PART II

Moving to Advanced and Specialist Applications

CHAPTER 11

Wireless Communication – Bluetooth and Zigbee

11.1 Introducing Wireless Data Communication

For many years flying radio-controlled model aircraft has been a popular pastime. The plane "pilot" holds the radio control unit, which sends simple data control messages to the aircraft soaring above. The control unit is configured to transmit at a particular radio frequency. The pilot may be displaying a coloured tag, showing on which frequency the radio is operating. If someone else arrives and wants to fly a plane at the same place, then he or she must set his radio to a different frequency, or there will be interference between the two, possibly leading to a spectacular plane crash. If more people come, then even greater care must be taken, to make sure that each has a unique radio frequency. Contrast this simple image of radio data communication with a more recent setting. Picture instead a busy airport lounge: hundreds of travellers are waiting for their flights. Many are on their mobile phones; others have laptops or tablets in use, maybe with wireless-linked mice, keyboards or headphones. Potentially thousands of data messages are flying through the air. All must get through reliably; none should interfere with any other.

This chapter explores some of the issues and mysteries relating to wireless data communications. It's a big topic, so we focus on two well-known protocols, Bluetooth and Zigbee; these are particularly applicable in the world of the mbed. We start with a very swift review of some important aspects of wireless communication. There's much of the theoretical background that we cannot cover, so do check Ref. [1] for further information on any topic where you want to see the detail.

11.1.1 Some Wireless Preliminaries

The traditional way of transferring data between devices or sub-systems has been through electrical connection; wires, cables or PCB (printed circuit board) tracks. Yet this physical connection is in many situations inconvenient, annoying, or just plain impossible to implement. This is particularly true for applications where things need to move around, or engage and disengage. It is, of course, possible to make a data connection without any physical link. Alternatives to wired connection are very familiar, and include infrared, radio or visible light. All are examples of electromagnetic waves; they can be found in the

Fast and Effective Embedded Systems Design. http://dx.doi.org/10.1016/B978-0-08-100880-5.00011-6

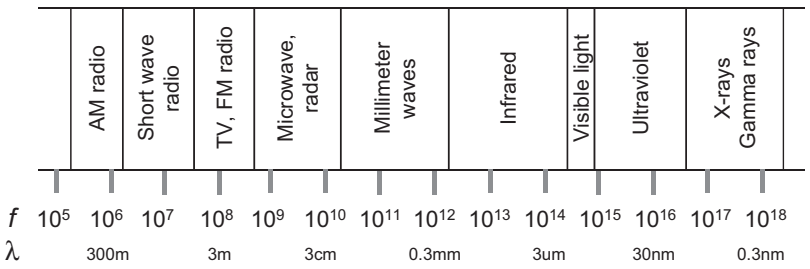

Figure 11.1
The electromagnetic spectrum.

electromagnetic spectrum, seen in Fig. 11.1. This shows a very wide range of frequencies, where frequency and wavelength are related by Eq. (11.1). Here f is the frequency, λ the wavelength, and v is the speed of radiation, known to be approximately 3×10^8 m/s.

$$v = f\lambda \tag{11.1}$$

The spectrum usefully shows where various wireless activities sit. Any frequency on the spectrum, from the lowest frequency up to visible light, can be used for data communication. Almost all of this is very strictly regulated by national and international agencies – you can't just start broadcasting a radio signal at any frequency you like! As more and more wireless activity has come along, the spectrum has become increasingly crowded. The International Telecommunication Union, a United Nations agency, manages the allocation of the radio spectrum between different broadcasters and applications. It reserves certain frequency bands for Industrial, Scientific and Medical (ISM) applications. These are *unlicensed*, and some vary between countries. The 2.4 GHz band is however reserved for unlicensed use in all regions. It has become widely used, mainly for short-range, low power applications.

While the spectrum represents the "pure" radio frequencies, they only become useful once they are carrying information. This is done by the process of *modulation*; the information to be carried is imprinted onto the carrier frequency, through one of a number of different techniques. Amplitude Modulation (AM) and Frequency Modulation (FM) are the old favourites of the broadcast industry. One effect of modulation is to cause fluctuations around the base frequency; thus if we say that a certain radio station can be found at the frequency of 103 MHz, in fact it is in a narrow band of frequencies centred on 103 MHz. The word *bandwidth* is used to define a range of frequencies, for example within which a particular transmission may be taking place.

The relationship between the information a signal can convey and its bandwidth is direct – higher data rates demand wider bandwidth. In general, higher frequency bands offer more channels and bandwidth, so are used to provide larger networks and higher data

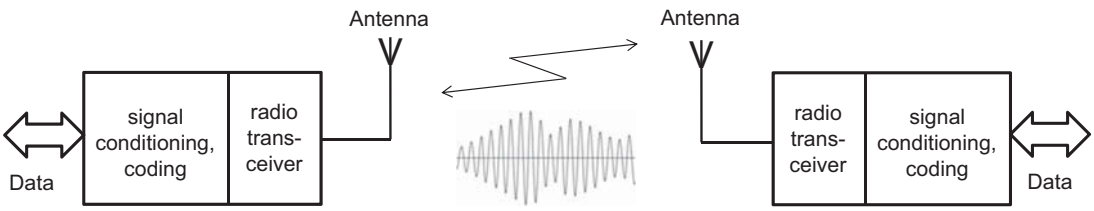

Figure 11.2
A basic radio data link.

throughput. However lower frequency propagates better than higher frequency, so can achieve better range, e.g. for neighbourhoods and within buildings.

The basics of a simple radio link are shown in Fig. 11.2. Data is acquired, conditioned, coded, and transmitted by the radio transceiver. The data is used in some way to modulate the carrier frequency. This is implied in the waveform at the centre of the Figure. In this example, the amplitude of a carrier wave is modulated by the signal, i.e. this is an example of amplitude modulation. The electrical signal so produced flows to the antenna, which causes an electromagnetic wave of the same frequency to be radiated. The range of this radio signal will depend on many things, including the power of the transmitter, the efficiency of the antenna, and the signal frequency itself. If there is another antenna within range, of appropriate dimension, then the signal can be received and detected by a receiver circuit. Of course if another radio is transmitting at the same frequency in the same range, then the signals will interfere, and confusion will reign.

The performance and efficiency of the antenna, at the frequency of operation, is determined by its physical shape and size. The length of the antenna should be a multiple or fraction of the wavelength. One of the big challenges in low-cost and small-size wireless communication has been to scale the antenna to the rest of the product, whether that is a mobile phone or a wearable health monitor. In so doing, trade-offs are made between physical size and efficiency. For example a wireless computer mouse, transmitting over a short distance with a low data rate, can operate with a less efficient antenna compared with a high data rate link operating over a greater distance. With new and miniature devices, it is now common to see antennae formed from a trace of PCB track, within a chip, or a tiny wire antenna. Examples of each of these appear later in this chapter, in Figs. 11.6 and 11.12.

Returning to our comparison of the radio-controlled aircraft and the airport lounge full of people on mobile devices – do these people run around and agree between themselves who is going to transmit on which frequency? This of course is absurd and impossible. One technique applied is the use of *spread spectrum* transmission. In this the transmitting frequency keeps changing, within a certain bandwidth. The receiver needs to know the

transmitter's frequency hopping pattern in order to receive the signal properly. This technique was originally applied in secure systems – if you can't keep track of the frequency hopping, then you can't snoop on the signal. However it's also useful when space and bandwidth is shared. Suppose several data links are all in action close to each other. If two are at the same continuous frequency, then we know they will interfere. However, if they're continuously switching frequency – having first agreed within their pairs or networks a frequency-hopping pattern – then if they do occasionally clash this can be detected and corrected. Most of the time the probability is that different networks will select a different frequency from each other, and will not interfere.

11.1.2 Wireless Networks

There are many situations where we need to provide connection between different systems or subsystems. In the domestic environment, the automated household is becoming a reality. Here different household appliances and gadgets may all be connected together. Elsewhere, there are other needs for networking. The modern motor vehicle may contain dozens of embedded systems, all engaged in very specific activity, but all interconnected. In the home or car situation, connections may be long term and stable. However, other networks or connections are transitory, for example when data is downloaded from a smartphone to a laptop over a wireless link.

Providing a network is about much more than just providing connectivity, important though this is. In a complex system it is also essential to deal in depth with how data is formatted and interpreted, how addressing is achieved, and how error correction can be implemented. All of this is pretty much independent of the physical interconnection itself. In order for different nodes to communicate on a network, there must therefore be very clear rules about how they create and interpret messages. We have already seen aspects of this with definitions of standards like I^2C (Inter-Integrated Circuit) and USB (Universal Serial Bus). This set of rules is called a *protocol*, taking the word from its diplomatic and legal origins.

To establish terminology, networks are sometimes divided into four categories, as shown in Fig. 11.3. The Personal Area Network (PAN) usually relates to devices worn on the person, such as smart watch, personal entertainment, or health or performance monitoring. The Local Area Network (LAN) typically applies to a single building, for example a network of computers in a home, company, school or office. The Neighbourhood Area Network (NAN) reflects a need to network in a wider area still, for example for a smart transport or smart energy system; this brings us towards the realisation of the smart city. The Wide Area Network (WAN) effectively includes national or global systems, most notably the Internet. Each has distinct demands, based both on technical issues, and the type of data generated. Of course, as is often the case, one type of network can link to

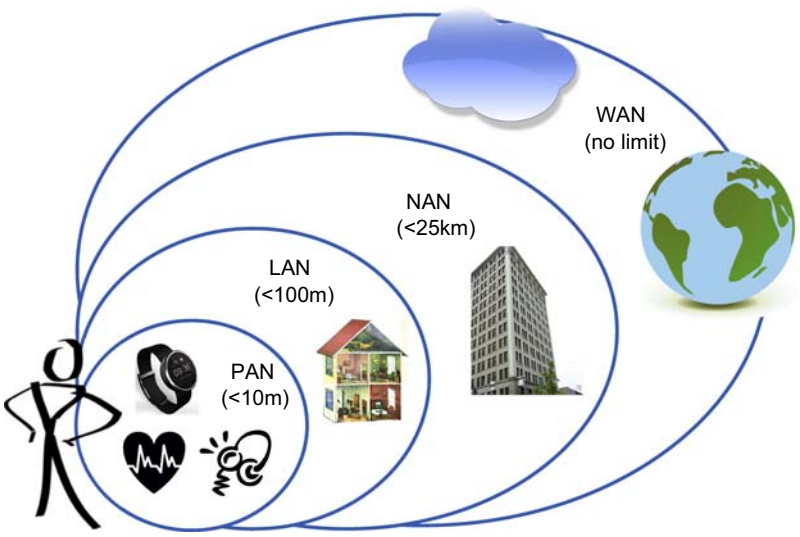

Figure 11.3
Network ranges.

another. Internet communications with the mbed on wired WANs will be introduced in Chapter 12; this chapter will predominantly discuss wireless connectivity over short ranges, applicable to a PAN or LAN setup.

