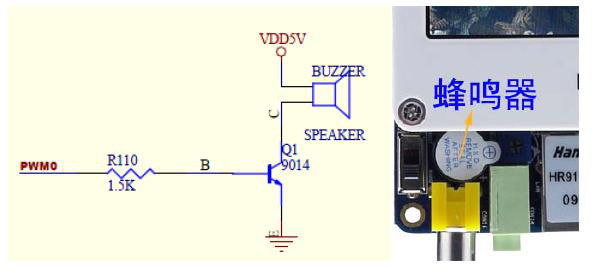
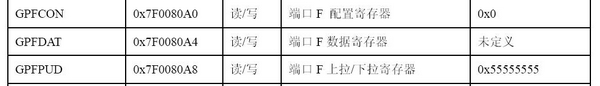
mini6410的蜂鸣器Buzzer是通过PWM控制的，原理图如下所示，其中，连接蜂鸣器的PWM0对应GPF14，该引脚可通过软件设置为PWM输出，也可以作为普通的GPIO使用。

C:\Users\lixin\AppData\Local\Temp\enhtmlclip\Image(3).png

由以上可知，我们需要在驱动程序中，首先把GPF14端口设置为PWM功能输出，再设定相应的Timer就可以控制PWM的输出频率了。友善给的驱动代码如下所示：

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/init.h>

#include <linux/delay.h>

#include <linux/poll.h>

#include <asm/irq.h>

#include <asm/io.h>

#include <linux/interrupt.h>

#include <asm/uaccess.h>

#include <mach/hardware.h>

#include <plat/regs-timer.h>

#include <mach/regs-irq.h>

#include <asm/mach/time.h>

#include <linux/clk.h>

#include <linux/cdev.h>

#include <linux/device.h>

#include <linux/miscdevice.h>

#include <mach/map.h>

#include <mach/regs-clock.h>

#include <mach/regs-gpio.h>

#include <plat/gpio-cfg.h>

#include <mach/gpio-bank-e.h>

#include <mach/gpio-bank-f.h>

#include <mach/gpio-bank-k.h>

#define DEVICE\_NAME "pwm" //定义设备名

#define PWM\_IOCTL\_SET\_FREQ 1 //定义宏变量，用于后面的ioctl中的switch case

#define PWM\_IOCTL\_STOP 0 //定义信号量lock

static struct semaphore lock;

/\* freq: pclk/50/16/65536 ~ pclk/50/16

\* if pclk = 50MHz, freq is 1Hz to 62500Hz

\* human ear : 20Hz~ 20000Hz

\*/

static void PWM\_Set\_Freq( unsigned long freq ) //设置pwm的频率，配置各个寄存器

{

unsigned long tcon;

unsigned long tcnt;

unsigned long tcfg1;

unsigned long tcfg0;

struct clk \*clk\_p;

unsigned long pclk;

unsigned tmp;

tmp = readl(S3C64XX\_GPFCON); //参考第一个表，设置GPF14为TOUT0，pwm输出

tmp &= ~(0x3U << 28);

tmp |= (0x2U << 28);

writel(tmp, S3C64XX\_GPFCON);

tcon = \_\_raw\_readl(S3C\_TCON); //读定时器配置寄存器TCON到tcon

tcfg1 = \_\_raw\_readl(S3C\_TCFG1); //读寄存器TCFG1到tcfg1

tcfg0 = \_\_raw\_readl(S3C\_TCFG0); //读寄存器TCFG0到tcfg0

//prescaler = 50

tcfg0 &= ~S3C\_TCFG\_PRESCALER0\_MASK; //S3C\_TCFG\_PRESCALER0\_MASK=255(11111111),为定时器0和1的预分频值得掩码，TCFG[0~8]

tcfg0 |= (50 - 1); //tcfg0=00110001,预分频为50

//mux = 1/16

tcfg1 &= ~S3C\_TCFG1\_MUX0\_MASK; //定时器0分割值得掩码 (15<<0)

