

ASM9260T 使用遇到的问题汇总

作者:向仔州

文件系统存储文件问题.....	2
文件系统烧写自己做的文件，下载问题.....	2
TFTP 下载文件问题.....	3
Asm9260t Alsa 操作问题.....	4
网口使用 udhcpc 无法获取 DNS 问题.....	5
内核编译后生成 kernel 烧写文件遇到的问题.....	5
Wifi RTL8188eus 编译，安装.....	6
ASM9260T USB 烧写程序注意事项.....	8
交叉编译器里面库文件和头文件路径问题.....	8
ASM9260T-GPIO 使用方法.....	9
GPIO 作为中断的使用方法.....	11
ASM9260T 使用 RTL8188EUS，启动 AP 功能方法.....	14
ASM9260T 驱动网卡芯片 KSZ8041RNL 网线接口检测问题.....	18
VBUS 电源内核报错 arch_interrupt 85 : VBUS error workaround.....	19
ASM9260T 挂载 SD 卡问题.....	20
ASM9260T 不同 linux2.6.32 内核版本的 SD 卡内核启动无法产生/dev 节点问题.....	21
SD 卡出现只能读无法写数据的问题.....	22
SocketCAN 总线移植到 linux2.6.32 上的方法.....	23
ASM9260T CAN 引脚映射设置，比如我想其它规定的 GPIO 也有 CAN 功能.....	24
电路板启动后文件系统自启动程序流程.....	24
增大 yaffs2 文件系统存储容量，需要修改 uboot.....	26
擦除清空 yaffs2 目录下的文件.....	27

文件系统存储文件问题

CRAMFS 文件系统是专门针对闪存设计的只读压缩的文件系统，其容量上限为 256M,采用 zlib 压缩，文件系统类型可以是 EXT2 或 EXT3

所以你在文件系统任意目录下存放的文件在开发板重启后都会被删除

```
# ls
bin      etc      lib      mnt      proc      sys      usr
dev      home     linuxrc   opt      sbin      tmp      var
#
```

但是 asm9260t 又在/mnt 下自动挂载了 yaffs2 文件系统用来存放写入和读出的文件

```
# ls
hello.c      lost+found  test.wav
# pwd
/mnt/yaffs2
#
```

在/mnt/yaffs2 下可以存储文件

检查磁盘空间大小

```
# df -h
Filesystem      Size  Used Available Use% Mounted on
tmpfs           13.7M    0    13.7M  0% /tmp
tmpfs           13.7M    0    13.7M  0% /dev
/dev/mtdblock3  50.0M   3.0M   47.0M  6% /mnt/yaffs2
#
```

发现 yaffs2 只能存储 50M 的东西，但是后面可以修改因为 Nandflash 是 128M 的

文件系统烧写自己做的文件，下载问题

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt/yaffs2# ls
iwlist simple.script wpa_cli wpa.conf wpa_supplicant
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt/yaffs2#
```

只有在开发板上用 U 盘把自己做的文件拷贝在 roms/mnt/yaffs2 目录上是可以的

如果你是想先把自己做的文件拷贝在 roms 文件系统的/mnt/yaffs2 目录下，然后烧写编译进去，这样的话开发板 yaffs2 目录还是没有你拷贝的文件

yaffs2 只支持你开发板 u 盘拷贝的文件，如果是在 PC 机上做文件系统烧录，那你必须把文件放在除了 yaffs2 以外的其他地方，这样你烧录进开发板文件才不会丢失。

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt# ls
cramfs  etc  ext  fat  nfs  sd  test  tmp  ubifs  udisk  yaffs2
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt#
```

你的命令工具可以都放在除 yaffs2 以外的其他目录下执行。但是你的读写文本文件，conf 后缀文件，必须放在 yaffs2 可读可写目录下。

TFTP 下载文件问题

```
# tftp -g -r hello 192.168.14.12
tftp: can't open 'hello': Read-only file system
#
```

但是我 ping 主机是通的

```
# ping 192.168.14.12
PING 192.168.14.12 (192.168.14.12): 56 data bytes
64 bytes from 192.168.14.12: seq=0 ttl=64 time=10.000 ms
64 bytes from 192.168.14.12: seq=1 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=2 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=3 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=4 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=5 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=6 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=7 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=8 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=9 ttl=64 time=0.000 ms
64 bytes from 192.168.14.12: seq=10 ttl=64 time=0.000 ms
^C
--- 192.168.14.12 ping statistics ---
11 packets transmitted, 11 packets received, 0% packet loss
round-trip min/avg/max = 0.000/0.909/10.000 ms
```

而且主机自己 tftp localhost 测试也是通的。

```
round-trip min/avg/max
# cd /mnt/yaffs2/                                然后进入 yaffs2 目录
# ls
lost+found  test.wav
# pwd
/mnt/yaffs2
#
# tftp -g -r hello 192.168.14.12
# ls
hello      lost+found  test.wav
# ./hello
-/bin/sh: ./hello: Permission denied
# ll
-/bin/sh: ll: not found
# ls -l
-rw-r--r--    1 0          0                  5733 Jan  1 01:12 hello
drwx-----    1 0          0                  2048 Jan  1 00:00 lost+found
-rw-r--r--    1 0          0                1920044 Jan  1 00:34 test.wav
# chmod 755 hello
# ./hello
hello C .....
#
```

最后发现可以了，看来和文件系统 cramfs 问题一样，其他位置都是只读的，只要 yaffs2 可以自由读写

Asm9260t Alsa 操作问题

```
null          tty27
pcmC0D0c      tty28
pcmC0D0p      tty29
ptmx          tty3
pts            tty30
ram0          tty31
#
```

ALSA 的设备节点是在 /dev 目录下的，但是我们

的 aplay, amixer, arecord 软件都是默认在 /dev/snd 目录下去找，所以我要重新映射设备节点的硬连接到 snd 目录下

```
mkdir /dev/snd
ln /dev/controlC0 /dev/snd/controlC0
ln /dev/pcmC0D0c /dev/snd/pcmC0D0c
ln /dev/pcmC0D0p /dev/snd/pcmC0D0p
ln /dev/timer /dev/snd/timer

# mkdir /dev/snd
cmC0D0p # ln /dev/controlC0 /dev/snd/controlC0
# ln /dev/pcmC0D0c /dev/snd/pcmC0D0c
# ln /dev/pcmC0D0p /dev/snd/pcmC0D0p
# ln /dev/timer /dev/snd/timer
# ls
```

```
# ls /dev/snd/
controlC0  pcmC0D0c  pcmC0D0p  timer
#
```

然后可以正常操作 alsa 了

芯片默认上电 line 和耳机是直通的

amixer cset numid=12 off

1. 配置声卡线性输入

amixer cset numid=4 1

2. 配置声卡 playback 输出

amixer cset numid=14 1

3. 录音，该命令可录制 10s 长度 48000 波特率 16 位的 wav 音频文件

arecord -d 10 -fdat test.wav

4. 播放音频文件

aplay test.wav

网口使用 udhcpc 无法获取 DNS 问题

要事先准备 simple.script(为 udhcpc 所用), 可以放入/mnt/yaffs2, 或者放到其它目录, 因为紫芯的文件系统是 cramfs 只读文件系统, 所以先放在 yaffs2 下做调试

```
# pwd  
/mnt/yaffs2  
# ls  
hello          lost+found      simple.script  test.wav      tmmpl  
#
```

