

专注于嵌入式 & Linux

以Kernel为中心，坚持学习各种资源建设。

<	2018年8月						>
日	一	二	三	四	五	六	
29	30	31	1	2	3	4	
5	6	7	8	9	10	11	
12	13	14	15	16	17	18	
19	20	21	22	23	24	25	
26	27	28	29	30	31	1	
2	3	4	5	6	7	8	

搜索

<input type="text"/>	<input type="button" value="找找看"/>
<input type="text"/>	<input type="button" value="谷歌搜索"/>

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签

Android开发(24)
Linux驱动(15)
ARM-Linux学习(7)
Linux应用(7)
BootLoader学习(7)
Cubieboard2学习(7)
ARM裸机开发(5)
C语言(4)
USB学习(3)
数据结构与算法(3)
更多

随笔档案⁽⁹⁴⁾

2014年3月 (7)
2013年10月 (1)
2013年8月 (12)
2013年7月 (3)
2013年6月 (2)
2013年5月 (4)
2013年4月 (2)
2013年1月 (7)
2012年11月 (2)
2012年10月 (3)
2012年9月 (1)
2012年8月 (6)
2012年7月 (3)

博客园 首页 新随笔 联系 管理 订阅 XML

随笔- 94 文章- 0 评论- 179

Linux设备驱动剖析之SPI（二）

957至962行，一个SPI控制器用一个master来描述。这里使用SPI核心的spi_alloc_master函数请求分配master。它在drivers/spi/spi.c文件中定义：



```
00000471 struct spi_master *spi_alloc_master(struct device *dev, unsigned size)
00000472 {
00000473     struct spi_master *master;
00000474
00000475     if (!dev)
00000476         return NULL;
00000477
00000478     master = kzalloc(size + sizeof *master, GFP_KERNEL);
00000479     if (!master)
00000480         return NULL;
00000481
00000482     device_initialize(&master->dev);
00000483     master->dev.class = &spi_master_class;
00000484     master->dev.parent = get_device(dev);
00000485     spi_master_set_devdata(master, &master[1]);
00000486
00000487     return master;
00000488 }
```



478至480行，这里分配的内存大小是*master加size，包含了两部分内存。

482行，设备模型中的初始设备函数，不说。

483行，spi_master_class在SPI子系统初始化的时候就已经注册好了。

484行，设置当前设备的父设备，关于设备模型的。

485行，&master[1]就是master之后的另一部分内存的起始地址。

回到s3c64xx_spi_probe函数，966行，就是取出刚才申请的第二部分内存的起始地址。

966至980行，根据预先定义的变量、函数进行填充。

983行，有点意思，说明该驱动支持哪些SPI模式。

985至997行，写过Linux驱动都应该知道，IO内存映射。

999至1003行，SPI IO管脚配置，将相应的IO管脚设置为SPI功能。

1006至1032行，使能SPI时钟。

1034至1040行，创建单个线程的工作队列，用于数据收发操作。

1043行，硬件初始化，初始化SPI控制器寄存器。

2012年6月 (2)
2012年5月 (7)
2012年4月 (3)
2012年3月 (16)
2012年2月 (13)

积分与排名

积分 - 135080
排名 - 2392

最新评论

1. Re:Linux设备驱动剖析之Input (三)
很美
--haoxing990
2. Re:Qt下libusb-win32的使用方法
我觉得所有问题都是都和驱动有关:
1每次运行后显示程序异常, 是因为没有安装驱动
2访问不了鼠标等是因为只能访问安装了inf-wizard.exe生成的驱动程序的USB设备
--蕾小蕾
3. Re:Qt下libusb-win32的使用方法
另外运行 inf-wizard.exe 时, 需要管理员权限, 否则可能会出现"System Policy has been modified to reject unsigned drivers"的错误
--Beny
4. Re:Qt下libusb-win32的使用方法
@史毅磊刚刚发现问题所在了, 原来要先使用 libusb自带的inf-wizard.exe 工具先给你的usb安装驱动, 再运行就没问题了。楼主文章有说了, 但没注意, 或许楼主可强调一下, 引起注意@lknlfy.....
--Beny
5. Re:Qt下libusb-win32的使用方法
@史毅磊我也遇到同样问题, 不知道你解决了没有? ...
--Beny

