CHAPTER 12

Using an LTE Modem

Chapter 5 discussed how to use WiFi and Ethernet network adaptors to make Intel Galileo boards to connect to other devices and the Internet.

These adaptors are perfect if you need a stationary project like a board that controls your house or if you a building a mobile project that's limited to your WiFi coverage like a mini-robot that vacuums your floors.

What about building a drone that can fly long distances or a robot that walks around and monitors your neighborhood?

In these cases, Ethernet and WiFi adaptors are not applicable; you will need an efficient method to control your project remotely with a good coverage.

One possible solution is to use a modem that interfaces with Intel Galileo boards. Once the modem is properly configured to work with Intel Galileo and the connection between modem and carriers is established, the data exchange occurs like any other network adaptor.

Therefore, this chapter shows:

- How to physically connect an LTE modem to Intel Galileo.
- How to make Intel Galileo communicate with and configure the modem to camp in your preferable carrier provider.
- How to create a network adaptor in the Linux userspace context to make the data change possible.

This chapter contains the most expensive project of this book; LTE modems usually cost more than $100, but the same procedures used for the LTE modem can be used in 3G modems, which are more affordable (in the order of $65). If you have a 3G modem, the procedures mentioned in this chapter are applicable as well. The only difference is that you need to know the configuration required by your carrier, like the Access Point Names (APNs). You also need to have a data plan attributed to the SIM card that will be used in the modem tests.

This project teaches you how to create the network interface provided by the modem work and apply it to your project.

The LTE modem used in this chapter is module Intel XMM7160. You can also use Intel XMM7260 as well as any other LTE modem that is compliant with the 3GPP specifications. The 3GPP TS 27.007 V12.5.0 (2014-06) specifications are available from `http://www.3gpp.org/DynaReport/27007.htm`.

An Introduction to XMM7160 and XMM7260

The XMM 7160 and XMM 7260 modems were the first low-power consumption LTE modems with hardware and software entirely developed by Intel. Both modems are LTE (4G), and their differences are explained in Table 12-1.

Table 12-1. *Differences between the XMM 7160 and XMM 7260 Modems*

Feature	XMM 7160	XMM 7260
Launch year	2013	2014
LTE category	4	6
Peak downlink speed (downstream)	150 Mbits/s	300 Mbits/s
Supports 2G(GSM/Edge)	Yes	Yes
3G(HSPA+)	Yes	Yes
4G (LTE)	Yes	Yes
Bands supported	Quadband GSM/EDGE 8-band WCDMA 15-band LTE	Quadband GSM/EDGE 8-band WCDMA 21-band LTE

The modems are part of a capsulated module that provides mini-PCIe (mPCIe) or NGFF bus formats. If your laptop contains a 3G or 2G modem, there is a high probability of this modem being modular in one of the interfaces mentioned (mPCIe or NGFF) instead of built in, and you can replace your old 3G modem with a faster and great 4G modem like Intel XMM7160 or XMM7260. For this, you need to open your laptop as recommended by the service guide of your manufacturer and replace the modem manually. Once it's been replaced, you need to make sure you are using the right antennas and SIM card, and that the laptop has the Intel modem drivers properly installed. In this chapter, you learn how to integrate the modem with Intel Galileo boards.

The initial idea of XMM7160 and XMM7260 was to create a reference design modem that allows other manufacturers like Huawei and Foxconn to create their own modems based on Intel SoCs. However, there are laptops in the market using the XMM7160 or XMM7260 as end modems.

This chapter does not require you to use Intel modems; you can use any modem 2G/3G/4G that supports standards AT commands and can communicate using the cdc-acm interface, which will be explained later.

In summary, the Intel XMM modem was used in this chapter because it is a reference design and it works as a common model independently of a specific vendor. The AT commands discussed will work on any modem that uses Intel XMM as reference or on any modem that's compliant with 3GPP specifications.

Project Details

The interface used to communicate with modems is typically serial and AT commands are sent to the modem in order to set up the carrier configuration, search for a network signal, open the data channel, and establish the connection with other devices.

At the end of this configuration and connection process, the modem will provide a new interface to the Intel Galileo boards with an IP address, and all communication with the Internet and other devices is done through this new interface. In other words, the new IP behaves like any other adaptor, such as WiFi or Ethernet.

This project connects the modem to Intel Galileo using a USB OTG adaptor with a SIM card slot. The modem antennas are also selected in order to make sure it is possible to camp in the 4G network. Once the hardware configuration is ready, you must read the correct device driver in the Linux context in order to allow communication with the modem using a serial port.

With the modem device driver properly loaded and the serial port ready, a series of AT commands is sent to the modem in order to configure the modem with the correct APN, check the SIM card connection, and camp the modem in the network, thereby opening a data channel.

Finally, a PPP connection is established and an IP number that proves Internet access is provided.

Materials List

This project requires an LTE modem, a pair of antennas compatible with the modem and the LTE bands, a SIM card related to your carrier, and more, as listed in Table 12-2. Table 12-3 lists the optional materials.

Table 12-2. *Required Project Materials*

Quantity	Components
1	LTE modem card with mPCIe or NGFF bus. Intel XMM 7160 or Intel XMM 7260 is recommended.
1	USB OTG mPCIe or NGFF adaptor with SIM card slot (generic).
1	NGFF to mPCIe or mPCIe to NGFF adaptor (only if the OTG-USB adaptor contains an incompatible bus with the modem module used; read “The NGFF/mPCIe Adaptor”).
2	LTE/UMTS antennas with SMA male connector (part number TG.30.8111) Apex Taoglas or GA-107 Taoglas (part number GA.107.201111).
1	Micro-SIM card with data plan already set (depends on your carrier/plan).
1	OTG-USB 2.0 adaptor with micro-USB male to USB A female, only if your board is Intel Galileo instead of Intel Galileo Gen 2. This is for physical connection only because OTG-USB is not supported in this case.
2	SMA bulkhead female/jack to IPEX MHF (part number 071113-04).
1	Serial debugger cable (FTDI if your board is Intel Galileo Gen2 or audio jack serial if it is Intel Galileo).

