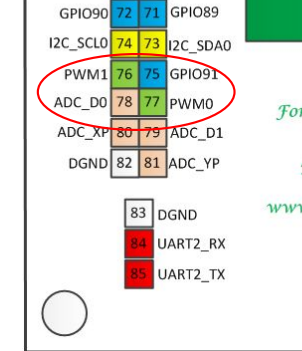
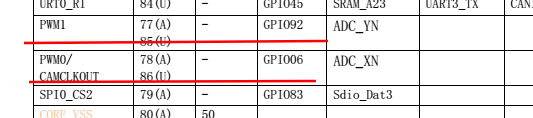
# 3. PWM控制输出

## 3.1利用 LED\_PWM

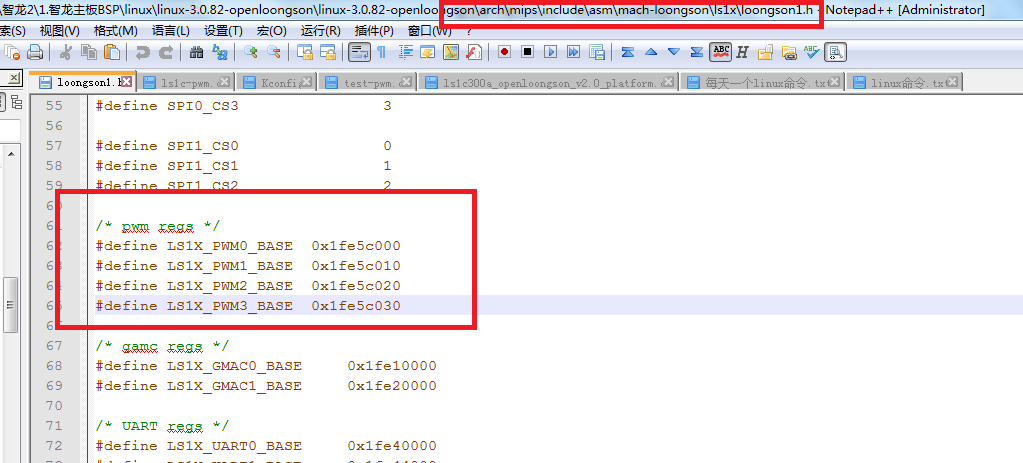
(1) PWM0 PWM1 说明

原始功能没有复用

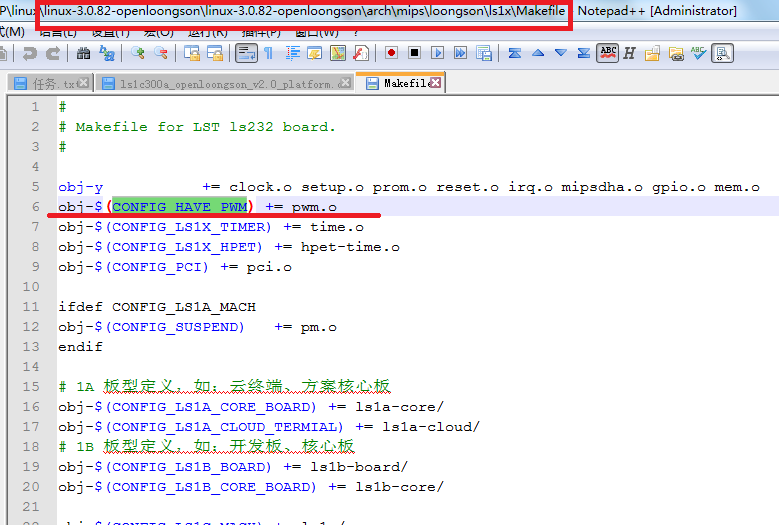


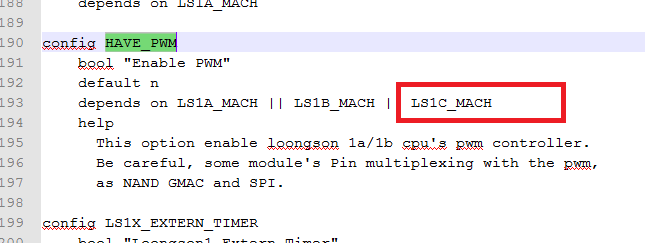


LS1X\_PWM0\_BASE 0x1fe5c000 此处是物理地址0xbfefc0000 为核心虚地址空间的程序空间地址

