Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

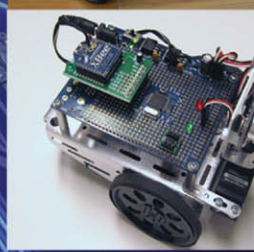
We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

The Official Guide

PROGRAMMING AND CUSTOMIZING THE MULTICORE PROPELLER™ MICROCONTROLLER



PARALLAX INC.

Shane Avery, Chip Gracey, Vern Graner, Martin Hebel, Joshua Hintze,
André LaMothe, Andy Lindsay, Jeff Martin, and Hanno Sander

PROGRAMMING AND CUSTOMIZING THE MULTICORE PROPELLER™ MICROCONTROLLER

5

WIRELESSLY NETWORKING PROPELLER CHIPS

Martin Hebel

Introduction

This chapter looks at how your Propeller can be part of a *wireless sensor network* (WSN) to share data through wireless communications. WSNs are not intended for large data transfers, such as files, but small amounts of data back and forth. The Propeller is an amazing controller, and its ability to perform parallel processing makes data communications fast and simple for use in a WSN. While the main task is being carried out, other cogs can be sending or receiving data on the network.

With a lot to discuss and learn along the way, the final completed project of this chapter, depicted in Fig. 5-1, will be a three-node network that has:

- A tilt-controller node transmitting drive and control data.
- A robot (bot) node that receives the data; has a compass and ultrasonic range finder; and is transmitting data on drive, range, and direction. It also has the ability to "map" what is in front of it for remote display.
- A node that accepts data from the bot and displays the information graphically on the TV.

This chapter highlights communications to, from, and between Propeller chips using XBee® transceivers from Digi International. Topics covered in this chapter include:

- Networking and XBee overview
- PC-to-XBee communications
- Configuring the XBee manually and with the Propeller

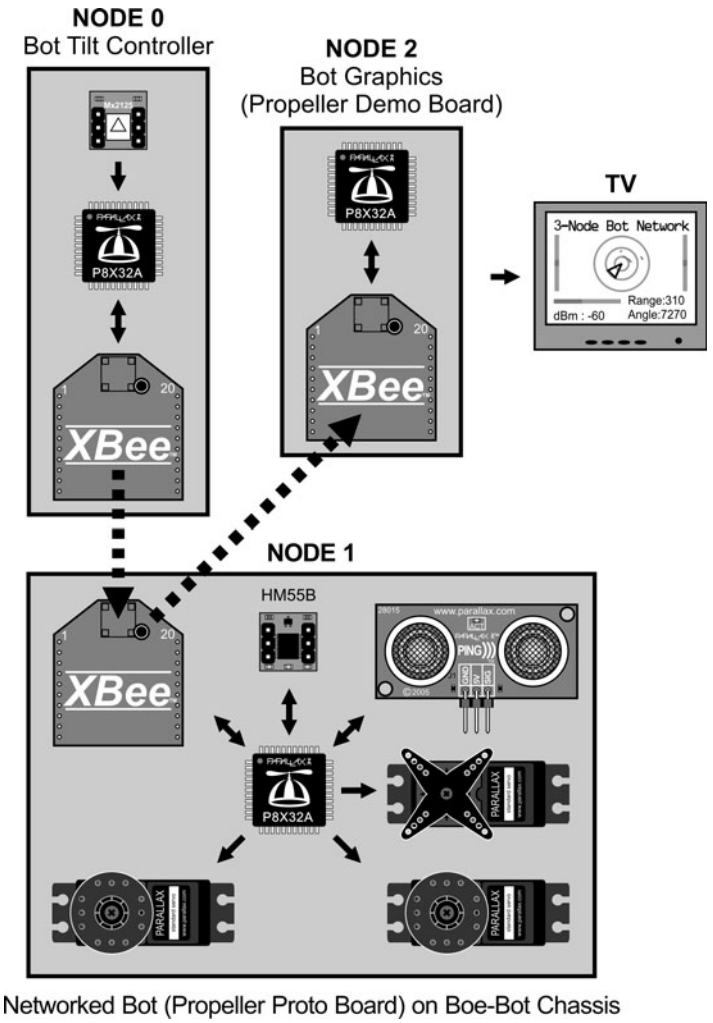


Figure 5-1 Three-node network for monitoring and control.