执行 udhcpc -i eth0 -s ./simple.script 就是可以了, 然后 ping 百度没问题

```
# udhcpc -i eth0 -s ./simple.script  
udhcpc (v1.14.4) started  
Setting IP address 0.0.0.0 on eth0  
Sending discover...  
Sending select for 192.168.14.40...  
Lease of 192.168.14.40 obtained, lease time 7200  
Setting IP address 192.168.14.40 on eth0  
Deleting routers  
route: SIOCDELRT: No such process  
Adding router 192.168.14.1  
Recreating /etc/resolv.conf  
  Adding DNS server 221.5.203.98  
# ping www.baidu.com  
PING www.baidu.com (61.135.169.121): 56 data bytes  
64 bytes from 61.135.169.121: seq=0 ttl=54 time=50.000 ms  
64 bytes from 61.135.169.121: seq=1 ttl=54 time=40.000 ms  
^C  
--- www.baidu.com ping statistics ---  
2 packets transmitted, 2 packets received, 0% packet loss  
round-trip min/avg/max = 40.000/45.000/50.000 ms  
^
```

网口获取 dhcp 一定要点斜杠 simple.script

内核编译后生成 kernel 烧写文件遇到的问题

```
root@ubuntu:/home/xiang/ASM9260T/ASM9260T-linux/linux-2.6.32.27_9260kernel# ./mkrawImage  
./mkrawImage: line 1: /usr/local/arm/arm-2008q3-linux/bin/arm-none-linux-gnueabi-objcopy: No such file or directory  
Image Name:  Linux kernel  
Created:  Mon Jan  8 01:44:24 2018  
Image Type:  ARM Linux Kernel Image (uncompressed)  
Data Size:  4542912 Bytes = 4436.44 kB = 4.33 MB  
Load Address: 0x20008000  
Entry Point: 0x20008000
```

找不到交叉编译路径的目录

```
JX-2.6.32.27_9260kernel# ls  
lib      Makefile      mkuImage      Mod  
linux.bin  mkimage      mm           net  
MAINTAINERS  mkrawImage  modules.order  raw
```

这是因为官方的 mkrawImage 这个执行文件里面路径不是我们指定的交叉编译器路径, 所以我们要修改 mkrawImage 里面的路径代码。打开文件看就知道怎么改了

```
/opt/ASM9260T_CROSS/arm-2008q3-linux/bin/arm-none-linux-gnueabi-objcopy -O binary -R .note.
```

我修改成了我自己交叉编译器的路径

Wifi RTL8188eus 编译，安装

 `asm9260linux rtl8188driver_transplant.doc` 移植方法按照这个文档来。

 `rtl8188EUS_linux_v4.3.0.9_15178.20150907.tar.gz`

要编译的代码在这个压缩包里面

按照文档要求编译

然后 `insmod` 就可以成功加载 wifi 了

```
# ifconfig -a
```

```
wlan0      Link encap:Ethernet HWaddr EC:3D:FD:E6:00:9D  
          BROADCAST MULTICAST MTU:1500 Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

如果在 `insmod` 的时候发现

```
8188eu: Unknown symbol register_netdevice  
insmod: can't insert '8188eu.ko': unknown symbol in module, or unknown parameter
```

是因为 8188 驱动程序编译的时候 linux 内核路径没有指定对。

然后要修改你存放 wifi 用户名和密码文件的 `ctrl_interface`

```
1 ctrl_interface=/var/run/wpa_supplicant  
2 ctrl_interface_group=0  
3 update_config=1  
4  
5 network={  
6     ssid="TSARIMAKERSPACE"  
7     psk="20160928"  
8 }
```

因为 ASM9260T 操作 wifi RTL8188EUS 文档下要求

使 `wpa_supplicant` 目录为临时可写，后续要用。

```
mount -t tmpfs tmfps /wpa_net/run/wpa_supplicant
```

要在根目录下创建的 `wpa_net` 零时文件目录

这样的话我们 `wpa_supplicant` 操作 `wpa.conf` 文件里面 `ctrl_interface` 接口路径也要改成一样

```

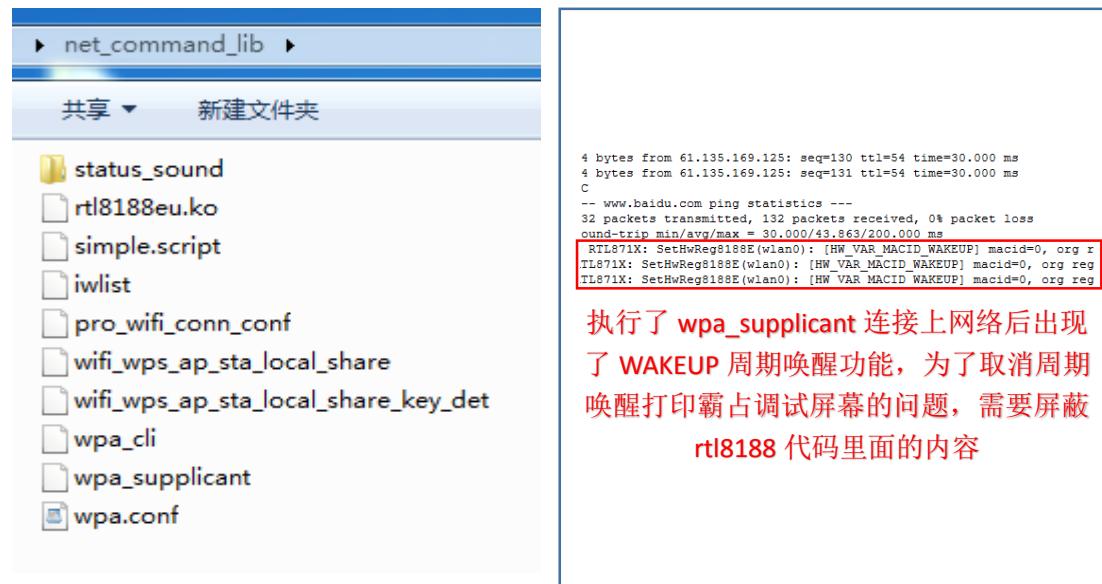
1 ctrl_interface=/wpa_net/run/wpa_supplicant
2 ctrl_interface_group=0
3 update_config=1
4
5 network={
6     ssid="TSARIMAKERSPACE"
7     psk="20160928"
8 }

```

8. 在根文件系统中建立一个目录/wpa_net/run/wpa_supplicant
9. 事先准备好 wpa_cli, wpa_supplicant, tkip_wpa.conf (存 wifi 用户名及密码), 可以放入/mnt/yaffs2, 或其他目录。
10. 事先准备好 simple.script (为 udhcpc 作用), 可以放入/mnt/yaffs2 或其他目录。
11. insmod 8192cu.ko 加载驱动, 加载完以后可以看见 wlan0。
12. ifconfig wlan0 up 启动 wlan0。
13. 使 wpa_supplicant 目录为临时可写, 后续要用。 mount -t tmpfs tmfps /wpa_net/run/wpa_supplicant
14. 加入 wifi 网络, 事先需要配置好 tkip_wpa.conf, 里面要有用户名, 密码等。 ./wpa_supplicant -Dwext -iwlan0 -c /mnt/yaffs2/ tkip_wpa.conf -d -B
15. 加载 dhcp, 为开发板自动分配一个地址。(在某些系统中 simple.script 脚本前面注意加 ./)。 udhcpc -i wlan0 -s ./simple.script
16. 至此分配完成, 加入了网络, 可以 ping 了。

这个命令一定要执行

Asm9260t linux wifi 工具在 net_command_lib 目录下



```
rtl8188EUS_linux_v4.3.0.9_15178.20150907/hal/rtl8188e# vim rtl8188e_hal_init.c
```

去 RTL8188EUS 的目录下，找到 rtl8188e_hal_init.c 文件

```
/*          val32 = rtw_read32(adapter, reg_macid_sleep);  
          DBG_8192C(FUNC_ADPT_FMT ": [HW_VAR_MACID_SLEEP] macid=%d, org reg_0x%03x=0x%08X\n",  
          FUNC_ADPT_ARG(adapter), id, reg_macid_sleep, val32);*/  
  
