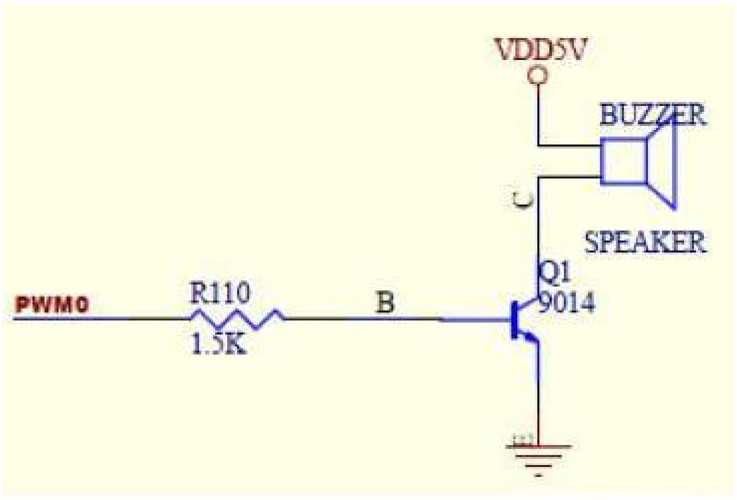
PWM，英文名Pulse Width Modulation，是脉冲宽度调制缩写，它是通过对一系列脉冲的宽度进行调制，等效出所需要的波形（包含形状以及幅值），对模拟信号电平进行数字编码，也就是说通过调节占空比的变化来调节信号、能量等的变化，占空比就是指在一个周期内，信号处于高电平的时间占据整个信号周期的百分比，例如方波的占空比就是50%。是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。



占空比为25%信号

PWM信号把模拟信号转化为数字电路所需要的编码，现在基本是采用数字电路，因此在很多场合都采用PWM信号，我们经常见到的就是交流调光电路，也可以说是无级调速，高电平占多一点，也就是占空比大一点亮度就亮一点，占空比小一点亮度就没有那么亮，前提是PWM的频率要大于我们人眼识别频率，要不然会出现闪烁现象。
除了在调光电路应用，还有在直流斩波电路、蜂鸣器驱动、电机驱动、逆变电路、加湿机雾化量等都会

有应用。



PWM的参数说明

https://www.kernel.org/doc/Documentation/pwm.txt

`period`

PWM信号的总周期（读/写）。
值以纳秒为单位，是活动和非活动的总和
PWM的时间。

`duty_cycle（占空比）`

PWM信号的有效时间（读/写）。
值以纳秒为单位，且必须小于周期。
在NORMAL模式下，表示一个周期内高电平持续的时间
在INVERTED模式下，表示一个周期中低电平持续的时间

`polarity`

改变PWM信号的极性（读/写）。
写入此属性仅在PWM芯片支持更改时才有效
极性。只有PWM不能改变极性
启用。值是字符串“normal”或“inversed”。

`enable`

启用/禁用PWM信号（读/写）。

- 0 - 禁用
- 1 - 启用

用户层查看PWM

如果在内核配置中启用了CONFIG_SYSFS，则会提供一个简单的sysfs接口来使用用户空间的PWM。它在/ sys / class / pwm /中公开。每个被探测的PWM控制器/芯片将被输出为pwmchipN，其中N是PWM芯片的基础。你在目录里面会发现：

1 `echo 0 > /sys/class/pwm/pwmchip0/export` /*设置PWM4输出，调出pwm0目录下设备节点，用于以下配置 */

2 `echo 1000000 >/sys/class/pwm/pwmchip0/pwm0/period` /*设置PWM4一个周期的持续时间，单位为ns，即1K Hz */

3 `echo 500000 >/sys/class/pwm/pwmchip0/pwm0/duty_cycle` /*设置一个周期中的”ON”时间，单位为ns，即占空比=duty_cycle/period=50% */

4 `echo 1 >/sys/class/pwm/pwmchip0/pwm0/enable` /*设置PWM4使能 */

PWM的SYSFS使用

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <linux/ioctl.h>

#define dbmsg(fmt, args ...) printf("%s[%d]: "fmt"\n", __FUNCTION__,
__LINE__,##args)

#define DUTY                "duty"
#define PERIOD              "1000000"
#define DUTYCYCLE           "500000"
#define LENGTH              100

int fd_period = 0,fd_duty = 0,fd_enable = 0,duty_m = 0;

int usage()
{
    printf("usage:\n");
    printf("./pwm-sysfs-test duty <0/1> : 0-->static; 1-->dynamic \n");
    return 0;
}

int pwm_setup()
{
  int fd,ret;

  fd = open("/sys/class/pwm/pwmchip0/export", O_WRONLY);
  if(fd < 0)
  {
      dbmsg("open export error\n");
      return -1;
  }
  ret = write(fd, "0", strlen("0"));
  if(ret < 0)
  {
      dbmsg("creat pwm0 error\n");
      return -1;
  }else
    dbmsg("export pwm0 ok\n");

  fd_period = open("/sys/class/pwm/pwmchip0/pwm0/period", O_RDWR);
  fd_duty = open("/sys/class/pwm/pwmchip0/pwm0/duty_cycle", O_RDWR);
  fd_enable = open("/sys/class/pwm/pwmchip0/pwm0/enable", O_RDWR);

  if((fd_period < 0)||(fd_duty < 0)||(fd_enable < 0))
  {
      dbmsg("open error\n");
```

