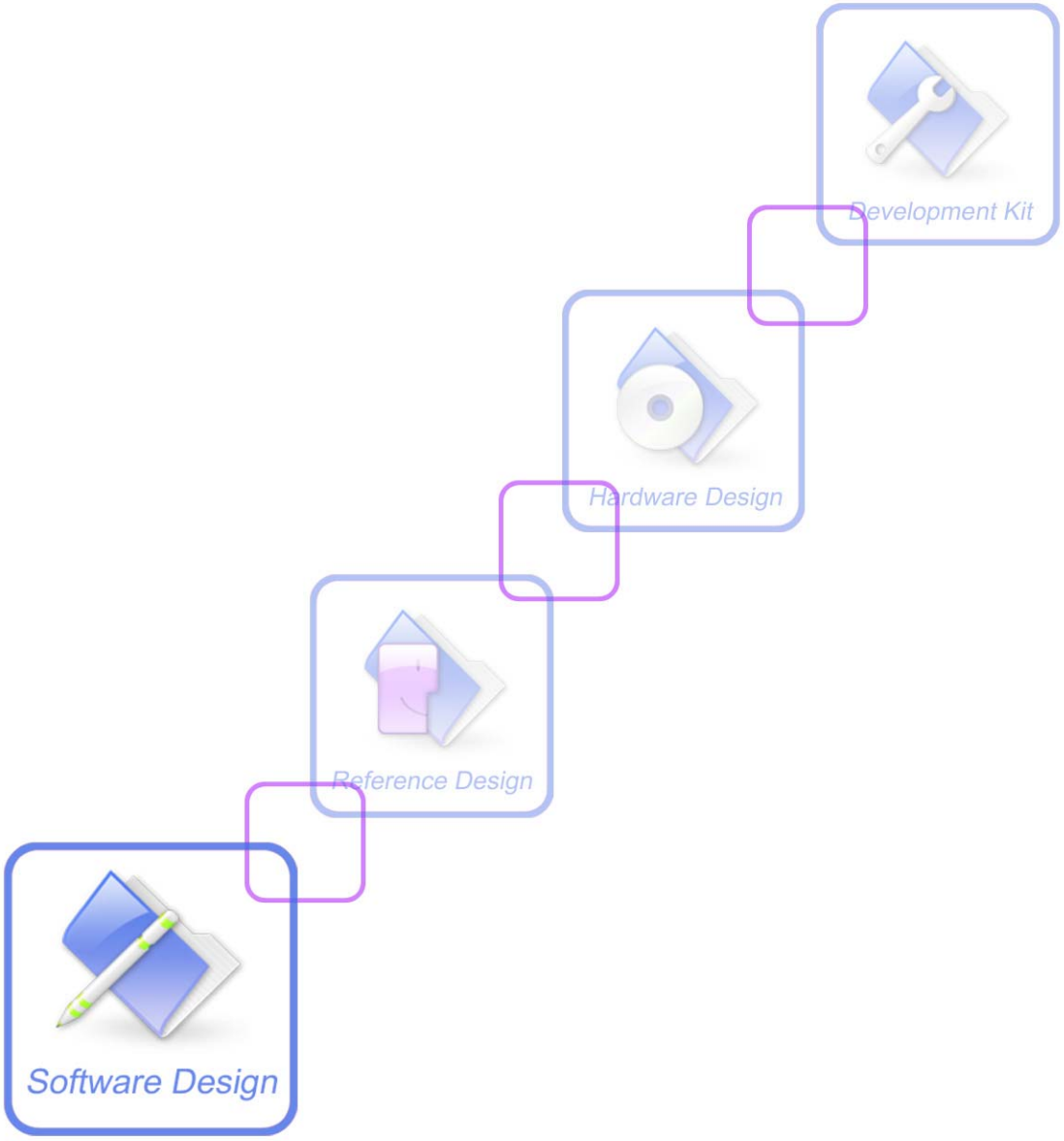
SIMCom
A company of SIM Tech

How to use Linux driver



Document Title:	How to use Linux driver
Version:	1.4
Date:	2013-10-28
Status:	Release
Author	aaron

General Notes

SIMCom offers this information as a service to its customers, to support application and engineering efforts that use the products designed by SIMCom. The information provided is based upon requirements specifically provided to SIMCom by the customers. SIMCom has not undertaken any independent search for additional relevant information, including any information that may be in the customer's possession. Furthermore, system validation of this product designed by SIMCom within a larger electronic system remains the responsibility of the customer or the customer's system integrator. All specifications supplied herein are subject to change.

Copyright

Version History

Version	Chapter	Author	Comments
V1.0	New Version	aaron	New version
V1.1	Chapter 2	aaron	We recommend to use the driver option instead of driver generic
V1.2	Chapter 1.1.3	aaron	Add flag for short packet transmission in some case
V1.3	SCOPE	aaron	Delete description about driver issued by SIMCOM
V1.4	Chapter 1.1.2	wjl	Update the VID/PID micro definition used in sample code

Contents

SCOPE

This document is a brief description on:

1. How to modify, build and use the driver on Linux issued by Linux kernel in order to use SIMCom devices.

1 Driver issued by Linux kernel

In fact the kernel with version of 2.6.20 and later has a common driver named usbserial which can also be used by SIMCom device.

Succeeding sections will use the kernel code of 2.6.35 as an example to depict how to modify, build and use kernel driver for SIMCom device in fail detail.

