WCDMA&LTE Linux USB Driver User Guide

WCDMA/LTE Standard/Automotive/LTE-A Module Series

Rev. WCDMA&LTE_Linux_USB_Driver_User_Guide_V1.9

Date: 2019-03-25

Status: Preliminary

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
7th Floor, Hongye Building, No.1801 Hongmei Road, Xuhui District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/sales.htm

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm
Or Email to: support@quectel.com

GENERAL NOTES

QUECTEL OFFERS THE INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS' REQUIREMENTS. QUECTEL MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. QUECTEL DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

COPYRIGHT

THE INFORMATION CONTAINED HERE IS PROPRIETARY TECHNICAL INFORMATION OF QUECTEL CO., LTD. TRANSMITTING, REPRODUCTION, DISSEMINATION AND EDITING OF THIS DOCUMENT AS WELL AS UTILIZATION OF THE CONTENT ARE FORBIDDEN WITHOUT PERMISSION. OFFENDERS WILL BE HELD LIABLE FOR PAYMENT OF DAMAGES. ALL RIGHTS ARE RESERVED IN THE EVENT OF A PATENT GRANT OR REGISTRATION OF A UTILITY MODEL OR DESIGN.

Copyright © Quectel Wireless Solutions Co., Ltd. 2019. All rights reserved.

About the Document

History

Revision	Date	Author	Description
1.0	2015-02-27	Joe WANG	Initial
1.1	2015-3-25	Carl YIN	Updated supported products
1.2	2015-3-30	Kent XU	Added Zero Packet feature in Section 3.2.2 and 3.3.2
1.3	2015-06-24	Carl YIN	1. Added GobiNet and QMI WWAN description in Section 3.4 and 3.5 2. Added building drivers as a kernel module in Section 3.2.4/3.3.4/3.4.3/3.5.4 3. Added power management in Chapter 4 4. Added FAQ and kernel log in Chapter 6
1.4	2015-12-16		1. Deleted Auto-Connect of GobiNet and QMI WWAN 2. Updated the usage of quectel-CM
1.5	2016-05-13	Carl YIN/ Neo HOU	Updated supported modules
1.6	2016-08-23	Kent XU	Added EC20 R2.0 in supported modules
1.7	2017-05-24	Kent XU	Added EG91/EG95/EG06/EP06/EM06/BG96 in supported modules
1.8	2017-09-01	Kent XU	Updated description of supported modules and added AG35 in supported modules.
1.9	2019-03-25	Carl YIN	1. Updated description of supported modules and added EG12/EM12/EP12/EG16/EG18 in supported modules. 2. Add ECM, MBIM, QMAP, AT$QCRMCALL in Chapter 5.

Contents

Table Index

Figure Index

1 Introduction

This document introduces how to integrate the USB driver for Quectel module in Linux OS, and how to use the module after the USB driver is loaded successfully.

1.1. Applicable Products

The document is applicable to the following Quectel modules.

Table 1: Applicable Products

Module Series	Modules	Supported Driver	Comment
WCDMA	UC15	USB Serial	Refer to Section 3.2
	UC20	USB Serial	Refer to Section 3.2
		GobiNet	Refer to Section 3.4
		QMI WWAN	Refer to Section 3.5
	UG95/UG96	CDC ACM	Refer to Section 3.3
LTE Standard	EC25/EC21/EC20/ EG91/EG95	USB Serial	Refer to Section 3.2
		GobiNet	Refer to Section 3.4
		QMI WWAN	Refer to Section 3.5
Automotive	AG35	USB Serial	Refer to Section 3.2
		GobiNet	Refer to Section 3.4
		QMI WWAN	Refer to Section 3.5
LTE-A	EG06/EP06/EM06/E G12/EP12/EM12/EG 16/EG18	USB Serial	Refer to Section 3.2
		GobiNet	Refer to Section 3.4
		QMI WWAN	Refer to Section 3.5

2 Products Overview

USB on Quectel UMTS/HSPA/LTE module contains several different functional interfaces. The following table describes the interface information of different modules in the Linux system.

Table 2: Interface Information

Product			USB Driver	Interface
UC15	VID: 0x05c6	PID: 0x9090	USB Serial Option	ttyUSB0→DM
UC20	VID: 0x05c6	PID: 0x9003		
EC25	VID: 0x2c7c	PID: 0x0125		
EC21	VID: 0x2c7c	PID: 0x0121		ttyUSB1→For GPS NMEA message output
EC20	VID: 0x05c6	PID: 0x9215		
EG91	VID: 0x2c7c	PID: 0x0191		
EG95	VID: 0x2c7c	PID: 0x0195		
BG96	VID: 0x2c7c	PID: 0x0296		ttyUSB2→For AT command communication
AG35	VID: 0x2c7c	PID: 0x0435		
EG06	VID: 0x2c7c	PID: 0x0306		
EP06	VID: 0x2c7c	PID: 0x0306		ttyUSB3→For PPP connections or AT command communication
EM06	VID: 0x2c7c	PID: 0x0306		
EG12	VID: 0x2c7c	PID: 0x0512		
EP12	VID: 0x2c7c	PID: 0x0512		
EM12	VID: 0x2c7c	PID: 0x0512		
EG16	VID: 0x2c7c	PID: 0x0512		
EG18	VID: 0x2c7c	PID: 0x0512		
UC20	VID: 0x05c6	PID: 0x9003	GobiNet or QMI WWAN	ethX or wwanX→ Interface 4 can be used as USB network adapter
EC25	VID: 0x2c7c	PID: 0x0125		
EC21	VID: 0x2c7c	PID: 0x0121		
EC20	VID: 0x05c6	PID: 0x9215		
EG91	VID: 0x2c7c	PID: 0x0191		
EG95	VID: 0x2c7c	PID: 0x0195		
EG06	VID: 0x2c7c	PID: 0x0306		
EP06	VID: 0x2c7c	PID: 0x0306		
EM06	VID: 0x2c7c	PID: 0x0306		
BG96	VID: 0x2c7c	PID: 0x0296		
AG35	VID: 0x2c7c	PID: 0x0435		

UG95/UG96 VID: 0x1519	PID: 0x0020	CDC ACM	ttyACM0→For PPP connections or AT command communication
			ttyACM1→Trace 1
			ttyACM2→Trace 2
			ttyACM3→For AT command communication
			ttyACM4→For AT command communication
			ttyACM5→Reserved
			ttyACM6→Reserved

3 System Setup

This chapter mainly describes the general organization of the USB stack in Linux and how to use USB serial, CDC ACM, GobiNet and QMI WWAN drivers, as well as how to compile and load the drivers.

3.1. Linux USB Driver Structure

USB is a kind of hierarchical bus structure. The data transmission between USB devices and host is achieved by USB controller. The following picture illustrates the architecture of USB driver. Linux USB host driver includes three parts: USB host controller driver, USB core and USB device drivers.

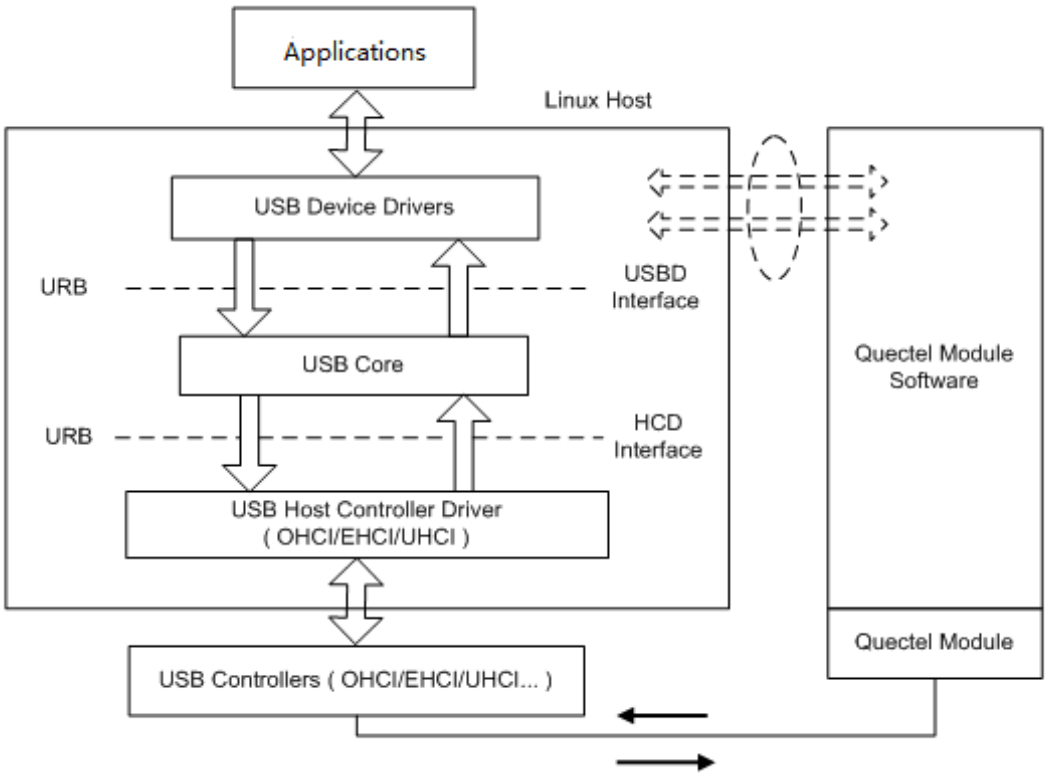

Figure 1: USB Driver Structure

USB host controller driver, the bottom of the hierarchical structure, is a software module which interacts directly with hardware.

USB core, the core of the whole USB host driver, is responsible for the management of USB bus, USB bus devices and USB bus bandwidth; it provides the interfaces for USB device drivers, through which the applications can access the USB system files.

USB device drivers interact with the applications, and mainly provide the interfaces for accessing the specific USB devices.

