

产品名称 Product Name	
域格 9X07 模块	
产品版本 Produce Version	Total 20 pages 共 20 页
ALL	

9X07 模块 LINUX 集成用户手册

版本_V 1.60



上海域格信息技术有限公司

YUGA Technology Co., Ltd.

All rights reserved

版权所有 侵权必究



目 录

1. Linux 系统加载域格 9X07 模块 USB 串口驱动系统组件.....	4
2. Linux 系统域格 9X07 模块驱动加载.....	4
2.1 查看设备接入.....	4
2.2 过滤相关接口.....	4
3. Linux 系统下交互 AT 过程.....	6
4. Linux 系统下 pppd 拨号上网过程.....	8
5. Linux 系统域格 9X07 模块 GobiNet 拨号说明.....	11
5.1 编译 GobiNet 驱动.....	11
5.2 加载 GobiNet 驱动.....	11
5.3 GobiNet 拨号相关命令.....	11
5.4 网卡获取 IP.....	12
5.5 查看 IP 地址与网络测试.....	12
6. Linux 系统域格 9X07 模块 QMI 驱动.....	13
6.1 添加模块 VID 和 PID.....	13
6.2 添加支持 Raw IP 模式.....	13
6.3 修改内核配置.....	19
7. Linux 下 FAQ.....	20
7.1 问：内核里添加了域格模块 VID PID，为何 ls /dev/ttyU*仍查看不到端口？.....	20
7.2 问：linux 下如何通过 echo cat 手动发送 AT 命令？.....	20
7.3 问：为何在 linux 下读取不到模块的主动上报信息？.....	20



版本信息

版本号	发布日期	修改人	详细描述
1.60	2017/08/18	域格文档组	添加 gobinet 相关注意事项
1.50	2017/07/26	域格文档组	更新 QMI 驱动处理方法
1.40	2017/03/27	域格文档组	1、更新 GobiNet 拨号说明 2、新增 QMI 驱动处理方法
1.30	2016/10/12	域格文档组	更新 AT 交互命令及相关说明
1.20	2016/08/15	域格文档组	新增 GobiNet 拨号相关内容
1.10	2016/07/11	域格文档组	初始化版本



1. Linux 系统加载域格 9X07 模块 USB 串口驱动系统组件

在 Linux 系统中通常使用 USB 转串口的驱动。添加驱动需要配置 Linux 内核，方法如下：

```
cd kernel
```

```
make menuconfig
```

```
device drivers -> usb support -> usb serial converter support
```

选中如下组件：

```
USB driver for GSM and CDMA modems
```

选中后保存配置。

2. Linux 系统域格 9X07 模块驱动加载

2.1 查看设备接入

使用 lsusb 查看 usb 设备，确认发现设备。

```
test@yuge-info:~$ lsusb
Bus 001 Device 003: ID 05c6:9025 Qualcomm, Inc. Qualcomm HSUSB Device
```

如上图，模块默认的 VID、PID 为 0x05C6、0x9025。

2.2 过滤相关接口

9X07 模块仅需加载 AT 口及 modem 口，其他无关端口需过滤，以下提供三种解决方法，可根据实际情况选择处理。

1、kernel 版本支持 blacklist 的，在 option.c (路径一般为 drivers/usb/serial/option.c) 中的 option_ids 中添加 blacklist，可使驱动在加载时自动跳过 blacklist 指定的 interface。

在文件中找到 option_ids[] 的数组，添加域格模块的 VID 和 PID，VID 0x05C6 一般已存在，可根据实际情况，参照以下方法添加 VID、PID：

```
#define QUALCOMM_VENDOR_ID          0x05C6
#define YUGA_PRODUCT_9X07          0x9025
```

设置 interface 0、1、4 不加载 option 驱动，添加内容如下：



```

/***** Add for YUGA 9X07 *****/
static const struct option_blacklist_info YUGA_9X07_blacklist = {
    .reserved = BIT(0) | BIT(1) | BIT(4),
};
/*****

