# 完整代码

## 按键--input子系统

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/types.h>

#include <linux/interrupt.h>

#include <linux/irq.h>

#include <linux/gpio.h>

#include <linux/input.h> //包含input子系统的相关头文件

#include <mach/gpio.h>

static struct input\_dev \*btn\_dev; //定义一个input\_dev结构指针

struct btn\_t {

int gpio;

int key;

char \*name;

};

static struct btn\_t btns[] = {

{ EXYNOS4\_GPX3(2), KEY\_L , "K1" }, //26

{ EXYNOS4\_GPX3(3), KEY\_S , "K2" }, //1F

{ EXYNOS4\_GPX3(4), KEY\_ENTER,"K3" }, //1C

{ EXYNOS4\_GPX3(5), KEY\_LEFTSHIFT,"K4" }, //9E

};

/\*\*

第六列表示上报事件: 0001 --> EV\_KEY; 0000 ---> EV\_SYN

第七列表示键值: 0026 --> KEY\_L

第八列表示按键事件: 0001 --> PRESS, 0000 --> RELEASE

\*\*/

static irqreturn\_t btn\_irq\_handler(int irq, void \*dev)

{

struct btn\_t \* p = (struct btn\_t \*)dev;

int pinval = 0;

//printk("%s is success\n",p->name);

pinval = gpio\_get\_value(p->gpio); /\* 松开 : 1 \*/

//void input\_event(struct input\_dev \*dev,unsigned int type, unsigned int code, int value)

//input\_sync(input\_dev)

if (pinval){

input\_event(btn\_dev,EV\_KEY,p->key,0); //上报松开事件

input\_sync(btn\_dev); //上报同步事件

}

else{

input\_event(btn\_dev,EV\_KEY,p->key,1); //上报按下事件

input\_sync(btn\_dev); //上报同步事件

}

return IRQ\_HANDLED;

}

static int \_\_init btn\_drv\_init(void)

{

int irq;

int i;

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

irq = gpio\_to\_irq(btns[i].gpio);

request\_irq(irq, btn\_irq\_handler, IRQ\_TYPE\_EDGE\_BOTH,

btns[i].name, &btns[i]);

}

/\* 分配一个input\_dev结构体 \*/

//struct input\_dev \*input\_allocate\_device(void)

btn\_dev = input\_allocate\_device();

/\* 设置 能产生哪类事件 \*/

set\_bit(EV\_KEY, btn\_dev->evbit); //设置 能产生按键事件

set\_bit(EV\_REP, btn\_dev->evbit); //设置 能产生重复事件

/\* 能产生这类操作里的哪些事件: L,S,ENTER,KEY\_BACK \*/

set\_bit(KEY\_L, btn\_dev->keybit);

set\_bit(KEY\_S, btn\_dev->keybit);

set\_bit(KEY\_ENTER, btn\_dev->keybit);

set\_bit(KEY\_LEFTSHIFT, btn\_dev->keybit);

/\* 注册input dev \*/

//int input\_register\_device(struct input\_dev \*dev)

input\_register\_device(btn\_dev);

return 0;

}

static void \_\_exit btn\_drv\_exit(void)

{

int irq;

int i;

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

irq = gpio\_to\_irq(btns[i].gpio);

free\_irq(irq, &btns[i]);

}

//void input\_unregister\_device(struct input\_dev \*dev)

input\_unregister\_device(btn\_dev); //注销 input dev

//input\_free\_device(struct input\_dev \*dev)

input\_free\_device(btn\_dev); //释放input dev内存

}

module\_init(btn\_drv\_init);

module\_exit(btn\_drv\_exit);

MODULE\_LICENSE("GPL");

## bus\_dev\_drv驱动机制platform驱动

编写my\_device.c文件，向系统中注册一个platform\_device;

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/types.h>

#include <linux/interrupt.h>

#include <linux/list.h>

#include <linux/timer.h>

#include <linux/init.h>

#include <linux/serial\_core.h>

#include <linux/platform\_device.h> //包含platform驱动相关的头文件

/\*保证程序可以成功卸载\*/

static void my\_release(struct device \*dev) {return ;}

/\*定义一个platform\_dervice变量\*/

static struct platform\_device my\_device={

.name =”my\_device”,

.dev = {

.release = my\_release,

},

};

static int \_\_init my\_dev\_init(void)

