

**FIBOCOM\_MODULE LINUX驱动集成指导**

文档版本：V1.0.0

更新日期：2017.03.25

适用型号

|  |  |  |
| --- | --- | --- |
| 序号 | 产品型号 | 说明 |
| 1 | FIBOCOM MODULE | ALL |

版权声明

版权所有©2017 深圳市广和通无线股份有限公司。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。

注意

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

版本记录

|  |  |  |
| --- | --- | --- |
| 文档版本 | 更新日期 | 说明 |
| V1.0.0 | 2017-03-25 | 初始版本 |

**目录**

[1 前言 6](#_Toc24132)

[1.1 说明 6](#_Toc6379)

[2 广和通公司模块设备端口形态 7](#_Toc18352)

[2.1 广和通公司模块设备端口形态列表 7](#_Toc3873)

[2.2 端口形态详细说明 7](#_Toc11715)

[3 LINUX系统使用说明 9](#_Toc25198)

[3.1 LINUX内核设备驱动架构 9](#_Toc1130)

[4 LINUX驱动集成指导 10](#_Toc18798)

[4.1 PPP驱动集成指导 10](#_Toc30895)

[4.1.1 内核编译配置项 10](#_Toc12913)

[4.1.2 详细配置步骤 10](#_Toc10056)

[4.2 ACM驱动集成指导 10](#_Toc14430)

[4.2.1 驱动源码修改 10](#_Toc29823)

[4.2.2 内核编译配置项 13](#_Toc30564)

[4.2.3 详细配置步骤 13](#_Toc30860)

[4.3 OPTION驱动集成指导 14](#_Toc2174)

[4.3.1 驱动源码修改 14](#_Toc19355)

[4.3.2 内核编译配置项 16](#_Toc25049)

[4.3.3 详细配置步骤 16](#_Toc5901)

[4.4 ECM驱动集成指导 16](#_Toc2968)

[4.4.1 内核编译配置项 16](#_Toc15798)

[4.4.2 详细配置步骤 17](#_Toc21388)

[4.5 NCM驱动集成指导 17](#_Toc15148)

[4.5.1 驱动源码修改 17](#_Toc16343)

[4.5.1.1 linux 2.6.38及以上的内核集成 17](#_Toc6750)

[4.5.1.2 linux2.6.32至38内核集成 17](#_Toc16288)

[4.5.1.3 linux2.6.26内核集成 18](#_Toc9738)

[4.5.1.4 linux2.6.22内核集成 18](#_Toc21037)

[4.5.1.5 低于linux2.6.22内核集成 19](#_Toc18897)

[4.5.2 内核编译配置项 19](#_Toc24619)

[4.5.3 详细配置步骤 20](#_Toc4344)

[4.6 编译 20](#_Toc8178)

[5 驱动加载确认 21](#_Toc22547)

[5.1.1 系统文件信息 21](#_Toc25845)

[5.1.2 内核调试信息 22](#_Toc12591)

[6 注意事项 23](#_Toc31694)

# 前言

## 说明

本文主要是针对广和通公司模块设备基于Android/Linux 系统的驱动集成开发活动进行相关的指导说明。本文档主要面向基于上述系统的产品开发商的驱动开发人员。

本文适用于：

Android4.0及以上系统

Linux2.6.22及以上内核版本

# 广和通公司模块设备端口形态

## 广和通公司模块设备端口形态列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模块型号 | 端口形态 | 网络通讯协议 | 网络端口（Net\_Dev） | AT指令端口（ttyAT） |
| L8系列模块 | 3ACM+3NCM | NCM (推荐) | wwan0 | ttyACM2 |
| PPP | ttyACM0 | ttyACM2 |
| L7系列模块 | ECM+5USB | ECM(推荐) | eth0 | ttyUSB2 |
| PPP | ttyUSB0 | ttyUSB2 |
| H3系列模块 | 7ACM | PPP | ttyACM0 | ttyACM3 |

## 端口形态详细说明

|  |  |  |
| --- | --- | --- |
| 端口形态 | 端口顺序 | 厂商ID/产品ID |
| 3ACM+3NCM | MODEM+TRACE+AT+NCM+NCM+NCM | 1519/0443 或 8087/095A |