阅读排行榜

1. Android NDK开发 (2) ----- JNI多线程(20546)
2. Android应用开发提高篇 (4) ----- Socket编程 (多线程、双向通信) (13240)
3. Android NDK开发 (1) ----- Java与C互相调用实例详解(9812)
4. Linux内存映射 (mmap) (9257)
5. Android应用开发基础篇 (16) ----- ScaleGestureDetector (缩放手势检测) (8062)

评论排行榜

1. Barebox for Tiny6410(网卡驱动移植)(16)
2. Android NDK开发 (1) ----- Java与C互相调用实例详解(13)
3. Linux内存映射 (mmap) (11)
4. Android应用开发提高篇 (2) ----- 文本朗读TTS (TextToSpeech) (10)
5. Android应用开发基础篇 (12) ----- Socket通信(9)

1045至1048行, 锁, 工作队列等初始化。

1050至1054行, spi_register_master在drivers/spi/spi.c文件中定义:



```
00000511 int spi_register_master(struct spi_master *master)
00000512 {
00000513     static atomic_t      dyn_bus_id = ATOMIC_INIT((1<<15) - 1);
00000514     struct device        *dev = master->dev.parent;
00000515     int                   status = -ENODEV;
00000516     int                   dynamic = 0;
00000517
00000518     if (!dev)
00000519         return -ENODEV;
00000520
00000521     /* even if it's just one always-selected device, there must
00000522      * be at least one chipselect
00000523      */
00000524     if (master->num_chipselect == 0)
00000525         return -EINVAL;
00000526
00000527     /* convention: dynamically assigned bus IDs count down from the max */
00000528     if (master->bus_num < 0) {
00000529         /* FIXME switch to an IDR based scheme, something like
00000530          * I2C now uses, so we can't run out of "dynamic" IDs
00000531          */
00000532         master->bus_num = atomic_dec_return(&dyn_bus_id);
00000533         dynamic = 1;
00000534     }
00000535
00000536     spin_lock_init(&master->bus_lock_spinlock);
00000537     mutex_init(&master->bus_lock_mutex);
00000538     master->bus_lock_flag = 0;
00000539
00000540     /* register the device, then userspace will see it.
00000541      * registration fails if the bus ID is in use.
00000542      */
00000543     dev_set_name(&master->dev, "spi%u", master->bus_num);
00000544     status = device_add(&master->dev);
00000545     if (status < 0)
00000546         goto done;
00000547     dev_dbg(dev, "registered master %s%s\n", dev_name(&master->dev),
00000548             dynamic ? " (dynamic)" : "");
00000549
00000550     /* populate children from any spi device tables */
00000551     scan_boardinfo(master);
00000552     status = 0;
00000553
00000554     /* Register devices from the device tree */
00000555     of_register_spi_devices(master);
00000556 done:
00000557     return status;
00000558 }
```



524行, 一个SPI控制器至少有一个片选, 因此片选数为0则出错。

528至534行, 如果总线号小于0则动态分配一个总线号。

543至548行, 把master加入到设备模型中。

551行, scan_boardinfo函数同样是在driver/spi/spi.c中定义:



```
00000414 static void scan_boardinfo(struct spi_master *master)
00000415 {
00000416     struct boardinfo      *bi;
00000417
00000418     mutex_lock(&board_lock);
00000419     list_for_each_entry(bi, &board_list, list) {
00000420         struct spi_board_info *chip = bi->board_info;
00000421         unsigned              n;
00000422
00000423         for (n = bi->n_board_info; n > 0; n--, chip++) {
```

