# 平台设备的识别问题

——华清远见 曹忠明

在初学系统移植的时候，很多同学碰到这样的问题，比如要添加 LCD 的支持，网上很多资料说要添加一些代码，可是为什么添加这些代码缺不是很清楚。这里我们分析一些这些代码和驱动之间的关系。

比如我们这里要添加 LCD 的支持，以 S3C2410 为例，我们会在 arch/arm/mach-s3c2410/mach-smdk2410.c 中添加如下代码：

```
static struct s3c2410fb_display s3c2410_lcd_cfg[] __initdata = {
{
	.lcdcon5 = S3C2410_LCDCON5_FRM565 |
		S3C2410_LCDCON5_INVVCLK |
		S3C2410_LCDCON5_INVVLINE |
		S3C2410_LCDCON5_INVVFRAME |
		S3C2410_LCDCON5_PWREN |
		S3C2410_LCDCON5_HWSWP,

		.type		= S3C2410_LCDCON1_TFT,
		.width		= 320,
		.height		= 240,
		.pixclock	= 100000, /* HCLK/10 */
		.xres		= 320,
		.yres		= 240,
		.bpp		= 16,
		.left_margin	= 13,
		.right_margin	= 8,
		.hsync_len	= 4,
		.upper_margin	= 2,
		.lower_margin	= 7,
		.vsync_len	= 4,
		},
};
static struct s3c2410fb_mach_info s3c2410_fb_info __initdata = {
	.displays	= s3c2410_lcd_cfg,
	.num_displays	= ARRAY_SIZE(s3c2410_lcd_cfg),
	.default_display = 0,
	.lpcsel		= ((0xCE6) & ~7) | 1<<4,
};
在函数 smdk2410_init 中添加如下内容
s3c24xx_fb_set_platdata(&s3c2410_fb_info);
```

添加这些代码之间的关系是什么我呢，我们发现前两个结构体之间有包含关系，这些数据用来描述我们 LCD 屏的物理特性。而 s3c24xx_fb_set_platdata(&s3c2410_fb_info)才是使用这些变量的函数。这个函数做了些什么工作呢，我们看下他的源代码：

```
arch/arm/plat-s3c24xx/devs.c
```

```
void __init s3c24xx_fb_set_platdata(struct s3c2410fb_mach_info *pd)
{
        struct s3c2410fb_mach_info *npd;
        npd = kmemdup(pd, sizeof(*npd), GFP_KERNEL);
        if (npd) {
            s3c_device_lcd.dev.platform_data = npd;
            npd->displays = kmemdup(pd->displays,
                sizeof(struct s3c2410fb_display) * npd->num_displays,
                GFP_KERNEL);
            if (!npd->displays)
                printk(KERN_ERR "no memory for LCD display data\n");
        } else {
            printk(KERN_ERR "no memory for LCD platform data\n");
        }
}
```

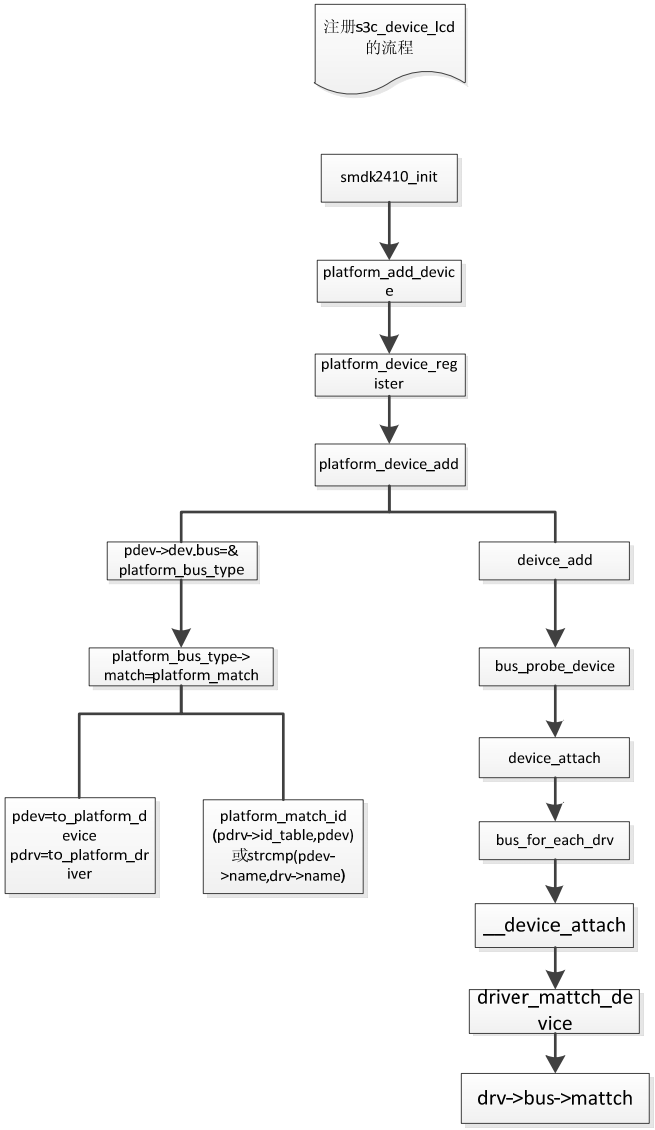
s3c_device_lcd.dev.platform_data = npd 就是将我们用来描述 LCD 的物理信息的结构体赋值给 s3c_device_lcd 的成员，s3c_device_lcd 又是什么定义呢，在那里使用的呢？

```
static struct resource s3c_lcd_resource[] = {
    [0] = {
        .start = S3C24XX_PA_LCD,
        .end   = S3C24XX_PA_LCD + S3C24XX_SZ_LCD - 1,
        .flags = IORESOURCE_MEM,
        },
    [1] = {
        .start = IRQ_LCD,
        .end   = IRQ_LCD,
        .flags = IORESOURCE_IRQ,
        }
};
static u64 s3c_device_lcd_dmamask = 0xffffffffUL;
struct platform_device s3c_device_lcd = {
    .name           = "s3c2410-lcd",
    .id             = -1,
    .num_resources      = ARRAY_SIZE(s3c_lcd_resource),
    .resource   = s3c_lcd_resource,
    .dev                = {
        .dma_mask           = &s3c_device_lcd_dmamask,
        .coherent_dma_mask  = 0xffffffffUL
    }
};
```