/*          val32 = rtw_read32(adapter, reg_macid_sleep);  
          DBG_8192C(FUNC_ADPT_FMT ": [HW_VAR_MACID_WAKEUP] macid=%d, org reg_0x%03x=0x%08X\n",  
          FUNC_ADPT_ARG(adapter), id, reg_macid_sleep, val32);*/
```

这样就屏蔽了打印，但是周期唤醒功能还是存在，先用用看这个功能长时间用有没有问题

ASM9260T USB 烧写程序注意事项



在目录找到这个压缩包解压，然后按照里面的文档安装。然后烧写程序安装官方文档操作

交叉编译器里面库文件和头文件路径问题

```
root@ubuntu:/opt/ASM9260T_CROSS/arm-2008q3-linux# ls  
arm-none-linux-gnueabi bin lib libexec share
```

所有自己编译的 include 头文件和 lib 库文件都放在这个目录 libc 里面

```
root@ubuntu:/opt/ASM9260T_CROSS/arm-2008q3-linux# ls  
bin include lib libc
```

常用的头文件库文件都放在这个里面

```
root@ubuntu:/opt/ASM9260T_CROSS/arm-2008q3-linux/arm-none-linux-gnueabi/libc/usr# ls  
bin include info lib man sbin share
```

```
root@ubuntu:/opt/ASM9260T_CROSS/arm-2008q3-linux/arm-none-linux-gnueabi/libc/usr#
```

这是头文件的位置，在 libc/usr/include 目录下

```
/opt/ASM9260T_CROSS/arm-2008q3-linux/arm-none-linux-gnueabi/libc# ls  
lib sbin thumb2 usr
```

```
/opt/ASM9260T_CROSS/arm-2008q3-linux/arm-none-linux-gnueabi/libc#
```

库文件放在 libc/ 目录下的 lib 里面

ASM9260T-GPIO 使用方法

Pincontrol.h 文件下有 GPIO 操作函数

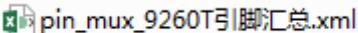
```
void set_pin_mux(int port,int pin,int mux_type);
void set_GPIO_pull_up(int port,int pin);
void set_GPIO_pull_down(int port,int pin);
void asm9260_gpio_init(void);
int get_pin_mux_val(int port,int pin);
void set_GPIO(int port,int pin);
void clear_GPIO(int port,int pin);
void write_GPIO(int port,int pin,int value);
int read_GPIO(int port,int pin);
```

void set_pin_mux(int port,int pin,int mux_type);

port:是指那组 GPIO

pin:是指这组 GPIO 中的哪一个 GPIO 引脚

mux_type: GPIO 的管脚模式

GPIO 管脚模式需要查表


GPIO	7	6	5	4	3	2	1	0
Pad_Name	PIO0_7	PIO0_6	PIO0_5	PIO0_4(pd)	PIO0_3(pu)	PIO0_2(pu)	PIO0_1(pu)	PIO0_0(pu)
PinFunction0 MUXSEL=0 0 0	UART2_RXD(1)	UART2_TXD	UART2_CLK	UART1_CTS(1)	UART1_RTS(0)	UART1_RXD(1)	UART1_TXD	UART1_CLK
PinFunction1 MUXSEL=0 0 1	I2S0_TX2(o)	I2S0_RX2(o)	I2S0_TX1(o)	I2S0_RX0(o)	I2S0_RX0(1)	I2S0_LRC(o)	I2S0_BCLK(o)	I2S0_MCLK
PinFunction2 MUXSEL=0 1 0	SPI0_MOSI*	SPI0_MISO*	SPI0_SEL*	SPI0_SCK*	SPI1_MOSI*	SPI1_MISO*	SPI1_SEL*	SPI1_SCK*
PinFunction3 MUXSEL=0 1 1	MII_PPS_OUT(o)				JTAG_RST(1)	JTAG_TMS(1)	JTAG_TDO(o)	JTAG_TDI(1)
PinFunction4 MUXSEL=1 0 0	I2C1_SDA	I2C1_SCL	I2C0_SDA	I2C0_SCL	I2C_start(1)	test_node_0(1)	test_reset_n(1)	test_clk(1)
PinFunction5 MUXSEL=1 0 1	chain2_e1	chain2_e1	chain1_e1	chain0_e1				
PinFunction6 MUXSEL=1 1 0								
PinFunction7 MUXSEL=1 1 1								

将里面的三位转换成 16 进制，填入 set pin mux 的 mux type，就可以设置

write_GPIO(int port,int pin,int value); 该函数是写 GPIO

port:是指那组 GPIO

pin:是指这组 GPIO 中的哪一个 GPIO 引脚

value: 写 1 高电平，写 0 底电平。

```
int i = 0;
printf("init asm9260t gpio module_driver\n");
set_pin_mux(17, 4, 0);//使用某个GPIO要先设置管脚
for(i=0;i<5;i++)
{
    write_GPIO(17,4,0);//设置管脚输出电平
    mdelay(1000);
    write_GPIO(17,4,1);
    mdelay(1000);
}
return 0;
```

记住 GPIO 输出，要先设置 GPIO 管脚然后输出电平

GPIO 作为输入的方法

```
void set_pin_dir(int port, int pin, int bOut)
```

port:是指那组 GPIO

pin:是指这组 GPIO 中的哪一个 GPIO 引脚

bOut 设置 0 为输入，设置 1 为输出，上下拉没法选择，I2C 设备建议外部连接上来电阻

这里有个问题，`set_pin_dir` 函数在 C 文件定义了，但是紫芯官方没在头文件定义，所以要自己将函数加进 `pincontrol.h`

```
linux-2.6.32.27_9260kernel/arch/arm/mach-asm92xx/include/mach# ls
  gpio_spi.h i2c.h irqs.h lradc_keypad.h memory.h rtc.h system.h uncompress.h
  hardware.h io.h keypad.h mac.h [pincontrol.h] serial.h timex.h vmalloc.h
  hwecc.h irq.h lradc.h mci.h pwm.h spi.h uart_reg.h xpt2046_ts.h
linux-2.6.32.27_9260kernel/arch/arm/mach-asm92xx/include/mach#
```

```
void clear_GPIO(int port,int pin);
void write_GPIO(int port,int pin,int value);
int read_GPIO(int port,int pin);
void set_pin_dir(int port, int pin, int bOut);
```

加上 `set_pin_dir` 函数，这个 `dir` 函数貌似用不到

```
set_pin_mux(0,1,0);
for(i=0;i<5;i++)
{
    while(1){
        z = read_GPIO(0,1);
        printk("gpio input data = %d\n",z);
        mdelay(500);
    }
}
```

这样就可以读单个 GPIO 了，每个 GPIO 低电平返回 0，高电平返回 1，这是固定的
我使用 IMX6 的时候每个 GPIO 低电平返回 0，高电平返回 GPIO 编号，ASM9260T 就不会这么装 B，固定模式高电平返回 0，低电平返回 1

GPIO 作为中断的使用方法

ASM9260T 芯片 GPIO 作为中断引脚有几个地方要注意

比如我用 做外部中断 IO

第 1 步，申请 GPIO 为外部输入中断功能

void io_irq_enable_edge(int port,int pin,int type) //GPIO 中断功能申请函数

port:是指那组 GPIO

pin:是指这组 GPIO 中的哪一个 GPIO 引脚

type: 是指 IO 口触发那种中断

```
#define GPIO_IRQ_LEVEL_LOW          0  
#define GPIO_IRQ_LEVEL_HIGH         1  
#define GPIO_IRQ_EDGE_FALLING      0  
#define GPIO_IRQ_EDGE_RISING        1
```