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 所所属驱动 |
| MODME | 调制解调器端口(ttyACM0)，即可通AT亦可用于PPP交互 | (drivers/usb/class/cdc-acm.c) |
| TRACE | MA Trace端口(ttyACM1)，用于模块常规调试信息抓取 | (drivers/usb/class/cdc-acm.c) |
| AT | AT 端口(ttyACM2)，用于AT指令交互 | (drivers/usb/class/cdc-acm.c) |
| NCM | NCM网络设备端口(wwan0)，用于网络数据交互 | (drivers/net/usb/cdc\_ncm.c) |

|  |  |  |
| --- | --- | --- |
| 端口形态 | 端口顺序 | 厂商ID/产品ID |
| ECM+5USB | ECM+AT+MODEM+AT+TRACE+AMT | 2cb7/0001 |

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 所属驱动 |
| ECM | 以太网络设备端口(eth0)，用于网络数据交互 | (drivers/net/usb/cdc\_ether.c) |
| AT | AT 端口(ttyUSB0)，用于AT指令交互 | (drivers/usb/serial/option.c) |
| MODEM | 调制解调器端口(ttyUSB1)，NA | (drivers/usb/serial/option.c) |
| AT | AT 端口(ttyUSB2)，用于AT指令交互 | (drivers/usb/serial/option.c) |
| TRACE | MA Trace端口(ttyUSB3)，用于模块常规调试信息抓取 | (drivers/usb/serial/option.c) |
| AMT | NA | (drivers/usb/serial/option.c) |

|  |  |  |
| --- | --- | --- |
| 端口形态 | 端口顺序 | 厂商ID/产品ID |
| 7ACM | MODEM+TRACE+3GTRACE+AT+AT+AT+OCT-TRACE | 1519/0020 |

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 所属驱动 |
| MODEM | 调制解调器端口(ttyACM0)，即通AT亦能PPP交互 | (drivers/usb/class/cdc-acm.c) |
| TRACE | MA Trace端口(ttyACM1)，用于模块常规调试信息抓取 | (drivers/usb/class/cdc-acm.c) |
| 3G-TRACE | Artemis Trace端口(ttyACM2)，不常用 | (drivers/usb/class/cdc-acm.c) |
| AT | AT 端口(ttyACM3)，用于AT指令交互， | (drivers/usb/class/cdc-acm.c) |
| AT | AT 端口(ttyACM4)，用于AT指令交互 | (drivers/usb/class/cdc-acm.c) |
| AT | AT 端口(ttyACM5)，用于AT指令交互 | (drivers/usb/class/cdc-acm.c) |
| OCT-TRACE | OCT Trace端口(ACM6)，不常用 | (drivers/usb/class/cdc-acm.c) |

# LINUX系统使用说明

## LINUX内核设备驱动架构

广和通模块和Linux系统主要通过USB接口进行数据通信。Linux系统的Linux 内核需要根据模块设备上报的USB 设备接口加载USB 驱动，USB 驱动正确加载后，模块才能正常工作。 Linux系统的Linux 内核驱动架构，如图2-1所示：