3.2. USB Serial Driver

If customers are using UC15/UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 and requiring USB serial driver, please read this section for details. Otherwise, please skip this section.

When the module is attached to the USB serial driver, the driver will create device files in directory */dev*, named as below:

ttyUSB0/ttyUSB1/ttyUSB2…

The following parts show how to integrate USB serial driver.

3.2.1. Add VID and PID

In order to recognize the module, customers should add module VID and PID information as below:

File: *[KERNEL]/drivers/usb/serial/option.c*

```
static const struct usb_device_id option_ids[] = {
#if 1 //Added by Quectel
    { USB_DEVICE(0x05C6, 0x9090) }, /* Quectel UC15 */
    { USB_DEVICE(0x05C6, 0x9003) }, /* Quectel UC20 */
    { USB_DEVICE(0x2C7C, 0x0125) }, /* Quectel EC25 */
    { USB_DEVICE(0x2C7C, 0x0121) }, /* Quectel EC21 */
    { USB_DEVICE(0x05C6, 0x9215) }, /* Quectel EC20 */
    { USB_DEVICE(0x2C7C, 0x0191) }, /* Quectel EG91 */
    { USB_DEVICE(0x2C7C, 0x0195) }, /* Quectel EG95 */
    { USB_DEVICE(0x2C7C, 0x0306) }, /* Quectel EG06/EP06/EM06 */
    { USB_DEVICE(0x2C7C, 0x0512) }, /* Quectel EG12/EP12/EM12/EG16/EG18 */
    { USB_DEVICE(0x2C7C, 0x0296) }, /* Quectel BG96 */
    { USB_DEVICE(0x2C7C, 0x0435) }, /* Quectel AG35 */
#endif
```

For EC20 module, if the following files and statements exist in the kernel source files, please delete them,

as they will conflict with EC20's USB driver.

File: *[KERNEL]/drivers/usb/serial/qcserial.c*

```
{USB_DEVICE(0x05c6, 0x9215)}, /* Acer Gobi 2000 Modem device (VP413) */
```

File: *[KERNEL]/drivers/net/usb/qmi_wwan.c*

```
{QMI_GOBI_DEVICE(0x05c6, 0x9215)},          /* Acer Gobi 2000 Modem device (VP413) */
```

3.2.2. Add the Zero Packet Mechanism

As required by the USB protocol, customers need to add the mechanism for processing zero packets during bulk out transmission.

For Linux kernel version higher than 2.6.34:

File: *[KERNEL]/drivers/usb/serial/usb_wwan.c*

```
static struct urb *usb_wwan_setup_urb(struct usb_serial *serial, int endpoint,
                                      int dir, void *ctx, char *buf, int len,void (*callback) (struct urb *))
{
……
    usb_fill_bulk_urb(urb, serial->dev,
                      usb_sndbulkpipe(serial->dev, endpoint) | dir,
                      buf, len, callback, ctx);
    #if 1     //Added by Quectel for zero packet
    if (dir == USB_DIR_OUT) {
        struct usb_device_descriptor *desc = &serial->dev->descriptor;
        if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProduct == cpu_to_le16(0x9090))
            urb->transfer_flags |= URB_ZERO_PACKET;
        if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProduct == cpu_to_le16(0x9003))
            urb->transfer_flags |= URB_ZERO_PACKET;
        if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProduct == cpu_to_le16(0x9215))
            urb->transfer_flags |= URB_ZERO_PACKET;
        if (desc->idVendor == cpu_to_le16(0x2C7C))
            urb->transfer_flags |= URB_ZERO_PACKET;
    }
    #endif
    return urb;
}
```

For Linux kernel version lower than 2.6.35:

File: *[KERNEL]/drivers/usb/serial/option.c*

```
/* Helper functions used by option_setup_urbs */
static struct urb *option_setup_urb(struct usb_serial *serial, int endpoint,
            int dir, void *ctx, char *buf, int len,
            void (*callback)(struct urb *))
{
……
      usb_fill_bulk_urb(urb, serial->dev,
                  usb_sndbulkpipe(serial->dev, endpoint) | dir,
                  buf, len, callback, ctx);
      #if 1     //Added by Quectel for zero packet
      if (dir == USB_DIR_OUT) {
            struct usb_device_descriptor *desc = &serial->dev->descriptor;
            if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProduct == cpu_to_le16(0x9090))
                  urb->transfer_flags |= URB_ZERO_PACKET;
            if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProduct == cpu_to_le16(0x9003))
                  urb->transfer_flags |= URB_ZERO_PACKET;
            if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProduct == cpu_to_le16(0x9215))
                  urb->transfer_flags |= URB_ZERO_PACKET;
            if (desc->idVendor == cpu_to_le16(0x2C7C))
                  urb->transfer_flags |= URB_ZERO_PACKET;
      #endif
      return urb;
}
```

3.2.3. Add Reset Resume

Some USB host controllers/USB hubs will lose power or be reset when MCU entering into suspend/sleep mode, and they cannot resume USB devices after MCU exits from suspend/sleep mode. Please add the following statements to enable reset-resume process.

For Linux kernel version higher than 3.4:

File: *[KERNEL]/drivers/usb/serial/option.c*

```
static struct usb_serial_driver option_1port_device = {
……
#ifdef CONFIG_PM
      .suspend              = usb_wwan_suspend,
      .resume               = usb_wwan_resume,
#if 1    //Added by Quectel
      .reset_resume     = usb_wwan_resume,
#endif
#endif
};
```

For Linux kernel version lower than 3.5:

File: *[KERNEL]/drivers/usb/serial/ usb-serial.c*

```
/* Driver structure we register with the USB core */
static struct usb_driver usb_serial_driver = {
        .name =         "usbserial",
        .probe =        usb_serial_probe,
        .disconnect =   usb_serial_disconnect,
        .suspend =      usb_serial_suspend,
        .resume =       usb_serial_resume,
#if 1 //Added by Quectel
        .reset_resume = usb_serial_resume,
#endif
        .no_dynamic_id =        1,
        .supports_autosuspend = 1,
};
```

3.2.4. Enlarge Bulk out URBs

For Linux kernel version lower than 2.6.29, bulk out URBs need to be enlarged to get faster uplink speed.

File: *[KERNEL]/drivers/usb/serial/option.c*

```
#define N_IN_URB 4
#define N_OUT_URB 4 //Quectel 1
#define IN_BUFLEN 4096
#define OUT_BUFLEN 4096 //Quectel 128
```

3.2.5. Use GobiNet or QMI WWAN

If customers are using UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 and requiring GobiNet or QMI WWAN, please add the following statements to prevent these modules' interface 4 from being used as USB serial device.

For Linux kernel version higher than 2.6.30:

File: *[KERNEL]/drivers/usb/serial/option.c*

```
static int option_probe(struct usb_serial *serial, const struct usb_device_id *id) {
    struct usb_wwan_intf_private *data;
    ……
#if 1   //Added by Quectel
//Quectel UC20's interface 4 can be used as USB network device
    if      (serial->dev->descriptor.idVendor       ==      cpu_to_le16(0x05C6)     &&
```

```
serial->dev->descriptor.idProduct == cpu_to_le16(0x9003)
            && serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
            return -ENODEV;
//Quectel EC20's interface 4 can be used as USB network device
    if          (serial->dev->descriptor.idVendor          ==          cpu_to_le16(0x05C6)          &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9215)
            && serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
            return -ENODEV;
//Quectel EC25&EC21&EG91&EG95&EG06&EP06&EM06&BG96/AG35's interface 4 can be used as
 USB network device
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)
            && serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
            return -ENODEV;
#endif
    /* Store device id so we can use it during attach. */
    usb_set_serial_data(serial, (void *)id);
    return 0;
}
```

For Linux kernel version lower than 2.6.31:

File: *[KERNEL]/drivers/usb/serial/option.c*

```
static int option_startup(struct usb_serial *serial)
{
……
    dbg("%s", __func__);
#if 1   //Added by Quectel
//Quectel UC20's interface 4 can be used as USB network device
    if          (serial->dev->descriptor.idVendor          ==          cpu_to_le16(0x05C6)          &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9003)
            && serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
            return -ENODEV;
//Quectel EC20's interface 4 can be used as USB network device
    if          (serial->dev->descriptor.idVendor          ==          cpu_to_le16(0x05C6)          &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9215)
            && serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
            return -ENODEV;
//Quectel EC25&EC21&EG91&EG95&EG06&EP06&EM06&BG96/AG35's interface 4 can be used as
 USB network device
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)
            && serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
            return -ENODEV;
#endif
……
```

```
}
```

3.2.6. Modify Kernel Configuration

There are several mandatory selected items in kernel configuration, please follow the steps below to configure the kernel:

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Set environment variables, and import board's defconfig. The following is an example for Raspeberry Pi board.

```
export ARCH=arm

export CROSS_COMPILE=arm-none-linux-gnueabi-

make bcmrpi_defconfig
```

Step 3: Compile the kernel.

```
make menuconfig
```

Step 4: Enable CONFIG_USB_SERIAL_OPTION

```
[*] Device Drivers →
    [*] USB Support →
        [*] USB Serial Converter support →
            [*] USB driver for GSM and CDMA modems
```

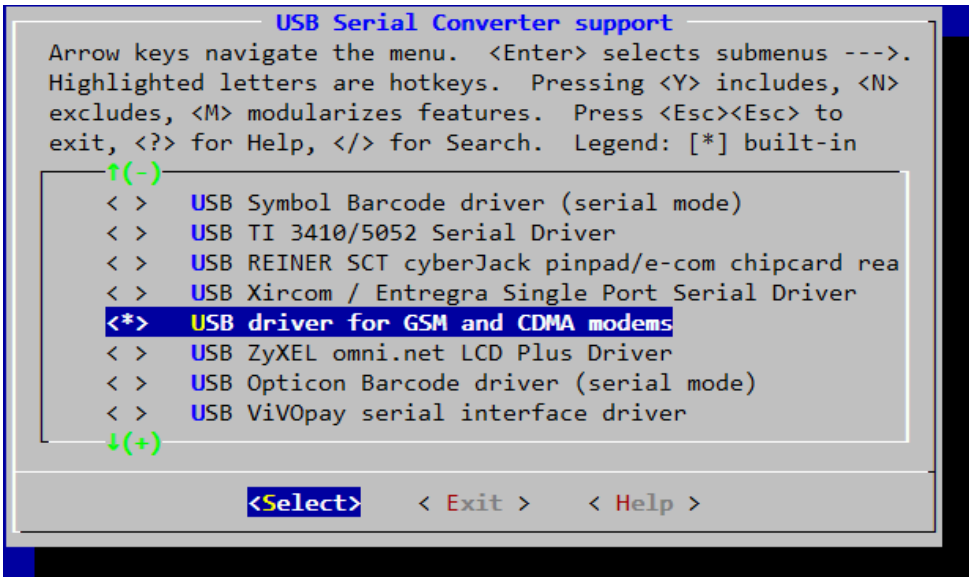

Figure 2: Configure USB Serial in Kernel

3.2.7. Build and Load Driver as A Kernel Module for PC in Linux

Please follow the steps below to build the driver as a kernel module, and use modprobe command to load the module with Linux OS on PC.

