Hi3510 SSP 接口的无线网卡移植

Application Notes

文档版本 01

发布日期 2006-09-16

BOM编码 N/A

深圳市海思半导体有限公司

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。

深圳市海思半导体有限公司

地址： 深圳市龙岗区坂田华为基地华为电气生产中心 邮编：518129

网址： http://www.hisilicon.com

客户服务电话： 0755-28788858

客户服务传真： 0755-28788838

客户服务邮箱： support@hisilicon.com.

版权所有 © 深圳市海思半导体有限公司 2006。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

 、Hisilicon、海思，均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

前 言

概述

本节介绍本文档的内容、对应的产品版本、适用的读者对象、行文表达约定、历史修订记录等。

产品版本

与本文档相对应的产品版本如下所示。

产品名称	产品版本
Hi3510 通信媒体处理器芯片	Hi3510 V100
	Hi3510 V101
	Hi3510 V110
Hi3510 DVS 方案	Hi3510 DMS V100R001

读者对象

本指南为进行无线网卡模块移植的软件工程师和硬件工程师而编写，使用本书的工程师应该：

- 熟练操作 Linux 系统
- 熟练掌握 C 语言
- 掌握基本的 Linux 环境编程
- 熟悉无线网卡的基本知识

内容简介

本文档介绍了 Hi3510 通信媒体处理器芯片（以下简称 Hi3510）SSP 接口实现无线网卡移植的方案，详细描述了移植过程涉及到的硬件接口、软件说明，最后还介绍了无线网卡的配置使用。全书共分为 4 章。

章节	内容
1　概述	简要介绍 Hi3510 芯片 SSP 接口实现无线网卡移植的方案。
2　硬件接口	详细描述无线网卡接口信号和接口读写时序。
3　软件说明	给出详细的移植过程的软件说明。
4　配置使用	详细描述无线网卡的配置使用，且给出操作实例。

约定

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	以本标志开始的文本表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
警告	以本标志开始的文本表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
注意	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备或器件损坏、数据丢失、设备性能降低或不可预知的结果。
窍门	以本标志开始的文本能帮助您解决某个问题或节省您的时间。
说明	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

通用格式约定

格式	说明
宋体	正文采用宋体表示。

格式	说明
黑体	一级、二级、三级标题采用黑体。
楷体	警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。
“Terminal Display”格式	“Terminal Display”格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。

命令行格式约定

格式	意义
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用加粗字体表示。
斜体	命令行参数（命令中必须由实际值进行替代的部分）采用*斜体*表示。
[ ]	表示用“[ ]”括起来的部分在命令配置时是可选的。
{ x | y | ... }	表示从两个或多个选项中选取一个。
[x | y | ...]	表示从两个或多个选项中选取一个或者不选。
{ x | y | ... } *	表示从两个或多个选项中选取多个，最少选取一个，最多选取所有选项。
[x | y | ...] *	表示从两个或多个选项中选取多个或者不选。

图形界面元素引用约定

格式	意义
“”	带双引号“”的格式表示各类界面控件名称和数据表，如单击“确定”。
>	多级菜单用“>”隔开。如选择“文件 > 新建 > 文件夹”，表示选择“文件”菜单下的“新建”子菜单下的“文件夹”菜单项。

修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修改日期	版本	修改说明
2006-09-16	01	第一次发布。