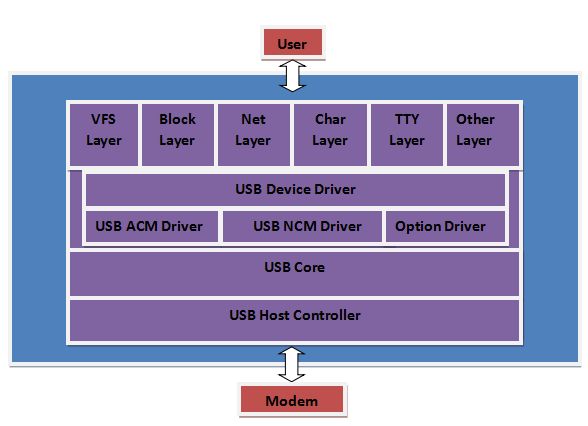


图2-1驱动架构

如图2-1所示，在Linux 系统的USB 驱动架构中与4G设备相关的驱动模块是

USB ACM、Option、ECM、NCM驱动模块。

ACM驱动：支持如modem 端口，AT 端口等；该驱动模块的代码（cdc-acm.c）已经内置于Linux 内核的源码。

OPTION驱动：USB转串口驱动，支持如modem 端口，AT 端口等；该驱动模块的代码（option.c）已经内置于Linux 内核的源码。

ECM驱动：主要用于传输网络数据。

NCM驱动：USB标准的NCM 网络设备驱动，主要用于传输网络数据。

# LINUX驱动集成指导

## PPP驱动集成指导

本节介绍PPP驱动集成步骤。

### 内核编译配置项

CONFIG\_PPP=y

CONFIG\_PPP\_MULTILINK=y

CONFIG\_PPP\_FILTER=y

CONFIG\_PPP\_ASYNC=y

CONFIG\_PPP\_SYNC\_TTY=y

CONFIG\_PPP\_DEFLATE=y

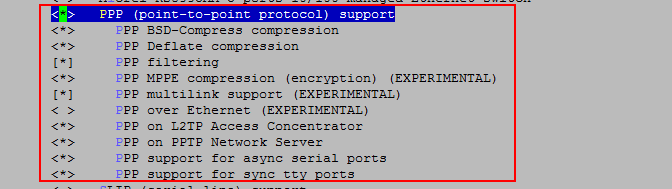
CONFIG\_PPP\_BSDCOMP=y

### 详细配置步骤

1. 打开Terminal工具，进入kernel目录（假定为“/home/ght /linux-3.0.8/”)，执行make <configuration>命令（假定使用标准make menuconfig）。

2. 按下述指引完成PPP拨号相关配置：

进入**Device Drivers** → **Network device support** 菜单后，选择红框所有选项

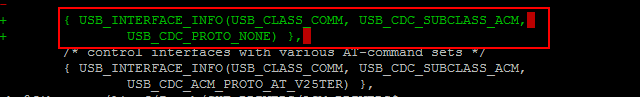


## ACM驱动集成指导

本节介绍ACM驱动集成步骤。

### 驱动源码修改

1. 驱动代码修改：在drivers/usb/class/cdc-acm.c文件的static const struct usb\_device\_id acm\_ids[ ] 数组内添加红框所示代码：

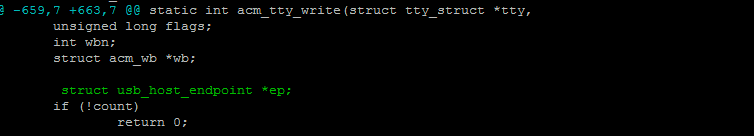


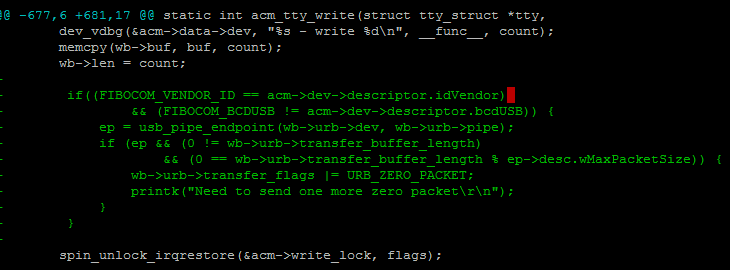
上图绿色代码如下：

{ USB\_INTERFACE\_INFO(USB\_CLASS\_COMM, USB\_CDC\_SUBCLASS\_ACM,

USB\_CDC\_PROTO\_NONE) },

2. 修改文件 drivers/usb/class/cdc-acm.c的函数 acm\_tty\_write，添加所示绿色代码：





上图绿色代码如下：

struct usb\_host\_endpoint \*ep;

        if((0x1519 == acm->dev->descriptor.idVendor) && (0x0110 != acm->dev->descriptor.bcdUSB)) {

                ep = usb\_pipe\_endpoint(wb->urb->dev, wb->urb->pipe);

        if (ep && (0 != wb->urb->transfer\_buffer\_length)

