



BSP

## FAQs

文档版本      11

发布日期      2016-08-10

**版权所有 © 深圳市海思半导体有限公司 2016。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**、海思和其他海思商标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

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

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。

## 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：+86-755-28788858

客户服务传真：+86-755-28357515

客户服务邮箱：[support@hisilicon.com](mailto:support@hisilicon.com)



## 前 言

### 概述

本文档描述使用 BSP 遇到问题的解决方案。

### 产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3531	V100
Hi3535	V100
Hi3536	V100
Hi3516A	V100
Hi3516D	V100
Hi3521A	V100
Hi3520D	V300
Hi3531A	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200
Hi3519	V100
Hi3519	V101
Hi3516C	V300
Hi3559	V100




## 读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

## 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	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	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	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	表示能帮助您解决某个问题或节省您的时间。
 说明	表示是正文的附加信息，是对正文的强调和补充。

## 修订记录

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。

### 文档版本 11 (2016-08-10)

4.14 小节，内容涉及更新。

### 文档版本 10 (2016-07-18)

增加 3.17 小节。

### 文档版本 09 (2016-05-13)

增加 3.16 “I2C 内核态接口原子操作注意事项”。

### 文档版本 08 (2016-02-25)

新增 3.15、5.10 小节。



文档版本 07 (2015-10-10)

新增 3.14 小节。

文档版本 06 (2015-07-16)

产品版本中添加 Hi3521A/Hi3520DV300 和 Hi3531A。

文档版本 05 (2015-06-09)

新增 2.5、4.14 小节。

文档版本 04 (2014-07-07)

**第 3 章 外设类**

新增 3.13 小节

文档版本 03 (2014-02-26)

**第 5 章 Flash**

新增 5.9。

文档版本 02 (2013-12-25)

**第 3 章 外设类**

修改 3.9。

**第 8 章 Kernel**

新增 8.12 和 8.13。

文档版本 01 (2012-09-15)

第 1 次正式发布。



# 目 录

前 言.....	3
1 SDK 环境、使用类.....	5
1.1 为什么执行 server_install 脚本报错? .....	5
1.2 MMZ 和 MMB 分别指什么, 如何配置 MMZ 的区域和大小? .....	5
1.2.1 名词解释 .....	5
1.2.2 MMZ 的原理.....	5
1.2.3 MMZ 驱动模块参数 .....	5
1.3 为什么会有 MMB LEAK 之类的打印? .....	5
1.4 用 fopen 打开大于 4G 的文件失败, 如何操作大于 4G 的文件? .....	5
1.5 为什么 NFS 挂载不上? .....	5
1.6 为什么无法烧写文件系统? 或者 flash 出现非常多的 bad block? .....	5
1.7 为什么无法启动文件系统, 提示 No init found? .....	5
1.8 为什么无法启动文件系统, 提示不能打开 console? .....	5
1.9 为什么 tftp 无法使用? .....	5
1.10 为什么在服务器上执行 SDK 的某些编译命令或者脚本, 报“找不到这个文件”或者类似的错误? .....	5
1.11 制作 cramfs 文件系统的时候, 报出文件大小被截为 16M 的警告, 同时, 制作的文件大小不对。..	5
2 工具类.....	5
2.1 如何在单板上使用 GDB? .....	5
2.2 如何让 gdb 在调试过程中忽略信号量事件? .....	5
2.3 CPU 占用率 100%, 如何优化? .....	5
2.4 如何查看和修改寄存器.....	5
2.5 如何使用 DDR 带宽统计工具 hiddrs.....	5
2.6 如何让 udhcpd 占用更少的内存? .....	5
2.7 为什么有时候 udhcpd 无法获得 IP 地址.....	5
2.8 当前 udhcpd 如何支持 DHCP+ 功能.....	5
3 外设类.....	5
3.1 如何使用 USB 键盘鼠标? .....	5
3.2 为什么 socket 组播会丢包?.....	5
3.3 如何修改以太网 PHY 地址?.....	5



3.4 GMAC 网口 ping 不通？ .....	5
3.5 GMAC 网口状态查询 .....	5
3.6 TOE 怎样使能？ .....	5
3.7 TOE 模式下应用程序使用 socket 接口时的注意事项： .....	5
3.8 应用程序使用 socket 接口时，如何正确工作在非阻塞模式下？ .....	5
3.9 Atheros 8035 型号的 phy 使用注意事项 .....	5
3.10 网线插到到单板上，为什么还是会报 Phy no link？ .....	5
3.11 为什么有时会出现串口无任何响应的情况？ .....	5
3.12 Hi3518 制作 RMII 软件版本 .....	5
3.13 Uboot 下如何调试部分不兼容的 U 盘？ .....	5
3.14 使用 netstat 命令查看链接状态，IP 地址显示为 NULL .....	5
3.15 为什么 USB3.0 口过流保护芯片去掉，会导致某些 USB3.0 U 盘上电启动不识别？ .....	5
3.16 I <sup>2</sup> C 内核态接口原子操作注意事项 .....	5
3.17 为什么 USB2.0 对接 BCM WIFI 模组时，模组加载固件失败？ .....	5
<b>4 PCIe .....</b>	<b>5</b>
4.1 如何配置内核选项，把 PCIe 控制器驱动编进内核？ .....	5
4.2 如何配置 PCIe 的控制时钟？ .....	5
4.3 如何查看 PCIe 设备的 BAR 地址的分配信息？ .....	5
4.4 如何查看 PCIe 地址映射信息？ .....	5
4.5 PCIe MCC 模块的驱动插入后为何不起作用？ .....	5
4.6 PCIe-网卡、PCIe-sata 使用注意事项！ .....	5
4.7 通过 PCIe 实现从启动失败常见的情况 .....	5
4.8 为什么运 SDK 视频预览的业务后，偶尔会打印 “unknown Hi-irq triggered” .....	5
4.9 Hi3531/Hi3532 PCIe BAR 地址，在 reset 时默认会映射到哪个地址？在移动窗口时，有什么需要注意的？ .....	5
4.10 在 PCIe MCC 驱动中，为了支持 Hi3531 和 Hi3531 级联，不再支持主片往从片 DMA 写操作，所有 DMA 操作都由从片发起。原来主片往从片 DMA 写操作，可以通过从片往主片 DMA 读操作替代。 .....	5
4.11 为什么 Hi3531/Hi3531 级联，从片使用发布包 PCIe MCC 中直接编译出来的 ko 会起不来？ .....	5
4.12 Hi3532 作板卡的说明 .....	5
4.13 PCIe MCC 支持主设备复位从设备吗？ .....	5
4.14 使用某些 PCIE 转 SATA 卡（如：marvel9215）连接 sata 盘读写数据，导致出现 vo 低带宽现象的解决方法（以 Hi3536 为例） .....	5
<b>5 Flash .....</b>	<b>5</b>
5.1 为什么 Hi3531 NAND Flash 我配置了硬件 ECC，但却不生效 .....	5
5.2 如何标记 flash 上的坏块？ .....	5
5.3 Hi3531 的 NAND FLASH 控制器支持哪几种纠错方式 .....	5
5.4 使用大容量 NAND 应该注意的地方 .....	5
5.5 NAND ECC/PAGESIZE 逻辑/硬件/软件的配置原理: .....	5
5.6 为什么在 NAND 上, u-boot 保存环境变量后，系统无法启动 .....	5



5.7 为什么烧写 NAND 时, 不能合并成一个文件? .....	5
5.8 为什么 NAND 上文件系统读出来后, 不能再写到 NAND 上使用。 .....	5
5.9 怎样把 SPI flash 由 4 线模式修改为 2 线模式? .....	5
5.10 如何正确使用 mtd-utils 的 nandwrite 裸写工具.....	5
<b>6 文件系统类.....</b>	<b>5</b>
6.1 NAND 挂载 cramfs 文件系统注意事项.....	5
<b>7 快速启动优化.....</b>	<b>5</b>
<b>8 Kernel .....</b>	<b>5</b>
8.1 pid 和 tgid 区别 .....	5
8.2 普通进程和实时进程优先级 .....	5
8.3 如何设置 dmesg buf 的大小? .....	5
8.4 在 /proc/meminfo 显示的 MemTotal 为什么不等于 cmdline 里的配置 mem=xxxM? .....	5
8.5 在 Linux 系统中怎样判断栈是否溢出? .....	5
8.6 遇到“段错误”(segmentation fault)怎样生成 core dump 文件来进行问题分析? .....	5
8.7 Cache 使用注意事项.....	5
8.8 客户跑一个很简单的程序, top 信息的 loadaverage 值比较大, 达到 2.95, 而 cpu 的占用率比较低。 根据之前的理解: .....	5
8.9 当内核的内存配置为 512MB 或以上时, 为什么会出现如下错误“vmap allocation for size 528384 failed: use vmalloc=<size> to increase size.”? .....	5
8.10 内核能否管理两块不连续的内存? .....	5
8.11 在使用 NPTL 的工具链之后, 用 top 怎样查看各个线程的 CPU 占有率? .....	5
8.12 为什么我们的 glibc 工具链中的 backtrace 工具无法打印或保存、定位程序崩溃的堆栈信息? .....	5
8.13 如何实现一个 1ms 的定时器? .....	5





## 插图目录

图 3-1 USB3.0 U 盘启动识别流程 .....	5
图 3-2 内核态接口中申请锁的操作 .....	5
图 3-3 写操作示例图 .....	5
图 4-1 menconfig 界面 .....	5
图 5-1 Nand Flash 块结构图 .....	5



# 1 SDK 环境、使用类

## 1.1 为什么执行 server\_install 脚本报错？

典型的错误提示如下：

```
./server_install
\33[32m
you must use 'root' to execute this shell
\33[39m
./cross.install: 25: Syntax error: "do" unexpected (expecting "fi")
./cross.install: 28: Syntax error: "do" unexpected (expecting "fi")
./cross.install: 30: Syntax error: "do" unexpected (expecting "fi")
```

这是因为 SDK 发布的脚本都是基于 bash 的，而您使用的 linux 服务器可能安装的是 dash 或者其他的命令程序。推荐解决方法：卸载 dash 或者把默认的 sh 改成 bash。一般删除原来的 sh 软链接，重新建立一个指向 bash 的软链接即可：

```
cd /bin
rm -f sh
ln -s /bin/bash /bin/sh
```

## 1.2 MMZ 和 MMB 分别指什么，如何配置 MMZ 的区域和大小？

关于 MMZ 和 MMB 的概念解释如下：

### 1.2.1 名词解释

mmz: Media-Memory-Zone，媒体内存域，也就是分配池

mmmb: Media-Memory-Block，媒体内存块

MMZ 管理的物理内存区域不属于 linux 内核控制，是单独给媒体驱动（如解码器、DEMUX）使用的物理内存区域。MMB 是指从 MMZ 中分配的内存块。



## 1.2.2 MMZ 的原理

MMZ 驱动管理用户创建的分配池，用户程序分配内存的时候可以指定要在哪个分配池中分配内存，分配器将查找满足要求的分配池并从中分配合适的内存块给程序使用。

## 1.2.3 MMZ 驱动模块参数

“mmz=” 用来定义 media-mem 的分配池，格式为：

mmz=<name>,<gfp>,<phys\_start\_addr>,<size>:<name>,<gfp>,<phys\_start\_addr>:.....

- <name>: 字符串，分配池的名字，例如 ddr。
- <gfp>: 数字，表示分配池的属性，主要用于在有多种内存的单板上指定 MMZ 位于哪种内存上（比如 DDR、SDRAM、DDR2、DDR3），为 0 表示自动，目前一般都直接将该值置为 0。
- <phys\_start\_addr>: 分配池的物理起始位置，16 进制数，如 0x86000000；**注意 MMZ 的内存区域不能与 linux 内核的内存区域重叠，MMZ 的物理起始位置就要从“内存起始地址+linux 内核使用的内存大小”开始。**在 Hi353x 平台上，内存的起始地址固定为 0x80000000；举例说明如下：假设单板的 bootargs 为 'mem=96M console=ttyAMA0,115200 root=xxxx'，这表示 linux 内核将使用 96M 的内存空间，那么 MMZ 的起始地址应该配置为 0x80000000+96M = 0x86000000。
- <size>: 分配池的大小，可以使用如下两种表示方式：0x100000、1M。**注意分配池的大小加上 linux 内核的内存大小不能超过物理内存的实际大小。**比如单板上的物理内存是 256M 大小，linux 内核使用了 96M，MMZ 就只能使用最多 256-96=160M。

以上每一个参数都是必需的，参数之间用“,”号分隔，可以指定多个分配池，之间用“:”号分隔。例如：modprobe mmz mmz=ddr,0,0x86000000,64M:vdec,0,0x8A000000,64M。

## 1.3 为什么会有 MMB LEAK 之类的打印？

常见的打印类似这样：

```
MMB LEAK(pid=11093): 0x880BA000, 3686400 bytes, ''
mmz_userdev_release: mmb<0x880BA000> mapped to userspace 0x408d1000 will be
force unmapped!
MMB LEAK(pid=11093): 0x884FD000, 2764800 bytes, ''
mmz_userdev_release: mmb<0x884FD000> mapped to userspace 0x40d14000 will be
force unmapped!
MMB LEAK(pid=11093): 0x8843E000, 520192 bytes, 'decctrl'
mmz_userdev_release: mmb<0x8843E000> mapped to userspace 0x40c55000 will be
force unmapped!
MMB LEAK(pid=11093): 0x884BD000, 262144 bytes, 'Hdec'
mmz_userdev_release: mmb<0x884BD000> mapped to userspace 0x40cd4000 will be
force unmapped!
```



这个打印并不代表内存泄露。这是应用程序在退出时还有资源没有释放干净，SDK 检测到这种情况后强制释放资源时给出的提示信息。请检查应用程序的去初始化动作是否完善（比如创建的通道是不是都销毁了，打开的设备是不是都关闭了）。

## 1.4 用 fopen 打开大于 4G 的文件失败，如何操作大于 4G 的文件？

使用 google 或者 baidu 搜索可以得到多种解决的方法，推荐使用如下方法：

在 makefile 编译选项里加上如下选项：

```
-D_GNU_SOURCE -D_XOPEN_SOURCE=600 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE  
-D_FILE_OFFSET_BITS=64
```

## 1.5 为什么 NFS 挂载不上？

推荐如下的方法挂载 NFS：

```
mount -t nfs -o nolock -o tcp xxx.xxx.xxx.xxx:/xxx/sdk_root /mnt
```