```
        return -1;
    }

    ret = write(fd_period, PERIOD,strlen(PERIOD));
    if(ret < 0)
    {
        dbmsg("change period error\n");
        return -1;
    }else
      dbmsg("change period ok\n");

    ret = write(fd_duty, DUTYCYCLE, strlen(DUTYCYCLE));
    if(ret < 0)
    {
        dbmsg("change duty_cycle error\n");
        return -1;
    }else
      dbmsg("change duty_cycle ok\n");

    ret = write(fd_enable, "1", strlen("1"));
    if(ret < 0)
    {
        dbmsg("enable pwm0 error\n");
        return -1;
    }else
      dbmsg("enable pwm0 ok\n");

    duty_m = atoi(DUTYCYCLE)/2;
    printf("duty_m: %d \n",duty_m);

    return 0;
}

int main ( int argc, char *argv[] )
{
    int ret;
    int num;
    if(argc < 2)
    {
      usage();
      return -1;
    }

    if(strncmp(argv[1],DUTY, sizeof(DUTY)) == 0)
    {
      dbmsg("%s", DUTY);
      if(argc != 3)
      {
        usage();
        return -1;
      }
      pwm_setup();
    }

    return 0;
}
```

PWM应用编程

The main useful user API are the following:
`devm_pwm_get()` or `pwm_get()` / `pwm_put()`: this API is used to look up, request, then free a PWM device.
`pwm_init_state()`, `pwm_get_state()`, `pwm_apply_state()`: this API is used to initialize, retrieve and apply the current PWM device state.
`pwm_config()`: this API updates the PWM device configuration (period and duty cycle).

修改设备树

```
beeper {
compatible = "pwm-beeper";
pwms = <&pwm 0 1000000 0>;
pinctrl-names = "default";
pinctrl-0 = <&pwm0_pin>;
};
```

修改配置文件

Activate PWM framework in the kernel configuration through the Linux menuconfig tool, Menuconfig or how to configure kernel (`CONFIG_PWM=y`):

```
Device Drivers  --->
   [*] Pulse-Width Modulation (PWM) Support  --->
```

添加驱动

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <asm/gpio.h>
#include <linux/pwm.h>

//#include <plat/gpio-cfg.h>

#define PWM_ON  0x100001
#define PWM_OFF 0x100002

struct pwm_device *pwm_dev_2;
struct pwm_device *pwm_dev_3;

static long pwm_ioctl(struct file *file,
                        unsigned int cmd,
                        unsigned long arg)
{
    int ret;
    switch(cmd) {
        case PWM_ON:
                ret = pwm_config(pwm_dev_2,200000,500000);
                if(ret < 0){
                    printk("pwm_dev_2 ioctl fail");
                    return 0;
                }
                ret = pwm_config(pwm_dev_3,300000,500000);
                if(ret < 0){
                    printk("pwm_dev_3 ioctl fail");
                }
                pwm_enable(pwm_dev_2);
                pwm_enable(pwm_dev_3);
            break;
        case PWM_OFF:
                ret = pwm_config(pwm_dev_2,0,500000);
                if(ret < 0){
                    printk("pwm_dev_2 ioctl fail");
                    return 0;
                }
                ret = pwm_config(pwm_dev_3,0,500000);
                if(ret < 0){
                    printk("pwm_dev_3 ioctl fail");
                }
                pwm_disable(pwm_dev_2);
                pwm_disable(pwm_dev_3);
            break;
    }
    return 0;
}
//定义初始化硬件操作方法
static struct file_operations pwm_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = pwm_ioctl
};
//定义初始化混杂设备对象
static struct miscdevice pwm_misc = {
    .minor = MISC_DYNAMIC_MINOR, //动态分配次设备号
    .name = "mypwm",             //dev/mypwm
    .fops = &pwm_fops
};
static int pwm_init(void)
{
    int ret;
    printk("regisger pwm_misc device\n");
    //1.申请pwm资源，设置输出为0
    pwm_dev_2 = pwm_request(1,"pwm_2");
    if(pwm_dev_2 == NULL){
        printk("pwm_dev_2 register fail\n");
    }
    pwm_dev_3 = pwm_request(2,"pwm_3");
    if(pwm_dev_3 == NULL){
        printk("pwn_dev_3 register fail\n");
    }


    ret = pwm_config(pwm_dev_2,0,500000);
    if(ret < 0){
        printk("pwm_config_2 init fail\n");
        return 0;
    }
    ret = pwm_config(pwm_dev_3,0,500000);
    if(ret < 0){
        printk("pwm_config_3 init fail\n");
        return 0;
    }

    ret = pwm_enable(pwm_dev_2);
    if(ret == 0){
        printk("pwm_enable_dev_2 init success\n");
    }
    if(ret < 0 ){
        printk("pwm_enable_dev_2 init fail\n");
        return 0;
    }
    ret = pwm_enable(pwm_dev_3);
    if(ret == 0){
        printk("pwm_enable_dev_3 init success\n");
    }
    if(ret < 0 ){
        printk("pwm_enable_dev_3 init fail\n");
        return 0;
    }
    //2.注册混杂设备
    misc_register(&pwm_misc);
    return 0;
}

static void pwm_exit(void)
{
    printk("unregister pwm_misc device\n");
    //1.卸载混杂设备
    misc_deregister(&pwm_misc);
    //2.释放pwm资源
    pwm_config(pwm_dev_2,0,500000);
    pwm_disable(pwm_dev_2);
    pwm_free(pwm_dev_2);

    pwm_config(pwm_dev_3,0,500000);
    pwm_disable(pwm_dev_3);
    pwm_free(pwm_dev_3);
}
module_init(pwm_init);
module_exit(pwm_exit);
MODULE_LICENSE("GPL");
```

运行测试

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

#define PWM_ON  0x100001
#define PWM_OFF 0x100002

int main(void)
{
    int fd;
    int a;
    fd = open("/dev/mypwm", O_RDWR);
    if (fd < 0)
        return -1;

    while(1) {
            ioctl(fd, PWM_ON);
    }
    close(fd);
    return 0;
}
```