Table 12-3. *Optional Materials for Incompatible Form Factors*

Quantity	Components
1	NGFF to mPCIe or mPCIe to NGFF adaptor if your modem form factor does not match the connector type provided by the USB-OTG adaptor.
1	Ethernet cable or WiFi mPCIe if you want more than one terminal for debugging using an SSH connection.
1	Nano-to-micro-SIM card adaptor if your SIM card is not micro-SIM.
1	Scissors and tape or needle-nosed pliers (read "Connecting the Modem Card").

You might ask if Intel Galileo boards provide an mPCIe connector, why you simply do not connect the modem directly to Intel Galileo's mPCIe bus. Intel Galileo boards do not provide a built-in interface that allows you to read the SIM card and, without communication, the SIM card can't camp the modem in the LTE network.

Thus, the USB-OTG adaptor is necessary. It also must be able to provide a connector to your modem. Sometimes it is difficult to find this kind of adaptor that matches your modem. If your OTG-USB connection does not offer the same connection bus of your modem, you must include a second adaptor in the project in order to connect the modem to the OTG-USB adaptor.

The following conditions require extra materials:

- If your OTG USB adaptor with the SIM card slot contains an incompatible connector with your modem card. You will need a NGFF to mPCIe or mPCIe to NGFF adaptor.
- If your OTG USB adaptor contains a SIM card slot that's incompatible with your SIM card format. For example, if your OTG USB adaptor accepts micro-SIM cards, but your SIM card is nano, you will need a nano-SIM to micro-SIM adaptor.
- If you want to debug the modem with more than one shell, you might also need an Ethernet cable or a WiFi mPCIe card to open several shells using SSH.

Considerations Related to Antennas

There are several affordable antennas in the market; my preferable brand is Taoglas Limited. These antennas usually cost $10-20 and they are built in different form factors, including mounted in magnetic mountings that allow you to connect the antennas in any metallic structure, thereby avoiding holes, screws, or any other kind of apparatus.

This book recommends two LTE/UMTS antennas with an SMA male connector Apex Taoglas (part number TG.30.8111) or GA-107 Taoglas (part number GA.107.201111).

If you are using an LTE modem, the TG.30.8111 antenna is recommended. Note this antenna also offers GPS reception. That means if your modem supports a GPS sensor in the M2M module, you will be able to retrieve your location as well.

The XMM 7160 and XMM 7260 cards have a GPS module, but unfortunately, the AT commands are confidential and consequently I did not receive authorization to publish information about them in this book. If you are using a modem that is not LTE, I recommend the GA.107.201111 antenna, which includes magnetic mounting and reaches 3.5G camping in the best scenarios in 2.2GHz. It does not mean your LTE modem will not work with this antenna but the bands that you will camp will not provide the optimal download streaming.

The datasheets for both antennas are in the `datasheet` folder of this chapter. You also can find them at http://taoglas.com/images/product_images/original_images/TG.30.8111.pdf and http://www.taoglas.com/images/product_images/original_images/GA.107%20Magnetic%20Telematic%20Cellular%20Penta-band%20Antenna%20300410.pdf.

The cost of the recommended antennas is $12-16 each, plus shipping.

There are several other decent antennas with different form factors, but make sure the antennas can camp and get optimal performance.

Preparing the Hardware

This book uses the worst scenario as its example, in other words, it uses a XMM 7160 with an NGFF bus that's connected to an OTG-USB adaptor that accepts only mPCIe modems and a nano-SIM card that must be coupled to a micro-SIM card slot. Thus, a NGFF to mPCIe adaptor is included in this project and a nano-to-micro-SD card is used as well.

Of course, if your scenario is simpler, you don't need to follow all these steps.

Step 1: Preparing the SIM Card

The OTG-USB adaptor contains an mPCIe slot, as shown in Figure 12-1 and a micro-SIM card slot, as shown in Figure 12-2.

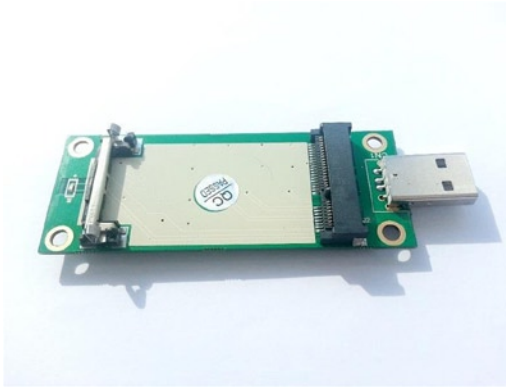

Figure 12-1. *mPCIe to OTG-USB adaptor, top view*

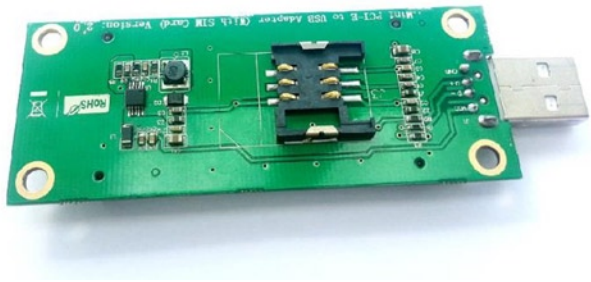

Figure 12-2. *mPCIe to OTG-USB adaptor, bottom view*

If your SIM card uses a micro-SIM form factor, simply insert your SIM in the bottom of the mPCIe-OTG adaptor. Otherwise, you need to use a SIM card adaptor to connect the SIM card to the mPCIe/OTG-USB adaptor. Figure 12-3 shows an example of a micro-SIM card adaptor used with a nano-SIM card.

Figure 12-3. *Nano-to-micro-SIM adaptor (1) and nano-SIM card (2)*

Just insert the nano-SIM and make sure the nano-SIM is completely tied and flat against the surface of the SIM adaptor. Then cut a piece of tape a little bit bigger than the SIM card adaptor and fasten it over the surface without contacts. Fold the tape into the borders. Figure 12-4 shows this procedure; note that the nano-SIM's contacts aren't covered by the tape.

