# Linux设备总线驱动模型

## 1.基本概念

从[Linux](http://lib.csdn.net/base/linux)2.6起引入了一套新的驱动管理和注册机制:Platform\_device和 Platform\_driver。Linux中大部分的设备驱动，都可以使用这套机制，设备Platform\_device表示，驱动用 Platform\_driver进行注册。

Linux platform driver机制和传统的device driver机制（即：通过driver\_register函数进行注册）相比，一个十分明显的优势在于platform机制将设备本身的资源注册进内核，由内核统一管理，在驱动程序中用使用这些资源时，通过platform device提供的标准接口进行申请并使用。

和传统的驱动一样，platform机制也分为三个步骤：

(1) 总线注册阶段：

内核启动初始化时的main.c文件中的kernel\_init()-->do\_basic\_setup()-->driver\_init()-->platform\_bus\_init()-->bus\_register(&platform\_bus\_type)，注册了一条platform总线（虚拟总线）。

(2) 添加设备时：

设备注册的时候Platform\_device\_register()-->platform\_device\_add()-->(pdev->dev.bus =&platform\_bus\_type)->device\_add()，就这样把设备给挂到虚拟的总线上。

(3) 驱动注册时：

Platform\_driver\_register()-->driver\_register()-->bus\_add\_driver()-->driver\_attach()-->bus\_for\_each\_dev(), 对在每个挂在虚拟的platform bus的设备作\_\_driver\_attach()--> driver\_probe\_device(),判断drv\_bus\_match()是否执行成功，此时通过指针执行platform\_match--> strncmp(pdev->name , drv->name , BUS\_ID\_SIZE),如果相符就调用really\_probe(实际就是执行相应设备的platform\_driver->probe(platform\_device)。)开始真正的探测，如果probe成功，则绑定设备到该驱动。

## 2.写一个简单的platform驱动

以led为例，实现基于platform架构的led驱动。

### 2.1 platform device部分

代码如下：

#include <linux/init.h>

#include <linux/module.h>

#include <linux/fs.h>

#include <asm/io.h>

#include <asm/uaccess.h>

#include <linux/device.h>

#include <linux/cdev.h>

#include <linux/platform\_device.h>

static void led\_platform\_device\_release**(**struct device **\***dev**)**

**{**

**return;**

**}**

static struct resource led\_resource**[]** **=**

**{**

**[**0**]** **=**

**{**

**.**start **=** 0x56000050**,**

**.**end **=** 0x56000050**+**16**,**

**.**flags **=** IORESOURCE\_MEM**,**

**},**

**};**

static struct platform\_device led\_platform\_device **=**

**{**

**.**name **=** "jz2440 led"**,**

**.**id **=** **-**1**,**

**.**num\_resources **=** ARRAY\_SIZE**(**led\_resource**),**

**.**resource **=** led\_resource**,**

**.**dev **=** **{**

**.**release **=** led\_platform\_device\_release**,**

**},**

**};**

static int \_\_init led\_platform\_device\_init**()**

**{**

printk**(**"led\_platform\_device\_init\n"**);**

**return** platform\_device\_register**(&**led\_platform\_device**);**

**}**

static void \_\_exit led\_platform\_driver\_exit**()**

**{**

printk**(**"led\_platform\_driver\_exit\n"**);**

platform\_device\_unregister**(&**led\_platform\_device**);**

**}**

module\_init**(**led\_platform\_device\_init**);**

module\_exit**(**led\_platform\_driver\_exit**);**

MODULE\_LICENSE**(**"GPL"**);**

在上述代码中，驱动加载时通过platform\_device\_register函数在总线上注册platform device。告诉了操作led需要操作的寄存器地址0x56000050。驱动卸载时调用platform\_device\_unregister函数在总线上删除该platform device。

### 2.2 platform driver部分

代码如下：

#include <linux/init.h>

#include <linux/module.h>

#include <linux/fs.h>

#include <asm/io.h>

#include <asm/uaccess.h>

#include <linux/device.h>

#include <linux/cdev.h>

#include <linux/platform\_device.h>

#define DEVICE\_NAME "led"

struct led\_dev\_t

**{**

struct cdev cdev**;**

**}**led\_dev**;**

static struct class **\***led\_drv\_class**;**

int led\_major**;**

volatile unsigned long **\***gpfcon **=** **NULL;**

volatile unsigned long **\***gpfdat **=** **NULL;**

static int led\_drv\_ioctl**(**struct inode **\***inode**,** struct file **\***file**,** unsigned int cmd**,**unsigned long arg**)**

**{**

**if(**cmd **==** 1**)**

**{**

//某个点亮

**\***gpfdat **&=** **~(**0x1**<<(**arg **+** 4**));**

**}**

**else**

**{**

//某个熄灭

**\***gpfdat **|=** **(**0x1**<<(**arg **+** 4**));**

**}**

**}**

static int led\_drv\_open**()**

**{**

//配置GPF4,5,6为输出

**\***gpfcon **&=** **~((**0x3**<<(**4**\***2**))** **|** **(**0x3**<<(**5**\***2**))** **|** **(**0x3**<<(**6**\***2**)));**

