A31 平台 SPI 设备驱动开发说明文档

2013-02-21

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

版本历史

版本	时间	备注
V1.0	2013-02-21	建立初始版本

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

目 录

1. 前言 1
  1.1. 编写目的 1
  1.2. 适用范围 1
  1.3. 相关人员 1
2. SPI 模块介绍 2
  2.1. 功能介绍 2
  2.2. 硬件介绍 2
  2.3. 源码结构介绍 3
  2.4. 配置介绍 3
3. SPI 体系结构描述 6
4. SPI 常用数据结构描述 7
5. SPI 常用接口描述 10
6. SPI 设备驱动开发 demo 14

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

1. 前言

1.1. 编写目的

了解 SPI 设备驱动在 A31 平台上的开发。

1.2. 适用范围

Allwinner A31 平台。

1.3. 相关人员

A31 平台 SPI 设备驱动开发人员。

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

2. SPI 模块介绍

2.1. 功能介绍

对 SPI 设备的读写操作给予支持。

2.2. 硬件介绍

1）SPI 总线工作原理

SPI 总线通过四条线完成 MCU 与各种外围器件的通讯，这四条线分别是：串行时钟线（SCLK）、主机输出/从机输入数据线（MOSI）、主机输入/从机输出数据线（MISO）、从机片选线（SS）。

当 SPI 工作时，在移位寄存器中的数据被逐位从输出引脚（MOSI）输出（高位在前），同时将输入引脚（MISO）中接收到的数据逐位移入到移位寄存器（高位在前）。因此，在主机发送完一个字节后，从外围器件接收到的数据被移入到主机的移位寄存器中，即完成一个字节数据传输的实质是两个器件寄存器内容的交换。主机的 SPI 时钟信号（SCLK）使传输同步，SPI 总线的内部结构如图 1 所示。

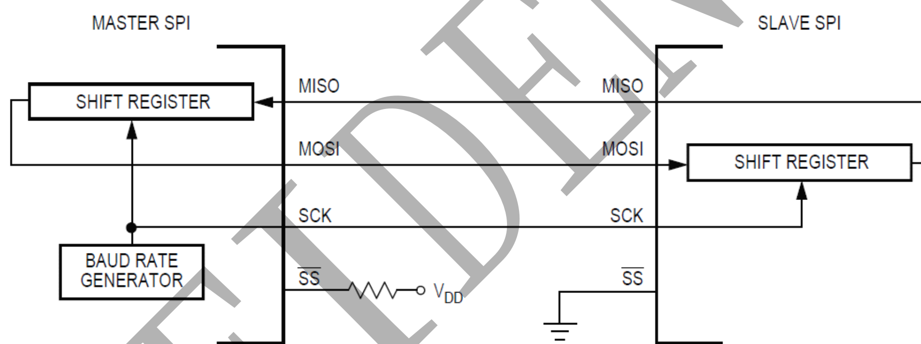

图 1 SPI 总线内部接口图

2）SPI 总线工作模式

根据时钟极性(CPOL)及时钟相位(CPHA)的不同，可以组合成 4 种工作模式，分别是：SPI0 模式，SPI1 模式，SPI2 模式和 SPI3 模式。其中使用最广泛的是 SPI0 模式和 SPI3 模式，详见表 1。

SPI MODE	CPOL	CPHA	Leading Edge	Trailing Edge
0	0	0	Rising，Sample	Failing，Setup
1	0	1	Rising，Setup	Failing，Sample
2	1	0	Failing，Sample	Rising，Setup
3	1	1	Failing，Setup	Rising，Sample

表 1 SPI 总线工作模式

CPOL：CPOL 定义了时钟空闲状态电平，对传输协议没有重大影响。为 0 时，表示时钟空闲状态为低电平；为 1 时，表示时钟空闲状态为高电平。

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

CONFIDENTIAL

CPHA：CPHA 定义了数据的采样时间。为 0 时，表示在时钟的第一个跳变沿(上升沿或下降沿)进行数据采样；为 1 时，表示在时钟的第二个跳变沿(上升沿或下降沿)进行数据采样。