如果有：

```
rpcbind: server localhost not responding, timed out  
RPC: failed to contact local rpcbind server (errno 5).  
rpcbind: server localhost not responding, timed out  
RPC: failed to contact local rpcbind server (errno 5).  
rpcbind: server localhost not responding, timed out  
RPC: failed to contact local rpcbind server (errno 5).
```

之类的打印，一般是 -o nolock 选项没有加；

如果 nfs 经常失去响应(特别是读写大文件的时候，伴随 nfs server not responding, still trying 之类的打印)，一般是没有加 -o tcp 选项。

## 1.6 为什么无法烧写文件系统？或者 flash 出现非常多的 bad block？

常见错误可能如下：

```
hisilicon # nand write.yaffs 0x82000000 0x700000 0x1928ac0  
NAND write: device 0 offset 0x700000, size 0x1928ac0  
Attempt to write non page aligned data, length 26380992 4096 128  
26380992 bytes written: ERROR
```



这是由于制作 yaffs 文件系统的时候指定的 pagesize 和 ecc 参数与单板上的 nand flash 的实际物理参数不匹配导致的，请确认单板上的 nand flash 的 pagesize 和 ecc 参数后重新制作一个文件系统。注意 pagesize 和 ecc 参数错误并不一定导致烧写错误，可能也能烧进去，但是还是会启动不起来，并且打印 bad block n 之类，碰到此类错误，也请重新制作文件系统，并且用

“nand scrub NandFlash 地址 长度”命令把 yaffs 文件系统所在的区域清理一遍才能再次烧写 yaffs 文件系统，比如“nand scrub 400000 1000000”表示从 0x400000 开始清理 64M。如果最后一个参数不传，则表示从此地址开始清理至 nand flash 结束，比如“nand scrub 400000”表示清理从 0x400000 开始的所有 flash 空间。

那么怎样才能知道 nand flash 的 pagesize 和 ecc 类型呢？就看内核启动时候的打印，内核启动的时候会有许多打印，找到其中的这几句：

```
Hisilicon Nand Flash Controller V300 Device Driver, Version 1.00
Nand ID: 0xAD 0xDC 0x10 0x95 0x54 0xAD 0xDC 0x10
Nand(Hardware): Block:128K Page:2K Ecc:1bit Chip:512M OOB:64Byte
NAND device: Manufacturer ID: 0xad, Chip ID: 0xdc (Hynix NAND 512MiB 3,3V 8-bit)
```

红色字体的那一行里面就有 page 和 ecc 的大小。如果找不到这个打印，说明海思的发布包不支持这种类型的 flash。

## 1.7 为什么无法启动文件系统，提示 No init found?

常见的错误可能如下：

```
ata2: failed to resume link (SControl 0)
ata2: SATA link down (SStatus 0 SControl 0)
yaffs: dev is 32505858 name is "mtdblock2" rw
yaffs: passed flags ""
VFS: Mounted root (yaffs2 filesystem) on device 31:2.
Freeing init memory: 100K
Kernel panic - not syncing: No init found. Try passing init= option to kernel.
See Linux Documentation/init.txt for guidance.
```

可能的原因 1：yaffs 文件系统制作时输入的 pagesize 和 ecctype 错误，这两个参数如果与实际 nand flash 的属性不一样的话，内核将无法识别 yaffs 文件系统。SDK make build 的时候，默认按照随 SDK 发布的参考板制作文件系统，采用的 pagesize 和 ecctype 参数不一定与您的一致。

可能的原因 2：bootargs 配置错误，比如误把 bootargs 配置为：setenv bootargs 'bootargs=mem=96M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=yaffs2 mtdparts=hinand:4M(boot),60M(rootfs),-(others)' 之类或者 bootargs 中的 rootfstype 配置错误，比如烧写的是 jffs2 文件系统，bootargs 却配置为 yaffs2，也可能导致内核无法识别文件系统。



## 1.8 为什么无法启动文件系统，提示不能打开 console?

这是因为用来制作文件系统的 rootbox 里面没有/dev/console 文件，或者 console 文件的属性不对。正常的 console 文件属性如下：

```
cd SDK根目录
ls ./pub/rootbox/dev/ -l
总计 0
crw-r--r-- 1 root root 5, 1 2010-10-18 18:52 console
crw-r--r-- 1 root root 204, 64 2010-10-18 18:52 ttyAMA0
crw-r--r-- 1 root root 204, 65 2010-10-18 18:52 ttyAMA1
crw-r--r-- 1 root root 204, 64 2010-10-18 18:52 ttyS000
```

## 1.9 为什么 tftp 无法使用?

请按照如下步骤进行检查：

- 网线是否已经插上单板并连接正常？插任意一个网口都可以。
- 插在单板上的网线在其他地方是否可以正常使用？网线本身有没有问题？
- 单板与 PC 之间是否是通过网线直连？如果不是，这中间的网络是否需要认证或者代理？某些交换机会屏蔽非交换机本身动态分配的 IP 地址，在 boot 下直接用 setenv ipaddr 的方式配置 IP 地址可能会导致网络不通；某些公司的网管会屏蔽掉非特定范围内的其他 MAC 地址或者 IP 地址，也可能导致单板无法访问网络。推荐采用单板与 PC 直连的方式进行 tftp 操作。
- PC 上是否启用了 IPv6 协议？单板在 boot 下不支持 IPv6 协议，请关闭 PC 上对的 IPv6 协议的支持。

## 1.10 为什么在服务器上执行 SDK 的某些编译命令或者脚本，报“找不到这个文件”或者类似的错误？

这个很可能是因为服务器上安装的是 64 位的操作系统，而 SDK 要求服务器上有 32 位的 C 库导致的。解决办法，在服务器上安装 32 位的运行库。推荐通过 google 或者 baidu 搜索获取在服务器上安装 32 位运行库的方法。

参考命令(仅在 Ubuntu 64bit server 版服务器上测试过，注意服务器需要能够连接 Internet)：

```
apt-get install libc6-i386
apt-get install gcc-multilib g++-multilib libc6-dev-i386 libzip-dev
apt-get install ia32-libs lib32asound2 libasound2-plugins
apt-get install -y lib32nss-mdns lib32gcc1 lib32ncurses5 lib32stdc++6 lib32z1
libc6 libcanberra-gtk-module
dpkg -i --force-all getlibs-all.deb
```



getlibs-all.deb

<----- 请先下载 getlibs-all.deb 到服务器再运行。

## 1.11 制作 cramfs 文件系统的时候，报出文件大小被截为 16M 的警告，同时，制作的文件大小不对。

常见的错误如下：

```
root@Athena:~$ mkcramfs ./tools/ root.img
```

```
Directory data: 37924 bytes
```

```
Everything: 43936 kilobytes
```

```
Super block: 76 bytes
```

```
CRC: 40e6dc31
```

```
warning: file sizes truncated to 16MB (minus 1 byte)
```

```
warning: gids truncated to 8 bits (this may be a security concern)
```

出现这个错误原因是，本身的 cramfs 不支持单个文件大小大于 16M，如果需要制作单个文件大小大于 16M 的文件系统，请按照下面的流程检查修改。

- 请确保对 cramfs 文件系统的支持和对 MTD 驱动的支持
- 修改 mkcramfs 的源码，使其支持制作特殊的文件系统，从 <http://sourceforge.net/projects/cramfs/> 下载 cramfs 的源码，解压之后修改 cramfs/linux/cramfs\_fs.h 的 #define CRAMFS\_SIZE\_WIDTH 24，此处的 24 代表 16M，例如你需要支持 256M 的文件，可将其设置为 28。
- 修改内核的 cramfs 文件系统。修改内核源码目录的：include/linux/cramfs\_fs.h 文件，同样的方式修改宏 CRAMFS\_SIZE\_WIDTH



## 2 工具类

### 2.1 如何在单板上使用 GDB?

gdb 的可执行程序位于“~/osdrv/tools/board/gdb”目录下，请把对应编译器的 gdb 拷贝到单板的/usr/bin 目录下,改名为 gdb，加上可执行权限（用 `chmod a+x gdb` 命令），然后即可在单板上使用 gdb。或者挂载 nfs 目录后，使用 gdb 的绝对路径运行 gdb。

### 2.2 如何让 gdb 在调试过程中忽略信号量事件?

一般碰到的问题是 gdb 会打印：

```
Program received signal SIG32, Real-time event 32.  
0x4052d940 in __rt_sigsuspend () from /lib/libc.so.0
```

之类，然后停住等待用户输入命令“c”才继续运行。

这种消息往往我们并不关注，可以 gdb 的命令行里面用命令：

```
handle SIG32 pass noprint nostop
```

使 gdb 忽略 SIG32。其他消息也类似。

### 2.3 CPU 占用率 100%，如何优化?

- 步骤 1. 在单板上运行 `telnetd`，打开 telnet 服务。
- 步骤 2. 运行应用程序，在程序正常运行的状态下（不要按 `CTRL+Z` 把程序放到后台），telnet 登录单板。
- 步骤 3. 在 telnet 终端上运行“`top -d 1`”，查看哪个 pid 的线程 CPU 占有率最高，记下这个值。
- 步骤 4. 按‘q’退出 top，或者另外再起一个 telnet 终端登录单板。
- 步骤 5. 在 telnet 终端上运行 gdb，不需要带任何参数，进入 gdb 命令行后，输入“`attach pid`”，这里的 pid 指步骤 3 中记录下的 pid 的值。Attach 完成后，输入 `bt full`，查看堆栈信息，一般情况下，堆栈的最顶端（最先打印出来的那几个函数）就是 CPU 占用最高的那个





线程的函数。检查一下这个线程的循环里面是否有释放 CPU 的操作，加上 `usleep(20000)` 或者使用其他方式在这个线程里面出让 CPU。

步骤 6. 重新编译程序，验证 CPU 占用率是否有下降，如果没有，重复步骤 2 至步骤 6。

----结束

## 2.4 如何查看和修改寄存器

- 在程序中使用 `HI_SYS_ReadRegister`, `HI_SYS_WriteRegister` 接口进行寄存器的读写操作。
- 在单板命令行下，使用工具。

单板/`usr/sbin` 目录下，有一系列寄存器操作相关的工具，分别介绍如下：

工具	参数	功能	备注
<code>hmm</code>	参数 1: <code>address</code> , 必选 参数 2: <code>value</code> , 可选	把 <code>address</code> 对应的地址改写为 <code>value</code>	如果没有输入 <code>value</code> , <code>himm</code> 会把 <code>address</code> 对应的值打印出来，然后提示输入新的值。
<code>himd</code>	参数 1: <code>address</code> , 必选 参数 2: <code>length</code> , 可选	从 <code>address</code> 开始，以大端方式打印 <code>length</code> 字节长度的内容。	如果没有输入 <code>length</code> , 那么默认打印 256 字节。
<code>himd.l</code>	参数 1: <code>address</code> , 必选 参数 2: <code>length</code> , 可选	从 <code>address</code> 开始，以小端方式打印 <code>length</code> 字节长度的内容。	如果没有输入 <code>length</code> , 那么默认打印 256 字节。
<code>himc</code>	参数 1: <code>address</code> , 必选 参数 2: <code>value</code> , 必选 参数 3: <code>length</code> , 必选	从 <code>address</code> 开始，把 <code>length</code> 字节长度的内容全部设置为 <code>value</code> 。	一般仅用于物理内存内容的修改。

参数 `address` 既可以是寄存器地址，也可以是内存地址，所以以上工具也可以用于内存查看、修改。

## 2.5 如何使用 DDR 带宽统计工具 `hiddrs`

`Hiddrs` 用于实时统计 `ddr` 的带宽使用情况，参数说明如下。

`-d` 表征 `ddr` 控制器，0 代表控制器 0，1 代表控制 1，2 代表同时统计控制器 0 和控制器 1，默认是 0。

`-f` 表征 `ddr` 工作频率，默认 400MHz。

`-w` 表征 `ddr` 位宽，有 32bit 和 16bit，默认是 32bit。



-i 表征时间间隔，单位是秒。

-h 显示帮助信息。

示例：hiddrs -d 0 -f 400 -w 32 -i 1。

## 2.6 如何让 udhcpc 占用更少的内存？

这个问题的表面现象是用 system 调用的方式执行 udhcpc 会失败。解释：由于 system 是通过 fork 实现的，而子进程会复制父进程的 VM 空间，当父进程占用较多 VM 空间，很容易导致 system 调用失败。其本质是子进程分配 VM 空间失败导致的。

解决方法：执行：echo 1 > /proc/sys/vm/overcommit\_memory 即可。

更好的解决办法是不使用 system 调用方法，而是使用 posix\_spawn 调用，简单的示例如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <spawn.h>
#include <sys/wait.h>
int main(int argc, char* argv[])
{
    pid_t pid;
    int err;
    char *spawnedArgs[] = {"/bin/ls", "-l", "/home ", NULL}; /* posix_spawn需要指
    定子进程的命令的全路径（绝对路径） */
    char *spawnedEnv[] = {NULL};

    printf("Parent process id=%ld\n", getpid());
    if( (err=posix_spawn(&pid, spawnedArgs[0], NULL, NULL,
        spawnedArgs, spawnedEnv)) !=0 )
    {
        fprintf(stderr, "posix_spawn() error=%d\n", err), exit(-1);
    }
    printf("Child process id=%ld\n", pid);