Figure 12-4. *Inserting the nano-SIM and taping it to the adaptor*

Then insert the SIM adaptor to the mPCIe-OTG SIM card's slot, as shown in Figure 12-5.

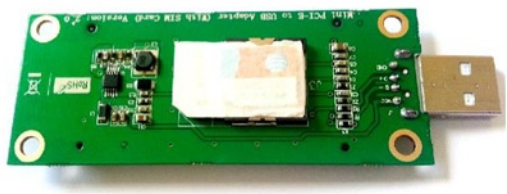

Figure 12-5. *Connecting the SIM card*

You need to make sure the nano-SIM's contacts are properly touching the contacts of the SIM card slot. Otherwise, you will receive errors when trying to read the SIM card.

Step 2: The NGFF/mPCIe Adaptor

If your modem card's bus is compatible with your mPCIe to OTG-USB adaptor, move to Step 3. This step is unnecessary if you don't need to use adaptors. This chapter covers the worst-case scenario, so the equipment used is a modem card with NGFF format and an mPCIe to OTG/USB adaptor that forces you to use an NGFF to mPCIe adaptor, as shown in Figure 12-6.

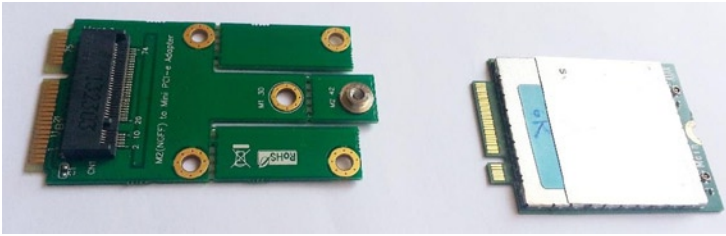

Figure 12-6. *NGFF to mPCIe adaptor (left) and modem card (right)*

Connecting both devices, you will have the configuration shown in Figure 12-7.

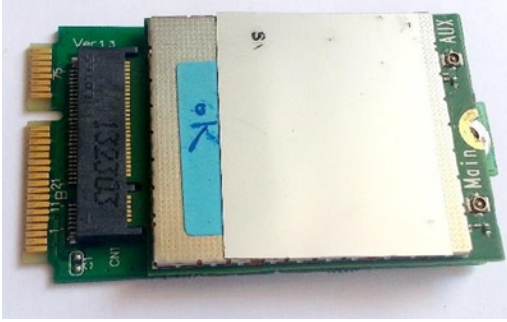

Figure 12-7. *Modem card connected to the NGFF to mPCIe adaptor*

Step 3: Adapting the Modem Card

The modem card that's directly connected to the NGFF to mPCIe adaptor is connected to the mPCIe to OTG-USB adaptor, as shown in Figure 12-8. The red arrow on this figure shows the locker mechanism. If your modem is not using a NGFF to mPCIe adaptor, you will be able to lock the modem to this locker mechanism without any problem.

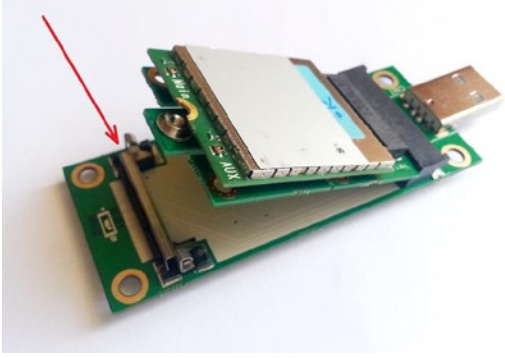

Figure 12-8. *Connecting the modem card to the mPCIe to OTG-USB adaptor*

However, if you are using the adaptor as described in the Step 2, there is a good chance that you will not be able to use the locker because the modem will not fit. In this case you have two options—remove the locker mechanism using needle-nose pliers and screw on the NGFF/mPCIe to the mPCIe/OTG-USB adaptor or simply tape the modem card, as shown in Figure 12-9.

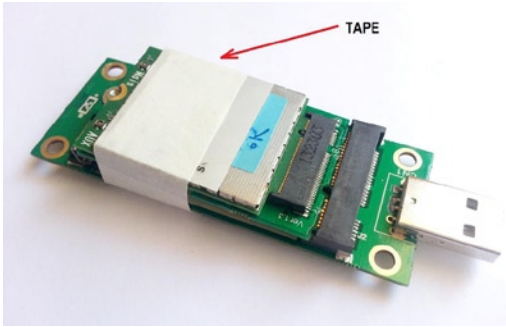

Figure 12-9. *Taping the modem card to the mPCIe to USB-OTG adaptor*

Step 4: Connecting the Antennas

The modem card is provided by a mini RF connector, more precisely specified as I-Pex 20449-001E. You need to use the SMA bulkhead cables female/jack to IPEX MHF in order to connect the modem to the antennas.

Figure 12-10 shows the cables and gives you an idea how fragile they are.

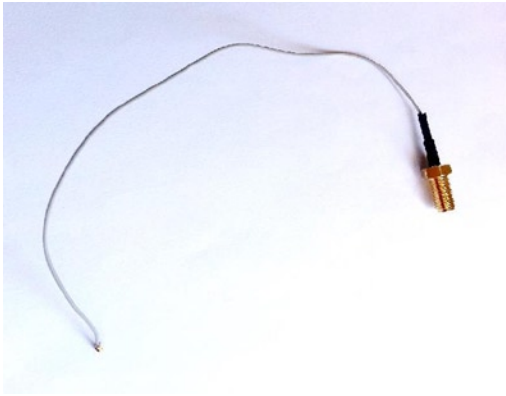

Figure 12-10. *SMA bulkhead female/jack to IPEX MHF*

Figure 12-11 shows the main antenna (MAIN) and the auxiliary (AUX) connected to the mini-RF connector.