- PC-to-Propeller and Propeller-to-Propeller communications with the XBee
- Transparent and API data modes of the XBee
- Forming a multi-node Propeller network for robot control and monitoring

This chapter will work through several examples of communications, but really, the intent and focus is on *how* to perform the communications with the Propeller. It is left to you, the reader, to take the principles discussed, combine them with your imagination or needs, and develop a Propeller network of your own. Many other projects and information from this text can be combined with this chapter for truly amazing projects!

Resources: Demo code and other resources for this chapter are available for free download from ftp.propeller-chip.com/PCMProp/Chapter_05.

Overview of Networking and XBee Transceivers

The ability to communicate wirelessly has had such a significant impact on personal and data communications that many today cannot envision life without the use of cell phones, Wi-Fi networks and Bluetooth® features in personal devices. The ability of these devices to communicate on their respective networks (even your Bluetooth headset forms a network with the player) relies on key features:

- The use of addressing to send data to specific destination devices and to identify the source of the data
- The use of framing and packets to encompass the data itself in a "package" with necessary information (such as the destination address)
- The use of error checking to ensure the data arrives at the destination without errors
- The use of acknowledgements back to the source so that the sender knows the data arrived correctly at its destination

Simple two-device (or two-node) systems may not need all these features. It's really dependent on the needs of the network, but if ensuring data arrives correctly to an intended destination is vital, then these features are a must.

The XBee uses a fully implemented protocol and communicates on a *low-rate wireless personal area network* (LR-WPAN), sometimes referred to as a *wireless sensor network* (WSN) with RF data rates of 250 kbps between nodes. For the seasoned network readers, LR-PANs operate using IEEE 802.15.4, a standardized protocol similar to Wi-Fi (IEEE 802.11) and Bluetooth (IEEE 802.15.1). The XBee is currently available in the XBee 802.15.4 series and the XBee ZigBee/Mesh series. The 802.15.4 series (often referred to as Series 1) is the simplest and allows point-to-point communications on a network. The ZigBee/Mesh series (Series 2) uses the ZigBee® communications standard on top of 802.15.4 for WSNs to provide self-healing mesh networks with routing. This chapter will focus exclusively on the XBee 802.15.4 and its higher-power sibling the XBee-Pro 802.15.4. These will be referred to as simply the XBee.

Key benefits of using the XBee include the ability to perform addressing of individual nodes on the network, data is fully error-checked and delivery acknowledged, and data can be sent and received transparently—simply send and receive data as if the link between devices were directly wired. XBees operate in the 2.4 GHz frequency spectrum.

An image and a drawing of an XBee are shown in Fig. 5-2. The XBee is a 20-pin module with 2.0 mm pin spacing. This can cause some aggravation when working with breadboards and protoboards, which have 2.54 mm (0.1 in) pin spacing, but solutions to this will be addressed.