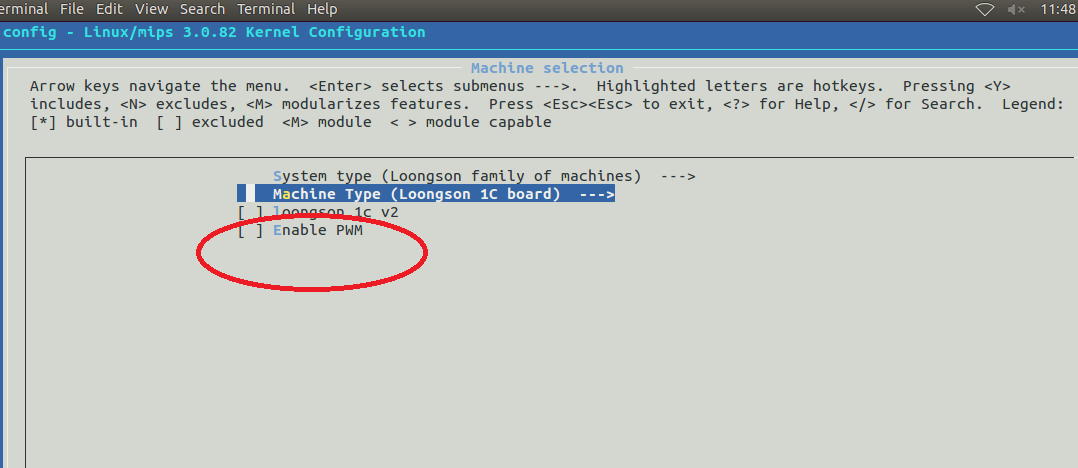


使用的平台文件 pwm.c 中

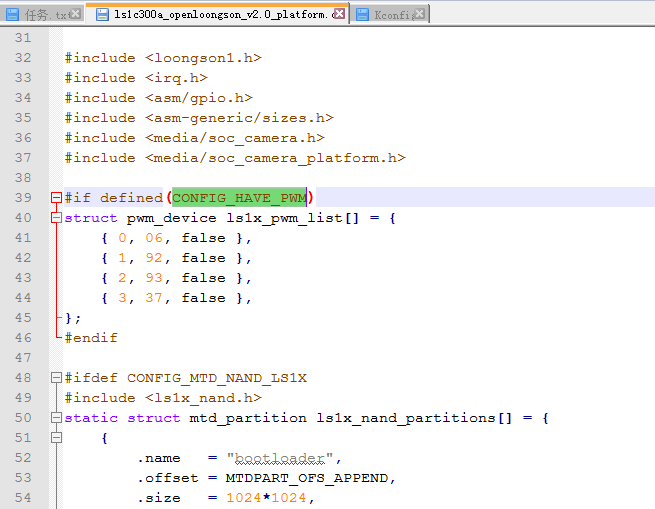




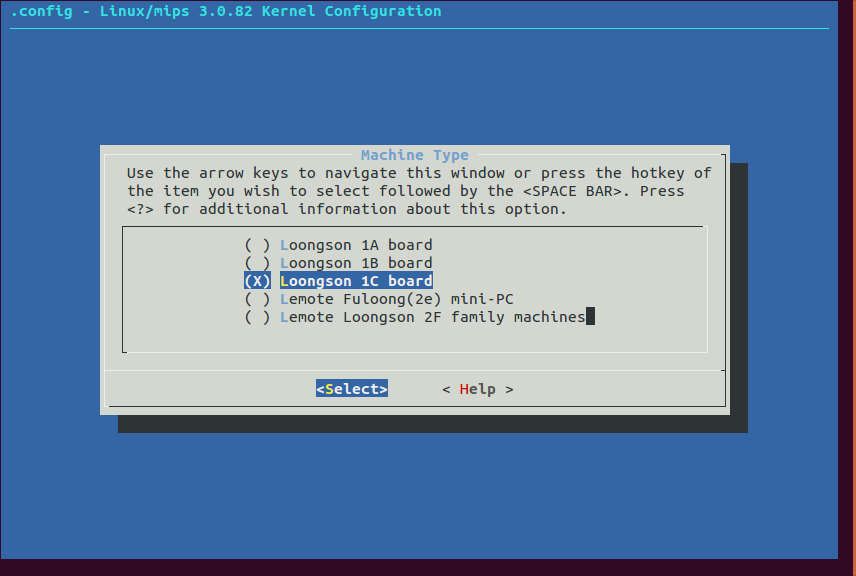
这就需要在内核配置时，添加以下选项：

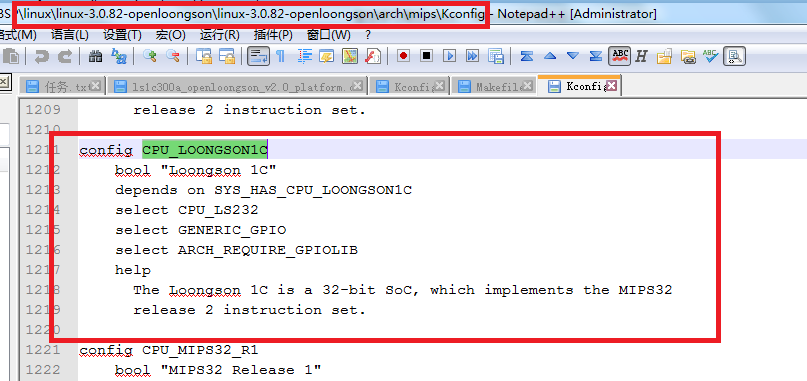


添加了HAVE\_PWM后，就有了 平台设备pwm\_device的定义 ls1x\_pwm\_list

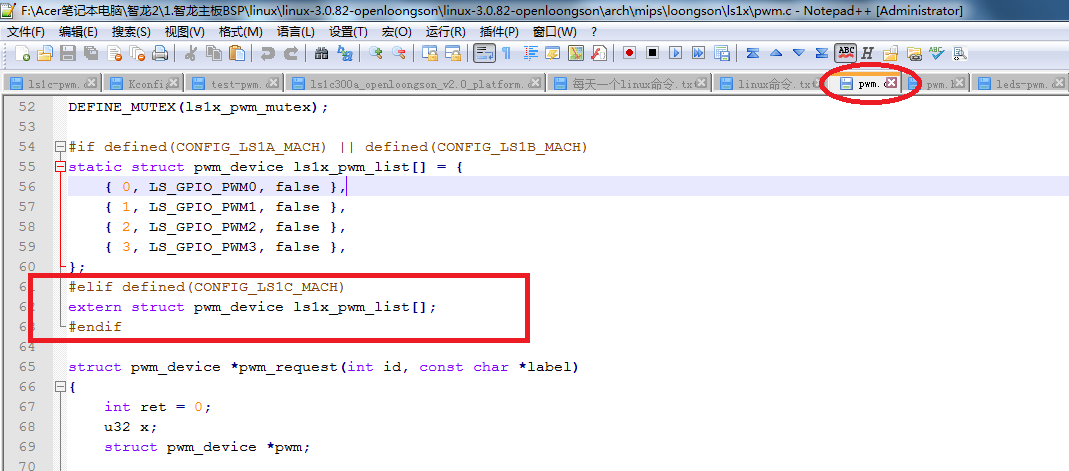