```

添加 blacklist 到 option_ids 数组中:

```

static const struct usb_device_id option_ids[] = {
    /***** Add for YUGA 9X07 modem *****/
    { USB_DEVICE(QUALCOMM_VENDOR_ID, YUGA_PRODUCT_9X07),
      .driver_info=(kernel_ulong_t)&YUGA_9X07_blacklist},
    /*****

```

2、对于 kernel 版本不支持在 option_ids 数组中设置 blacklist 的, 要先添加模块的 VID 和 PID:

```

#define QUALCOMM_VENDOR_ID          0x05C6
#define YUGA_PRODUCT_9X07          0x9025

static const struct usb_device_id option_ids[] = {
    /***** Add for YUGA 9X07 *****/
    { USB_DEVICE(QUALCOMM_VENDOR_ID, YUGA_PRODUCT_9X07) },
    /*****

```

在 option_probe 函数内判断当前 interface 号进行过滤:

```

/***** Add for YUGA 9X07 *****/
if(!le16_to_cpu(serial->dev->descriptor.idVendor) == QUALCOMM_VENDOR_ID &&
    !le16_to_cpu(serial->dev->descriptor.idProduct) == YUGA_PRODUCT_9X07 &&
    (serial->interface->cur_altsetting->desc.bInterfaceNumber <= 1 ||
    serial->interface->cur_altsetting->desc.bInterfaceNumber == 4)) {
    printk(KERN_INFO"Discover YUGA 9X07\n");
    return -ENODEV;
}
/*****

```

3、对于使用 usb-serial.ko 驱动的用户, 在 option.c 文件中的 option_ids[] 数组内添加模块的 VID 和 PID:

```

#define QUALCOMM_VENDOR_ID          0x05C6
#define YUGA_PRODUCT_9X07          0x9025

static const struct usb_device_id option_ids[] = {
    /***** Add for YUGA 9X07 *****/
    { USB_DEVICE(QUALCOMM_VENDOR_ID, YUGA_PRODUCT_9X07) },
    /*****

```

在 usb-serial.c 文件中的 usb_serial_probe() 函数开始增加如下判断来过滤端口:



```

/***** Add for YUGA 9X07 *****/
if(interface->cur_altsetting->desc.bInterfaceNumber <= 1 ||
    interface->cur_altsetting->desc.bInterfaceNumber == 4)
    return -EDOM;
/*****

```

编译完成后，通过查询命令：ls /dev/ttyU*，应有 ttyUSB0、ttyUSB1 两个端口，如下图：

```

test@yuge-info:~$ ls /dev/ttyU*
/dev/ttyUSB0 /dev/ttyUSB1

```

3. Linux 系统下交互 AT 过程

- 1) 将 USIM/SIM 卡正确插入应用终端，将 4G 全频天线连接到模块的射频连接器。模块开机，加载 USB 驱动，获取 USB 端口：ttyUSB0、ttyUSB1。

ttyUSB0 -> Modem

ttyUSB1 -> AT

- 2) 启动 Linux 系统串口应用程序 minicom，使用如下指令：

```
#minicom -s
```

在 minicom 菜单中选择“Serial port setup”，配置“Serial device”为/dev/ttyUSB1(模块的串口 AT(ttyUSB1)，Modem(ttyUSB0)都可以发 AT 命令)；修改完毕后退出到 minicom 菜单，选择“Save setup as dfl”保存配置后选择“exit”退出 minicom 配置。

- 3) 通过 minicom 发送 AT 指令进行系统测试

```
#minicom
```

将得到如下的返回结果：

```
Welcome to minicom 2.7
```

```
OPTIONS: I18n
```

```
Compiled on Jan  1 2014, 17:13:22.
```

```
Port /dev/ttyUSB1
```

```
Press CTRL-A Z for help on special keys
```



输入如下指令，打开回显：

ATE

如果系统工作正常，将得到如下的返回结果：

OK

输入如下指令，查询产品信息：

ATI

将得到如下信息：

Manufacturer: Yuga Co.,Ltd.