推荐排行榜

1. Android NDK开发（1）----- Java与C互相调用实例详解(7)
2. 使用FFmpeg捕获一帧摄像头图像(3)
3. 从MACHINE_START开始(3)
4. Android应用开发提高篇（6）----- FaceDetector（人脸检测）(2)
5. Android应用开发基础篇（4）----- TabHost（选项卡）(2)

```
00000424         if (chip->bus_num != master->bus_num)
00000425             continue;
00000426         /* NOTE: this relies on spi_new_device to
00000427          * issue diagnostics when given bogus inputs
00000428          */
00000429         (void) spi_new_device(master, chip);
00000430     }
00000431 }
00000432 mutex_unlock(&board_lock);
00000433 }
```


 复制

419至431做了两件事情，首先遍历board_list这个链表，每找到一个成员就将它的总线号与master的总线号进行比较，如果相等则调用spi_new_device函数创建一个spi设备。

```
 复制
00000336 struct spi_device *spi_new_device(struct spi_master *master,
00000337                                   struct spi_board_info *chip)
00000338 {
00000339     struct spi_device *proxy;
00000340     int status;
00000341
00000342     /* NOTE: caller did any chip->bus_num checks necessary.
00000343      *
00000344      * Also, unless we change the return value convention to use
00000345      * error-or-pointer (not NULL-or-pointer), troubleshootability
00000346      * suggests syslogged diagnostics are best here (ugh).
00000347      */
00000348
00000349     proxy = spi_alloc_device(master);
00000350     if (!proxy)
00000351         return NULL;
00000352
00000353     WARN_ON(strlen(chip->modalias) >= sizeof(proxy->modalias));
00000354
00000355     proxy->chip_select = chip->chip_select;
00000356     proxy->max_speed_hz = chip->max_speed_hz;
00000357     proxy->mode = chip->mode;
00000358     proxy->irq = chip->irq;
00000359     strcpy(proxy->modalias, chip->modalias, sizeof(proxy->modalias));
00000360     proxy->dev.platform_data = (void *) chip->platform_data;
00000361     proxy->controller_data = chip->controller_data;
00000362     proxy->controller_state = NULL;
00000363
00000364     status = spi_add_device(proxy);
00000365     if (status < 0) {
00000366         spi_dev_put(proxy);
00000367         return NULL;
00000368     }
00000369
00000370     return proxy;
00000371 }
```

 复制

349至351行，spi_alloc_device函数的定义：

```
 复制
00000229 struct spi_device *spi_alloc_device(struct spi_master *master)
00000230 {
00000231     struct spi_device *spi;
00000232     struct device *dev = master->dev.parent;
00000233
00000234     if (!spi_master_get(master))
00000235         return NULL;
00000236
00000237     spi = kzalloc(sizeof *spi, GFP_KERNEL);
00000238     if (!spi) {
00000239         dev_err(dev, "cannot alloc spi_device\n");
00000240         spi_master_put(master);
00000241         return NULL;
00000242     }
00000243 }
```

```
00000244 spi->master = master;
00000245 spi->dev.parent = dev;
00000246 spi->dev.bus = &spi_bus_type;
00000247 spi->dev.release = spidev_release;
00000248 device_initialize(&spi->dev);
00000249 return spi;
00000250 }
```