{

/\*注册platform\_device\*/

platform\_device\_register(&my\_device);

return 0;

}

static void \_\_exit my\_dev\_exit(void)

{

/\*注销platform\_device\*/

platform\_device\_unregister(&my\_device);

}

module\_init( my\_dev\_init);

module\_exit( my\_dev\_exit);

MODULE\_LICENSE("GPL");

编写my\_driver.c文件，向系统中注册一个platform\_driver;

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/types.h>

#include <linux/interrupt.h>

#include <linux/list.h>

#include <linux/timer.h>

#include <linux/init.h>

#include <linux/serial\_core.h>

#include <linux/ platform\_device.h > //包含platform驱动相关的头文件

/\* 编写一个探索函数（probe），函数只打印一些调试信息，然后返回0 \*/

static int my\_probe(struct platform\_device \*pdev)

{

printk(":: probe : device driver match success !\n");

return 0;

}

/\*编写一个移除函数(remove), 函数只打印一些调试信息，然后返回0\*/

static int my\_remove(struct platform\_device \*pdev)

{

return 0;

}

/\*定义一个platform\_driver变量\*/

struct platform\_driver my\_drv={

.probe =my\_probe,

.remove = my\_remove,

.driver = {

.owner = THIS\_MODULE,

.name =”my\_driver”,

}

};

static int my\_drv\_init(void)

{

/\*注册platform\_driver\*/

platform\_driver\_register(&my\_drv);

return 0;

}

static void my\_drv\_exit(void)

{

/\*注销platform\_driver\*/

platform\_driver\_unregister(&my\_drv);

}

module\_init( my\_drv\_init);

module\_exit( my\_drv\_exit);

MODULE\_LICENSE("GPL");

## framebuffer驱动

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/clk.h>

#include <linux/fb.h>

#include <linux/io.h>

#include <linux/dma-mapping.h>

#include <mach/gpio.h>

#include <mach/map.h>

#include <plat/gpio-cfg.h>

static struct fb\_info \*sice\_fbinfo;

static struct clk \*bus\_clk;

static struct clk \*lcd\_clk;;

/\*\* \* 定义寄存器 \* \*\*/

static void \_\_iomem \*regs\_base;

#define VIDCON0 (regs\_base+0x0)

#define VIDCON1 (regs\_base+0x4)

#define VIDTCON0 (regs\_base +0x10)

#define VIDTCON1 (regs\_base +0x14)

#define VIDTCON2 (regs\_base +0x18)

#define WINCON0 (regs\_base +0x20)

#define SHADOWCON (regs\_base +0x34)

#define VIDOSD0A (regs\_base +0x40)

#define VIDOSD0B (regs\_base +0x44)

#define VIDOSD0C (regs\_base +0x48)

#define VIDW00ADD0B0 (regs\_base +0xA0)

#define VIDW00ADD1B0 (regs\_base +0xD0)

#define VIDW00ADD2 (regs\_base +0x100)

#define LCDBLK\_CFG (S3C\_VA\_SYS + 0x0210)

static struct fb\_ops sice\_fb\_ops = {

.owner = THIS\_MODULE,

.fb\_fillrect = cfb\_fillrect,

.fb\_copyarea = cfb\_copyarea,

.fb\_imageblit = cfb\_imageblit,

};

static int \_\_init tiny4412\_fb\_init(void)

{

u32 data;

unsigned int buffer\_size;

dma\_addr\_t map\_dma;

/\*\*创建并初始化一个struct fb\_info \*\*/

sice\_fbinfo = framebuffer\_alloc(0,NULL); // fb\_info分配内存

sice\_fbinfo->fbops = &sice\_fb\_ops; // fb操作函数集合

/\*初始化fb\_info.var成员\*/

sice\_fbinfo->var.xres = 800;

sice\_fbinfo->var.yres = 480;

sice\_fbinfo->var.xres\_virtual = 800;

sice\_fbinfo->var.yres\_virtual = 480;

sice\_fbinfo->var.xoffset = 0;

sice\_fbinfo->var.yoffset = 0;

sice\_fbinfo->var.pixclock = 33036;

sice\_fbinfo->var.left\_margin = 36;