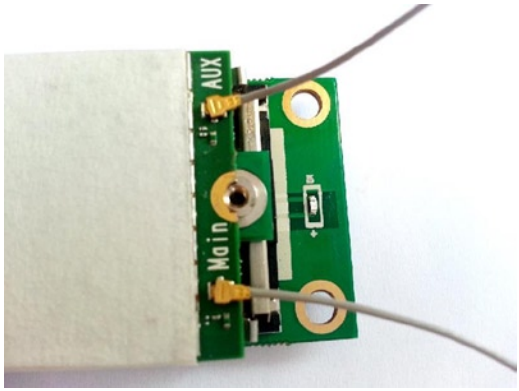

Figure 12-11. *Connecting the cable to the mini-RF connectors*

Using the SMA bulkhead cables female/jack IPEX MHF, you must connect to the antennas using its male SMA connectors, as shown in Figure 12-12.

Figure 12-12. *Connecting the antennas*

Step 5: Connecting the Modem to Intel Galileo

If your board is Intel Galileo Gen 2, you just need to connect the device to the OTG-USB connector, as shown in Figure 12-13.

Figure 12-13. *Connecting the devices to Intel Galileo Gen 2*

However, if you are using Intel Galileo only, you need the OTG-USB 2.0 adaptor with micro-USB male to USB A female to connect the devices.

Power on your board. If you assembled everything as expected, you are ready to start!

The next section explains how to prepare the software and confirm that everything is working properly.

Preparing the Software

These next steps guide you through the process of checking if the hardware is working and setting up the modem.

Step 1: Checking the Modem

Connect the proper cable to debug the board, FTDI, or audio jack for Intel Galileo Gen 2 and Intel Galileo respectively, and open a Linux terminal shell as explained in Chapter 1.

Connect the modem as shown in Figure 12-13 and type the following command:

```
root@clanton:~# lsusb
Bus 002 Device 006: ID 1519:f214
Bus 001 Device 001: ID 1d6b:0001
Bus 002 Device 001: ID 1d6b:0002
```

The first device just after `lsusb` command is the modem and it means a new device was recognized in OTG-USB with the respective vendor and product IDs.

If you see only the last two devices, it means your mPCIe/OTG-USB adaptor is not working or you have a problem with your modem card.

Step 2: Loading the CDC-ACM Driver

Communication with modem is done using the CDC-ACM serial driver by sending and receiving responses of AT commands. By default the driver is not loaded during the board startup and you must therefore load the driver manually using the `modprobe` command.

Then type the following command:

```
root@clanton:~# modprobe cdc-acm
```

If your modem is connected as shown in Figure 12-13, you will see a series of messages saying the device is not a modem.

```
[  135.720822] cdc_acm 2-1:1.2: This device cannot do calls on its own.
It is not a modem.
[  135.738315] cdc_acm 2-1:1.2: ttyACM0: USB ACM device
[  135.748033] cdc_acm 2-1:1.4: This device cannot do calls on its own.
It is not a modem.
[  135.765753] cdc_acm 2-1:1.4: ttyACM1: USB ACM device
[  135.775111] cdc_acm 2-1:1.6: This device cannot do calls on its own.
It is not a modem.
[  135.792294] cdc_acm 2-1:1.6: ttyACM2: USB ACM device
[  135.801241] cdc_acm 2-1:1.8: This device cannot do calls on its own.
It is not a modem.
[  135.818946] cdc_acm 2-1:1.8: ttyACM3: USB ACM device
[  135.827775] cdc_acm 2-1:1.10: This device cannot do calls on its own.
It is not a modem.
[  135.845711] cdc_acm 2-1:1.10: ttyACM4: USB ACM device
[  135.860799] usbcore: registered new interface driver cdc_acm
[  135.866545] cdc_acm: USB Abstract Control Model driver for USB modems and
ISD N adapters
```

Do not worry! Your device is a modem and you will be able to use it as a USB modem without problems. You can just ignore these messages. If you want to get rid of such messages, you need to generate your own image after you change the `.../linux/driverusb/class/cdc-acm.c` driver code:

```
/*
 * USB driver structure.
 */

static const struct usb_device_id acm_ids[] = {
        /* quirky and broken devices */
        { USB_DEVICE(0x0870, 0x0001), /* Metricom GS Modem */
        .driver_info = NO_UNION_NORMAL, /* has no union descriptor */
        },
...
...
...
```

```
        { USB_DEVICE(YOUR_VENDOR_ID, YOUR_PRODUCT_ID) }, /*  ADD YOUR MODEM
        HERE !!!!!!*/

...
...
...

        /* Motorola H24 HSPA module: */
        { USB_DEVICE(0x22b8, 0x2d91) }, /* modem                          */
        { USB_DEVICE(0x22b8, 0x2d92) }, /* modem           + diagnostics  */
        { USB_DEVICE(0x22b8, 0x2d93) }, /* modem + AT port                */
        { USB_DEVICE(0x22b8, 0x2d95) }, /* modem + AT port + diagnostics  */

...
...
...
```

Once you've changed the code, recompile and flash your Intel Galileo or generate a new BSP SD card release, as explained in Chapter 2. Again, this is not mandatory. Do this only if you do not want to see these error messages.

At this point you should have at least one `ttyACM` device available. If it is the first time you have connected the modem and you did not block any ttyACM devices, the `ttyACM0` should be available. You can check it with a simple `ls` command:

```
root@clanton:~# ls /dev/ttyACM0
/dev/ttyACM0
```

If the `ttyACM0` is not found, you should see a message like this one:

```
root@clanton:~# ls /dev/ttyACM0
ls: /dev/ttyACM0: No such file or directory
```

The next step is to see if your modem card can respond to AT commands.

Step 3: Checking if the Modem Responds to AT Commands

At this point, the modem is connected, the driver is loaded, and the ttyACM device is available to receive the AT commands. Before you send the AT commands, you can program the `ttyACM` device to echo the commands and responses. In order to change the ttyACM setting, use the `sty` command as follows:

```
root@clanton:~# stty -F /dev/ttyACM0 -echo
```

The -F specifies the device to be changed, such as ttyACM0, and the -echo option enables the echo of input commands with their respective responses as well.

