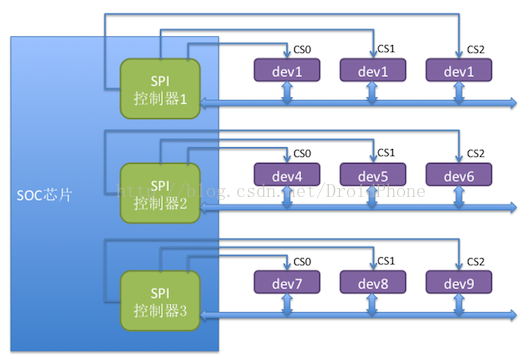
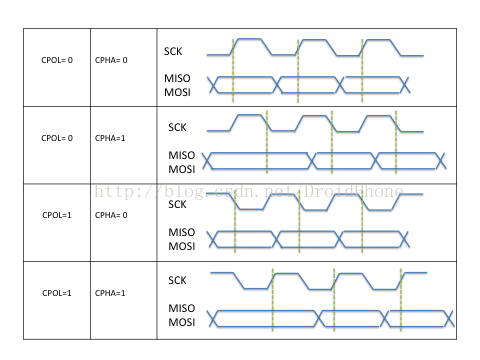
# Linux驱动\_SPI子系统

## 1. SPI子系统概述

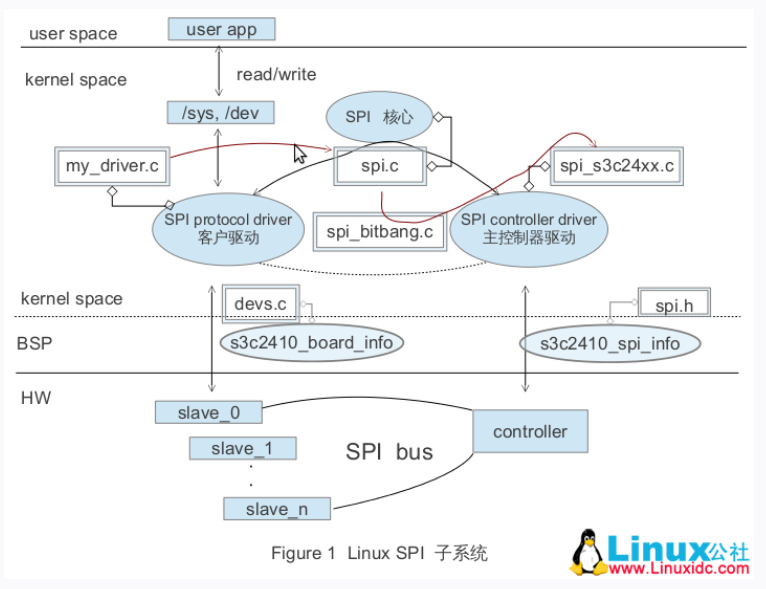
### 1.1 SPI硬件结构



### 1.2 SPI工作4种模式



### 1.3 2410 SPI子系统框架



## 2. SPI设备驱动开发

### 2.1 spi\_register\_board\_info

static struct spi\_board\_info spi\_info\_jz2440**[]** **=** **{**

**{**

**.**modalias **=** "oled"**,** //spi设备名称

**.**max\_speed\_hz **=** 10000000**,** //传输速率

**.**bus\_num **=** 1**,** //使用哪个spi控制器

**.**mode **=** SPI\_MODE\_0**,** //spi模式

**.**chip\_select **=** S3C2410\_GPF**(**1**),** //片选引脚

**.**platform\_data **=** **(**const void **\*)**S3C2410\_GPG**(**4**),** //私有数据

**},**

**};**

static int spi\_info\_jz2440\_init**(**void**)**

**{**

**return** spi\_register\_board\_info**(**spi\_info\_jz2440**,** ARRAY\_SIZE**(**spi\_info\_jz2440**));**

**}**

### 2.2 spi\_register\_driver

static struct spi\_driver spi\_oled\_drv **=** **{**

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

**.**name **=** "oled"**,**

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

**},**

**.**probe **=** spi\_oled\_probe**,**

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

**};**

static int spi\_oled\_init**(**void**)**

**{**

**return** spi\_register\_driver**(&**spi\_oled\_drv**);**

**}**

### 2.3 probe

当上面两者name匹配上时，probe函数调用。

static struct spi\_device **\***spi\_oled\_dev**;**

static struct file\_operations oled\_ops **=** **{**

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

**.**unlocked\_ioctl **=** oled\_ioctl**,**

**.**write **=** oled\_write**,**

**};**

static int \_\_devinit spi\_oled\_probe**(**struct spi\_device **\***spi**)**

**{**

//保存spi\_device

spi\_oled\_dev **=** spi**;**

//获取私有数据，这里为dc的引脚

spi\_oled\_dc\_pin **=** **(**int**)**spi**->**dev**.**platform\_data**;**

//设置dc引脚为输出

s3c2410\_gpio\_cfgpin**(**spi\_oled\_dc\_pin**,** S3C2410\_GPIO\_OUTPUT**);**

//设置spi选择引脚为输出

s3c2410\_gpio\_cfgpin**(**spi**->**chip\_select**,** S3C2410\_GPIO\_OUTPUT**);**

//分配缓冲区保存接收应用层数据

ker\_buf **=** kmalloc**(**4096**,** GFP\_KERNEL**);**

//创建字符设备

major **=** register\_chrdev**(**0**,** "oled"**,** **&**oled\_ops**);**

class **=** class\_create**(**THIS\_MODULE**,** "oled"**);**