(2)内核中选择Loongson 1C board的CPU

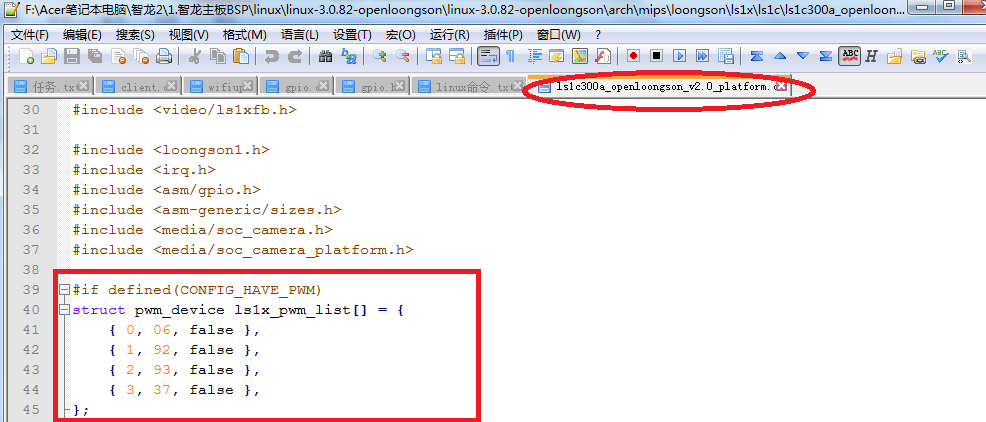




则在pwm.c中定义了外部的ls1x\_pwm\_list，分析驱动pwm.c

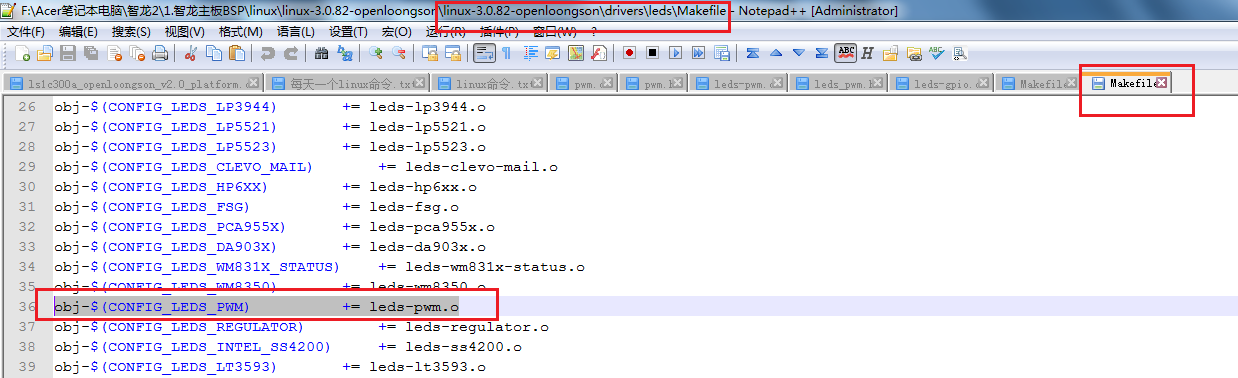


这个变量在 平台文件中定义，分析ls1c300a\_openloongson\_v2.0\_platform.c

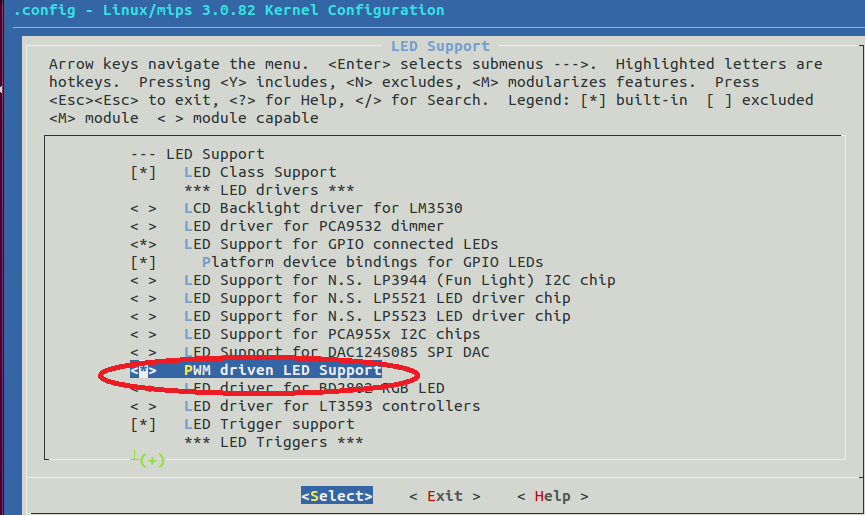


下面看leds\_pwm：

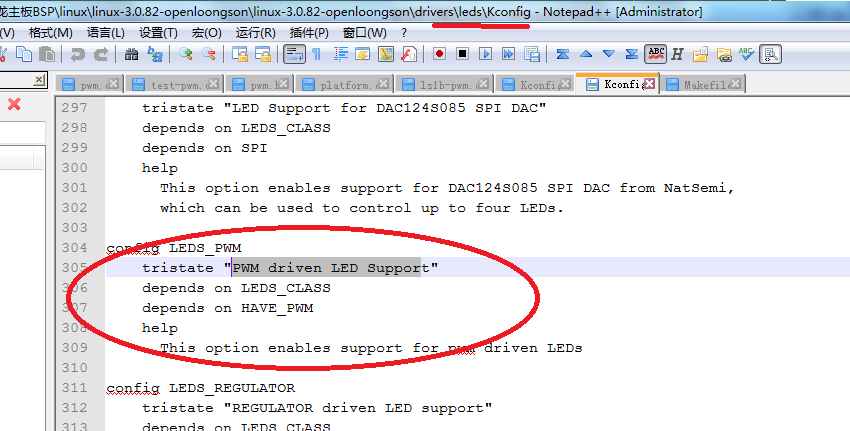
