**嵌入式课程设计报告**

**一、课程设计题目**

基于OK6410的蜂鸣器电子琴

**二、实验设备**

PC机、OK-6410开发板、串口

**三、具体内容及要求**

使用OK-6410开发板实现PWM驱动蜂鸣器的简易电子琴，通过串口连接实现电脑按键控制，发出相应音阶声音。

**四、设计思路及问题**

1.设计思路

主要需要实现以下要求

驱动文件：开发板蜂鸣器驱动，键盘驱动

调试代码：各个音阶参数，对传输按键信息进行响应

通信协议：串口连接PC和开发板，实现电脑控制蜂鸣器

2.问题

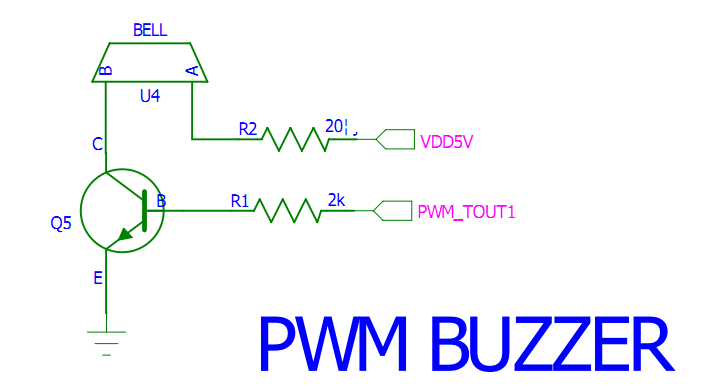
A、蜂鸣器驱动

B、PWM波占空比的调整，改变频率，实现不同音阶

C、实现蜂鸣器发出不同音阶的声音

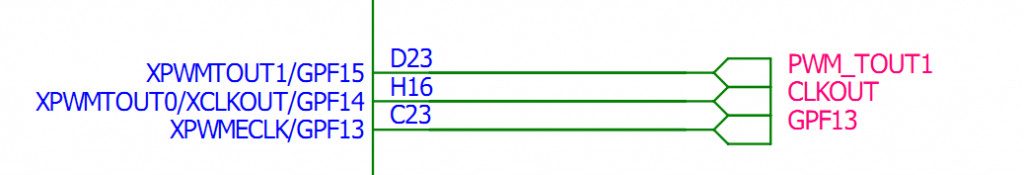
**五、实验原理**

1. 蜂鸣器

查看蜂鸣器buzzer在底板中的管脚信息

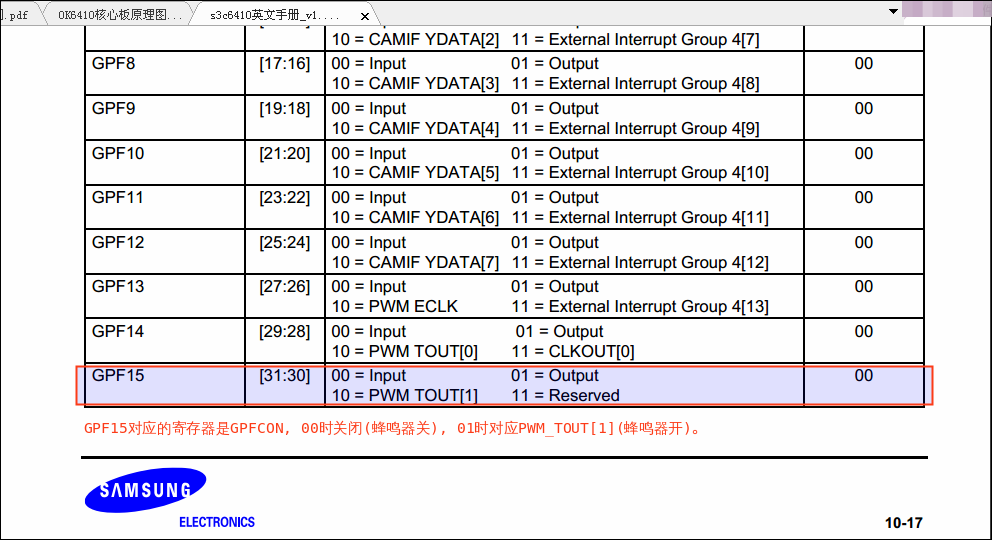
**图一 Ok-6410蜂鸣器原理图**

查看蜂鸣器在总线中的信息

、

**图二 Ok-6410蜂鸣器管脚图**

翻看S3C6410芯片手册，查看GPF15相关信息



**图三 GPF15寄存器**

1. PWM定时器