11.1.3 A Word on Protocols

With large networked systems, protocols can become very complicated, defining every aspect of the communication link. Some of these aspects are obvious, while others are less so. To aid in the process of defining a protocol, the International Organisation for Standardisation (ISO) devised a "protocol for protocols", called the *Open Systems Interconnect* (OSI) model. This is shown in Fig. 11.4. Each layer of the OSI model provides a defined set of services to the layer above, and each therefore depends on the services of the layer below. The lowest three layers depend on the network itself and are sometimes called the *media layers*. The physical layer defines the physical and electrical link, specifying, for example, what sort of connector is used and how the data is represented electrically. The link layer is meant to provide reliable data flow, and includes activities such as error checking and correcting. The network layer places the data within the context of the network and includes activities such as node addressing.

The upper layers of the OSI model are all implemented in software. This takes place on a host computer, and the layers are sometimes called the *host layers*. The software

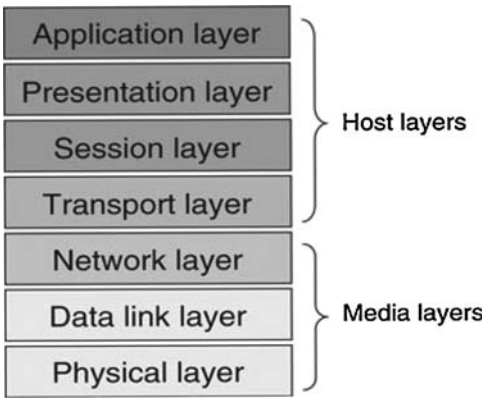

Figure 11.4
The ISO Open Systems Interconnect (OSI) model.

implementation is often called a *protocol stack*. For a given protocol and hardware environment, it can be supplied as a standard software package. A designer adopting a protocol stack may need to interface with it at the bottom end, providing physical interconnect, and at the top end, providing a software interface with the application.

This model forms a framework against which new protocols can be defined and a useful point of reference when studying the various protocols already available. In practice, any one protocol is unlikely to prescribe for every layer of the OSI model, or it may only follow it in an approximate way.

The IEEE (the Institute of Electrical and Electronic Engineers) plays a major role in defining standards and protocols. Unsurprisingly, it is very active in the field of networked communications, and maintains a set of standards for Local Area Networks, allocated the number 802. A small number of these which are relevant to this and the next chapter are shown in Table 11.1. More can be found at Ref. [2].

The activities of the working groups shown in Table 11.1 underpin much of the content of this chapter and the next.

Table 11.1: Example IEEE 802 working groups.

IEEE Working Group	Description
802.3	Ethernet
802.11	Wireless LAN, including Wi-Fi
802.15	Wireless PAN
802.15.1	Bluetooth
802.15.3	High-rate wireless PAN
802.15.4	Low-rate wireless PAN, e.g. Zigbee

11.2 Bluetooth

11.2.1 Introducing Bluetooth

Bluetooth is a digital radio protocol, intended primarily for PAN applications, and operating in the 2.4 GHz radio band. It was developed by the Swedish phone company Ericsson, who took the name of a tenth century Viking king to name their communication protocol. Bluetooth provides wireless data links between such devices as mobile phones, wireless audio headsets, computer interface devices like mice and keyboards, and systems requiring the use of remote sensors. Bluetooth standards are now controlled by the Bluetooth Special Interest Group [3], though the IEEE has made an important contribution. It is a formidably complex protocol. There is a core specification, and then over 40 different *profiles*, which specify different Bluetooth applications, for example for audio, printers, file transfer, cable replacement, and so on.

There are three Bluetooth classes, based on output power, with Class 2 being the most common. The main characteristics are as follows:

- The approximate communication range is up to 100 m for Class 1 Bluetooth devices, up to 10 m for Class 2 devices, and 1 m for Class 3.
- Bluetooth is relatively low power; devices of Classes 1 to 3 use around 100, 2.5 and 1 mW respectively.
- Data rates up to 3 Mbps can be achieved. Recent higher data rate versions are being adopted.
- Up to 8 devices can be simultaneously linked, in a *piconet*. A Bluetooth device can belong to more than one piconet.
- Spread-spectrum frequency hopping is applied, with the transmitter changing frequency in a pseudo-random manner 1600 times per second, i.e. a *slot* duration of 0.625 ms.

When Bluetooth devices detect one another, they determine automatically whether they need to interact. Each device has a Media Access Control (MAC) address which communicating devices can recognise and initialise interaction if required. The MAC address (a component of the Data Link Layer seen in Fig. 11.4) is unique to the device, and is embedded in its hardware by the manufacturer. This process follows three phases:

- Discovery, when a slave module broadcasts its name and MAC address, seeking for devices to link to.
- Pairing, when slave and master exchange identification and authentication data, exploring whether a link should be established. Note that some devices do not demand full authentication, though PCs usually do.
- Connecting, initiated by the master, through which a link is finally established.

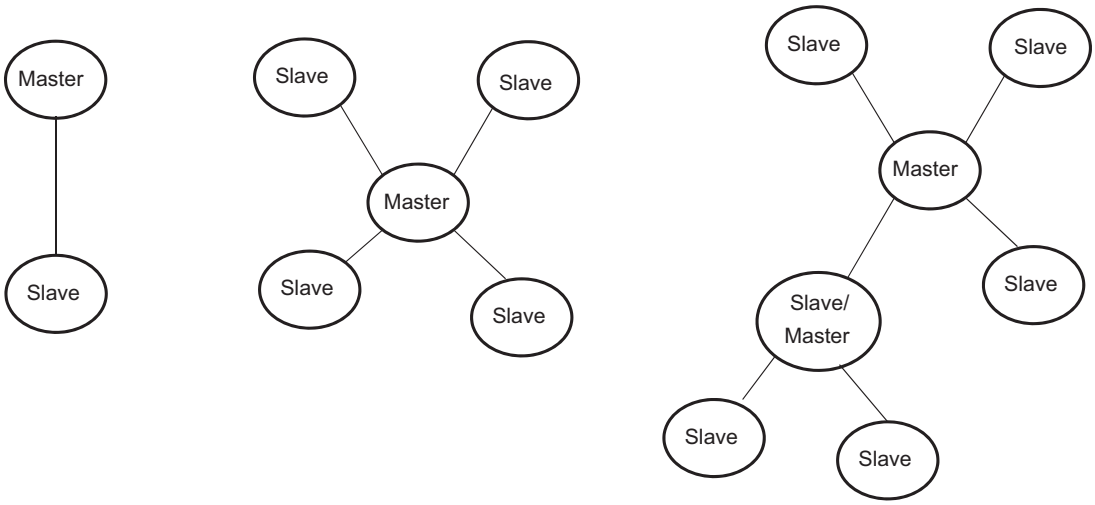

Figure 11.5
Possible bluetooth networks.

A piconet contains a master, and up to seven slaves. Once communication is established, members of the piconet synchronise their frequency hopping, managed by the master, so they remain in contact. The network patterns which can be formed are shown in a simple way in Fig. 11.5. The simplest is just one master and one slave. This acts as a straightforward replacement of a wired connection, and for many applications is all that is needed. Slaves may only communicate with their master, when permitted by the master, and not with each other. Piconets can also link, through one or more shared devices; this is called a *scatter-net*.

A single room or space can contain many piconets. Think again of the busy airport waiting lounge, with numerous people using laptops and mobile phones, many radiating Bluetooth data. The use of spread spectrum allows many Bluetooth devices to share the same physical space. However when the number of piconets increases indefinitely, the number of collisions increase, and the performance begins to degrade.

11.2.2 The RN-41 and RN-42 Bluetooth Modules

The RN-41 (appearing in Fig. 11.6) and RN-42 modules are devices which allow easy introduction to Bluetooth. The devices were developed by the company Roving Networks, who in 2012 were acquired by Microchip Technology. The modules have a simple manual, [4]; much essential information is however in the Advanced User Guide, [5]. Versions of each, under the names of Roving Networks or Microchip, are widely available on the web. The RN-41 and RN-42 modules have identical control and functionality; however the RN-41 is a Class 1 Bluetooth device, whereas the RN-42 is Class 2. They operate at Baud

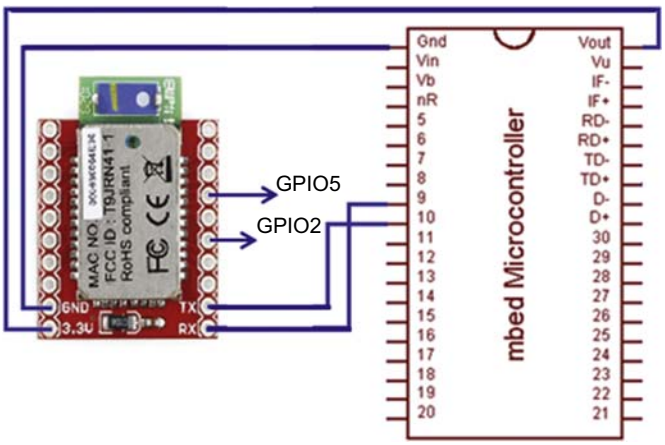

Figure 11.6
The RN-41 (with chip antenna) connected to the mbed. *RN-41 image reproduced courtesy of SparkFun Electronics.*

rates between 1200 bps and 921 kbps and include auto-detect and auto-connection features. In this chapter the RN-41 module is used, however all software examples should work equally for a RN-42 device.