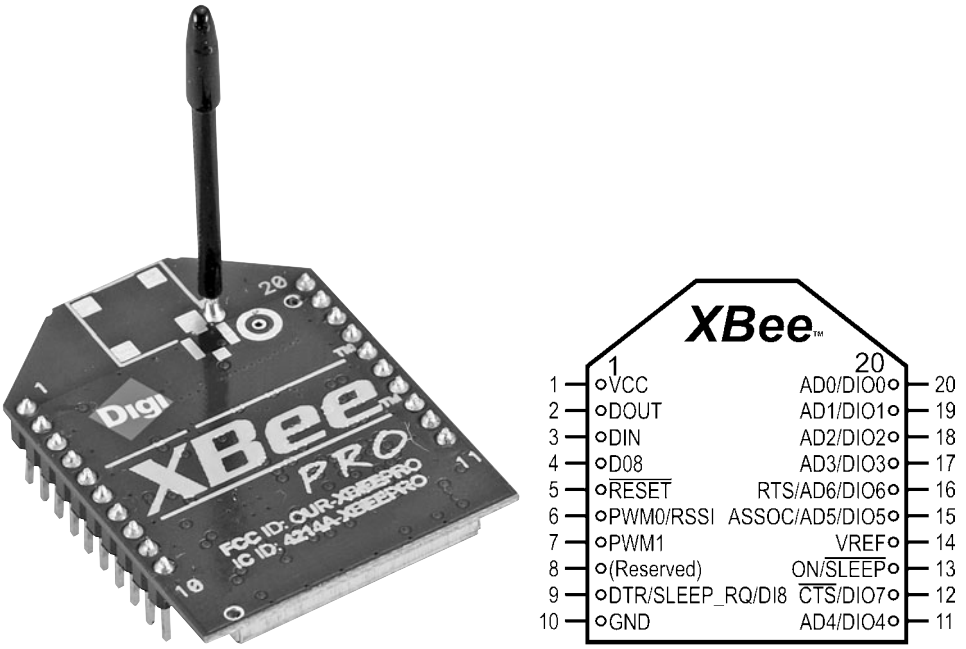


Figure 5-2 XBee module and pins.

Don't get scared! The XBee has a large number of pins, but for most of this chapter, we will use only four:

- Vcc, Pin 1: 2.8 V to 3.4 V (Propeller Vdd voltage)
- GND, Pin 10 (Propeller Vss)
- DOUT, Pin 2: Data out of the XBee (data received by Propeller)
- DIN, Pin 3: Data into the XBee (data to be transmitted by Propeller)

Other pins include a sleep pin (Sleep_RQ) for low power consumption, flow control pins (RTS/CTS), analog-to-digital (ADC) inputs, digital inputs and outputs (DIO), among others. This chapter will discuss some of these other pin functions, but the focus is on simply sending and receiving data between the Propeller and XBees using the DOUT and DIN pins.

Note: Please see the XBee manuals on Digi's web site for in-depth discussion and information: www.digi.com and included in the distribution files.

The XBee has a current draw of around 50 mA and a power output of 1 mW with a range of about 100 m (300 ft) outdoors. The XBee-Pro has a current draw of 55 mA when idle or receiving data and 250 mA when transmitting. With a power output of 100 mW, it has a range outdoors of 1600 m (1 mi) line sight. They both have sleep

modes, with current draws of less than 10 μA, but can't send or receive data while sleeping. There are different antenna styles as well, though the whip antenna is probably the most popular.

Tip: Don't get too excited about the distances. Line-of-sight communications rely on height as well as distance. Due to ground reflections and deconstructive interference (Fresnel losses), the heights of the antennas need to be taken into account. For good communications at 100 m, a height of 1.4 m (4.6 ft) is recommended.

Information: For more insight on distance, height issues, and calculations, search the web for "Fresnel clearance calculation."

Though the XBee is ready to go right out of the box, it is feature-rich and can be configured for specific applications.

Hardware Used in This Chapter

The following is a list of hardware used in this chapter and their sources, but as you read through, you'll find it's not written in stone. We recommend you read through the chapter to understand how the hardware is used before making an expensive investment.

- 2—Propeller Demo Boards (Parallax)
- 1—Propeller Proto Board (Parallax)
- 1—Prop Plug (Parallax)
- 3—XBee 802.15.4 (Series 1) modem/transceivers (www.digikey.com)
- 3—AppBee-SIP-LV XBee carrier boards (www.selmaware.com or other styles available on www.sparkfun.com)
- 1—PING))) ultrasonic sensor (Parallax)
- 1—HM55B compass module (Parallax)
- 1—Memsic 2125 accelerometer/inclinometer (Parallax)
- 1—Boe-Bot chassis (Parallax)
- 1—Ping Servo Mounting Bracket Kit (Parallax)
- 2—Additional Boe-Bot battery holders or other portable battery source
- Miscellaneous resistors

Testing and Configuring the XBee

An important step in constructing a complex project is to make sure the individual devices work properly and their use is understood. In this section, the XBees will be tested, configuration settings explored, and means of configuring these devices discussed.