脉宽调制（PWM）基本原理：控制方式就是对逆变电路开关器件的通断进行控制，使输出端得到一系列幅值相等的脉冲，用这些脉冲来代替正弦波或所需要的波形。逆变电路的基本作用是在驱动信号的控制下，将直流电源转换成频率和电压可以任意调节的交流电源。采样控制理论中有一个重要结论：冲量相等而形状不同的窄脉冲加在具有惯性的环节上时，其效果基本相同。PWM控制技术就是以该结论为理论基础。

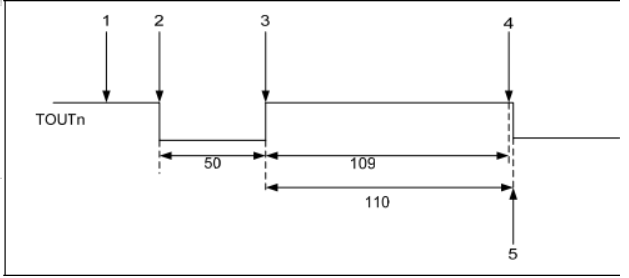
在蜂鸣器控制中PWM作用于电源部分，产生20—20000HZ的声音频率，在这个频率区间的声音才能使人听见。

PWM软件法控制充电电流的基本思想就是利用单片机具有的PWM端口，在不改变PWM方波周期的前提下，通过软件的方法调整单片机的PWM控制寄存器来调整PWM的占空比，从而控制充电电流。该方法所要求的单片机必须具有ADC端口和PWM端口这两个必须条件，另外ADC的位数尽量高，单片机的工作速度尽量快。在调整充电电流前，单片机先快速读取充电电流的大小，然后把设定的充电电流与实际读取到的充电电流进行比较，若实际电流偏小则向增加充电电流的方向调整PWM的占空比；若实际电流偏大则向减小充电电流的方向调整PWM的占空比。在软件PWM的调整过程中要注意ADC的读数偏差和电源工作电压等引入的纹波干扰，合理采用算术平均法等数字滤波技术。

S3C6410X中有5个定时器，这些定时器产生内部中断。其中，Timer0和Timer1具有PWM功能，而Timer2,3,4没有此功能。

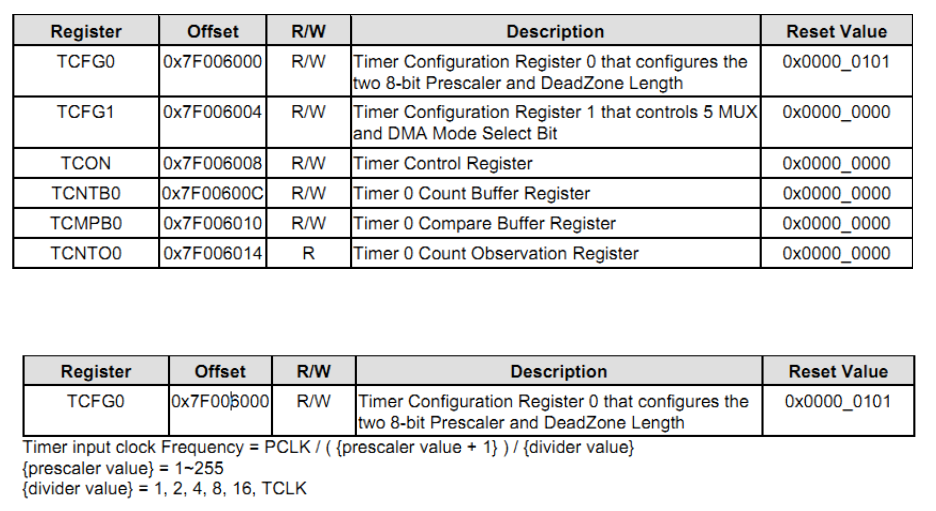
PWM具有两种操作模式：自动装载模式，一次触发模式。为实现PWM功能，芯片提供了16个功能寄存器。这些功能寄存器都连接APB总线。