**\***gpfcon **|=** **((**0x1**<<(**4**\***2**))** **|** **(**0x1**<<(**5**\***2**))** **|** **(**0x1**<<(**6**\***2**)));**

**return** 0**;**

**}**

static struct file\_operations led\_drv\_fops **=** **{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** led\_drv\_open**,**

**.**ioctl **=** led\_drv\_ioctl**,**

**};**

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

**{**

printk**(**"led\_probe\n"**);**

dev\_t dev\_id**;**

**if(**led\_major**)**

**{**

//静态

dev\_id **=** MKDEV**(**led\_major**,**0**);**

register\_chrdev\_region**(**dev\_id**,**1**,**DEVICE\_NAME**);**

**}**

**else**

**{**

//动态

alloc\_chrdev\_region**(&**dev\_id**,**0**,**1**,**DEVICE\_NAME**);**

led\_major **=** MAJOR**(**dev\_id**);**

**}**

cdev\_init**(&**led\_dev**.**cdev**,** **&**led\_drv\_fops**);**

led\_dev**.**cdev**.**owner **=** THIS\_MODULE**;**

cdev\_add**(&**led\_dev**.**cdev**,**dev\_id**,**1**);**

led\_drv\_class **=** class\_create**(**THIS\_MODULE**,** DEVICE\_NAME**);**

class\_device\_create**(**led\_drv\_class**,** **NULL,** MKDEV**(**led\_major**,** 0**),** **NULL,** DEVICE\_NAME**);**

struct resource **\***pIORESOURCE\_MEM **=** **NULL;**

pIORESOURCE\_MEM **=** platform\_get\_resource**(**pdev**,**IORESOURCE\_MEM**,**0**);**

gpfcon **=** ioremap**(**pIORESOURCE\_MEM**->**start**,**pIORESOURCE\_MEM**->**end **-** pIORESOURCE\_MEM**->**start**);**

gpfdat **=** gpfcon **+** 1**;**

**return** 0**;**

**}**

static int \_\_devexit led\_remove**(**struct platform\_device **\***pdev**)**

**{**

printk**(**"led\_remove\n"**);**

class\_device\_destroy**(**led\_drv\_class**,**MKDEV**(**led\_major**,**0**));**

class\_destroy**(**led\_drv\_class**);**

cdev\_del**(&**led\_dev**.**cdev**);**

unregister\_chrdev\_region**(**MKDEV**(**led\_major**,**0**),**1**);**

iounmap**(**gpfcon**);**

**return** 0**;**

**}**

static struct platform\_driver led\_platform\_driver **=**

**{**

**.**probe **=** led\_probe**,**

**.**remove **=** \_\_devexit\_p**(**led\_remove**),**

**.**driver **=** **{**

**.**name **=** "jz2440 led"**,**

**.**owner **=** THIS\_MODULE**,**

**}**

**};**

static int \_\_init led\_platform\_driver\_init**()**

**{**

printk**(**"led\_platform\_driver\_init\n"**);**

**return** platform\_driver\_register**(&**led\_platform\_driver**);**

**}**

static void \_\_exit led\_platform\_driver\_exit**()**

**{**

printk**(**"led\_platform\_driver\_exit\n"**);**

platform\_driver\_unregister**(&**led\_platform\_driver**);**

**}**

module\_init**(**led\_platform\_driver\_init**);**

module\_exit**(**led\_platform\_driver\_exit**);**

MODULE\_LICENSE**(**"GPL"**);**

这部分代码和传统的led驱动很类似，驱动加载时调用platform\_driver\_register在总线上注册platform driver。驱动卸载时调用platform\_driver\_unregister在总线上删除该platform driver。

当led platform\_driver和platform\_device都加载到总线中，由于它们的name相同，则会引发led\_probe函数的执行。led\_probe中会创建字符设备，这和传统的驱动一样，不同的是platform driver会调用platform\_get\_resource获取操作led所用的寄存器地址(platform device中注册的)。

### 2.3 测试

无论先加载platform\_device还是先加载platform\_driver都会触发probe函数。

# insmod led\_platform\_driver.ko

led\_platform\_driver\_init

# insmod led\_platform\_device.ko

led\_platform\_device\_init

led\_probe

# insmod led\_platform\_device.ko

led\_platform\_device\_init

# insmod led\_platform\_driver.ko

led\_platform\_driver\_init

led\_probe

无论先卸载platform\_device还是先加载platform\_driver都会触发remove函数。

# rmmod led\_platform\_driver.ko

led\_platform\_driver\_exit

led\_remove

# rmmod led\_platform\_device.ko

led\_platform\_device\_exit

led\_remove