To visualize the behavior of the ttyACM device when an AT command is sent, you should keep watching the device's responses. You have two options at this point. If you have only one cable for debugging using serial, you can keep watching the ttyACM device in the background. For example:

```
root@clanton:~# cat /dev/ttyACM0 &
```

However, if you have an Ethernet cable or a WiFi card, you can open multiple Linux terminals using SSH, as explained in Chapter 5. This way, you can send the AT command in one shell and debug the ttyACM responses in the other. For the shell, you observe the ttyACM responses and do not need to run **cat** in background:

```
root@clanton:~# cat /dev/ttyACM0
```

Then send a simple AT command to the device using an **echo** command and check if you receive **OK** as a response:

```
root@clanton:~# echo "AT" > /dev/ttyACM0
root@clanton:~# AT

OK
```

As you can see, once the command is sent, the ttyACM0 replies. In this example, the modem successfully responded with OK, which means the system is operational and ready to be configured.

If you do not receive a response, check if you are sending the command to the right ttyACM device and re-check the hardware.

Step 4: Checking the SIM Card Connection

If you have a nano-SIM and you are using a nano-SIM to micro-SIM adaptor, you need to check if the nano-SIM is good enough to guarantee that the SIM card's contacts are properly contacting the pins of the SIM card slot.

To test if your SIM card responds to your AT commands, you need to send the following command:

```
root@clanton:~# AT+CPIN?
+CPIN: READY

OK
```

If you receive **+CPIN:READY** followed by **OK**, your SIM card is fine. If you receive **ERROR**, you need to re-check the SIM card; see the section entitled "Preparing the SIM Card" in this chapter.

Step 5: Configuring the APN and Attaching it to the Network

Every carrier has a gateway that allows the device to access the Internet. This gateway is called the Access Point Name (APN) and each carrier contains different APNs for different technologies, such as GPRS, 3G, and 4G.

Before you continue with this chapter, it is fundamental that you know the configuration that your SIM card needs. Thus, if you operator is AT&T, Verizon, Sprint, Claro, or any other, you need to find out how the APN must be set.

You have three options to discover the APN required by your card—search the Internet; call your provider and request support; or if your SIM card is the same one you used in your mobile device, you can simply check the configurations of your mobile device. Following the last option and assuming your device is Android, you would check the configuration as follows:

1. Go to the settings of your device.
2. Disable the WiFi and select More Networks. Then click on Mobile Networks, as shown in Figure 12-14.

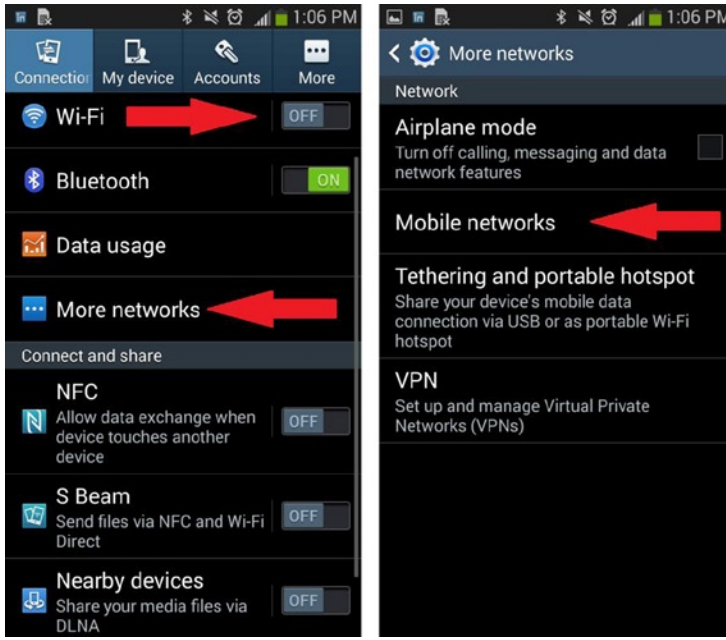

Figure 12-14. *Disabling the WiFi and selecting mobile networks*

You must disable here because some devices claim to be Android-compatible but aren't. In this case, the Google Play application is not available. When WiFi is ON, the devices camp in a very low band like 2G to reduce the battery consumption. If you check the APN in this case, an APN for 2G or 3G might be enabled instead of 4G because the network updates the configuration automatically and it's transparent to you. The next step is to select the APN of your carrier provider, check the APN name, provide a username and password if necessary, and then select the type of IP connection (IPv4 or IPv6) and the MCC and MNC numbers. For example, the red arrows in Figure 12-15 shows the AT&T APN settings in the United States.

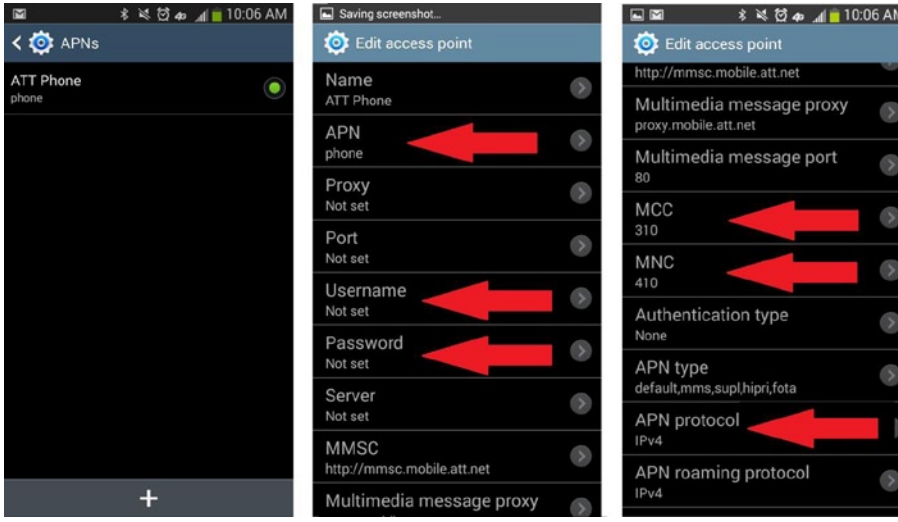

Figure 12-15. *Disabling the WiFi and selecting mobile networks*

As you can see in Figure 12-15, the AT&T 4G data connection requires the following settings:

- **APN:** Phone number
- **Username:** Not necessary
- **Password:** Not necessary
- **MCC:** 310
- **MNC:** 410
- **APN protocol:** IPv4

You can now configure your modem card. Before you program your modem, first check if the modem has some APN stored in it. You need to execute the `echo AT+CGDCONT?` command as follows:

```
root@clanton:~# echo "AT+CGDCONT?" > /dev/ttyACM0
+CGDCONT: 1,"IPV4V6","","",0,0
OK
```

The command returned says there is already an APN set in the modem card. It is empty. You can add a second one or you can replace this one.