    /* Wait for the spawned process to exit */
    (void)wait(NULL);

    return 0;
}
```

posix\_spawn 的更多用法，请自行上网搜索。



## 2.7 为什么有时候 udhcpd 无法获得 IP 地址.

单板上的 udhcpd 默认只发起 3 次 IP 请求, 有时候服务器反映比较慢, 需要增加请求次数。

增加请求次数的命令参数为: “udhcpd -t 50”。

## 2.8 当前 udhcpd 如何支持 DHCP+ 功能

可以使用如下参数:

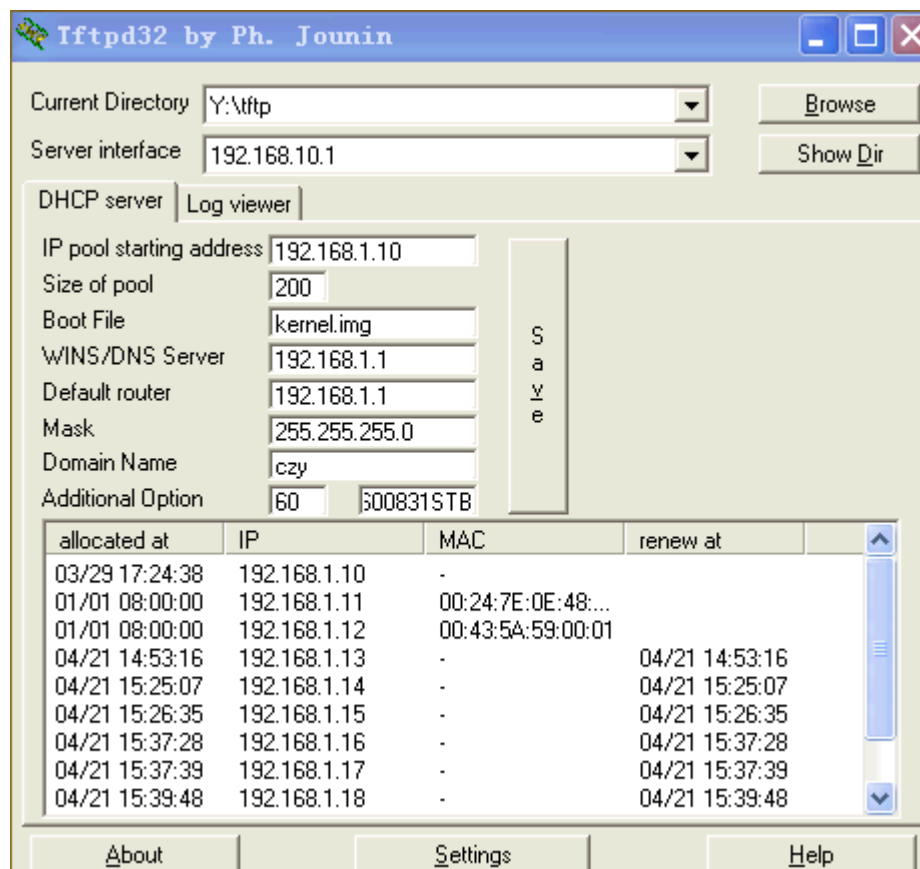
- -V 表示发给服务器的卖主信息
- -D 表示客户端期待的卖主信息.

例如: udhcpd -V question -D answer -f

- 单板发送 question 到服务器
- 服务器应该回答 answer 到单板
- 单板判断收到的数据是否为 answer, 如果不是, 配置失败, 如果是, 配置成功;

以下是带参数的配置:

```
# ./busybox udhcpd -V600831STB -D600831STB
udhcpd (v0.9.9-pre) started
Jan  1 01:31:15 udhcpd[800]: udhcpd (v0.9.9-pre) started
Sending discover...
Jan  1 01:31:15 udhcpd[800]: Sending discover...
Offer vendor class: 600831STB
Client vendor class: 600831STB
Sending select for 192.168.1.19...
Jan  1 01:31:17 udhcpd[800]: Sending select for 192.168.1.19...
Sending select for 192.168.1.19...
Jan  1 01:31:19 udhcpd[800]: Sending select for 192.168.1.19...
Offer vendor class: 600831STB
Client vendor class: 600831STB
Lease of 192.168.1.19 obtained, lease time 168960
Jan  1 01:31:19 udhcpd[800]: Lease of 192.168.1.19 obtained, lease time 168960
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.1.1
#
```



以下是不带参数的配置：



```
# ./busybox udhcpd
udhcpd (v0.9.9-pre) started
Jan 1 01:32:58 udhcpd[810]: udhcpd (v0.9.9-pre) started
Sending discover...
Jan 1 01:32:58 udhcpd[810]: Sending discover...
Sending discover...
Jan 1 01:33:01 udhcpd[810]: Sending discover...
Sending select for 10.85.180.250...
Jan 1 01:33:01 udhcpd[810]: Sending select for 10.85.180.250...
Received DHCP NAK
Jan 1 01:33:01 udhcpd[810]: Received DHCP NAK
/usr/share/udhcpd/default.script: exec: line 7: /usr/share/udhcpd/default.nak: not found
Sending discover...
Jan 1 01:33:04 udhcpd[810]: Sending discover...
Sending discover...
Jan 1 01:33:07 udhcpd[810]: Sending discover...
Sending select for 10.85.181.24...
Jan 1 01:33:07 udhcpd[810]: Sending select for 10.85.181.24...
Received DHCP NAK
Jan 1 01:33:07 udhcpd[810]: Received DHCP NAK
/usr/share/udhcpd/default.script: exec: line 7: /usr/share/udhcpd/default.nak: not found
Sending discover...
Jan 1 01:33:10 udhcpd[810]: Sending discover...

Sending discover...
Jan 1 01:33:13 udhcpd[810]: Sending discover...
Sending select for 10.85.181.93...
Jan 1 01:33:13 udhcpd[810]: Sending select for 10.85.181.93...
Lease of 10.85.181.93 obtained, lease time 28800
Jan 1 01:33:13 udhcpd[810]: Lease of 10.85.181.93 obtained, lease time 28800
deleting routers
route: SIOCDELRT: No such process
adding dns 10.72.55.81
adding dns 10.72.255.100
adding dns 10.98.48.39
#
#
```

# 3 外设类

## 3.1 如何使用 USB 键盘鼠标？

SDK 的内核默认情况下已经支持 USB 键盘鼠标，插入设备之后，就能直接使用，如下



sample 供参考：usb-key-mouse.zip

## 3.2 为什么 socket 组播会丢包？

用户态应用程序在接收 UDP 组播数据时,同时进行其它有延时的操作(如写码流数据到 USB 存储设备), 应该程序将延迟接收 UDP 数据包，而 socket 默认接收缓存只有 108544Byte，这样可能会使 socket 接收缓存满，无法接收新的 UDP 数据包，出现丢包现象。

以下命令可以增加接收缓冲区，解决以上问题。

```
echo 20000000 > /proc/sys/net/core/rmem_max
echo 20000000 > /proc/sys/net/core/rmem_default
echo 20000000 > /proc/sys/net/core/netdev_max_backlog
```

这种改动，需要根据实际码流发送速度和接收程序的延时调整。

## 3.3 如何修改以太网 PHY 地址？

- 内核下的方法:

在 osdvr/kernel/linux -3.0.y 目录下运行

“make ARCH=arm CROSS\_COMPILE=arm-hisiv200-linux- menuconfig”

在 menuconfig 菜单下，选择以下选项:

```
Device Drivers --->
    [*] Network device support --->
        [*] Ethernet (1000 Mbit) --->
```



```
<M> STMicroelectronics 10/100/1000 Ethernet driver --->
```

```
(1) STMMAC MAC #0 PHY ID << 表示 phy 地址是 1.
```

```
(2) STMMAC MAC #1 PHY ID << 表示 phy 地址是 2.
```

- u-boot 代码中, 修改方法如下:

u-boot 代码中,修改 “include\configs\godnet.h” 文件:

```
#define CONFIG_NET_STMMAC
#define CONFIG_TNK
#ifdef CONFIG_NET_STMMAC
    #define STMMAC_GMACADDR      (0x101c0000)
    #define STMMAC_DMAADDR      (0x101c1000)
    #define STMMAC_IOSIZE       (0x10000)
    #define STMMAC_FRQDIV       (0)
    #define STMMAC_PHYADDR0     (1)    << 表示 phy 地址是 1.
    #define STMMAC_PHYADDR1     (2)    << 表示 phy 地址是 2.
    #define STMMAC_PHYNAME      "0:01"
    #define STMMAC_RGMII
    #define CONFIG_PHY_GIGE
#endif /* CONFIG_NET_STMMAC */
```

### 3.4 GMAC 网口 ping 不通?

网口 ping 不通通常会有如下可能:

- IP 地址与其他设备的 IP 地址重复:

现象: PC 端 ping 单板延迟很大, 时断时通;

判断办法: PC 端想单板发送 ping 包, 同时拔掉单板对应网口的网线, 看是否依然可以 ping 通。

- MAC 地址与其他设备的 MAC 地址重复:

现象: PC 端 ping 单板 ping 不通;

判断办法: 从 PC 端 ping 单板, 同时在 PC 端使用 arp -a 命令查看 PC 端的 arp 列表。查看 IP 地址对应的 MAC 地址是否与单板上的 MAC 地址是对应的;

### 3.5 GMAC 网口状态查询

GMAC0: himd.l0x101c1014

GMAC1: himd.l0x101c1114

高 16bit 是逻辑状态机信息, 低十六位为上报的中断信息。具体的 bit 位信息可以参照 datasheet 上面的相关寄存器说明。比较重要的 bit 位如下图所示:

- GMAC 发送、接收状态:



22:20	TS: Transmit Process State	000	RO
These bits indicate the Transmit DMA FSM state. This field does not generate an interrupt.			
<ul style="list-style-type: none"><li>• 3'b000: Stopped; Reset or Stop Transmit Command issued.</li><li>• 3'b001: Running; Fetching Transmit Transfer Descriptor.</li><li>• 3'b010: Running; Waiting for status.</li><li>• 3'b011: Running; Reading Data from host memory buffer and queuing it to transmit buffer (Tx FIFO).</li><li>• 3'b100: TIME_STAMP write state.</li><li>• 3'b101: Reserved for future use.</li><li>• 3'b110: Suspended; Transmit Descriptor Unavailable or Transmit Buffer Underflow.</li><li>• 3'b111: Running; Closing Transmit Descriptor.</li></ul>			
19:17	RS: Receive Process State	000	RO
These bits indicate the Receive DMA FSM state. This field does not generate an interrupt.			
<ul style="list-style-type: none"><li>• 3'b000: Stopped: Reset or Stop Receive Command issued.</li><li>• 3'b001: Running: Fetching Receive Transfer Descriptor.</li><li>• 3'b010: Reserved for future use.</li><li>• 3'b011: Running: Waiting for receive packet.</li><li>• 3'b100: Suspended: Receive Descriptor Unavailable.</li><li>• 3'b101: Running: Closing Receive Descriptor.</li><li>• 3'b110: TIME_STAMP write state.</li><li>• 3'b111: Running: Transferring the receive packet data from receive buffer to host memory.</li></ul>			

● 中断状态类型:

register5 表示为当前的寄存器, []中的标识为 0~14bit 位, 其中 11、12bit 为保留。

16	NIS: Normal Interrupt Summary	0	R_SS_WC
Normal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Register 7:			
<ul style="list-style-type: none"><li>• Register 5[0]: Transmit Interrupt</li><li>• Register 5[2]: Transmit Buffer Unavailable</li><li>• Register 5[6]: Receive Interrupt</li><li>• Register 5[14]: Early Receive Interrupt</li></ul>			
Only unmasked bits affect the Normal Interrupt Summary bit.			
This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes NIS to be set is cleared.			
15	AIS: Abnormal Interrupt Summary	0	R_SS_WC
Abnormal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Register 7:			
<ul style="list-style-type: none"><li>• Register 5[1]: Transmit Process Stopped</li><li>• Register 5[3]: Transmit Jabber Timeout</li><li>• Register 5[4]: Receive FIFO Overflow</li><li>• Register 5[5]: Transmit Underflow</li><li>• Register 5[7]: Receive Buffer Unavailable</li><li>• Register 5[8]: Receive Process Stopped</li><li>• Register 5[9]: Receive Watchdog Timeout</li><li>• Register 5[10]: Early Transmit Interrupt</li><li>• Register 5[13]: Fatal Bus Error</li></ul>			
Only unmasked bits affect the Abnormal Interrupt Summary bit.			
This is a sticky bit and must be cleared each time a corresponding bit that causes AIS to be set is cleared.			





### 3.6 TOE 怎样使能？

现在的发布包网口默认使用的是 `bypass` 功能，使能 TOE 功能只需要如下操作即可：

修改文件系统的 `/etc/init.d/S81toe` 启动脚本：

将 `insmod /hitoe/stmmac.ko` 注释掉；

再将下列三句话使能，打开 TOE 功能：

```
echo 8192 > /proc/sys/vm/min_free_kbytes
echo 200 > /proc/sys/vm/vfs_cache_pressure
insmod /hitoe/stmmac.ko hitoe=1
```

这几个设置的详细说明在 `S81toe` 启动脚本中有详细的描述，这里不再赘述，另外由于这里保留空间增大了，相应的 `bootargs` 中 `MEM` 的配置也要相应的增大，否则会出现内存不够用的情况。

### 3.7 TOE 模式下应用程序使用 socket 接口时的注意事项：

在 TOE 模式下，不管是阻塞模式还是非阻塞模式，协议处理是由硬件完成的，当出现内核 buffer 满时，都会给上层应用返回 `EAGAIN`，而在 `Bypass` 模式下，如果是阻塞模式，则出现内核 buffer 满时，不会返回 `EAGAIN`，而阻塞继续发送！另外经过查询得知 `send` 的返回值是这样处理的：阻塞模式与非阻塞模式下，`send` 返回值 `< 0 && (errno == EINTR || errno == EWOULDBLOCK || errno == EAGAIN)` 表示暂时发送失败，需要重试，如果 `send` 返回值 `<= 0, && errno != EINTR && errno != EWOULDBLOCK && errno != EAGAIN` 时，连接异常，才需要关闭。

因此建议应用程序在调用 `send` 时，若返回值 `< 0 && (errno == EINTR || errno == EWOULDBLOCK || errno == EAGAIN)` 应该延时等待一段时间再重新尝试发送，而不要直接调用 `close` 断开连接，则不会出现此问题。

### 3.8 应用程序使用 socket 接口时，如何正确工作在非阻塞模式下？

**非阻塞 IO 和阻塞 IO：**

在网络编程中对于一个网络句柄会遇到阻塞 IO 和非阻塞 IO 的概念，这里对于这两种 socket 先做一下说明：

**基本概念：**

- **阻塞 IO：** socket 的阻塞模式意味着必须要做完 IO 操作（包括错误）才会返回。
- **非阻塞 IO：** 非阻塞模式下无论操作是否完成都会立刻返回，需要通过其他方式来判断具体操作是否成功。



## IO 模式设置:

对于一个 socket 是阻塞模式还是非阻塞模式有两种方式来处理:

- **方法 1:** fcntl 设置;用 F\_GETFL 获取 flags,用 F\_SETFL 设置 flags|O\_NONBLOCK; fcntl 函数可以将一个 socket 句柄设置成非阻塞模式:

```
flags = fcntl(sockfd, F_GETFL, 0); //获取文件的 flags 值。
```

```
fcntl(sockfd, F_SETFL, flags | O_NONBLOCK); //设置成非阻塞模式;
```

设置之后每次对于 sockfd 的操作都是**非阻塞**的。

```
flags = fcntl(sockfd, F_GETFL, 0);
```

```
fcntl(sockfd, F_SETFL, flags & ~O_NONBLOCK); //设置成阻塞模式;
```

设置之后每次对于 sockfd 的操作都是**阻塞**的。

- **方法 2:** recv、send 系列的参数。(读取, 发送时, 临时将 sockfd 或 filefd 设置为非阻塞)

recv、send 函数的最后有一个 flag 参数可以设置成 MSG\_DONTWAIT

临时将 sockfd 设置为非阻塞模式, 而无论原有是阻塞还是非阻塞。

```
recv(sockfd, buff, buff_size, MSG_DONTWAIT); //非阻塞模式的消息发送
```

```
send(sockfd, buff, buff_size, MSG_DONTWAIT); //非阻塞模式的消息接受
```

## 3.9 Atheros 8035 型号的 phy 使用注意事项



### 注意

该型号的 PHY 在 GTX\_CLK 管脚上的延迟需要通过配置 debug 寄存器来实现, 而不是像通用的 PHY 一样, 在硬件链路上通过添加器件实现。

相关寄存器描述如下:

4.2.25 rgmii rx clock delay control Offset: 0x00				
Bit	Name	Type		Description
15	Sel_clk125m_dsp	Mode	R/W	Control bit for rgmii interface rx clock delay: 1 = rgmii rx clock delay enable 0 = rgmii rx clock delay disable
		HW Rst.	1	
		SW Rst.	1	
14:0	Reserved	Mode	RO	
		HW Rst.	2EE	
		SW Rst.	2EE	



通过 mdio 操作如下寄存器可以对 debug 寄存器进行读写操作：

#### 4.1.23 Debug Port Address Offset

Offset: 0x1D  
Mode: Read/Write  
Hardware Reset: 0  
Software Reset: 0

Bit	Name	Description
15:6	RES	Reserved
5:0	ADDRESS_OFFSET	Address index to access the debug registers

#### 4.1.24 Debug Port Data

Offset: 0x1E  
Mode: Read/Write  
Hardware Reset: 0x82EE  
Software Reset: 0x82EE

Bit	Name	Description
15:0	DATA	Data contents of the debug registers as addressed by the "Debug Register Summary" register

操作示例如下：

- uboot 下：  
mw 0x101c0014 0x5  
mw 0x101c0010 0x1743  
mw 0x101c0014 0x0100  
mw 0x101c0010 0x1783
- kernel 下：  
在 drivers/net/stmmac/stmmac\_main.c 文件的 stmmac\_open 函数中加入如下操作：  
priv->mii->write(priv->mii, 0x2, 0x1d, 0x5);  
priv->mii->write(priv->mii, 0x2, 0x1e, 0x0100);



### 注意

该型号的 PHY 调试中出现过 boot 下百兆不通，读寄存器发现被适应成千兆；这是因为该 PHY 寄存器 0F 的读取值应该为 0x2000，但实际值为 0xA000，这样驱动算法会得出该速率为千兆的能力。

解决办法请直接注掉 uboot 下文件 common/miiphyutil.c 中红色部分

```
函数 int miiphy_speed (char *devname, unsigned char addr)
{
```



```
        u16 bmcrr, anlpar;  
        u16 btsr, val;  
        miiphy_read(devname, addr, PHY_BMSR, &val);  
        if(val & BMSR_ESTATEN){  
#if 0  
            if (miiphy_is_1000base_x (devname, addr)) {  
                return _1000BASET;  
            }  
#endif
```

### 3.10 网线插到到单板上，为什么还是会报 Phy no link?

3531 demo 板第一次执行网络操作的时候，会出现如下的打印：

```
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2  
No such device: 0:2
```



```
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
No such device: 0:2
PHY not link!
```

明明网线已经连接到单板上了，为什么还会报 PHY not link 呢？这是因为 demo 板上的 phy 需要一些时间与对端设备的 phy 进行工作模式以及速度的协商，完成后还要一定时间的复位。可能会导致插上网线后立即执行网络操作（ping、tftp 或者其他操作）时出现如上失败的情况。只需等到单板 phy 的电源指示灯（一般为绿色）变亮后，再执行网络操作即可。

### 3.11 为什么有时会出现串口无任何响应的情况？

调试代码时经常出现 PC 侧的终端程序无法响应键盘输入、也不再打印任何输出信息的情况。造成这种现象有两种原因：

- 程序或内核挂死
- 程序还在运行、但串口无响应

要判断究竟是哪种原因，这里介绍两种办法：

- 首先需要在内核启动后开启 telnet，然后待出现挂死现象后，telnet 单板，如果不能连上，说明是第一个原因。反之则是第二个原因
- 第二种方法，则是出现串口无响应现象后，通过仿真器或 telnet 查看串口内部寄存器。以 Hi3531 为例，查看串口寄存器 0x20080000 第 11bit，以及 0x20080004 的第 3bit，这两者只要有一个值为 1，就表示出现串口溢出错误，导致串口无响应。

针对第二种原因，可以修改串口驱动，出现溢出错误后直接重新初始化串口，从而避免串口无响应。请用以下文件来替换发布包的

“~/osdrv/kernel/linux-3.0.y/drivers/tty/serial/amba-pl011.c”，然后重新编译内核即可。

注意，该文件基于 linux-3.0.y 版本。



amba-pl011.c

### 3.12 Hi3518 制作 RMII 软件版本

## 第一步修改 uboot 表格:

1. 修改 `crg` 寄存器 `0x200300cc` 中 bit 3 为 `rmii` 接口模式, bit 2 为选择 PAD 输入时钟(为兼容硬件 `demo` 板选择外部时钟源), 配置初值为 `0xe`, 见表格红色部分;

[3]	RW	mii_rmii_mode	ETH MII, RMII 模式配置。 0: MII 模式; 1: RMII 模式。
[2]	RW	eth_rmiic_sel	ETH RMII CLK 时钟源选择。 0: 选择内部 CRG 时钟; 1: 选择 PAD 输入时钟。

模块名	pili																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			</
-----	------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

- ## 2. 修改管脚复用寄存器 0x200f005c,使用 RMII CLK 模式

配置初值为 0x3，见表格红色部分；

[1:0]	RW	muxctrl_reg23 MII_TXCK 管脚的具体复用情况。 00: GPIO3_3; 01: MII_TXCK; 10: VOU1120_DATA7; 11: RMII_CLK。
-------	----	--

模块名	standby_pin_ctrl																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
-----	------------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--



## 第二步 uboot 命令行配置 rmii 模式

使用第一步中的表格制作 uboot 镜像文件，当单板使用该 uboot 起来后，输入命令：

```
setenv mdio_intf rmii
```

然后保存，即可设置成 rmii 模式，见下图最后一行。

```
hisilicon # pri
bootdelay=1
baudrate=115200
ethaddr=00:00:23:34:45:66
netmask=255.255.254.0
bootfile="uImage"
bootargs=mem=256M console=t
2M(rootfs),16M(test)
serverip=192.168.0.234
ipaddr=192.168.0.123
bootcmd=nand read 0x8200000
stdin=serial
stdout=serial
stderr=serial
verify=n
ver=U-Boot 2010.06 (Feb 13
mdio_intf=rmii
```

## 第三步制作 rmii 内核镜像

编译内核 uImage 镜像时，在打开 menuconfig 之后，按如下路径进入配置 rmii 模式：

```
Device Drivers --->
[*] Network device support --->
[*] Ethernet (10 or 100Mbit) --->
<*> hieth(switch fabric) family network device support --->
(1) hieth mii/rmii mode for up port (mii:0/rmii:1)
(1) hieth mii/rmii mode for down port (mii:0/rmii:1)
```

## 3.13 Uboot 下如何调试部分不兼容的 U 盘？

Uboot 下目前支持 usb2.0 的 OHCI 协议，和 usb3.0 的 XHCI 协议，由于两种协议的不同，在相同的 uboot 初始化流程中，出现了部分可以在 usb2.0 端口成功识别但是在 usb3.0 端口无法识别的 usb3.0 U 盘。





在 uboot 下测试 U 盘兼容性过程中,发现部分 U 盘在上电后到被控制器识别所需时间不同,当前 uboot 代码中将延迟参数 CONFIG\_USB\_HUB\_MIN\_POWER\_ON\_DELAY 配置为 3000ms,有较佳的兼容性,但是仍有部分 U 盘在 usb3.0 端口无法识别,如 Kingston 的 DTG3 和 DataTraveler 111, EAGET 的 F30, Apacer, SSK 的 SFD201。

通过调节参数 CONFIG\_USB\_HUB\_MIN\_POWER\_ON\_DELAY 为 1000ms 时也可以正常兼容以上型号,但该值又会影响其他型号。

所以如果出现 U 盘在 uboot 下无法识别,可以尝试修改:

文件 common/usb.c 中宏定义 CONFIG\_USB\_HUB\_MIN\_POWER\_ON\_DELAY 的值。

### 3.14 使用 netstat 命令查看链接状态, IP 地址显示为 NULL

使用 glibc 的系统可能会出现此种情况。netstat 通过调用 libc 的 getnameinfo 接口获取链接的 ip 地址,此接口的 glibc 与 uclibc 实现有差异,netstat 调用 glibc 的 getnameinfo 时返回 NULL。在 netstat 后加 -n 参数查看,可以显示正常的 IP 地址。或者修改 busybox 的 networking/netstat.c:

找到 ip\_port\_str()函数,将以下代码替换

```
host = numeric ? xmalloc_sockaddr2dotted_noport(addr)
               : xmalloc_sockaddr2host_noport(addr);
```

修改为

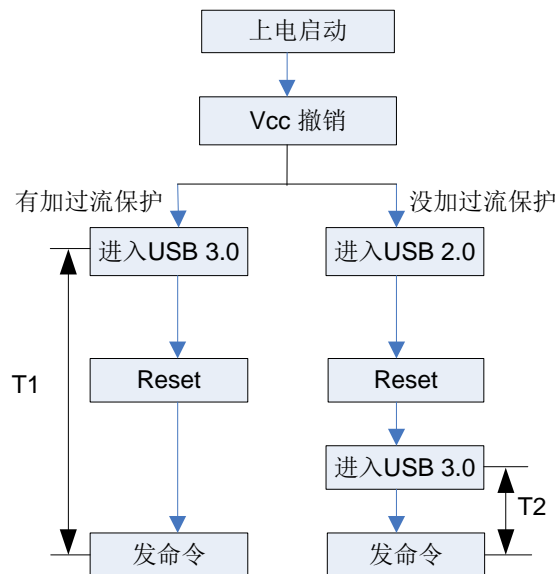
```
host = NULL;
if (!numeric)
    host = xmalloc_sockaddr2host_noport(addr);
if (!host)
    host = xmalloc_sockaddr2dotted_noport(addr);
```

### 3.15 为什么 USB3.0 口过流保护芯片去掉,会导致某些 USB3.0 U 盘上电启动不识别?

USB3.0 口去掉过流保护芯片,发现一款台电科技的 USB3.0 的 U 盘插着上电后,不能被识别。造成这个问题的原因是控制器处于 USB3.0 的时间太短,USB3.0 的 U 盘还未初始化完全导致识别失败。如图 3-1 所示。



图3-1 USB3.0 U 盘启动识别流程



去掉过流保护芯片的 USB3.0 控制器，会先进入 USB2.0 状态，再 reset 后进入 USB3.0 状态。这样，控制器处于 USB3.0 状态的时间 T2 远远小于有过流保护芯片直接进入 USB3.0 状态的时间 T1，导致 u 盘的状态还没有准备好，最终 host 端发送命令失败。解决方法是在 drivers/usb/core/hub.c 文件中将 HUB\_ROOT\_RESET\_TIME 的值延长为 100 即可：

```
#define HUB_ROOT_RESET_TIME    100;
```

## 3.16 I<sup>2</sup>C 内核态接口原子操作注意事项

在 I<sup>2</sup>C 设备驱动中用到的内核态接口函数有 i2c\_master\_send, i2c\_master\_recv 和 i2c\_transfer，这几个接口函数内部都会根据原子或非原子操作申请不同的锁，如图 3-2 所示。

图3-2 内核态接口中申请锁的操作

```
if (in_atomic() || irqs_disabled()) {
    ret = i2c_trylock_adapter(adap);
    if (!ret)
        /* I2C activity is ongoing. */
        return -EAGAIN;
} else {
    i2c_lock_adapter(adap);
}
```

调用 I<sup>2</sup>C 接口函数之前如果使用了原子锁或在中断中调用都会使当前操作处于原子操作中，如果处于原子操作会运行图 3-2 中上半部的 if 分支，否则会运行下半部的 else 分支。



### 注意

在原子操作中会通过 `i2c_trylock_adapter(adap)` 来尝试请求锁，如果出错返回 `-EAGAIN`，则表示没有得到锁，而不是 I2C 通信出现问题。这种情况下读写操作是没有执行的，对于要写的值没有写进去；对于读操作值没有意义。因此需要进一步判断错误返回值是否等于 `-EAGAIN`，如果是就要根据情况决定是否重复调用，以图 3-3 I2C 写为例进行说明。

图3-3 写操作示例图

```
/*
 * i2c_write_foo() - write i2c sensor register sample
 * @client: i2c slave
 * @reg: sensor register address
 * data: sensor register data to written
 *
 * Before the call, you need to do the following,
 * and i2c_lock is global in this file.
 *     spinlock_t i2c_lock;
 *     spin_lock_init(&i2c_lock);
 *
 * Return: zero on success, else a error code.
 */
int i2c_write_foo(struct i2c_client* client, unsigned char reg,
                  unsigned char data)
{
    int ret;
    unsigned char buf[2];
    unsigned long flags;

    buf[0] = reg;
    buf[1] = data;
    do {
        spin_lock_irqsave(&i2c_lock, flags);
        ret = i2c_master_send(client, buf, 2);
        spin_unlock_irqrestore(&i2c_lock, flags);
        if (ret == 2) {
            break;
        } else if (ret == -EAGAIN) {
            continue;
        } else {
            printk("[%s %d]i2c_master_send error, ret = %d",
                   __func__, __LINE__, ret);
            return ret;
        }
    } while (1);

    return 0;
}
```



## 3.17 为什么 USB2.0 对接 BCM WIFI 模组时，模组加载固件失败？

Hi3518EV200(含 Hi3518EV201、Hi3516CV200)对接 BCM43143 WIFI 模组时（其它品牌模组暂未发现），加载固件失败。此模组需要模拟断连一次，模组的 firmware 在规定的计数内（可配）断连不成功，则加载驱动失败。打印 log 如下：