1 概述

无线局域网 WLAN（Wireless LAN）是使用无线连接的局域网（LAN），它使用无线电波作为数据传送的媒介。

无线局域网最通用的标准是 IEEE 定义的无线网络通信工业标准——IEEE802.11 系列。主要包括 802.11a、802.11b 和 802.11g 等。

无线网卡通过 802.11 协议，可以实现无线局域网的接入。常见的无线网卡主要通过 USB 接口，PCI 总线或 SPI 接口实现与主机的连接。

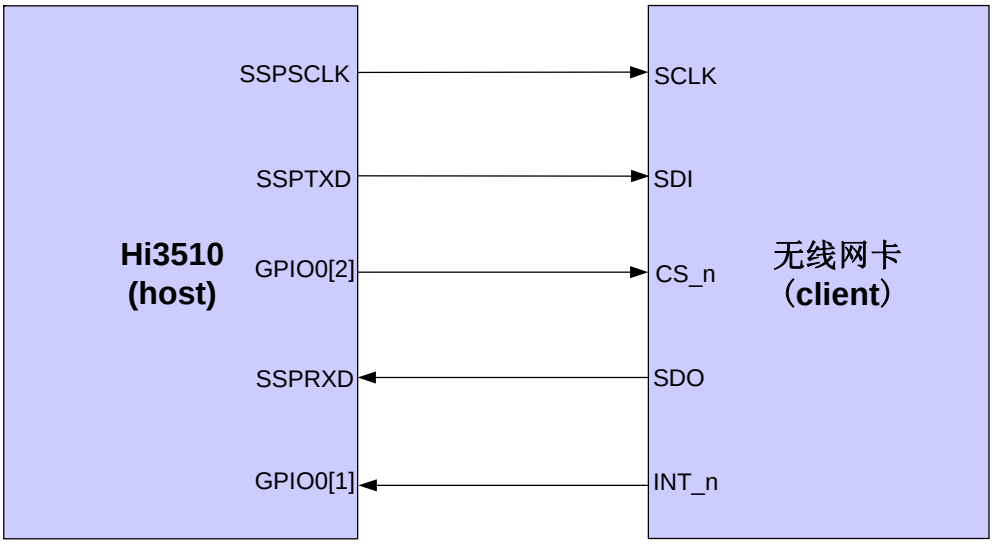
本文描述了在 Linux 操作系统下，如何使用 Hi3510 芯片的 SSP（Synchronous Serial Protocol）接口，实现 SPI（Serial Peripheral Interface）接口的无线网卡的移植、配置和使用。本文中描述的移植过程基于 USI 公司提供的 WIFI 模组，其中 WIFI 芯片是 Marvell 公司的 88w8686，可以支持 802.11b 和 802.11g 两种协议。用来移植的驱动软件包由 Marvell 公司提供。

2 硬件说明

2.1 硬件接口

Hi3510 内部集成 SSP 接口，支持 SPI 模式，可以与无线网卡连接，实现无线网卡的读写操作。接口信号如图 2-1 所示。

图2-1 无线网卡接口信号图



Hi3510 端接口信号的具体含义如表 2-1 所示。

表2-1 Hi3510 接口信号描述

信号	描述
SSPSCLK	SSP 总线时钟。
SSPTXD	SSP 总线数据发送。
GPIO0[2]	片选信号。SSP 接口的片选信号 SSPSFRM 不满足无线网卡 SPI 接口的时序要求，所以使用 GPIO 来模拟。

信号	描述
SSPRXD	SSP 总线数据接收。
GPIO0[1]	无线网卡的中断信号接收。来自无线网卡的中断信号 INT_n 不满足 SSP 接口的时序要求，所以使用 GPIO 来接收。