To set the APN, you need to use the `CGDCONT` AT command, but send the APN name, the protocol type, and the APN number in the modem card list. Considering there is only one APN, if you want to include a second one, you just specify the number 2 in the command and include the second one. If you want to replace the APN already set, just use the number 1. This example includes a second APN so you can see how a modem card with several APNs is configured.

When strings are part of the arguments, the command's syntax can be a bit confusing. The strings must be between double quotes. For example, the command in this case will be:

```
AT+CGDCONT=2,"IP","phone"
```

Where `2` indicates that you are adding a second APN and preserving the first one, `"IP"` is because the protocol is IPv4, and `"phone"` is the APN name used by AT&T for 4G.

However, considering that you are sending the AT commands using `echo` in the Linux terminal, if you type `echo` without the double quotes, the modem will not understand the command. If the string requires double quotes, you must also differentiate between `echo's` double quotes and the string's double quotes. To do this, you need to use `\"`.

With this in your mind, the previous command will be:

```
root@clanton:~# echo "AT+CGDCONT=2,\"IP\",\"phone\"" > /dev/ttyACM0
OK
```

The `OK` means the command was accepted and the APN was created. Recheck the list of APNs running using the **AT+CGDCONT?** command.

```
root@clanton:~# echo "AT+CGDCONT?" > /dev/ttyACM0
+CGDCONT: 1,"IPV4V6","","",0,0
+CGDCONT: 2,"IP","phone","0.0.0.0",0,0,0,0,0,0
OK
```

The APN was added successfully. The next step is to activate PDP in the network using the following command:

```
root@clanton:~# echo "AT+CGACT=1,2" > /dev/ttyACM0
OK
```

The number **1** means "**activate**" and **2** is the number of the profile index that you programmed with the AT+CGDCONT command. If you want to deactivate the PDP later, you can send the same command but use **0** instead of 1 as the first argument.

This command may take a few seconds, so don't expect an immediate response.

Once the network is activated, it is necessary to inform the MCC (Mobile Country Code) and MNC (Mobile Network Code), which together informs your network operator code using the AT+COPS command. The following example shows the AT&T network operator code in the United States:

```
root@clanton:~# echo "AT+COPS=1,2,\"310410\"" > /dev/ttyACM0
OK
```

The last AT command to be sent is AT+CGDATA and it's responsible for opening the data stream with the modem to establish a connection with the network. An LTE modem uses the M-RAW_IP mode; the 1 argument means "attach" as shown in the following example:

```
root@clanton:~# echo "AT+CGDATA=\"M-RAW_IP\",1" > /dev/ttyACM0
CONNECT
```

Step 6: Creating an IP Interface with pppd

At this point, your modem card is connected to the network provider and the data stream is open. It's time to establish an interface to the modem and acquire a local IP address. The IP will then be able to connect to the Internet, open sockets, create programs to exchange data with other devices, and other possibilities, such as the ones you learned about in Chapter 5.

The modem can have an IP in the network NAT but Intel Galileo must be able to have an IP that links to this connection. One of the solutions is to use the point-to-point Protocol (PPP) and create a link with the external IP to a local IP on Intel Galileo.

The SPI and BSP images created for Intel Galileo contain an application called pppd (Point to Point Protocol Daemon) that creates this local IP and establishes the link with the Internet.

You must first create a configuration file that will be used by pppd. This configuration file contains relevant information about how to communicate with the modem card. Transfer the configuration file in the code folder named **options-att** to your board in the **/etc/ppp/peers** directory. Listing 12-1 shows the contents of this file.

Listing 12-1. The options-att Configuration File

```
ttyACM0
115200
lock
crtscts
passive
novj
defaultroute
noipdefault
usepeerdns
```

```
noauth
hide-password
persist
holdoff 10
maxfail 5
debug
connect '/usr/sbin/chat -f /etc/ppp/isp_chat'
```

Each of these options is explained in the official documentation at https://ppp.samba.org/pppd.html. They are adapted to Intel Galileo in the following list:

- **ttyACM0:** Uses the serial port called `ttyname` to communicate with the peer.
- **115200:** A decimal number that's taken as the desired baud rate for the serial device. This is the maximum rate for the USB with Intel Galileo.
- **Lock:** Specifies that `pppd` should create a UUCP-style lock file for the serial device to ensure exclusive access to the device.
- **Crtscts:** Specifies that `pppd` should set the serial port to use hardware flow control using the RTS and CTS signals.
- **Passive:** Enables the `passive` option in the LCP (Link Control Protocol). With this option, `pppd` will attempt to initiate a connection. If no reply is received from the peer, `pppd` will wait passively for a valid LCP packet from the peer, instead of exiting, as it would without this option.
- **Novj:** Disables Van Jacobson-style TCP/IP header compression in both the transmit and the receive directions.
- **Defaultroute:** Adds a default route to the system routing tables, using the peer as the gateway, when IPCP negotiation is successfully completed. This entry is removed when the PPP connection is broken.
- **Noipdefault:** Disables the default behavior when no local IP address is specified, which is to determine (if possible) the local IP address from the hostname. With this option, the peer will have to supply the local IP address during IPCP negotiation (unless it specified explicitly on the command line or in an `options` file).
- **Usepeerdns:** Asks the peer for up to two DNS server addresses. The addresses supplied by the peer (if any) are passed to the `/etc/ppp/ip-up` script in the `DNS1` and `DNS2` environment variables, and the `USEPEERDNS` environment variable will be set to 1. In addition, `pppd` will create an `/etc/ppp/resolv.conf` file containing one or two nameserver lines with the address(es) supplied by the peer.