图 2 为 SPI 总线 4 种工作模式的时序对比图。

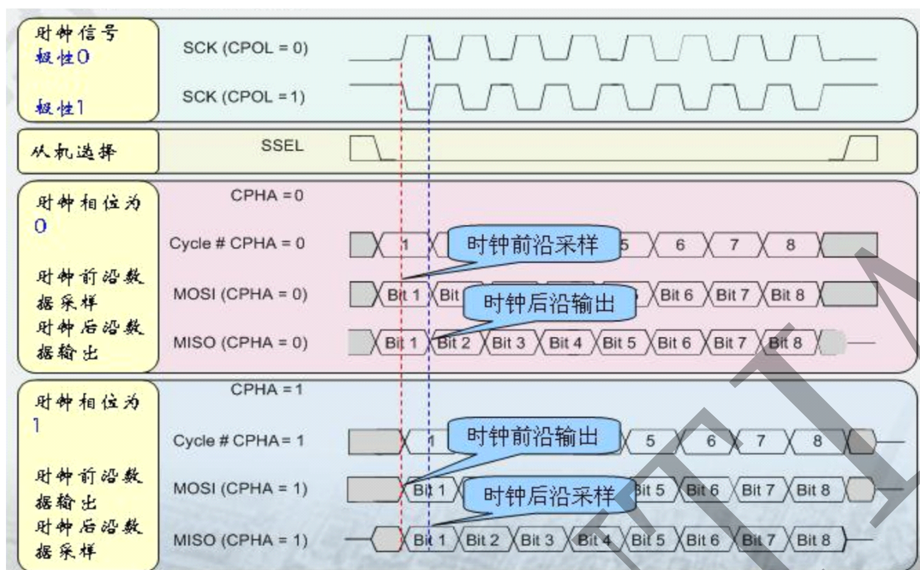

图 2 SPI 总线工作模式时序对比图

2.3. 源码结构介绍

Linux 驱动位于 linux-3.3\drivers\spi\spi-sun6i.c 中。

2.4. 配置介绍

1）sys_config.fex 配置说明：

在 sys_config.fex 中有 4 组 spi 总线可供使用，分别是 spi0、spi1、spi2 和 spi3。配置如下：

```
[spi0_para]
spi_used        = 0

[spi1_para]
spi_used        = 0

[spi2_para]
spi_used        = 0

[spi3_para]
spi_used        = 0
```

其中，若使用哪一组 spi 总线，将对应的 spi_used 置为 1 即可。

2）menuconfig 配置说明：

对于 SPI 总线控制器的配置，可通过命令 make ARCH=arm menuconfig 进入配置主界面，并按以下步骤操作：

首先，选择 Device Drivers 选项进入下一级配置，如图 3 所示：

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

```
.config - Linux/arm 3.3.0 Kernel Configuration
                         Linux/arm 3.3.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        General setup  --->
    [*] Enable loadable module support  --->
    -*- Enable the block layer  --->
        System Type  --->
    [ ] FIQ Mode Serial Debugger
        Bus support  --->
        Kernel Features  --->
        Boot options  --->
        CPU Power Management  --->
        Floating point emulation  --->
        Userspace binary formats  --->
        Power management options  --->
    [*] Networking support  --->
        Device Drivers  --->
        File systems  --->
        Kernel hacking  --->
        Security options  --->
    -*- Cryptographic API  --->
        Library routines  --->
    ---
        Load an Alternate Configuration File
        Save an Alternate Configuration File

                    <Select>    < Exit >    < Help >
```

图 3 Device Drivers 选项配置

然后，选择 SPI support 选项，进入下一级配置，如图 4 所示：

```
.config - Linux/arm 3.3.0 Kernel Configuration
                         Device Drivers
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        Generic Driver Options  --->
    < > Connector - unified userspace <-> kernelspace linker  --->
    < > Memory Technology Device (MTD) support  --->
    < > Parallel port support  --->
    [*] Block devices  --->
        Misc devices  --->
        SCSI device support  --->
    < > Serial ATA and Parallel ATA drivers  --->
    [*] Multiple devices driver support (RAID and LVM)  --->
    < > Generic Target Core Mod (TCM) and ConfigFS Infrastructure  --->
    [*] Network device support  --->
    [ ] ISDN support  --->
    < > Telephony support  --->
        Input device support  --->
        Character devices  --->
    <*> I2C support  --->
    [*] SPI support  --->
        PPS support  --->
        PTP clock support  --->
    -*- GPIO Support  --->
    < > Dallas's 1-wire support  --->
    <*> Power supply class support  --->
    <*> Hardware Monitoring support  --->
    < > Generic Thermal sysfs driver  --->
    [ ] Watchdog Timer Support  --->
        Sonics Silicon Backplane  --->
        Broadcom specific AMBA  --->
        Multifunction device drivers  --->
    [*] Voltage and Current Regulator Support  --->
    <*> Multimedia support  --->
        Graphics support  --->
    <*> Sound card support  --->
    [*] HID Devices  --->

                    <Select>    < Exit >    < Help >
```