sice\_fbinfo->var.right\_margin = 80;

sice\_fbinfo->var.upper\_margin = 15;

sice\_fbinfo->var.lower\_margin = 22;

sice\_fbinfo->var.hsync\_len = 10;

sice\_fbinfo->var.vsync\_len = 8;

sice\_fbinfo->var.activate = FB\_ACTIVATE\_NOW;

sice\_fbinfo->var.bits\_per\_pixel = 32;

sice\_fbinfo->var.red.offset = 16;

sice\_fbinfo->var.red.length = 8;

sice\_fbinfo->var.green.offset = 8;

sice\_fbinfo->var.green.length = 8;

sice\_fbinfo->var.blue.offset = 0;

sice\_fbinfo->var.blue.length = 8;

/\*初始化fb\_info.fix成员\*/

sice\_fbinfo->fix.type = FB\_TYPE\_PACKED\_PIXELS;

sice\_fbinfo->fix.visual = FB\_VISUAL\_TRUECOLOR;

sice\_fbinfo->fix.line\_length = 800\*32/8; //每行的字节数

/\*分配显示缓冲区内存\*/

buffer\_size = 800\*480\*32/8;

sice\_fbinfo->fix.smem\_len = buffer\_size;

//DMA内存申请

sice\_fbinfo->screen\_base = dma\_alloc\_writecombine(NULL,

buffer\_size, &map\_dma, GFP\_KERNEL);

memset(sice\_fbinfo->screen\_base, 0x0, buffer\_size);

sice\_fbinfo->fix.smem\_start = map\_dma;

/\* 硬件初始化 \*/

/\*使能时钟\*/

bus\_clk = clk\_get\_sys("exynos4-fb.0", "lcd");

clk\_enable(bus\_clk);

lcd\_clk = clk\_get\_sys("exynos4-fb.0", "sclk\_fimd");

clk\_enable(lcd\_clk);

/\* 寄存器基地址进行映射\*/

regs\_base = ioremap(EXYNOS4\_PA\_FIMD0,SZ\_32K);

/\* 设置GPF为LCD pin功能 \*/

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF0(0), 8, S3C\_GPIO\_SFN(2));

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF1(0), 8, S3C\_GPIO\_SFN(2));

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF2(0), 8, S3C\_GPIO\_SFN(2));

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF3(0), 4, S3C\_GPIO\_SFN(2));

/\* 设置LCD的数据接口为 RGB \*/

data = \_\_raw\_readl(LCDBLK\_CFG);

data |= (1 << 1);

\_\_raw\_writel(data, LCDBLK\_CFG);

/\*设置VIDCON0, 1寄存器\*/

data =(1<<17)|(25<<6)| (1<<4)|(1<<1)|(1<<0);

writel(data, VIDCON0);

data = (1<<7)|(1<<6)|(1<<5);

writel(data, VIDCON1);

/\*设置时钟信号寄存器\*/

data = (14<<16)|(21<<8)|(7<<0);

writel(data, VIDTCON0);

data = (35<<16)|(79<<8)|(9<<0);

writel(data, VIDTCON1);

data = (479<<11)|(799<<0);

writel(data, VIDTCON2);

/\* 设置缓冲区地址寄存器 \*/

data = sice\_fbinfo->fix.smem\_start;

writel(data, VIDW00ADD0B0);

data = sice\_fbinfo->fix.smem\_start + 800\*480\*32/8;

writel(data, VIDW00ADD1B0);

data = 800\*32/4;

writel(data, VIDW00ADD2);

/\* write 'OSD' registers to control position of framebuffer \*/

data = 0;

writel(data, VIDOSD0A);

data = (799<<11) |(479<<0) ;

writel(data, VIDOSD0B);

/\*使用channel 0 通道\*/

data = readl(SHADOWCON);

data |= (1<<0) ;

writel(data, SHADOWCON);

/\*配置WINCON0寄存器\*/

data = (0xb << 2)| (1 << 15)| (0x0 << 9)| (1<<0);

writel(data, WINCON0);

/\* 注册fb\_info \*/

register\_framebuffer(sice\_fbinfo);

return 0;

}

static void \_\_exit tiny4412\_fb\_exit(void)