```
~ # ./bcm43143 -n nvram_wubb-738gn.nvm brcm43143.bin.trx -C 10
version: 0.2
argv=-n
nvfn=nvram_wubb-738gn.nvm
argv=brcm43143.bin.trx
fwfn=brcm43143.bin.trx
argv=-C
cnt=10
Vendor 0xa5c ID 0xbd1e
claiming interface 0
Found device: vend=0xa5c prod=0xbd1e
ID : Chip 0xa887 Rev 0x2 RamSize 458752 RemapBase 0x60000000 BoardType 0
BoardRev 0
Final fw_path=brcm43143.bin.trx
Final nv_path=nvram_wubb-738gn.nvm
File Length: 358596
start
rdl.state 0x4
elapsed download time 0.214301
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=0
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=1
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=2
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
```



```
No devices found
Error: usbdev_find ... cnt=3
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=4
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=5
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=6
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=7
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=8
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=9
Vendor 0xa5c ID 0xbd1e
Vendor 0x1d6b ID 0x2
Vendor 0x1d6b ID 0x1
No devices found
Error: usbdev_find ... cnt=10
Run out of cnt
Error: can not find bdc device
```

#### 【解决方案】

建议如下：

1. 打开 EOP 预加重。



2. 在 BCM43143 USB2WIFI firmware 中，增加判断，在 Firmware 加载不成功时，通过一次或多次动态逐渐降低阈值（推荐 600mv、575mv、550mv、525mv，4 档），实现对 BCM WIFI 自动适配，确保在超时之前，Firmware 加载成功。
3. 如果对功耗不是特别关注，考虑不使用 SVB 方案，采用单板固定电压供电。具体电压，参考芯片手册中的设计要求。
4. 切换阈值方法和开关 EOP 预加重的方法如下。

```
//打开eop 预加重
himm 0x20120080 0x1900
himm 0x20120080 0x1920

//关闭eop 预加重
himm 0x20120080 0x1c00
himm 0x20120080 0x1c20

//disconnect threshold = 650mV
himm 0x20120080 0x1b0a
himm 0x20120080 0x1b2a

//disconnect threshold = 625mV
himm 0x20120080 0x190a
himm 0x20120080 0x192a

//disconnect threshold = 612mV
himm 0x20120080 0x1d0a
himm 0x20120080 0x1d2a

//disconnect threshold = 600mV
himm 0x20120080 0x110a
himm 0x20120080 0x112a

//disconnect threshold = 587mV
himm 0x20120080 0x150a
himm 0x20120080 0x152a

//disconnect threshold = 575mV
himm 0x20120080 0x1f0a
himm 0x20120080 0x1f2a

//disconnect threshold = 562mV
himm 0x20120080 0x050a
himm 0x20120080 0x052a

//disconnect threshold = 550mV
```



```
himm 0x20120080 0x170a
himm 0x20120080 0x172a

//disconnect threshold = 537mV
himm 0x20120080 0x0d0a
himm 0x20120080 0x0d2a

//disconnect threshold = 525mV
himm 0x20120080 0x070a
himm 0x20120080 0x072a

//disconnect threshold = 500mV
himm 0x20120080 0x0f0a
himm 0x20120080 0x0f2a
```

由于此方案会增加 wifi 连接的延时，具体档位选择和计数总数可自行权衡。



# 4 PCIe

---

## 4.1 如何配置内核选项，把 PCIe 控制器驱动编进内核？

在 RC 模式下，内核启动必须执行 PCIe 控制器驱动，完成控制器初始化和其它 PCIe EP 设备的枚举。在 EP 模式下，内核不执行 PCIe 控制器驱动（默认由逻辑完成 EP 模式的配置）。RC 模式下的具体配置：

运行 “make ARCH=arm CROSS\_COMPILE=arm-hisiv100-linux- menuconfig”

在 menuconfig 菜单下，选择以下选项：

```
Bus Support --->
  [*] PCI support --->
  [*] PCI Express support --->
Bus Support --->
  [*] PCI support --->
  [*] Hisilicon PCI Express support --->
```

EP 模式下，请将 PCI support 关闭。

```
Bus Support --->
  [] PCI support --->
```

注意：EP 模式下一定不能将该选项选上。EP 模式下，板子不能通过 PCIe 接口外接其它设备。

## 4.2 如何配置 PCIe 的控制时钟？

PCIe 模块的工作时钟，主要有两个来源：一个是来自芯片内部，称作内部时钟；一个是来自芯片外部，称作外部时钟。当 PCIe 工作在 RC 模式下时，通常使用内部时钟；如果要使用外部时钟，则时钟信号必须外接专用的时钟源。当 PCIe 工作在 EP 模式下时，通常使用主设备的输出时钟。



主设备 (RC) 的时钟通过 PCIe0 PHY 控制寄存器 0 (0x200500B4) 的第 0 比特来控制。时钟控制必须在 PCIe 系统控制器使能之前完成。

运行下面命令查看当前时钟的使用状况 (以 hi3531 为例):

```
himd.l 0x200500B4
```

运行下面命令, 让 PCIe 使用外部时钟:

```
himd.l 0x200500B4 0x05605000
```

运行下面命令, 让 PCIe 使用内部时钟:

```
himd.l 0x200500B4 0x05605001
```

另外, PCIe 工作在 RC 模式下, 必须将时钟输出, 以供从设备使用。以 hi3531 为例, 寄存器 PERI\_CRG30 (0x20030078) 的第 7, 8 比特必须为 0, 才能识别到从设备。除非从设备的时钟不由主设备提供, 我们不建议主从时钟不同源这种用法!

## 4.3 如何查看 PCIe 设备的 BAR 地址的分配信息?

PCIe 设备 BAR 地址在系统启动时分配, 其信息存放在 PCIe 配置空间中。以 Hi3531 为例, 配置空间中偏移地址 0x10、0x14、0x18 中分别存放了 BAR0、BAR1、BAR2 的地址信息。其中 BAR0 大小 8M, 为可预取。BAR1 大小 64K, BAR2 大小 1M, 都是不可以预取的。

对 PCIe 控制器 0 下面接的第一个设备:

```
himd.l 0x40100000
0000: 353219e5 00100140 04800001 00000008
0010: 30800008 31500000 31400000 37800001
0020: 00000000 00000000 00000000 00000000
0030: 00000000 00000040 00000000 00000149
0040: 5bc35001 00000008 00000000 00000000
0050: 00807005 00000000 00000000 00000000
0060: 00000000 00000000 00000000 00000000
0070: 00020010 00008702 00002010 00423c11
```

上面蓝色字体就是从设备的 BAR 地址信息。

对 PCIe 控制器 1 下面的第一个设备而言, 配置空间基址为: 0x70300000。

## 4.4 如何查看 PCIe 地址映射信息?

我们的 PCIe 地址映射信息, 保存在 PCIe 配置空间的 IATU 寄存器组中。IATU 寄存器组共有 6 组。每组寄存器有输入和输出两个方向。寄存器组和方向的选择, 都通过 Viewport 寄存器控制 (PCIe 配置空间偏移量为 0x900)。下面以 hi3531 为例, 说明如何使用该寄存器组:





```
himm 0x40100900 0x0
```

选择 IATU 寄存器组 0，方向：从设备看主设备（输入）。

```
himm 0x40100900 0x800000001
```

选择 IATU 寄存器组 1，方向：主设备看从设备（输出）。

例如：查看已选择的 IATU 寄存器组的地址映射信息

```
himd.l 0x40100900
0000: 00000000 00000000 00000000 00000000
0010: 00000000 0000ffff 00000000 00000000
0020: 00000000 00000000 00000000 00000000
0030: 00000000 00000000 00000000 00000000
0040: 00000000 00000000 00000000 00000000
0050: 00000000 00000000 00000000 00000000
0060: 00000000 00000000 00000000 00000000
```

以上信息表示 IATU 寄存器组 0 在输入模式下还没作任何配置。

## 4.5 PCIe MCC 模块的驱动插入后为何不起作用？

PCIe MCC 的使用有如下几点需要注意：

- 主侧和从侧所使用的 MCC ko，必须分别编译！编译主侧的 MCC ko 时，需要依赖 PCIe 控制器的驱动，请确保 PCIe 控制器的驱动已经被编译进内核中（在 menuconfig 中，PCIe 编译选项应该是选上的），同时内核必须确保已经编译完成了。编译从侧的驱动则没有此要求。
- 主侧的 MCC ko，只能运行在已经把 PCIe 控制器驱动编译进内核的镜像中。

对于 PCIe 主侧的 PCIe MCC 的 ko 来说，如果以上两个条件有一个不满足，都可能导致驱动没有起作用。

## 4.6 PCIe-网卡、PCIe-sata 使用注意事项！

使用以上设备时，请注意内核中对于的配置选项都应该选上（以 Hi3531 为例）！

PCIe-网卡：

```
Device Drivers --->
[*] Network device support --->
[*] Ethernet (1000 Mbit) --->
<*> SysKonnnect Yukon2 support
```

PCIe-sata:

```
对 Silicon Image 3124/3132:
Device Drivers --->
```



```

-- Serial ATA and Parallel ATA drivers --->
<*> Silicon Image 3124/3132 SATA support --->

```

对 JMB 362:

```

Device Drivers --->
-- Serial ATA and Parallel ATA drivers --->
<*> AHCI SATA support --->

```

## 4.7 通过 PCIe 实现从启动失败常见的情况

- 编译从片内核时，请务必选上 menuconfig 中的如下选项  
General setup --->  
[\*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
- 为支持超过 4M 以上的 cramfs，请修改 .config 中的宏  
CONFIG\_BLK\_DEV\_RAM\_SIZE=65536
- 目前发布的级联启动中，加载到从片的文件均不宜超过 7M
- 如果你采用发布包中提供的从启动程序(booter)来启动从设备，请按如下设置配置 u-boot 的环境变量  
setenv bootargs 'mem=64M console=ttyAMA0,115200'  
setenv bootcmd 'bootm 0x81000000 0x82000000'

## 4.8 为什么运 SDK 视频预览的业务后，偶尔会打印 “unknown Hi-irq triggered”

该打印只是起提醒作用，并非错误。

主从级联应用场景中，主从之间完成一个消息的通信过程如下：

从侧发起一个消息写，随后触发主侧一个中断，主侧响应中断，在共享内存区中取得消息；接下来主片可能会回复消息给从片，采用同样的程序，先写消息到共享内存，然后触发从侧中断。从侧进入中断服务程序，处理已经存在共享内存的消息。

按照最初的想法，本侧每次向对端提交中断的时候，先查看对端中断状态，若中断状态未被清除，则等待，待对端中断处理完毕，再触发对端中断。这种方法能保证一个消息，一个中断，但会造成对端频繁进入中断服务程序，效率较低。为提高消息交互的效率，考虑以下方法，对实时性要求较低的消息，通过定时器对多个消息进行一次性处理。每次发送消息后，即使上一个中断还没有被响应，触发对端中断时不需要等待对端的中断状态被清除，直接提交中断。这样就存在一种情况，本侧发送一个消息过去后，写了对端的中断，但还没来得及触发对端中断，这个对端的上一个中断刚好正在处理，顺便也把本次发送的消息给处理了；这个时候对端的中断状态也被清除了。等到本侧触发对端中断的时候，对端在检测中断状态的时候，却没有找到相应的中断状态标志，于是就打印了上面那个信息（“unknown Hi-irq triggered”）。

以上这个机制经过深入分析，不会导致丢消息，也不会有其他异常！



## 4.9 Hi3531/Hi3532 PCIe BAR 地址，在 reset 时默认会映射到哪个地址？在移动窗口时，有什么需要注意的？

在系统 reset 之后。Hi3531/Hi3532 的地址映射没有打开，也就是说，窗口不会映射到从设备的任何地址空间。只有执行窗口配置操作之后，地址映射才会打开。

PCIe 窗口移动时有一个问题需要注意：配置窗口时要求地址对齐。

Hi3531/Hi3532 芯片 PCIe 控制器中，共有 3 个可用 BAR 地址，大小分别为 8M、64K 和 1M。如果你用 8M 的窗口去映射从侧的某个区域，给定的这个区域的起始地址注意也应该是 8M 对齐的。比如你要映射到从侧的 0x84200000，这个没有问题；但是如果你想直接映射到 0x84270000，就会有问题，因为它不是 8M 对齐。如果你想操作 0x84270000 这个地址，你可以把窗口起始地址映射到 0x84200000，然后通过偏移 0x70000 去访问这个地址。

另外，1M 的窗口要求 1M 地址的对齐，64K 的则要求 64K 地址对齐。

## 4.10 在 PCIe MCC 驱动中，为了支持 Hi3531 和 Hi3532 级联，不再支持主片往从片 DMA 写操作，所有 DMA 操作都由从片发起。原来主片往从片 DMA 写操作，可以通过从片往主片 DMA 读操作替代。

SPC070 以前版本支持主从片之间双向 DMA 写操作。经长期测试发现，双向 DMA 写操作，加上主从片之间的其它非 DMA 的数据传输，可能会导致一些不可预知的异常。因此，在 SPC070 的版本取消了对主片向从片 DMA 写操作的支持，该操作可由从片 DMA 读替代。从片 DMA 同时进行读写操作，加上其它非 DMA 的数据传输，在实验板上通过长期测试，运行稳定。

从片 DMA 读写操作，在软件上通过两个任务链表加以管理（原来只是一个任务链表），实现了 PCIe 收、发两个通道同时收发数据的功能。本方案对 PCIe 通道的利用率，与主片、从片同时发起 DMA 写操作比较，基本一致。

## 4.11 为什么 Hi3531/Hi3532 级联，从片使用发布包 PCIe MCC 中直接编译出来的 ko 会起不来？

发布包中，默认情况下，主片是 Hi3531，从片是 Hi3532。可以在 osdrv/drv/pcie/hi35xx\_dev/slave/config.h 中看到配置如下：

```
// #define CONFIG_GODNET 1 /* Hi3531 */
#define CONFIG_GODCUBE 1 /* Hi3532 */