定时器具有双缓冲特性，这样就能在不停止当前定时器操作的情况下，为下次定时器运行装入新的数值。尽管为定时器设置了新数值，但当前的定时操作能够成功完成。定时器从TCNTBn读取的值是为下次延时定时用的，并不影响当前定时器的运行。当TCNTn减小到0的时候，TCNTBn的值会自动复制到TCNTn中，这就是说的自动装载操作。定时器的当前计数值可以从定时计数观察寄存器中TCNTOn读取。如果TCNTn为0且从装载也为0的话则TCNTn不在进行下次操作。

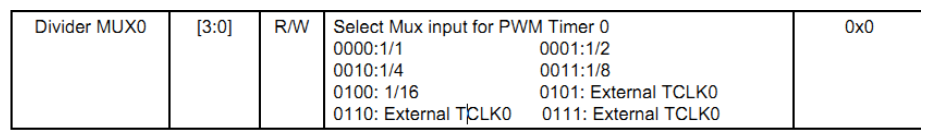


**图四 PWM波**

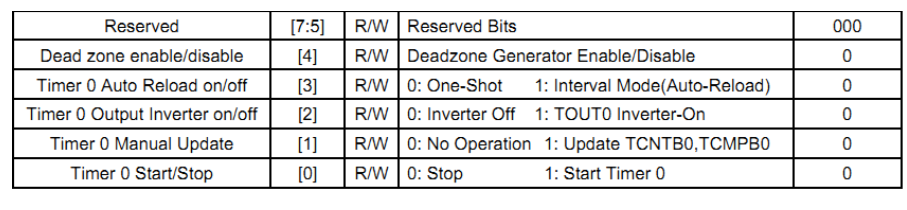
对于PWM 功能，要用到寄存器TCMPBn,当递减计数器down-counter的值和比较寄存器TCMPBn的值相同时，定时控制逻辑模块就会改变输出电平。因此比较寄存器TCMPBn决定了PWM的输出。



**图五 定时工作0工作寄存器**



**图六 TCFG1**

****

**图七 TCON**

**表一 音符对应频率**

|  |  |  |  |
| --- | --- | --- | --- |
| 音符 | 低音/Hz | 中音/Hz | 高音/Hz |
| 1 | 262 | 523 | 1046 |
| 2 | 294 | 578 | 1175 |
| 3 | 330 | 659 | 1318 |
| 4 | 349 | 698 | 1397 |
| 5 | 392 | 784 | 1568 |
| 6 | 440 | 880 | 1760 |
| 7 | 494 | 988 | 1976 |

**六、实验代码及流程**

A、蜂鸣器驱动

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/init.h>

#include <asm/io.h>

#include <linux/interrupt.h>

#include <linux/device.h>

#include <asm/uaccess.h>

#include <linux/miscdevice.h>

#include <mach/map.h>

#include <mach/regs-clock.h>

#include <mach/regs-gpio.h>

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

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

#define DEVICE\_NAME "my\_buzzer"

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/init.h>

#include <linux/miscdevice.h>

#include <linux/delay.h>

#include <asm/uaccess.h>

#include <linux/device.h>

#include <linux/cdev.h>

#include <asm/irq.h>

#include <mach/gpio.h>

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

#include <plat/gpio-cfg.h>

#include <mach/hardware.h>

#include <linux/io.h>

#define BUZZER\_MAJOR 240

int buzzer\_open(struct inode \*inode, struct file \*filp)