The simplest use of the RN-41 module is to replace a wired serial link with Bluetooth. Their default configuration is as a Slave device, so they are immediately ready for this. For example, most laptop computers have Bluetooth capability installed, so it is possible to replace the USB cable to an mbed with a Bluetooth device and allow wireless communication to the laptop. The RN-41 has however a range of configurable features, and can also be used for more sophisticated applications. The RN-41 can be purchased mounted on a breakout board, as shown in Fig. 11.5. If you're confident in your soldering, you can save money by soldering connecting wires direct to an unmounted module.

11.2.3 Getting to Know the RN-41

The RN-41 can be used just as a "black box", without the need to understand the internal detail of how it works. However it helps to understand Bluetooth better if that detail is explored, and enables original designs to be made. Skip this section if you just want a "light touch" Bluetooth experience. Here, we will explore communicating directly with an RN-41, via a PC, before making any mbed connection.

Once the RN-41 is powered, and without any further connection or intervention, it will start announcing itself as a Bluetooth Slave device, and will seek for a pairing. Therefore try simply powering an RN-41, supplying it with 3.3 V. An easy way to do this is from an mbed. In other words, connect *only* the power supply connections of the circuit of Fig. 11.6.

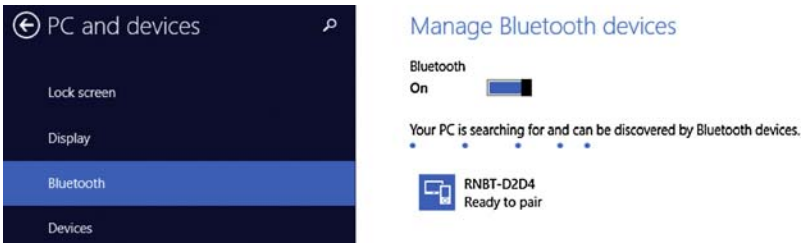

Figure 11.7
Setting up the PC link to the RN-41 module.

Power the module, and check the window of your PC Bluetooth device manager. The RN-41 is in Discovery mode. The computer will detect it, and offer a pairing. Fig. 11.7 shows a Windows 8.1 implementation of this process, with D2D4 being the last four digits of the MAC address. (Note that two different modules are used in Figs. 11.6 and 11.7.) If you click on the displayed Bluetooth symbol, you will be led through a simple sequence which results in the RN-41 being paired and then connected to the PC. You may need to specify a *passkey* which is set by default to "1234", as specified in the Advanced User's Guide [5].

You will also need to set up a host terminal application, like Tera Term for Windows (as used here), or CoolTerm for Apple Mac. Having made the above connection, and when you open Tera Term, you should find an offer of a COM port with Bluetooth linkage, as seen in Fig. 11.8. Accept this. In the screen which follows, select **Setup → Serial Port**, and set the Baud rate to 115200. If connection with the first COM port that you try is refused, check to see if another Bluetooth-linked one is available.

You can now communicate with the RN-41 by Tera Term. Normal RN-41 usage is in Data mode, when it transfers data through its serial port, and transmits or receives it through Bluetooth. However to reconfigure it, or to interrogate its status, it needs to be put into

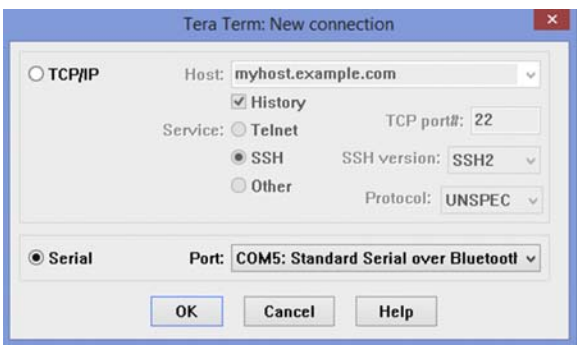

Figure 11.8
Configuring Tera Term for the bluetooth link.

Table 11.2: Example RN-41 commands.

Command	Response
Entry and Exit to Command Mode	
$$$	Enter command mode (no <cr> needed).
- - - <cr>	Exit command mode.
Get commands	
D	Display basic settings: Address, Name, UART Settings, Security, Pin code, Bonding, Remote Address
GB	Returns the Bluetooth Address of the device
GK	Returns the current connection status: 1 = connected, 0 = not connected
SP,<text>	Sets the security passkey for pairing (the default "1234" will already be set)
Set commands	
C,<address>	Connect to a remote MAC address, specified in <address>
R,1	Forces a complete device reboot (similar to a power cycle).
SA,<value>	Sets authentication needed when a remote device attempts to connect. Includes 0 = Open, 1 (default) = remote host prompted to confirm pairing, 4 = Pin code authentication
SM,<value>	Sets the mode, depending on <value>. Includes 0 = Slave, 1 = Master, 3 = Auto-connect Master, 6 = Pairing.

<cr> is Carriage Return, whose ASCII value is 0x0D.

Command mode. A small selection of possible commands is shown in Table 11.2. Within these there are Get commands, which simply seek information from the device, and Set commands, which change its internal settings. These only take effect after a power down or reboot. More can be found in Ref. [5].

With the RN-41 connected, and Tera Term open, try entering "$$$" on the computer keyboard. The RN-41 should respond with "CMD", as seen in Fig. 11.9. Then enter D,

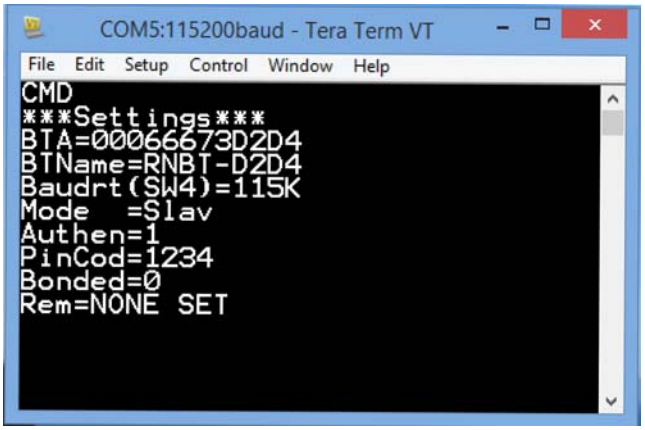

Figure 11.9
Tera Term display of RN-41 messages.

Table 11.3: RN-41 diagnostic LED connections.

Pin	Status	Description
GPIO2	High	The module is connected to another device over Bluetooth.
	Low	The module is not connected over Bluetooth.
GPIO5	Low	The module is connected to another device over Bluetooth.
	Toggle at 1 Hz	The module is discoverable and waiting for a connection.
	Toggle at 10 Hz	The module is in command mode.

and the basic device settings should be displayed, again as shown in Fig. 11.9. Exit Command mode by entering "- - - <cr>". Tera Term should then display the word "END".

It's useful also to implement diagnostic capability which does not depend on Tera Term interaction. This can be done by connecting LEDs to the GPIO2 (General Purpose Input/Output) and GPIO5 pins of the RN-41. The information they provide is shown in Table 11.3. Try repeating the connection sequence above, now with LEDs connected. On power-up the GPIO5 LED should blink slowly, and the GPIO2 LED will not be lit. This lights when a connection is made with the PC. Once Command mode is entered, the GPIO5 LED will blink rapidly, at 10 Hz. When you exit Command mode, GPIO5 should go low, and GPIO2 remain high.

11.2.4 Simple Bluetooth: Sending mbed Data to a PC

Return now to the circuit of Fig. 11.6, and include the simplest possible connection to an mbed, making all connections shown. The GPIO2 and 5 links are described above, and are optional.

Program Example 11.1 applies an RN-41 with an mbed to send data from mbed to PC (acting as Bluetooth Master) over a Bluetooth serial link, with data received on Tera Term. The data is a continuous count through the ASCII values representing numerical characters 0—9. The on-board mbed LEDs are also configured to represent the count value. The serial API functions given in Table 7.9 are used in this and subsequent programs. In the example we convert the ASCII byte to the relevant numerical value by bit masking and clearing the higher 4 bits. This leaves just a value between 0x00 and 0x09. In reality the bit masking of x makes negligible difference because the **led BusOut** object can only take the lower four bits of the 8-bit number anyway.

```
/* Program Example 11.1: Bluetooth serial test data program
   Data is transferred from mbed to PC via Bluetooth.       */

#include "mbed.h"
Serial rn41(p9,p10);      //name the serial port rn41
BusOut led(LED4,LED3,LED2,LED1);
```

```
int main() {
  rn41.baud(115200);     // set baud for RN41
  while (1) {
    for (char x=0x30;x<=0x39;x++){  // ASCII numerical characters 0-9
      rn41.putc(x);         // send test char data on serial to RN41
      led = x & 0x0F;               // set LEDs to count in binary
      wait(0.5);
    }
  }
}
```

Program Example 11.1: Bluetooth serial test, mbed to PC

Compile Program Example 11.1, and download to the circuit of Fig. 11.6. To test this program, the host computer must of course be Bluetooth-enabled and already linked to the RN-41 module, as described in Section 11.2.3. If a Bluetooth connection is successfully set up on the host PC, the data sent over Bluetooth from Program Example 11.1 will appear on the Tera Term or CoolTerm screen, as an endless and repeated count of 0 to 9.