1.1 Modify the driver

One needs to add the vendor ID and product ID of SIMCom to kernel driver in order to support SIMCom device.

drivers\usb\serial\option.c:.

```
#define OLIVETTI_VENDOR_ID                      0x0b3c
#define OLIVETTI_PRODUCT_OLICARD100             0xc000

/*add by simcom*/
#define SIMCOM_WCDMA_VENDOR_ID                  0x05C6
#define SIMCOM_WCDMA_PRODUCT_ID                 0x9000
/*end by simcom*/

/* some devices interfaces need special handling due to a number of reasons */
enum option_blacklist_reason {
                OPTION_BLACKLIST_NONE = 0,
                OPTION_BLACKLIST_SENDSETUP = 1,
                OPTION_BLACKLIST_RESERVED_IF = 2
};

struct option_blacklist_info {
        const u32 infolen;      /* number of interface numbers on blacklist */
        const u8  *ifaceinfo;   /* pointer to the array holding the numbers */
        enum option_blacklist_reason reason;
};

static const u8 four_g_w14_no_sendsetup[] = { 0, 1 };
static const struct option_blacklist_info four_g_w14_blacklist = {
        .infolen = ARRAY_SIZE(four_g_w14_no_sendsetup),
        .ifaceinfo = four_g_w14_no_sendsetup,
        .reason = OPTION_BLACKLIST_SENDSETUP
};

static const struct usb_device_id option_ids[] = {
        { USB_DEVICE(SIMCOM_WCDMA_VENDOR_ID, SIMCOM_WCDMA_PRODUCT_ID) },   /*add by simcom*/
        { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_COLT) },
        { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA) },
        { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA_LIGHT) },
```

1.1.1 Support system suspend/resume

Add .reset_resume call-back function if kernel support, for some USB HOST controller issue a bus reset to USB devices when system resume, USB port will be unloaded, and loaded later, the reset_resume call-back funcion will avoid the port unloading when system resume, for more detail please refer to kernel USB driver documents`.

```
974 static struct usb_driver option_driver = {
975     .name       = "option",
976     .probe      = usb_serial_probe,
977     .disconnect = usb_serial_disconnect,
978 #ifdef CONFIG_PM
979     .suspend    = usb_serial_suspend,
980     .resume     = usb_serial_resume,
981     .supports_autosuspend = 1,
982 #endif
983     .id_table   = option_ids,
984     .no_dynamic_id =    1,
985 };
```

```
968 static struct usb_driver option_driver = {
969     .name       = "option",
970     .probe      = usb_serial_probe,
971     .disconnect = usb_serial_disconnect,
972 #ifdef CONFIG_PM
973     .suspend    = usb_serial_suspend,
974     .resume     = usb_serial_resume,
975     .reset_resume =     usb_serial_resume,
976     .supports_autosuspend = 1,
977 #endif
978     .id_table   = option_ids,
979     .no_dynamic_id =    1,
980 };
```

1.1.2 Support low power mode

For kernel 2.6.36, add the follow highlight code to end of option_probe function:

```
1070   /* Don't bind network interfaces on Huawei K3765 & K4505 */
1071   if (serial->dev->descriptor.idVendor == HUAWEI_VENDOR_ID &&
1072     (serial->dev->descriptor.idProduct == HUAWEI_PRODUCT_K3765 ||
1073       serial->dev->descriptor.idProduct == HUAWEI_PRODUCT_K4505) &&
1074     serial->interface->cur_altsetting->desc.bInterfaceNumber == 1)
1075     return -ENODEV;
1076
1077   if (serial->dev->descriptor.idVendor == SIMCOM_WCDMA_VENDOR_ID &&
1078       serial->dev->descriptor.idProduct == SIMCOM_WCDMA_PRODUCT_ID)
1079   {
1080 #ifdef CONFIG_PM
1081     serial->interface->needs_remote_wakeup = 1        /* autosuspend (15s delay) */
1082     device_init_wakeup(&serial->interface->dev, 1);
1083     serial->dev->autosuspend_delay = 15 * HZ;   /* for kernel 2.6.36 */
1084     usb_enable_autosuspend(serial->dev);
1085 #endif /* CONFIG_PM */
1086   }
1087
1088   data = serial->private = kzalloc(sizeof(struct usb_wwan_intf_private), GFP_KERNEL);
1089
1090   if (!data)
1091     return -ENOMEM;
1092   data->send_setup = option_send_setup;
1093   spin_lock_init(&data->susp_lock);
1094   data->private = (void *)id->driver_info;
1095   return 0;
1096 }
```

For kernel 2.6.38, add the follow highlight code to end of option_probe function:

```
1070   /* Don't bind network interfaces on Huawei K3765 & K4505 */
1071   if (serial->dev->descriptor.idVendor == HUAWEI_VENDOR_ID &&
1072     (serial->dev->descriptor.idProduct == HUAWEI_PRODUCT_K3765 ||
1073       serial->dev->descriptor.idProduct == HUAWEI_PRODUCT_K4505) &&
1074     serial->interface->cur_altsetting->desc.bInterfaceNumber == 1)
1075     return -ENODEV;
1076
1077   if (serial->dev->descriptor.idVendor == SIMCOM_WCDMA_VENDOR_ID &&
1078       serial->dev->descriptor.idProduct == SIMCOM_WCDMA_PRODUCT_ID)
1079   {
1080 #ifdef CONFIG_PM
1081     pm_runtime_set_autosuspend_delay(&serial->dev, 12 * 1000);    /*  for kernel 2.6.38 and above */
1082     usb_enable_autosuspend(serial->dev);
1083 #endif /* CONFIG_PM */
1084   }
1085
1086   data = serial->private = kzalloc(sizeof(struct usb_wwan_intf_private), GFP_KERNEL);
1087
1088   if (!data)
1089     return -ENOMEM;
1090   data->send_setup = option_send_setup;
1091   spin_lock_init(&data->susp_lock);
1092   data->private = (void *)id->driver_info;
1093   return 0;
1094 }
```

1.1.3 Add short packet flag

Since the max packet size of BULK endpoint on SIMCOM module in High USB speed

is 512 bytes, in Full USB speed is 64 bytes, in addition the USB protocol says :

> An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's reported *wMaxPacketSize* value. When a bulk IRP involves more data than can fit in one maximum-sized data payload, all data payloads are required to be maximum size except for the last data payload, which will contain the remaining data. A bulk transfer is complete when the endpoint does one of the following:
>
> - Has transferred exactly the amount of data expected
> - Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet
>
> When a bulk transfer is complete, the Host Controller retires the current IRP and advances to the next IRP. If a data payload is received that is larger than expected, all pending bulk IRPs for that endpoint will be aborted/retired.