无线网卡端接口信号的具体含义如表 2-2 所示。

表2-2 无线网卡接口信号描述

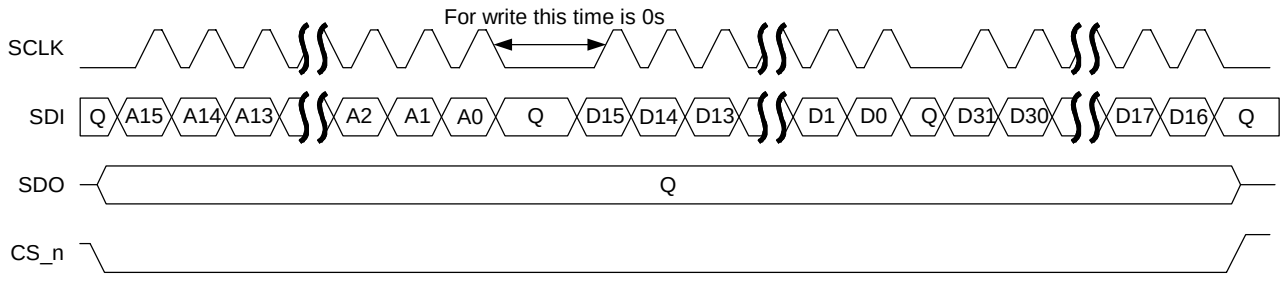
信号	描述
SCLK	时钟输入。
SDI	数据输入。
CS_n	片选输入信号，低有效。
SDO	数据输出。
INT_n	无线网卡发送到 Hi3510 的中断输出，低有效。

2.2 接口时序

本节主要介绍无线网卡 SPI 模式的读写时序，其中的读写都是指对无线网卡的操作。

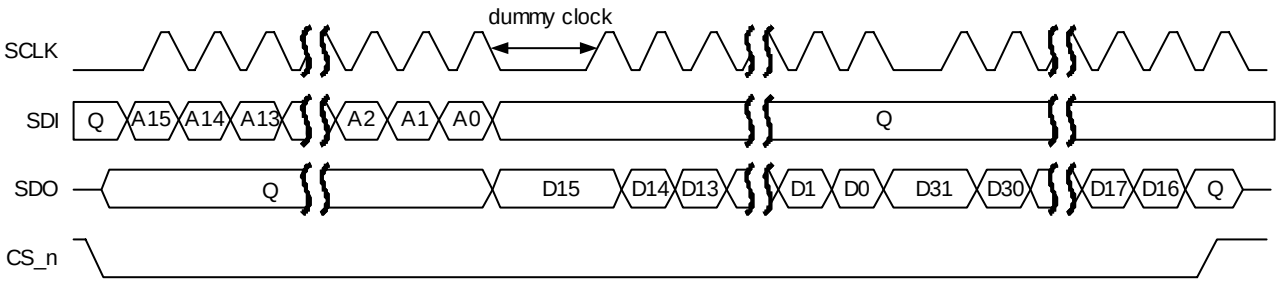
无线网卡的接口写时序如图 2-2 所示。

图2-2 无线网卡接口写时序图



无线网卡的接口读时序如图 2-3 所示。

图2-3 无线网卡接口读时序图



在读时序中，往寄存器写入地址后，读取之前需要若干个 dummy clock 周期，不同的芯片版本和环境可能周期数不同，请在调试过程中确定具体的周期个数，并且这个周期的个数需要在接口程序中指定，具体情况请参见“3 软件说明”。