{

unregister\_framebuffer(sice\_fbinfo); //注销fb\_info

//dma内存释放

dma\_free\_writecombine(NULL,PAGE\_ALIGN(sice\_fbinfo->fix.smem\_len),

sice\_fbinfo->screen\_base,sice\_fbinfo->fix.smem\_start);

framebuffer\_release(sice\_fbinfo);//fb\_info内存释放

clk\_disable(lcd\_clk);

clk\_put(lcd\_clk);

clk\_disable(bus\_clk);

clk\_put(bus\_clk);

}

module\_init(tiny4412\_fb\_init);

module\_exit(tiny4412\_fb\_exit);

MODULE\_LICENSE("GPL");

# 挖空代码

## 按键--input子系统

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/types.h>

#include <linux/interrupt.h>

#include <linux/irq.h>

#include <linux/gpio.h>

#include <linux/input.h> //包含input子系统的相关头文件

#include <mach/gpio.h>

static struct input\_dev \*btn\_dev; //定义一个input\_dev结构指针

struct btn\_t {

int gpio;

int key;

char \*name;

};

static struct btn\_t btns[] = {

{ EXYNOS4\_GPX3(2), KEY\_L , "K1" }, //26

{ EXYNOS4\_GPX3(3), KEY\_S , "K2" }, //1F

{ EXYNOS4\_GPX3(4), KEY\_ENTER,"K3" }, //1C

{ EXYNOS4\_GPX3(5), KEY\_LEFTSHIFT,"K4" }, //9E

};

/\*\*

第六列表示上报事件: 0001 --> EV\_KEY; 0000 ---> EV\_SYN

第七列表示键值: 0026 --> KEY\_L

第八列表示按键事件: 0001 --> PRESS, 0000 --> RELEASE

\*\*/

static irqreturn\_t btn\_irq\_handler(int irq, void \*dev)

{

struct btn\_t \* p = (struct btn\_t \*)dev;

int pinval = 0;

//printk("%s is success\n",p->name);

pinval = gpio\_get\_value(p->gpio); /\* 松开 : 1 \*/

if (pinval){

//上报松开事件

//上报同步事件

}

else{

//上报按下事件

//上报同步事件

}

return IRQ\_HANDLED;

}

static int \_\_init btn\_drv\_init(void)

{

int irq;

int i;

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

irq = gpio\_to\_irq(btns[i].gpio);

request\_irq(irq, btn\_irq\_handler, IRQ\_TYPE\_EDGE\_BOTH,

btns[i].name, &btns[i]);

}

/\* 分配一个input\_dev结构体 \*/

btn\_dev =

/\* 设置 能产生哪类事件 \*/

set\_bit( , btn\_dev->evbit); //设置 能产生按键事件

set\_bit( , btn\_dev->evbit); //设置 能产生重复事件

/\* 能产生这类操作里的哪些事件: L,S,ENTER,KEY\_BACK \*/

set\_bit( , btn\_dev->keybit);

set\_bit( , btn\_dev->keybit);

set\_bit( , btn\_dev->keybit);

set\_bit( , btn\_dev->keybit);

/\* 注册input dev \*/

return 0;

}

static void \_\_exit btn\_drv\_exit(void)

{

int irq;

int i;

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

irq = gpio\_to\_irq(btns[i].gpio);

free\_irq(irq, &btns[i]);

}

//注销 input dev

//释放input dev内存

}

module\_init(btn\_drv\_init);

module\_exit(btn\_drv\_exit);

MODULE\_LICENSE("GPL");

## bus\_dev\_drv驱动机制platform驱动

编写my\_device.c文件，向系统中注册一个platform\_device;

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/types.h>

#include <linux/interrupt.h>

#include <linux/list.h>

#include <linux/timer.h>

#include <linux/init.h>

#include <linux/serial\_core.h>

#include <linux/platform\_device.h> //包含platform驱动相关的头文件

/\*保证程序可以成功卸载\*/

/\*定义一个platform\_dervice变量\*/

static struct platform\_device my\_device={

.name = ,

.dev = {

.release = my\_release,

},

};

static int \_\_init my\_dev\_init(void)

{

/\*注册platform\_device\*/

return 0;

}

static void \_\_exit my\_dev\_exit(void)