Exercise 11.1

If you have been following the sequence to date, then you probably still have the usual USB cable connecting host computer and mbed. Now break that umbilical, and power the mbed from a battery in the usual way (as in Figs. 4.10B or 11.10B). Remove the USB cable and demonstrate fully remote communication over Bluetooth.

■

11.2.5 Simple Bluetooth: Receiving Bluetooth Data From a PC

As well as sending data to a PC over Bluetooth, it is of course possible to receive data from the host PC, as Program Example 11.2 demonstrates. The program first of all sends a character string from mbed to PC, which will appear on Tera Term if enabled. It then will receive any data from the host PC keyboard; the remote mbed displays the lower four bits of the received byte on the four on-board LEDs.

```
/* Program Example 11.2: Bluetooth serial test data program
   Data is transferred bidirectionally between mbed and PC via Bluetooth.
                                                                    */

#include "mbed.h"
Serial rn41(p9,p10);           //name the serial port rn41
BusOut led(LED4,LED3,LED2,LED1);
```

```
int main() {
  rn41.baud(115200);           // setup baud rate
  rn41.printf("Serial data test: outputs received data to LEDs\n\r");
  while (1) {
    if (rn41.readable()) {     // if data available
      char x=rn41.getc();      // get data
      led=x;                   // output LSByte to LEDs
    }
  }
}
```

Program Example 11.2: Bluetooth bidirectional data test

Download Program Example 11.2 to your Bluetooth-enabled mbed (i.e. the circuit of Fig. 11.6) and run, with Tera Term on the Host Computer screen. The mbed first transmits the message "Serial data test: outputs received data to LEDs". It then awaits data transmitted from the computer keyboard. If you press numerical values between 0 and 9 on the keyboard, the corresponding binary representation should be shown on the remote mbed.

Exercise 11.2

Add a battery and an LCD display (use wiring implemented in Fig. 8.2) to the Bluetooth enabled mbed. Update Program Example 11.2 to allow a user to type text which will appear on the wireless LCD display.

■

11.2.6 More Advanced Bluetooth: Communicating Between Two mbeds

By implementing Bluetooth it is possible to have two or more mbeds communicating wirelessly with each other. This configuration is a little more involved, as one of the RN-41 modules must be set as Master, using the commands shown in Table 11.2. Both Master and Slave use the circuit of Fig. 11.6 (with or without diagnostic LEDs), except that the Master has a simple slide switch added. This can be a simple on/off switch (single pole – double throw) linking pins 26 and pin 40. When open, the internal pull-down resistor configured by **DigitalIn** establishes Logic 0; when closed then the input is connected to 3.3 V, i.e. Logic 1.

The Master mbed is configured as seen in the **initialise_connection()** function, in Programme Example 11.3. It should be possible to follow what this is doing, by reading the comments, and checking with Table 11.2 (or better still the Advanced User Manual). The program needs the MAC address of the slave RN-41 module. This is written on it (as seen in Fig. 11.6), and is available in the simple interrogation of module settings, seen in

Fig. 11.9. The Slave RN-41 module in this example has a MAC address of 00066673D2D4, identified as the target in the program example.

```
/* Program Example 11.3: Paired Bluetooth master program
                                                          */
#include "mbed.h"
Serial rn41(p9,p10);        //Name and define the serial link to the RN-41
BusOut led(LED4,LED3,LED2,LED1);
DigitalIn Din(p26);         // digital switch input on pin 26

char x;
void initialise_connection(void);

int main() {
  rn41.baud(115200);
  wait(1.0); //can only enter Command mode 500ms after power-up
  initialise_connection();
  while (1) {
    if (Din==1)           // if digital input switched high
      x=0x0F;             // override count, with 0x0F
    else {
      x++;                // else increment and
      if (x>0x0F)         // output count value
        x=0;
      }
    rn41.putc(x);         // send char data on serial
    led = x;              // set LEDs to count in binary
    wait(0.5);
  }
}
void initialise_connection() {
  rn41.putc('$');     // Enter command mode
  rn41.putc('$');     //
  rn41.putc('$');     //
  wait(1.2);
  rn41.putc('S');     //enter Master mode
  rn41.putc('M');
  rn41.putc(',');
  rn41.putc('3');     //"Auto-connect" Master mode
  rn41.putc(0x0D);    //CR
  wait(0.1);
  rn41.putc('C');     // Causes RN-41 to attempt connection
  rn41.putc(',');     // Send MAC address of target slave device
  rn41.putc('0');     //
  rn41.putc('0');     //
  rn41.putc('0');     //
  rn41.putc('6');     //
  rn41.putc('6');     //
  rn41.putc('6');     //
  rn41.putc('7');     //
  rn41.putc('3');     //
  rn41.putc('D');     //
  rn41.putc('2');     //
```

```
    rn41.putc('D');     //
    rn41.putc('4');     //
    rn41.putc(0x0D);
    wait(1.0);
    rn41.putc('-');     // Exit command mode
    rn41.putc('-');     //
    rn41.putc('-');     //
    rn41.putc(0x0D);    //
    wait(0.5);
  }
```

Program Example 11.3: Paired Bluetooth Master program

We need further to adjust configuration of the Slave module. For simplicity, we take the opportunity to dispense with the pairing phase of Bluetooth connection, by discarding some security. This is done by setting the authentication of the Slave with the "SA" command (Table 11.2) to 0, i.e. an "Open" connection. The Slave should apply Program Example 11.4, an adjusted version of Program Example 11.2.

```
/* Program Example 11.4: Bluetooth paired slave program
                                                              */
#include "mbed.h"
Serial rn41(p9,p10);              //name the serial port rn41
BusOut led(LED4,LED3,LED2,LED1);

int main() {
  rn41.baud(115200); // setup baud rate
  rn41.putc('$');    // Enter command mode
  rn41.putc('$');
  rn41.putc('$');
  wait(1.2);
  rn41.putc('S');    //set Authentication to 0
  rn41.putc('A');
  rn41.putc(',');
  rn41.putc('0');
  rn41.putc(0x0D);
  rn41.putc('-');    // Exit command mode
  rn41.putc('-');
  rn41.putc('-');
  rn41.putc(0x0D);
  wait(0.5);

  while (1) {
    if (rn41.readable()) {    // if data available
     char x=rn41.getc();      // get data
     led=x;                   // output LSByte to LEDs
    }
  }
}
```

Program Example 11.4: Bluetooth Slave program

You should at this stage have two Bluetooth-enabled mbed circuits, as seen in Fig. 11.10. Ensure that neither RN-41 module is still paired with your PC from a previous experiment. Switch both circuits on, when fully configured. If you have the diagnostic LEDs fitted (described above, but not shown in the figure), you will readily see the Master device enter Command mode, through the change in flash rate of the GPIO5 LED. The GPIO2 LED on *both* modules should then quickly light. Shortly after, the four LEDs on each mbed should start counting in synchronisation. If this does not happen, try changing the order of power-up. If problems persist, it can be useful to reconnect each module in turn to the PC. A status check should reveal the state they have been programmed into, which should accord with the program with which they have been loaded.

Once running, try "discovering" the active RN-41s on your PC screen. It is unlikely that they will be detected, as they are currently paired, and hence not "discoverable".

Exercise 11.3

Test the range and characteristics of your Bluetooth link. For example:

1. In an open space, carrying the Slave mbed, slowly walk away from the other, continuously testing the link. What range can you achieve? Does it fit in with the Class 1 Bluetooth expectation? Is it impacted by orientation of one or both RN-41 modules?
2. Explore the behaviour of the link in different physical configurations. You may need the help of a friend for this. Does it transmit through a wall or between floors of a building? What happens if one node is placed in a saucepan or other metallic enclosure?

■

Exercise 11.4

Explore the data rate available to your Bluetooth data link. Take the sine wave generation program of Program Example 4.3 (or another waveform if you wish). Embed this into the Master program, Program Example 11.3, in place of the counting section of the program. Transmit that waveform data over the Bluetooth link, while still sending the data to the DAC. Rewrite the Slave program to send the received data to its DAC. Download and run these programs. Both Master and Slave DACs will output a sine wave. Explore with care if there is any delay, and what data rate can be sustained. You may wish to do some of these tests with a triangular or square wave instead.

■

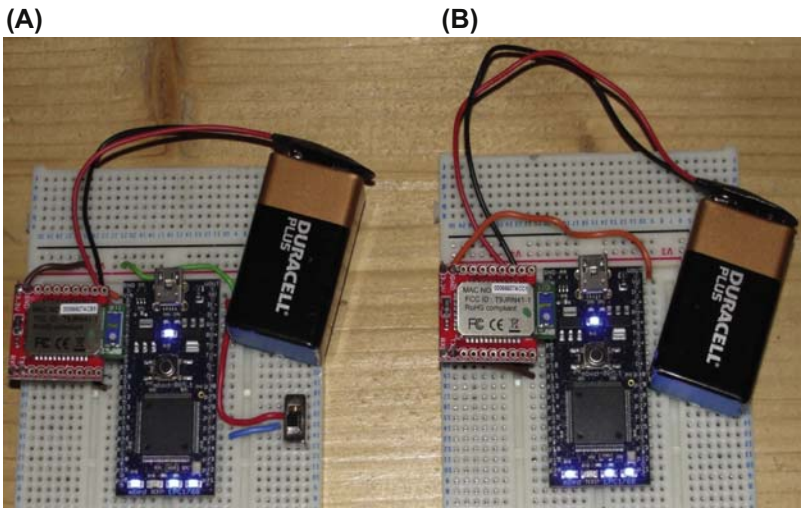

Figure 11.10
Two battery powered mbeds communicating via RN-41 Bluetooth modules (A) Bluetooth master with switch input (B) Bluetooth slave without switch.