So one needs to send an zero-length packet additional if one wants to transmit the data stream with length exactly multiple of wMaxPacketSize.

Fortunately one needs not to send zero packet manually, one only needs to modify a little driver code:

drivers\usb\serial\usb_wwan.c:

```
/* Setup urbs */
static void usb_wwan_setup_urbs(struct usb_serial *serial)
{
	int i, j;
	struct usb_serial_port *port;
	struct usb_wwan_port_private *portdata;

	dbg("%s", __func__);

	for (i = 0; i < serial->num_ports; i++) {
		port = serial->port[i];
		portdata = usb_get_serial_port_data(port);

		/* Do indat endpoints first */
		for (j = 0; j < N_IN_URB; ++j) {
			portdata->in_urbs[j] = usb_wwan_setup_urb(serial,
					port->
					bulk_in_endpointAddress,
					USB_DIR_IN,
					port,
					portdata->
					in_buffer[j],
					IN_BUFLEN,
					usb_wwan_indat_callback);
		}

		/* outdat endpoints */
		for (j = 0; j < N_OUT_URB; ++j) {
			portdata->out_urbs[j] = usb_wwan_setup_urb(serial,
					port->
					bulk_out_endpointAddress,
					USB_DIR_OUT,
					port,
					portdata->
					out_buffer
					[j],
					OUT_BUFLEN,
					usb_wwan_outdat_callback);

			portdata->out_urbs[j]->transfer_flags |= URB_ZERO_PACKET; //add by simcom
		}
	} ? end for i=0;i<serial->num_por... ?
} ? end usb_wwan_setup_urbs ?
```

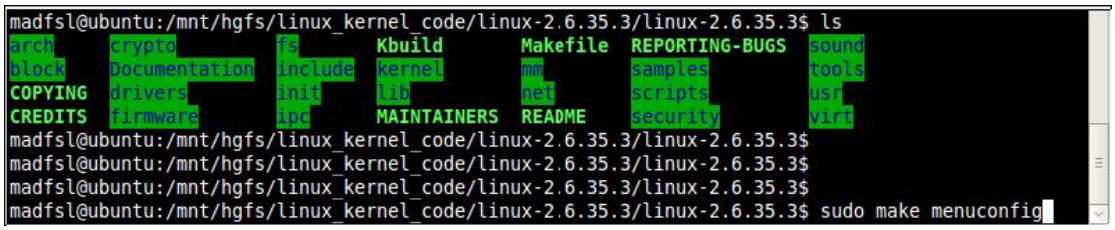
NOTE: This modification is only for the driver option.ko

1.2 Build the driver

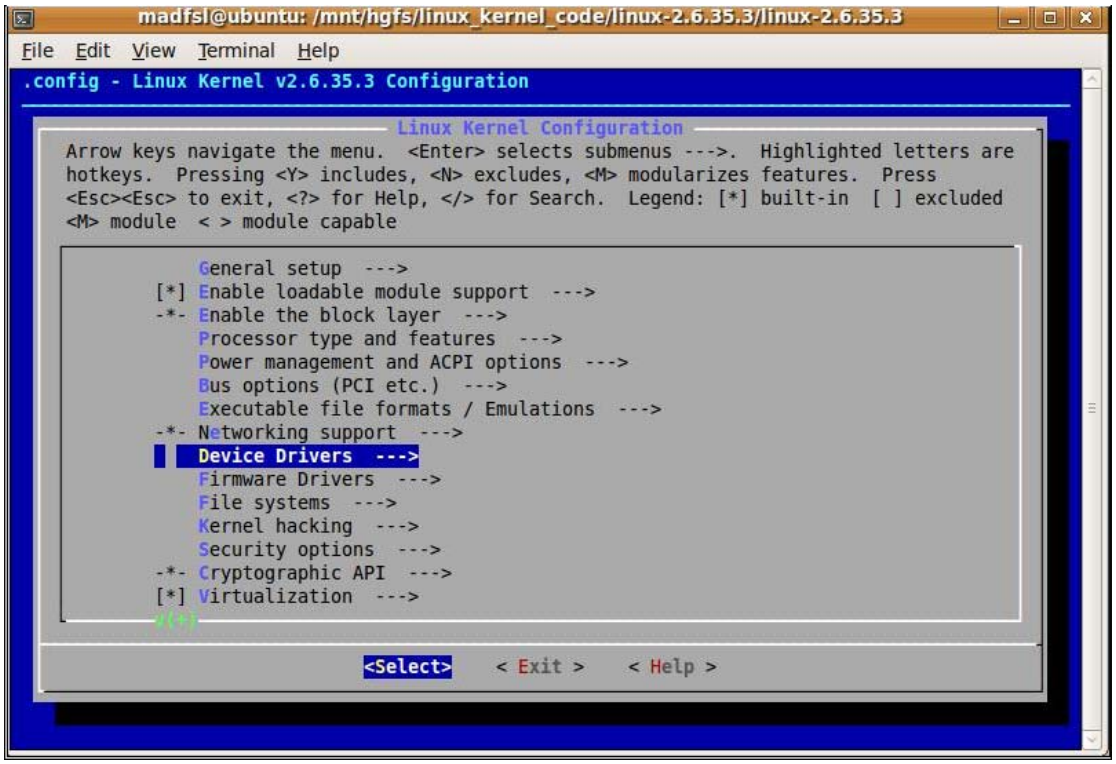
One needs to setup the kernel development environment first which include kernel source code and cross compiler environment.