```
static int __init button_init(void){  
    int irqres;  
    printk("enter button \n");  
    io_irq_enable_edge(0, 4, GPIO_IRQ_EDGE_FALLING);  
    irqres = request_irq((INT_GPIO0+0/4),buttons_irq,IRQF_TRIGGER_FALLING,"key",NULL);  
    return 0;  
}
```

第 2 步，申请中断函数

request_irq((INT_GPIO0+portgroup/4),.....)

关键是这个移位算法 (INT_GPIO0+ portgroup /4)

表 6-2 中断源

中断源	SRC	向量	说 明
ARM_COMMRX	0	0x0000	调试通信通道接收中断
ARM_COMMTX	1	0x0004	调试通信通道发送中断
RTC_IRQ	2	0x0008	RTC 中断
GPIO0_IRQ	3	0x000C	GPIO0 中断
GPIO1_IRQ	4	0x0010	GPIO1 中断
GPIO2_IRQ	5	0x0014	GPIO2 中断
GPIO3_IRQ	6	0x0018	GPIO3 中断
GPIO4_IRQ/I2S1_IRQ	7	0x001C	GPIO4 中断/I2S1 中断

因为 ASM9260T 中断源没有像 IMX6，三星平台那种 IO 口静态映射表，所以要自己写数字 (INT_GPIO0+ portgroup /4)

Portgroup: 表示你外部中断的引脚属于 GPIO 哪一组

因为 1 个引脚是分到一个 GPIO 组里面，比如说 GP0_4，就是 GPIO0 组第 4 个引脚。GP1_4 就是第 1 组 4 号引脚，GPIO0_IRQ 就是包含 GPIO0~GPIO3 组的 32 个引脚共享中断号。

所以 portgroup /4 不管你是用 GPIO0,GPIO1,GPIO2,GPIO3 组的任何一个引脚，都必须分配到 GPIO0_IRQ，也就是 3 号中断源，如果是 GPIO4 组就分配到 GPIO1_IRQ 也就是 4 号中断源

```
request_irq((INT_GPIO0+portgroup/4),.....)
```

(INT_GPIO0+portgroup/4) 这个公式你只需要根据你使用的引脚编号对应上 GPIO 组，然后把 GPIO 组的编号写给 portgroup 变量就行了。其余的你不需要修改，/4 除以 4 也不需要修改。

第 3 步：在中断函数里面执行判断 IO 口是否发生中断，和清除中断标志位

```
int get_io_irq_status(int port,int pin) //gpio 引脚是否发生中断再次确认函数
```

port:是指那组 GPIO

pin:是指这组 GPIO 中的哪一个 GPIO 引脚

返回值为 1 表示，GPIO 确实发生了外部中断，返回值为 0 表示没有发生外部中断

```
void io_irq_clr(int port,int pin) //中断函数执行完前一定要执行这句清除中断标志位，否则中断会不停的执行。
```

port:是指那组 GPIO

pin:是指这组 GPIO 中的哪一个 GPIO 引脚

```
9 static int irq_status = 0;
10
11 static irqreturn_t buttons_irq(int irq,void* dev_id){
12     printk("enter buttons irq process\n");
13     if(get_io_irq_status(0,4)!=0) //再次判断指定引脚中断是否确实发生
14     {
15         irq_status = get_io_irq_status(0,4); //返回引脚发生中断的状态，其实可以不要这句，正式程序不写这句
16         printk("irq_status = %d\n",irq_status);
17
18     }
19     io_irq_clr(0,4); //清除中断标志位
20     return IRQ_RETVAL(IRQ_HANDLED);
21 }
22
23
```

我在 IMX6 上就没有执行清除中断标志位，也没有什么问题，也许是官方 SDK 自动帮你清除了。这个还是根据实际使用平台来确定。

```
3
4 static void __exit button_exit(void){
5
6     printk("exit button \n");
7     io_irq_disable(0, 4); //清除引脚中断功能
8     free_irq((INT_GPIO0+0/4),NULL); //清除绑定的中断源
9 }
```

记得不使用要清除中断功能

```
38
39 static int irq_status = 0;
40
41 static irqreturn_t buttons_irq(int irq,void* dev_id){
42     printk("enter buttons irq process\n");
43     if(get_io_irq_status(0,4)!=0) //再次判断指定引脚中断是否确实发生
44     {
45         irq_status = get_io_irq_status(0,4); //返回引脚发生中断的状态，其实可以不要这句，正式程序不写这句
46         printk("irq_status = %d\n",irq_status);
47     }
48     io_irq_clr(0,4); //清除中断标志位
49     return IRQ_RETVAL(IRQ_HANDLED);
50 }
51
52
53
54 static int __init button_init(void){
55     int irqres;
56     printk("enter button \n");
57     io_irq_enable_edge(0, 4, GPIO_IRQ_EDGE_FALLING);
58     irqres = request_irq((INT_GPIO0+0/4),buttons_irq,IRQF_TRIGGER_FALLING,"key",NULL);
59     return 0;
60 }
61
62
63 static void __exit button_exit(void){
64     printk("exit button \n");
65     io_irq_disable(0, 4); //清除引脚中断功能
66     free_irq((INT_GPIO0+0/4),NULL); //清除绑定的中断源
67
68
69 }
70
```

这是 ASM9260T 整个中断申请，释放，使用流程

ASM9260T 使用 RTL8188EUS，启动 AP 功能方法

因为 ASM9260T 是用 linux2.6.32 版本的内核，而且 ASM9260T 的文件系统是 Cramfs(只读文件系统)，所以有一些注意事项

因为供应商很多使用的 RTL8188 驱动是 linux3.x 以上的，所以我找到紫芯官方给的 RTL8188EUS_STA_AP 的驱动

 RTL8188EUS_STA_AP.tar.bz2

解压压缩包

```
root@ubuntu:/home  
RTL8188EUS_OLD  
root@ubuntu:/home
```

得到 RTL8188EUS_OLD 文件目录

```
root@ubuntu:/home/xiang/ASM9260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD# ls  
driver LICENSE README.md wpa_supplicant_hostapd
```

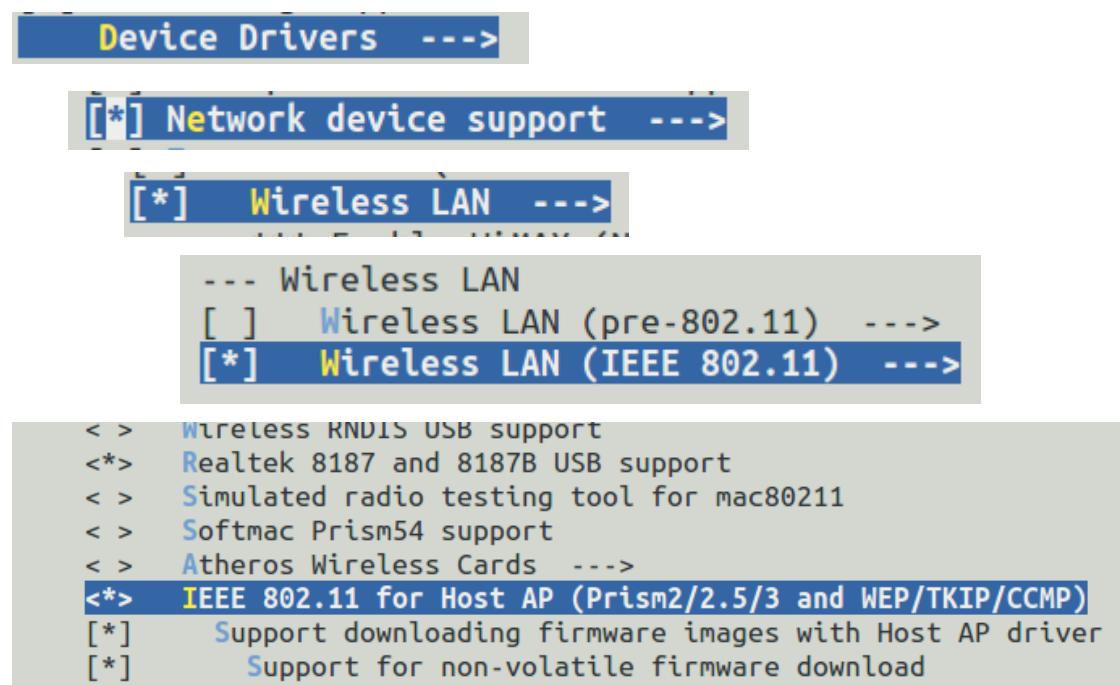
```
root@ubuntu:/home/xiana/ASM9260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD#
```

一定要看 README,要结合紫芯的  asm9260t linux下rtl8188驱动移植.doc world 文档看