图 4 SPI support 选项配置

最后，选择 SUN6I SPI Controller 选项，可选择直接编译进内核，也可以选择编译成模块。如图 5 所示：

CONFIDENTIAL

```
.config - Linux/arm 3.3.0 Kernel Configuration
                                SPI support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

      --- SPI support
      [ ]   Debug support for SPI drivers (NEW)
            *** SPI Master Controller Drivers ***
      < >   Altera SPI Controller (NEW)
      < >   Utilities for Bitbanging SPI masters (NEW)
      < >   GPIO-based bitbanging SPI Master (NEW)
      < >   OpenCores tiny SPI (NEW)
      < >   ARM AMBA PL022 SSP controller (NEW)
      <*>   SUN6I SPI Controller
      < >   Xilinx SPI controller common module (NEW)
      < >   DesignWare SPI controller core support (NEW)
            *** SPI Protocol Masters ***
      < >   User mode SPI device driver support (NEW)
      < >   Infineon TLE62X0 (for power switching) (NEW)

                     <Select>    < Exit >    < Help >
```

图 5 SUN6I SPI Controller 选项配置

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

3. SPI 体系结构描述

位于 drivers/spi 目录下的文件 spi-sun6i.c，是基于 sun6i 平台实现的 SPI 总线控制器驱动。它的职责是为系统中 4 条 SPI 总线实现相应的读写方法，但是控制器驱动本身并不会进行任何的通讯，而是等待设备驱动调用其函数。

图 6 是基于 SUN6I 平台的 SPI 驱动层次架构图，图 6 中有 4 块 SPI master，分别对应 SUN6I 平台上的 4 块 SPI 控制器。

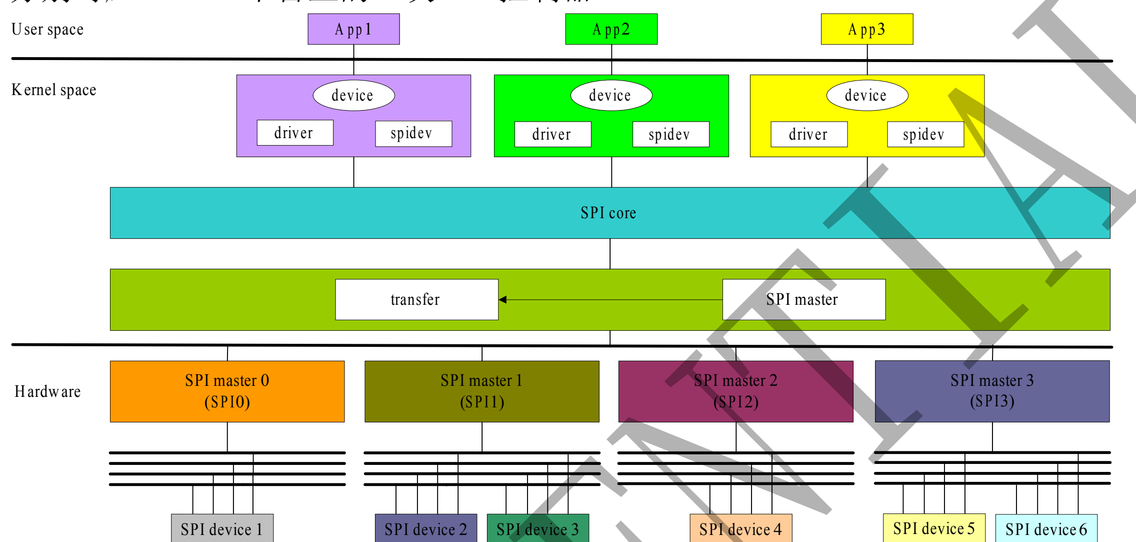

图 6 SPI 驱动层次架构图

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

4. SPI 常用数据结构描述

1）spi_master

```
struct spi_master {
        struct device    dev;
        struct list_head list;
        s16 bus_num;                 /* 总线编号，从零开始 */
        u16 num_chipselect;     /* 支持的片选的数量。从设备的片选号不能大于这个数 */
        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;
        int (*setup)(struct spi_device *spi);       /* 根据 spi 设备更新硬件配置 */
        /* 添加消息到队列的方法。这个函数不可睡眠，它的职责是安排发生的传送并且
调用注册的回调函数 complete() */
        int (*transfer)(struct spi_device *spi, struct spi_message *mesg);
        /* cleanup 函数会在 spidev_release 函数中被调用，spidev_release 被登记为 spi dev
    的 release 函数 */
        void (*cleanup)(struct spi_device *spi);
}
```