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Build the driver.

```
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/usb/serial obj-m=option.o modules
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/usb/serial obj-m=usb_wwan.o modules
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/usb/serial obj-m=qcserial.o modules
```

Step 3: Load the driver and reboot.

```
sudo cp drivers/usb/serial/option.ko /lib/modules/`uname -r`/kernel/drivers/usb/serial
sudo cp drivers/usb/serial/usb_wwan.ko /lib/modules/`uname -r`/kernel/drivers/usb/serial
sudo cp drivers/usb/serial/qcserial.ko /lib/modules/`uname -r`/kernel/drivers/usb/serial
sudo depmod
sudo reboot
```

3.3. CDC ACM Driver for UG95/UG96

If customers are using UG95/UG96 and requiring CDC ACM driver, please read this section for details. Otherwise, please skip this section.

When the module is attached to CDC ACM driver, the driver will create device files in directory */dev*, named as below:

ttyACM0/ttyACM1/ttyACM2…

The following parts show how to integrate the CDC ACM driver.

3.3.1. Modify Driver Source Code

The device is attached to CDC ACM driver according to the USB class type, so customers do not need to add PID and VID information in driver source code.

3.3.2. Add the Zero Packet Mechanism

As required by the USB protocol, customers need to add the mechanism for processing zero packets during transmission to file *[KERNEL]/drivers/usb/class/cdc-acm.c*:

This document takes the Linux 3.2 as an example, and there may be a little difference to other versions; but they are basically the same.

Please add the following statements to the *acm_probe* function, as shown below:

```
.....
for (i = 0; i < ACM_NW; i++) {
      struct acm_wb *snd = &(acm->wb[i]);
      snd->urb = usb_alloc_urb(0, GFP_KERNEL);
      if (snd->urb == NULL) {
            dev_err(&intf->dev,
                  "out of memory (write urbs usb_alloc_urb)\n");
            goto alloc_fail7;
      }
      if (usb_endpoint_xfer_int(epwrite))
            usb_fill_int_urb(snd->urb, usb_dev,
                  usb_sndbulkpipe(usb_dev, epwrite->bEndpointAddress),
                  NULL, acm->writesize, acm_write_bulk, snd, epwrite->bInterval);
      else
            usb_fill_bulk_urb(snd->urb, usb_dev,
                  usb_sndbulkpipe(usb_dev, epwrite->bEndpointAddress),
                  NULL, acm->writesize, acm_write_bulk, snd);
      snd->urb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
      #if 1    //Added by Quectel for zero packet
      if (usb_dev->descriptor.idVendor == 0x1519 && usb_dev->descriptor.idProduct == 0x0020)
            snd->urb->transfer_flags |= URB_ZERO_PACKET;
      #endif
     snd->instance = acm;
}
usb_set_intfdata(intf,acm)
.....
```

3.3.3. Add Reset Resume

Some USB host controllers/USB hubs will lose power or be reset when MCU entering into suspend/sleep mode, and they cannot resume USB devices after MCU exits from suspend/sleep mode. Please add the following statements to enable reset-resume process.

For Linux kernel version lower than 2.6.35:

File: *[KERNEL]/drivers/usb/class/cdc-acm.c*

```
static struct usb_driver acm_driver = {
        .name =         "cdc_acm",
        .probe =        acm_probe,
        .disconnect =   acm_disconnect,
#ifdef CONFIG_PM
        .suspend =      acm_suspend,
        .resume =       acm_resume,
#if 1 //Added by Quectel
        .reset_resume = acm_resume,
#endif
#endif
        .id_table =     acm_ids,
#ifdef CONFIG_PM
        .supports_autosuspend = 1,
#endif
};
```

3.3.4. Modify Kernel Configuration

There are several mandatory selected items in kernel configuration, please follow the steps below to configure the kernel:

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Set environment variables, and import board's defconfig. The following is an example for Raspeberry Pi board.

```
export ARCH=arm

export CROSS_COMPILE=arm-none-linux-gnueabi-

make bcmrpi_defconfig
```

Step 3: Compile the kernel.

```
make menuconfig
```

Step 4: Enable CONFIG_USB_ACM

```
[*] Device Drivers →
    [*] USB Support →
        [*] USB Modem (CDC ACM) support
```

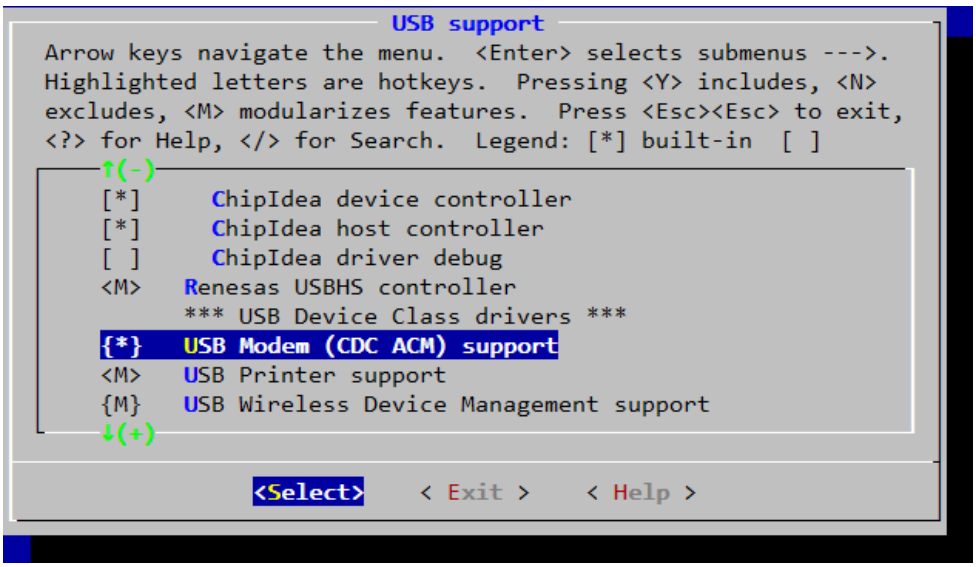

Figure 3: Configure CDC ACM Driver in Kernel

3.3.5. Build and Load Driver as a Kernel Module for PC in Linux

Please follow the steps below to build the driver as a kernel module, and use modprobe command to load the module with Linux OS on PC.

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Build the driver.

```
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/usb/class obj-m=cdc-acm.o modules
```

Step 3: Load the driver and reboot.

```
sudo cp drivers/usb/class/cdc-acm.ko /lib/modules/`uname -r`/kernel/drivers/usb/class
sudo depmod
sudo reboot
```

3.4. GobiNet Driver

If customers are using UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12 /EM12/EG16/EG18 and requiring GobiNet driver, please read this section for details. Otherwise, please skip this section.

When the module is attached to GobiNet driver, the driver will create a network device and a QMI channel.

The network device is named as *ethX* (usbX if the kernel version is 2.6.39 or older), and the QMI channel is named as */dev/qcqmiX*. The network device is used for data transmission, and QMI channel is used for QMI message interaction.

The following parts show how to integrate the GobiNet driver.

3.4.1. Modify Driver Source Code

The GobiNet driver is provided by Quectel as a form of source file. Customers should copy the source files to *[KERNEL]/drivers/net/usb/* (or *[KERNEL]/drivers/usb/net/* if the kernel version is lower than 2.6.22).

3.4.2. Modify Kernel Configuration

There are several mandatory selected items in kernel configuration, please follow the steps below to configure the kernel:

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Set environment variables, and import board's defconfig. The following is an example for Raspeberry Pi board.

```
export ARCH=arm

export CROSS_COMPILE=arm-none-linux-gnueabi-

make bcmrpi_defconfig
```

Step 3: Compile the kernel.

```
make menuconfig
```

Step 4: Enable CONFIG_USB_USBNET

```
[*] Device Drivers →
    -*- Network device support →
        USB Network Adapters →
            {*} Multi-purpose USB Networking Framework
```

Step 5: Please add the following statements to file *[KERNEL]/drivers/net/usb/Makefile* (or *[KERNEL]/drivers/usb/net/Makefile* if the kernel version is lower than 2.6.22).

```
obj-y += GobiNet.o
GobiNet-objs := GobiUSBNet.o QMIDevice.o QMI.o
```

For EC20 module, if the following files and statements exist in the kernel source files, please delete them, as they will conflict with EC20's USB driver.