//为了让mdev根据这些信息来创建设备节点

device\_create**(**class**,** **NULL,** MKDEV**(**major**,** 0**),** **NULL,** "oled"**);** /\* /dev/oled \*/

**return** 0**;**

**}**

从代码可见spi probe后会产生spi\_device结构体，spi\_oled\_dc\_pin不属于spi协议中的引脚，所以单独处理。

### 2.4 spi读写操作

spi\_write**(**spi\_oled\_dev**,** **&**cmd**,** 1**);**

spi\_write\_then\_read**(**spi\_flash**,** tx\_buf**,** 4**,** rx\_buf**,** 2**);**

spi\_read**(**spi\_flash**,** rx\_buf**,** 2**);**

## 3. SPI控制器驱动开发

### 3.1 spi\_register\_master

spi\_register\_master函数将spi\_master注册到spi子系统中，所以先需要调用spi\_alloc\_master分配一个spi\_master。

static int spi\_s3c2440\_init**(**void**)**

**{**

//2440有两个spi控制器

spi0\_controller **=** create\_spi\_master\_s3c2440**(**0**,** 0x59000000**,** IRQ\_SPI0**);**

spi1\_controller **=** create\_spi\_master\_s3c2440**(**1**,** 0x59000020**,** IRQ\_SPI1**);**

**return** 0**;**

**}**

//num:控制器0/1

//reg\_base\_phy:寄存器地址

//irq:中断号

static struct spi\_master **\***create\_spi\_master\_s3c2440**(**int bus\_num**,** unsigned int reg\_base\_phy**,** int irq**)**

**{**

int ret**;**

struct spi\_master **\***master**;**

struct s3c\_spi\_info **\***info**;**

struct platform\_device **\***pdev**;**

//创建一个platform\_device

pdev **=** platform\_device\_alloc**(**"s3c2440\_spi"**,** bus\_num**);**

platform\_device\_add**(**pdev**);**

//创建一个spi\_master

master **=** spi\_alloc\_master**(&**pdev**->**dev**,** **sizeof(**struct s3c\_spi\_info**));**

//控制器0/1

master**->**bus\_num **=** bus\_num**;**

//该控制器最大的片选个数

master**->**num\_chipselect **=** 0xffff**;**

//该控制器spi支持的模式

master**->**mode\_bits **=** SPI\_CPOL **|** SPI\_CPHA **|** SPI\_CS\_HIGH**;**

//每次传输前进行的设置，例如spi的模式和速率

master**->**setup **=** s3c2440\_spi\_setup**;**

//spi实际传输函数

master**->**transfer **=** s3c2440\_spi\_transfer**;**

//s3c\_spi\_info为spi\_master对应的私有数据

info **=** spi\_master\_get\_devdata**(**master**);**

//存储物理地址/irq/platform\_device

info**->**reg\_base **=** **(**unsigned int**)**ioremap**(**reg\_base\_phy**,** 0x18**);**

info**->**irq **=** irq**;**

info**->**pdev **=** pdev**;**

//硬件初始化

s3c2440\_spi\_controler\_init**(**bus\_num**);**

//申请irq中断

ret **=** request\_irq**(**irq**,** s3c2440\_spi\_irq**,** 0**,** "s3c2440\_spi"**,** master**);**