在drivers/leds/ 下的makefile, leds\_pwm的编译依赖于 CONFIG\_LEDS\_PWM.



CONFIG\_LEDS\_PWM 在内核配置中如下配置

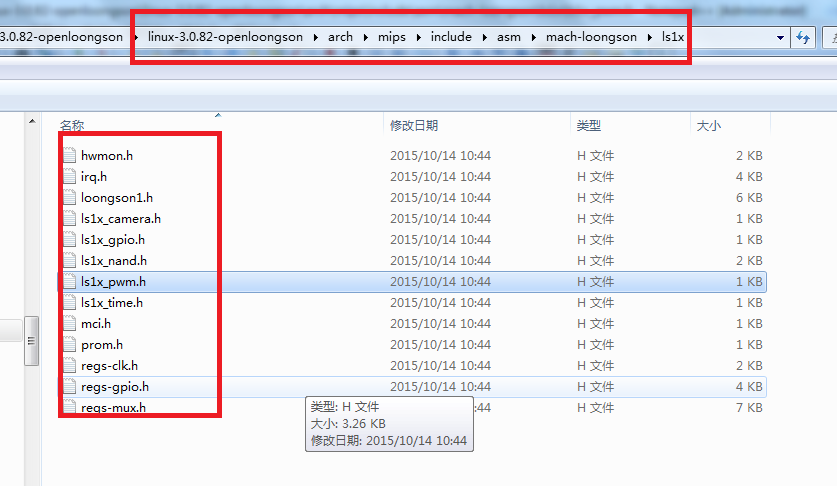


该配置选项的出现依赖于 前图中的 HAVE\_PWM配置项



其中， struct pwm\_device 在文件

linux-3.0.82-openloongson\linux-3.0.82-openloongson\arch\mips\include\asm\mach-loongson\ls1x\ls1x\_pwm.h