File: *[KERNEL]/drivers/usb/serial/qcserial.c*

```
{USB_DEVICE(0x05c6, 0x9215)}, /* Acer Gobi 2000 Modem device (VP413) */
```

File: *[KERNEL]/drivers/net/usb/qmi_wwan.c*

```
{QMI_GOBI_DEVICE(0x05c6, 0x9215)},          /* Acer Gobi 2000 Modem device (VP413) */
```

3.4.3. Build and Load Driver as a Kernel Module for PC in Linux

Please follow the steps below to build the driver as a kernel module, and use modprobe command to load the module with Linux OS on PC.

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Build the driver.

```
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/net/usb obj-m=GobiNet.o modules
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/usb/serial obj-m=qcserial.o modules
```

Step 3: Load the driver and reboot.

```
sudo cp drivers/net/usb/GobiNet.ko /lib/modules/`uname -r`/kernel/drivers/net/usb
sudo cp drivers/usb/serial/qcserial.ko /lib/modules/`uname -r`/kernel/drivers/usb/serial
sudo depmod
sudo reboot
```

3.5. QMI WWAN Driver

If customers are using UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 and requiring QMI WWAN driver, meanwhile, the Linux kernel version is higher than 3.3, please read this section for details. Otherwise, please skip this section.

When the module is attached to QMI WWAN driver, the driver will create a network device and a QMI channel. The network device is named as *wwanX*, and QMI channel is named as */dev/cdc-wdmX*. The network device is used for data transmission, and QMI channel is used for QMI message interaction.

The following parts show how to integrate the QMI WWAN driver.

3.5.1. Add VID and PID

QMI WWAN driver source file is *[KERNEL]/drivers/net/usb/qmi_wwan.c*.

In order to recognize the module, customers should add module PID and VID information as below:

For Linux kernel version higher than 4.9.0:

File: *[KERNEL]/drivers/net/usb/qmi_wwan.c*

```
static const struct usb_device_id products[] = {
    { QMI_FIXED_INTF(0x05C6, 0x9003, 4) },    /* Quectel UC20 */
    { QMI_FIXED_INTF(0x05C6, 0x9215, 4) },    /* Quectel EC20 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0125, 4) },    /* Quectel EC25 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0121, 4) },    /* Quectel EC21 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0191, 4) },    /* Quectel EG91 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0195, 4) },    /* Quectel EG95 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0306, 4) },    /* Quectel EG06/EP06/EM06 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0512, 4) },    /* Quectel EG12/EP12/EM12/EG16/EG18 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0296, 4) },    /* Quectel BG96 */
    { QMI_QUIRK_SET_DTR (0x2C7C, 0x0435, 4) },    /* Quectel AG35 */
#endif
```

For Linux kernel version lower than 4.9.0:

File: *[KERNEL]/drivers/net/usb/qmi_wwan.c*

```
static const struct usb_device_id products[] = {
#if 1    //Added by Quectel
#ifndef QMI_FIXED_INTF
/* map QMI/wwan function by a fixed interface number */
#define QMI_FIXED_INTF(vend, prod, num) \
        .match_flags                                =    USB_DEVICE_ID_MATCH_DEVICE    |
USB_DEVICE_ID_MATCH_INT_INFO, \
        .idVendor                = vend, \
        .idProduct               = prod, \
        .bInterfaceClass       = 0xff, \
        .bInterfaceSubClass = 0xff, \
        .bInterfaceProtocol = 0xff, \
        .driver_info           = (unsigned long)&qmi_wwan_force_int##num,
#endif
    { QMI_FIXED_INTF(0x05C6, 0x9003, 4) },    /* Quectel UC20 */
    { QMI_FIXED_INTF(0x2C7C, 0x0125, 4) },    /* Quectel EC25 */
```

```
    { QMI_FIXED_INTF(0x2C7C, 0x0121, 4) },    /* Quectel EC21 */
    { QMI_FIXED_INTF(0x05C6, 0x9215, 4) },    /* Quectel EC20 */
    { QMI_FIXED_INTF(0x2C7C, 0x0191, 4) },    /* Quectel EG91 */
    { QMI_FIXED_INTF(0x2C7C, 0x0195, 4) },    /* Quectel EG95 */
    { QMI_FIXED_INTF(0x2C7C, 0x0306, 4) },    /* Quectel EG06/EP06/EM06 */
    { QMI_FIXED_INTF(0x2C7C, 0x0512, 4) },    /* Quectel EG12/EP12/EM12/EG16/EG18 */
    { QMI_FIXED_INTF(0x2C7C, 0x0296, 4) },    /* Quectel BG96 */
    { QMI_FIXED_INTF(0x2C7C, 0x0435, 4) },    /* Quectel AG35 */
#endif
```

For EC20 module, if the following files and statements exist in the kernel source files, please delete them, as they will conflict with EC20's USB driver.

File: *[KERNEL]/drivers/usb/serial/qcserial.c*

```
{USB_DEVICE(0x05c6, 0x9215)}, /* Acer Gobi 2000 Modem device (VP413) */
```

File: *[KERNEL]/drivers/net/usb/qmi_wwan.c*

```
{QMI_GOBI_DEVICE(0x05c6, 0x9215)},          /* Acer Gobi 2000 Modem device (VP413) */
```

3.5.2. Add Support for Raw IP Mode

QMI WWAN driver source file is *[KERNEL]/drivers/net/usb/qmi_wwan.c*.

EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 only support raw IP mode (IP packets not encapsulated in Ethernet frames). So Ethernet header must be stripped when packets are sent to the module, and be added when packets are received from the module.

For Linux kernel version higher than 4.9.0:
The upstream kernel had support raw IP mode, then customer just need to enable it by next commands.

```
root@imx6qdlsabresd:/media# ifconfig wwan0 down
root@imx6qdlsabresd:/media# echo 1 > /sys/class/net/wwan0/qmi/raw_ip
```

For Linux kernel version lower than 4.9.0:

Please add the following statements to support raw IP mode.

File: *[KERNEL]/drivers/net/usb/qmi_wwan.c*

```
#include <linux/usb/usbnet.h>
#include <linux/usb/cdc-wdm.h>
```

```
#if 1    //Added by Quectel
#include <linux/etherdevice.h>struct sk_buff *qmi_wwan_tx_fixup(struct usbnet *dev, struct sk_buff *skb, gfp_t flags)
{
    if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
        return skb;

    // Skip Ethernet header from message
    if (dev->net->hard_header_len == 0)
        return skb;
    else
        skb_reset_mac_header(skb); 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;
}

#include <linux/version.h>
#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
static int qmi_wwan_rx_fixup(struct usbnet *dev, struct sk_buff *skb)
{
    __be16 proto;

    if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
        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;
}

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

……

/* if follow function exist, modify it as below */
static int qmi_wwan_bind(struct usbnet *dev, struct usb_interface *intf)
{
……

#if 1 //Added by Quectel
        if (dev->udev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
            if (intf->cur_altsetting->desc.bInterfaceClass != 0xff) {
                dev_info(&intf->dev,     "Quectel module not qmi_wwan mode! please check 'at+qcfg=\"usbnet\"'\n");
                return -ENODEV;
            }
            dev_info(&intf->dev,                                                    "Quectel EC25&EC21&EG91&EG95&EG06&EP06&EM06&EG12&EP12&EM12&EG16&EG18&BG96&AG35 work on RawIP mode\n");
```

```
        dev->net->flags |= IFF_NOARP;
#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
        /* 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 status;
}

……

/* if follow function exist, modify it as below */
static int qmi_wwan_bind_shared(struct usbnet *dev, struct usb_interface *intf)
{
……

#if 1 //Added by Quectel
    if (dev->udev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
        if (intf->cur_altsetting->desc.bInterfaceClass != 0xff) {
            dev_info(&intf->dev,    "Quectel module not qmi_wwan mode! please check
'at+qcfg=\"usbnet\"'\n");
            return -ENODEV;
        }
        dev_info(&intf->dev,                                                    "Quectel
EC25&EC21&EG91&EG95&EG06&EP06&EM06&EG12&EP12&EM12&EG16&EG18&BG96&AG35
work on RawIP mode\n");
        dev->net->flags |= IFF_NOARP;
#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
        /* 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 status;
}
……

/* if follow struct exist, modify it as below */
static const struct driver_info      qmi_wwan_info =
{
……
#if 1 //Added by Quectel
    .tx_fixup           = qmi_wwan_tx_fixup,
    .rx_fixup           = qmi_wwan_rx_fixup,
#endif
}
……

/* if follow struct exist, modify it as below */
static const struct driver_info qmi_wwan_force_int4 = {
……
#if 1 //Added by Quectel
    .tx_fixup           = qmi_wwan_tx_fixup,
    .rx_fixup           = qmi_wwan_rx_fixup,
#endif
};

/* if follow struct exist, modify it as below */
static const struct driver_info qmi_wwan_shared = {
……
#if 1 //Added by Quectel
    .tx_fixup           = qmi_wwan_tx_fixup,
    .rx_fixup           = qmi_wwan_rx_fixup,
```

```
#endif
};
```

3.5.3. Modify Kernel Configuration

There are several mandatory selected items in kernel configuration, please follow the steps below to configure the kernel:

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Set environment variables, and import board's defconfig. The following is an example for Raspeberry Pi board.

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
make bcmrpi_defconfig
```

Step 3: Compile the kernel.