Following is a step-by-step instruction on how to build the driver into kernel.

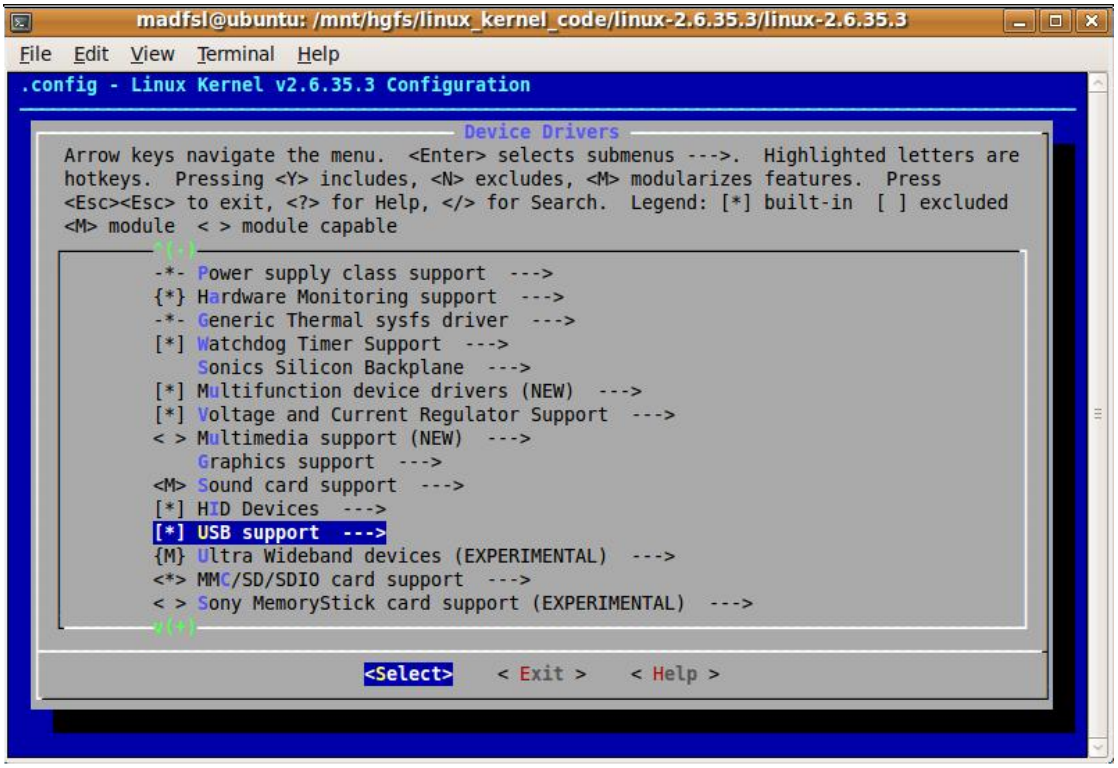
1) Use “sudo make menuconfig” to configure the kernel.



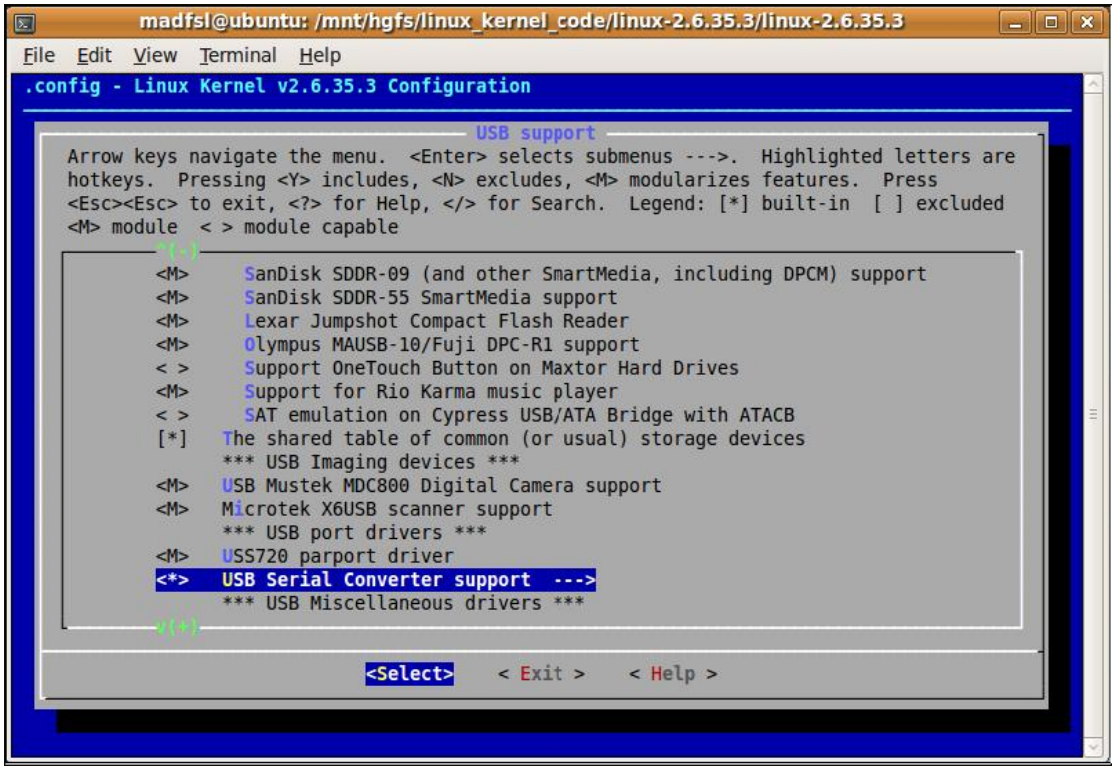
2) Enter into menu “Device Drivers”