//注册spi\_master

spi\_register\_master**(**master**);**

**return** master**;**

**}**

### 3.2 s3c2440\_spi\_setup

//每次和spi\_device进行传输前，需要进行一次设置

static int s3c2440\_spi\_setup**(**struct spi\_device **\***spi**)**

**{**

struct s3c\_spi\_info **\***info**;**

struct clk **\***clk**;**

int cpol **=** 0**;**

int cpha **=** 0**;**

int div**;**

//获取spi\_device对应的spi\_master的私有数据

info **=** spi\_master\_get\_devdata**(**spi**->**master**);**

clk **=** clk\_get**(NULL,** "plck"**);**

//根据spi\_device对应的传输模式设置寄存器

**if** **(**spi**->**mode **&** 1**)**

**{**

cpha **=** 1**;**

**}**

**if** **(**spi**->**mode **&** 2**)**

**{**

cpol **=** 1**;**

**}**

/\* 写入SPCON0,1 \*/

/\* [6:5] : 01, polling mode

\* [4] : 1 = enable

\* [3] : 1 = master

\* [2] : CPOL

\* [1] : CPHA

\* [0] : 0 = normal mode

\*/

writeb**((**1**<<**5**)** **|** **(**1**<<**4**)** **|** **(**1**<<**3**)** **|** **(**cpol **<<** 2**)** **|** **(**cpha **<<** 1**),** info**->**reg\_base **+** S3C2410\_SPCON**);**

/\*

\* 设置传输频率

\* Baud rate = PCLK / 2 / (Prescaler value + 1)

\* Prescaler value = PCLK/(2\*Baud rate) - 1

\* Prescaler Value : 0,1,...,255

\*/

div **=** DIV\_ROUND\_UP**(**clk\_get\_rate**(**clk**),** spi**->**max\_speed\_hz **\*** 2**)** **-** 1**;**

clk\_put**(**clk**);**

**if** **(**div **>** 255**)**

div **=** 255**;**

writeb**(**div**,** info**->**reg\_base **+** S3C2410\_SPPRE**);**

**return** 0**;**

**}**

### 3.3 s3c2440\_spi\_controler\_init

配置gpio为硬件spi并使能spi clk。

static void s3c2440\_spi\_controler\_init**(**int which**)**

**{**

struct clk **\***clk **=** clk\_get**(NULL,** "spi"**);**

/\* 使能spi controller 0/1的时钟 \*/

clk\_enable**(**clk**);**

clk\_put**(**clk**);**

/\* GPIO \*/

**if** **(**which **==** 0**)**

**{**

/\* SPI controller 0 \*/

/\*

\* GPE11 SPIMISO

\* GPE12 SPIMOSI

\* GPE13 SPICLK

\*/

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPE**(**11**),** S3C2410\_GPE11\_SPIMISO0**);**

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPE**(**12**),** S3C2410\_GPE12\_SPIMOSI0**);**

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPE**(**13**),** S3C2410\_GPE13\_SPICLK0**);**

**}**

**else** **if** **(**which **==** 1**)**

**{**

/\* SPI controller 1 \*/

/\*

\* GPG5 SPIMISO

\* GPG6 SPIMOSI

\* GPG7 SPICLK

\*/

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPG**(**5**),** S3C2410\_GPG5\_SPIMISO1**);**

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPG**(**6**),** S3C2410\_GPG6\_SPIMOSI1**);**

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPG**(**7**),** S3C2410\_GPG7\_SPICLK1**);**