11.2.7 Evaluating Bluetooth

Bluetooth is an exciting technology that allows short range wireless communication. This has many valuable applications where wires are intrusive, expensive or difficult to install. Recent enhancements to Bluetooth have enabled streaming of high quality audio data and increased range, so the opportunities and applications for Bluetooth are continuously growing. An important new version of the protocol is Bluetooth Smart, previously called Bluetooth Low Energy (BLE); this is intended for low power applications. Space does not allow us to cover BLE in this book. However it is interesting to note that some mbed enabled boards have this capability, and there is support information on the mbed web site.

11.3 Zigbee

11.3.1 Introducing Zigbee

While Bluetooth is continuously pressed for ever increasing data rates, there are other applications where high data rates are not the main criterion of interest. Instead, some situations require a combination of low data rates, extreme low power, and continuous operation over months or years.

Zigbee is intended for low power systems, with low data rates. It applies and builds on the IEEE 802.15.4 Low-Rate WPAN standard (Table 11.1). Like Bluetooth it operates in the

ISM bands of the radio spectrum. Zigbee is managed by members of the Zigbee Alliance [6]. Its name refers to the waggle dance of bees when they return to the hive after seeking nectar. (If this name seems completely odd to you, remember this – the bees are themselves little data carriers, communicating through their "dance" information about the location of pollen they have found.) Zigbee has some similarities to Bluetooth, but aims to be simpler, cheaper, with smaller software overhead and with different target applications. Like Bluetooth, Zigbee devices apply spread spectrum communication.

There are three Zigbee device types:

The end device: This is the simplest device, with just enough capability to undertake simple measurement actions, and pass back the data. It can only do this to its "parent", i.e. the router or coordinator which allows it to join. It is likely to spend a large part of its time in the Sleep mode. It wakes up briefly, just to confirm it is still part of the network.

The router: After joining a Zigbee PAN, it can receive and transmit data, and can allow further routers and end devices to join the network. It may need to buffer data, if an End Device is asleep. It can also exercise a useful function (e.g. measurement). It cannot sleep.

The coordinator: This is the most capable Zigbee device; there can be only one in a network. It launches a network by selecting a PAN ID, and a channel to communicate over. It can allow routers and end devices to join the network. The coordinator cannot sleep, so must normally be mains powered.

Zigbee is particularly appropriate for home automation, and other measurement and control systems, with the ability to use small, cheap microcontrollers. Data rates vary from 20 kbps (in 858 MHz band) to 250 kbps (in the 2.4 GHz band). Because of this low data rate expectation, Zigbee end devices are able to implement Sleep modes; power consumption can hence be minimal.

Each network, once established, is defined by a 64-bit (previously 16-bit) PAN ID (Personal Area Network identifier). All Zigbee devices have a 16-bit and a 64-bit address. The latter – sometimes called the extended address – is assigned at manufacture, and is unique. However, the device receives a 16-bit address as it joins a network; hence this address is sometimes called the *network address*. This address is not permanent. If the device left the network and then re-joined, it would probably be assigned a different address. Zigbee data can be sent unicast (from one device to a specific target device) or broadcast (throughout the entire network).

Fig. 11.11 shows example Zigbee networks of increasing complexity. This shows the three device types, and how they might connect. The simplest connection is just a pair, of which one must be a coordinator, and the other can be a router or end device. A step up in

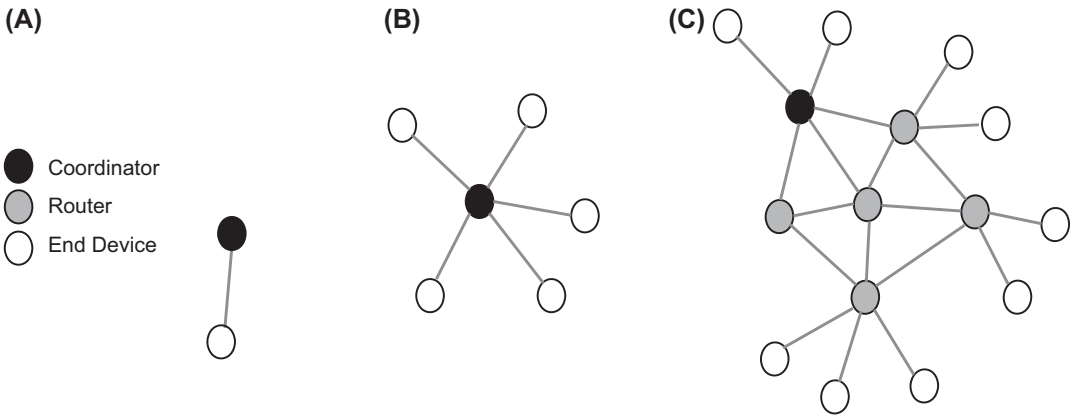

Figure 11.11
Zigbee Networks (A) pair (B) star (C) mesh.

complexity is the star, with a single coordinator linking to a surrounding set of end devices. Finally there is an advanced mesh structure, fully exploiting the Zigbee capability. Here routers take on an important role. Think of this diagram installed in a smart multi-storey building. The coordinator might be placed in a central location. Each floor has a router, and distributed end devices monitor temperature, air quality and light conditions. Because Zigbee devices can transfer data, a wider physical range is achieved.

11.3.2 Introducing XBee Wireless Modules

The XBee wireless modules, made by Digi International, can be used to rapidly configure Zigbee networks. There are a variety of these modules; some are superseded, and not all communicate with each other. We apply here the "ZB" types; these are configurable and flexible, and should work with Zigbee compliant devices from other makers. Different firmware versions allow them to take on coordinator, router or end device roles. Two of these modules are pictured in Fig. 11.12, one with a whip, and one with a PCB antenna. There are standard versions and PRO versions, which tend to be higher power. Summary data is given in Table 11.4.

The XBees operate in either AT (Application Transparent) or API mode. In AT mode the radio link is effectively transparent; data sent to one will immediately be transmitted to the module whose destination address is held by the transmitter. This is the simplest, and default, configuration; it reflects behaviour we have seen with the Bluetooth RN-41 devices. Like the RN-41, AT mode can operate in transparent or Command forms. As we shall see, the API mode allows considerably more sophistication. A network can contain a mix of modules operating in both modes, configured according to the role they play.

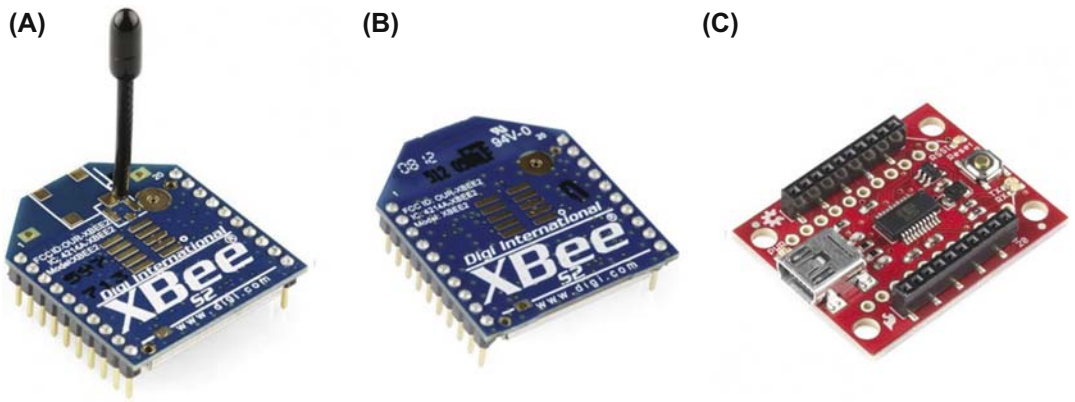

Figure 11.12
XBee modules (A) XBee ZB with whip antenna (B) XBee ZB with pcb antenna (C) XBee Explorer USB interface. *Images courtesy of SparkFun Electronics.*

Table 11.4: XBee module characteristics.

Variable	XBee	XBee PRO (International Version)
Range, Indoor/Urban	Up to 40 m	Up to 60 m
Range, outdoor/line of sight	Up to 120 m	Up to 1500 m
Transmit power	2 mW	10 mW
Transmit peak current	40 mA	170 mA
Power-down current	< 1 µA	3.5 µA (typical)
Data throughput	Up to 35,000 bps	Up to 35,000 bps

Importantly, XBees are not just conduits of data. They have input and output capability with, for example, pins that can be configured to read analog or digital data.

11.3.3 Linking to the XBee From a PC

When setting up Bluetooth links earlier in this chapter, we had the distinct advantage that the modern PC is equipped with Bluetooth, so we were able to make quick and easy connection. However PCs don't have Zigbee capability. Hence it is useful to be able to set that up. This requires two extra pieces of kit. The first is a USB interface, such as the Explorer, shown in Fig. 11.12C. This contains a USB-to-serial converter, which links with the XBee, plus various useful diagnostic and support features. The second is the official interface software from Digi, known as XCTU, which can be downloaded from the Digi web site, [7]. This brings the Explorer interface to life, and allows diagnostic testing

of an XBee, downloading of new firmware, plus access to remote XBee devices, by radio link. Using XCTU, the XBee device can be configured as coordinator, router or end device.

To try the following practical work for yourself, you will need to download XCTU, and have the Zigbee-related parts listed in Appendix D. Recognising that software is continuously revised, it is useful to view the excellent introductory video on the Digi XCTU page, linked from Ref. [7]. Here we use XCTU version 6.3.0.

To proceed, carefully mount an XBee into the USB Explorer, and connect to your PC. Launch XCTU, and click on the "Discover Radio Modules…" icon, which appears top left of the screen. This is the symbol with the magnifying glass sign on a little XBee outline, seen in Fig. 11.13. Accept the default offerings in the pop-ups which follow, and the software will then proceed to identify the XBee you've plugged in, displaying data as shown on the left of Fig. 11.13. You will be invited to add this to the XCTU display. Clicking on this panel also calls up a much more detailed screen, as shown in part on the right of the Figure. You can scroll down this to access a very wide range of details about the XBee, or search for a particular parameter from the search box. Parameters displayed can be read, and written to. The buttons across the top of this block, Read – Write – Default – Update – Profile, provide access to the main features of XCTU.