这里定义了 s3c_device_lcd 也就是我们的平台设备，这个结构用来描述 lcd 的物理信息，比如控制器的物理地址，及屏的物理信息。那么这个内容如何去操作我们的 LCD 呢。

这里涉及到 linux 内核的设计思想，linux 内核想把设备与驱动完全分离，设备是设备，驱动是驱动。所以就出现了平台设备(platform_deivce)和驱动(platform_driver),这两个结果互

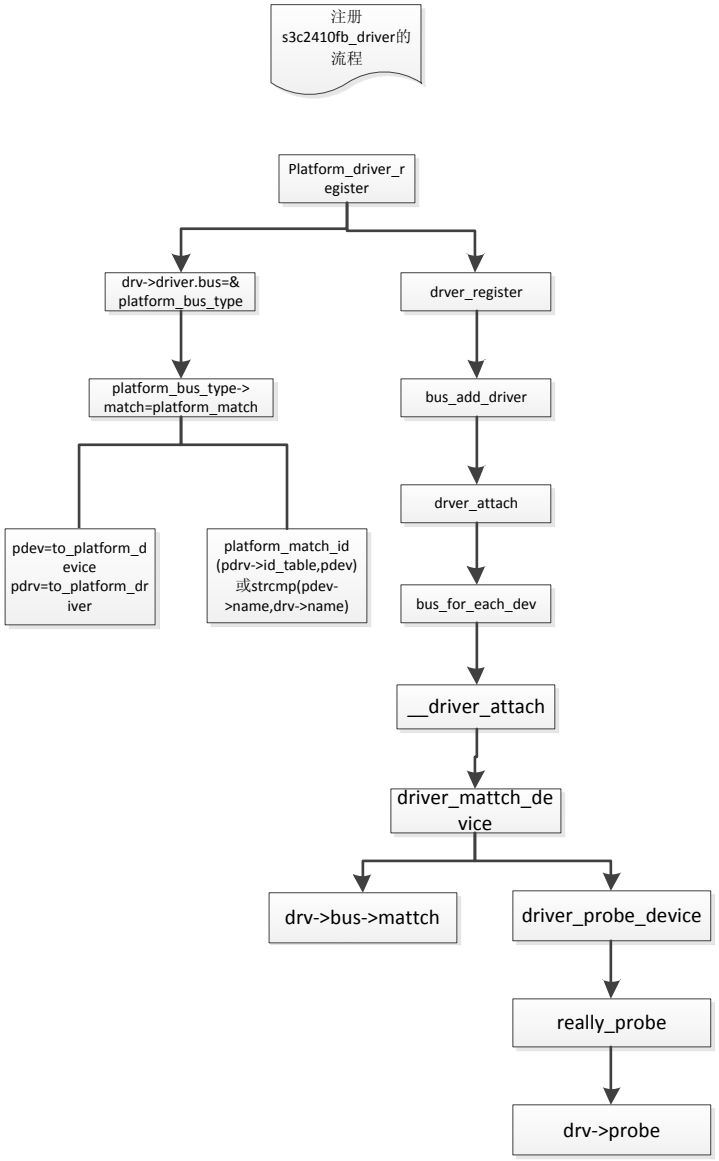
不相干，一个用来描述设备，一个是驱动，但是他们有共同的成员是 bus_type 通过是个东西进行关联。上面我们定义了一个平台设备，并且在 arch/arm/mach-s3c2410/mach-smdk2410.c 中将这个设备注册到我们的系统中去。注册流程为：




在 s3c2410fb.c 中驱动中定义了这么一个结构：

```
static struct platform_driver s3c2410fb_driver = {
        .probe          = s3c2410fb_probe,
        .remove         = __devexit_p(s3c2410fb_remove),
        .suspend  = s3c2410fb_suspend,
        .resume         = s3c2410fb_resume,
        .driver         = {
            .name   = "s3c2410-lcd",
            .owner  = THIS_MODULE,
        },
};
```

我们看一下这个驱动的注册流程吧：




这里在最后，我们的驱动通过“name”找到了它的平台设备，并在执行 probe 的时候把，平台设备的相关内容带进来。其实设备和驱动的识别是相互的，先添加驱动后添加设备则是设备找驱动，先添加设备后添加驱动则是驱动找设备，不过现在的内核里多数情况是后者，也就是先添加了设备后添加驱动，在驱动注册的时候查询设备列表，并用相关的数据对我们的设备进行初始化！