**}**

**}**

### 3.4 s3c2440\_spi\_transfer

s3c2440\_spi\_transfer仅仅启动传输，剩余的传输在中断完成。

struct s3c\_spi\_info **{**

int irq**;**

unsigned int reg\_base**;**

//休眠操作

struct completion done**;**

//保存当前处理的spi\_transter

struct spi\_transfer **\***cur\_t**;**

//当前已完成的传输个数

int cur\_cnt**;**

struct platform\_device **\***pdev**;**

**};**

static int s3c2440\_spi\_transfer**(**struct spi\_device **\***spi**,** struct spi\_message **\***mesg**)**

**{**

struct spi\_master **\***master **=** spi**->**master**;**

struct s3c\_spi\_info **\***info**;**

struct spi\_transfer **\***t **=** **NULL;**

//获取spi\_master对应的私有数据

info **=** spi\_master\_get\_devdata**(**master**);**

//1. 选中芯片

s3c2410\_gpio\_setpin**(**spi**->**chip\_select**,** 0**);** /\* 默认为低电平选中 \*/

//2. 发数据

//2.1 发送第1个spi\_transfer之前setup

master**->**setup**(**spi**);**

//2.2 从spi\_message中逐个取出spi\_transfer,执行它

list\_for\_each\_entry **(**t**,** **&**mesg**->**transfers**,** transfer\_list**)** **{**

//处理这个spi\_transfer

info**->**cur\_t **=** t**;**

info**->**cur\_cnt **=** 0**;**

init\_completion**(&**info**->**done**);**

//如果该spi\_transfer的speed\_hz或bits\_per\_word不是0, 则调用setup来设置

//假如spi\_transfer中tx\_buf有内容，则为发送

**if** **(**t**->**tx\_buf**)**

**{**

/\* 发送一个字节 \*/

writeb**(((**unsigned char **\*)**t**->**tx\_buf**)[**0**],** info**->**reg\_base **+** S3C2410\_SPTDAT**);**

/\* 它会触发中断 \*/

/\* 休眠 \*/

wait\_for\_completion**(&**info**->**done**);**

**}**

**else** **if(**t**->**rx\_buf**)**

**{**

/\* 接收也是需要发送来驱动，接收的时候随便发送一些数据 \*/

writeb**(**0xff**,** info**->**reg\_base **+** S3C2410\_SPTDAT**);**

/\* 它会触发中断 \*/

/\* 休眠 \*/

wait\_for\_completion**(&**info**->**done**);**

**}**

//如果该spi\_transfer的cs\_change为1则取消片选

**}**

/\* 2.3 唤醒等待的进程 \*/

mesg**->**status **=** 0**;**

mesg**->**complete**(**mesg**->**context**);**

/\* 3. 取消片选 \*/

s3c2410\_gpio\_setpin**(**spi**->**chip\_select**,** 1**);** /\* 默认为低电平选中 \*/

**return** 0**;**

**}**

### 3.5 s3c2440\_spi\_irq

除了第一个字节，其他的传输在中断中完成。

static irqreturn\_t s3c2440\_spi\_irq**(**int irq**,** void **\***dev\_id**)**

**{**

struct spi\_master **\***master **=** **(**struct spi\_master **\*)**dev\_id**;**

struct s3c\_spi\_info **\***info **=** spi\_master\_get\_devdata**(**master**);**

struct spi\_transfer **\***t **=** info**->**cur\_t**;**

**if** **(!**t**)**

**{**

/\* 误触发 \*/

**return** IRQ\_HANDLED**;**

**}**

/\* 处理 spi\_transfer \*/

**if** **(**t**->**tx\_buf**)** /\* 是发送 \*/

**{**

//启动transfer的时候已经发送了一个字节

info**->**cur\_cnt**++;**

**if** **(**info**->**cur\_cnt **<** t**->**len**)**/\* 没发完? \*/

writeb**(((**unsigned char **\*)**t**->**tx\_buf**)[**info**->**cur\_cnt**],** info**->**reg\_base **+** S3C2410\_SPTDAT**);**