#ifndef \_\_ASM\_ARCH\_LS1X\_PWM\_H

#define \_\_ASM\_ARCH\_LS1X\_PWM\_H

struct pwm\_device {

unsigned int id;

unsigned int gpio;

bool used;

};

#endif /\* \_\_ASM\_ARCH\_LS1X\_PWM\_H \*/

（3）Linux内核配置选中pwm驱动

Machine selection --->

System type(Loongson family of machines) --->

Macine Type(Loongson 1C board) --->

[ ] loongson 1c v2

[\*] Enable PWM

在LED类中配置PWM

Device Drivers --->

[\*] LED Support --->

<\*> PWM driven LED Support

在平台文件ls1c300a\_openloongson\_v2.0\_platform.c 中添加：

#ifdef CONFIG\_LEDS\_PWM

static struct led\_pwm ls1x\_pwm\_leds[] = {

{

.name = "ls1x\_pwm\_led1",

.pwm\_id = 0,

.max\_brightness = 255,

.pwm\_period\_ns = 7812500,

},

{

.name = "ls1x\_pwm\_led2",

.pwm\_id = 1,

.max\_brightness = 255,

.pwm\_period\_ns = 7812500,

},

};

static struct led\_pwm\_platform\_data ls1x\_pwm\_data = {

.num\_leds = ARRAY\_SIZE(ls1x\_pwm\_leds),

.leds = ls1x\_pwm\_leds,

};

static struct platform\_device ls1x\_leds\_pwm = {

.name = "leds\_pwm",

.id = -1,

.dev = {

.platform\_data = &ls1x\_pwm\_data,

},

};

#endif //#ifdef CONFIG\_LEDS\_PWM

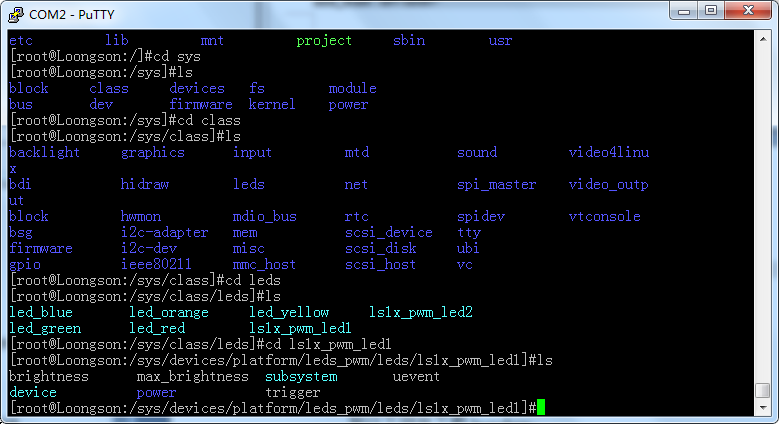
上一个平台文件中，函数 static struct platform\_device \*ls1b\_platform\_devices[] \_\_initdata 中添加：

#ifdef CONFIG\_LEDS\_PWM

&ls1x\_leds\_pwm,

#endif

平台文件增加成功：leds\_pwm



将 PWM0 占比调到最大(高电平占100%)

echo 255 > brightness

将 PWM0 占比调到最小(低电平占100%)

echo 0 > brightness

## 3.2自己编写驱动文件

首先在平台文件中添加资源：

平台文件\arch\mips\loongson\ls1x\ls1c\ls1c300a\_openloongson\_v2.0\_platform.c添加：

#ifdef CONFIG\_LS1C\_PWM\_DRIVER

static struct resource ls1c\_pwm0\_resource[] = {

[0]={

.start = LS1X\_PWM0\_BASE,

.end = (LS1X\_PWM0\_BASE + 0x0f),

.flags = IORESOURCE\_MEM,

},

[1]={

.start = LS1X\_PWM1\_BASE,

.end = (LS1X\_PWM1\_BASE + 0x0f),

.flags = IORESOURCE\_MEM,

},

[2]={

.start = LS1X\_PWM2\_BASE,

.end = (LS1X\_PWM2\_BASE + 0x0f),

.flags = IORESOURCE\_MEM,

},

[3]={

.start = LS1X\_PWM3\_BASE,

.end = (LS1X\_PWM3\_BASE + 0x0f),

.flags = IORESOURCE\_MEM,

},

};

static struct platform\_device ls1c\_pwm\_device = {

.name = "ls1c-pwm",

.id = -1,

.num\_resources = ARRAY\_SIZE(ls1c\_pwm0\_resource),

.resource = ls1c\_pwm0\_resource,

};