- **Noauth:** Does not require the peer to authenticate itself. This option is privileged.
- **hide-password:** When logging the contents of PAP packets, this option causes `pppd` to exclude the password string from the log. This is the default.
- **Persist:** Does not exit after a connection is terminated; instead tries to reopen the connection. The `maxfail` option still has an effect on persistent connections.
- **holdoff n:** Specifies how many seconds to wait before re-initiating the link after it terminates. This option has an effect only when the `persist` or `demand` options are used. The holdoff period is not applied if the link was terminated because it was idle.
- **maxfail n:** Terminates after n consecutive failed connection attempts. A value of 0 means no limit. The default value is 10 and you are using 5.
- **Debug:** Enables connection-debugging facilities. If this option is given, `pppd` will log the contents of all control packets sent or received in a readable form. The packets are logged through syslog with facility daemon and level debug. This information can be directed to a file by setting up `/etc/syslog.conf` appropriately.
- **connect "script":** Usually you need to do something to prepare the link before the PPP protocol can be started; for instance, with a dial-up modem, commands need to be sent to the modem to dial the appropriate phone number. This option specifies a command for `pppd` to execute (by passing it to a shell) before attempting to start PPP negotiation. The **chat** program is often useful here, as it provides a way to send arbitrary strings to a modem and respond to received characters.

In this example, the connect option is calling `/usr/sbin/chat -f /etc/ppp/isp_chat`. The `isp_chat` file is available in the `code` folder of this chapter and you need to copy this file into the `/etc/ppp` directory.

This option calls the program chat that receives and sends data through the userspace and the `pppd` daemon. The `-f` option specifies the script that the chat program must run before connecting.

Listing 12-2 shows the contents of the `isp_chat` script.

Listing 12-2. The `isp_chat` Script

```
'' ATD*99#
TIMEOUT 30
CONNECT
```

The chat script is a sequence of pairs of strings and expected strings. In general, it contains what is expected before something that must be sent.

If you use '' nothing is expected as a response and this is the case of the first line of the isp_chat script. The '' makes a pair with ATD*99## that represents the dial string to the modem.

In the second line, the script sets a TIMEOUT with the maximum of 30 seconds.

Finally the third line waits for the CONNECT and nothing is sent as a reply.

To get the local IP, it is necessary to run mknod and pppd commands.

```
root@clanton:~# mknod /dev/ppp c 108 0
```

The first mknod command is necessary only when you are using the regular Intel Galileo release and want to create a device that can open a file node to communicate with the modem using PPP. However, if you recompiled the kernel enabling all options to support PPP, your customized kernel will have the /dev/ppp device already available and ready to be used. In the c 108 0 option, the c stands for character file non-buffered, the and 108 and 0 refer to the major and minor numbers, respectively, used to identify the device. It's the same methodology used with any regular char driver. Then you call pppd as follows:

```
root@clanton:~# pppd call options-att debug nodetach
Script /usr/sbin/chat -t3 -f /etc/ppp/isp_chat finished (pid 1486), status
= 0x0
Serial connection established.
using channel 1
Using interface ppp0
Connect: ppp0 <--> /dev/ttyACM0
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0xbcbf1398> <pcomp> <accomp>]
rcvd [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0x52181044> <pcomp> <accomp>]
sent [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0x52181044> <pcomp> <accomp>]
rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0xbcbf1398> <pcomp> <accomp>]
kernel does not support PPP filtering
[ 921.149938] PPP Deflate Compression module registered
sent [CCP ConfReq id=0x1 <deflate 15> <deflate(old#) 15>]
sent [IPCP ConfReq id=0x1 <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns2
0.0.0.0>]
rcvd [LCP ProtRej id=0x2 80 fd 01 01 00 0c 1a 04 78 00 18 04]
Protocol-Reject for 'Compression Control Protocol' (0x80fd) received
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=0x1]
sent [IPCP ConfNak id=0x1 <addr 0.0.0.0>]
rcvd [IPCP ConfNak id=0x1 <addr 10.8.194.5> <ms-dns1 172.26.38.1> <ms-dns2
172.26.38.2>]
sent [IPCP ConfReq id=0x2 <addr 10.8.194.5> <ms-dns1 172.26.38.1> <ms-dns2
172.26.38.2>]
rcvd [IPCP ConfReq id=0x2 <addr 10.8.194.5>]
sent [IPCP ConfAck id=0x2 <addr 10.8.194.5>]
```

```
rcvd [IPCP ConfAck id=0x2 <addr 10.8.194.5> <ms-dns1 172.26.38.1> <ms-dns2
172.26.38.2>]
local IP address 10.8.194.5
remote IP address 10.8.194.5
primary DNS address 172.26.38.1
secondary DNS address 172.26.38.2
Script /etc/ppp/ip-up started (pid 1501)
Script /etc/ppp/ip-up finished (pid 1501), status = 0x0
```

This second `pppd` command will start the PPP daemon, calling an `options-att` file (do not worry, you will learn about this later) in debug mode. `nodetach` means the `pppd` will not detach until you or any other input device asks to kill it. For example, if you press Ctrl+C, you will kill the daemon.

After a few seconds, the Internet `ppp0` will be available with a local IP.

Step 7: Testing the Internet Connection

Everything is in place at this moment—your pppd is running, you have a local IP, and the modem is camped with the data streaming opened. It's now time to test your Internet connection.

Try to `ping` a server to check if the interface works.

```
root@clanton:~# ping www.google.com
PING www.google.com (74.125.239.145): 56 data bytes
64 bytes from 74.125.239.145: seq=0 ttl=54 time=49.897 ms
64 bytes from 74.125.239.145: seq=1 ttl=54 time=78.188 ms
64 bytes from 74.125.239.145: seq=2 ttl=54 time=78.101 ms
```

The command shows that the Internet access is operational and the modem is ready to exchange data though the internet.