234至242行，错误检测和分配内存。

246行，该spi设备属于SPI子系统初始化时注册的那条叫“spi”的总线。

248行，设备模型方面的初始化，不说。

回到spi_new_device函数，355至362行，是一些赋值，其中359行比较关键，设备名字拷贝，362行，之前说过了，设置为NULL。看364行spi_add_device函数的定义：

```
 复制
00000262 int spi_add_device(struct spi_device *spi)
00000263 {
00000264     static DEFINE_MUTEX(spi_add_lock);
00000265     struct device *dev = spi->master->dev.parent;
00000266     struct device *d;
00000267     int status;
00000268
00000269     /* Chipselects are numbered 0..max; validate. */
00000270     if (spi->chip_select >= spi->master->num_chipselect) {
00000271         dev_err(dev, "cs%d >= max %d\n",
00000272             spi->chip_select,
00000273             spi->master->num_chipselect);
00000274         return -EINVAL;
00000275     }
00000276
00000277     /* Set the bus ID string */
00000278     dev_set_name(&spi->dev, "%s.%u", dev_name(&spi->master->dev),
00000279         spi->chip_select);
00000280
00000281
00000282     /* We need to make sure there's no other device with this
00000283      * chipselect **BEFORE** we call setup(), else we'll trash
00000284      * its configuration. Lock against concurrent add() calls.
00000285      */
00000286     mutex_lock(&spi_add_lock);
00000287
00000288     d = bus_find_device_by_name(&spi_bus_type, NULL, dev_name(&spi->dev));
00000289     if (d != NULL) {
00000290         dev_err(dev, "chipselect %d already in use\n",
00000291             spi->chip_select);
00000292         put_device(d);
00000293         status = -EBUSY;
00000294         goto done;
00000295     }
00000296
00000297     /* Drivers may modify this initial i/o setup, but will
00000298      * normally rely on the device being setup. Devices
00000299      * using SPI_CS_HIGH can't coexist well otherwise...
00000300      */
00000301     status = spi_setup(spi);
00000302     if (status < 0) {
00000303         dev_err(dev, "can't %s %s, status %d\n",
00000304             "setup", dev_name(&spi->dev), status);
00000305         goto done;
00000306     }
00000307
00000308     /* Device may be bound to an active driver when this returns */
00000309     status = device_add(&spi->dev);
00000310     if (status < 0)
00000311         dev_err(dev, "can't %s %s, status %d\n",
00000312             "add", dev_name(&spi->dev), status);
00000313     else
00000314         dev_dbg(dev, "registered child %s\n", dev_name(&spi->dev));
00000315
00000316 done:
```

```
00000317     mutex_unlock(&spi_add_lock);
00000318     return status;
00000319 }
```



270至275行，片选号是从0开始的，如果大于或者等于片选数的话则返回出错。

288至295行，遍历spi总线，看是否已经注册过该设备。

301至306行，spi_setup函数的定义：



```
00000645 int spi_setup(struct spi_device *spi)
00000646 {
00000647     unsigned    bad_bits;
00000648     int         status;
00000649
00000650     /* help drivers fail *cleanly* when they need options
00000651      * that aren't supported with their current master
00000652      */
00000653     bad_bits = spi->mode & ~spi->master->mode_bits;
00000654     if (bad_bits) {
00000655         dev_dbg(&spi->dev, "setup: unsupported mode bits %x\n",
00000656             bad_bits);
00000657         return -EINVAL;
00000658     }
00000659
00000660     if (!spi->bits_per_word)
00000661         spi->bits_per_word = 8;
00000662
00000663     status = spi->master->setup(spi);
00000664
00000665     dev_dbg(&spi->dev, "setup mode %d, %s%s%s%s"
00000666         "%u bits/w, %u Hz max --> %d\n",
00000667         (int) (spi->mode & (SPI_CPOL | SPI_CPHA)),
00000668         (spi->mode & SPI_CS_HIGH) ? "cs_high, " : "",
00000669         (spi->mode & SPI_LSB_FIRST) ? "lsb, " : "",
00000670         (spi->mode & SPI_3WIRE) ? "3wire, " : "",
00000671         (spi->mode & SPI_LOOP) ? "loopback, " : "",
00000672         spi->bits_per_word, spi->max_speed_hz,
00000673         status);
00000674
00000675     return status;
00000676 }
```



653至658行，如果驱动不支持该设备的工作模式则返回出错。

663行，调用控制器驱动里的s3c64xx_spi_setup函数，只看前一部分代码：



```
00000795 static int s3c64xx_spi_setup(struct spi_device *spi)
00000796 {
00000797     struct s3c64xx_spi_csinfo *cs = spi->controller_data;
00000798     struct s3c64xx_spi_driver_data *sdd;
00000799     struct s3c64xx_spi_info *sci;
00000800     struct spi_message *msg;
00000801     u32 psr, speed;
00000802     unsigned long flags;
00000803     int err = 0;
00000804
00000805     if (cs == NULL || cs->set_level == NULL) {
00000806         dev_err(&spi->dev, "No CS for SPI(%d)\n", spi->chip_select);
00000807         return -ENODEV;
00000808     }
00000809
...