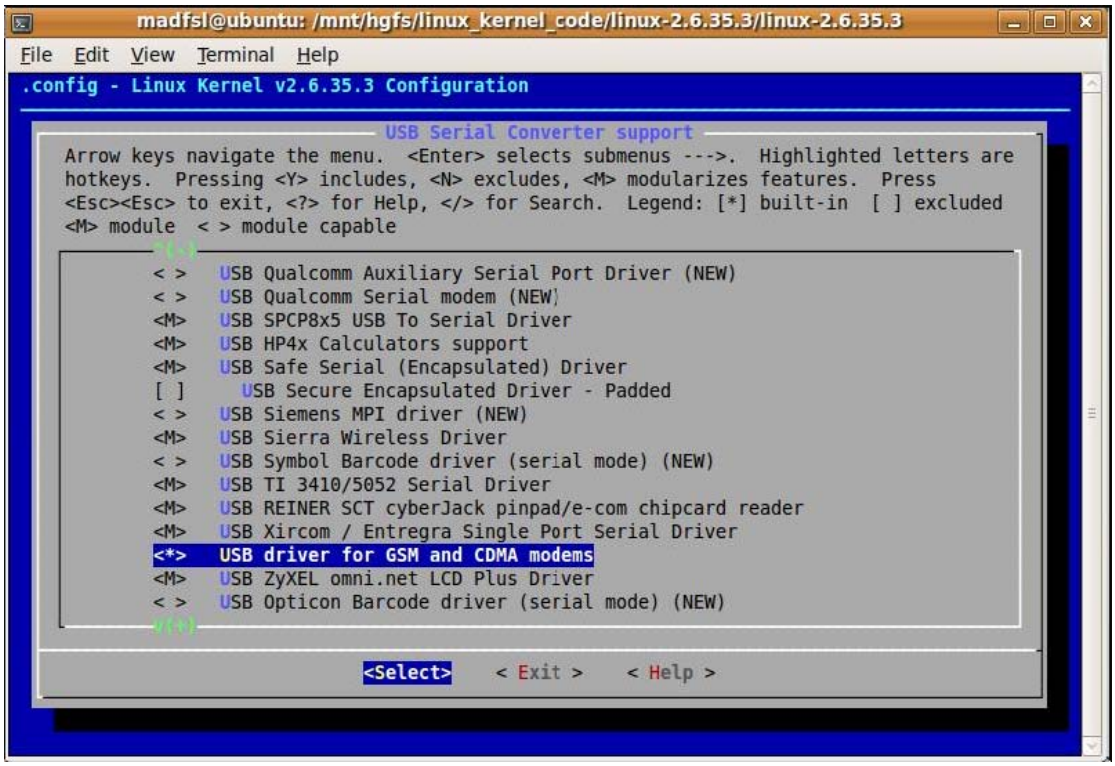
3) Continue enter into menu “USB support”



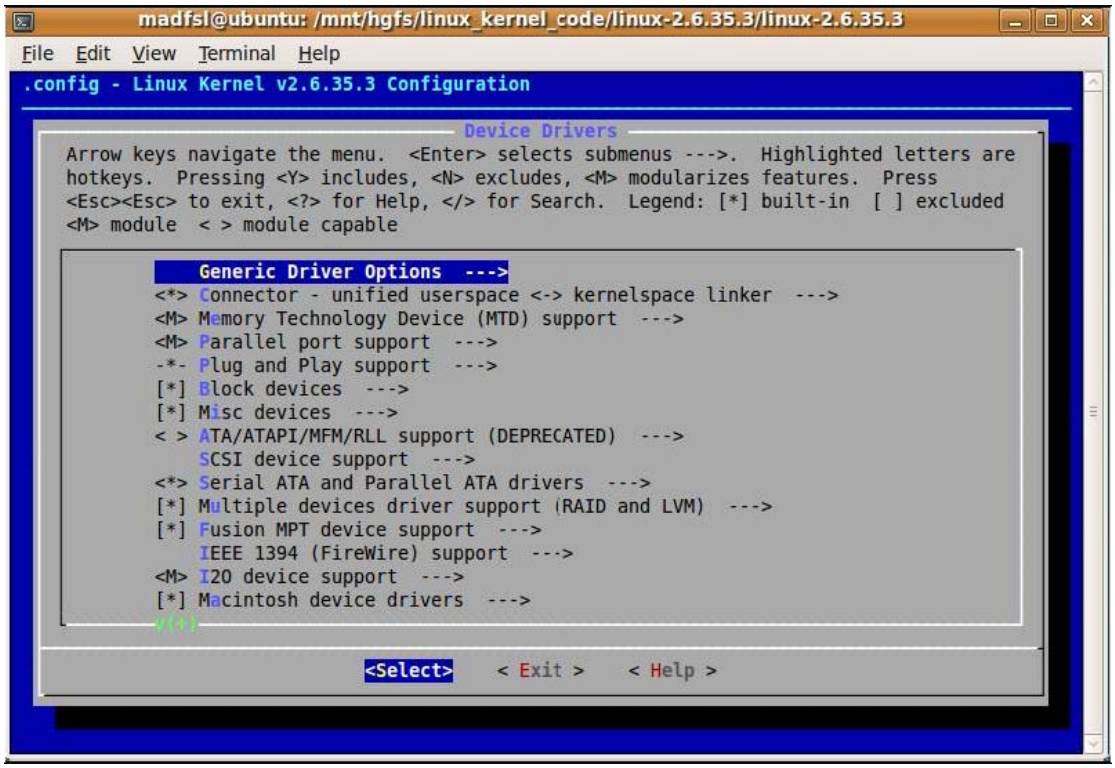
4) Continue enter into menu “USB Serial Converter support”