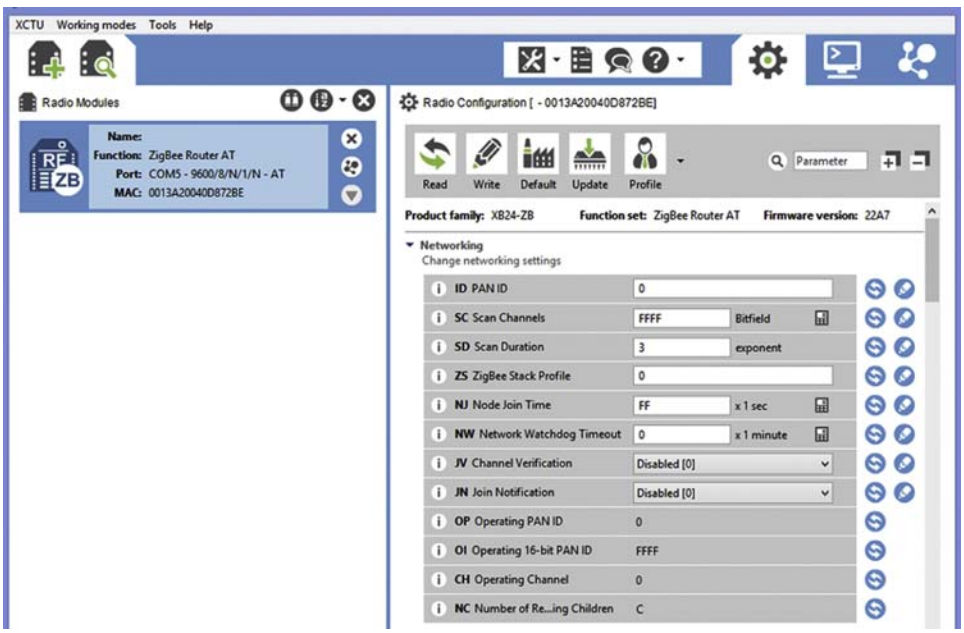

Figure 11.13
XCTU main screen, showing a Router module having been discovered.

Label and number your XBees, and write down the MAC address of this first one, you will need it later. Note that all XBee radios have 0013 A200 as their higher bytes. For example, the experiments which follow were done with XBees with these addresses:

Coordinator: 0013 A200 40D8 72BE Router: 0013 A200 40D8 72B2

Disconnect the USB Explorer, carefully take out the first device, and put in the second. Reconnect, and again "discover" it through XCTU, and get it displayed on the screen, as in Fig. 11.13. You will need to "remove" the previous device from the display, when prompted. Record its MAC address.

11.3.4 Configuring an XBee Pair

We will now set up a coordinator – router XBee pair, ready to communicate with each other. With one XBee loaded in the USB Explorer – this will become the coordinator – click the "Update" button on the XCTU screen. You will be offered a range of firmware that can be transferred to the XBee module, as shown in Fig. 11.14. Broadly, these are coordinator, router and end device, in AT or API mode, with further options in terms of data input and output. Select "Zigbee Coordinator AT", and the newest Firmware version. Then click "Update". The updating process takes a minute or more, with interesting information appearing on the screen as it does so. At the end, the original front screen of Fig. 11.13 is updated, to show the new module status.

Each XBee must also have the same PAN ID selected, in the range 0 to 0xFFFFFFFFFFFFFFFF, and each must have the address of the other as its destination address. Complete this through XCTU. Fig. 11.15 shows the coordinator being configured, with the router's address set up as destination. A PAN ID of 0123456789ABCDEF has been arbitrarily chosen (a simpler one might have been better!).

Now disconnect the USB Explorer, and put in the second XBee, which will be used as router. If this XBee is new, you will probably see from the XCTU display that it is already configured as a "Zigbee Router AT". If not, download this firmware to it, as done with the first device. Configure it with the same PAN address as the coordinator, and set the coordinator MAC address as destination.

11.3.5 Implementing a Zigbee Link with XBee and the mbed

We now move to setting up a Zigbee link with a PC. This is going to feel very similar to the sections earlier in the chapter, when we set up Bluetooth. Indeed Programme Example 11.5 is just about the same as Example 11.1; the "Zigbee-ness" of the link is for now cleverly contained with the XBee.

Connect the router XBee in the circuit of Fig. 11.16. This looks suspiciously like the circuit of Fig. 11.6. An easy way to do this is to mount the XBee in a "breakout board", as

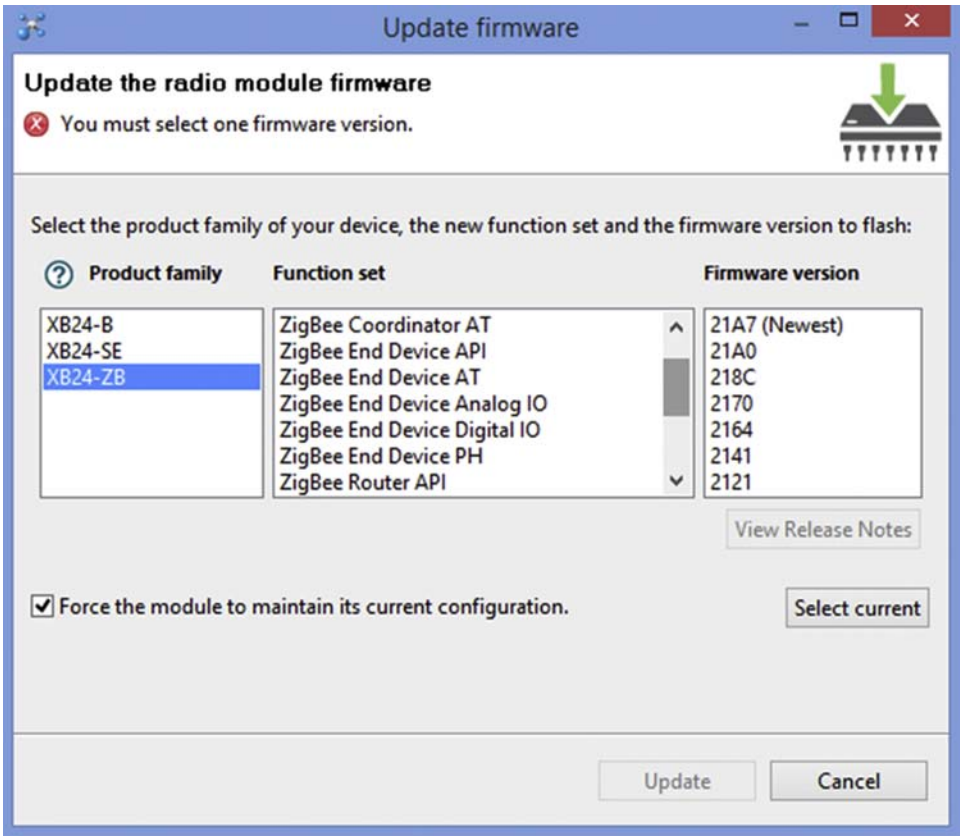

Figure 11.14
Updating the XBee firmware.

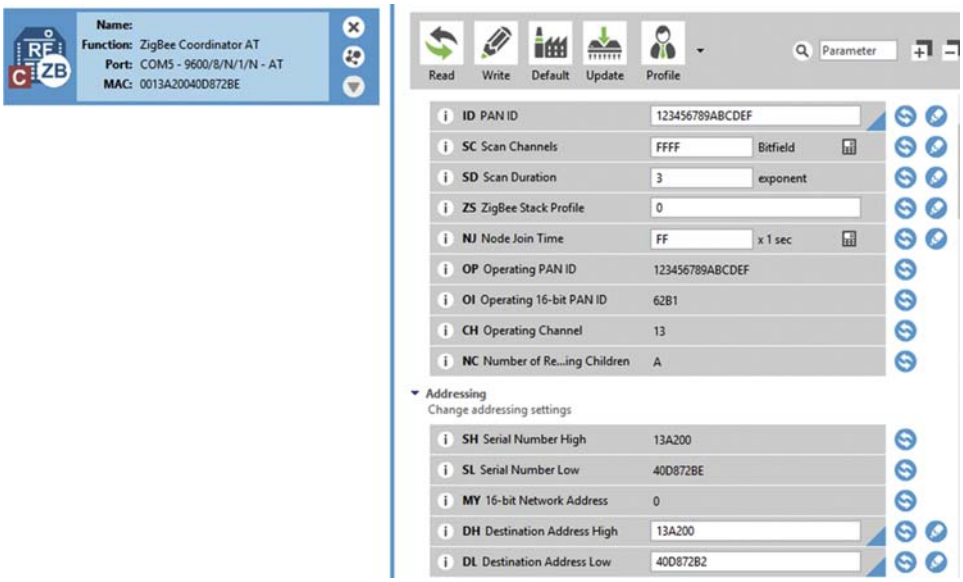

Figure 11.15
Setting PAN ID and addresses with XCTU.

listed in Appendix D, and solder jumper wires from this, to be connected into a breadboard holding the mbed. Compile and download Programme Example 11.5 into the mbed. Put the coordinator XBee in the USB Explorer, plugged into the host PC. Set up Tera Term or CoolTerm, and let things run. The terminal should continuously count from 0 to 9, as in the earlier Bluetooth example, with the mbed LEDs counting up in synchronism. Your first Zigbee link is up and running! You'll be even more convinced if you power the router circuit from a battery, for example as in Fig. 11.10.

```
/* Program Example 11.5: Zigbee serial test data program
Data is transferred from mbed to PC via Zigbee.
  Requires a set of "paired" XBee modules.     */
#include "mbed.h"
Serial xbee(p9,p10);    //name the serial port xbee
BusOut led(LED4,LED3,LED2,LED1);

int main() {
  xbee.baud(9600);  // set baud rate for xbee
  while (1) {
    for (char x=0x30;x<=0x39;x++){  // ASCII numerical characters 0-9
      xbee.putc(x);   // send test char data on serial to XBee
      led = x & 0x0F;              // set LEDs to count in binary
      wait(0.5);
    }
  }
}
```