**else**

complete**(&**info**->**done**);** /\* 唤醒 \*/

**}**

**else** /\* 接收 \*/

**{**

/\* 读/存数据 \*/

**((**unsigned char **\*)**t**->**rx\_buf**)[**info**->**cur\_cnt**]** **=** readb**(**info**->**reg\_base **+** S3C2410\_SPRDAT**);**

info**->**cur\_cnt**++;**

**if** **(**info**->**cur\_cnt **<** t**->**len**)**/\* 没收完? \*/

writeb**(**0xff**,** info**->**reg\_base **+** S3C2410\_SPTDAT**);**

**else**

complete**(&**info**->**done**);** /\* 唤醒 \*/

**}**

**return** IRQ\_HANDLED**;**

**}**

## 4. SPI子系统代码分析

### 4.1 SPI控制器核心代码

#### 4.1.1 spi\_alloc\_master

struct spi\_master **\***spi\_alloc\_master**(**struct device **\***dev**,** unsigned size**)**

**{**

struct spi\_master **\***master**;**

**if** **(!**dev**)**

**return** **NULL;**

//分配空间，大小为私有数据size+spi\_master大小

master **=** kzalloc**(**size **+** **sizeof** **\***master**,** GFP\_KERNEL**);**

device\_initialize**(&**master**->**dev**);**

master**->**dev**.**class **=** **&**spi\_master\_class**;**

master**->**dev**.**parent **=** get\_device**(**dev**);**

spi\_master\_set\_devdata**(**master**,** **&**master**[**1**]);**

**return** master**;**

**}**

其中spi\_master\_class的在spi\_init初始化。

static struct class spi\_master\_class **=** **{**

**.**name **=** "spi\_master"**,**

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

**.**dev\_release **=** spi\_master\_release**,**

**};**

static int \_\_init spi\_init**(**void**)**

**{**

int status**;**

buf **=** kmalloc**(**SPI\_BUFSIZ**,** GFP\_KERNEL**);**

status **=** bus\_register**(&**spi\_bus\_type**);**

status **=** class\_register**(&**spi\_master\_class**);**

**return** 0**;**

**}**

spi\_init函数向系统注册了一个名为spi的总线类型，同时也为SPI控制器注册了一个名为spi\_master的设备类。这样，以后在sysfs中就会出现以下两个文件节点：

sys**/**bus**/**spi

sys**/**class**/**spi\_master

spi\_master用于描述一个spi控制器。

struct spi\_master **{**

struct device dev**;**

struct list\_head list**;**

//该控制器对应的SPI总线号

s16 bus\_num**;**

//控制器支持的片选数量，即能支持多少个spi设备

u16 num\_chipselect**;**

u16 dma\_alignment**;**

u16 mode\_bits**;**

u16 flags**;**

#define SPI\_MASTER\_HALF\_DUPLEX BIT(0) /\* can't do full duplex \*/

#define SPI\_MASTER\_NO\_RX BIT(1) /\* can't do buffer read \*/

#define SPI\_MASTER\_NO\_TX BIT(2) /\* can't do buffer write \*/

spinlock\_t bus\_lock\_spinlock**;**

struct mutex bus\_lock\_mutex**;**

bool bus\_lock\_flag**;**

//每次传输前设置函数

int **(\***setup**)(**struct spi\_device **\***spi**);**

//传输函数

int **(\***transfer**)(**struct spi\_device **\***spi**,**

struct spi\_message **\***mesg**);**

/\* called on release() to free memory provided by spi\_master \*/

void **(\***cleanup**)(**struct spi\_device **\***spi**);**

bool queued**;**

struct kthread\_worker kworker**;**

struct task\_struct **\***kworker\_task**;**

struct kthread\_work pump\_messages**;**

spinlock\_t queue\_lock**;**

struct list\_head queue**;**

struct spi\_message **\***cur\_msg**;**

bool busy**;**

bool running**;**

bool rt**;**

int **(\***prepare\_transfer\_hardware**)(**struct spi\_master **\***master**);**

int **(\***transfer\_one\_message**)(**struct spi\_master **\***master**,** struct spi\_message **\***mesg**);**

int **(\***unprepare\_transfer\_hardware**)(**struct spi\_master **\***master**);**