```
root@ubuntu:/home  
8188eu.ko 8188  
8188eu.mod.c aut
```

编译出来 8188eu.ko 的驱动

确认内核配置



The screenshot shows the Linux kernel configuration interface. The path selected is:

- Device Drivers --->
- [*] Network device support --->
- [*] Wireless LAN --->
- Wireless LAN
- [] Wireless LAN (pre-802.11) --->
- [*] **Wireless LAN (IEEE 802.11) --->**

Under the IEEE 802.11 for Host AP section, the following options are listed:

- < > Wireless RNDIS USB support
- <*> Realtek 8187 and 8187B USB support
- < > Simulated radio testing tool for mac80211
- < > Softmac Prism54 support
- < > Atheros Wireless Cards --->
- <*> IEEE 802.11 for Host AP (Prism2/2.5/3 and WEP/TKIP/CCMP)**
- [*] Support downloading firmware images with Host AP driver
- [*] Support for non-volatile firmware download

一定要把内核配置的 AP 模式选择上

编译 hostapd

然后 hostapd 一定要使用紫芯官方给的源码包来编译，不要用网上的 hostapd 源码，因为紫芯官方和 RTL8188 厂商做了手脚。hostapd 源码包也在 RTL8188 驱动包下

```
root@ubuntu:/home/xiang/ASM9260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD# ls  
driver LICENSE README.md wpa_supplicant_hostapd
```

```
root@ubuntu:/home/xiana/ASM9260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD#
```

Wpa_supplicant_hostapd 目录里面就是

```

root@ubuntu:/home/xiang/ASMB260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD# cd wpa_supplicant_hostapd/
root@ubuntu:/home/xiang/ASMB260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD/wpa_supplicant_hostapd# ls
hostapd-0.8  p2p_hostapd.conf  rtl_hostapd_2G.conf  rtl_hostapd_5G.conf
root@ubuntu:/home/xiang/ASMB260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD/wpa_supplicant_hostapd#

```

这个就是 hostapd 软件源码包

这个是官方给的配置文件例子，直接复制粘贴到开发板先使用，然后再修改

```

root@ubuntu:/home/xiang/ASMB260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD/wpa_supplicant_hostapd/hostapd-0.8/hostapd# ls
Android.mk  config_file.o  dump_state.c  eap_register.h  hostapd.8  hostapd_cli.o  hostapd.vl
changeLog   ctrl_iface.c  dump_state.d  eap_register.o  hostapd.accept  hostapd.conf  hostapd.wpa
compile.sh  ctrl_iface.d  dump_state.h  eap_testing.txt  hostapd_cli  hostapd.deny  logwatch
config_file.c  ctrl_iface.h  dump_state.o  hlr_auc_gw.c  hostapd_cli.1  hostapd.eap_user  main.c
config_file.d  ctrl_iface.o  eap_register.c  hlr_auc_gw.milenage_db  hostapd_cli.c  hostapd.radius_clients  main.d
config_file.h  defconfig  eap_register.d  hostapd  hostapd_cli.d  hostapd.sin_db  main.o
root@ubuntu:/home/xiang/ASMB260T/wifi rtl8188eus/RTL8188EUS/RTL8188EUS_OLD/wpa_supplicant_hostapd/hostapd-0.8/hostapd#

```

按照我写的 linuxUSB_wifi 文档的方法吧 hostapd 软件编译出来

然后将 rtl_hostapd_2G.conf 和 hostapd, hostapd_cli 拷贝到开发板上。

7、启动 wifi 脚本

```

insmod 8188eu.ko

ifconfig wlan0 192.168.1.1

mkdir /var/run

hostapd -d /etc/rtl_hostapd_2G.conf &

```

按照 README 文档来启动 RTL8188 的 AP 模式

insmod 8188eu.ko 加载 rtl8188eus 驱动

```

RTL871X: rtw_wx_get_rts, rts_thresh=2347
RTL871X: rtw_wx_get_frag, frag_len=2346
wlan0    unassociated  Nickname:<WIFI@REALTEK>
          Mode:Auto Frequency=2.412 GHz Access Point: Not-Associated
          Sensitivity:0/0
          Retry:off RTS thr:off Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality:0 Signal level:0 Noise level:0
          Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
          Tx excessive retries:0 Invalid misc:0 Missed beacon:0
#

```

查看驱动是否添加成功

ifconfig wlan0 192.168.1.1

给 wifi 网卡配置 IP 地址，然后会自动启动

注意：这个给网卡分配的 IP 地址一定要根据 udhcpd 的 dhcpd.conf 文件来配置

```

1 start 192.168.2.2
2 end 192.168.2.254
3 interface wlan0
4 opt dns 192.168.2.1
5 option subnet 255.255.255.0
6 opt router 192.168.2.1
7 option domain local
8 option lease 864000

```

我的 dhcpd.conf 配置的地址范围是 192.168.2.2~192.168.2.254，所以我们设置网卡 IP 地址为 192.168.2.2 本身或者以上的 IP 段都可以

```

ADDRCONF(NETDEV_UP): wlan0: link is not ready      这是启动完成
# ifconfig
wlan0      Link encap:Ethernet  HWaddr EC:3D:FD:E6:00:9D
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

#
查看 IP 配置成功
# hostapd -d rtl_hostapd_2G.conf -B      执行 hostapd 服务, 这样手机才能看到热点

```

```

mkdir[ctrl_interface]: Read-only file system
Failed to setup control interface
wlan0: Unable to setup interface.

```

这点很关键

hostapd 启动失败, 这是因为我们的 AS9260T 平台的文件系统是只读系统

```

##### hostapd configuration file ####

interface=wlan0
ctrl_interface=/var/run/hostapd
ssid=rtwapp
channel=6
wpa=2
wpa_passphrase=87654321

```

这个 interface 是使用 hostapd_cli 查看有多少手机连接到我开发板网卡, 使用的接口

我们的 rtl_hostapd_2G.conf 文件里面 ctrl_interface 指定的目录正好在 Cramfs 文件系统的只读路径下, 不是在 yaffs2 可读写的目录。

mount -t tmpfs tmfps /var/run 我们用 mount -t tmpfs 命令将该路径下文件变成可写

```

ELOOP: remaining socket: sock=6 eloop_data=
# mount -t tmpfs tmfps /var/run/
#

```

```

# hostapd -d rtl_hostapd_2G.conf -B      再次启动 hostapd 软件

```

```

WPA: group state machine enterin
rtl871x_set_key_ops
rtl871x_set_beacon_ops
wlan0: Setup of interface done. hostapd 启动成功

```

344 0	0 SW	[as9260_std_spi.]
429 0	0 SW	[yaffs-bg-1]
430 0	3052 S	-/bin/sh
439 0	0 SW	[RTW_CMD_THREAD]
444 0	1888 S	hostapd -d rtl_hostapd_2G.conf -B

进程中能看到 hostapd 程序

手机可以看到 AP 热点了，我用手机连接