Program Example 11.5: Zigbee serial test, mbed to PC

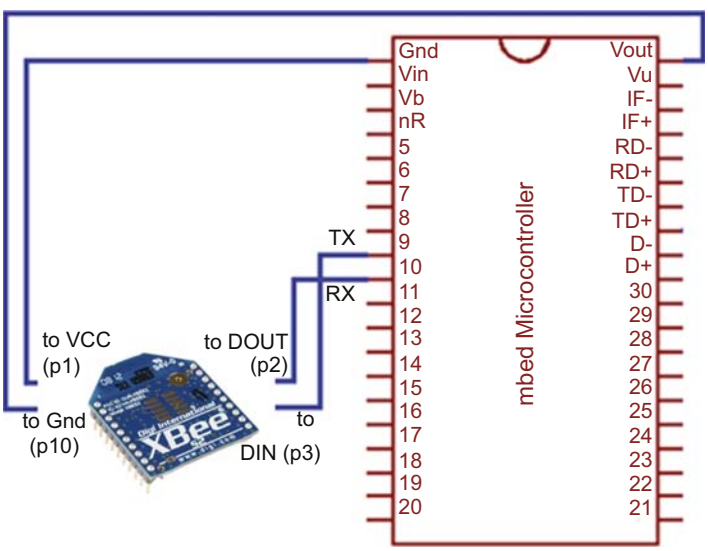

Figure 11.16
Connecting an mbed to the Xbee.

It's a simple step now to replicate Program Example 11.2, and demonstrate data moving in the other direction. Program Example 11.6 does this. Download it into the mbed, keeping the same physical setup as in the previous example. With Tera Term active, the PC keyboard number presses should be reflected in the mbed LEDs.

```
/* Program Example 11.6: Zigbee
Data is transferred bidirectionally between mbed and PC via Zigbee */
#include "mbed.h"
Serial xbee(p9,p10);      //set up the serial port and name xbee
BusOut led(LED4,LED3,LED2,LED1);

int main() {
  xbee.baud(9600);     // set up baud rate
  xbee.printf("Serial data test: outputs received data to LEDs\n\r");
  while (1) {
    if (xbee.readable()) {   // if data available
      char x=xbee.getc();    // get data
      led=x;                 // output LSByte to LEDs
    }
  }
}
```

Program Example 11.6: Zigbee bidirectional data test

App Board

It is a further simple step to set up an XBee to XBee link, with no PC intervention. In some ways this replicates the link of Fig. 11.10. The mbed application board has a very useful XBee socket, item 12 on Fig. 2.7, with connections shown in Table 2.1. We use this as the coordinator. The router should be placed in (or remain in) the circuit of Fig. 11.16. The overall system is represented in Fig. 11.17.

Programme Examples 11.7 and 11.8 should be downloaded and run in their respective mbeds. All program features should be familiar. The library of Table 8.5 is used for the

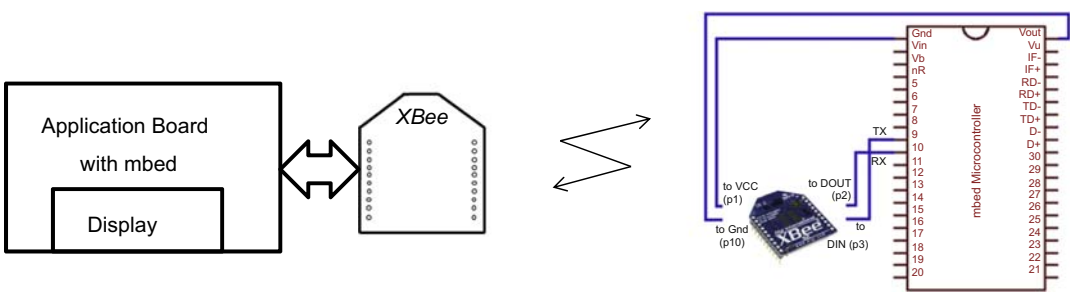

Figure 11.17
Zigbee link applying the app board.

C12832 LCD. The coordinator should either detect and display the incoming "data" from the router, or it should display "Wireless link lost!".

```
/* Program Example 11.7
Paired Zigbees - coordinator program
Requires a paired set of XBees, configured in XCTU
Hardware: XBee and mbed located in app board        */

#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11);
Serial xbee(p9,p10);       //name the serial port xbee
BusOut led(LED4,LED3,LED2,LED1);
char x,j;
int main(){
   lcd.cls();          //clear lcd screen
   lcd.locate(0,3);    //locate the cursor
   lcd.printf("Zigbee Test Program");
   wait (1);
   while(1) {
    if (xbee.readable()){   // if data available
    j=0;
    x=xbee.getc();          // get data
    led=x;                  // output LSByte to LEDs
    lcd.locate(0,15);
    lcd.printf("Remote data = %d",x);
    }
    else {        //count no of times there is no data
      j++;
      wait (0.01);
      }
      if (j>250){
       lcd.locate(0,15);
       lcd.printf("Wireless link lost!");
       j=0;      //reset counter
       }
    }
}
```

Program Example 11.7: XBee to Xbee link – coordinator

```
/* Program Example 11.8
Paired Zigbees - router program
The router generates data and sends to coordinator  */

#include "mbed.h"
Serial xbee(p9,p10);              //name the serial port xbee
BusOut led(LED4,LED3,LED2,LED1);
char x;
```

```
int main() {
  xbee.baud(9600);
  while (1) {
    x++;                 // increment x
    if (x>0x0F)          // limit to 4 bits
      x=0;
    xbee.putc(x);         // send char data on serial
    led = x;              // set LEDs to count in binary
    wait(0.5);
  }
}
```

Program Example 11.8: XBee to Xbee link – router

11.3.6 Introducing the XBee API

So far we've done some neat things with the XBees, but in truth we've done little more than replace a length of wire with a radio link, as we did with Bluetooth. We haven't got anywhere near the flexibility that the introductory section on Zigbee seemed to imply. This section is intended to provide a glimpse of how the real power of Zigbee can be implemented. To avoid this becoming a book on Zigbee (and that would be a very interesting thing to do!), this remains a glimpse. However it should give the motivation, confidence and pointers to go much further.

We have already mentioned the distinction between operating in AT and API modes. It is the API mode which unlocks the power of Zigbee through the XBee. Using the API, the XBee can for example change the destination address dynamically, perform error checking, reconfigure remote radios, and exploit the remote XBee I/O capability. It is for this last reason that the next example uses the API.

In API mode all data is packaged into *frames*, which carry both the data itself, plus a range of ID, addressing, and error-checking capability. The general frame structure is shown in Fig. 11.18. The frame always starts with the identifier 0x7E, then two bytes which indicate the length of the following "Frame Data" section. The first byte of the Frame Data is a single byte Command Identifier (also called API identifier). This indicates the purpose of the frame, and thus determines the structure of the Frame Data section

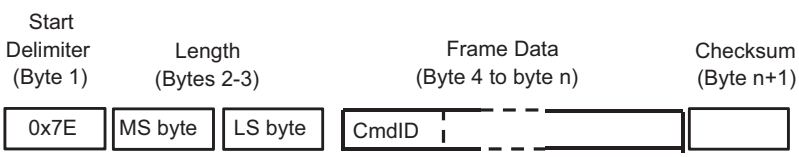

Figure 11.18
General XBee API data frame structure.

Table 11.5: Frame Structure for "Zigbee I/O data sample Rx indicator."

Byte	Purpose	Description
1	Start delimiter	0x7E
2	Length MS byte	
3	Length LS byte	
4	Command ID	0x92
5-12	64-bit address of sender	MS byte first
13–14	16-bit address of sender	MS byte first
15	Receive options	01: Packet acknowledged 02: Packet was broadcast
16	Number of sample sets	Number of sample sets in payload (always set to 1)
17–18	Digital Channel Mask	Bitmask field indicating which digital IO lines on the sending device have sampling enabled (if any)
19	Analog Channel Mask	Bitmask field indicating which analogue IO lines on the sending device have sampling enabled (if any)
20–21	Digital samples (if any)	All enabled digital inputs are mapped within these two bytes
22-	Analog samples	Each enabled analog input returns a 2-byte value indicating the ADC output.
	Checksum	Single byte: disregarding first three bytes, add all other bytes, keep only the lowest 8 bits of the result and subtract the result from 0xFF.

which follows. The final byte is always a *Checksum*, which provides a simple means of error checking.

There are 18 API identifiers possible, which allow commands, data or status information to be transferred. Each has its own distinct format within the Frame Data. In this simple demonstration we will use the remote XBee to make readings on just one of its analog inputs. This requires use of the "Zigbee IO Data Sample Rx Indicator", with frame structure shown in Table 11.5.