#define PCIE_SLAVE_CONNECTOR0 1
// #define PCIE_SLAVE_CONNECTOR1 1
```



注：宏 `CONFIG_GODXXX` 指定了从片使用的是哪个芯片，默认是 `Hi3532` (`CONFIG_GODCUBE`)。

如果您使用的从片是 `Hi3531`，则修改 `osdrv/drv/pcie/hi35xx_dev/slave/config.h` 如下：

```
#define CONFIG_GODNET 1
//#define CONFIG_GODCUBE 1
//请根据实际情况配置 PCIE_SLAVE_CONNECTORX:
//#define PCIE_SLAVE_CONNECTOR0 1 /* 控制器0 */
#define PCIE_SLAVE_CONNECTOR1 1 /* 控制器1 */
```

这里需要将 `CONFIG_GODCUBE` 关闭，并打开 `CONFIG_GODNET` (`Hi3531`)；由于 `Hi3531` 有两个 PCIe 控制器，用户可以使用其中一个来与主片连接（硬件决定）。这需要根据实际情况配置使用从片的哪个控制器：`PCIE_SLAVE_CONNECTOR0` 代表从片 `Hi3531` 使用控制器 0，`PCIE_SLAVE_CONNECTOR1` 代表从片使用的是控制器 1（默认）。

这里需要手动配置是因为如果从片是 `Hi3531`，软件无法预知它实际使用哪个控制器。也就是说不能确定哪个地址空间有效。因此需要进行配置。

## 4.12 Hi3532 作板卡的说明

- 问题提出：

Windows 7 以下的 Windows 操作系统，不支持对 PCIe 扩展配置空间的访问。但 `Hi3532` PCIe 一些关键功能的实现，比如 DMA 操作，窗口映射等，要求必须对 PCIe 扩展配置空间进行访问。为了兼容各个版本的 Windows 操作系统，用原来直接对扩展配置空间进行访问的方法显然不可行。且 `Hi3532` 复位时 PCIe 窗口映射默认是没有打开的。如果不能进行窗口映射操作，PCIe 设备（`Hi3532`）是无法进行地址映射和 DMA 操作的。

- 解决方案：

针对该情况，提出如下解决方案：在 PC 和 `Hi3532` 级联的场景中，利用 `Hi3532` 3 个 Bar 地址中的一个 (`BAR2`)，固定用于映射到 `Hi3532` PCIe 的配置空间 (4K)，而这个“事先”映射，需要由 `Hi3532` 的 u-boot 来完成。这样，PC 就可以通过访问这个 Bar 地址，来实现对整个 PCIe 配置空间的访问。

本方案要求在 PC 对 `Hi3532` 进行任何操作之前，`Hi3532` 自己先将 `Bar2` 映射到本侧的 PCIe 配置空间上面。为实现该操作，`Hi3532` 必须添加使用一块容量较小（比如 2MB）的 flash 存放启动的 u-boot，在 u-boot 中将 `Bar2` 映射到本侧的 PCIe 配置空间。在 PC 和 `Hi3532` 起来之后，PC 就可以通过 `Bar2`，对 `Hi3532` PCIe 整个配置空间进行访问。

- 具体的改动：

`Hi3532` spi flash 上的 u-boot (arch/arm/cpu/godcube/start.S)：



start.S

在 u-boot 启动之后，在 DDR 初始化之前，先做如下配置：



- 步骤 1. disable Hi3532 PCIe 控制器;
- 步骤 2. 设置 Hi3532 时钟为内部时钟;
- 步骤 3. 将 Hi3532 的 Bar2 配置映射到 Hi3532 的配置空间上;
- 步骤 4. 切换 Hi3532 时钟为外部时钟;
- 步骤 5. enable Hi3532 PCIe 控制器;
- 步骤 6. 循环等待, 检测一个寄存器 (0x20030028), 如果 PC 更新了该寄存器的值 (由 0x11 更新为 0x1), 则跳过 DDR 初始化 (DDR 初始化由 PC 完成), u-boot 正常启动; 否则继续检测。

运行在 PC 上的 PCIe MCC 模块:

所有 PCIe 窗口映射的配置操作, 都必须通过 Hi3532 的 Bar2 来完成配置。代码 (~/drv/pcie\_mcc/hi35xx\_dev/host/hi35xx\_dev\_host.c)



hi35xx\_dev\_host.c

----结束

- 其它说明:
  - 注意不要将 Hi3532 的 Bar2 映射到其他地方。
  - Hi3532 从 spi flash 启动;
  - 其它操作不变。

## 4.13 PCIe MCC 支持主设备复位从设备吗?

PCIe MCC 支持主设备复位从设备。说明如下:

- 复位后完全保留复位之前的设备状态, 包括设备功能状态、地址映射等 (DMA 相关的寄存器除外);
- 主设备可以连续复位从设备, 但注意复位期间需要有足够的时间等待从设备启动;

编译好主片驱动后, 在 drv/pcie\_mcc/koes 下面会生成一个可执行文件 booter。该文件是启动从设备和复位从设备的一个简单示例, 由 drv/pcie\_mcc/multi-boot/example 目录下的源码编译而成。具体的用法如下,

启动从设备:

```
$. /booter start_device
```

复位从设备:

```
$. /booter reset_device
```



更详细的信息，请参考~/pcie\_mcc/multi\_boot/example/boot\_test.c 和驱动代码。



说明  
主设备复位从设备的功能是在 SPC70 的版本中才实现的，以前的版本中不支持该功能。

## 4.14 使用某些 PCIE 转 SATA 卡（如：marvel9215）连接 sata 盘读写数据，导致出现 v0 低带宽现象的解决方法（以 Hi3536 为例）

- 问题原因：  
该 PCIE 转 SATA 卡默认请求数据大小为 512Byte，导致 pcie 带宽占用率过高。
- 解决方法：  
通过修改 pcie 转 sata 卡的 Max\_Read\_Request\_Size 寄存器，可限制它的读请求最大为 256Byte，降低 pcie 带宽占用率。  
具体操作步骤如下：

步骤 1. 执行命令：cp arch/arm/configs/hi3536\_full\_defconfig .config

步骤 2. 执行命令：make ARCH=arm CROSS\_COMPILE=arm-hisiv300-linux- menuconfig

步骤 3. 在 menconfig 界面中，配置选项 Bus support --->Hisilicon PCI Express support---> PCI Express configs--->limit pcie max read request size，如下图 4-1 所示：



图4-1 menconfig 界面

```
[*] Patch physical to virtual translations at runtime
General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
System Type --->
[ ] 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 --->
[ ] Virtualization --->

[*] PCI support
[ ] PCI Debugging
[ ] Enable PCI resource re-allocation detection
< > PCI Stub driver
[ ] PCI IOV support
[ ] PCI PRI support
[ ] PCI PASID support
[*] PCI Express support
[*] Root Port Advanced Error Reporting support
[ ] PCI Express ECRC settings control
< > PCIe AER error injector support
[*] PCI Express ASPM control
[ ] Debug PCI Express ASPM
Default ASPM policy (BIOS default) --->
< > Support for PCI Hotplug --->
< > PCCard (PCMCIA/CardBus) support --->
[ ] Hisilicon PCI Express support --->

--- Hisilicon PCI Express support
[ ] PCI Express configs --->
```





```
[*] PCI Express controller 0 sel  
(0x10000000) Total memory size of PCI Express EP devices  
(0x800000) Sum of configuration header size mapped for all PCIe EP devices  
[*] limit pcie max read request size
```

步骤 4. 设置完成后，保存并退出。

步骤 5. 执行命令：make ARCH=arm CROSS\_COMPILE=arm-hisiv300-linux- uImage -j 128

步骤 6. 新编译出的内核镜像存放路径为 arch/arm/boot/uImage。

---结束





# 5 Flash

## 5.1 为什么 Hi3531 NAND Flash 我配置了硬件 ECC, 但却不生效

Hi3531 的 NAND flash 驱动支持两种 ECC 识别方式

- 使用硬件 ECC 配置;
- 自动探测, 通过探测到 NAND 的型号, 自动选择最好的 ECC 纠错方式

使用硬件配置 ECC 类型, 启动时打印的信息中, 有以下特定字符: “Nand(Hardware)”

```
NAND: Special NAND ID table version 1.21
Nand ID: 0xAD 0xDC 0x10 0x95 0x54 0xAD 0xDC 0x10
Nand(Hardware): Block:128K Page:2K Ecc:1bit Chip:512M OOB:64Byte
512 MiB
```

如果是自动探测, 则打印 “Nand(Auto)”。

使用自动探测的 ECC 类型, 必须是非 NAND 启动的单板。

## 5.2 如何标记 flash 上的坏块?

默认情况下, SDK 的 NAND flash 读写函数已经内置了 flash 坏块处理策略, 用户不需要关注。以下方法, 仅仅用于用户想要强行标记 flash 某些块为坏块进行某种测试的场景。正常情况下, 不需要使用这些方法。

- u-boot-2010.06 下标记坏块方法

u-boot-2010.06 标记 NAND 坏块的命令如下:

```
nand markbad offset
```

该命令会标记 offset 位置所在的 NAND 块为坏块, 如希望标记 1M 位置的块为坏块, 命令如下:

```
nand markbad 0x100000
```



offset 最好为 NAND 的块大小的整数倍。标记坏块后可以用如下命令查看 NAND 坏块：

```
nand bad
```

- 内核下标记坏块方法

相关代码如下：

```
#define MEMSETBADBLOCK    _IOW('M', 12, __kernel_loff_t)
int fd;
unsigned long long offset;
fd = open("/dev/mtd1", O_RDWR);
offset = 0x100000;
if (ioctl(fd, MEMSETBADBLOCK, &offset))
{
    printf("Mark bad block 0x%llx failed!\n", offset);
}
```

该段程序会标记 open 的 mtd 分区的 offset 偏移位置所在的 NAND 块为坏块；

如希望标记 mtd1 分区偏移 1M 位置的块为坏块，需要在 open 函数时指定 mtd1（为对应分区的字符设备节点），

设置 offset 为 0x10000，如上段代码所示；

注意：u-boot-2010.06 下的 offset 是相对与整个 NAND 的偏移位置，内核下的 offset 是相对 open 的对应分区的偏移位置。

## 5.3 Hi3531 的 NAND FLASH 控制器支持哪几种纠错方式

Hi3531 的 NAND Flash 控制器支持情况见下表

页大小	ECC 纠错	是否支持
2 KB	1 bit	支持
	4 bytes	支持(ubi/ubifs)
	8 bytes	不支持
	24 bits/1 KB	不支持
4 KB	1 bit	支持
	4 bytes	支持
	8 bytes	不支持
	24 bits/1 KB	支持
8 KB	1 bit	不支持
	4 bytes	不支持



页大小	ECC 纠错	是否支持
	8 bytes	不支持
	24 bits/1 KB	支持

## 说明

- 4Bytes 纠错可以当成 4bit 纠错使用，不会有问題
- 1bit、4Bytes、8Bytes 表示每 512 字节(不一定就恰好是 512 字节，可能会多几个字节)最多可以纠正 1bit、4Bytes、8Bytes 错误。24bit1K 表示每 1K 的数据中，最大可以纠正 24bit 错误。
- 2k4bytes 只支持 ubi/ubifs 文件系统。
- **不支持**是因为当前没有这种 NAND 器件。比如：目前没有 2K-24bit1K 的这种器件；

## 5.4 使用大容量 NAND 应该注意的地方

大容量 NAND 是指容量 >4G 的 NAND 器件。4G 是一个 32 位无符号整型所能表示的最大极限，超过这个界限，32 位变量会发生回绕。当前测试过的最大容量为 8G。使用 >4G 的器件时，应该注意文件系统（UBI/YAFFS2）分区大小不能超过 4G，否则文件系统可能回绕。

## 5.5 NAND ECC/PAGESIZE 逻辑/硬件/软件的配置原理：

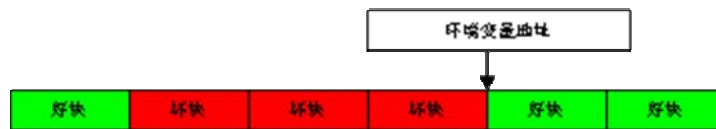
NAND ECC/PAGESIZE 配置从硬件到软件的顺序如下：

- 逻辑上电时，逻辑从硬件上下拉电阻把 ECC/PAGESIZE 读到 NAND 寄存器 NFC\_CON 中；
- 如果是 NAND 启动，逻辑按 NFC\_CON 的配置值，去读取 NAND 上的内容；
- 软件初始化 DDR，初始化所有已设置寄存器，如果已配置 NFC\_CON，改写 NFC\_CON 寄存器；
- 软件启动后，驱动读 NAND 寄存器 NFC\_CON，得到 ECC/PAGESIZE；

## 5.6 为什么在 NAND 上, u-boot 保存环境变量后, 系统无法启动.

有些用户在 NAND 上烧写完 u-boot 后，系统能正常启动，但保存环境变量后，系统无法启动。

原因如下图所示，u-boot 占三个好块，NAND 起始位置有三个坏块。保存环境变量后，fastboot 内容被擦除，系统无法启动。这个时候，要把环境变量地址往后挪。



## 5.7 为什么烧写 NAND 时, 不能合并成一个文件?

参考 如下 “5.8 为什么 NAND 上文件系统读出来后, 不能再写到 NAND 上使用。”

## 5.8 为什么 NAND 上文件系统读出来后, 不能再写到 NAND 上使用。

NAND 有一个特性, 擦除完成后, 只能写一次, 如果要再次写入数据, 需求重新擦除;  
NAND 上的数据, 擦除完成后, 为 0xFF, 如果用户第 1 次写入 0xFF, 第 2 次写入其它数据, 就可能会出错, 因为擦除完后写入了多次。

出现错误的操作如下图所示:

- 为空 – 表示擦除完成后 NAND 状态.
- 非空 – 表示 NAND 擦除后, 有发生过写操作.

擦除完成后 NAND 上状态:

0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 第1页为空(全0xFF)	0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 第2页为空(全0xFF)	0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 第3页为空(全0xFF)
--	--	--

合并文件后的文件系统, 或者从 NAND 上读出来文件系统, 数据类似以下这种形式:

0x110x220xFF0xFF 0x330x440xFF0xFF 0xFF0x550xFF 0xFF 0xFF0xaa0xFF 0xFF 第1页数据	0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 0xFF0xFF 0xFF 0xFF 第2页数据(全0xFF)	0xFF0x110xFF 0xFF 0xFF0xFF 0x330xFF 0xFF0xFF 0x440xFF 0xFF0xFF 0xaa0xFF 第3页数据
---	--	---

把第 2 步的数据写到 NAND 器件上后, NAND 上的数据状态



0x110x220xFF0xFF 0x330x440xFF0xFF 0xFF0x550xFF0xFF 0xFF0xaa0xFF0xFF 第1页非空	0xFF0xFF0xFF0xFF 0xFF0xFF0xFF0xFF 0xFF0xFF0xFF0xFF 0xFF0xFF0xFF0xFF 第2页非空(全0xFF)	0xFF0x110xFF0xFF 0xFF0xFF0x330xFF 0xFF0xFF0x440xFF 0xFF0xFF0xaa0xFF 第3页非空
---	--	---

第3步以后, 文件系统看到的数据:

0x110x220xFF0xFF 0x330x440xFF0xFF 0xFF0x550xFF0xFF 0xFF0xaa0xFF0xFF 第1页非空	0xFF0xFF0xFF0xFF 0xFF0xFF0xFF0xFF 0xFF0xFF0xFF0xFF 0xFF0xFF0xFF0xFF 第2页(全0xFF, 为空)	0xFF0x110xFF0xFF 0xFF0xFF0x330xFF 0xFF0xFF0x440xFF 0xFF0xFF0xaa0xFF 第3页非空
---	--	---

对于非只读的文件系统, 即使用户没有进行写 NAND 操作, 文件系统自己也会有一些写操作(比如定时垃圾回收, 数据同步).

文件系统, 无法区别第2页是因为(1.擦除完成后 NAND 上状态), 还是(3.第2步的数据写到 NAND 器件上后, NAND 上的数据状态), 文件系统认为第2页为空, 如果对第2页进行第2次写操作(第1次写了 0xFF), 发生错误。

#### 说明

- Cramfs, squashfs 等只读文件系统, 不存以上问题;
- 内核, uboot 等数据, 不会被再次改写, 不会存在以上问题;

## 5.9 怎样把 SPI flash 由 4 线模式修改为 2 线模式?

在 uboot 的~/drivers/mtd/spi/hisfc350/hisfc350\_spi\_ids.c 中, 找到对应器件的 ID 表, 举例如下, 将 QUAD (四线) 能力关闭掉 (读写都关闭), 驱动检测器件无 QUAD 能力, 则不会使能 4 线能力, 工作在最高的 2 线能力下, 内核下的修改和 uboot 下类似, 这里不再赘述。