{

/\*注销platform\_device\*/

}

module\_init( my\_dev\_init);

module\_exit( my\_dev\_exit);

MODULE\_LICENSE("GPL");

编写my\_driver.c文件，向系统中注册一个platform\_driver;

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/types.h>

#include <linux/interrupt.h>

#include <linux/list.h>

#include <linux/timer.h>

#include <linux/init.h>

#include <linux/serial\_core.h>

#include <linux/ platform\_device.h > //包含platform驱动相关的头文件

/\* 编写一个探索函数（probe），函数只打印一些调试信息，然后返回0 \*/

static int my\_probe(struct platform\_device \*pdev)

{

printk(":: probe : device driver match success !\n");

return 0;

}

/\*编写一个移除函数(remove), 函数只打印一些调试信息，然后返回0\*/

static int my\_remove(struct platform\_device \*pdev)

{

return 0;

}

/\*定义一个platform\_driver变量\*/

struct platform\_driver my\_drv={

.probe = ,

.remove = ,

.driver = {

.owner = THIS\_MODULE,

.name = ,

}

};

static int my\_drv\_init(void)

{

/\*注册platform\_driver\*/

return 0;

}

static void my\_drv\_exit(void)

{

/\*注销platform\_driver\*/

}

module\_init( my\_drv\_init);

module\_exit( my\_drv\_exit);

MODULE\_LICENSE("GPL");

## framebuffer驱动

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/clk.h>

#include <linux/fb.h>

#include <linux/io.h>

#include <linux/dma-mapping.h>

#include <mach/gpio.h>

#include <mach/map.h>

#include <plat/gpio-cfg.h>

static struct fb\_info \*sice\_fbinfo;

static struct clk \*bus\_clk;

static struct clk \*lcd\_clk;;

/\*\* \* 定义寄存器 \* \*\*/

static void \_\_iomem \*regs\_base;

#define VIDCON0 (regs\_base+0x0)

#define VIDCON1 (regs\_base+0x4)

#define VIDTCON0 (regs\_base +0x10)

#define VIDTCON1 (regs\_base +0x14)

#define VIDTCON2 (regs\_base +0x18)

#define WINCON0 (regs\_base +0x20)

#define SHADOWCON (regs\_base +0x34)

#define VIDOSD0A (regs\_base +0x40)

#define VIDOSD0B (regs\_base +0x44)

#define VIDOSD0C (regs\_base +0x48)

#define VIDW00ADD0B0 (regs\_base +0xA0)

#define VIDW00ADD1B0 (regs\_base +0xD0)

#define VIDW00ADD2 (regs\_base +0x100)

#define LCDBLK\_CFG (S3C\_VA\_SYS + 0x0210)

static struct fb\_ops sice\_fb\_ops = {

.owner = THIS\_MODULE,

.fb\_fillrect = cfb\_fillrect,

.fb\_copyarea = cfb\_copyarea,

.fb\_imageblit = cfb\_imageblit,

};

static int \_\_init tiny4412\_fb\_init(void)

{

u32 data;

unsigned int buffer\_size;

dma\_addr\_t map\_dma;

/\*\*创建并初始化一个struct fb\_info \*\*/

sice\_fbinfo = // fb\_info分配内存

sice\_fbinfo->fbops = // fb操作函数集合

/\*初始化fb\_info.var成员\*/

sice\_fbinfo->var.xres = 800;

sice\_fbinfo->var.yres = 480;

sice\_fbinfo->var.xres\_virtual = 800;

sice\_fbinfo->var.yres\_virtual = 480;

sice\_fbinfo->var.xoffset = 0;

sice\_fbinfo->var.yoffset = 0;

sice\_fbinfo->var.pixclock = 33036;

sice\_fbinfo->var.left\_margin = 36;

sice\_fbinfo->var.right\_margin = 80;

sice\_fbinfo->var.upper\_margin = 15;

sice\_fbinfo->var.lower\_margin = 22;

sice\_fbinfo->var.hsync\_len = 10;

sice\_fbinfo->var.vsync\_len = 8;

sice\_fbinfo->var.activate = FB\_ACTIVATE\_NOW;

sice\_fbinfo->var.bits\_per\_pixel = 32;

sice\_fbinfo->var.red.offset = 16;

sice\_fbinfo->var.red.length = 8;

sice\_fbinfo->var.green.offset = 8;

sice\_fbinfo->var.green.length = 8;

sice\_fbinfo->var.blue.offset = 0;

sice\_fbinfo->var.blue.length = 8;