Model: CLM920_NC5

Revision: CLM920_NC5-V1 [Jul 31 2017 14:44:25]

IMEI: 868323022554940

+GCAP: +CGSM

输入如下指令，查询产品 APP 版本信息：

AT+APPVER

将得到如下信息：

APP_VERSION: Jul 31 2017 16:07:28

输入如下指令，查询 PIN 码信息：

AT+CPIN?

将得到如下信息：

+CPIN: READY

输入如下指令，查询信号：

AT+CSQ

将得到如下信号强度和误码率信息：

+CSQ: 31,99



输入如下指令，查询系统信息：

```
AT^SYSINFO
```

将得到如下信息：

```
^SYSINFO: 2,3,0,9,1 //注：注册在 SRLTE 时，该指令返回 6 位参数，详见 AT 手册
```

输入如下指令，查询 CS 域（短信、语音业务）注册状态：

```
AT+CREG?
```

将得到如下注册信息：

```
+CREG: 0,1
```

输入如下指令，查询 PS 域（数据域）注册状态：

```
AT+CGREG?
```

将得到如下注册信息：

```
+CGREG: 0,1
```

输入如下指令，查询网络运营商信息：

```
AT+COPS?
```

将得到如下运营商信息（不同运营商返回信息不同，以下以中国移动 SIM 卡为例）

```
+COPS: 0,0,"CHINA MOBILE CMCC",7
```

4. Linux 系统下 pppd 拨号上网过程

- 1) 重复模块的 USB 加载过程和 AT 交互流程。确保模块正确注册到网络，信号强度 CSQ 返回的第一个参数在 9 以上；
- 2) 确认 Linux 系统带有 pppd 和 chat 可执行程序。如果系统没有 pppd，请安装 kppp，里面带有 pppd 应用程序（推荐使用 pppd 2.4.3、pppd 2.4.5）；
- 3) 在电信 2G、3G 模式下，拨号号码可与其他制式统一，支持使用*99#拨号；

拨号上网有两种方式：

- a) 直接使用我们提供的拨号脚本 yuga.lte-pppd（默认 APN 为空，可根据需要设置），



注意给脚本执行权限；

b) 分别写 pppd 脚本和 chat 脚本：

(1) /etc/ppp/peers/lte 文件，内容如下：

```
# Usage: root>pppd call lte&
/dev/ttyUSB0
115200
crtscts
modem
debug
nodetach
usepeerdns
noipdefault
defaultroute
user card
password card
connect '/usr/sbin/chat -s -v -f /etc/ppp/lte-connect-chat'
```

(2) /etc/ppp/lte-connect-chat 文件，内容如下：

```
#/etc/ppp/lte-connect-chat
#chat script for LTE.
TIMEOUT 15
ABORT "DELAYED"
ABORT "BUSY"
ABORT "ERROR"
ABORT "NO DIALTONE"
ABORT "NO CARRIER"
"" AT
OK ATE0
```



```
OK ATDT*99***1#
```

```
CONNECT
```

两个脚本写好后，执行 `pppd call lte&`，拨号上网。

注：

- ① 域格 9X07 模块，各制式下可统一使用 `chat` 脚本中的 `ATDT*99***1#` 拨号。
- ② `pppd` 脚本中的用户名和密码是注册在电信 2G、3G 时使用的，对其他网络无影响。