```



从797行就可以知道在实例化struct spi_board_info时，其controller_data成员就应该指向struct s3c64xx_spi_csinfo的对象。

spi_setup函数结束了，回到spi_add_device函数，309至314行，将该设备加入到设备模型。一直后退，回到spi_register_master函数，就剩下555行of_register_spi_devices这个函数，由于本文所讲的驱动没有使用到设备树方面的内容，所以该函数里什么也没做，直接返回。

到这里，SPI控制器驱动的初始化过程已经说完了。接下来要说的是SPI设备驱动。其实Linux中已经实现了一个通用的SPI设备驱动，另外还有一个是用I/O口模拟的SPI驱动，在这里，只说前者。

初始化函数是在drivers/spi/spidev.c文件中定义：


```
 复制
00000658 static int __init spidev_init(void)
00000659 {
00000660     int status;
00000661
00000662     /* Claim our 256 reserved device numbers. Then register a class
00000663      * that will key udev/mdev to add/remove /dev nodes. Last, register
00000664      * the driver which manages those device numbers.
00000665      */
00000666     BUILD_BUG_ON(N_SPI_MINORS > 256);
00000667
00000668     status = register_chrdev(SPIDEV_MAJOR, "spi", &spidev_fops);
00000669     if (status < 0)
00000670         return status;
00000671
00000672
00000673     spidev_class = class_create(THIS_MODULE, "spidev");
00000674     if (IS_ERR(spidev_class)) {
00000675         unregister_chrdev(SPIDEV_MAJOR, spidev_spi_driver.driver.name);
00000676         return PTR_ERR(spidev_class);
00000677     }
00000678
00000679     status = spi_register_driver(&spidev_spi_driver);
00000680     if (status < 0) {
00000681         class_destroy(spidev_class);
00000682         unregister_chrdev(SPIDEV_MAJOR, spidev_spi_driver.driver.name);
00000683     }
00000684     return status;
00000685 }
```


```
 复制
```

668至670行，注册字符设备，参数spidev_fops是struct file_operations的实例，这里就可以知道，用户程序的open、write等操作最终会调用这里的函数。

673至677行，创建spidev这一类设备，为后面自动生成设备节点做准备。



679至684行，注册spi设备驱动，spi_register_driver函数的定义在drivers/spi/spi.c中：

```
 复制
00000182 int spi_register_driver(struct spi_driver *sdrv)
00000183 {
00000184     sdrv->driver.bus = &spi_bus_type;
00000185     if (sdrv->probe)
00000186         sdrv->driver.probe = spi_drv_probe;
00000187     if (sdrv->remove)
00000188         sdrv->driver.remove = spi_drv_remove;
00000189     if (sdrv->shutdown)
00000190         sdrv->driver.shutdown = spi_drv_shutdown;
00000191     return driver_register(&sdrv->driver);
00000192 }
```


```
 复制
```

184行，该驱动所属的总线。185至190行，一些函数指针的赋值。191行，将驱动注册进设备模型，注册成功的话就会在总线上寻找设备，调用总线上的`match`函数，看能否与之匹配起来，匹配成功的话，驱动中的`probe`函数就会被调用。