#endif //#ifdef CONFIG\_LS1C\_PWM\_DRIVER

上一个文件中，函数 static struct platform\_device \*ls1b\_platform\_devices[] \_\_initdata 中添加：

#ifdef CONFIG\_LS1C\_PWM\_DRIVER

&ls1c\_pwm\_device,

#endif

在 \drivers\char\Kconfig中添加：

config LS1C\_PWM\_DRIVER

bool "ls1c pwm driver"

default n

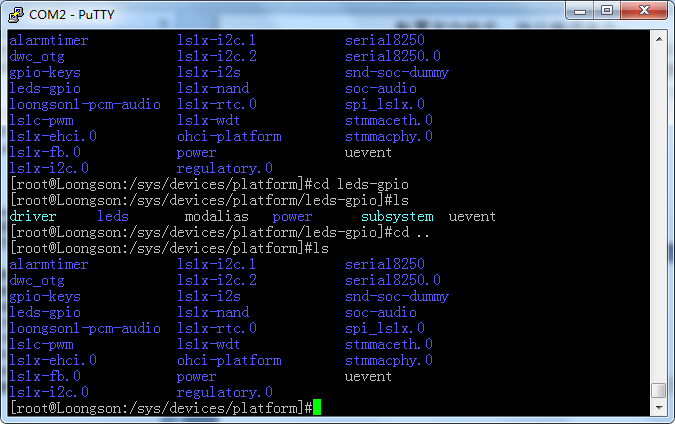
help

pwm.

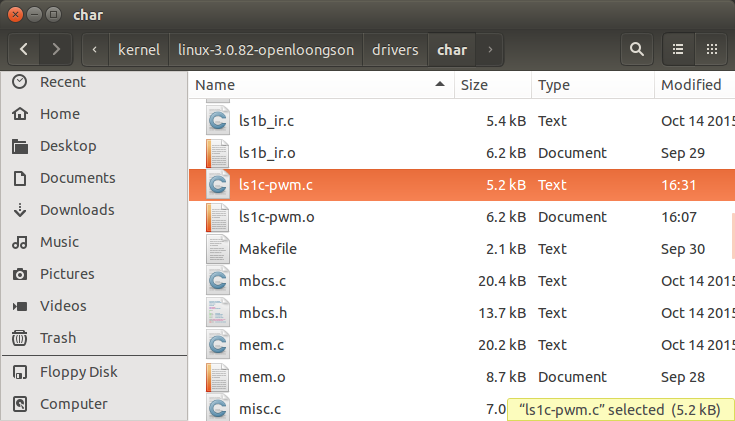
在 char 下的Makefile文件中添加语句：

obj-$(CONFIG\_LS1C\_PWM\_DRIVER) += ls1c-pwm.o

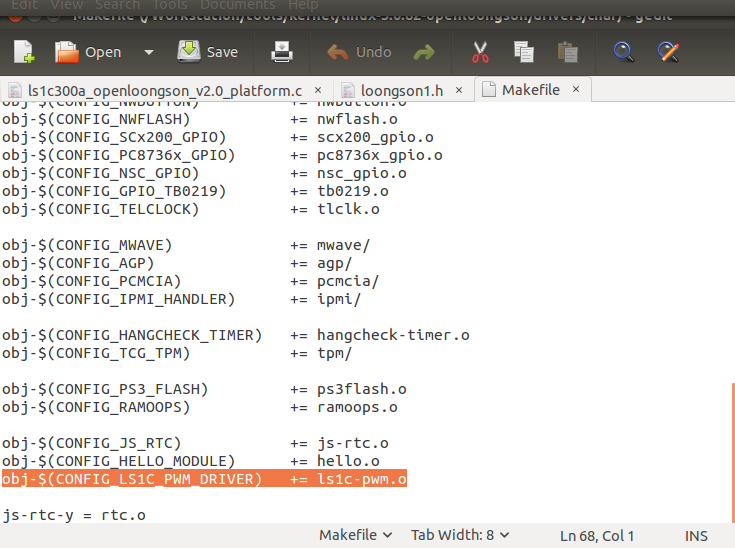
平台设备增加成功：ls1c-pwm



编写驱动文件 ls1c\_pwm.c（根据1B中的资料修改）， 放到内核代码driver/char下，



修改Makefile文件，添加语句：obj-$(CONFIG\_LS1C\_PWM\_DRIVER) += ls1c-pwm.o



以下 设备驱动ls1c\_pwm.c 代码：

/\*ls1c\_pwm.c\*/

#include <linux/miscdevice.h>

#include <linux/errno.h>

#include <linux/fs.h>

#include <asm/io.h>

#include <asm/uaccess.h>

#include <asm/ioctl.h>

#include <linux/platform\_device.h>

#include <linux/timer.h>

#define CMD\_PWM\_GET \_IO('c',0x01)