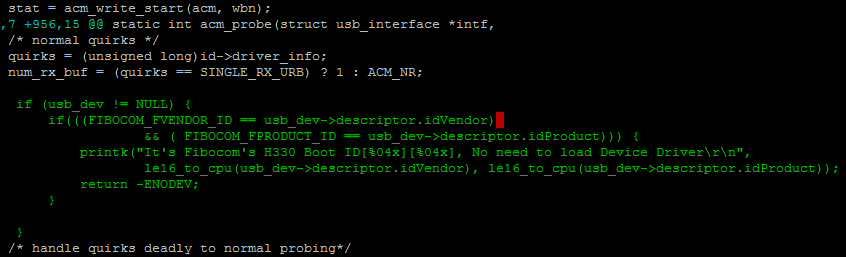
               && (0 == wb->urb->transfer\_buffer\_length % ep->desc.wMaxPacketSize)) {

                wb->urb->transfer\_flags |= URB\_ZERO\_PACKET;

                }

        }

1. 修改文件 drivers/usb/class/cdc-acm.c的函数 acm\_probe，添加所示绿色代码：



上图绿色代码如下：

if (usb\_dev != NULL) {

if(((FIBOCOM\_FVENDOR\_ID == usb\_dev->descriptor.idVendor)

&& ( FIBOCOM\_FPRODUCT\_ID == usb\_dev->descriptor.idProduct))) {

printk("It's Fibocom's H330 Boot ID[%04x][%04x], No need to load Device Driver\r\n",

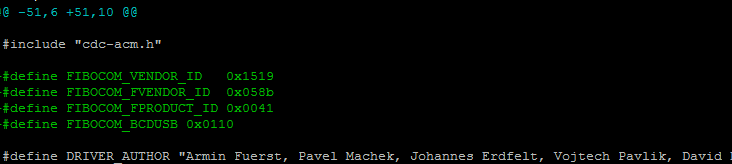
le16\_to\_cpu(usb\_dev->descriptor.idVendor), le16\_to\_cpu(usb\_dev->descriptor.idProduct));

return -ENODEV;

}

}

1. 添加上述代码使用的宏定义



上图绿色代码如下：

#define FIBOCOM\_VENDOR\_ID 0x1519

#define FIBOCOM\_FVENDOR\_ID 0x058b

#define FIBOCOM\_FPRODUCT\_ID 0x0041

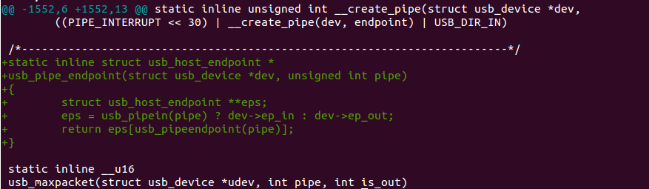
#define FIBOCOM\_BCDUSB 0x0110

5. 头文件修改

先查看include/linux/usb.h函数是否有定义：usb\_pipe\_endpoint函数，如果有就不需要修改，如果没有这按照下述步骤修改。

修改文件: include/linux/usb.h

添加函数: usb\_pipe\_endpoint



上图绿色代码：

static inline struct usb\_host\_endpoint \*

usb\_pipe\_endpoint(struct usb\_device \*dev, unsigned int pipe)

{

        struct usb\_host\_endpoint \*\*eps;

        eps = usb\_pipein(pipe) ? dev->ep\_in : dev->ep\_out;

        return eps[usb\_pipeendpoint(pipe)];

}

### 内核编译配置项

CONFIG\_USB\_ANNOUNCE\_NEW\_DEVICES=y (**若此选项存在，建议配置，若不存在，请忽略**)

CONFIG\_USB\_ACM=y

### 详细配置步骤

1. 打开Terminal工具，进入kernel目录（假定为“/home/ght /linux-3.0.8/”)，执行make <configuration>命令（假定使用标准make menuconfig）。

2. 按下述指引完成ACM驱动相关配置：

进入**Device Drivers → USB support** 菜单后选择 **USB announce new devices**及**USB Modem(CDC ACM) support**选项：