```
make menuconfig
```

Step 4: Enable CONFIG_USB_NET_QMI_WWAN

```
[*] Device Drivers →
    -*- Network device support →
        USB Network Adapters →
            {*} Multi-purpose USB Networking Framework
                <*>    QMI WWAN driver for Qualcomm MSM based 3G and LTE modems
```

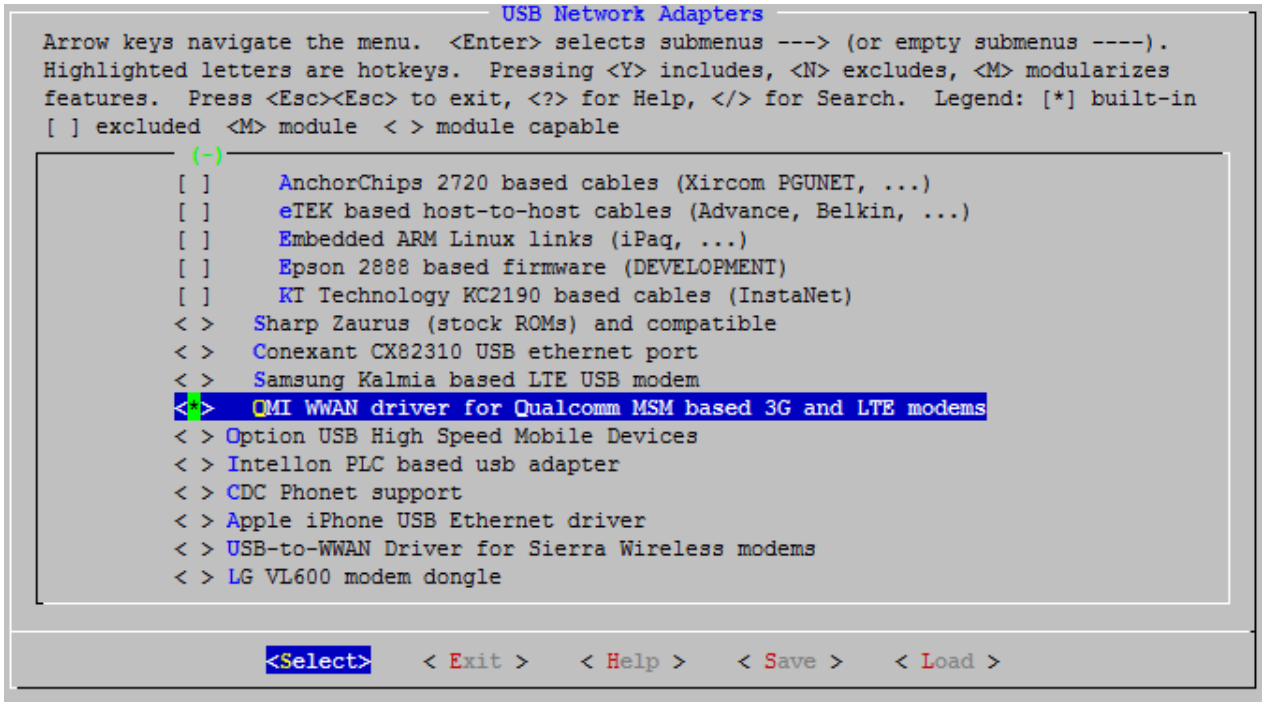

Figure 4: Configure QMI WWAN Driver in Kernel

3.5.4. Build and Load Driver as a Kernel Module for PC in Linux

Please follow steps below to build the driver as a kernel module, and use modprobe command to load the module with Linux OS on PC.

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Build the driver.

```
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/net/usb obj-m=qmi_wwan.o modules
sudo make -C /lib/modules/`uname -r`/build M=`pwd`/drivers/usb/serial obj-m=qcserial.o modules
```

Step 3: Load the driver and reboot.

```
sudo cp drivers/net/usb/qmi_wwan.ko /lib/modules/`uname -r`/kernel/drivers/net/usb
sudo cp drivers/usb/serial/qcserial.ko /lib/modules/`uname -r`/kernel/drivers/usb/serial
sudo depmod
sudo reboot
```

3.6. Configure Kernel to Support PPP

If customers need to use PPP function, please follow the steps below to configure kernel to support PPP.

Step 1: Change to kernel directory.

```
cd <your kernel directory>
```

Step 2: Set environment variables, and import board's defconfig. The following shows an example.

```
export ARCH=arm

export CROSS_COMPILE=arm-none-linux-gnueabi-

make bcmrpi_defconfig
```

Step 3: Compile the kernel.

```
make menuconfig
```

Step 4: Enable CONFIG_PPP_ASYNC CONFIG_PPP_SYNC_TTY CONFIG_PPP_DEFLATE.

```
[*] Device Drivers →
    [*] Network device support →
        [*] PPP (point-to-point protocol) support
```

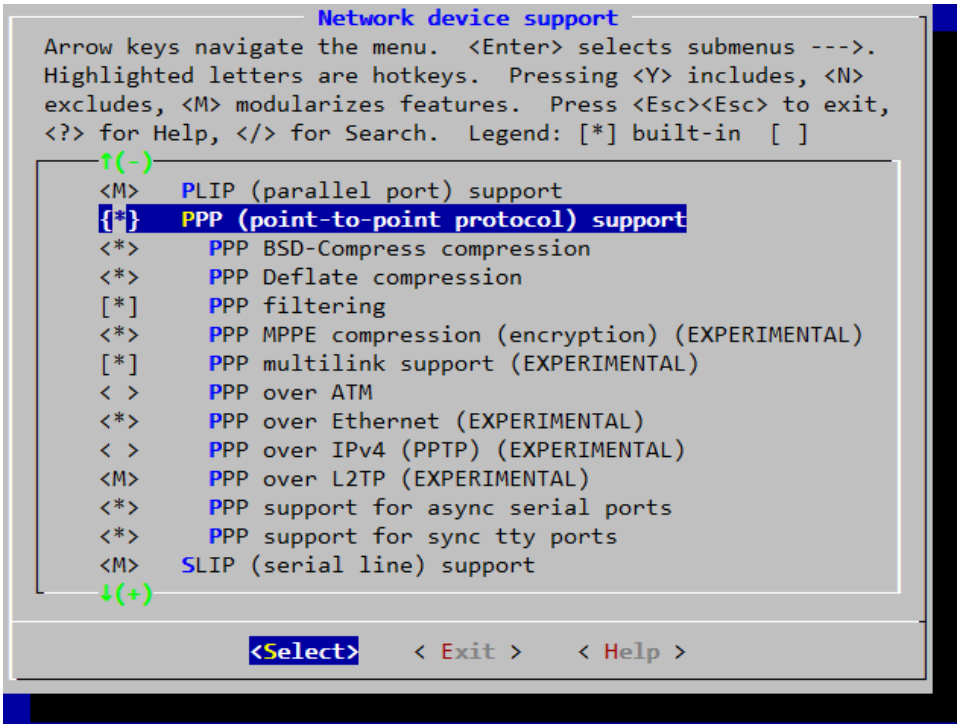

Figure 5: Configure PPP in Kernel

4 Power Management

The Linux USB system provides two advanced power management features: USB Auto Suspend and USB Remote Wakeup. This chapter introduces how to enable the features. If they are required by your product, please read this chapter for details. Otherwise, please ignore this chapter.

When USB communication between the USB host and the USB devices is idle for some time (for examples 3 seconds), the USB host can make the USB devices enter into suspend mode automatically. This feature is called USB Auto Suspend.

USB Remote Wakeup allows a suspended USB device to remotely wake up the USB host over the USB which may also be suspended (e.g. deep sleep mode). The USB device performs an activity to wake up the USB host, then the USB host will be woken up by the remote activity.

4.1. Enable USB Auto Suspend

For USB serial driver, please add the following statements to *option_probe()* function in file *[KERNEL]/drivers/usb/serial/option.c*.

```
static int option_probe(struct usb_serial *serial, const struct usb_device_id *id) {
    struct usb_wwan_intf_private *data;
    ……
#if 1 //Added by Quectel
//For USB Auto Suspend
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9090)) {
        pm_runtime_set_autosuspend_delay(&serial->dev->dev, 3000);
        usb_enable_autosuspend(serial->dev);
    }
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9003)) {
        pm_runtime_set_autosuspend_delay(&serial->dev->dev, 3000);
        usb_enable_autosuspend(serial->dev);
    }
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9215)) {
        pm_runtime_set_autosuspend_delay(&serial->dev->dev, 3000);
```

```
            usb_enable_autosuspend(serial->dev);
        }
        if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
            pm_runtime_set_autosuspend_delay(&serial->dev->dev, 3000);
            usb_enable_autosuspend(serial->dev);
        }
#endif
        /* Store device id so we can use it during attach. */
        usb_set_serial_data(serial, (void *)id);
        return 0;
}
```

For CDC ACM driver, please add the following statements to *acm_probe ()* function in file *[KERNEL]/drivers/usb/class/cdc-acm.c*.

```
static int acm_probe(struct usb_interface *intf,
                     const struct usb_device_id *id)
{
        struct usb_cdc_union_desc *union_header = NULL;
        ……
#if 1 //Added by Quectel
//For USB Auto Suspend
 if((usb_dev->descriptor.idVendor == 0x1519) && (usb_dev->descriptor.idProduct == 0x0020))
{
            pm_runtime_set_autosuspend_delay(&usb_dev->dev, 3000);
            usb_enable_autosuspend(usb_dev);
        }
#endif

        return 0;
alloc_fail8:
        if (acm->country_codes) {
……
}
```

4.2. Enable USB Remote Wakeup

For USB serial driver, please add the following statements to *option_probe()* function in file *[KERNEL]/drivers/usb/serial/option.c*.

```
static int option_probe(struct usb_serial *serial, const struct usb_device_id *id) {
        struct usb_wwan_intf_private *data;
        ……
```

```
#if 1 //Added by Quectel
//For USB Remote Wakeup
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9090)) {
        device_init_wakeup(&serial->dev->dev, 1); //usb remote wakeup
    }
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9003)) {
        device_init_wakeup(&serial->dev->dev, 1); //usb remote wakeup
    }
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct == cpu_to_le16(0x9215)) {
        device_init_wakeup(&serial->dev->dev, 1); //usb remote wakeup
    }
    if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
        device_init_wakeup(&serial->dev->dev, 1); //usb remote wakeup
    }
#endif
    /* Store device id so we can use it during attach. */
    usb_set_serial_data(serial, (void *)id);
    return 0;
}
```

For CDC ACM driver, please add the following statements to *acm_probe ()* function in file *[KERNEL]/drivers/usb/class/cdc-acm.c*.