spi_master 对应于一个 SPI 控制器，spi_master 注册过程中会扫描 spi_register_board_info 注册的信息，为每一个与本总线编号相同的信息建立一个 spi_device。

2）spi_transfer

```
struct spi_transfer {
        const void *tx_buf; /* 要写入设备的数据(必须是 dma_safe)，或者为 NULL */
        void *rx_buf; /* 要读取的数据缓冲(必须是 dma_safe)，或者为 NULL */
        unsigned len; /* tx 和 rx 的大小(字节数),他们总是相等的 */
        dma_addr_t tx_dma;     /* tx 的 dma 地址 */
        dma_addr_t rx_dma;     /* rx 的 dma 地址*/
        /* 影响此次传输之后的片选，指示本次 transfer 结束之后是否要重新片选并调用
setup 改变设置*/
        unsigned cs_change:1;
        u8 bits_per_word; /* 每个字长的比特数，如果是 0，使用默认值*/
        u16 delay_usecs;          /*此次传输结束和片选改变之间的延时，之后就会启动另一
个传输或者结束整个消息*/
        u32 speed_hz;       /*通信时钟，如果是 0，使用默认值*/
        struct list_head transfer_list;
};
```

spi_transfer 用于描述 SPI 传输，而一次完整的 SPI 传输过程可能不只包含 1 个 spi_transfer，它可能包含多个 spi_transfer，这些 spi_transfer 最终通过 spi_message 组织在一起。

3）spi_message

```
struct spi_message {
        /* 此次消息的传输队列，一个消息可以包含多个传输段 */
```

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

```
        struct list_head transfers;
        struct spi_device     *spi;          /* 传输的目的设备 */
        unsigned is_dma_mapped:1;            /* 表示是否是 DMA 传输方式 */
        void (*complete)(void *context);     /* 异步调用完成后的回调函数 */
        void *context;                       /* 回调函数的参数 */
        unsigned actual_length;              /* 此次传输的实际长度 */
        int status;      /* 执行的结果，成功被置 0，否则是一个负的错误码 */
        struct list_head queue;
        void *state;
};
```

spi_message 用来原子的执行 spi_transfer 表示的一串数组传输请求。这个传输队列是原子的，即在这个消息完成之前不会有其它消息占用 SPI 总线。消息的执行总是按照 FIFO 的顺序。

4）spi_device

```
struct spi_device {
        struct device       dev;
        struct spi_master   *master;                  /* 对应的控制器指针 */
        u32             max_speed_hz;                 /* spi 通信时钟 */
        u8              chip_select;          /* 片选号，用来区分同一控制器上的设备 */
        u8              mode;                 /* 各位的定义如下，主要是传输模式、片选极性 */
    #define  SPI_CPHA     0x01                          /* 时钟相位 */
    #define  SPI_CPOL     0x02                          /* 时钟极性 */
    #define  SPI_MODE_0 (0|0)
    #define  SPI_MODE_1 (0|SPI_CPHA)
    #define  SPI_MODE_2 (SPI_CPOL|0)
    #define  SPI_MODE_3 (SPI_CPOL|SPI_CPHA)
    #define  SPI_CS_HIGH     0x04                       /* 片选电位为高 */
    #define  SPI_LSB_FIRST   0x08                       /* 先输出低比特 */
    #define  SPI_3WIRE    0x10                          /* 输入输出共享接口，此时只能做半双工 */
    #define  SPI_LOOP     0x20
    #define  SPI_NO_CS    0x40
    #define  SPI_READY    0x80
        u8              bits_per_word;                /* 每个字长的比特数 */
        int             irq;                          /* 使用到的中断 */
        void            *controller_state;
        void            *controller_data;
        char            modalias[SPI_NAME_SIZE];      /* 设备名字 */
};
```