4) 测试连接 Internet

测试是否连接 Internet，用如下指令：

```
# ping 115.239.210.27
```

测试是否 ping 通 baidu 的 IP 地址。

如果 IP 地址能 ping 通，而 ping 域名不通，如下指令：

```
# ping www.baidu.com
```

则需要添加 DNS(114.114.114.114)到 `/etc/resolv.conf`。

5) 断开 Internet 连接：

- a) 调用我们提供的结束脚本 `ppp-off`
- b) 使用指令：`# killall pppd`



5. Linux 系统域格 9X07 模块 GobiNet 拨号说明

域格 9X07 模块支持 GobiNet 拨号，请参照本文档第 2 部分“Linux 系统域格 9X07 模块驱动加载”加载模块驱动。

模块驱动加载好后，再按以下内容操作 GobiNet 相关内容。

5.1 编译 GobiNet 驱动

GobiNet 驱动以原代码的形式提供，由用户在自己的系统编译。

将内核源码文件解压到相关文件夹下，如 `drivers/net/usb` 目录下。在解压后形成的 GobiNet 目录下执行 `make` 命令，即可在该目录下生成 `GobiNet.ko` 文件。

注：若内核版本等于或低于 2.6.24，编译驱动时提示找不到 `usbnet.h`，可将系统中的 `usbnet.h` 拷贝到驱动路径下再编译。

5.2 加载 GobiNet 驱动

通过 `insmod` 命令加载 GobiNet 驱动：`sudo insmod GobiNet.ko`。

使用 `ifconfig` 命令查看网卡信息，如果出现 `usb0` 表示驱动加载成功，如图。

```
usb0      Link encap:Ethernet  HWaddr 0a:76:22:f0:42:42
          inet6 addr: fe80::876:22ff:fef0:4242/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:4048 (4.0 KB)
```

注：GobiNet 需依赖 `usbnet`，加载 GobiNet 驱动前，请注意加载 `usbnet`。

5.3 GobiNet 拨号相关命令

1) 拨号前请查询以下命令，确认模块成功注册到网络，具体返回结果参考 AT 手册。

`AT+CPIN?`

`AT^SYSINFO`

`AT+CSQ`

`AT+CGREG?`

2) 拨号用户名、密码及 APN 设置，请参考以下命令，具体命令格式参考 AT 手册。



- 3GPP2（即电信 2G/3G）下拨号必须设置用户名、密码，命令如下：

```
AT^NETCFG=0,32774,,,4,,,0,"CARD","CARD",1
```

//其他专网需设置用户名、密码的，请参考 AT 手册做相应设置

- APN 设置

```
AT+CGDCONT=1,"IP","APN" //如需设置 APN,将斜体 APN替换为网络相应 APN
```

//各运营商公网 APN —— 移动: cmnet; 联通: 3gnet; 电信: ctnet

- 3) 确认模块注册上网络后，通过以下命令进行拨号及查询连接情况。

```
AT$QCRMCALL=1,1,1,2,1 //3GPP 发起拨号
```

//3GPP2(即电信 2G/3G)下使用 AT\$QCRMCALL=1,1,1,1,1

//拨号成功返回以下信息

```
$QCRMCALL: 1, V4 //表示: 已连接, 协议为 IPv4
```

```
AT$QCRMCALL? //查询。连接成功后, 返回如下信息
```

```
$QCRMCALL: 1, V4 //表示: 已连接, 协议为 IPv4
```

- 4) 断开拨号命令

```
AT$QCRMCALL=0,1
```

5.4 网卡获取 IP

模块成功获取 IP 后，通过 DHCP 将 IP 赋给网卡 usb0。可参考以下命令：

```
udhcpc -i usb0
```

5.5 查看 IP 地址与网络测试

输入 ifconfig 查看 usb0 的 IP 地址，如下图：

```
usb0      Link encap:Ethernet  HWaddr 0a:76:22:f0:42:42
          inet addr:10.62.171.67  Bcast:10.62.171.71  Mask:255.255.255.248
          inet6 addr: fe80::876:22ff:fef0:4242/64  Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:2  errors:0  dropped:0  overruns:0  frame:0
          TX packets:126  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:616 (616.0 B)  TX bytes:25108 (25.1 KB)
```