```
static int acm_probe(struct usb_interface *intf,
                     const struct usb_device_id *id)
{
    struct usb_cdc_union_desc *union_header = NULL;
    ……
#if 1 //Added by Quectel
//For USB Remote Wakeup
    if ((usb_dev->descriptor.idVendor == 0x1519) && (usb_dev->descriptor.idProduct == 0x0020))
{
        device_init_wakeup(&usb_dev->dev, 1); //usb remote wakeup
    }
#endif
    return 0;
alloc_fail8:
    if (acm->country_codes) {
……
}
```

5 Test the Module

Generally, AT and PPP functions are supported on UCxx/EC2x/EGxx/EP06/EM06/BG96/AG35/UG95/UG96 modules. If customers are using UC20/EC2x/EGxx/EP06/EM06/BG96/AG35 and have installed GobiNet or QMI WWAN driver, the USB network adapter function can also be used on the module. The following part shows how to test these functions.

5.1. Test AT Function

After the module is connected and USB driver is loaded successfully, there will create several device files in */dev*.

The AT port of UCxx/EC2x/EGxx/EP06/EM06/BG96/AG35 is /dev/ttyUSB2, and the AT port of UG95/UG96 is /dev/ttyACM3.

Then customers can use UART port tools such as “minicom” or “busybox microcom” to test AT function, as shown below:

```
# busybox microcom /dev/ttyUSB2
```

The following is an example for EC20:

Figure 6: AT Test Result for EC20

5.2. Test PPP Function

In order to set up PPP call, the following files are required. Please check if there are such files in the module :

1. pppd and chat program:
   If the two programs do not exist, customers can download the source code of them from *https://ppp.samba.org/download.html* and port them to the module.

2. One PPP script file named as */etc/ppp/ip-up* which is used to set DNS (Domain Name System). If there is no such file, please use *linux-ppp-scripts\ip-up* provided by Quectel.

3. Three scripts named as *quectel-ppp*, *quectel-chat-connect* and *quectel-chat-disconnect*. They are provided by Quectel in directory *linux-ppp-scripts*. Depending on different modules, customers may need to make some changes. For more information, please refer to *linux-ppp-scripts\readme*.

Customers should copy *quectel-ppp*, *quectel-chat-connect* and *quectel-chat-disconnect* to the directory */etc/ppp/peers*, then start to set up PPP call via the following command:

```
# pppd call quectel-ppp &
```

The process of dialing is shown as below (example of EC20):

```
# pppd options in effect:
debug          # (from /etc/ppp/peers/quectel-ppp)
nodetach           # (from /etc/ppp/peers/quectel-ppp)
dump           # (from /etc/ppp/peers/quectel-ppp)
noauth         # (from /etc/ppp/peers/quectel-ppp)
user test          # (from /etc/ppp/peers/quectel-ppp)
password ??????          # (from /etc/ppp/peers/quectel-ppp)
remotename 3gppp         # (from /etc/ppp/peers/quectel-ppp)
/dev/ttyUSB3           # (from /etc/ppp/peers/quectel-ppp)
115200         # (from /etc/ppp/peers/quectel-ppp)
lock           # (from /etc/ppp/peers/quectel-ppp)
connect chat -s -v -f /etc/ppp/peers/quectel-chat-connect        # (from /etc/ppp/peers/quectel-ppp)
disconnect chat -s -v -f /etc/ppp/peers/quectel-chat-disconnect       # (from /etc/ppp/peers/quectel-
ppp)
nocrtscts          # (from /etc/ppp/peers/quectel-ppp)
modem          # (from /etc/ppp/peers/quectel-ppp)
hide-password          # (from /etc/ppp/peers/quectel-ppp)
novj           # (from /etc/ppp/peers/quectel-ppp)
novjccomp          # (from /etc/ppp/peers/quectel-ppp)
ipcp-accept-local          # (from /etc/ppp/peers/quectel-ppp)
ipcp-accept-remote         # (from /etc/ppp/peers/quectel-ppp)
```

```
ipparam 3gppp         # (from /etc/ppp/peers/quectel-ppp)
noipdefault         # (from /etc/ppp/peers/quectel-ppp)
ipcp-max-failure 10         # (from /etc/ppp/peers/quectel-ppp)
defaultroute        # (from /etc/ppp/peers/quectel-ppp)
usepeerdns        # (from /etc/ppp/peers/quectel-ppp)
noccp          # (from /etc/ppp/peers/quectel-ppp)
abort on (BUSY)
abort on (NO CARRIER)
abort on (NO DIALTONE)
abort on (ERROR)
abort on (NO ANSWER)
timeout set to 30 seconds
send (AT^M)
expect (OK)
^M
OK
 -- got it

send (ATE0^M)
expect (OK)
^M
^M
OK
 -- got it

send (ATI;+CSUB;+CSQ;+CPIN?;+COPS?;+CGREG?;&D2^M)
expect (OK)
^M
^M
Quectel^M
EC20^M
Revision: EC20CQAR02A03E2G_BETA0914^M
^M
SubEdition: V01^M
^M
+CSQ: 23,99^M
^M
+CPIN: READY^M
^M
+COPS: 0,0,"CHN-CT",7^M
^M
+CGREG: 2,1,"FFFE","6916934",7^M
^M
OK
```

```
 -- got it

send (AT+CGDCONT=1,"IP","3gnet",,0,0^M)
expect (OK)
^M
^M
OK
 -- got it

send (ATD*99#^M)
expect (CONNECT)
^M
^M
CONNECT
 -- got it

Script chat -s -v -f /etc/ppp/peers/quectel-chat-connect finished (pid 3017), status = 0x0
Serial connection established.
using channel 3
Using interface ppp0
Connect: ppp0 <--> /dev/ttyUSB3
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0xf2b7d6ee> <pcomp> <accomp>]
rcvd [LCP ConfReq id=0x4 <asyncmap 0x0> <auth chap MD5> <magic 0x45c0e381> <pcomp>
<accomp>]
sent [LCP ConfAck id=0x4 <asyncmap 0x0> <auth chap MD5> <magic 0x45c0e381> <pcomp>
<accomp>]
rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0xf2b7d6ee> <pcomp> <accomp>]
rcvd [LCP DiscReq id=0x5 magic=0x45c0e381]
rcvd [CHAP Challenge id=0x1 <f8d54e0fa294c100101805a512176ff1>, name =
"UMTS_CHAP_SRVR"]
sent [CHAP Response id=0x1 <e8ad86182138523599fb54a172da7154>, name = "test"]
rcvd [CHAP Success id=0x1 ""]
CHAP authentication succeeded
CHAP authentication succeeded
sent [IPCP ConfReq id=0x1 <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns2 0.0.0.0>]
rcvd [IPCP ConfReq id=0x4]
sent [IPCP ConfNak id=0x4 <addr 0.0.0.0>]
rcvd [IPCP ConfNak id=0x1 <addr 100.65.245.137> <ms-dns1 61.132.163.68> <ms-dns2
202.102.213.68>]
sent [IPCP ConfReq id=0x2 <addr 100.65.245.137> <ms-dns1 61.132.163.68> <ms-dns2
202.102.213.68>]
rcvd [IPCP ConfReq id=0x5]
sent [IPCP ConfAck id=0x5]
rcvd [IPCP ConfAck id=0x2 <addr 100.65.245.137> <ms-dns1 61.132.163.68> <ms-dns2
```

```
202.102.213.68>]
Could not determine remote IP address: defaulting to 10.64.64.64
local  IP address 100.65.245.137
remote IP address 10.64.64.64
primary     DNS address 61.132.163.68
secondary DNS address 202.102.213.68
Script /etc/ppp/ip-up started (pid 3020)
Script /etc/ppp/ip-up finished (pid 3020), status = 0x0
```

Now PPP call is set up successfully.

Please use following commands to check *IP/DNS/Route*.

```
# ifconfig ppp0
ppp0          Link encap:Point-to-Point Protocol
              inet addr:100.65.245.137   P-t-P:10.64.64.64   Mask:255.255.255.255
              UP POINTOPOINT RUNNING NOARP MULTICAST   MTU:1500   Metric:1
              RX packets:15 errors:0 dropped:0 overruns:0 frame:0
              TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:3
              RX bytes:1057 (1.0 KiB)   TX bytes:1228 (1.1 KiB)

# cat /etc/resolv.conf
nameserver 61.132.163.68
nameserver 202.102.213.68

# route   -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.64.64.64     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
0.0.0.0         0.0.0.0         0.0.0.0         U     0      0        0 ppp0

# ping www.baidu.com
PING www.a.shifen.com (115.239.211.112) 56(84) bytes of data.
64 bytes from 115.239.211.112: icmp_seq=1 ttl=54 time=46.4 ms
```

Following commands can be used to terminate PPPD process to disconnect a PPP call:

```
# killall pppd
Terminating on signal 15
Connect time 0.4 minutes.
Sent 0 bytes, received 0 bytes.
```

5.3. Test GobiNet or QMI WWAN

If customers use UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 and requiring GobiNet or QMI WWAN driver, please read this section for details. Otherwise, please skip this section.

If customers want to set up data connection manually, Quectel provides a Connect Manager program to set up data connection. The Connect Manager is provided in the form of source code in directory *quectel-CM*.

Please follow steps below to test GobiNet or QMI WWAN:

Step 1: Compile Connect Manager.

For PC Linux:

```
# make
```

For emended Linux:

```
# make CROSS-COMPILE=arm-none-linux-gnueabi-
```

Please replace *arm-none-linux-gnueabi-* by cross compiler on the module.

The output of this step is quectel-CM.

Step 2: Prepare busybox udhcpc tool.

quectel-CM will call *busybox udhpc* to obtain IP and DNS, and *busybox udhpc* will call script file */usr/share/udhcpc/default.script* to set IP/DNS/Routing table for Linux board. Customers can download this tool's source code from *https://busybox.net/*, then enable CONFIG_UDHCPC in *busybox menuconfig*, and copy the script file *[BUSYBOX]/examples/udhcp/simple.script* to Linux board (renamed as */usr/share/udhcpc/default.script*).

Step 3: Use quectel-CM to setup data call.

After the module is connected and GobiNet or QMI WWAN driver is loaded successfully, a USB network adapter and a QMI channel will be created. The USB network adapter of GobiNet is named as ethX (or usbX if the kernel version is 2.6.39 or older), and the QMI channel is named as /dev/qcqmiX. The USB network adapter of QMI WWAN is named as wwanX, and the QMI channel name is named as /dev/cdc-wdmX.

quectel-CM will send QMI message to the module via QMI channel to setup data connection. Please refer

to the following message to use quectel-CM:

```
# quectel-CM -h
Usage: ./quectel-CM [-s [apn [user password auth]]] [-p pincode] [-f logfilename]
-s [apn [user password auth]] Set apn/user/password/auth get from your network provider
-p pincode                    Verify sim card pin if sim card is locked
-f logfilename                Save log message of this program to file
Example 1: ./quectel-CM
Example 2: ./quectel-CM -s 3gnet
Example 3: ./quectel-CM -s 3gnet carl 1234 0 -p 1234 -f gobinet_log.txt
```

The process of quectel-CM is shown as below (example of EC20 & GobiNet):

```
# quectel-CM –s ctnet &
[01-01_00:26:45:355] Quectel_ConnectManager_SR01A01V10
[01-01_00:26:45:356] ./quectel-CM profile = ctnet///, pincode =
[01-01_00:26:45:357] Find qmichannel = /dev/qcqmi2
[01-01_00:26:45:358] Find usbnet_adapter = eth2
[01-01_00:26:45:368] Get clientWDS = 7
[01-01_00:26:45:400] Get clientDMS = 8
[01-01_00:26:45:432] Get clientNAS = 9
[01-01_00:26:45:464] Get clientWDA = 10
[01-01_00:26:45:496] requestBaseBandVersion EC20CQAR02A03E2G_BETA0914 1 [Sep 14
2015 13:51:27]
[01-01_00:26:45:560] requestGetSIMStatus SIMStatus: SIM_READY
[01-01_00:26:45:624] requestGetProfile ctnet///0
[01-01_00:26:45:656] requestRegistrationState MCC: 460, MNC: 11, PS: Attached, DataCap: LTE
[01-01_00:26:45:688] requestQueryDataCall ConnectionStatus: DISCONNECTED
[01-01_00:26:45:720] requestRegistrationState MCC: 460, MNC: 11, PS: Attached, DataCap: LTE
[01-01_00:26:45:752] requestQueryDataCall ConnectionStatus: DISCONNECTED
[01-01_00:26:45:816] requestSetupDataCall WdsConnectionIPv4Handle: 0x43cc4478
[01-01_00:26:45:912] requestQueryDataCall ConnectionStatus: CONNECTED
[01-01_00:26:45:937] udhcpc (v1.20.2) started
[01-01_00:26:45:956] Sending discover...
[01-01_00:26:45:960] Sending select for 10.172.27.151...
[01-01_00:26:45:964] Lease of 10.172.27.151 obtained, lease time 7200
[01-01_00:26:45:984] deleting routers
route: SIOCDELRT: No such process
[01-01_00:26:46:003] adding dns 61.132.163.68
[01-01_00:26:46:003] adding dns 202.102.213.68
```

Step 4: Use the following commands to check IP/DNS/Route.

```
# ifconfig eth2
eth2      Link encap:Ethernet  HWaddr D2:B6:0C:28:AA:C6
          inet addr:10.172.27.151  Bcast:10.172.27.159  Mask:255.255.255.240
```

```
          inet6 addr: fe80::d0b6:cff:fe28:aac6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1224 (1.1 KiB)  TX bytes:1960 (1.9 KiB)

# cat /etc/resolv.conf
nameserver 61.132.163.68
nameserver 202.102.213.68

# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.172.27.145   0.0.0.0         UG    0      0        0 eth2
10.172.27.144   0.0.0.0         255.255.255.240 U     0      0        0 eth2

# ping www.baidu.com
PING www.a.shifen.com (115.239.211.112) 56(84) bytes of data.
64 bytes from 115.239.211.112: icmp_seq=1 ttl=53 time=24.8 ms
```

Step 5: Use the following command to terminate quectel-CM process to disconnect data connection:

```
# killall quectel-CM
[01-01_00:32:11:341] requestDeactivateDefaultPDP err = 0
[01-01_00:32:11:544] GobiNetThread exit
[01-01_00:32:11:545] main exit
```

5.4. Test "AT$QCRMCALL" on GobiNet or QMI WWAN

If customers use UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 and using GobiNet or QMI WWAN driver, and requiring **AT$QCRMCALL** to setup data call, please read this section for details. Otherwise, please skip this section.

Although it is recommended to use QMI tools like quectel-CM/libqmi/uqmi to setup data call, but some customers prefer **AT$QCRMCALL**.

And if the customer's MCU's USB Host Controller do not full support USB Interrupt type Endpoint. It has to use **AT$QCRMCALL** instead of QMI tools.

For GobiNet Driver, in order to use **AT$QCRMCALL**, the customer need to modify "qcrmcall_mode" in GobiUSBNet.c to 1.

For QMI WWAN Driver, no extra modifies is required.

Next log shows how to use "AT$QCRMCALL" to setup data call.
For details, please refer to <<ATC_QCRMCALL_QNETDEVSTATUS_V1.0.pdf>>.

```
root@imx6qdlsabresd:~# busybox microcom /dev/ttyUSB2
at+csq;+cgreg?;+cops?
+CSQ: 27,99
+CGREG: 0,1
+COPS: 0,0,"CHINA MOBILE",7
OK

AT$QCRMCALL=1,1
$QCRMCALL: 1,V4
OK

AT+QNETDEVSTATUS?
+QNETDEVSTATUS: 0,1,4,1
OK

root@imx6qdlsabresd:~# busybox udhcpc -fnq -i wwan0
udhcpc (v1.24.1) started
Sending discover...
Sending select for 10.166.47.120...
Lease of 10.166.47.120 obtained, lease time 7200
/etc/udhcpc.d/50default: Adding DNS 211.138.180.2
/etc/udhcpc.d/50default: Adding DNS 211.138.180.3
root@imx6qdlsabresd:~#
```

5.5. Test ECM or MBIM

If customers use UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12 /EM12/EG16/EG18 and requiring ECM or MBIM driver, please read this section for details. Otherwise, please skip this section.

The USB Interface 4 of these modules can be configured as QMI/ECM/GobiNet/QMI WWAN types of USB Network devices.

The default USB NET Type of EM06/EM12 is MBIM. Other modules are GobiNet/QMI WWAN.

The Customer can query and set the current USB NET Type of the modules by **AT+QCFG="usbnet"**.

Table 4: USB NET Type

AT+QCFG="usbnet"	USB Driver
AT+QCFG="usbnet",0	GobiNet or QMI WWAN
AT+QCFG="usbnet",1	CONFIG_USB_NET_CDCETHER [KERNEL]*/drivers/net/usb/ cdc_ether.c*
AT+QCFG="usbnet",2	USB_NET_CDC_MBIM [KERNEL]*/drivers/net/usb/ cdc_mbim.c*

ECM and MBIM have own USB Interface Class defined by USB-IF. So, there is no need to insert Quectel modules' VID and PID to the source code files.

For ECM mode, the modules will auto setup data call internally, the customer just need call DHCP tools to obtain IP and DNS. And IPV4 address is style of 192.168.225.X.

For MBIM mode, the customer can use MBIM tools like mbimcli/umbim to setup data call.

5.6. Test QMAP on GobiNet or QMI WWAN

If customers use EC21/EC25/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 and using GobiNet or QMI WWAN Driver and requiring QMAP (Multiplexing and Aggregation protocol), please read this section for details. Otherwise, please skip this section.

When using GobiNet or QMI_WWAN, only one Physical Network Card can be created by default, so only one PDN data call can be set up. However, multiple virtual Network Cards can be created by using multiplexing protocol over one Physical Network card, and customers can setup multiple PDN data calls.

When using GobiNet or QMI_WWAN, only one IP Packet in one URB can be transferred, so when there are high throughput and frequent URB interrupts, the Host CPU will become overloaded. However, aggregation protocol can be used to transfer multiple IP Packets in one URB with increased throughput by reducing the number of URB interrupts.

If customers need multiplexing or aggregation protocol, please contact Quectel FAE.

6 FAQ and Kernel Log

6.1. How to Check Whether USB Driver Exists in the Module

USB driver can be checked from the content of directory */sys/bus/usb/drivers*. For example:

```
carl@carl-OptiPlex-7010:~$ ls /sys/bus/usb/drivers
cdc_acm cdc_wdm ftdi_sio GobiNet hub option qmi_wwan usb usbfs usbhid usbserial usbserial_generic
```

If USB serial driver is required, please make sure *option* exists. If CDC ACM driver is required, please make sure *cdc_acm* exists. If GobiNet driver is required, please make sure *GobiNet* exists. If QMI WWAN driver is required, please make sure *qmi_wwan* exists.

6.2. How to Check Whether the Module Works Well with the Corresponding USB Driver

This chapter shows the kernel log about the module attaching the corresponding USB driver in Linux. If the module does not work well, please compare the kernel log in the module with the kernel log in this chapter to help you troubleshoot.

1. For UC15/UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 with USB serial driver: Kernel logs of these modules are almost the same except for the VID&PID information (marked by read box in the following figure).

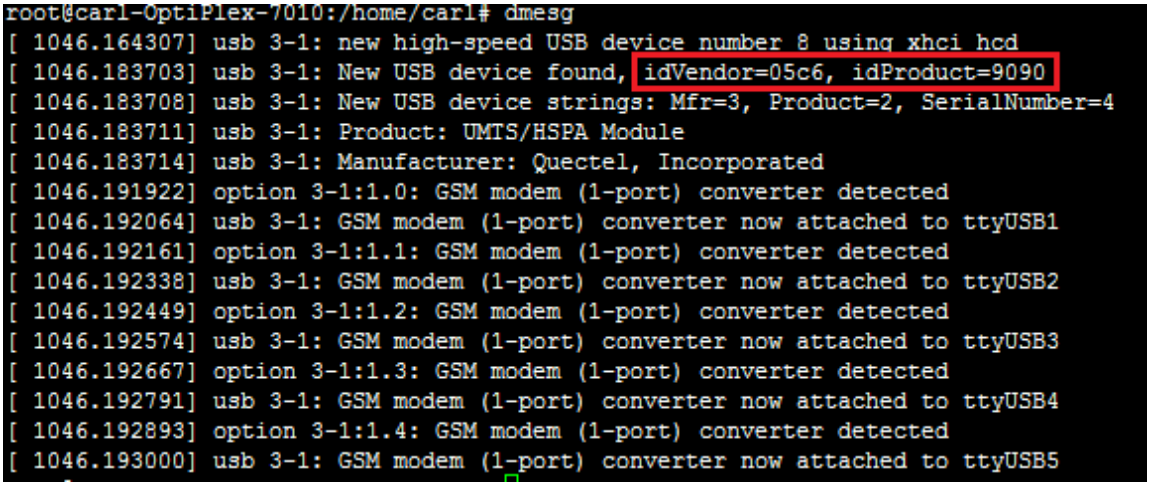

Figure 7: USB Serial for UC15

2. For UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 with USB serial and GobiNet driver: Kernel logs of these modules are almost the same except for the VID&PID information (marked by read box in the following figure).

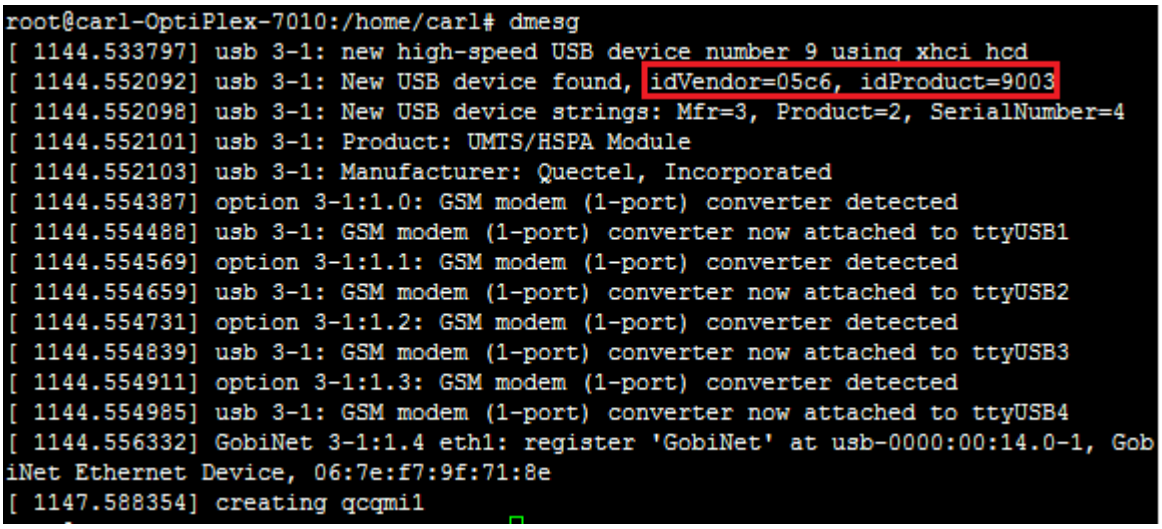

Figure 8: USB Serial and GobiNet for UC20

3. For UC20/EC20/EC21/EC25/EG91/EG95/BG96/AG35/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18 with USB serial and QMI WWAN driver: Kernel logs of these modules are almost the same except for the VID&PID information (marked by read box in the following figure).

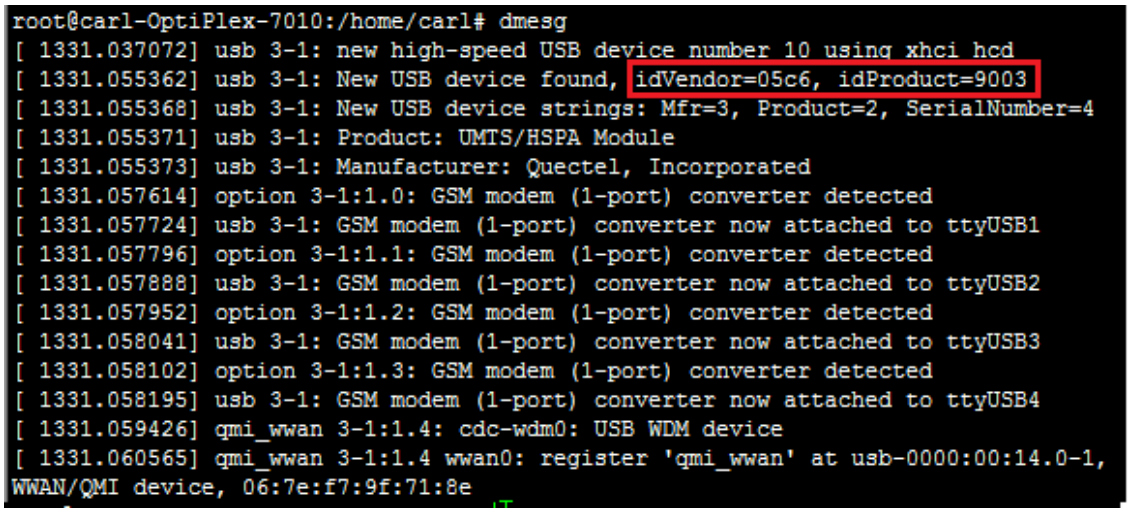

Figure 9: USB Serial and QMI WWAN for UC20

4. For UG95/UG96 with CDC ACM driver

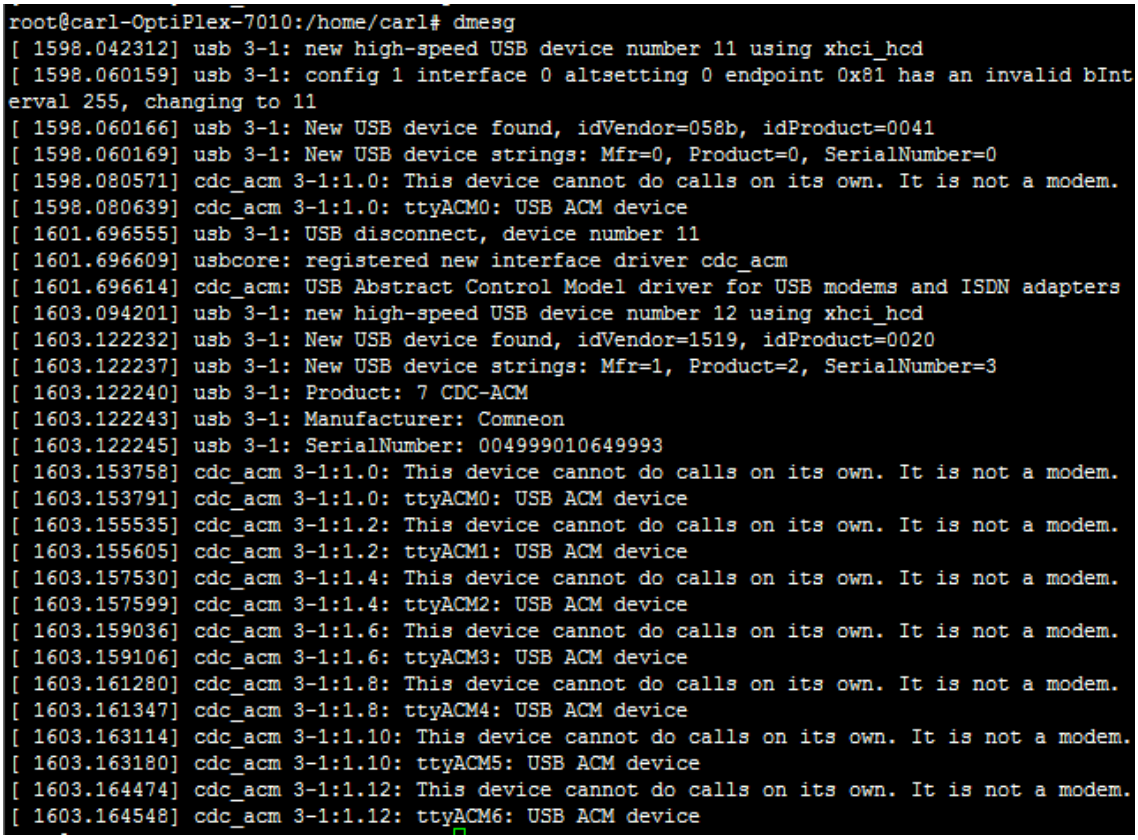

Figure 10: CDC ACM for UG95/UG96

7 Appendix A References

Table 3: Terms and Abbreviations

Abbreviations	Descriptions
ACM	Abstract Control Model
CDC	Communications Device Class
DNS	Domain Name System
NDIS	Network Driver Interface Specification
NMEA	National Marine Electronics Association
OS	Operating System
PC	Personal Computer
PID	Product ID
PPP	Point to Point Protocol
VID	Vendor ID
URB	USB Request Block
QMAP	QUALCOMM Multiplexing and Aggregation Protocol