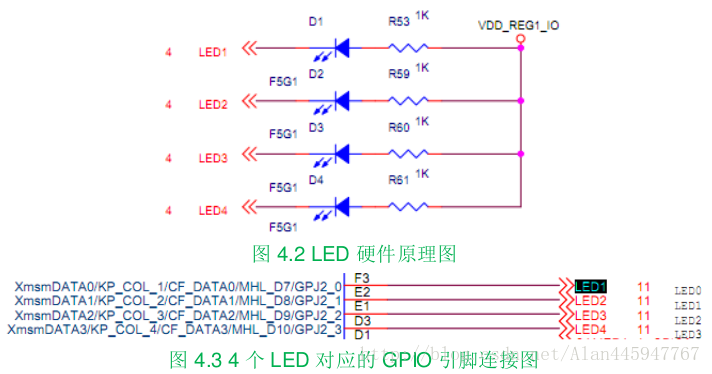
<http://blog.csdn.net/alan445947767/article/details/40536449>

LED 原理图都网蜂科技的 Webee210 核心板上,如下:



由以上两个图可知,LED1~LED4 对应的 GPIO 口为 GPJ2\_0 ~ GPJ2\_3,如果要让 LED1 亮,则设置 GPJ2CON[3:0]为输出引脚,设置 GPJ2DAT[0]为输出低电平,即可点亮 LED1 啦,就这么简单^\_^。

### LED 驱动实例一源码分析

LED 驱动实例一源码在 webee210\_drivers\2th\_led\led.c

#### 模块的入口函数分析

分析一个驱动首先从入口函数开始,而 Webee210\_led\_init 作为模块的入口函数,主要做了以下几件事:

第一、使用 register\_chrdev 注册字符设备,这个函数是内核提供的接口,内核已经帮我们实现了,不用我们操心。(注意了,有些 Linux 发行版本有可能会使用 register\_chrdev\_region 函数来注册字符设备。)

第二、使用内核提供的 class\_create、device\_create 函数分别创建类、在类下创建设备。  
第三、使用内核提供的 ioremap 函数映射 IO 地址,只有映射了,才能够在内核环境下使用这些 IO 地址。

/\* 驱动程序的入口函数 \*/

static int \_\_init Webee210\_led\_init(void)

{

/\* 注册字符设备,第一个参数设置为 0 表示由系统自动分配主设备号 \*/

major = register\_chrdev(0, "led\_drv", &led\_fops);

/\* 创建 led\_drv 类 \*/

led\_class = class\_create(THIS\_MODULE, "led\_drv");

/\* 在 led\_drv 类下创建/dev/LED 设备,供应用程序打开设备\*/

led\_device = device\_create(led\_class, NULL,MKDEV(major,0),DEVICE\_NAME);

/\* 将物理地址映射为虚拟地址 \*/

gpj2con = (volatile unsigned long \*)ioremap(0xE0200280, 16);

gpj2dat = gpj2con + 1;

return 0;

}

#### 模块的出口函数分析

驱动的出口函数的主要工作一般是注销 XXX、卸载 XXX,等等类似的清除工作,而 Webee210\_led\_exit 作为模块的出口函数,主要做了以下几件事:  
第一、使用内核提供的 unregister\_chrdev 函数注销字符设备  
第二、使用内核提供的 device\_unregister、class\_destroy 函数分别卸载类下的设备、卸载类。  
第三、使用内核提供的 iounmap 函数解除映射。

一目了然,这些都是一些清除工作。

/\* 驱动程序的出口函数 \*/

static void \_\_exit Webee210\_led\_exit(void)

{

unregister\_chrdev(major,"led\_drv"); /\* 注销字符设备 \*/

device\_unregister(led\_device);/\* 卸载类下的设备 \*/

class\_destroy(led\_class);/\* 卸载类 \*/

iounmap(gpj2con);/\* 解除映射 \*/

}

#### file\_operations 结构体

在入口函数注册字符设备的时候,register\_chrdev 函数的第三个参数就需要file\_operations 的实例。

major = register\_chrdev(0, "led\_drv", &led\_fops);