之后，就可通过 ping 测试是否连接 Internet。



6. Linux 系统域格 9X07 模块 QMI 驱动

域格 9X07 模块支持在 3.3 及之后的 linux 内核版本上使用 QMI 拨号。

模块加载好 QMI WWAN 驱动后，系统中将生成相应网卡和 QMI 通道。网卡名称为 wwanX，QMI 通道为 /dev/cdc-wdmX。其中，网卡用于传输数据，QMI 通道用于传输 QMI 消息。

以下为 QMI 驱动修改方法。

6.1 添加模块 VID 和 PID

QMI WWAN 驱动原文件为"[KERNEL]/drivers/net/usb/qmi_wwan.c"，域格 9X07 模块 VID、PID 添加方法如下：

[KERNEL]/drivers/net/usb/qmi_wwan.c

```
#if 1// Add for YUGA 9X07
#ifndef QMI_FIXED_INTF_YUGA
#define QMI_FIXED_INTF_YUGA(vendor,prod,num)\
    .match_flags=USB_DEVICE_ID_MATCH_DEVICE | USB_DEVICE_ID_MATCH_INT_INFO,\
    .idVendor =vendor,\
    .idProduct =prod, \
    .bInterfaceClass =0xff,\
    .bInterfaceSubClass=0xff,\
    .bInterfaceProtocol=0xff,\
    .driver_info =(unsigned long )&qmi_wwan_force_int##num,
#endif
#endif

static const struct usb_device_id products[] = {
    {QMI_FIXED_INTF_YUGA(0x05c6, 0x9025, 4)},
```