#define CMD\_PWM\_START \_IO('c',0x02)

#define CMD\_PWM\_STOP \_IO('c',0x03)

#define REG\_PWM\_CNTR 0x00 // 0xBFE5C010

#define REG\_PWM\_HRC 0x04 // 0xBFE5C014

#define REG\_PWM\_LRC 0x08 // 0xBFE5C018

#define REG\_PWM\_CTRL 0x0c // 0xBFE5C01C

#define REG\_GPIO\_CFG0 0xbfd010c0//GPIO GPIO[31:0] 配置寄存器 1 表示配置为GPIO， 0 表示无效

#define REG\_GPIO\_CFG2 0xbfd010c8//GPIO[95:64] 配置寄存器 1 表示配置为GPIO， 0 表示无效

#define SB2F\_GPIO\_MUX\_CTRL1 0xbfd00424//功能复用寄存器

static unsigned char \_\_iomem \*pwm\_base = NULL;

struct resource \*res;

static int ls1f\_pwm\_probe(struct platform\_device \*pdev);

struct platform\_device \*pwm\_dev;

struct resource \*res1 = NULL;

static int ls1f\_pwm\_getResourse(struct platform\_device \*pdev, unsigned int index);

static struct platform\_driver ls1f\_pwm\_driver = {

.probe = ls1f\_pwm\_probe,

.driver = {

.name = "ls1c-pwm",

},

};

static int ls1f\_pwm\_open(struct inode \*inode, struct file \*file)

{

//配置PWM0 PWM1 原始功能没有复用

//long val = readl(SB2F\_GPIO\_MUX\_CTRL1);

//val &= 0xfffffffc;

//writel(val, SB2F\_GPIO\_MUX\_CTRL1);

long val = readl(REG\_GPIO\_CFG0);

//配置GPIO6引脚为普通功能，而非GPIO功能

val &= 0xffffffbf;

writel(val, REG\_GPIO\_CFG0);

val = readl(REG\_GPIO\_CFG0);

//配置GPIO92引脚为普通功能，而非GPIO功能

val &= 0xefffffff;

writel(val, REG\_GPIO\_CFG2);

return 0;

}

static int ls1f\_pwm\_close(struct inode \*inode, struct file \*file)

{

writel(0x0, pwm\_base + REG\_PWM\_CTRL);

return 0;

}

static ssize\_t ls1f\_pwm\_read(struct file \*file, char \_\_user \*buf, size\_t count, loff\_t \*ptr)

{

unsigned int pwm\_val;

pwm\_val = readl(pwm\_base);

if (copy\_to\_user(buf, &pwm\_val, sizeof(unsigned int)))

return -EFAULT;

return 4;

}

static ssize\_t ls1f\_pwm\_write(struct file \*file, const char \_\_user \*buf, size\_t count, loff\_t \*ptr)

{

unsigned int hrc\_val, lrc\_val;

unsigned int data[2] = {0x0};

if (copy\_from\_user(data, buf, sizeof(data)))

{

printk("Write error!\n");

return -EIO;

}

hrc\_val = data[1] - 1;

lrc\_val = data[0] + data[1] -1;

//设置占空比

writel(hrc\_val, pwm\_base + REG\_PWM\_HRC);

writel(lrc\_val, pwm\_base + REG\_PWM\_LRC);

printk("hrc:%i ; lrc:%i\n",hrc\_val,lrc\_val);

return 0;

}

static int pwm\_start\_stop(unsigned int cmd, unsigned long arg)

{

printk("into: %s\n", \_\_FUNCTION\_\_);

printk("arg: %ld\n", arg);

if (arg > 3)

return -1;

//从platform中获取指定PWM的寄存器基址 pwm\_base

//通过改变arg，实现在pwm0、pwm1、pwm2、pwm3之间切换

ls1f\_pwm\_getResourse(pwm\_dev, arg);

switch (cmd) {

//启动PWM

case CMD\_PWM\_START:

printk("CMD\_PWM\_START\n");

writel(0x0, pwm\_base + REG\_PWM\_CNTR);

writel(0x01, pwm\_base + REG\_PWM\_CTRL);

break;

//停止PWM

case CMD\_PWM\_STOP:

printk("CMD\_PWM\_STOP\n");

writel(0x0, pwm\_base + REG\_PWM\_CTRL);

break;

default:

break;

}

return 0;

}

static long ls1f\_pwm\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg)