```
{
    "MX25L25635E/735E/635F",
    {0xc2, 0x20, 0x19}, 3, _32M, _64K, 4,
    {
        &READ_STD(0, INFINITE, 40/*50*/),
        &READ_FAST(1, INFINITE, 104),
        &READ_DUAL(2, INFINITE, 104),
        &READ_DUAL_ADDR(1, INFINITE, 84),
        // &READ_QUAD_ADDR(3, INFINITE, 75),
        0
    },
    {
```

```

        &WRITE_STD(0, 256, 75),
        0
    },
    {
        &ERASE_SECTOR_64K(0, _64K, 80),
        0
    },
    &spi_driver_mx25l25635e,
},

```

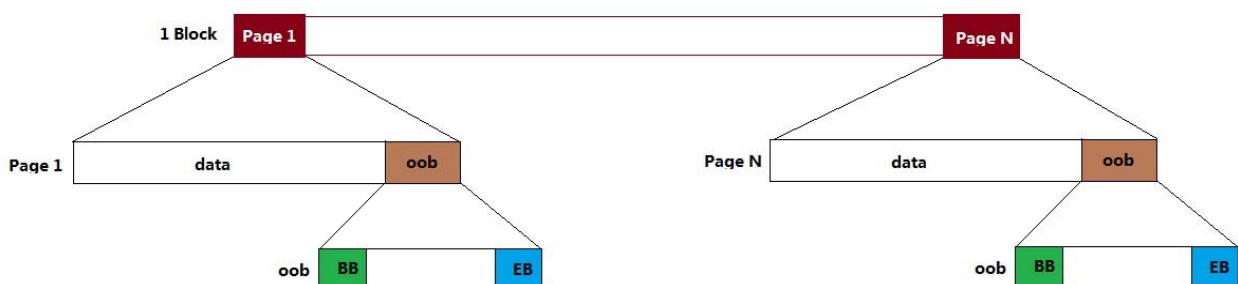
## 5.10 如何正确使用 mtd-utils 的 nandwrite 裸写工具

使用 mtd-utils 的 nandwrite 裸写工具时，如果写的是 u-boot.bin 镜像，且镜像大小大于 Nand Flash 的一个块的大小，一定要保证 u-boot.bin 镜像的数据按块对齐填充。否则，写进去的 u-boot.bin 镜像无法正常启动。

具体原因如下：

由于个别 Nand Flash 出厂时坏块标记位（BB，Bad Block）被标记为非全 0 的数，例如 0xFE，在判断坏块的时候，容易被 FMC 控制器利用 ECC 纠错为 0xFF（因为全 0xFF 在控制器的 ECC 算法上是合法可纠错的），故在每一个 page 的 OOB 信息的最后两个 byte 设置空块标记（EB，Empty Block）位。如图 5-1 所示，在 u-boot 启动时，逻辑上把 block 视为好块的前提条件是：block 的第一个 page 1 和最后一个 page N 的 BB=0xFF, EB=0x00。

图5-1 Nand Flash 块结构图



nandwrite 是根据镜像文件大小逐页写数据的，并且写 page 时，会自动将当页的 EB 位配置为 0x00。当写的最后一个 page 不是所在 block 的最后一个 page 时，由于所在 block 最后一个 page 的 EB=0xFF，逻辑就是视这个 block 为空块不会去读数据，导致 uboot 启动失败。

值得注意的是，由于 Nand Flash 器件出厂时保证第一个块为好块，故逻辑不会去判断第一个块的情况，所以当 u-boot.bin 镜像大小小于一个块大小时，uboot 是可以正常启动的。

此外，一旦 uboot 正常启动，软件上我们不会再去判断 EB 位，这也是为什么使用 nandwrite 写内核镜像和文件系统镜像时可以不考虑镜像大小是否块对齐。



# 6 文件系统类

---

## 6.1 NAND 挂载 cramfs 文件系统注意事项

在 nand 上使用 cramfs 时，必须 mount 到 romblock,不能 mount 到 mtblock.

cramfs 不是专门为 nand 器件设计，cramfs 文件系统本身不能跳坏块。内核有个块设备叫 romblock，这个设备实现了跳坏块功能。



# 7 快速启动优化

- 配置 boot 下的环境变量 bootdelay 为 0
  - 方法：在 boot 下的命令行中输入：setenv bootdelay 0;saveenv
  - 说明：为了方便进入 boot 命令行，boot 下默认设置 bootdelay 为 1，配置 bootdelay 为 0 可以加快 fastplay 启动时间约 1S(boot 中已修改代码配置默认值为 0)
- 配置 boot 阶段不做内核校验
  - 方法：在 boot 下的命令行中输入：setenv verify n;saveenv
  - 说明：如果内核出错，在 boot 阶段做不做校验，系统基本都会挂死，因此设置不做校验理论上不会产生影响，该操作可加快启动时间约 1S(boot 中已修改代码配置默认值为不做校验)
- 取消 kernel 阶段的 BogoMIPS 计算
  - 方法：配置 bootargs，在 bootargs 中加上 lpj=5996544
  - 说明：BogoMIPS 用于衡量 cpu 运行速度，设置 lpj=5996544 可以取消该计算过程，此举可加快启动时间约 0.2s
- 设置 bootcmd 如下：setenv bootcmd 'nand read 0x807ffc0 0x100000 0x400000;bootm 0x807ffc0'
  - 说明：设置成上述 bootcmd 后，boot 直接将内核镜像从 flash 读到 0x807ffc0，然后从 0x807ffc0 启动。
  - 相反如果按照默认 bootcmd 配置：nand read 0x82000000 0x100000 0x400000;bootm 0x82000000，则 boot 阶段先将 kernel 从 flash 读到 0x82000000 地址，再将镜像从 0x82000000 拷贝至 0x807ffc0，然后从 0x807ffc0 启动。





# 8 Kernel

## 8.1 pid 和 tgid 区别

task\_struct 结构体包含两个字段，pid 和 tgid。

简单理解: pid 为该 task\_struct 唯一的 id 号。tgid 是该 task 所在线程组的组长 id 号, thread group id, 若它没组长, 则它的 tgid = pid。tgid 的存在为了兼容 posix 标准。

getpid() return tgid。

In normal processes,the TGID is equal to the PID.

With threads, the TGID is the same for all threads in a thread group. This enables the threads to call getpid() and get the same PID.

In fact, the POSIX 1003.1c standard states that all threads of a multithreaded application must have the same PID.

## 8.2 普通进程和实时进程优先级

进程优先级范围:	0---139
实时进程优先级:	0---99
普通进程优先级:	100—139
nice 对应普通进程:	-20——19 <--> 100——139
内核默认进程优先级为:	120—对应 nice=0

## 8.3 如何设置 dmesg buf 的大小?

在 menuconfig 菜单下, 选择以下选项:

General setup ---> Kernel log buffer size



CONFIG\_LOG\_BUF\_SHIFT(当前内核默认18 =256K)

## 8.4 在/proc/meminfo 显示的 MemTotal 为什么不等于 cmdline 里的配置 mem=xxxM?

因为内核在启动时 reserved 了一些内存，如内核的代码段，pmem 里 reserved 的内存。我们看到的 MemTotal 的大小已经减去了内核早期保留的内存块。

相关代码见"arch/arm/mm/init.c"

```
totalram_pages += free_all_bootmem();
```

```
totalram_pages += totalhigh_pages;
```

也就是 memblock 里的 reserved 和 bootmem 里已分配出去，内核后面都看不到了。

totalram\_pages 是 zone/buddy allocator 接手内存管理后看到的内存总量。

详细的 meminfo 里各 mem 的描述请参考"./Documentation/filesystems/proc.txt"里的说明

```
MemTotal: Total usable ram (i.e. physical ram minus a few reserved
          bits and the kernel binary code)
```

```
MemFree: The sum of LowFree+HighFree
```

```
Buffers: Relatively temporary storage for raw disk blocks
          shouldn't get tremendously large (20MB or so)
```

```
Cached: in-memory cache for files read from the disk (the
         pagecache). Doesn't include SwapCached
```

```
.....
```

## 8.5 在 Linux 系统中怎样判断栈是否溢出?

linux 系统中共使用了 4 种堆栈:

- 第 1 种是系统引导初始化时临时使用的堆栈;
- 第 2 种是进入保护模式之后提供内核程序初始化使用的堆栈，位于内核代码地址空间固定位置处。该堆栈也是后来任务 0 使用的用户态堆栈;
- 第 3 种是每个任务通过系统调用，执行内核程序时使用的堆栈，我们称之为任务的内核态堆栈。每个任务都有自己独立的内核态堆栈;
- 第 4 种是任务在用户态执行的堆栈，位于任务（进程）逻辑地址空间近末端处。

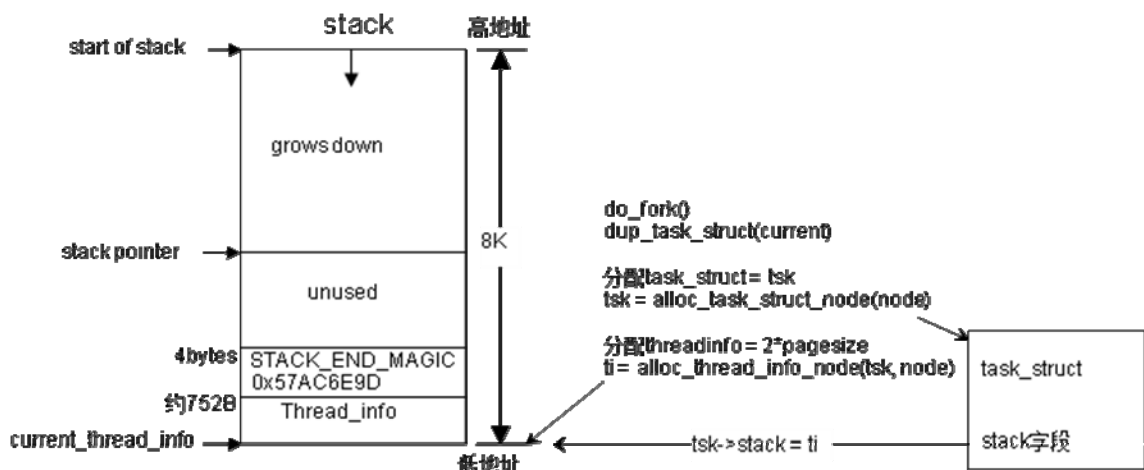
使用多个栈或在不同情况下使用不同栈的主要原因有两个。首先是由于从实模式进入保护模式，使得 CPU 对内存寻址访问方式发生了变化，因此需要重新调整设置栈区域。

另外，为了解决不同 CPU 特权级共享使用堆栈带来的保护问题，执行 0 级的内核代码和执行 3 级的用户代码需要使用不同的栈。当一个任务进入内核态运行时，就会使用其 TSS 段中给出的特权级 0 的堆栈指针 tss.ss0、tss.esp0，即内核栈。原用户栈指针会被保存在内核栈中。而当从内核态返回用户态时，就会恢复使用用户态的堆栈。



当一个进程运行在内核态时（比如通过系统调用），它就将开始使用它自己的内核栈，如果内核栈大小为 8K，那么此时的触发的中断处理也将使用这个栈，如果内核栈大小为 4K，那么此时的触发的中断处理则将使用单独的内核栈。

由于内核栈大小固定且比较小，很容易出现内核栈溢出的情况，所以不能在内核代码里使用递归调用（除非你非常清楚它递归的层次，但仍建议将递归改为循环，因为谁也不知道将来哪一天递归的层次是否会发生变化），也不建议使用较大或大小未知的栈变量（比如动态数组）等。由于 task\_struct 和内核栈共用同一块内存区域，所以内核栈溢出最直接的后果就是把 task\_struct 结构体踩坏，在 linux 下，这个结构体是至关重要的，每一个进程都是由这个 task\_struct 数据结构来定义，它也就是我们通常所说的 PCB，它是用来对进程进行控制的唯一手段，也是最有效的手段；到了 kernel 2.6.x 之后，和内核栈共用同一块内存区域的不再是 task\_struct，而是结构体 thread\_info，不过由于 thread\_info 结构体的第一个字段就是 task\_struct 指针，所以内核栈溢出的话同样会损坏 task\_struct，因为此时 task 指针指向一个不可预知的地址，相应的 task\_struct 结构体各个字段数据当然也就都是垃圾数据了。看如下图的堆栈结构更直观：



do\_fork()时，内核使用 slab 分配了 task\_struct 结构体 tsk；

threadinfo 不需要单独分配，因为 threadinfo 在该线程的栈的空间里（最底下）。也就是分配了栈空间，就有 threadinfo 的位置了。

ti = alloc\_pages(GFP\_KERNEL, 1); //order=1 代表 8K 的空间，ti 代表 threadinfo 上面一句相对于分配了 8K 的栈的空间，同时 threadinfo 就占据在这个空间的开始部分。threadinfo 的大小估算了约有 752B 大小(不精确)，然后 do\_fork 后面在每个 threadinfo 后面放了个 MAGIC 数值，便于观察是否栈溢出覆盖掉 threadinfo。

task\_struct 有个 stack 字段(void \* stack)，它指向自己的 stack 的起始地址。

tsk.stack = ti; //也就是通过 task\_struct 的 stack 就找到了 threadinfo 的首地址(指针)。同时 threadinfo 里有个 task 字段(struct task\_struct \*task)，它指向自己的这个 task\_struct。也就是它们两个互相交叉引用！

那么我们则么判断栈是否溢出呢？内核判断栈的结束地方如下：