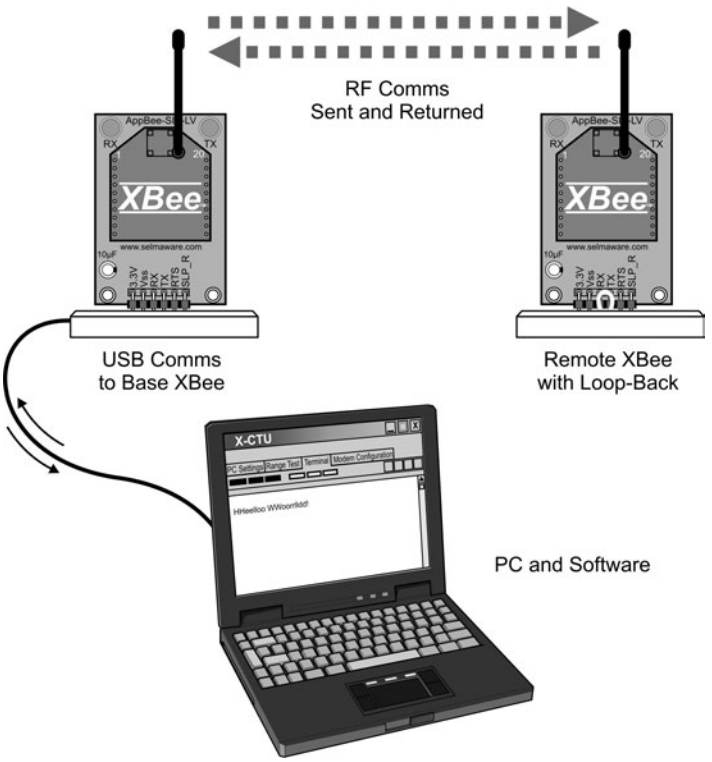


Figure 5-3 Configuration and testing diagram.

Figure 5-3 shows the diagram for this test. A PC will communicate directly to an XBee, and a remote XBee is set up with a loop-back jumper. In the loop-back, the DOUT line of the XBee is tied to its DIN so that any RF data it receives is looped back into the device to send it out again via RF.

The following is a list of the hardware and software used for this test, but there are many ways to achieve the same results. Essentially, a means is needed to communicate to an XBee serially from the PC and means to supply power to the base and remote XBees.

Equipment and other software:

- 2—Propeller Demo Boards (Parallax)
- 2—XBees (www.digikey.com)
- 1—Prop Plug (Parallax)
- 1—AppBee-SIP-LV from Selmaware Solutions (www.selmaware.com)
- X-CTU software from Digi International (www.digi.com)

The AppBee-SIP-LV is simply a carrier board for the XBee providing 3.3 V power from the Demo Board and access to I/O in a breadboard-compatible header. Figure 5-4 shows the AppBee-SIP-LV and a drawing of the physical connections to the XBee.