CS_n 时序是在一次传输的开始就拉低，直到传输结束之后才恢复为高电平的，并且传输过程中不允许拉高，不然网卡端会认为是一次新的传输，导致本次传输失败。

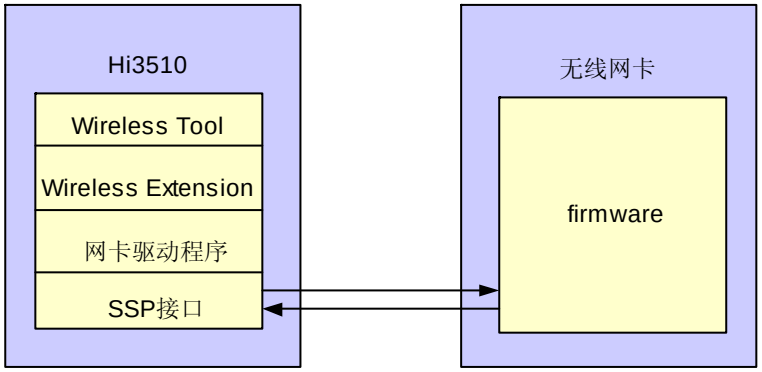
3 软件说明

软件主要包含以下几个部分：

- Wireless Extension
- 网卡驱动
- SSP 接口程序
- 运行在网卡上的程序

其中运行在无线网卡上的程序，由厂商以 firmware 的形式提供。软件结构如图 3-1 所示。

图3-1 软件结构图



3.1 内核加入 Wireless Extension

为了屏蔽对网卡的操作，在 Linux 操作系统里有一个通用的接口层 Wireless Extension，上层的应用是通过这个接口层来访问驱动的，无线网卡的驱动也是依赖于 Wireless Extension 的。要移植厂家的无线网卡驱动包，必须先将这部分内容编译进 Linux 内核。详细内容请参考 Wireless.Extension 的相关网站。

具体方法如下：

步骤 1 进入 Linux 内核源码根目录，运行 make menuconfig，进入“内核编译选项配置”界面。

步骤 2 在该界面做如下选择：

Device Drivers--->

Network device support--->

Wireless LAN (non-hamradio) --->

[*]Wireless LAN drivers (non-hamradio) & Wireless Extensions

<*>Hermes chipset 802.11b support (Orinoco/Prism2/Symbol)

<*>IEEE 802.11 for Host AP (Prism2/2.5/3 and WEP/TKIP/CCMP)

Networking--->

[*]Networking support

<*>IEEE 802.11i CCMP support

<*>IEEE 802.11i TKIP encryption

步骤 3 重新编译内核。

----结束

3.2 编写 SPI 读写程序

为了实现无线网卡 SPI 模式读写，需要 Hi3510 的 SSP 接口驱动程序和 SPI 读写程序。SPI 读写程序的编写，是完成移植的重要工作。

3.2.1 Hi3510 的 SSP 接口驱动程序

这部分程序不需要修改，可以直接使用随 Hi3510 开发包一起提供的 SSP 接口的驱动程序。

SSP 接口驱动程序的用户接口函数如表 3-1 所示。

表3-1 SSP 接口驱动程序的用户接口函数表

函数名	描述
hi_ssp_enable(void);	SSP 接口使能。
hi_ssp_disable(void);	SSP 接口不使能。
hi_ssp_set_frameform(unsigned char,unsigned char,unsigned char,unsigned char);	SSP 接口模式设置。
hi_ssp_set_serialclock(unsigned char,unsigned char);	SSP 接口时钟设置。

函数名	描述
hi_ssp_set_inturrupt(unsigned char);	SSP 接口中断设置。
hi_ssp_interrupt_clear(void);	SSP 接口中断清除。
hi_ssp_dmac_enable(void);	SSP 接口 DMA 模式使能。
hi_ssp_dmac_disable(void);	SSP 接口 DMA 不使能。
hi_ssp_busystate_check(void);	SSP 接口是否忙判断。
hi_ssp_readdata(void);	SSP 接口读取数据。
hi_ssp_writedata(unsigned short);	SSP 接口写数据。
hi_ssp_dmac_init(void *,void *);	SSP 接口 DMA 模式初始化。
hi_ssp_dmac_transfer(unsigned int,unsigned int,unsigned int);	SSP 接口 DMA 模式传送数据，同时发送和接收。
hi_ssp_dmac_exit(void);	退出 SSP 接口的 DMA 模式。

3.2.2 SPI 读写程序

这部分程序需要重新开发，需要实现的内容如下。

初始化 Hi3510 的 SSP 接口

使用 SSP 接口的 DMA 模式和 CPU 模式初始化过程是不一样的，具体情况请参考下面的程序。

```
hi_ssp_disable();
hi_ssp_set_frameform(SSP_CR0_FRF_MOT,
                     SSP_CR0_SPH,SSP_CR0_SPO,
                     SSP_CR0_DSS_16);
#if (USE_DMA == 1)
    hi_ssp_set_serialclock(0,6);
#else
    hi_ssp_set_serialclock(0,10);
#endif
    hi_ssp_set_inturrupt(0);
#if (USE_DMA == 1)
    hi_ssp_dmac_enable();
#else
    hi_ssp_dmac_disable();
#endif
    hi_ssp_enable();
hi_ssp_interrupt_clear();
```

```
#if (USE_DMA == 1)
    ret = wifi_dmac_ssp_init(&gspiinfo->mapreadbufaddr,
                             &gspiinfo->mapwritebufaddr);
        if(ret)
        {
            printk("wifi_dmac_ssp_init error \n");
        }
#endif
```