{

printk("into %s\n", \_\_FUNCTION\_\_);

printk("cmd: %d\n", cmd);

printk("arg: %ld\n", arg);

if (arg > 3)

return -1;

switch (cmd) {

case CMD\_PWM\_GET:

ls1f\_pwm\_getResourse(pwm\_dev, arg);

break;

case CMD\_PWM\_START:

case CMD\_PWM\_STOP:

return pwm\_start\_stop(cmd, arg);

default:

break;

}

return 0;

}

static const struct file\_operations ls1f\_pwm\_ops = {

.owner = THIS\_MODULE,

.open = ls1f\_pwm\_open,

.release = ls1f\_pwm\_close,

.read = ls1f\_pwm\_read,

.write = ls1f\_pwm\_write,

.unlocked\_ioctl = ls1f\_pwm\_ioctl,

};

static struct miscdevice ls1f\_pwm\_miscdev = {

MISC\_DYNAMIC\_MINOR,

"ls1f-pwm",

&ls1f\_pwm\_ops,

};

static int ls1f\_pwm\_getResourse(struct platform\_device \*pdev, unsigned int index)

{

res = platform\_get\_resource(pdev, IORESOURCE\_MEM, index);

if (res == NULL)

{

printk("Fail to get ls1f\_pwm\_resource!\n");

return -ENOENT;

}

printk("Resource start=0x%x, end = 0x%x\n", res->start, res->end);

if (res1 != NULL)

{

release\_mem\_region(res->start, 0x0f);

}

res1 = request\_mem\_region(res->start, 0x0f, "ls1f-pwm");

if (res1 == NULL)

{

printk("Fail to request ls1f\_pwm region!\n");

return -ENOENT;

}

pwm\_base = ioremap(res->start, res->end - res->start + 1);

if (pwm\_base == NULL)

{

printk("Fail to ioremap ls1f\_pwm resource!\n");

return -EINVAL;

}

return 0;

}

static int \_\_devinit ls1f\_pwm\_probe(struct platform\_device \*pdev)

{

pwm\_dev = pdev;

return ls1f\_pwm\_getResourse(pdev, 1);

}

static int \_\_init ls1f\_pwm\_init(void) {

if (misc\_register(&ls1f\_pwm\_miscdev))

{

printk(KERN\_WARNING "pwm: Couldn't register device 10, %d.\n", 255);

return -EBUSY;

}

return platform\_driver\_register(&ls1f\_pwm\_driver);

}

static void \_\_exit ls1f\_pwm\_exit(void)

{

misc\_deregister(&ls1f\_pwm\_miscdev);

release\_mem\_region(res->start, 0x20);

platform\_driver\_unregister(&ls1f\_pwm\_driver);

}

module\_init(ls1f\_pwm\_init);

module\_exit(ls1f\_pwm\_exit);

MODULE\_AUTHOR("sundm");

MODULE\_DESCRIPTION("loongson 1C PWM driver");

MODULE\_LICENSE("GPL");

编译成功后，编写测试代码：

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#define CMD\_PWM\_GET \_IO('c',0x01)

#define CMD\_PWM\_START \_IO('c',0x02)

#define CMD\_PWM\_STOP \_IO('c',0x03)

//sel为选择PWM通道(0或者1);value为占空比(0-100); freq为频率().

int pwmset(unsigned int sel, unsigned int value, unsigned int freq) {

int fd;

//data[0]的值代表的是高电平脉冲所占的时钟数，data[1]的值代表的是低电平脉冲所占的时钟数

unsigned int data[2] = {0x7ffffff,0x7ffffff};

if(value>100)

value = 100;

data[0] = freq / 100 \* value;

data[1] = freq / 100 \* (100-value);

printf("cys: ready to open!\n");

fd = open("/dev/ls1f-pwm",O\_RDWR);

sleep(2);

if (fd != -1)

{

printf("open ok!\n");

//共有四路pwm(pwm0...pwm3)，四路pwm可同时工作，也可选择使用其中任何一路

if (sel <0 || sel > 3)

return -1;

switch(sel) {

case 0:

//在驱动中没有用到ioctl的第三个参数，故直接赋予0就可以啦

ioctl(fd, CMD\_PWM\_START , 0);

write(fd, data, sizeof(data) );

break;

case 1:

ioctl(fd, CMD\_PWM\_START , 1);

write(fd, data, sizeof(data) );

break;

case 2:

ioctl(fd, CMD\_PWM\_START , 2);

write(fd, data, sizeof(data) );

break;

case 3:

ioctl(fd, CMD\_PWM\_START , 3);

write(fd, data, sizeof(data) );

break;

default:

break;

}

}

else

{

printf("Device open failure\n");

}

sleep(10);

close(fd);

return 0;

}

调用函数 pwmset(0,50,10000) 意思是在PWM0引脚产生占空比 50%, 周期10000时钟周期的波形。

GPIO92与其中一个KEY冲突，在平台文件中注释掉一个KEY 92