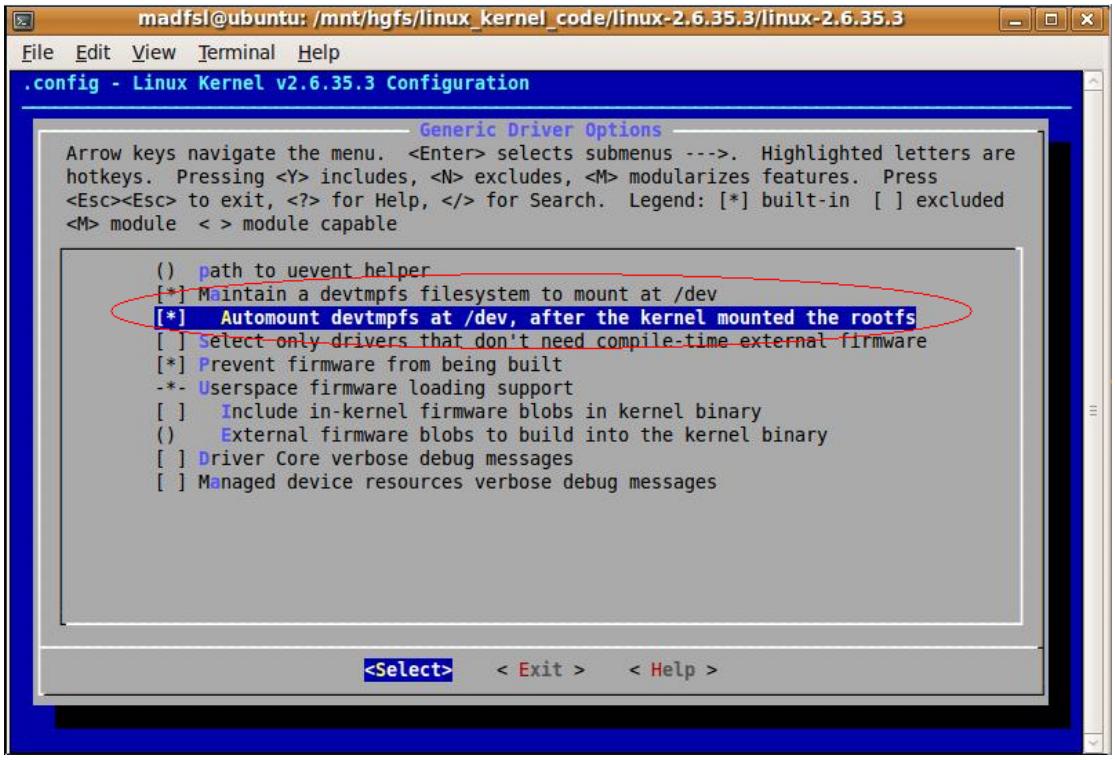
5) Type “y” to select menu “USB driver for GSM and CDMA modems”, of course one can type “m” to compile the driver as a module.



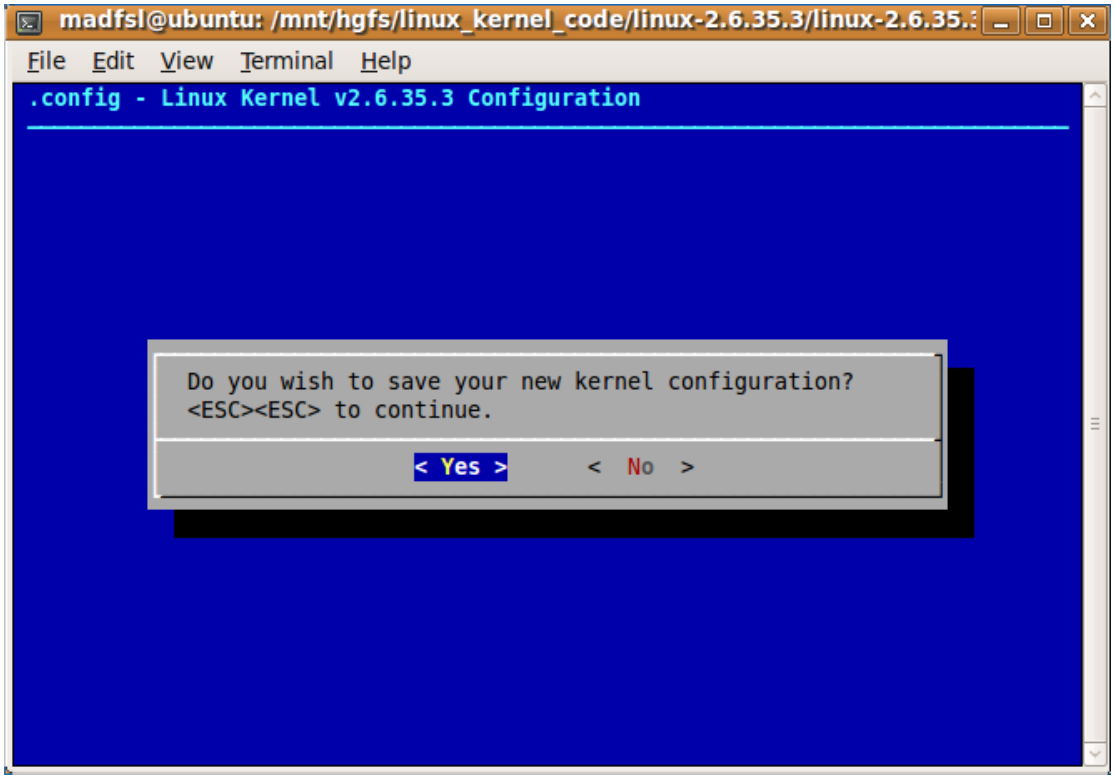
6) Some other options need to be configured, so please enter into menu "Device Drivers -> Generic Driver Options"