11.3.7 Applying the XBee API

We now set up a simple Zigbee link, in which one XBee is set up as a standalone device, making analog measurements. It periodically transmits data values to an XBee coordinator, running in an mbed app board. This is pictured in Fig. 11.19. The coordinator runs in API mode. The other XBee is configured as a router. Its pin 20, labelled ADO/DIO0, will be configured as analog input; to play this simple role, it can remain in AT mode. The XBee ADC is a 10-bit device, and has an input range of 1.2 V. A potentiometer has been chosen to provide this simple analog source, as shown. This has itself been placed in a potential divider, through the addition of the 20 kΩ resistor, to match the ADC input range. The 3.3 V can be supplied by a battery (e.g. a pair of AA cells), or a voltage regulator, or

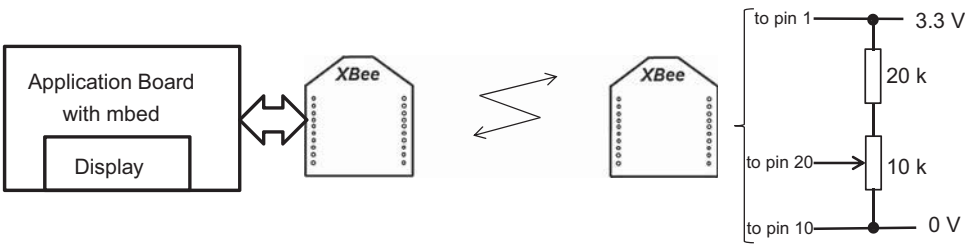

Figure 11.19
Diagnostic circuit for API trial.

Table 11.6: Router configuration for API trial.

Code	Name	Select This Value	Comment
ID	PAN ID	A 16-digit value of your choice. Keep it simple.	Must be the same as the coordinator
JV	Channel Verification	Enabled [1]	Router attempts to join coordinator when powered up
D0	AD0/DIO0 configuration	ADC [2]	Selects this pin as analog input
IR	IO Sampling Rate	200 × 1 ms	The input will be sampled and transmitted with a period of 200 ms

bench supply. A simple solution if you have a spare mbed available is to use it just for its 3.3 V regulated output, powering it through USB or battery pack.

The XBees need to be configured in turn, using an XBee USB explorer connected to a laptop running XCTU. The router should have the firmware "Zigbee router AT" downloaded, as seen in Fig. 11.14. It is then configured through XCTU, in the screen shown in Fig. 11.15, with values given in Table 11.6. Nothing else should be changed.

The coordinator should have the firmware "Zigbee coordinator API" downloaded. It should be configured through XCTU to have the same PAN address as the router. Nothing else needs to be changed.

Program Example 11.9 will run on the mbed in the application board, acting as coordinator. The mbed waits for transmissions from the router, determined by the internal 200 ms timer that has been set up. It should not be too difficult to work out the program features. Broadly speaking, it unpicks the data frame of Table 11.5, and displays chosen values within the frame. For this simple demo, certain important items are discarded. It is easy to change what is selected and displayed, to inspect other parts of the frame. Compile and download the program.

```
/* Simple XBee API application
Requires XBee and mbed in app board, plus remote XBee
The coordinator receives data from router, and displays on lcd */

#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11);
Serial xbee(p9,p10);              //name the serial port xbee
DigitalOut led (LED4);
char x,xhi,xlo,j,len,ftype;       //some useful internal variables
int result;

int main(){
   lcd.cls();                     //clear lcd screen
   lcd.locate(0,3);               //locate the cursor
   lcd.printf("Zigbee API Test");
   wait (1);

   while(1) {
    if (xbee.readable())          // if data is available
     x=xbee.getc();               // get data
      if (x==0x7E){               //test for start of frame
       led=1;                     //set diagnostic LED
       while (xbee.readable()==0); //wait for next byte
       x=xbee.getc(); //discard length msb, assume zero
       while (xbee.readable()==0);
       len=xbee.getc(); //save length lsb
       while (xbee.readable()==0);
       ftype=xbee.getc(); //save frame type
       j=1;
     //now discard 15 bytes: i.e. 64 bit address, 16-bit address, et al
       while(j<16){
        while (xbee.readable()==0);  //wait
        x=xbee.getc(); //discard
        j++;
       }
      while (xbee.readable()==0);
      xhi=xbee.getc();                        //get ms ADC byte
      while (xbee.readable()==0);
      xlo=xbee.getc();                        //get ls ADC byte
      result = xhi*256 + xlo;     //convert to 16 bit number
      lcd.locate(0,15);
      lcd.printf("length = %d",len);      //include values as desired
      lcd.printf("  data = %d",result);  //display result
      led = 0;
        }                                 //end of if
      }                                   //end of while(1)
  }
```

Program Example 11.9: Applying the XBee API – coordinator program

With both ends of the Fig. 11.19 data link powered, the XBees should find each other; LED 4 on the mbed then starts flashing 5 times a second, each time the data frame is detected. The app board display should show the text message "Zigbee API Test", followed by values for frame length and data. If the potentiometer is adjusted, the value on the screen should be immediately updated, giving a numerical value in the range 0 to 1023.

Exercise 11.5

Note the value of len displayed when Program Example 11.9 runs. Consider Quiz question 10.

1. Now display variable ftype instead of len, and explain the value found.
2. Display the least significant two bytes of the 64-bit sender address. Do they give the expected value?

■

Exercise 11.6

Replace the potentiometer in Fig. 11.9 with a light sensor, as seen in Fig. 5.7. Select suitable resistor value(s) for the XBee analog input range. Transmit light data over the Zigbee link.

■

You can reconfigure the XBee router in this example as an end device, and get the same behaviour. Link the device to XCTU with the USB Explorer, and download the "Zigbee End Device AT" firmware. You will be able to replicate all the settings of Table 11.6, except for the Channel Verification, code JV, which is an instruction only used by routers. With the circuit of Fig. 11.19 reconnected, the system behaviour should be the same as when a router was used.

This example, though still simple, gets you into the world of the XBee API, through which the true power of Zigbee can be exploited. Clearly the Program Example used, and the XBee settings made, neglect many important Zigbee features, which a more sophisticated program would make use of.

11.3.8 Conclusion on Zigbee and Further Work

There is huge scope to go much further with Zigbee, and these wonderful little XBee devices. However this book is about the mbed, and quite a bit more XBee-specific

knowledge would be needed. To progress further, try other API data frames, set up simple networks with router and end devices, and explore Sleep mode. One or more of these resources can be used:

- Ref. [8] is the official XBee data sheet. While it is a formidable 155-page document, there is much here of great value. Chapter 9 for example gives full detail on the API mode.
- Ref. [9] is an excellent guide to Zigbee and use of XBees. Unfortunately it doesn't use mbeds. Hoping that you are mbed–committed, you can still gain much good knowledge from its example projects, and with a bit of care convert them to the mbed environment.
- Ref. [10] contains an official library from Digi for XBee use. The authors haven't tried this, but it should allow you to access the full power of a professionally-written library.
- Ref. [11] provides some useful further insight and guidance on use of XCTU, and XBee configuration.

11.4 Mini Projects

11.4.1 Bluetooth Mini Project

Using the set-up of Fig. 11.10, add a liquid crystal display to each mbed device. Add also a unique sensor such as a light sensor, temperature sensor or ultrasonic range finder to each mbed. Implement a two-way communication program which allows the data from both sensors to be displayed on both displays simultaneously.

11.4.2 Zigbee Mini Project

Depending on how many XBees you have, set up a network with multiple nodes. The routers and/or end devices should be fitted with temperature sensors. Place one in each of several rooms in a house or other building, with an app board as coordinator. Display the temperature in each room.

Chapter Review

- Wireless links exploit the characteristics of the electromagnetic spectrum, notably in radio, infra-red or visible light.
- A wide range of protocols and technologies exist to implement wireless links across personal, local, neighbourhood and wide area networks.
- Bluetooth is a complex yet effective protocol defined within the IEEE 802 group, which allows Bluetooth-enabled devices to connect and transfer data wirelessly, with potentially high data rates.
- The RN-41 module can be used to give an mbed Bluetooth capability.
- Zigbee is another important protocol defined within the IEEE 802 group, targeted towards low data rate, distributed measurement systems, and extreme low power.

- The XBee module provides Zigbee capability, which can be linked to the mbed. The XBee has its own on-board processing power and input/out capability, so can also readily act as a stand-alone device.

Quiz

1. An FM signal has a frequency of 103.0 MHz. What is its wavelength?
2. The antenna for a Bluetooth device operating at 2.4 GHz is to have a "quarter-wavelength antenna", meaning that the length of the antenna should be one quarter of the radio signal wavelength. How long should it be?
3. Explain briefly the term *Spread Spectrum.* Why does this reduce interference between channels?
4. How many layers are there in the OSI model? Using Bluetooth and Zigbee as examples, justify the statement "The IEEE 802 standards tend to link to the … Data Link Layer and Physical Layer." You may need to do a little more background reading to do this.
5. What is the communication range of a Class 1 Bluetooth device?
6. What does the term 'MAC address' refer to?
7. Briefly describe the three types of Zigbee device, and the roles they play.
8. What is the nominal range of an XBee PRO device, when operating out of doors?
9. Think of a building which you know well, of moderate size. Sketch a Zigbee network for it, which monitors temperature, light and air quality in each room. Show where you would place the coordinator, routers and end devices.
10. When Program Example 11.9 runs, the app board display shows a frame length of 18. Explain carefully why this is so.

References

[1] C. Bear, W. Stallings, Wireless Communication Networks and Systems, Pearson, 2016.
[2] IEEE 802.15 Working Group for WPAN. http://www.ieee802.org/.
[3] The Official Bluetooth Website. http://www.bluetooth.com.
[4] Class 1 Bluetooth® Module with EDR Support, Microchip, March 2014. Document DS50002280A.
[5] Bluetooth Data Module Command Reference & Advanced Information User's Guide, March 2013. Roving Networks Version 0.02.
[6] The Zigbee Alliance Website. http://www.zigbee.org/.
[7] The Digi Website. http://www.digi.com.
[8] Xbee®/Xbee-PRO® ZB RF Modules. (2010). Digi International Inc.
[9] R. Faludi, Building Wireless Sensor Networks, O'Reilly, 2011.
[10] Digi International Library for mbed-XBee Communication. https://developer.mbed.org/teams/Digi-International-Inc/code/XBeeLib/.
[11] Z. Dannelly. XBee Basic Setup. https://developer.mbed.org/users/dannellyz/notebook/xbee-basic-setup/.