tcfg1 |= S3C\_TCFG1\_MUX0\_DIV16; //定时器0进行1/16分割 (4<<0)->0100

\_\_raw\_writel(tcfg1, S3C\_TCFG1); //将tcfg1的值写到分割寄存器中

\_\_raw\_writel(tcfg0, S3C\_TCFG0); //将tcfg0的值写到分频寄存器中

clk\_p = clk\_get(NULL, "pclk"); //得到pclk

pclk = clk\_get\_rate(clk\_p);

tcnt = (pclk/50/16)/freq; //得到定时器的输入时钟，进而设置PWM的调制频率

\_\_raw\_writel(tcnt, S3C\_TCNTB(0)); //PWM脉宽调制的频率等于定时器的输入时钟

\_\_raw\_writel(tcnt/2, S3C\_TCMPB(0)); //占空比是50%

tcon &= ~0x1f;

tcon |= 0xb; //禁用死区, 间隔模式开启, 逆变器关闭, 自动更新TCNTB0&TCMPB0, 开始定时器0

\_\_raw\_writel(tcon, S3C\_TCON); //把tcon的设置写到计数控制寄存器S3C\_TCON中

tcon &= ~2; //clear manual update bit

\_\_raw\_writel(tcon, S3C\_TCON);

}

void PWM\_Stop( void )

{

unsigned tmp;

tmp = readl(S3C64XX\_GPFCON); //设置GPF14为输出

tmp &= ~(0x3U << 28);

writel(tmp, S3C64XX\_GPFCON); //设置GPF14为低电平，使蜂鸣器停止

}

static int s3c64xx\_pwm\_open(struct inode \*inode, struct file \*file)

{

if (!down\_trylock(&lock)) //是否获得信号量。如果是，down\_trylock(&lock)=0,否则非0。

return 0;

else

return -EBUSY; //返回错误信息：请求资源不可用。

}

static int s3c64xx\_pwm\_close(struct inode \*inode, struct file \*file)

{

up(&lock);

return 0;

}

/\*cmd是1，表示设置频率；cmd是0，表示停止pwm\*/

static long s3c64xx\_pwm\_ioctl(struct file \*filep, unsigned int cmd, unsigned long arg)

{

switch (cmd)

{

//如果cmd=1，进入case PWM\_IOCTL\_SET\_FREQ

case PWM\_IOCTL\_SET\_FREQ:

if (arg == 0) //如果设置的频率参数是0

return -EINVAL; //返回错误信息，表示向参数传递了无效的参数

PWM\_Set\_Freq(arg); //否则设置给定的频率

break;

case PWM\_IOCTL\_STOP: //如果cmd=0，进入

default:

PWM\_Stop(); //停止蜂鸣器

break;

}

return 0;

}

static struct file\_operations dev\_fops = //初始化设备的文件操作的结构体

{

.owner = THIS\_MODULE,

.open = s3c64xx\_pwm\_open,

.release = s3c64xx\_pwm\_close,

.unlocked\_ioctl = s3c64xx\_pwm\_ioctl,

};

static struct miscdevice misc = {

.minor = MISC\_DYNAMIC\_MINOR,

.name = DEVICE\_NAME,

.fops = &dev\_fops,

};

static int \_\_init dev\_init(void)

{

int ret;

sema\_init(&lock, 1); //初始化一个互斥锁

ret = misc\_register(&misc); //注册一个misc设备

printk (DEVICE\_NAME"\tinitialized\n");

return ret;

}

static void \_\_exit dev\_exit(void)

{

misc\_deregister(&misc); //注销设备

}

module\_init(dev\_init);

module\_exit(dev\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("FriendlyARM Inc.");

MODULE\_DESCRIPTION("S3C6410 PWM Driver");