7) Type "y" to select the following two options.



8) Exit and save the configuration.



After configuration, these items will be configured:

CONFIG_USB = y

CONFIG_USB_SERIAL=y

CONFIG_USB_SERIAL_OPTION=y

CONFIG_DEVTMPFS=y

CONFIG_DEVTMPFS_MOUNT=y

2) Use “sudo make” to compile the kernel or use “sudo make modules” to compile the driver as a module

```
madfsl@ubuntu:/mnt/hgfs/linux_kernel_code/linux-2.6.35.3/linux-2.6.35.3$ sudo ma
ke
[sudo] password for madfsl:
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/x86/Kconfig
```

1.3 Use the driver

As you move through this chapter new kernel firmware or new driver: option.ko(compiled as module) is ready.

1.3.1 Install the driver(driver as module only)

If one compiles the driver as a module one needs to install it first. one can use the following command to install the driver:

modprobe option.ko

This command will install all the needed drivers.

```
USB-Serial-COM4
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ ls
option.ko     usb_wwan.ko   usbserial.ko
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ mod
probe option.ko
usbcore: registered new interface driver usbserial
usbserial: USB Serial Driver core
USB Serial support registered for GSM modem (1-port)
usbcore: registered new interface driver option
option: v0.7.2:USB Driver for GSM modems
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$
```

If all right the driver will be installed to the system, one can use the following command to query the result:

lsmod |grep option

```
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ lsm
od |grep option
option                 12548  0
usb_wwan                7381  1 option
usbserial              23430  2 option,usb_wwan
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$
```

Note: this installation procedure is invalid when rebooting the system, so if one wants to install the driver automatically when starting the system, one should better put the installation instruction to the startup script.

1.3.2 Use the driver

After the driver installed one can use SIMCom device via the driver, now plug the SIMCom device to the host device via USB connector, and if the device is identified by the driver, 5 device files named ttyUSB0, ttyUSB1, ttyUSB2, ttyUSB3 and ttyUSB4 will be created in directory /dev

The relationship between the device files and SIMCom composite device is like this:

Device file	SIMCom composite device
ttyUSB0	DIAG interface
ttyUSB1	NMEA interface
ttyUSB2	ATCOM interface
ttyUSB3	MODEM interface
ttyUSB4	Wireless Ethernet Adapter interface

SIMCom device is plugged in:

```
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ usb 2-1: new full speed USB device us
ing fsl-ehci and address 2
option 2-1:1.0: GSM modem (1-port) converter detected
usb 2-1: GSM modem (1-port) converter now attached to ttyUSB0
option 2-1:1.1: GSM modem (1-port) converter detected
usb 2-1: GSM modem (1-port) converter now attached to ttyUSB1
option 2-1:1.2: GSM modem (1-port) converter detected
usb 2-1: GSM modem (1-port) converter now attached to ttyUSB2
option 2-1:1.3: GSM modem (1-port) converter detected
usb 2-1: GSM modem (1-port) converter now attached to ttyUSB3
option 2-1:1.4: GSM modem (1-port) converter detected
usb 2-1: GSM modem (1-port) converter now attached to ttyUSB4
```

Device files are created:

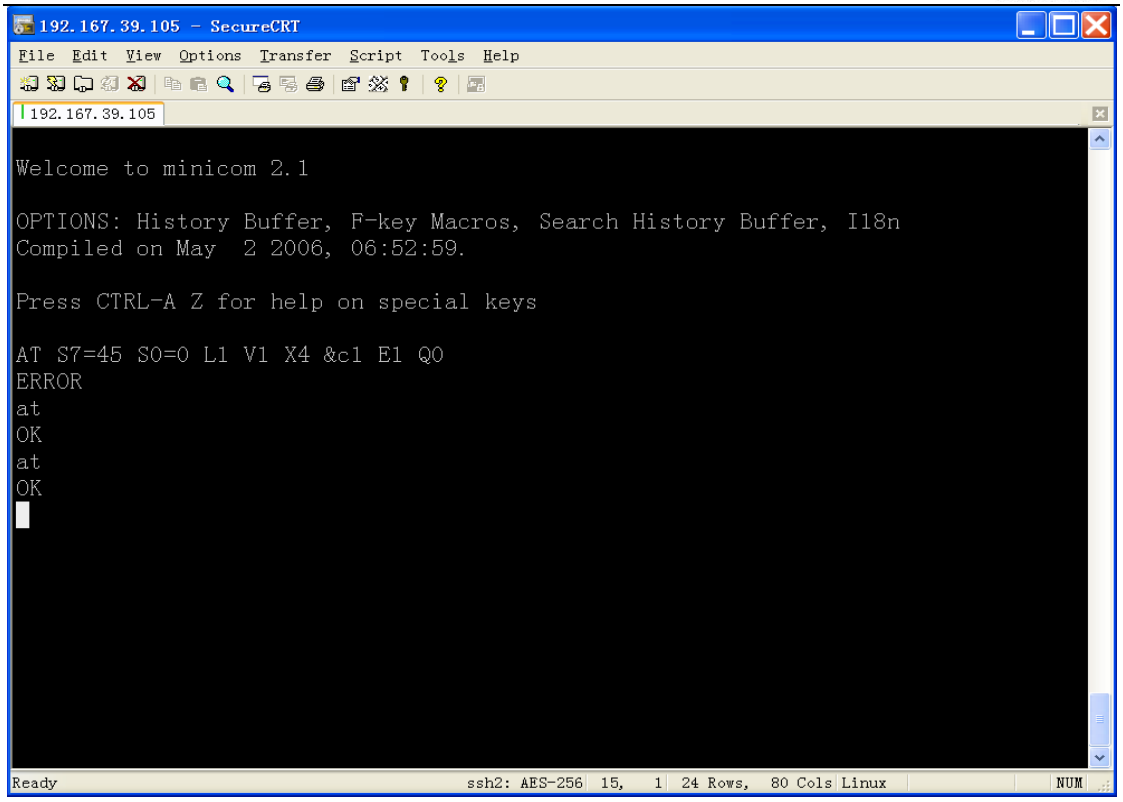
```
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ ls
/dev |grep USB
ttyUSB0
ttyUSB1
ttyUSB2
ttyUSB3
ttyUSB4
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$
```