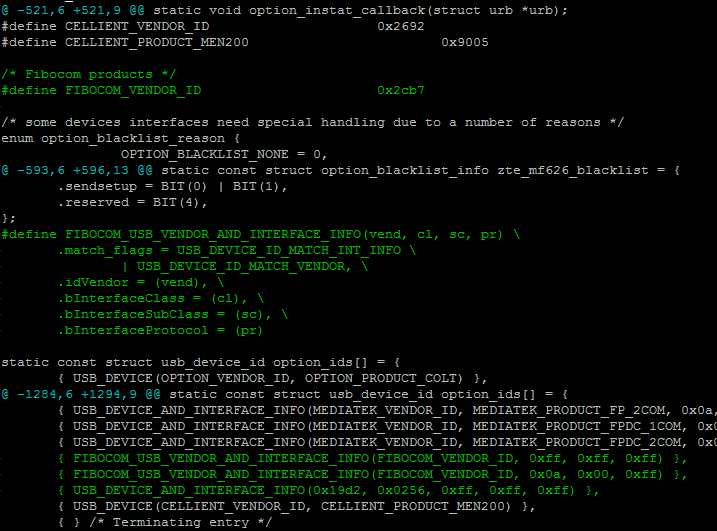


## OPTION驱动集成指导

本节介绍OPTION驱动集成步骤。

### 驱动源码修改

1. 驱动源码修改：在drivers/usb/serial/option.c文件添加绿色所示代码：



上图三部分绿色代码如下：

/\* Fibocom products \*/

#define FIBOCOM\_VENDOR\_ID 0x2cb7

#define FIBOCOM\_USB\_VENDOR\_AND\_INTERFACE\_INFO(vend, cl, sc, pr) \

.match\_flags = USB\_DEVICE\_ID\_MATCH\_INT\_INFO \

| USB\_DEVICE\_ID\_MATCH\_VENDOR, \

.idVendor = (vend), \

.bInterfaceClass = (cl), \

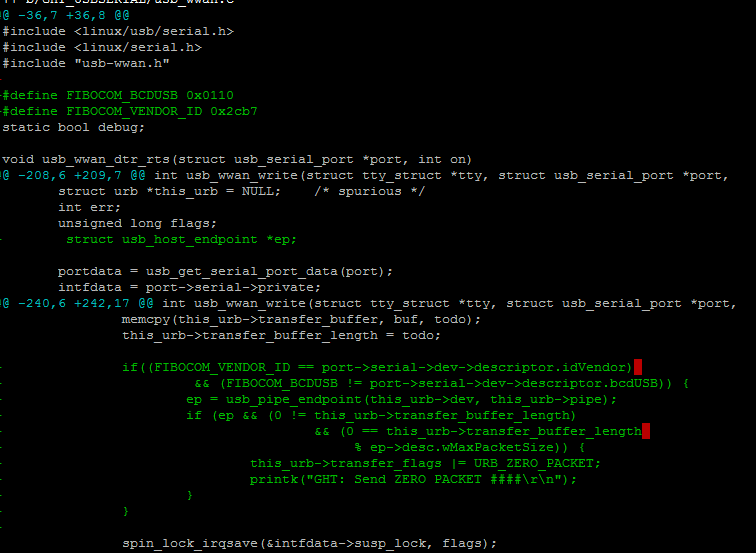
.bInterfaceSubClass = (sc), \

.bInterfaceProtocol = (pr)

{ FIBOCOM\_USB\_VENDOR\_AND\_INTERFACE\_INFO(FIBOCOM\_VENDOR\_ID, 0xff, 0xff, 0xff) },

{ FIBOCOM\_USB\_VENDOR\_AND\_INTERFACE\_INFO(FIBOCOM\_VENDOR\_ID, 0x0a, 0x00, 0xff) },

{ USB\_DEVICE\_AND\_INTERFACE\_INFO(0x19d2, 0x0256, 0xff, 0xff, 0xff) },

2. 驱动源码修改：在drivers/usb/serial/usb\_wwan.c文件添加绿色所示代码：

上图三部分绿色代码如下：

#define FIBOCOM\_BCDUSB 0x0110

#define FIBOCOM\_VENDOR\_ID 0x2cb7

struct usb\_host\_endpoint \*ep;

if((FIBOCOM\_VENDOR\_ID == port->serial->dev->descriptor.idVendor)

&& (FIBOCOM\_BCDUSB != port->serial->dev->descriptor.bcdUSB)) {

ep = usb\_pipe\_endpoint(this\_urb->dev, this\_urb->pipe);

if (ep && (0 != this\_urb->transfer\_buffer\_length)