/\*初始化fb\_info.fix成员\*/

sice\_fbinfo->fix.type = FB\_TYPE\_PACKED\_PIXELS;

sice\_fbinfo->fix.visual = FB\_VISUAL\_TRUECOLOR;

sice\_fbinfo->fix.line\_length = 800\*32/8; //每行的字节数

/\*分配显示缓冲区内存\*/

buffer\_size = 800\*480\*32/8;

sice\_fbinfo->fix.smem\_len = buffer\_size;

//DMA内存申请

sice\_fbinfo->screen\_base = (NULL,

buffer\_size, &map\_dma, GFP\_KERNEL);

memset(sice\_fbinfo->screen\_base, 0x0, buffer\_size);

sice\_fbinfo->fix.smem\_start = map\_dma;

/\* 硬件初始化 \*/

/\*使能时钟\*/

bus\_clk = clk\_get\_sys("exynos4-fb.0", "lcd");

clk\_enable(bus\_clk);

lcd\_clk = clk\_get\_sys("exynos4-fb.0", "sclk\_fimd");

clk\_enable(lcd\_clk);

/\* 寄存器基地址进行映射\*/

regs\_base = ioremap(EXYNOS4\_PA\_FIMD0,SZ\_32K);

/\* 设置GPF为LCD pin功能 \*/

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF0(0), 8, S3C\_GPIO\_SFN(2));

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF1(0), 8, S3C\_GPIO\_SFN(2));

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF2(0), 8, S3C\_GPIO\_SFN(2));

s3c\_gpio\_cfgrange\_nopull(EXYNOS4\_GPF3(0), 4, S3C\_GPIO\_SFN(2));

/\* 设置LCD的数据接口为 RGB \*/

data = \_\_raw\_readl(LCDBLK\_CFG);

data |= (1 << 1);

\_\_raw\_writel(data, LCDBLK\_CFG);

/\*设置VIDCON0, 1寄存器\*/

data =(1<<17)|(25<<6)| (1<<4)|(1<<1)|(1<<0);

writel(data, VIDCON0);

data = (1<<7)|(1<<6)|(1<<5);

writel(data, VIDCON1);

/\*设置时钟信号寄存器\*/

data = (14<<16)|(21<<8)|(7<<0);

writel(data, VIDTCON0);

data = (35<<16)|(79<<8)|(9<<0);

writel(data, VIDTCON1);

data = (479<<11)|(799<<0);

writel(data, VIDTCON2);

/\* 设置缓冲区地址寄存器 \*/

data = sice\_fbinfo->fix.smem\_start;

writel(data, VIDW00ADD0B0);

data = sice\_fbinfo->fix.smem\_start + 800\*480\*32/8;

writel(data, VIDW00ADD1B0);

data = 800\*32/4;

writel(data, VIDW00ADD2);

/\* write 'OSD' registers to control position of framebuffer \*/

data = 0;

writel(data, VIDOSD0A);

data = (799<<11) |(479<<0) ;

writel(data, VIDOSD0B);

/\*使用channel 0 通道\*/

data = readl(SHADOWCON);

data |= (1<<0) ;

writel(data, SHADOWCON);

/\*配置WINCON0寄存器\*/

data = (0xb << 2)| (1 << 15)| (0x0 << 9)| (1<<0);

writel(data, WINCON0);

/\* 注册fb\_info \*/

return 0;

}

static void \_\_exit tiny4412\_fb\_exit(void)

{

//注销fb\_info

//dma内存释放

(NULL,PAGE\_ALIGN(sice\_fbinfo->fix.smem\_len),

sice\_fbinfo->screen\_base,sice\_fbinfo->fix.smem\_start);

//fb\_info内存释放

clk\_disable(lcd\_clk);

clk\_put(lcd\_clk);

clk\_disable(bus\_clk);

clk\_put(bus\_clk);

}

module\_init(tiny4412\_fb\_init);

module\_exit(tiny4412\_fb\_exit);

MODULE\_LICENSE("GPL");