**1 CPU 计数器控制寄存器**

1）**配置定时器输入时钟**

TCFG0-时钟配置寄存器0，用于获得预分频值（1~255）

TCFG1-时钟配置寄存器1，用于获得分割值（2，4，8，16，32）

定时器输入时钟频率=PLCK/{预分频+1}/{分割值}

2）**配置PWM 的占空比**

TCNTB0-定时器0 计数缓存寄存器 ，是由定时器的输入时钟分频得到,是脉宽调制的频率

TCMTB0-定时器0 比较缓存寄存器 ，用于设定PWM 的占空比 ，寄存器值为高电平的

假设TCNTB0 的频率是160，如果TCMTB0 是110，则PWM 在110 个周期是高电平，50 周期是低电平，从而占空比为11：5

**3）定时器控制寄存器TCON**

TCON[0~4]用于控制定时器0

**2 内核中基于信号量的Llinux 的并发控制**

在驱动程序中，当多个线程同时访问相同的资源时，可能会引发“竞态”，因此必须对共享资源进行并发控制。信号量（绝大多数作为互斥锁使用）是一种进行并发控制的手段（还有自旋锁，它适合于保持时间非常短的时间）。信号量只能在进程的上下文中使用。

sema\_init(&lock,1)初始化一个互斥锁，即他把信号量lock 设置为1

void up (&lock) 释放信号量，唤醒等待者

int down\_trylock(&lock) 尝试获得信号量lock ，如果能够立刻获得，就获得信号量并返回为0.否则返回非0.并且它不会导致休眠，可以在中断上下文中使用。在PWM 中，当计

数值溢出时，就会引发计数中断。所以在这里用这个函数来获得信号。

 测试程序pwm.c：

#include <stdio.h>

#include <termios.h>

#include <unistd.h>

#include <stdlib.h>

#define PWM\_IOCTL\_SET\_FREQ 1

#define PWM\_IOCTL\_STOP 2

#define ESC\_KEY 0x1b

static int getch(void)

{

struct termios oldt,newt;

int ch;

if (!isatty(STDIN\_FILENO))

{

fprintf(stderr, "this problem should be run at a terminal\n");

exit(1);

}

// save terminal setting

if(tcgetattr(STDIN\_FILENO, &oldt) < 0)

{

perror("save the terminal setting");

exit(1);

}

// set terminal as need

newt = oldt;

newt.c\_lflag &= ~( ICANON | ECHO );

if(tcsetattr(STDIN\_FILENO,TCSANOW, &newt) < 0)

{

perror("set terminal");

exit(1);

}

ch = getchar();

// restore termial setting

if(tcsetattr(STDIN\_FILENO,TCSANOW,&oldt) < 0)

{

perror("restore the termial setting");

exit(1);

}

return ch;

}

static int fd = -1;

static void close\_buzzer(void);

static void open\_buzzer(void)

{

fd = open("/dev/pwm", 0);

if (fd < 0)

{

perror("open pwm\_buzzer device");

exit(1);

}

// any function exit call will stop the buzzer

atexit(close\_buzzer);

}

static void close\_buzzer(void)

{

if (fd >= 0)

{

ioctl(fd, PWM\_IOCTL\_STOP);

close(fd);

fd = -1;

}

}

static void set\_buzzer\_freq(int freq)

{

// this IOCTL command is the key to set frequency

int ret = ioctl(fd, PWM\_IOCTL\_SET\_FREQ, freq);

if(ret < 0)

{

perror("set the frequency of the buzzer");

exit(1);

}

}

static void stop\_buzzer(void)

{

int ret = ioctl(fd, PWM\_IOCTL\_STOP);

if(ret < 0)

{

perror("stop the buzzer");

exit(1);

}

}

int main(int argc, char \*\*argv)

{

int freq = 1000 ;

open\_buzzer();

printf( "\nBUZZER TEST ( PWM Control )\n" );

printf( "Press +/- to increase/reduce the frequency of the BUZZER\n" ) ;

printf( "Press 'ESC' key to Exit this program\n\n" );

while( 1 )

{

int key;

set\_buzzer\_freq(freq);

printf( "\tFreq = %d\n", freq );

key = getch();

switch(key)

{

case '+':

if( freq < 20000 )

freq += 10;

break;

case '-':

if( freq > 11 )

freq -= 10 ;

break;

case ESC\_KEY:

case EOF:

stop\_buzzer();

exit(0);

default:

break;

}

}

}