&& (0 == this\_urb->transfer\_buffer\_length

% ep->desc.wMaxPacketSize)) {

this\_urb->transfer\_flags |= URB\_ZERO\_PACKET;

printk("GHT: Send ZERO PACKET ####\r\n");

}

}

### 内核编译配置项

CONFIG\_USB\_SERIAL=y

CONFIG\_USB\_SERIAL\_OPTION=y

### 详细配置步骤

1. 打开Terminal工具，进入kernel目录（假定为“/home/ght /linux-3.0.8/”)，执行make <configuration>命令（假定使用标准make menuconfig）。

2. 按下述指引完成ACM驱动相关配置：

进入**Device Drivers → USB support → USB Serial Converter support**菜单后选择**USB driver for GSM and CDMA modems**选项：



## ECM驱动集成指导

本节介绍ECM驱动集成步骤。

### 内核编译配置项

CONFIG\_USB\_NET\_CDCETHER=y

CONFIG\_USB\_USBNET=y

### 详细配置步骤

1. 打开Terminal工具， 进入kernel目录 （假定为 “/home/ght /linux-3.0.8/” )， 执行make <configuration>命令（假定使用标准 make menuconfig） 。  
   2. 按照下述指引完成 ECM 驱动配置:  
   进入**Device Drivers → Network device support → USB NetworkAdapters** 菜单后选  
   择 **Multi-purpose USB Networking Framework** 及**CDC Ethernet support (smart devices such as cable modems)**选项：





## NCM驱动集成指导

本节介绍NCM驱动集成步骤。

### 驱动源码修改

#### linux 2.6.38及以上的内核集成

针对linux2.6.38及以上内核版本，请直接按照[3.5.2](#_NCM驱动配置)章节添加配置即可

#### linux2.6.32至38内核集成

请参照下述步骤进行集成：

1. 登陆<https://www.kernel.org/pub/linux/kernel/v2.6/>网址，下载2.6.38以上的内核源码

2. 解压下载的内核源码，将以下文件拷贝至对应的目录

drivers/net/usb/cdc\_ncm.c

include/linux/usb/cdc.h

include/linux/atomic.h

3. 修改自身源码drivers/net/usb/目录的Makefile及Kconfig文件

将obj-$(CONFIG\_USB\_NET\_CDC\_NCM) += cdc\_ncm.o添加至Makefile的末尾。

将下述语句添加至Kconfig文件末尾。

config USB\_NET\_CDC\_NCM

tristate "CDC NCM support"

depends on USB\_USBNET

default y

4. 修改include/linux/usb/usbnet.h

将#define FLAG\_MULTI\_PACKET 0x2000 以及 int (\*manage\_power)(struct usbnet \*, int);添加至文件中即可，添加位置可参照2.6.38的usbnet.h.

#### linux2.6.26内核集成

请参照下述步骤进行集成：

1. 登陆<https://www.kernel.org/pub/linux/kernel/v2.6/>网址，下载2.6.38的内核源码

2. 解压下载的内核源码，将以下文件拷贝至对应的目录

drivers/net/usb/cdc\_ncm.c

include/linux/usb/cdc.h

include/linux/atomic.h

3. 修改drivers/net/usb/usbnet.c

删除usbnet\_start\_xmit函数的static声明：

int usbnet\_start\_xmit (struct sk\_buff \*skb, struct net\_device \*net)

并添加下述语句：

EXPORT\_SYMBOL\_GPL(usbnet\_start\_xmit);

4. 修改自身源码drivers/net/usb/目录的Makefile及Kconfig文件

将obj-$(CONFIG\_USB\_NET\_CDC\_NCM) += cdc\_ncm.o添加至Makefile的末尾。

将下述语句添加至Kconfig文件末尾。

config USB\_NET\_CDC\_NCM

tristate "CDC NCM support"

depends on USB\_USBNET

default y

5. 修改include/linux/usb/usbnet.h

将#define FLAG\_MULTI\_PACKET 0x1000 以及 int (\*manage\_power)(struct usbnet \*, int);添加至文件中即可，添加位置可参照2.6.38的usbnet.h.