实现 CPU 模式和 DMA 模式的读写函数

CPU 模式主要用于调试过程。在实际使用的过程中，为了保证比较小的 CPU 占用率，我们优先考虑使用 DMA 模式。CPU 和 DMA 模式的读写函数的实现请参考下面的程序。

说明

gspi_write_data_direct 是向上层提供的 SPI 模式写函数，gspi_read_data_direct 是向上层提供的 SPI 模式读函数，USE_DMA 部分是 DMA 模式专用的的程序。

- 写程序

```
int gspi_write_data_direct(gspi_card_rec_p cardp,
                           u8 * data, u16 reg,
                           u16 n)
{
gspihost_info_t *gspiinfo = cardp->ctrlr;
unsigned short tmp;
#if (USE_DMA == 1)
    u16 dma_trans_size;
    int retry = 0;
    int timeout = 100;
    DECLARE_WAITQUEUE(wait, current);
#endif //USE_DMA

_ENTER();

if (!cardp || !gspiinfo) {
    {
    printk("Cardp null\n");
    return -1;
    }

if (gspi_acquire_io(gspiinfo))
    {
    printk(KERN_ERR "gspi_acquire_io failed\n");
    return -1;
    }
```

```
    /* N bytes are sent wrt to 16 bytes, convert it to 8 bytes */
    n = (n * 2);

    ssp_sfrm(SFRM_DOWN);

    reg |= 0x8000;

    memcpy(gspiinfo->iodata, &reg, sizeof(u16));
    memcpy(gspiinfo->iodata + sizeof(u16), data, (n - 2));

    wmb();
#if (USE_DMA == 1)
    memcpy((char *)dmac_ssp_s.ssptxchbufmapaddress, gspiinfo->iodata, n);
    do {
        dma_trans_size = n/2;
        ssp_dma_add_wait(&wait);
        if(hi_ssp_dmac_transfer(dmac_ssp_temp_s.rxbufaddress,
                                dmac_ssp_s.ssptxchbufaddress,
                                dma_trans_size) !=0)
            {
            PRINTK("read dma transfer wrong ,retry.\n");
            retry = 1;
            }
        schedule();
        ssp_dma_rm_wait(&wait);
        }
    while (retry && --timeout);
    gspiinfo->dma_txack = 0;

#else
    /*CPU mode  Write data through SSDR by CPU mode*/
    {
    int i;
    u16 *dat = (u16 *) gspiinfo->iodata;

    for (i = 0; i < (n / 2); i++)
        {
        while (hi_ssp_busystate_check());
        hi_ssp_writedata(*dat);
        _PRINTK("ssp write: 0x%04x \n",(unsigned int)*dat);

        while (hi_ssp_busystate_check());
        tmp = hi_ssp_readdata();
```

```
        _PRINTK("ssp read: 0x%04x \n",(unsigned int)tmp);

        dat++;
        }
    }
#endif //USE_DMA

    if ((n % 4) != 0)
    {
        while (hi_ssp_busystate_check());
        hi_ssp_writedata(0);
        _PRINTK("ssp write: 0x%04x \n",0);

        while (hi_ssp_busystate_check());
        tmp = hi_ssp_readdata();
        _PRINTK("ssp read: 0x%04x \n",(unsigned int)tmp);
    }

    ssp_sfrm(SFRM_UP);
    gspi_release_io(gspiinfo);

    _LEAVE();
    return 0;
}
```