```
445 0      3052 R    ps
# RTL871X: +OnAuth
RTL871X: auth alg=0, seq=1
RTL871X: going to alloc stainfo for sa=a4:71:74:8f:6c:7d
RTL871X: issue_auth
RTL871X: OnAssocReq
RTL871X: IEEE 802.11 element parse ignored unknown element (id=127 elen=8)
RTL871X: unknown vendor specific information element ignored (vendor OUI ac:85:3d len=4)
RTL871X: unknown vendor specific information element ignored (vendor OUI ac:85:3d len=8)
RTL871X: allocate new AID = (1)
RTL871X: update_bcn_fixed_ie
RTL871X: HT: STA a4:71:74:8f:6c:7d HT Capabilities Info: 0x116e
RTL871X: bss_cap_update_on_sta_join STA a4:71:74:8f:6c:7d - no greenfield, num of non-gf
RTL871X: bss_cap_update_on_sta_join, updated=1
RTL871X: update_sta_info_apmode
RTL871X: ### Set STA_(2) info
RTL871X: issue_asocrsp
```

出现了这个问题，最后手机显示连接失败。

看来我们要打开 udhcpd 来启动 DNS 自动分配 IP 功能

在文件系统/etc 目录下创建 udhcpd.conf

```
1 start 192.168.2.2
2 end 192.168.2.254
3 interface wlan0
4 opt dns 192.168.2.1
5 option subnet 255.255.255.0
6 opt router 192.168.2.1
7 option domain local
8 option lease 864000
```

保存 udhcpd.conf 文件

然后将 udhcpd.conf 文件拷贝到开发板/etc 目录下

```
# udhcpd -f dhcpd.conf
udhcpd (v1.14.4) started
udhcpd: can't open '/var/lib/misc/udhcpd.leases': No such file or directory
```

在启动 udhcpd 的时候可能找不到 leases 文件，所以我们要在 /var/lib/misc 目录下创建 leases 文件。

因为 ASM9260T 使用的是 Cramfs 只读文件系统，我们要将该路径下的目录设置成可写权限
mount -t tmpfs tmfps /var/lib/misc

```
# mount -t tmpfs tmfps /var/lib/misc/
#
```

这样 mount 之后，本身在 /var/lib/misc 目录下的 leases 文件就不见了，所以我建议不要在文件系统烧写前将 lease 文件放在 /var/lib/misc 目录下，我们放在 yaffs2 目录下，然后在板子启动脚本执行的时候执行 cp 命令，将 leases 文件拷贝到 /var/lib/misc 目录下

```
# udhcpd -f dhcpd.conf & 启动 udhcpd 自动分配 IP 服务
```

```
Sending OFFER of 192.168.2.2
Sending OFFER of 192.168.2.2
Sending ACK to 192.168.2.2
Sending OFFER of 192.168.2.3
Sending OFFER of 192.168.2.3
Sending ACK to 192.168.2.3
```

地址分配成功，其实应该没什么问题了

```
# ping 192.168.2.3
PING 192.168.2.3 (192.168.2.3): 56 data bytes
64 bytes from 192.168.2.3: seq=0 ttl=64 time=120.000 ms
64 bytes from 192.168.2.3: seq=1 ttl=64 time=20.000 ms
64 bytes from 192.168.2.3: seq=2 ttl=64 time=30.000 ms
64 bytes from 192.168.2.3: seq=3 ttl=64 time=50.000 ms
```

Ping 下手机 IP 地址，数据包收发成功

```
# hostapd_cli all_sta
Selected interface 'wlan0'
a4:71:74:8f:6c:7d
dot11RSNACurrentState=STAAddress=a4:71:74:8f:6c:7d
dot11RSNACurrentVersion=1
dot11RSNACurrentSelectedPairwiseCipher=00-0f-ac-4
dot11RSNACurrentTKIPLocalMICFailures=0
dot11RSNACurrentTKIPRemoteMICFailures=0
hostapdWPAPTKState=11
hostapdWPAPTKGroupState=0
```

为了保险起见我们用
hostapd_cli 软件查看
连接上我开发板的手
机网卡地址，没问题

ASM9260T 驱动网卡芯片 KSZ8041RNL 网线接口检测问题

我们用 ifconfig 或者 cat 来查看网线连接上主板网口没有

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:E0:A3:A4:98
          inet addr:192.168.14.40  Bcast:192.168.14.
          inet6 addr: fe80::2e0:a3ff:fea4:9867/64 Sc
          UP BROADCAST RUNNING MULTICAST  MTU:1500
```

不管网线连没连接上主板，这里显示的都是 RUNNING

```
# cat /sys/class/net/eth0/carrier
1
```

不管网线连没连接上主板，这里显示的都是 1

这是因为 ASM9260T 网卡 KSZ8041RNL 驱动的问题

根据官方发过来的资料进行修改内核

 ASM9260T-KSZ8041RNL-netRJ45chenk 在这个目录里面找到 c 和 h 文件

```
asmmac_ethtool.c
asmmac_main.c
asmmac.h
```

该文件用于 linux-2.6.32 版本

```
linux-2.6.32.27_9260kernel/drivers/net/asm92xx_mac#
```

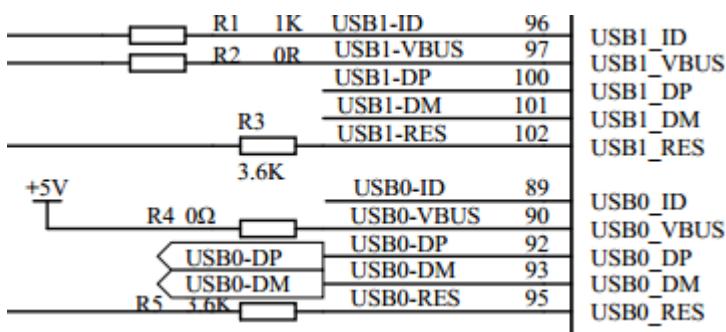
将这三个文件用来覆盖以前的内核版本，覆盖位置在内核目录下 /drivers/net/asm92xx_nac
然后 make

然后烧写内核进开发板就可以了

内核报错 arch_interrupt 85 : VBUS error workaround

```
can: controller area network core (rev 20090105 and 0)
NET: Registered protocol family 29
arch_interrupt 85: VBUS error workaround (delay&reset coming)
can: raw protocol (rev 20090105)
can: broadcast manager protocol (rev 20090105 t)
lib80211: common routines for IEEE802.11 drivers
registered taskstats version 1
asm9260_rtc asm9260_rtc: setting system clock to 1970-01-01 00:00:00 UTC (0)
VFS: Mounted root (cramfs filesystem) readonly on device 31:2.
Freeing init memory: 124K
arch_interrupt 85: VBUS error workaround (delay&reset coming)
```

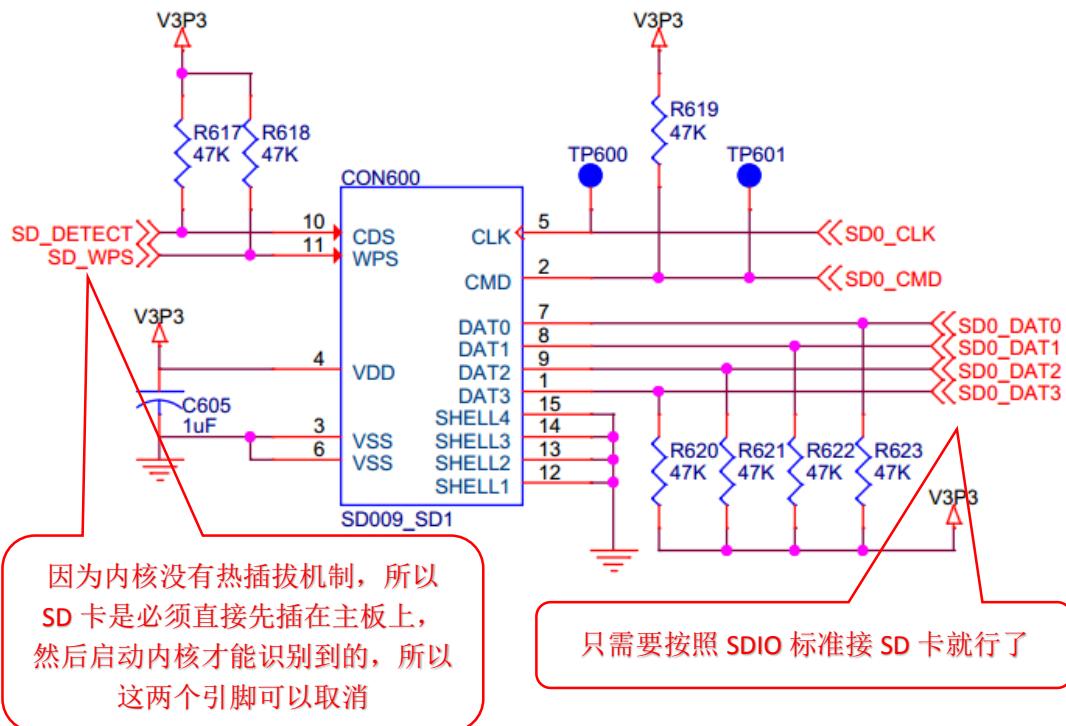
该报错的原因是硬件电路问题



该报错的原因是因为 USB_VBUS 电源供电问题，电脑 USB 给主板供电很多都是 4.7V，但是 ASM9260T 的 USB_VBUS 供电必须在 5V 以上