**};**

#### 4.1.2 spi\_register\_master

int spi\_register\_master**(**struct spi\_master **\***master**)**

**{**

static atomic\_t dyn\_bus\_id **=** ATOMIC\_INIT**((**1**<<**15**)** **-** 1**);**

struct device **\***dev **=** master**->**dev**.**parent**;**

struct boardinfo **\***bi**;**

int status **=** **-**ENODEV**;**

int dynamic **=** 0**;**

**if** **(!**dev**)**

**return** **-**ENODEV**;**

**if** **(**master**->**num\_chipselect **==** 0**)**

**return** **-**EINVAL**;**

**if** **(**master**->**bus\_num **<** 0**)** **{**

master**->**bus\_num **=** atomic\_dec\_return**(&**dyn\_bus\_id**);**

dynamic **=** 1**;**

**}**

spin\_lock\_init**(&**master**->**bus\_lock\_spinlock**);**

mutex\_init**(&**master**->**bus\_lock\_mutex**);**

master**->**bus\_lock\_flag **=** 0**;**

//创建 /sys/class/spi\_master/spi0

dev\_set\_name**(&**master**->**dev**,** "spi%u"**,** master**->**bus\_num**);**

status **=** device\_add**(&**master**->**dev**);**

//这里定义了传输函数

**if** **(**master**->**transfer**)**

dev\_info**(**dev**,** "master is unqueued, this is deprecated\n"**);**

**else** **{**

status **=** spi\_master\_initialize\_queue**(**master**);**

**}**

mutex\_lock**(&**board\_lock**);**

//加入到spi\_master\_list链表

list\_add\_tail**(&**master**->**list**,** **&**spi\_master\_list**);**

//轮询board\_list链表中的spi\_board\_info

//并检查两者是否匹配，后面再分析匹配的过程

list\_for\_each\_entry**(**bi**,** **&**board\_list**,** list**)**

spi\_match\_master\_to\_boardinfo**(**master**,** **&**bi**->**board\_info**);**

mutex\_unlock**(&**board\_lock**);**

//device tree的spi处理，暂不分析

of\_register\_spi\_devices**(**master**);**

done**:**

**return** status**;**

**}**

### 4.2 SPI设备核心代码

#### 4.2.1 spi\_register\_board\_info

struct boardinfo **{**

struct list\_head list**;**

struct spi\_board\_info board\_info**;**

**};**

int \_\_devinit spi\_register\_board\_info**(**struct spi\_board\_info const **\***info**,** unsigned n**)**

**{**

struct boardinfo **\***bi**;**

int i**;**

bi **=** kzalloc**(**n **\*** **sizeof(\***bi**),** GFP\_KERNEL**);**

//遍历spi\_board\_info数组

**for** **(**i **=** 0**;** i **<** n**;** i**++,** bi**++,** info**++)** **{**

struct spi\_master **\***master**;**

//拷贝spi\_board\_info到boardinfo中

memcpy**(&**bi**->**board\_info**,** info**,** **sizeof(\***info**));**

mutex\_lock**(&**board\_lock**);**

//将boardinfo添加到board\_list链表中

list\_add\_tail**(&**bi**->**list**,** **&**board\_list**);**

//遍历spi\_master\_list，寻找匹配的spi\_master和spi\_board\_info

list\_for\_each\_entry**(**master**,** **&**spi\_master\_list**,** list**)**

spi\_match\_master\_to\_boardinfo**(**master**,** **&**bi**->**board\_info**);**

mutex\_unlock**(&**board\_lock**);**

**}**

**return** 0**;**

**}**

### 4.3 SPI设备驱动核心代码

//将spi\_driver添加到spi\_bus中

int spi\_register\_driver**(**struct spi\_driver **\***sdrv**)**

**{**

sdrv**->**driver**.**bus **=** **&**spi\_bus\_type**;**

**if** **(**sdrv**->**probe**)**

sdrv**->**driver**.**probe **=** spi\_drv\_probe**;**

**if** **(**sdrv**->**remove**)**

sdrv**->**driver**.**remove **=** spi\_drv\_remove**;**

**if** **(**sdrv**->**shutdown**)**

sdrv**->**driver**.**shutdown **=** spi\_drv\_shutdown**;**

**return** driver\_register**(&**sdrv**->**driver**);**

**}**

### 4.4 SPI总线和设备的匹配

static void spi\_match\_master\_to\_boardinfo**(**struct spi\_master **\***master**,** struct spi\_board\_info **\***bi**)**