/\* file\_operations 的实例 \*/

static const struct file\_operations led\_fops = {

.owner = THIS\_MODULE,

.open = webee210\_led\_open,

.write = webee210\_led\_write,

};

#### led\_fops 结构体成员函数

led\_fops 结构体的主要成员有 webee210\_led\_open、webee210\_led\_write,

这两个函数正是该 LED 驱动的重点对象。

static volatile unsigned long \*gpj2con = NULL;

static volatile unsigned long \*gpj2dat = NULL;

/\* 应用程序执行 open 时,最终调用到驱动程序的 webee210\_led\_open 函数 \*/

static int webee210\_led\_open(struct inode \* inode, struct file \* filp)

{

printk("webee210\_led\_open\n");

/\* Webee210 开发板上的 LED1,LED2,LED3,LED4

\* 对应 GPJ2\_0、GPJ2\_1、GPJ2\_2、GPJ2\_3 引脚

\* 配置 GPJ2\_0、GPJ2\_1、GPJ2\_2、GPJ2\_3 为输出

\*/

\*gpj2con |= S5PV210\_GPJ2\_0\_OUTP|S5PV210\_GPJ2\_1\_OUTP|S5PV210\_GPJ2\_2\_OUTP|S5PV210\_GPJ2\_3\_OUTP;

return 0;

}

/\* 应用程序执行 write 时,最终调用到驱动程序的 webee210\_led\_write 函数 \*/

static int webee210\_led\_write(struct file \* file, const char \_\_user \* buffer, size\_t count, loff\_t \* ppos)

{

int val;

printk("webee210\_led\_write\n");

copy\_from\_user(&val, buffer, count);

if (val == 1)

{

/\* 点灯 \*/

\*gpj2dat &= ~((1<<0) | (1<<1) | (1<<2) | (1<<3));

}

else

{

/\* 灭灯 \*/

\*gpj2dat |= (1<<0) | (1<<1) | (1<<2) | (1<<3);

}

return 0;

}

webee210\_led\_open 函数很简单,只是将 LED1-4 对应的 GPIO 管脚设置为输出功能。

webee210\_led\_write 函数首先通过内核提供的 copy\_from\_user 函数,将应用程序传递过来 count 字节的 buf 数据写进 val。( 注意:应用空间和内核空间的包括指针数据 的传递不能单纯用 memcpy 函数,必须使用 copy\_from\_user函数或 copy\_to\_user 函数)。

< driver / led.c >

/\*

\* Name:led.c

\* Copyright (C) 2014 Webee.JY (2483053468@qq.com)

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License version 2 as

\* published by the Free Software Foundation.

\*/

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/fs.h>

#include <linux/init.h>

#include <linux/delay.h>

#include <asm/uaccess.h>

#include <asm/irq.h>

#include <asm/io.h>

#include <linux/device.h> //class\_create

#define DEVICE\_NAME "LED"

#define S5PV210\_GPJ2\_0\_OUTP (1<<0)

#define S5PV210\_GPJ2\_1\_OUTP (1<<4)

#define S5PV210\_GPJ2\_2\_OUTP (1<<8)

#define S5PV210\_GPJ2\_3\_OUTP (1<<12)

static struct class \*led\_class;

static struct device \*led\_device;

int major;

static volatile unsigned long \*gpj2con = NULL;

static volatile unsigned long \*gpj2dat = NULL;

/\* 应用程序执行open时，最终调用到驱动程序的webee210\_led\_open函数 \*/

static int webee210\_led\_open(struct inode \* inode, struct file \* filp)

{

printk("webee210\_led\_open\n");

/\* Webee210开发板上的LED1,LED2,LED3,LED4

\* 对应GPJ2\_0、GPJ2\_1、GPJ2\_2、GPJ2\_3引脚

\* 配置GPJ2\_0、GPJ2\_1、GPJ2\_2、GPJ2\_3为输出

\*/

\*gpj2con |= S5PV210\_GPJ2\_0\_OUTP|S5PV210\_GPJ2\_1\_OUTP|S5PV210\_GPJ2\_2\_OUTP|S5PV210\_GPJ2\_3\_OUTP;

return 0;