{

unsigned int tmp;

tmp = readl(S3C64XX\_GPFCON);

tmp = (tmp & ~(0xc0000000) | (0x40000000)); //set the GPIO output mode

writel(tmp, S3C64XX\_GPFCON);

printk("$$$$$$$$$$$buzzer\_open$$$$$$$$$\n");

return 0;

}

ssize\_t buzzer\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

{

printk("$$$$$$$$$$buzzer\_read$$$$$$$$$\n");

return count;

}

ssize\_t buzzer\_write(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

{

char mbuf[10];

unsigned int tmp;

copy\_from\_user(mbuf,buf,count);

switch(mbuf[0])

{

case 0:

tmp = readl(S3C64XX\_GPFDAT);

tmp |= (0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

case 1:

tmp = readl(S3C64XX\_GPFDAT);

tmp |= (0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

case 2:

tmp = readl(S3C64XX\_GPFDAT);

tmp |= (0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

case 3:

tmp = readl(S3C64XX\_GPFDAT);

tmp |= (0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

case 4:

tmp = readl(S3C64XX\_GPFDAT);

tmp |= (0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

case 5:

tmp = readl(S3C64XX\_GPFDAT);

tmp |= (0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

case 6:

tmp = readl(S3C64XX\_GPFDAT);

tmp &= ~(0x8000);

writel(tmp, S3C64XX\_GPFDAT);

break;

default:

break;

}

printk("$$$$$$$$$$buzzer\_write$$$$$$$$$\n");

return count;

}

int buzzer\_release(struct inode \*inode, struct file \*filp)

{

printk("$$$$$$$$$$buzzer\_release$$$$$$$$$\n");

return 0;

}

struct file\_operations my\_fops = {

.owner = THIS\_MODULE,

.open = buzzer\_open,

.read = buzzer\_read,

.write = buzzer\_write,

.release = buzzer\_release,

};

static int buzzer\_init(void)

{

int rc;

printk("Test buzzer dev\n");

rc = register\_chrdev(BUZZER\_MAJOR, "buzzer", &my\_fops);

if(rc < 0)

{

printk("register %s dev error\n", "buzzer");

return -1;

}

printk("$$$$$$$$$ register buzzer dev OK\n");

return 0;

}

static void buzzer\_exit(void)

{

unregister\_chrdev(BUZZER\_MAJOR, "buzzer");

printk("Good Bye!\n");

}

MODULE\_LICENSE("GPL");

module\_init(buzzer\_init);

module\_exit(buzzer\_exit);

B、PWM频率驱动

#include<linux/kernel.h>

#include<linux/init.h>

#include<linux/module.h>

#include<linux/types.h>

#include<linux/fcntl.h>

#include<linux/cdev.h>

#include<linux/version.h>

#include<linux/ctype.h>

#include<linux/delay.h>

#include<linux/pagemap.h>

#include<linux/vmalloc.h>

#include<linux/fs.h>

#include<linux/slab.h>

#include<linux/errno.h>

#include<mach/hardware.h>

#include<mach/map.h>

#include<mach/gpio.h>

#include<asm/io.h>

#include<asm/irq.h>

#include<asm/signal.h>

#include<asm/uaccess.h>

#include<plat/regs-timer.h>

#include<plat/gpio-cfg.h>

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

#define simple\_MAJOR 241

#define simple\_MINOR 0

#define BEEP\_S3C6410\_ON 0

#define BEEP\_S3C6410\_OFF 1

#define PWM\_TIMER1\_AUTO\_RELOAD (1<<11)

#define PWM\_TIMER1\_MANUAL\_UPDATE (1<<9)

#define PWM\_TIMER1\_START (1<<8)

struct simple\_dev

{

struct cdev cdev;

};

struct simple\_dev \*simple\_devices;

static unsigned char simple\_inc = 0;

static void startBeep(void)