6.2 添加支持 Raw IP 模式

域格 9X07 模块仅支持 raw IP 模式，数据包发送到模块时须去除以太报头，从模块接收数据包时须添加以太报头。参照以下修改可支持 raw IP 模式。



[KERNEL]/drivers/net/usb/qmi_wwan.c

```
#include <linux/usb/usbnet.h>
#include <linux/usb/cdc-wdm.h>
#if 1// Add for YUGA 9X07
#include <linux/version.h>
#include <linux/etherdevice.h>

/* very simplistic detection of IPv4 or IPv6 headers */
static bool possibly_iphdr(const char *data)
{
    return (data[0] & 0xd0) == 0x40;
}

struct sk_buff *qmi_wwan_tx_fixup_YUGA(struct usbnet *dev, struct sk_buff *skb, gfp_t
flags)
{
    if (dev->udev->descriptor.idVendor != cpu_to_le16(0x05c6))
        return skb;

    // Skip Ethernet header from message
    if (skb_pull(skb, ETH_HLEN)) {
        return skb;
    } else {
        dev_err(&dev->intf->dev, "Packet Dropped ");
    }

    // Filter the packet out, release it
    dev_kfree_skb_any(skb);
    return NULL;
}
#endif

static int qmi_wwan_rx_fixup_YUGA(struct usbnet *dev, struct sk_buff *skb)
```



```
{
    __be16 proto;
    if (dev->udev->descriptor.idVendor != cpu_to_le16(0x05c6))
        return 1;

    /* This check is no longer done by usbnet */
    if (skb->len < dev->net->hard_header_len)
        return 0;

    switch (skb->data[0] & 0xf0) {
    case 0x40:
        proto = htons(ETH_P_IP);
        break;

    case 0x60:
        proto = htons(ETH_P_IPV6);
        break;

    case 0x00:
        if (is_multicast_ether_addr(skb->data))
            return 1;

        /* possibly bogus destination - rewrite just in case */
        skb_reset_mac_header(skb);
        goto fix_dest;

    default:
        /* pass along other packets without modifications */
        return 1;
    }

    if (skb_headroom(skb) < ETH_HLEN)
        return 0;

    skb_push(skb, ETH_HLEN);
    skb_reset_mac_header(skb);
    eth_hdr(skb)->h_proto = proto;
    memset(eth_hdr(skb)->h_source, 0, ETH_ALEN);
}
```



```
fix_dest:
    memcpy(eth_hdr(skb)->h_dest, dev->net->dev_addr, ETH_ALEN);
    return 1;
}

.....

/* If the follow function exist, modify it as below. If not, add it. */
static int qmi_wwan_bind_shared(struct usbnet *dev, struct usb_interface *intf)
{
    int rv;
    struct usb_driver *subdriver = NULL;
    atomic_t *pmcount = (void *)&dev->data[1];

    if (dev->driver_info->data &&
        !test_bit(intf->cur_altsetting->desc.bInterfaceNumber, &dev->driver_info->data)) {
        dev_info(&intf->dev, "not on our whitelist - ignored");
        rv = -ENODEV;
        goto err;
    }

    atomic_set(pmcount, 0);

    /* collect all three endpoints */
    rv = usbnet_get_endpoints(dev, intf);
    if (rv < 0)
        goto err;

    /* require interrupt endpoint for subdriver */
    if (!dev->status) {
        rv = -EINVAL;
        goto err;
    }
}
```




```
subdriver = usb_cdc_wdm_register(intf, &dev->status->desc, 512,
    &qmi_wwan_cdc_wdm_manage_power);
if (IS_ERR(subdriver)) {
    rv = PTR_ERR(subdriver);
    goto err;
}

/* can't let usbnet use the interrupt endpoint */
dev->status = NULL;

/* save subdriver struct for suspend/resume wrappers */
dev->data[0] = (unsigned long)subdriver;

#if 1
    if ((dev->udev->descriptor.idVendor == cpu_to_le16(0x05c6))) {
        dev_info(&intf->dev, "yuge 9x07 work on RawIP mode\n");
        dev->net->flags |= IFF_NOARP;
    }
#endif
/* make MAC addr easily distinguishable from an IP header */
if (possibly_iphdr(dev->net->dev_addr)) {
    dev->net->dev_addr[0] |= 0x02; /* set local assignment bit */
    dev->net->dev_addr[0] &= 0xbf; /* clear "IP" bit */
}
#endif

usb_control_msg(
    interface_to_usbdev(intf),
    usb_sndctrlpipe(interface_to_usbdev(intf), 0),
    0x22, //USB_CDC_REQ_SET_CONTROL_LINE_STATE
    0x21, //USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE
    1, //active CDC DTR
    intf->cur_altsetting->desc.bInterfaceNumber,
    NULL, 0, 100);
```



```
    }
#endif

err:
    return rv;
}
.....

/* If the follow function exist, modify it as below. If not, add it. */
static void qmi_wwan_unbind_shared(struct usbnet *dev, struct usb_interface *intf)
{
    struct usb_driver *subdriver = (void *)dev->data[0];

    if (subdriver && subdriver->disconnect)
        subdriver->disconnect(intf);

    dev->data[0] = (unsigned long)NULL;
}

/* If the follow function exist, modify it as below. If not, add it. */
static const struct driver_info qmi_wwan_force_int4 = {
    .description= "Qualcomm WWAN/QMI device",
    .flags= FLAG_WWAN,
    .bind= qmi_wwan_bind_shared,
    .unbind= qmi_wwan_unbind_shared,
    .manage_power= qmi_wwan_manage_power,
    .data= BIT(4), /* interface whitelist bitmap */

    .tx_fixup      = qmi_wwan_tx_fixup_YUGA,
    .rx_fixup      = qmi_wwan_rx_fixup_YUGA,
};
```