}

/\* 应用程序执行write时，最终调用到驱动程序的webee210\_led\_write函数 \*/

static int webee210\_led\_write(struct file \* file, const char \_\_user \* buffer, size\_t count, loff\_t \* ppos)

{

int val;

printk("webee210\_led\_write\n");

copy\_from\_user(&val, buffer, count);

if (val == 1)

{

/\* 点灯 \*/

\*gpj2dat &= ~((1<<0) | (1<<1) | (1<<2) | (1<<3));

}

else

{

/\* 灭灯 \*/

\*gpj2dat |= (1<<0) | (1<<1) | (1<<2) | (1<<3);

}

return 0;

}

static const struct file\_operations led\_fops = {

.owner = THIS\_MODULE,

.open = webee210\_led\_open,

.write = webee210\_led\_write,

};

/\* 驱动程序的入口函数 \*/

static int \_\_init Webee210\_led\_init(void)

{

/\* 注册字符设备，第一个参数设置为0表示由系统自动分配主设备号 \*/

major = register\_chrdev(0, "led\_drv", &led\_fops);

/\* 创建led\_drv类 \*/

led\_class = class\_create(THIS\_MODULE, "led\_drv");

/\* 在led\_drv类下创建/dev/LED设备，供应用程序打开设备\*/

led\_device = device\_create(led\_class, NULL, MKDEV(major, 0), NULL, DEVICE\_NAME);

/\* 将物理地址映射为虚拟地址 \*/

gpj2con = (volatile unsigned long \*)ioremap(0xE0200280, 16);

gpj2dat = gpj2con + 1;

return 0;

}

/\* 驱动程序的出口函数 \*/

static void \_\_exit Webee210\_led\_exit(void)

{

unregister\_chrdev(major,"led\_drv"); /\* 注销字符设备 \*/

device\_unregister(led\_device); /\* 卸载类下的设备 \*/

class\_destroy(led\_class); /\* 卸载类 \*/

iounmap(gpj2con); /\* 解除映射 \*/

}

/\* 用于修饰入口/出口函数，换句话说，相当于

\* 告诉内核驱动程序的入口/出口函数在哪里

\*/

module\_init(Webee210\_led\_init);

module\_exit(Webee210\_led\_exit);

/\* 该驱动支持的协议、作者、描述 \*/

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("webee");

MODULE\_DESCRIPTION("Character drivers for leds");

< Makefile >

ifneq ($(KERNELRELEASE),)

obj-m :=led.o

else

module-objs :=led.o

KERNELDIR :=/home/gec/linux\_kernel/linux2.6.35.7/

PWD :=$(shell pwd)

default:

$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

clean:

$(RM) \*.ko \*.mod.c \*.mod.o \*.o \*.order \*.symvers \*.cmd

## LED 驱动实例二

### LED 驱动实例二源码分析

LED 驱动实例二源码在 webee210\_drivers\3th\_led\_misc\webee210\_led.c

### 模块的入口函数分析

分析一个驱动首先从入口函数开始,而 Webee210\_led\_init 作为模块的入口函数,主要做了以下几件事:

第一、使用内核提供的 gpio\_direction\_output 函数将对应的 GPIO 管脚设置为输出,并使用内核提供的 gpio\_set\_value 函数将对应的 GPIO 管脚输出高电平。这二步结合起来的效果就是,先关闭 webee210 开发板上的 4 个LED。

第二、使用内核提供的 misc\_register 函数注册一个混杂(misc)设备,也可以成为字符设备。

static int \_\_init Webee210\_led\_init(void)

{

int ret;

int i;

for (i = 0; i < 4; i++)

{

/\* 使用 gpio\_direction\_output 函数前必须调用 gpio\_request \*/

gpio\_request(gpio\_table[i], NULL);