{

unsigned long tcon,tcon1 , tcon2 , tcnt , pwm\_PCLK , tcmp;

s3c\_gpio\_cfgpin(S3C64XX\_GPF(15) , (0x2U<<30));//10 = PWM TOUT[1]

tcon = \_\_raw\_readl(S3C2410\_TCON);

tcon |= PWM\_TIMER1\_AUTO\_RELOAD|PWM\_TIMER1\_MANUAL\_UPDATE;

\_\_raw\_writel(tcon , S3C2410\_TCON);

printk("S3C2410\_TCON %ld\n" , tcon);

tcon1 = \_\_raw\_readl(S3C2410\_TCFG0);

printk("S3C2410\_TCFG0 = %ld\n" , tcon1);

tcon1 |= 0x0000000F;

printk("S3C2410\_TCFG0 = %ld\n" , tcon1);

tcon2 = \_\_raw\_readl(S3C2410\_TCFG1);

printk("S3C2410\_TCFG1 = %ld\n" , tcon2);

tcon2 |= 0x000000F0;

printk("S3C2410\_TCFG1 = %ld\n" , tcon2);

pwm\_PCLK = 66500000/(tcon1\*tcon2);

printk("pwm\_PCLK = %ld\n" , pwm\_PCLK);

tcnt = pwm\_PCLK/1000;

printk("tcnt is %ld\n" , tcnt);

tcmp = tcnt/10000;

\_\_raw\_writel(tcnt , S3C2410\_TCNTB(1));

\_\_raw\_writel(tcmp , S3C2410\_TCMPB(1));

tcon = \_\_raw\_readl(S3C2410\_TCON);

tcon |= PWM\_TIMER1\_START|PWM\_TIMER1\_AUTO\_RELOAD;

\_\_raw\_writel(tcon , S3C2410\_TCON);

}

static void stopBeep(void)

{

unsigned long tcon;

tcon = \_\_raw\_readl(S3C2410\_TCON);

tcon &=~(PWM\_TIMER1\_AUTO\_RELOAD);

tcon &=~(PWM\_TIMER1\_START);

\_\_raw\_writel(tcon , S3C2410\_TCON);

printk("S3C2410\_TCON = %ld\n" , tcon);

s3c\_gpio\_cfgpin(S3C64XX\_GPF(15) , 0);

}

int simple\_open(struct inode \*inode , struct file \*filp)

{

struct simple\_dev \*dev;

if(simple\_inc > 0)

{

return -ERESTARTSYS;

}

simple\_inc++;

dev = container\_of(inode->i\_cdev , struct simple\_dev , cdev);

filp->private\_data = dev;

return 0;

}

int simple\_release(struct inode \*inode , struct file \*filp)

{

simple\_inc--;

return 0;

}

static long simple\_ioctl(struct file \*filp , unsigned int cmd , unsigned long arg)

{

switch(cmd)

{

case BEEP\_S3C6410\_ON:

startBeep();

break;

case BEEP\_S3C6410\_OFF:

stopBeep();

break;

default:

break;

}

return 1;

}

struct file\_operations simple\_fops = {

.owner = THIS\_MODULE,

.open = simple\_open,

.release = simple\_release,

.unlocked\_ioctl = simple\_ioctl,

};

void simple\_cleanup\_module(void)

{

dev\_t devno = MKDEV(simple\_MAJOR , simple\_MINOR);

if(simple\_devices)

{

cdev\_del(&simple\_devices->cdev);

kfree(simple\_devices);

}

unregister\_chrdev\_region(devno , 1);

}

int simple\_init\_module(void)

{

int result;

dev\_t dev = 0;

dev = MKDEV(simple\_MAJOR , simple\_MINOR);

result = register\_chrdev\_region(dev , 1 , "DEMO");

if(result < 0)

{

printk(KERN\_WARNING"simle can't get major %d\n" , simple\_MAJOR);

return result;

}

simple\_devices = kmalloc(sizeof(struct simple\_dev) , GFP\_KERNEL);

if(!simple\_devices)