```
static inline unsigned long *end_of_stack(struct task_struct *p)
{
    return (unsigned long *) (task_thread_info(p) + 1);
}
```



}也就是 thread\_info 的上面就是栈的结束，上面指的是地址向上。这里的加 1 实际上偏移了一个 struct threadinfo 大小。

内核在栈底(end of stack)保存了一个 STACK\_END\_MAGIC 数值。

我们获取了栈底的这个地址，读取出的数值是否是这个 STACK\_END\_MAGIC 数值，就可以很好的判断是否栈溢出了！具体的修改方法举例如下：

在 kernel/softirq.c 中的 asmlinkage void \_\_do\_softirq(void)函数中加下面红色字体的代码

```
.....
do {
    if (pending & 1) {
        int printkflg = 0;
        static int ncount = 0;
        unsigned long *sp_magic = NULL;
        unsigned int tmp=current_thread_info();

        unsigned int vec_nr = h - softirq_vec;
        int prev_count = preempt_count();

        kstat_incr_softirqs_this_cpu(vec_nr);

        sp_magic = (unsigned long *)((struct thread_info *)tmp +1);
        if(current_thread_info()->task->pid != 0)
        {
            register unsigned long sp asm ("sp");
            unsigned int tmp2=sp;
            static int minstack=0xFFFFFFFF;

            if((*sp_magic != 0x57AC6E9D)) ncount++;

            if( (tmp2 - tmp < minstack) || ((*sp_magic != 0x57AC6E9D) &&
(ncount < 2)))
            {
                minstack = tmp2 - tmp;
                printkflg = 1;
                printk("-----sp_magic = 0x%x, 0x%x,
0x%x\n", *sp_magic, 0x57AC6E9D, sp_magic);
                printk("-----Dump At %s %d,
StackS %p(id=%d)-- %p ?= %08x -----\n",
                __FUNCTION__, __LINE__, current_thread_info(),
current_thread_info()->task->pid, &tmp, sp);
            }
        }
    }
}
```



```
trace_softirq_entry(vec_nr);
h->action(h);
trace_softirq_exit(vec_nr);

if(current_thread_info()->task->pid != 0)
{
    if( (printkflg) || ((*sp_magic != 0x57AC6E9D) && (ncount < 2)))
    {
        register unsigned long sp asm ("sp");
        unsigned int tmp2=sp;

        printkflg = 0;
        printk("-----sp_magic = 0x%x, 0x%x,
0x%x\n", *sp_magic, 0x57AC6E9D, sp_magic);
        printk("-----Dump At %s %d,
StackS %p(id=%d)-- %p ?= %08x -----\n",
            __FUNCTION__, __LINE__, current_thread_info(),
            current_thread_info()->task->pid, &tmp2, sp);
        dump_stack();
    }
}

.....
```

## 8.6 遇到“段错误”(segmentation fault)怎样生成 core dump 文件来进行问题分析？

可以在 shell 下通过设置如下命令来生成 core dump 文件：

```
ulimit -S -c unlimited > /dev/null 2>&1
```

但是要能在 core dump 文件中能正常的显示出错误信息，则编译可执行文件时，还必须用 **-g** 调试选项来编译生成文件。

## 8.7 Cache 使用注意事项

Hi3531 采用 ARM Cortex-A9 作为 cpu，包括 L1、L2 两级 cache。ARM 架构下的 Cache 有一些特性需要用户在编码时注意，从而避免 L1、L2 以及 memory 之间的数据一致性问题。下面针对不同场景，分别加以说明：



## DMA 同步处理：

这种场景是处理器和外设会直接操作物理内存。举例来说：网络控制器会直接将网络数据包搬运到物理内存中，它是看不到 Cache 的；而处理器也会对这个网络数据包进行操作，但处理器是看得见 Cache 的。如果存在 Cache 命中的情况，处理器就会直接操作 Cache 中的数据，而不会访问物理地址，这时就会存在数据不一致的情况。

因此这时就需要调用 Cache 的操作接口做数据的同步处理。

另外不管是 L1 还是 L2 Cache，执行一次操作的最小单位是一个 cache line（32Byte）。因此用户必须保证分配物理内存时，起始地址 32Byte 对齐，且分配长度是 32Byte 的整数倍。否则就会出现物理内存区域非 cache line 对齐时，附近一些内存区域被补齐到 cache line 范围内，如果这部分区域正好也被使用，就有可能出现数据一致性问题。

## 处理器侧的同步处理：

这种场景是只有处理器可以看到需要进行数据同步的地址，其他模块看不到，也与外设没有什么关联。可以设想这样一个例子：同一片物理内存地址以带 Cache 的方式映射到了两个不同的虚拟地址空间中，这两个虚拟地址空间分别属于两个不同的进程，这两个进程要通过这片内存通讯，恰好进程 1 中这块内存的虚拟地址是 Cache 命中的，而进程 2 中这块内存的虚拟地址没有 Cache 命中，这时的情况就需要进行“处理器侧同步处理”。

这种场景使用内核标准的 L1 Cache 的操作接口即可。

## 8.8 客户跑一个很简单的程序，top 信息的 loadaverage 值比较大，达到 2.95，而 cpu 的占用率比较低。根据之前的理解：

loadaverage 的值是衡量 cpu 等待完成的任务排队的情况。请解释该值过高的原因，是否影响业务？

top 命令中 load average 显示的是最近 1 分钟、5 分钟和 15 分钟的系统平均负载。系统平均负载表示：

系统平均负载被定义为在特定时间间隔内运行队列中(在 CPU 上运行或者等待运行多少进程)的平均进程数。如果一个进程满足以下条件则其就会位于运行队列中：

- 它没有在等待 I/O 操作的结果
- 它没有主动进入等待状态(也就是没有调用 'wait' )
- 没有被停止(例如：等待终止)

Update：在 Linux 中，进程分为三种状态，一种是阻塞的进程 blocked process，一种是可运行的进程 runnable process，另外就是正在运行的进程 running process。当进程阻塞时，进程会等待 I/O 设备的数据或者系统调用。

进程可运行状态时，它处在一个运行队列 run queue 中，与其他可运行进程争夺 CPU 时间。系统的 load 是指正在运行 running one 和准备好运行 runnable one 的进程的总数。比如现在系统有 2 个正在运行的进程，3 个可运行进程，那么系统的 load 就是 5。load average 就是一定时间内的 load 数量。

例如：

[?\[Copy to clipboard\]](#)[View Code](#) BASH

```

1          # uptime
2
3          7:51pm up 2 days, 5:43, 2 users, load average: 8.13, 5.90, 4.94

```

命令输出的最后内容表示在过去的 1、5、15 分钟内运行队列中的平均进程数量。

一般来说只要每个 CPU 的当前活动进程数不大于 3 那么系统的性能就是良好的，如果每个 CPU 的任务数大于 5，那么就表示这台机器的性能有严重问题。对于上面的例子来说，假设系统有两个 CPU，那么其每个 CPU 的当前任务数为： $8.13/2=4.065$ 。这表示该系统的性能是可以接受的。

从以上的解释可以看出我们每个 CPU 的当前任务数为  $2.95/2=1.475$ ，因此我认为是很正常的！

## 8.9 当内核的内存配置为 512MB 或以上时，为什么会出现如下错误 “vmap allocation for size 528384 failed: use vmalloc=<size> to increase size.” ？

这是由于内核只占据了 3G~4G 的虚拟空间，也就是说只有 1GB，除了异常向量等使用的空间，实际可用的只有 992MB 左右，而业务分配的 malloc 区域比较大，超过了 512MB，因此当 mem 配置为 512MB 时，malloc 区域会不够用，就会报如上的错误。可以通过如下方法解决此问题，就是把内核所占据的空间由 1GB 调整到 2GB，具体调整方法如下：

修改内核态和用户态地址为 2: 2 模式：

在 menuconfig 中直接修改选项 **Memory split** 即可：

```

Kernel Features --->
Memory split (2G/2G user/kernel split) --->
| |      ( ) 3G/1G user/kernel split
| |      | |      (X) 2G/2G user/kernel split
| |      | |      ( ) 1G/3G user/kernel

```

扩大 vmalloc 区域为 0x7e000000 (1912MB)：

修改 arch/arm/mach-godnet/include/mach/vmalloc.h 即可。

## 8.10 内核能否管理两块不连续的内存？

内核已经支持管理两块不连续的内存，只需要在 bootargs 里增加如下配置即可，如 “**mem=256M@0x80000000 mem=256M@0xCFF00000**”，这里 256M 表示每一块内存的大小，0x80000000、0xCFF00000 表示每一块内存的起始地址，但是这里还有一点，因为从 0x80000000 到 0xC0000000 之前的空洞太大，无法映射出 0xC0000000 之后的空间，





因此还要像 8.9 节中提到的方法把内核和用户空间的占空比修改为 2:2（目前是 1:3）才能使用。

另外由于这样修改多 mmz 也有影响，因此 mmz 的代码也要修改，修改方法如下：

在~/drv/mmz/目录下的 media-mem.c 中，将\_check\_mmz 函数中之前的判断条件：

```
if(!((new_start>=__pa(high_memory)) || (new_start<PHYS_OFFSET &&
new_end<=PHYS_OFFSET))) {
    printk(KERN_ERR "ERROR: Conflict MMZ:\n");
    printk(KERN_ERR HIL_MMZ_FMT_S "\n", hil_mmz_fmt_arg(zone));
    printk(KERN_ERR "MMZ conflict to kernel memory (0x%08lX,
0x%08lX)\n", (long unsigned int)PHYS_OFFSET, __pa(high_memory)-1);
```

修改为如下：

```
if (pfn_valid(new_start>>PAGE_SHIFT) || pfn_valid(new_end >> PAGE_SHIFT))
{
    printk(KERN_ERR "ERROR: Conflict MMZ:\n");
    printk(KERN_ERR HIL_MMZ_FMT_S "\n", hil_mmz_fmt_arg(zone));
}
```

## 8.11 在使用 NPTL 的工具链之后，用 top 怎样查看各个线程的 CPU 占有率？

具体方法如下：

执行 top 命令，开始显示进程后，按 shift 键和 h（也就是大写的 H）就能显示线程了。

如果还想用 prctl 修改线程名后，top 中显示的线程名即为修改后的线程名，则可以用如下的 top：



top

## 8.12 为什么我们的 glibc 工具链中的 backtrace 工具无法打印或保存、定位程序崩溃的堆栈信息？

这是因为我们工具链默认没有打开 backtrace 功能，因此在编译应用程序时，需要带上一个参数-funwind-tables，才能使用 backtrace，否则 backtrace 返回总是 0。

例如我们要编译调试程序 test.c，编译命令如下：





```
arm-hisiv200-linux-gcc -g -funwind-tables -rdynamic test.c
```

## 8.13 如何实现一个 1ms 的定时器？

在我们的芯片中一般有多组定时器，以 Hi3518 为例，就有两组 Dual-Timer，分别为 Dual-Timer0、Dual-Timer1，其中 Dual-Timer0 被操作系统用作系统时钟。所以我们以 Dual-Timer1 的 timer1 为例来说明如何实现 1ms 的定时器（选择的时钟源为 3MHz，定时为 1ms）。

- 步骤 1. 选择 timer1 的时钟源，可以选择系统总线时钟或晶振时钟，这里选择外部晶振的 3MHz 时钟。所以我们需要把系统控制器的 0x20050000 的 19、20bit 设置成 0。
- 步骤 2. 向 TIMERx\_LOAD 写入初值。
- 步骤 3. 由于我们选择的是 3MHz 的时钟源，要想 1ms 产生一次时钟中断的话，需要向该寄存器写入的值为  $3000000 / 1000$  也就是 3000。
- 步骤 4. 向 TIMERx\_VALUE 写入初值。
- 步骤 5. 该值一般和 TIMERx\_LOAD 的值保持一致，如果不一致第一次产生中断的时间并不是 1ms。
- 步骤 6. 通过 TIMERx\_CONTROL 设定 timer 的工作方式。
- 步骤 7. 通过该寄存器可以设定 timer 的计数模式（周期、单次计数）、分频因子（Hi3518 主要有 1、16、256 分频）、计数器操作模式（16/32bit 计数）、timer 是否使能、中断是否屏蔽等。
- 步骤 8. 我们希望能是周期计数（bit0 设置成 0）、32bit 模式（bit1 设置成 1）、不分频（bit[3:2] 设置成 0 即可），另外我们要处理该 timer 的中断，所以 bit5 设置成 1。还有就是计数器模式设置成周期模式，而非自由运行模式（可参考 Hi3518 720p IP Camera SoC 用户指南.pdf），bit6 需设置 1。
- 步骤 9. 注册该 timer 的中断处理程序。
- 步骤 10. 使能 timer，向 bit7 写 1。

----结束

具体的代码实现请参考如下对电机驱动的实现：



hisi\_motor.rar