At this point, you are ready to communicate through your modem card. You are ready to send your robot, your quad-copter, or the mobile project to remote distances and control it.

You can also use this connection in the home-automation project explained in Chapter 9.

Testing the Internet Bandwidth

There are several mechanisms for testing your Internet bandwidth, and each mechanism has its proponents and detractors. It really depends on many variables.

To test bandwidth with Intel Galileo, use a Python script called `speedtest-cli.py` that you can download from https://pypi.python.org/pypi/speedtest-cli.

This scripts test your Internet bandwidth using the web site `www.speedtest.net`. Python must be available in your Intel Galileo image. In this case, if you are using BSP SD card images, the regular image contains Python packages.

Basically it tries to reach the best and optimal servers based on your IP location and executes the download and upload tests.

The procedure is very simple:

1. Make sure your modem is set and you have a local IP available according to the procedure described in the section entitled "Testing the Internet Connection" in this chapter.
2. Go to `https://pypi.python.org/pypi/speedtest-cli` and download the latest Python version.
3. Extract the contents of the package using `tar -zxvf <package_file_name>` if you are using Linux or OSX. Use Winzip or 7Zip if you are using Windows.
4. Among the files extracted, there is a file named `speedtest_cli.py` that must be transferred to your Intel Galileo. If you do not know how to transfer files, see Chapter 5 and choose the best alternative for you.
5. Simply execute the script using Python:

```
root@clanton:~/speedtest# python speedtest_cli.py
```

6. As a result, this script will show the speed of downloading and uploading.

Why Aren't the Download/Upload Bandwidths Optimal?

In real LTE networks, will you never be able to reach the max of 100 Mbps because there are several factors that interfere with the performance. This includes latency issues, number of users occupying the data channel of different bands, the quality of the service provider, and many other variables.

If you are using AT&T in California with an LTE modem and the correct antenna, you might reach an average download speed of 7 to 12 Mbps and an upload speed of 4 to 7 Mbps.

Ideas for Improving the Project

The following sections discuss a couple of ways to improve this project.

Improving the Chat Script

The AT commands in this chapter are passed manually, and the main reason for this is to help you to check if your hardware is OK and if the settings allow you to connect to your carrier network for data connection.

It does not mean that all AT commands must be manually typed, because as you read in the section entitled, "Creating an IP Interface with pppd," the `isp_chat` listed in Listing 12-2 accepts AT commands.

Once your know all the AT commands with their specific arguments for making your modem work, you simply need to mode these AT commands to the script.

For example, the AT commands used to set up the AT&T APN in this chapter are as follows:

```
AT+CPIN?
+CPIN: READY
OK

AT+CGDCONT=2,"IP","phone"
OK

AT+CGACT=1,2
OK

AT+COPS=1,2,"310410"
OK

AT+CGDATA="M-RAW_IP",1
CONNECT
```

You simply include the respective AT command in the `isp_chat` script, as shown in Listing 12-3.

Listing 12-3. The `isp_att_chat` Script

```
OK AT+CPIN?
OK AT+CGDCONT=2,"IP","phone"
OK AT+CGACT=1,2
OK AT+COPS=1,2,"310410"
'' AT+CGDATA="M-RAW_IP",1
TIMEOUT 10
'' ATD*99#
TIMEOUT 30
CONNECT
```

Then change Listing 12-1 in order to call the `isp_att_chat` script.

```
ttyACM0
115200
lock
crtscts
passive
novj
defaultroute
```

```
noipdefault
usepeerdns
noauth
hide-password
persist
holdoff 10
maxfail 5
debug
connect '/usr/sbin/chat -f /etc/ppp/isp_att_chat'
```

Run the pppd command again:

```
root@clanton:~# pppd call options-att debug nodetach
```

This way, none of the AT commands have to be invoked from the terminal.

Loading the cdc-acm Driver Automatically

Instead of having to invoke the `modprobe cdc-acm` manually every time your board boots, you can load it automatically. The procedure is very simple if you are using BSP SD card images:

1. Connect your FDTI cable or serial jack cable and open the Linux terminal.
2. Enter the `/etc/modules-load.quark` directory by typing:

   ```
   root@clanton:~# cd /etc/modules-load.quark
   ```

3. If your board is Intel Galileo Gen 2, edit the `galileo_gen2.conf` file; otherwise, you need to edit `galileo.conf` if your board is Intel Galileo. You can edit directly in the Linux terminal using the `vi` editor or any other method. These files represent a configuration file with all the modules that must be loaded just after the boot.
4. Insert the `cdc-acm` line into the configuration file.

   ```
   pch-udc
   g-serial vendor=0x8086 product=0xBABE
   stmmac
   ehci_hcd
   ohci_hcd
   ehci-pci
   usb_storage
   gpio_sch
   intel_qrk_gip
   gpio-pca953x
   ```

```
pca9685
at24
i2c_dev
spidev
spi-pxa2xx-pci
spi-pxa2xx
industrialio
adc1x8s102
iwlwifi
btusb
usbhid
evdev
sdhci-pci
mmc-block
cdc-acm
```

5. Reboot the board by typing `reboot` and pressing Enter.

If you are using custom SPI images, the procedure is very similar but you need to change this in the `yocto` build. In this case the configuration files `galileo.conf` and `galileo_gen2.conf` are present in the directory `.../meta-clanton_v1.0.3/meta-clanton-bsp/recipes-kernel/quark-init/files`. Do the same changes mentioned in Step 4, rebuild your SPI image, and flash your Intel Galileo with your new custom SPI image.

After the boot, the `cdc-acm` drive must be loaded automatically. You can check it by typing `lsmod` into the Linux terminal shell.

Summary

This chapter explains a solution based in modem cards for applications that require large-scale mobility when WiFi coverage is not enough.

It also explains how to connect modem cards to Intel Galileo boards by using adaptors, how to use the SIM card, what to do when there's a form factor adaptation, and how to select and connect the antennas.

In terms of software, this chapter introduced the modem setup using AT commands, explained how to establish a local IP using the PPP protocol, and covered the scripts and utilities needed to connect to the Internet.