/\* 配置 GPIO 管脚为输出 \*/

gpio\_direction\_output(gpio\_table[i], gpio\_cfg\_table[i]);

/\* 关闭所有 LED,如果想初始化为点亮 LED,则将第二个参数改为 0 \*/

gpio\_set\_value(gpio\_table[i],1);

}

/\* 注册一个主设备号为 10,次设备号为动态分配的混杂(字符)设备 \*/

ret = misc\_register(&misc);

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

return ret;

}

### 模块的出口函数分析

Webee210\_led\_exit 作为该驱动的出口函数,只注销混杂设备。

static void \_\_exit Webee210\_led\_exit(void)  
{  
 misc\_deregister(&misc);  
}

#### miscdevice 结构体

在入口函数注册混杂(misc)设备的时候,misc\_register 函数的参数就需要miscdevice 的实例。

static struct miscdevice misc = {

.minor = MISC\_DYNAMIC\_MINOR,

/\* 动态分配次设备号 \*/

.name = DEVICE\_NAME,

/\* 设备节点为/dev/WEBEE210\_LED \*/

.fops = &led\_fops,

/\* 文件操作函数集 \*/

};

细心的读者会发现,misc 结构体里面有个 file\_operations 成员。这又回到了file\_operations 结构体了。

static struct file\_operations led\_fops = {

.owner

= THIS\_MODULE,

.unlocked\_ioctl = webee210\_led\_ioctl,

};

#### led\_fops 结构体成员函数

led\_fops 结构体的主要成员只有 webee210\_led\_ioctl,这个函数正是该 LED驱动的重点对象。

/\* 应用程序执行 ioctl 时,最终调用到驱动程序的 webee210\_led\_ioctl 函数 \*/

static long webee210\_led\_ioctl (struct file \*filp, unsigned int cmd, unsigned

long arg)

{

if (arg > 4)

{

return -EINVAL;

}

switch(cmd)

{

/\* 当应用程序传入 1 时,LED 被点亮 \*/

case IOCTL\_GPIO\_ON:

/\* 设置 GPIO 引脚的输出电平为低电平 \*/

gpio\_set\_value(gpio\_table[arg],0);

return 0;

/\* 当应用程序传入 0 时,LED 被熄灭 \*/

case IOCTL\_GPIO\_OFF:

/\* 设置 GPIO 引脚的输出电平为高电平 \*/

gpio\_set\_value(gpio\_table[arg],1);

return 0;

default:

return -EINVAL;

}

}

webee210\_led\_ioctl 函数根据应用程序传递进来的 cmd 命令进行分析,然后调用内核提供的 gpio\_set\_value 函数去设置 GPIO 管脚的输出电平,也就是控制 LED 管脚的高低电平了。

### LED 驱动编译与测试

在 webee210\_led.c 文件里,如果缺少第 92 行,在加载驱动的时候就很可能会出现 BUG,但居然不影响使用。

gpio\_request(gpio\_table[i], NULL);如果没有上面这行,在第一次加载驱动的时候,就可能会出现下面这样的BUG , 究 其 原 因 就 是 调 用 gpio\_direction\_output 函 数 前 , 必 须 先 调 用gpio\_request 函 数 。 如 果 不 信 , 你 可 以 在 Source Insight 里 搜 索

gpio\_direction\_output 函数,会发现任何一个调用 gpio\_direction\_output 函数的文件,事先都有调用 gpio\_request 函数。当然了,这个 BUG 的解决就是在第92 行,添加上 gpio\_request 函数的调用。

< driver / webee210\_led.c >

/\*

\* Name:Webee210\_led.c

\* Copyright (C) 2014 Webee.JY (2483053468@qq.com)

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License version 2 as

\* published by the Free Software Foundation.