- 读程序

```
int gspi_read_data_direct(gspi_card_rec_p cardp,
                          u8 * data, u16 reg,
                          u16 n)
{
    gspihost_info_t *gspiinfo;
    u16 tmp;
#if (USE_DMA == 1)
    int retry = 0;
    u16 dma_trans_size;
      int timeout = 100;
int i;
   DECLARE_WAITQUEUE(wait, current);
#endif

    _ENTER();

    if (!cardp)
        {
```

```
            printk("Cardp null\n");
            return -1;
            }

        gspiinfo = cardp->ctrlr;

        if (gspi_acquire_io(gspiinfo))
            {
            printk(KERN_ERR "gspi_acquire_io failed\n");
            return -1;
            }

        n = ((n + g_dummy_clk_ioport) * 2);

    #if (USE_DMA == 1) //DMA mode
        do
            {
            retry = 0;
            dma_trans_size = n/2;
            _PRINTK("DMA read start,trans_size:%d.n:%d \n",dma_trans_size,n);

            ssp_sfrm(SFRM_DOWN);
            udelay(1);
            wmb();
            while (hi_ssp_busystate_check());
            hi_ssp_writedata(reg);
            //_PRINTK("ssp write: 0x%04x \n",(unsigned int)reg);
            while (hi_ssp_busystate_check());
            tmp = hi_ssp_readdata();
            //_PRINTK("ssp read: 0x%04x \n",(unsigned int)tmp);

            ssp_dma_add_wait(&wait);
            if(hi_ssp_dmac_transfer(dmac_ssp_s.ssprxchbufaddress,
                                    dmac_ssp_temp_s.txbufaddress,
                                    dma_trans_size) !=0)
                {
                _PRINTK("read    dma transfer wrong ,retry.\n");
                retry = 1;
                }
            schedule();
            ssp_dma_rm_wait(&wait);
            gspiinfo->dma_txack = 0;
            ssp_sfrm(SFRM_UP);
            rmb();
```

```
            _PRINTK("DMA read end,the data read is: \n");
            for (i = 0; i < dma_trans_size; i++)
                {
                _PRINTK("0x%04x: \n",(int)*((u16*)
                        (dmac_ssp_s.ssprxchbufmapaddress+i*2)));
                }

            memcpy(gspiinfo->iodata,
                    (unsigned char *)dmac_ssp_s.ssprxchbufmapaddress,
                    dma_trans_size*2);

        } while (retry && --timeout);

        if (!timeout)
            {
            printk("Timeout for gspi_read_data_direct\n");
            }
    #else //CPU mode
        {
        int i;

        ssp_sfrm(SFRM_DOWN); //cs_n down
        {
        u16 *dat = (u16 *) gspiinfo->iodata;

        wmb();
        while (hi_ssp_busystate_check());
        hi_ssp_writedata(reg);
        _PRINTK("ssp write: 0x%04x \n",(unsigned int)reg);

        while (hi_ssp_busystate_check());
        tmp = hi_ssp_readdata();
        _PRINTK("ssp read: 0x%04x \n",(unsigned int)tmp);

        for (i = 0; i < (n / 2); i++)
            {
            while (hi_ssp_busystate_check());
            hi_ssp_writedata(0x0000);
            PRINTK("ssp write %d: 0x%04x \n", i, (unsigned int)0x0000);

            while (hi_ssp_busystate_check());
            *dat = hi_ssp_readdata();
            _PRINTK("ssp read %d: 0x%04x \n", i, (unsigned int)*dat);
```

```
            dat++;
            }
        }

        ssp_sfrm(SFRM_UP);
        rmb();
        }
#endif //USE_DMA

        wmb();
        memcpy(data, gspiinfo->iodata + (g_dummy_clk_ioport + 1) * 2,
                (n - (g_dummy_clk_ioport + 1) * 2));
        gspi_release_io(gspiinfo);

        _LEAVE();
        return 0;
}
```