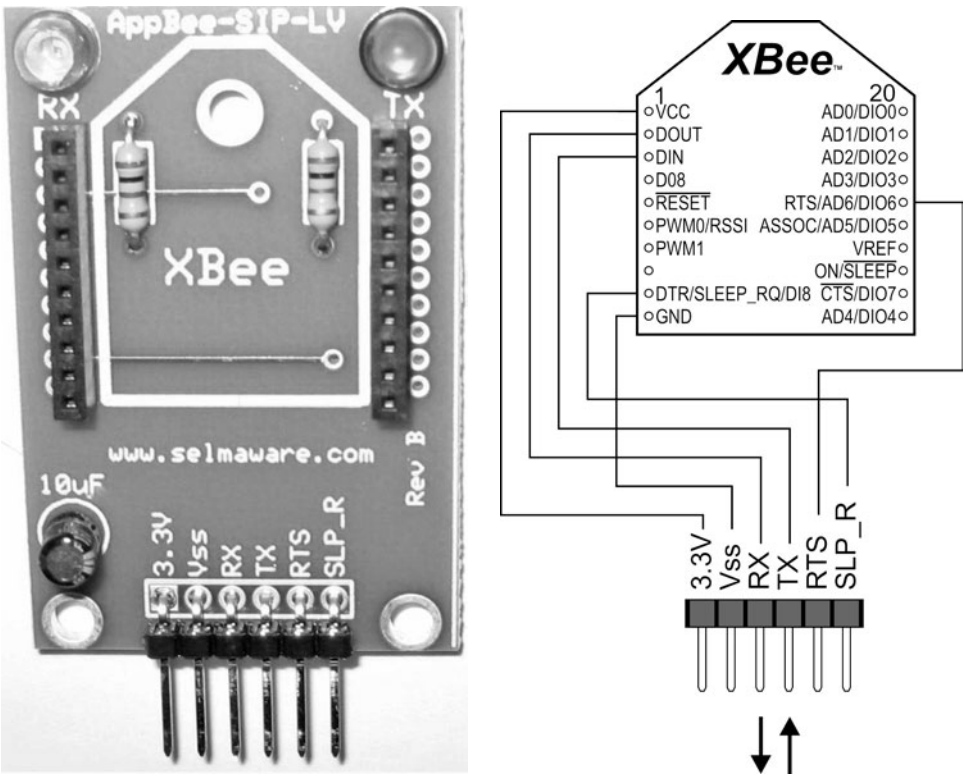


Figure 5-4 AppBee-SIP-LV carrier board and drawing with physical connections.

Tip: Another good source of carrier boards and other XBee accessories is www.sparkfun.com. Search their web site for XBee.

ESTABLISHING PC-TO-XBee COMMUNICATIONS

The first task is to communicate with the XBee directly from the PC for configuration changes and monitoring. Figure 5-5 shows two ways of establishing communications: using the Propeller as a serial pass-through device or communicating directly with the XBee using the Prop Plug as a serial interface. Either method allows the serial connection between the PC and the transceiver.

If you are using the Propeller to pass serial communications, the program Serial_Pass_Through.spin should be downloaded using F11. If the serial communications port is closed in the software, the Propeller may be cycled when the DTR is toggled, reloading the Propeller from EEPROM. Using F11 ensures a cycling of the Propeller will reload the correct program.

The program itself is simple but highlights the power of Propeller. Microcontrollers that provide multiserial communications are difficult to find. Two instances of the