6.3 修改内核配置

按如下方法修改内核配置：

cd kernel

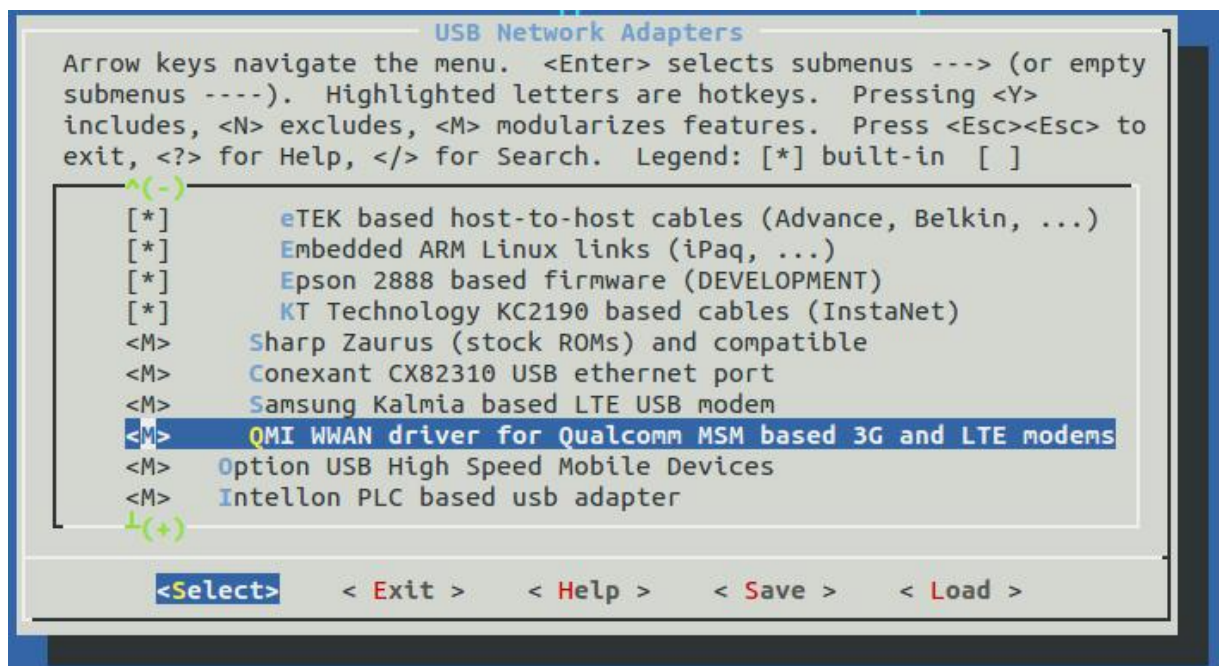
make menuconfig

Device Drivers → Network device support → USB Network Adapters → Multi-purpose

USB Networking Framework

如图，选择如下选项

QMI WWAN driver for Qualcomm MSM based 3G and LTE modems



完成后，重新编译内核。



7. Linux 下 FAQ

7.1 问：内核里添加了域格模块 VID PID，为何 ls /dev/ttyU* 仍查看不到端口？

答：首先，要确认已给模块上电，且 USB 已成功接入。可通过 lsusb 或 dmesg 查看接入的 USB 设备信息，确认模块已接入到系统，否则要先确认硬件连接是否正确。

通过 lsusb 或 dmesg 查看到模块信息后，再确认添加的 VID PID 是否正确。核对无误后，最终确认修改的信息是否被系统编译到。

以上信息都确认无误，就能通过 ls /dev/ttyU* 查看到端口了。

7.2 问：linux 下如何通过 echo cat 手动发送 AT 命令？

答：以向 ttyUSB2 发送命令 AT 为例，可按以下命令操作（通过 ctrl+c 退出）

```
sudo echo -en "AT\r\n" > /dev/ttyUSB2;cat /dev/ttyUSB2
```

```
test@yuge-info:~$ sudo echo -en "ATE\r\n" > /dev/ttyUSB2;cat /dev/ttyUSB2
OK
^C
test@yuge-info:~$ sudo echo -en "AT\r\n" > /dev/ttyUSB2;cat /dev/ttyUSB2
AT
OK
```

7.3 问：为何在 linux 下读取不到模块的主动上报信息？

答：不可在 generic.c 中添加模块 VID PID，需在 option.c 文件中添加，确认模块加载为 GSM modem。