再添加下述语句：

extern int usbnet\_start\_xmit(struct sk\_buff \*skb,

struct net\_device \*net);

#### linux2.6.22内核集成

请参照下述步骤进行集成：

1. 登陆<https://www.kernel.org/pub/linux/kernel/v2.6/>网址，下载2.6.38及2.6.26的内核源码

2. 解压下载的2.6.26内核源码，将以下文件拷贝到对应的目录

drivers/net/usb/usbnet.c

include/linux/usb/usbnet.h

并按照下述步骤修改usbnet.c

2.1 搜索关键词”print\_mac”,并注释掉包含printf\_mac的打印语句

2.2 注释DECLARE\_MAC\_BUF(mac);函数调用

2.3 删除usbnet\_start\_xmit函数的static声明：

int usbnet\_start\_xmit (struct sk\_buff \*skb, struct net\_device \*net)

并添加下述语句：

EXPORT\_SYMBOL\_GPL(usbnet\_start\_xmit);

3. 解压下载的2.6.38内核源码，将以下文件拷贝至对应的目录

drivers/net/usb/cdc\_ncm.c

include/linux/usb/cdc.h

include/linux/atomic.h

4. 修改自身源码drivers/net/usb/目录的Makefile及Kconfig文件

将obj-$(CONFIG\_USB\_NET\_CDC\_NCM) += cdc\_ncm.o添加至Makefile的末尾。

将下述语句添加至Kconfig文件末尾。

config USB\_NET\_CDC\_NCM

tristate "CDC NCM support"

depends on USB\_USBNET

default y

5. 修改include/linux/usb/usbnet.h

将#define FLAG\_MULTI\_PACKET 0x1000 以及 int (\*manage\_power)(struct usbnet \*, int);添加至文件中即可，添加位置可参照2.6.38的usbnet.h.

再添加下述语句：

extern int usbnet\_start\_xmit(struct sk\_buff \*skb,

struct net\_device \*net);

#### 低于linux2.6.22内核集成

针对低于2.6.22的Linux内核版本，可参照[2.2.1.4](#_linux2.6.22内核集成)节进行的集成，请联系广和通工程师一起联调。

### 内核编译配置项

CONFIG\_USB\_USBNET=y

CONFIG\_NETDEVICES=y

CONFIG\_USB\_NET\_CDC\_NCM=y

### 详细配置步骤

1. 打开Terminal工具， 进入kernel目录 （假定为 “/home/ght /linux-3.0.8/” )， 执行make <configuration>命令（假定使用标准 make menuconfig） 。  
   2. 按照下述指引完成 NCM 驱动配置:  
   进入**Device Drivers → Network device support → USB NetworkAdapters** 菜单后选  
   择 **Multi-purpose USB Networking Framework** 及**CDC NCM support**选项：