\*/

#include <linux/miscdevice.h>

#include <linux/gpio.h>

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/fs.h>

#define DEVICE\_NAME "WEBEE210\_LED"

#define S5PV210\_GPJ2\_0\_OUTP (1<<0)

#define S5PV210\_GPJ2\_1\_OUTP (1<<4)

#define S5PV210\_GPJ2\_2\_OUTP (1<<8)

#define S5PV210\_GPJ2\_3\_OUTP (1<<12)

/\* ioctl的命令参数 \*/

#define IOCTL\_GPIO\_ON 1

#define IOCTL\_GPIO\_OFF 0

/\* Webee210的LED1~4对应的GPIO引脚 \*/

static unsigned long gpio\_table [] =

{

S5PV210\_GPJ2(0),

S5PV210\_GPJ2(1),

S5PV210\_GPJ2(2),

S5PV210\_GPJ2(3),

};

/\* 将对应的GPIO引脚设置为输出 \*/

static unsigned int gpio\_cfg\_table [] =

{

S5PV210\_GPJ2\_0\_OUTP,

S5PV210\_GPJ2\_1\_OUTP,

S5PV210\_GPJ2\_2\_OUTP,

S5PV210\_GPJ2\_3\_OUTP,

};

/\* 应用程序执行ioctl时，最终调用到驱动程序的webee210\_led\_ioctl函数 \*/

static long webee210\_led\_ioctl (struct file \*filp, unsigned int cmd, unsigned long arg)

{

if (arg > 4)

{

return -EINVAL;

}

switch(cmd)

{

/\* 当应用程序传入1时，LED被点亮 \*/

case IOCTL\_GPIO\_ON:

/\* 设置GPIO引脚的输出电平为低电平 \*/

gpio\_set\_value(gpio\_table[arg],0);

return 0;

/\* 当应用程序传入0时，LED被熄灭 \*/

case IOCTL\_GPIO\_OFF:

/\* 设置GPIO引脚的输出电平为高电平 \*/

gpio\_set\_value(gpio\_table[arg],1);

return 0;

default:

return -EINVAL;

}

}

static struct file\_operations led\_fops = {

.owner = THIS\_MODULE,

.unlocked\_ioctl = webee210\_led\_ioctl,

};

static struct miscdevice misc = {

.minor = MISC\_DYNAMIC\_MINOR, /\* 动态分配次设备号 \*/

.name = DEVICE\_NAME, /\* 设备节点为/dev/WEBEE210\_LED \*/

.fops = &led\_fops, /\* 文件操作函数集 \*/

};

/\* 驱动程序的入口函数 \*/

static int \_\_init Webee210\_led\_init(void)

{

int ret;

int i;

for (i = 0; i < 4; i++)

{

/\* 使用gpio\_direction\_output函数前必须调用gpio\_request \*/

gpio\_request(gpio\_table[i], NULL);

/\* 配置GPIO管脚为输出 \*/

gpio\_direction\_output(gpio\_table[i], gpio\_cfg\_table[i]);

/\* 关闭所有LED，如果想初始化为点亮LED，则将第二个参数改为0 \*/

gpio\_set\_value(gpio\_table[i],1);

}

/\* 注册一个主设备号为10，次设备号为动态分配的混杂(字符)设备 \*/

ret = misc\_register(&misc);

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

return ret;

}

/\* 驱动程序的出口函数 \*/

static void \_\_exit Webee210\_led\_exit(void)

{

misc\_deregister(&misc);

}

/\* 用于修饰入口/出口函数，换句话说，相当于

\* 告诉内核驱动程序的入口/出口函数在哪里

\*/

module\_init(Webee210\_led\_init);

module\_exit(Webee210\_led\_exit);

/\* 该驱动支持的协议、作者、描述 \*/

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("webee");

MODULE\_DESCRIPTION("Character drivers for leds");

< driver / Makefile >

ifneq ($(KERNELRELEASE),)

obj-m :=webee210\_led.o

else

module-objs :=webee210\_led.o

KERNELDIR :=/home/gec/linux\_kernel/linux2.6.35.7/

PWD :=$(shell pwd)

default:

$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

clean:

$(RM) \*.ko \*.mod.c \*.mod.o \*.o \*.order \*.symvers \*.cmd