实现 GPIO 的中断程序

中断程序主要包括下面两个部分：

- 初始化 GPIO

```
void gpio_init(void)
{
    unsigned int tmp;
    ENTER();
    gpio_remap();

/* set gpio_0_1 dir: in  */
    gpio_dirsetbit(GPIO_0,1,0);

    /* set gpio_0 interrupt mode */
gpio_interruptset_byte(GPIO_0,
                        0x02,
                        SENSE_SINGLE,
                        SENSE_EDGE,
                        EVENT_FALLING_EDGE
                          );

    /* disable gpio_0_1 interrupt */
    tmp = 0x02;
    gpio_interruptdisable_byte(GPIO_0,tmp);
```

```
    /* clear the int_flags of gpio_0_1 */
    gpio_interruptclear_byte(GPIO_0,tmp);

    /* enable interrupt gpio_0_1 */
    gpio_interruptenable_byte(GPIO_0,tmp);

    _LEAVE();
}
```

- 注册中断服务程序

说明

这里我们使用 GPIO0[1]作为中断输入，GPIO0 的中断号为 6，所以这里配置为 6。

```
#define IRQ_GPIO0 6
int gspi_register_irq(gspihost_info_p gspiinfo)
{
    gspi_card_rec_p cardp = gspiinfo->card;
    gspiinfo->irq = IRQ_GPIO0;

    if (request_irq(gspiinfo->irq,
                    cardp->user_isr,
                    SA_SHIRQ, cardp->magic,
                    cardp->user_arg))
        {
        printk("failed to request GPIO0 as gpio_irq(%d)\n",
                gspiinfo->irq);
        return -1;
        }

    schedule_timeout(3 * HZ);
    return 0;
}
```

实现 CS_n 函数

由于 Hi3510 的 SSP 接口的 CS 信号不能满足 SPI 模式的时序要求，因此通过软件拉高和拉低 GPIO 管脚，来模拟 CS_n 信号

说明

输入参数为 SFRM_DOWN 时拉低 CS_n 信号，输入参数为 SFRM_UP 时拉高 CS_n 信号。

```
ssp_sfrm(int updown)
{
    unsigned long tmp;

    if (updown == SFRM_DOWN)
    {
```

```
    writel(4|readl(IO_ADDRESS(SSP_GPIO0_DIR)),IO_ADDRESS(SSP_GPIO0_DIR));
    tmp = readl(IO_ADDRESS(SSP_GPIO0_DIR));
    PRINTK("SSP_GPIO0_DIR: 0x%04x\n", (unsigned int)tmp);

    writel(0,IO_ADDRESS(SSP_GPIO0_2));
    tmp = readl(IO_ADDRESS(SSP_GPIO0_2));
    PRINTK("SSP_GPIO0_2: 0x%04x\n", (unsigned int)tmp);
    }
    else
    {
    writel(4|readl(IO_ADDRESS(SSP_GPIO0_DIR)),IO_ADDRESS(SSP_GPIO0_DIR));
    tmp = readl(IO_ADDRESS(SSP_GPIO0_DIR));
    PRINTK("SSP_GPIO0_DIR: 0x%04x\n", (unsigned int)tmp);

    writel(4,IO_ADDRESS(SSP_GPIO0_2));
    tmp = readl(IO_ADDRESS(SSP_GPIO0_2));
    PRINTK("SSP_GPIO0_2: 0x%04x\n", (unsigned int)tmp);
    }
    return;
}
```