spi_device 对应于真实的物理设备，每个 SPI 设备都需要一个 spi_device 来描述。

5）spi_board_info

```
struct spi_board_info {
        char modalias[SPI_NAME_SIZE];         /* 设备名称 */
        /* 私有数据，会被设置到 spi_device.dev.platform_data */
        const void *platform_data;
        /* 私有数据，会被设置到 spi_device.controller_data */
        void *controller_data;
        int irq;                              /* 设备中断号 */
        u32 max_speed_hz;                     /* SPI 的最大速率 */
        u16 bus_num;                          /* SPI 总线的编号 */
        u16 chip_select;                      /* 与片选有关 */
```

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

```
        u8 mode;                        /* 设备的一些模式，例如片选的高低，SPI
的连接方式 */
};
```

spi_board_info 用于存储对应 spi_device 的板级相关信息，包括使用的控制器序号、片选序号、比特率、SPI 传输模式等。

6）spi_driver

```
struct spi_driver {
        const struct spi_device_id *id_table;
        /* 与 spi device 匹配成功后调用，对设备和私有数据进行初始化 */
        int             (*probe)(struct spi_device *spi);
        /* 解除 spi_device 和 spi_driver 的绑定，释放 probe 申请的资源 */
        int             (*remove)(struct spi_device *spi);
        void            (*shutdown)(struct spi_device *spi);    /* 关闭 */
        int             (*suspend)(struct spi_device *spi, pm_message_t mesg);    /* 挂起 */
        int             (*resume)(struct spi_device *spi);      /* 恢复 */
        struct device_driverdriver;
};
```

spi_driver 主要提供驱动模型下的绑定方法和电源管理接口，其成员 driver.name 是和 spi_device 进行匹配的依据。该结构用于绑定在 spi_register_board_info 中注册的对应的 spi_device。

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

5. SPI 常用接口描述

1）spi_register_driver

- ***PROTOTYPE***

  int spi_register_driver(struct spi_driver *sdrv);
- ***ARGUMENTS***

  sdrv the driver to register;
- ***RETURNS***

  init result;

  = 0 init successful;

  < 0 init failed;
- ***DESCRIPTION***

  Register a spi device driver to spi sub-system;

2）spi_unregister_driver

- ***PROTOTYPE***

  static inline void spi_unregister_driver(struct spi_driver *sdrv);
- ***ARGUMENTS***

  sdrv the driver to unregister;
- ***RETURNS***

  None;
- ***DESCRIPTION***

  Unregister a spi device driver from spi sub-system;

3）spi_set_drvdata

- ***PROTOTYPE***

  static inline void spi_set_drvdata(struct spi_device *spi, void *data);
- ***ARGUMENTS***

  spi handle to spi device;
- ***RETURNS***

  None;
- ***DESCRIPTION***

  Set private data to spi device;

4）spi_get_drvdata

- ***PROTOTYPE***

  static inline void *spi_get_drvdata(struct spi_device *spi);
- ***ARGUMENTS***

  spi handle to spi device;
- ***RETURNS***

  The result of get spi device driver data;
- ***DESCRIPTION***

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

Get private data from spi device;

5）spi_write

- ***PROTOTYPE***

  static inline int spi_write(struct spi_device *spi, const u8 *buf, size_t len);
- ***ARGUMENTS***

  spi device to which data will be written;

  buf data buffer;

  len data buffer size;
- ***RETURNS***

  write result;

  = 0 write the buffer succeed;

  < 0 negative error code;
- ***DESCRIPTION***

  SPI synchronous write;

6）spi_read

- ***PROTOTYPE***

  static inline int spi_read(struct spi_device *spi, u8 *buf, size_t len);
- ***ARGUMENTS***

  spi device from which data will be read;

  buf data buffer;

  len data buffer size;
- ***RETURNS***

  read result;

  = 0 read the buffer succeed;

  < 0 negative error code;
- ***DESCRIPTION***

  SPI synchronous read;

7）spi_w8r8

- ***PROTOTYPE***

  static inline ssize_t spi_w8r8(struct spi_device *spi, u8 cmd);
- ***ARGUMENTS***

  spi device with which data will be exchanged;

  cmd command to be written before data is read back;
- ***RETURNS***

  execute result;

  > 0 the eight bit number returned by the device;

  < 0 negative error code;
- ***DESCRIPTION***

  SPI synchronous 8 bit write followed by 8 bit read;