**{**

struct spi\_device **\***dev**;**

//当spi\_board\_info的bus\_num和spi\_master的bus\_num一致时

//调用spi\_new\_device函数

**if** **(**master**->**bus\_num **!=** bi**->**bus\_num**)**

**return;**

//创建spi\_device

dev **=** spi\_new\_device**(**master**,** bi**);**

**}**

spi\_new\_device将device添加到spi总线上，并匹配名字相同的driver，对应的probe函数调用。

struct spi\_device **\***spi\_new\_device**(**struct spi\_master **\***master**,** struct spi\_board\_info **\***chip**)**

**{**

struct spi\_device **\***proxy**;**

int status**;**

//创建一个spi\_device并进行初步初始化

proxy **=** spi\_alloc\_device**(**master**);**

//将spi\_board\_info的内容存放到spi\_device

proxy**->**chip\_select **=** chip**->**chip\_select**;**

proxy**->**max\_speed\_hz **=** chip**->**max\_speed\_hz**;**

proxy**->**mode **=** chip**->**mode**;**

proxy**->**irq **=** chip**->**irq**;**

strlcpy**(**proxy**->**modalias**,** chip**->**modalias**,** **sizeof(**proxy**->**modalias**));**

proxy**->**dev**.**platform\_data **=** **(**void **\*)** chip**->**platform\_data**;**

proxy**->**controller\_data **=** chip**->**controller\_data**;**

proxy**->**controller\_state **=** **NULL;**

status **=** spi\_add\_device**(**proxy**);**

**return** proxy**;**

**}**

struct spi\_device **\***spi\_alloc\_device**(**struct spi\_master **\***master**)**

**{**

struct spi\_device **\***spi**;**

struct device **\***dev **=** master**->**dev**.**parent**;**

//创建spi\_device

spi **=** kzalloc**(sizeof** **\***spi**,** GFP\_KERNEL**);**

spi**->**master **=** master**;**

spi**->**dev**.**parent **=** **&**master**->**dev**;**

spi**->**dev**.**bus **=** **&**spi\_bus\_type**;**

spi**->**dev**.**release **=** spidev\_release**;**

device\_initialize**(&**spi**->**dev**);**

**return** spi**;**

**}**

int spi\_add\_device**(**struct spi\_device **\***spi**)**

**{**

static DEFINE\_MUTEX**(**spi\_add\_lock**);**

struct device **\***dev **=** spi**->**master**->**dev**.**parent**;**

struct device **\***d**;**

int status**;**

//片选引脚是有效的

**if** **(**spi**->**chip\_select **>=** spi**->**master**->**num\_chipselect**)** **{**

**return** **-**EINVAL**;**

**}**

//设置device名字

dev\_set\_name**(&**spi**->**dev**,** "%s.%u"**,** dev\_name**(&**spi**->**master**->**dev**),** spi**->**chip\_select**);**

mutex\_lock**(&**spi\_add\_lock**);**

//从spi bus中寻找相同名字的device

//如果找到则退出

d **=** bus\_find\_device\_by\_name**(&**spi\_bus\_type**,** **NULL,** dev\_name**(&**spi**->**dev**));**

**if** **(**d **!=** **NULL)** **{**

dev\_err**(**dev**,** "chipselect %d already in use\n"**,**spi**->**chip\_select**);**

put\_device**(**d**);**

status **=** **-**EBUSY**;**

**goto** done**;**

**}**

//调用spi\_master的setup回调

status **=** spi\_setup**(**spi**);**

//将device添加到spi总线上

status **=** device\_add**(&**spi**->**dev**);**

done**:**

mutex\_unlock**(&**spi\_add\_lock**);**

**return** status**;**

**}**

spi\_device用于描述一个spi设备。

struct spi\_device **{**

struct device dev**;**

struct spi\_master **\***master**;**

u32 max\_speed\_hz**;**

u8 chip\_select**;**

u8 mode**;**

u8 bits\_per\_word**;**

int irq**;**

void **\***controller\_state**;**

void **\***controller\_data**;**

char modalias**[**SPI\_NAME\_SIZE**];**

**};**