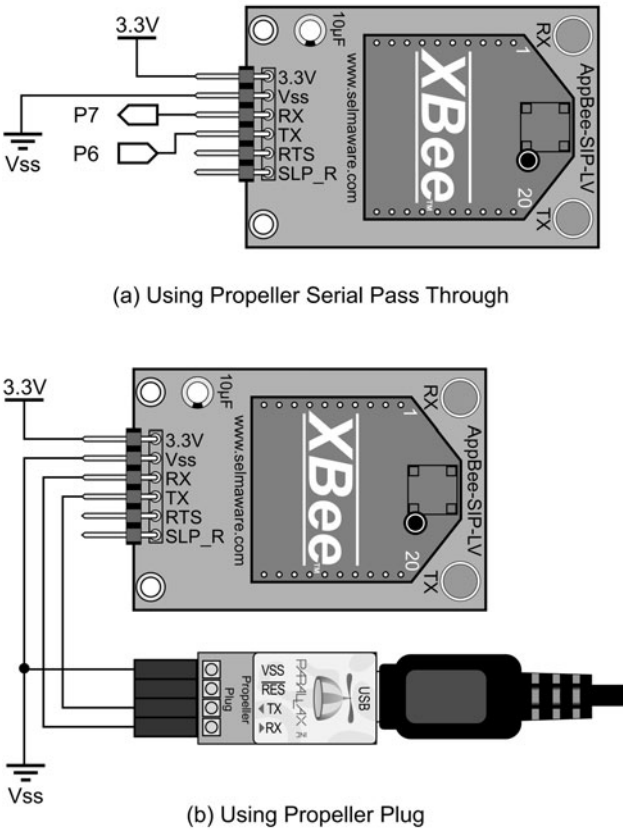


Figure 5-5 **Two methods of PC communications with XBee.**

FullDuplexSerial object establish the transparent link. Data from the PC is sent to the XBee, and data from the XBee is sent to the PC; with each method in separate cogs, it allows transfer speeds tested up to 115,200 bps. But for now we need to stick to 9600 bps since that is the default configuration on the XBee.

```
OBJ
  PC    : "FullDuplexSerial"
  XB    : "FullDuplexSerial"

Pub  Start

  PC.start(PC_Rx, PC_Tx, 0, PC_Baud) ' Initialize comms for PC
  XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
  cognew(PC_Comms,@stack)        ' Start cog for XBee--> PC comms
```

```
  PC.rxFlush                    ' Empty buffer for data from PC
  repeat
    XB.tx(PC.rx)                ' Accept data from PC and send to XBee

Pub PC_Comms
  XB.rxFlush                    ' Empty buffer for data from XB
  repeat
    PC.tx(XB.rx)                ' Accept data from XBee and send to PC
```

Caution: Watch the I/O numbers! If another configuration is used, modify the pin numbers in the CON section of the code.

If you are using the Propeller for passing serial data:

✓ Connect the hardware as shown in Fig. 5-5*a*.
✓ Download the Serial_Pass_Through.spin program to the Propeller using F11.

If you are using the Prop Plug to communicate directly, connect it as shown in Fig. 5-5*b*.

✓ If you haven't yet, download and install the X-CTU software available in the distributed files or from Digi's web site. There is no need to check for updates—this can take a long time and the basic installation has all that is needed for now.
✓ Open the X-CTU software. It should look similar to Fig. 5-6. Select the COM port that your Propeller is communicating through.
✓ At this point, use the Test/Query pushbutton to test communications with the XBee.

Caution: As always, only one software package can access the same COM port at any time. You'll get used to slapping your head when you can't communicate as you go between the Propeller tool software and X-CTU!

Tip: If communications fail, recheck your hardware and pin numbers, reload the Propeller program, and verify no other software is using the COM port. If you continue to have problems and it is not a brand-new XBee, the serial baud rate may have been changed or the XBee may be in API mode—test various baud rates and check the API box to test.

If all went well, you may have seen the RX and TX lights blink on the board and received a message informing you communications were okay, along with the firmware version on the XBee.

✓ Select the Modem Configuration tab on the X-CTU software.
✓ If your XBee was reconfigured, this would be a good time to click the Restore button to return it to the default configuration.
✓ Click the Read button.

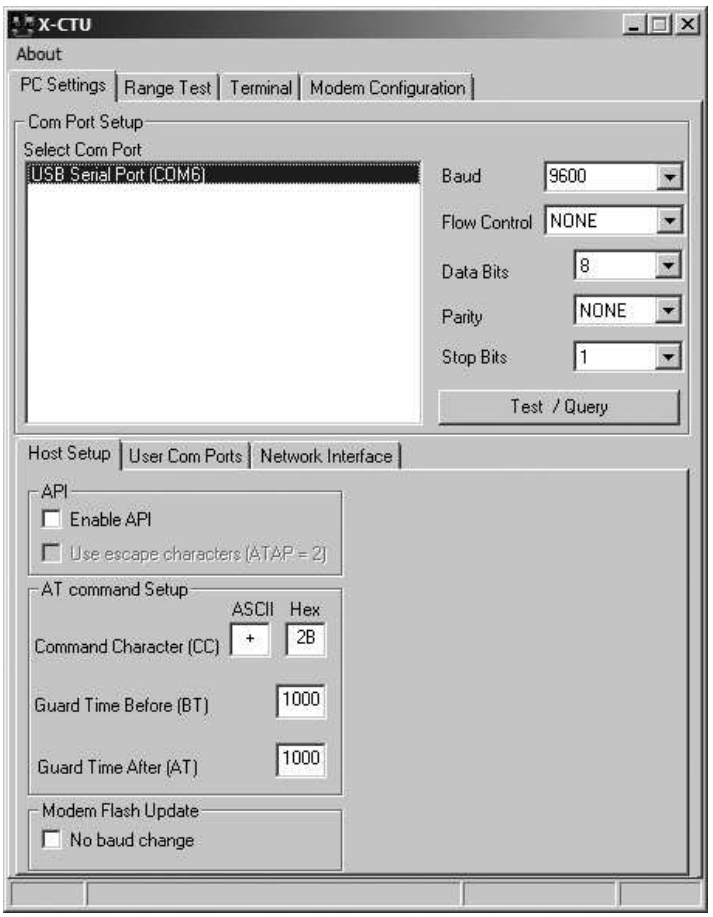


Figure 5-6 **X-CTU software showing COM port selection.**

The screen should have loaded with the configuration setting of the XBee as shown in Fig. 5-7. Many of them will be explained shortly—we're only going to use a handful of the settings available. But for now, let's test out some wireless communications.