## 编译

完成驱动源码修改及内核配置后，运行make命令，编译修改后的内核。

# 驱动加载确认

经过第四节的内核驱动集成，需要确定编译各驱动是否已经成功加载。本节主要介绍如何从内核系统调试信息及文件系统进行确认驱动加载状态。

### 系统文件信息

分别执行下述指令，并查看返回结果

ls /sys/bus/usb/drivers/cdc\_acm/

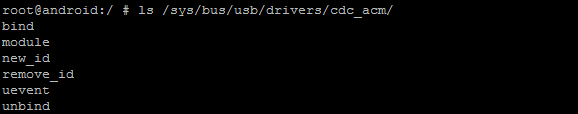
ls /sys/bus/usb/drivers/cdc\_option/

ls /sys/bus/usb/drivers/cdc\_ether/

ls /sys/bus/usb/drivers/cdc\_ncm/

以acm为例：

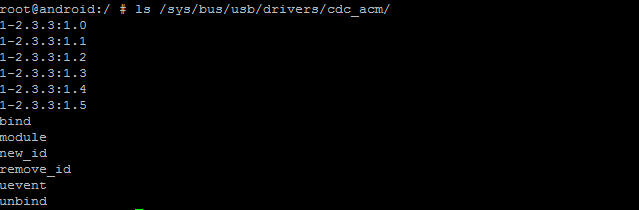
若显示下图，即acm驱动加载成功



若显示下图，即acm驱动加载失败，表示acm驱动未来加载。请参照第三节重新进行驱动集成。

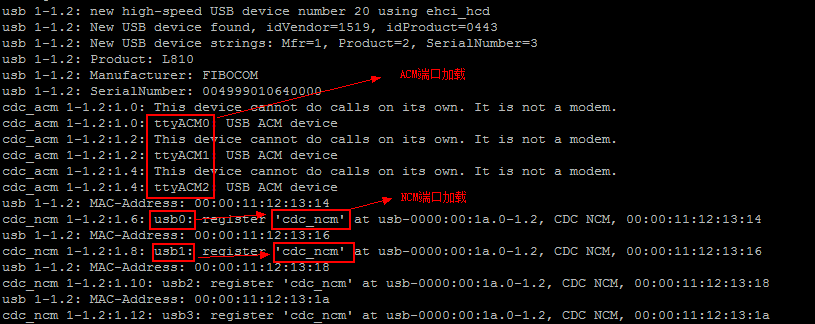


若显示下图，即有acm设备插入并且驱动加载成功，端口映射正常

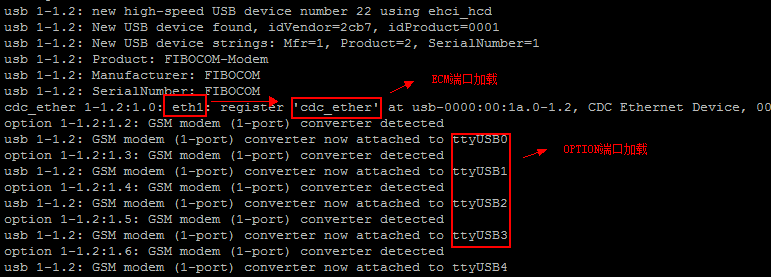


### 内核调试信息

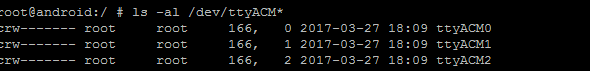
在系统开机阶段执行dmesg指令查看驱动加载情况，下图是3ACM+3NCM(L8系列模块)驱动加载示例:

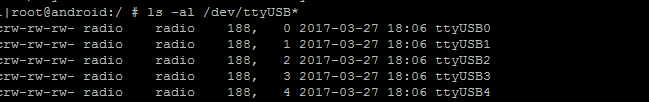


下图是ECM+5USB(L7系列模块)驱动加载示例:



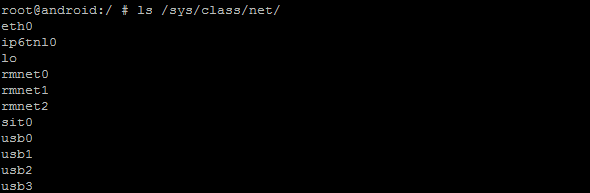
上述两图，演示了四个驱动加载的示例，其中ACM跟OPTION的通讯节点将生成于/dev/目录。执行ls -al /dev/ttyACM\* 或 ls -al /dev/ttyUSB\* 可查询到生成的节点。





而NCM及ECM的网络节点将生成于/sys/class/net/目录，执行ls /sys/class/net/可查询到生成的节点。

以NCM为例：



**注：对于NCM驱动网络设备节点usb0 usb1 usb2等，因内核版本不同其节点名称会有所不同，在linux-3.10及以上内核， NCM网络设备节点将改变成 wwan0、 wwan1、 wwan2等。**

# 注意事项

若经过驱动集成没有生成对应节点，请按照下述步骤提供调试信息，并将调试结果回馈我司FAE：

1. 反馈模块型号，将结果保存于文件
2. 在系统开机后执行dmesg指令，将输出结果导出保存成文本
3. 执行下述指令，将输出结果保存成文本，回发给

ls /sys/bus/usb/drivers/cdc\_acm/

ls /sys/bus/usb/drivers/cdc\_option/

ls /sys/bus/usb/drivers/cdc\_ether/

ls /sys/bus/usb/drivers/cdc\_ncm/

ls /sys/class/net/

ls -al /dev/ttyACM\*

ls -al /dev/ttyUSB\*