{

result = -ENOMEM;

goto fail;

}

memset(simple\_devices , 0 , sizeof(struct simple\_dev));

cdev\_init(&simple\_devices->cdev , &simple\_fops);

simple\_devices->cdev.owner = THIS\_MODULE;

simple\_devices->cdev.ops = &simple\_fops;

result = cdev\_add(&simple\_devices->cdev , dev , 1);

if(result)

{

printk(KERN\_NOTICE"Error %d adding DEMO\n" , result);

goto fail;

}

return 0;

fail:

simple\_cleanup\_module();

return result;

}

module\_init(simple\_init\_module);

module\_exit(simple\_cleanup\_module);

MODULE\_LICENSE("Dual BSD/GPL");

C、测试代码

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<unistd.h>

#define BEEP\_ON 0

#define BEEP\_OFF 1

int main(void)

{

int fd;

fd = open("/dev/z", O\_RDWR);

if(fd < 0)

{

printf("can't open \n");

exit(0);

}

while(1)

{

ioctl(fd , BEEP\_ON , 0);

sleep(1);

ioctl(fd , BEEP\_OFF , 0);

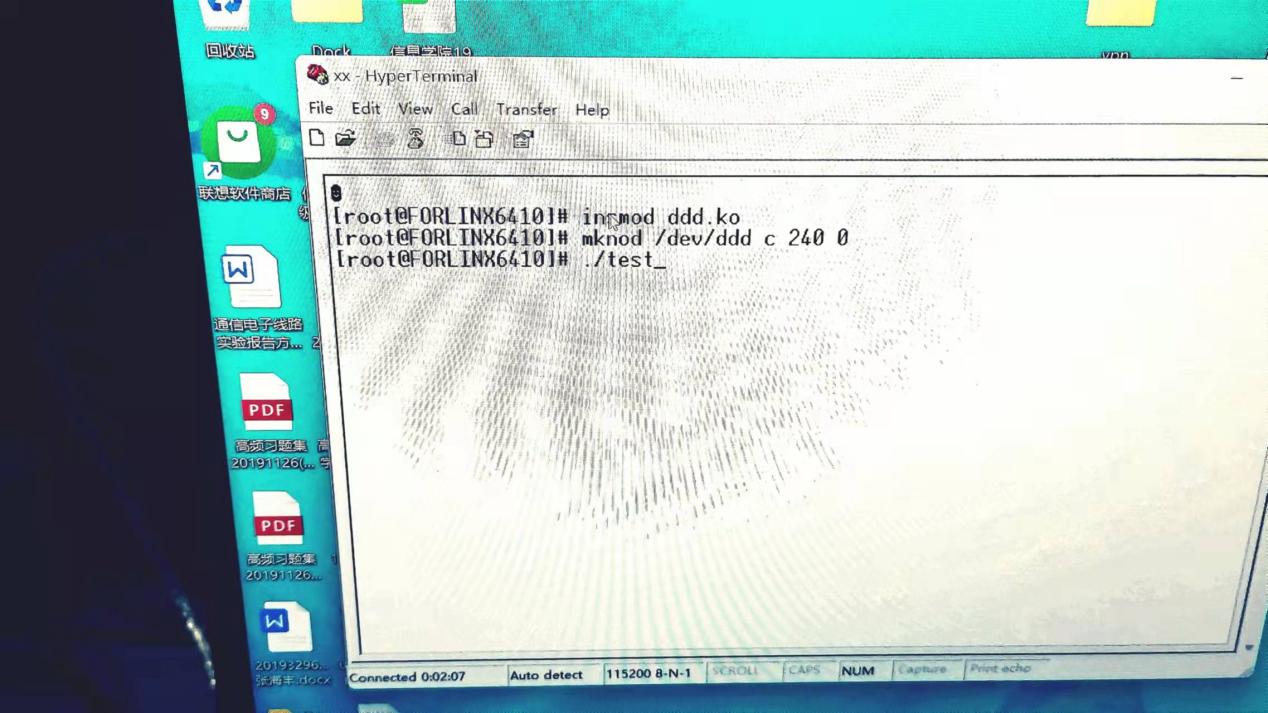
}

close(fd);