NOTE:

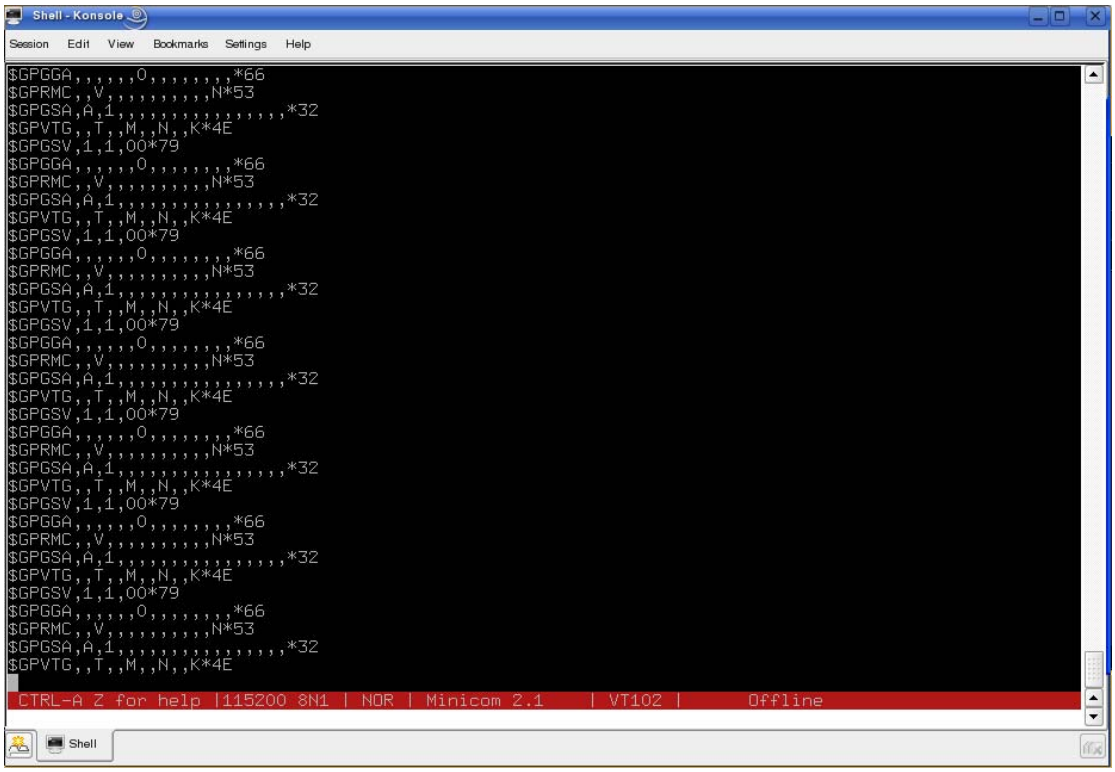
1 In some composite devices of SIMCom not all of the interfaces are existed, so the relationship is dynamic.

2 Only the NMEA, ATCOM and MODEM interface can be worked correctly with this driver.

If one gets the device files ready one can use tools such as minicom, wvdial etc to use the device。



ATCOM interface



NMEA interface

1.3.3 Remove the driver

One can use the following command to uninstall the driver:

rmmod option

```
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ rmm
od option.ko
usbcore: deregistering interface driver option
option: option_instat_callback: error -108
option1 ttyUSB4: GSM modem (1-port) converter now disconnected from ttyUSB4
option 2-1:1.4: device disconnected
option: option_instat_callback: error -108
option1 ttyUSB3: GSM modem (1-port) converter now disconnected from ttyUSB3
option 2-1:1.3: device disconnected
option1 ttyUSB2: GSM modem (1-port) converter now disconnected from ttyUSB2
option 2-1:1.2: device disconnected
option1 ttyUSB1: GSM modem (1-port) converter now disconnected from ttyUSB1
option 2-1:1.1: device disconnected
option1 ttyUSB0: GSM modem (1-port) converter now disconnected from ttyUSB0
option 2-1:1.0: device disconnected
USB Serial deregistering driver GSM modem (1-port)
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$
```

After removed one can use “lsmod |grep option” to check if the driver has been removed correctly.

Note: when removing the driver one must disconnect the device and close all the tools using the device first.