3.3 修改网卡驱动程序

网卡的驱动程序是由厂家提供的开发包，需要做一下简单的修改，主要修改的地方有两个。

加入清除中断的操作

由于移植过程中使用 GPIO 来模拟 SPI 模式的中断信号，而厂家提供的驱动开发包的中断服务程序里面没有清除中断的操作，所以需要增加以下操作：

- GPIO 的清除中断操作。
- 处理中断的过程中的中断屏蔽操作。

具体请参见开发包的 if_gspi.c 文件。

下面的程序可供参考：

```
static IRQ_RET_TYPE sbi_interrupt(int irq,
                                  void *dev_id,
                                  struct pt_regs *fp)
    {
    struct net_device *dev = dev_id;
    unsigned int tmp;

    PRINTM(INFO, "In the interrupt handler\n");
```

```
    /* disable gpio_0_1 interrupt */
    tmp = 0x02;
    gpio_interruptdisable_byte(GPIO_0,tmp);

    /* clear the int_flags of gpio_0_1 */
    gpio_interruptclear_byte(GPIO_0,tmp);
    disable_irq(dev->irq);

    wlan_interrupt(dev);

    /* enable interrupt gpio_0_1 */
    gpio_interruptenable_byte(GPIO_0,tmp);

    IRQ_RET;
}
```

加入初始化成功的提示信息

原来的驱动软件里在网卡模块初始化成功时没有提示信息，因此在网卡模块初始化的操作中需要加入该信息。另外在网卡模块初始化过程中还有一个 firmware 的加载过程，也相应的加入了加载成功的提示信息。

4 配置使用

4.1 插入模块

模块插入过程如下：

步骤 1 将 helper_gspi.bin 和 gspi8686.bin 拷贝到 rootbox/lib/firmware/目录下。

在加载驱动模块的同时，需要下载 firmware 到无线网卡，所以在加载驱动之前需要先将 firmware（gspi8686.bin）和加载的辅助程序（helper_gsbi.bin）拷贝到文件系统的指定目录下，并且在插入模块的时候指定其路径。

步骤 2 使用下面脚本插入模块。

```
insmod hi_ssp.ko
insmod gspi.ko
insmod gspi8686.ko helper_name= /lib/firmware/ helper_gspi.bin
fw_name=/lib/firmware/ gspi8686.bin
```

----结束

由于网卡是在 Hi3510 的 SSP 接口的 SPI 模式下连接的，所以需要按顺序加载以下模块：

1. SSP 模块（hi_ssp.ko）
2. SPI 模式读写模块（gspi.ko）
3. 无线网卡的驱动模块（gspi8686.ko）

加载成功提示信息如下（需要修改软件包增加提示信息，请参见“3.3 修改网卡驱动程序”）：

```
Firmware download successful.
Gspi8686 Init OK.
```

4.2 配置网卡

网卡启动成功之后需要使用无线网卡的配置工具（例如 Wireless Tool）对网卡进行配置，以便连接到 WLAN。配置工具的详细使用请参考使用帮助或相关网站。如果使用 Wireless Tool 下面的配置脚本可供参考：

```
ifconfig eth2 up
iwconfig eth2
iwconfig eth2 mode managed
iwconfig eth2 essid "mywlan"
iwconfig eth2 rate auto
ifconfig eth2 192.168.0.3 netmask 255.255.255.0 up
```

配置完成之后使用 iwlist 命令可以查看网络中的 AP 资源：

```
iwlist eth2 scanning
```

如果网卡配置成功会显示搜索到的 AP 的信息。

注意

在确认是否与其他计算机连通前，请断开有线网卡的连接。

搜索到 AP 后，运行 ping 命令，查看是否与 WLAN 上的其他计算机连通，如果连接成功表示移植成功。您就可以正常使用无线网络设备了。

4.3 使用示例

在确定无线网卡连接到 WLAN 之后，您就可以通过无线网口使用各种上层的网络协议进行网络通信了。

下面是使用 FTP 程序通过无线网卡下载和上传文件的例子。

首先您需要建立一个 FTP 服务器，假设 FTP 服务器的 IP 地址为 192.168.0.1。

在调试命令行输入：

```
ftpget 192.168.0.1 myfile remotefile
```

文件就可以下载到本地的文件系统了（当然您的系统必须支持 ftpget 命令）。

还可以通过 FTP 命令上传文件，命令如下：

```
ftpput 192.168.0.1 remotefile myfile
```