8）spi_w8r16

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

- ***PROTOTYPE***

  static inline ssize_t spi_w8r16(struct spi_device *spi, u8 cmd);
- ***ARGUMENTS***

  spi device with which data will be exchanged;
  cmd command to be written before data is read back;
- ***RETURNS***

  execute result;
  > 0 the sixteen bit number returned by the device;
  < 0 negative error code;
- ***DESCRIPTION***

  SPI synchronous 8 bit write followed by 16 bit read;

9）spi_write_then_read

- ***PROTOTYPE***

  int spi_write_then_read(struct spi_device *spi, const u8 *txbuf, unsigned n_tx, u8 *rxbuf, unsigned n_rx);
- ***ARGUMENTS***

  spi device with which data will be exchanged;
  txbuf data to be written (need not be dma-safe);
  n_tx size of txbuf, in bytes;
  rxbuf buffer into which data will be read (need not be dma-safe);
  n_rx size of rxbuf, in bytes
- ***RETURNS***

  execute result;
  = 0 success;
  < 0 negative error code;
- ***DESCRIPTION***

  SPI synchronous write followed by read;

10）spi_message_init

- ***PROTOTYPE***

  static inline void spi_message_init(struct spi_message *m);
- ***ARGUMENTS***

  m the pointer to spi message;
- ***RETURNS***

  None;
- ***DESCRIPTION***

  Init spi message;

11）spi_message_add_tail

- ***PROTOTYPE***

  static inline void spi_message_add_tail(struct spi_transfer *t, struct spi_message *m);
- ***ARGUMENTS***

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

t the pointer to spi transfer;
m the pointer to spi message;

- ***RETURNS***

None;

- ***DESCRIPTION***

Add spi transfer to spi message quene;

CONFIDENTIAL

Copyright © 2011-2013 Allwinner Technology. All Rights Reserved.

6. SPI 设备驱动开发 demo

以下代码是一个最简单的 SPI 设备驱动 demo，具体代码如下：

```
#include <linux/spi.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>

static int spi_driver_demo_probe(struct spi_device *spi)
{
        struct demo                  *demo;
        struct xxxx_platform_data     *pdata;

        /*  如果驱动需要设备板级信息  */
        pdata = &spi->dev.platform_data;
        if (!pdata)
                return -ENODEV;

        /*  为 demo 结构体开辟内存空间  */
        demo = kzalloc(sizeof *demo, GFP_KERNEL);
        if (!demo)
                return -ENOMEM;
        spi_set_drvdata(spi, demo);

        /*  其他语句  */
        ...

        return 0;
}

static int __devexit spi_driver_demo_remove(struct spi_device *spi)
{
        return 0;
}

static struct spi_driver spi_driver_demo = {
        .probe          = spi_driver_demo_probe,
        .remove         = __devexit_p(spi_driver_demo_remove),
        .driver         = {
                .name   = "xxxx",
                .owner  = THIS_MODULE,
        }
};
```

CONFIDENTIAL

```
static int __init spi_driver_demo_init(void)
{
    return spi_register_driver(&spi_driver_demo);
}

static void __exit spi_driver_demo_exit(void)
{
    spi_unregister_driver(&spi_driver_demo);
}

module_init(spi_driver_demo_init);
module_exit(spi_driver_demo_exit);

MODULE_AUTHOR("anchor");
MODULE_DESCRIPTION("SPI device driver demo");
MODULE_LICENSE("GPL");
```

上面是对 SPI 设备驱动部分的描述。此外，还需要对 SPI 设备进行声明，声明形式如下：

```
static struct xxxx_platform_data = {
    /* 用户定义 */
    …
};
static struct xxxx_platform_data xxxx_pdata __initdata = {
    /* 对定义的成员赋值 */
    …
};
static struct spi_board_info xxxx_spi_board_info[] __initdata = {
    {
        .modalias      = "xxxx",
        .platform_data = &xxxx_pdata,
        .max_speed_hz  = 12 * 1000 * 1000,
        .bus_num = 1,
        .chip_select  = 0,
        .mode = SPI_MODE_3,
    },
};
```

使用以下函数注册上面声明的信息（注册进 SPI 子系统）：

```
spi_register_board_info(xxxx_spi_board_info, ARRAY_SIZE(xxxx_spi_board_info));
```

CONFIDENTIAL