参数`spidev_spi_driver`是`struct spi_driver`的实例，它的定义为：

```
 复制  
00000641 static struct spi_driver spidev_spi_driver = {  
00000642     .driver = {  
00000643         .name =         "spidev",  
00000644         .owner =         THIS_MODULE,  
00000645     },  
00000646     .probe =         spidev_probe,  
00000647     .remove =         __devexit_p(spidev_remove),  
00000648  
00000649     /* NOTE:  suspend/resume methods are not necessary here.  
00000650      * We don't do anything except pass the requests to/from  
00000651      * the underlying controller.  The refrigerator handles  
00000652      * most issues; the controller driver handles the rest.  
00000653      */  
00000654 };  
 复制
```

下面看`spidev_probe`函数。在`drivers/spi/spidev.c`中定义的：

```
 复制  
00000565 static int __devinit spidev_probe(struct spi_device *spi)  
00000566 {  
00000567     struct spidev_data     *spidev;  
00000568     int                     status;  
00000569     unsigned long           minor;  
00000570  
00000571     /* Allocate driver data */  
00000572     spidev = kzalloc(sizeof(*spidev), GFP_KERNEL);  
00000573     if (!spidev)  
00000574         return -ENOMEM;  
00000575  
00000576     /* Initialize the driver data */  
00000577     spidev->spi = spi;  
00000578     spin_lock_init(&spidev->spi_lock);  
00000579     mutex_init(&spidev->buf_lock);  
00000580  
00000581     INIT_LIST_HEAD(&spidev->device_entry);  
00000582  
00000583     /* If we can allocate a minor number, hook up this device.  
00000584      * Reusing minors is fine so long as udev or mdev is working.  
00000585      */  
00000586     mutex_lock(&device_list_lock);  
00000587  
00000588     minor = find_first_zero_bit(minors, N_SPI_MINORS);  
00000589  
00000590     if (minor < N_SPI_MINORS) {  
00000591         struct device *dev;  
00000592  
00000593         spidev->devt = MKDEV(SPIDEV_MAJOR, minor);  
00000594  
00000595         dev = device_create(spidev_class, &spi->dev, spidev->devt,  
00000596             spidev, "spidev%d.%d",  
00000597             spi->master->bus_num, spi->chip_select);  
00000598         status = IS_ERR(dev) ? PTR_ERR(dev) : 0;  
00000599     } else {  
00000600         dev_dbg(&spi->dev, "no minor number available!\n");  
00000601         status = -ENODEV;  
00000602     }  
00000603     if (status == 0) {  
00000604  
00000605         set_bit(minor, minors);  
00000606  
00000607         list_add(&spidev->device_entry, &device_list);  
00000608     }  
00000609     mutex_unlock(&device_list_lock);  
00000610 }
```

```
00000611     if (status == 0)
00000612         spi_set_drvdata(spi, spidev);
00000613     else
00000614         kfree(spidev);
00000615
00000616     return status;
00000617 }
```

 复制标签: [Linux驱动](#) 好文要顶 关注我 收藏该文[lknlfy](#)

关注 - 0

粉丝 - 188

[+加关注](#)

0

0

« 上一篇: [Linux设备驱动剖析之SPI（一）](#)» 下一篇: [Linux设备驱动剖析之SPI（三）](#)

posted @ 2013-08-17 20:00 lknlfy 阅读(2599) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。**最新IT新闻:**

- [Uber](#)投资人的“警示信”：十年牛市或见顶，创业者准备提前过冬吧
 - 硅谷巨头纷纷推防手机上瘾功能 但被指力度不足
 - 微信“二次实名认证”实为骗局 警方：谨防上当
 - 老干爹、阿里爸爸...大企业为何要“山寨”自家商标
 - 夏普开始生产智能手机OLED显示屏 帮助苹果摆脱对三星的依赖
- » [更多新闻...](#)

最新知识库文章:

- [如何提高一个研发团队的“代码速度”?](#)
 - [成为一个有目标的学习者](#)
 - [历史转折中的“杭派工程师”](#)
 - [如何提高代码质量?](#)
 - [在腾讯的八年，我的职业思考](#)
- » [更多知识库文章...](#)

Copyright ©2017 lknlfy