所以为了保证 USB_VBUS 的供电，我们可以在电源断开给个 DC5V 的电源适配器，也可以用 3.3V 升压到 5V 单独给 USB_VBUS 供电

ASM9260T 挂载 SD 卡问题



```
Freeing init memory: 124K
mmcblk0: mmc0:0001 00000 3.72 GiB
mmcblk0: p1
```

mmcblk0
mmcblk0p1 在/dev 下面会产生 mmc 节点。记住是挂载 mmcblk0p1

```
# mount /dev/mmcblk0p1 /mnt/tmp/
# cd /mnt/tmp/
# ls
DHT11_test.ko          lmosquitto
ID.sh                  lws6
RGB_switch.ko          pkg
System Volume Information
我挂载到 tmp 目录下，成功
```

ASM9260T 不同 linux2.6.32 内核版本的 SD 卡内核启动无法产生/dev

节点问题

```
[*] Networking support -->
    Device Drivers -->
        File systems -->
            [*] SD Support -->
                <*> MMC/SD/SDIO card support -->
                <> Sony MemoryStick card support (EXPERIMENTAL)
                    --- MMC/SD/SDIO card support
                    [ ] MMC debugging
                    [ ] Allow unsafe resume (DANGEROUS)
                        *** MMC/SD/SDIO Card Drivers ***
                    <*> MMC block device driver
                    [*]     Use bounce buffer for simple hosts
                    <*> SDIO UART/GPS class support
                    <> MMC host test driver
                        *** MMC/SD/SDIO Host Controller Drivers ***
                    <> Secure Digital Host Controller Interface support
                    <M> Atmel SD/MMC Driver
                    <> MMC/SD/SDIO over SPI
                    <*> Alpscale ASM9260T MMC/SD Interface support
                    [*]     Alpscale ASM9260T MMC/SD 4 wire mode support
```

保证以上内核配置完整

```
Freeing init memory: 116K
mmcblk0: mmc0:0001 00000 3.72 GiB
mmcblk0: unknown partition table
mount mtd3 on /mnt/yaffs2
```

发现内核启动后没有自动生成 SD 卡节点

```
mmc0: new high speed SDHC card at address 0001
mmcblk0: mmc0:0001 00000 3.72 GiB
mmcblk0: p1
mount mtd3 on /mnt/usbffsd
```

如果内核启动 SD 卡成功

应该产生 mmcblk0: p1 设备节点

```
尝试将 kernel/drivers/mmc/host/as9260mci.c
797行 clkdiv *= 8 注释去掉
1073行 mmc->f_max = 60000000 改为 25000000
```

```
796
797 //      clkdiv *= 8;
798
1072      mmc->f_max = 25000000,
1073      mmc->f_max = 60000000;
1074      mmc->freq_avail = MMC_VDD修改这个速率为 25000000
```

重新编译内核，烧写进开发板，内核启动在/dev 下面产生 mmcblk0: p1 节点，不用 mount 去挂载，内核自动挂载在/mnt/sd 目录下，自己去/mnt/sd 目录下看就是了

SD 卡出现只能读无法写数据的问题

第1步：查看 SD 卡槽的写保护引脚是不是接了什么电平，被写保护了。

第2步，如果 SD 卡槽硬件写保护引脚没有被使能，就看软件。

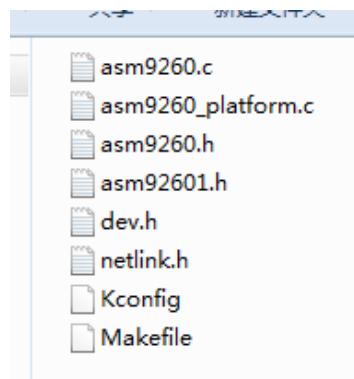
```
# cd /mnt/sd/
# ls
System Volume Information  write
creat
# ./creat
txt creat success
# ls
System Volume Information  creat
biaoding.txt                  write
# ./write
txt open success
txt write success
#
# sync
FAT: Invalid FSINFO signature: 0x00000000, 0x0910700d (sector = 1)
#
```

你看我写完数据之后执行 sync，这样就把缓存的数据写入 SD 卡了。

还有就是有些内核 sync 后不会输出这个字符串，但是确实是存储进 SD 卡了的

socketCAN 总线移植到 linux2.6.32 上的方法

这是官方给我的 linux2.6.32 上的 can 驱动



这就目录里面的文件

将这些文件完全覆盖到内核 kernel/drivers/net/can/asm9260/ 目录下

```
60T/ASM9260T-linux/linux-2.6.32.27_9260kernel/drivers/net/can/asm9260# ls  
0.h  asm9260.o  asm9260_platform.c  asm9260_platform.o  built-in.o  dev.h  Kconfig  Makefile  modules.order  netlink.h
```

覆盖完成后去配置 make menuconfig



cd /usr/sbin 进入 usr 的 bin 目录

执行./ip link set can0 up type can bitrate 500000

```
# ./ip link set can0 up type can bitrate 500000  
asm9260_platform asm9260_platform.0: setting BTR0=0x07 BTR1=0x1b BTR2=0x00
```

配置成功，然后你 ifconfig 就看得到 CAN0 设备节点了

但是移植后发现 can 还是用不了，官方又给了个新的内核，可以用 CAN1 的我把内核放在

这个目录下了 有完整socketcan的linux2.6.32内核

虽然用的是 CAN0 但是一定要将 CAN1 也同时选择上，否则无法启动 CAN0

ASM9260T CAN 引脚映射设置，比如我想其它规定的 GPIO 也有

CAN 功能

```
root@ubuntu:~/home/xiang/ASM9260T/rootfs/_rootfs/roms# /linux-2.6.32.27_9260kernel/arch/arm/mach-asm92xx#
```

asm9260_canserial.c 打开 C 文件

```
static inline void configure_can_pins (unsigned id)
{
    /*TXD & RXD*/
    switch(id)
    {
        case 0:
            /*9260T evk board doesn't use can0 interface*/
            break;
        case 1:
            //set_pin_mux(10, 5, 7); //can1_rx GP10_5
            //set_pin_mux(10, 6, 7); //can1_tx GP10_6
            set_pin_mux(16, 6, 7); //can1_tx GP16_6
            set_pin_mux(16, 7, 7); //can1_rx GP16_7
            break;
    }
}
```

修改这里

电路板启动后文件系统自启动程序流程

Cramfs 文件系统分两个目录下的 etc,

1.根目录下的/etc

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms# bin dev etc home lib linuxrc mnt opt proc sbi
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms#
```

2./mnt 目录下的 etc

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt# cramfs etc ext fat nfs sd test tmp ubifs udisk
```