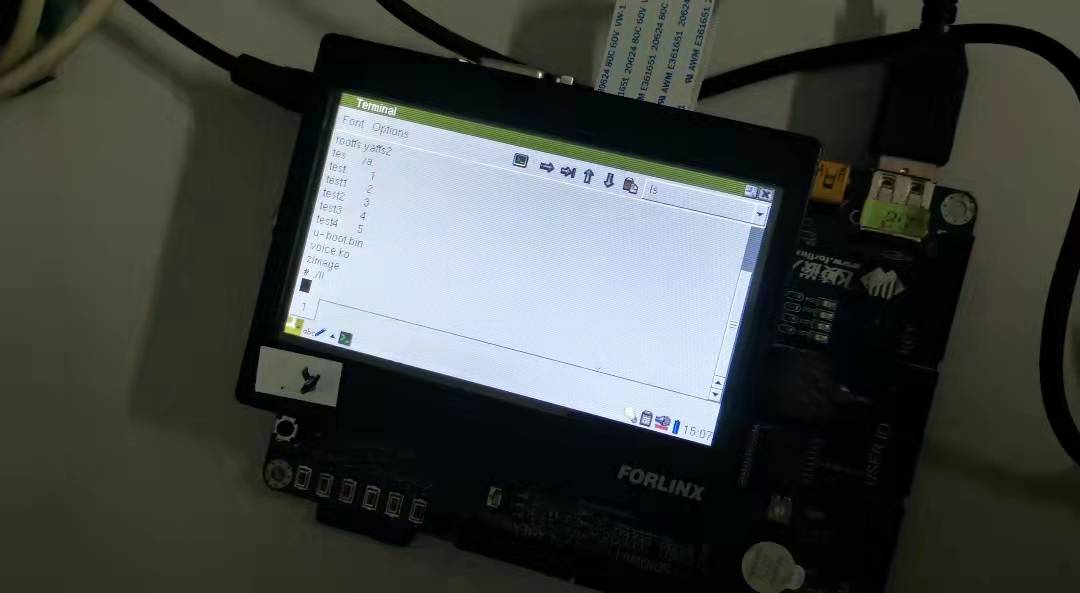
return 0;

}

**七、实验成果**



**图八 PC通过串口连接，控制开发板**

**图九 蜂鸣器运行**

**八、心得体会**

本次课设是对本学期嵌入式技术课程的一次总结和检验。通过这个课设，不仅检验了我们这学期所学的嵌入式编程所学的基本知识，如驱动程序和测试代码的编写，makefile文件，以及将烧写好的程序拷进开发板中并运行，同时检验了我们利用所学知识，按照要求设计实例的能力，进一步使我们所学的知识融会贯通，培养了我们开放创新的能力。

在完成课设的过程中，我们也暴露出许多问题。第一个是驱动的编写，由于开发板的版本太旧，网上难以找到相应的驱动，所以之能在网上找到相似处理器的驱动进行修改，才能在开发板上使用；第二个是通过修改PWM波占空比来实现输出频率的改变，即让蜂鸣器输出不同的音调。在二个问题时，遇到较大的困难。首先是配置定时器输入时钟，定时器输入时钟频率=PLCK/{预分频+1}/{分割值}，再除以所需要的频率。就能输出所需要的频率，但是TCFG0、TCFG1

两个寄存器的配置比较复杂，配置成功之后才能获得预分频值（1~255）以及分割值（2，4，8，16，32）。其次是PWM的占空比，通过查阅手册，知道需要修改TCNTB0和TCMTB0的值来改变占空比，TCNTB0为总周期长，TCMTB0为高电平长。最后是输出问题，修改了上述各参数后，但是蜂鸣器未能发出指定频率的音调，可能的原因是，某些引脚参数可能还有配置问题，但是蜂鸣器发声正常。