首先启动根目录下的/etc/init.d/rcS 脚本文件

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/etc/init.d# ls
rcS
```

```
mount -t tmpfs tmpfs /etc
cp -a /mnt/etc/* /etc/
mount -a
mkdir /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s

#get console from cmdline
console=$( cat /proc/cmdline | sed -e 's/.*console=/console /' | sed -e 's/,.*// .' )
echo $console > /etc/confconsole
```

因为/etc 目录下的文件涉及到读写权限问题，所以将/mnt/etc 目录下的一模一样的 rcS 文件拷贝过来执行

所以在根目录下/etc/init.d/rcS 最后几行写自己定义的启动脚本就执行不了。

那我们就要去看/mnt/etc/init.d 目录下的 rcS

```
#!/bin/sh
mount -t tmpfs tmpfs /etc
cp -a /mnt/etc/* /etc/
mount -a
mkdir -p /dev/pts
mount -t devpts devpts /dev/pts
echo $sbin/mdev > /proc/sys/kernel/hotplug
mdev -s

#get console from cmdline
console=$( cat /proc/cmdline | sed -e 's/.*console=/console /' | sed -e 's/.*//')
echo $console > /tmp/console
consoles=$( awk '{ print $2 }' /tmp/console)
In = /dev/scrn0; Out = /dev/ttys0DBG
```

其实/mnt/etc/init.d 目录下的 rcS 和/etc/init.d 目录下的 rcS 是一样的，它们执行完后会自动启动 profile 文件。所以我们关键点是去修改 profile 文件，但是修改的 profile 文件必须是 /mnt/etc 目录下的 profile

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt/etc# ls
fstab hotplug init.d inittab mdev.conf mtab profile
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/mnt/etc#
export ALSA_CONFIG_PATH=/usr/share/a
echo "Done"
echo "xxxxxxxx"
+
Set user path in /etc/profile
Done
xxxxxxxx
+
```

你看执行了自己定义的程序。

如果要执行你自己编写的 hello 程序，不能在 X86 上直接把程序放在/mnt/yaffs2 上面，然后做出文件系统烧录进开发板，因为 yaffs2 目录下的文件在 X86 上无法做进文件系统的。只能先在 X86 根目录下建立一个自己的文件

```
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms# ls
bin Cdatabase dev etc home lib linuxrc mnt opt proc sbin sys tmp usr var wpa_net
root@ubuntu:/home/xiang/ASM9260T/rootfs/_rootfs/roms/Cdatabase# ls
hello
```

将你写的程序 hello 拷贝进自己在根文件系统建立的自定义目录 Cdatabase

然你还能在 profile 文件里面指定开机后去根目录下执行 hello 程序，因为根目录是 cramfs 文件系统，是只读属性。

你只能在 profile 上添加开机后自动拷贝 Cdatabase 目录到 yaffs2 文件系统

```
echo "Done"
echo "xxxxxxxx"
cp -r /Cdatabase /mnt/yaffs2
chmod 755 /mnt/yaffs2/Cdatabase/hello
./mnt/yaffs2/Cdatabase/hello
```

开机自动将 Cdatabase 目录放到可读写的 yaffs2 文件系统，然后给权限，执行 yaffs2 文件系统的应用程序

这样就可以正常实现开机自启动应用程序功能了。

所以 Cramfs 文件系统就是在 X86 把你自己写好的文件放在根目录下，然后交给 shell 脚本程序将自己的应用程序移动到 yaffs2 下执行。如果你要做在线自动更新你就只能用 shell 去判断 yaffs2 是否有以前的应用程序，如果有就停止拷贝根目录的应用程序覆盖 yaffs2 的应用程序，直接执行 yaffs2 上的应用程序去更新系统。

在烧写新板子时一般 yaffs2 没有你的应用程序，所以 shell 判断成功，拷贝应用程序去覆盖 yaffs2 目录

增大 yaffs2 文件系统存储容量，需要修改 uboot

记住容量不能超过 nandflash 的大小，要保留 10M 的余量

Re: 用尔自恋

张宇斌
发给 袁 荣, YongWoo, jimmy.zha@alpscale.cn, kenand.xohn

是不是超容量了？如果是超容量了要改大小只要改Uboot里面的CONFIG_BOOTARGS宏就可以了

NAND分区说明

分区表从u-boot传递到kernel中

```
#define CONFIG_BOOTARGS      "console=ttyS4,115200n8 root=/dev/mtdblock2 init=/linuxrc \"\n      \"mtdparts=NAND:1M@0(loader),5M@1M(kernel),24M@6M(rootfs)ro,50M@30M(yaffs2),-@80M(ubifs)"
```

蓝色部分为V2.1版本中U-BOOT的启动参数，它将NAND分为5个分区：

分区	分区名	分区范围	说明
mtd0	bootloader	0--1MB	该分区放置sramloader和u-boot
mtd1	kernel	1MB--6MB	该分区放置内核
mtd2	rootfs	6MB--30MB	该分区放置根文件系统，格式为Cramfs
mtd3	yaffs2	30MB--80MB	该分区可读写，在挂载根文件系统后挂载为yaffs2
mtd4	ubifs	80MB--	该分区是一个空白的分区，保留

在 uboot 目录下

```
g/ASM9260T/uboot/u-boot-asm9260/include/configs#
```

```
vim asm9260.h
```

```
83 #define CONFIG_BOOTARGS | "console=ttyS4,115200n8 root=/dev/mtdblock2 init=/linuxrc \"\n84 | | | | \"mtdparts=NAND:1M@0(loader),5M@1M(kernel),24M@6M(rootfs)ro,50M@30M(yaffs2),-@80M(ubifs)"\n\n# df -h\nFilesystem           Size   Used Available Use% Mounted on\ntmpfs                 13.8M    0     13.8M  0% /tmp\ntmpfs                 13.8M    0     13.8M  0% /dev\n/dev/mmcblk0p1          3.7G   1.4M     3.7G  0% /mnt/sd\n/dev/mtdblock3          50.0M  1.1M     48.9M  2% /mnt/yaffs2
```

这是现在的版本 yaffs2 只有 50M 容量

修改 CONFIG_BOOTARGS 的参数

```
FIG_BOOTARGS | | "console=ttyS4,115200n8 root=/dev/mtdblock2 init=/linuxrc \"\n| | | | \"mtdparts=NAND:1M@0(loader),5M@1M(kernel),24M@6M(rootfs)ro,80M@30M(yaffs2),-@110M(ubifs)\"\n*_ubifs */
```

修改 yaffs2 为 80M，然后保存退出，确定 Makefile 交叉编译器路径。

```
root@ubuntu:/home/xiang/ASM9260T/uboot/u-boot-asm9260# make distclean\nGenerating include/autoconf.mk\nGenerating include/autoconf.mk.dep\n\nroot@ubuntu:/home/xiang/ASM9260T/uboot/u-boot-asm9260# make asm9260_config\nConfiguring for asm9260 board...\n\n/u-boot-asm9260# make
```

u-boot.bin
u-boot.lds

生成了 u-boot.bin 文件，然后把 u-boot.bin 烧写进开发板

```
# df -h
Filesystem           Size   Used  Available Use% Mounted on
tmpfs                 13.8M    0     13.8M  0% /tmp
tmpfs                 13.8M    0     13.8M  0% /dev
/dev/mmcblk0p1        3.7G   1.4M    3.7G  0% /mnt/sd
/dev/mtblock3         80.0M  1.1M    78.9M  1% /mnt/yaffs2
#
```

yaffs2 扩容到了 80M

擦除清理 yaffs2 目录下的文件

```
# pwd
/mnt/yaffs2
# ls
Cdatabase  lost+found  test
#
```

不论是我怎么下载内核，文件系统，yaffs2 目录下

的多余调试文件始终没有清空。

清空 yaffs2 目录下的文件方法是先清除整个 flash，然后重新烧写 uboot，内核，文件系统

1. 把拨码开关拨码到下载模式