TALKING XBee TO XBee USING LOOP-BACK

With a second XBee, supply power and connect a jumper between DOUT and DIN (or RX and TX on the carrier board), as illustrated in Fig. 5-8, using the AppBee-SIP-LV carrier board (or similar). Do not connect to any Propeller I/O at this time—we are simply using the board for power. We used a second Demo Board for this test.

✓ Power up the remote XBee with loop-back jumper in place.
✓ Click the X-CTU Terminal tab.
✓ Type "Hello World!"

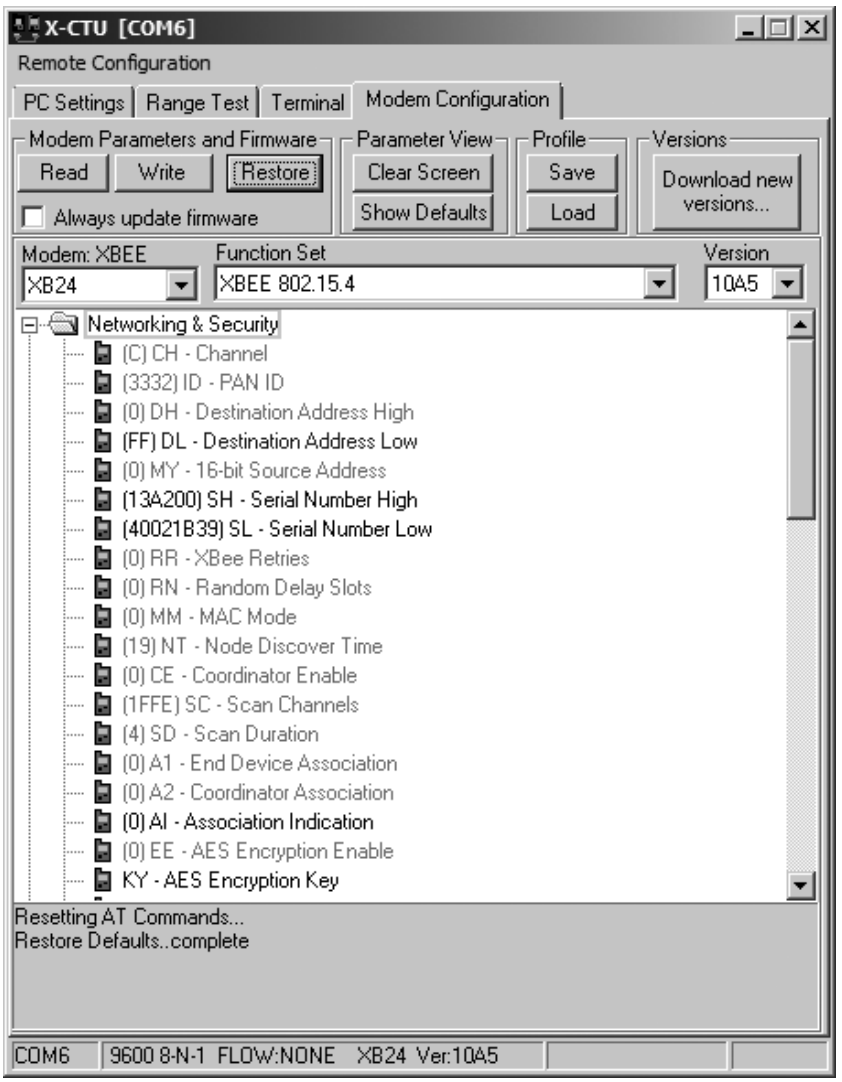


Figure 5-7 X-CTU software showing XBee configuration settings.

You should see the TX and RX lights flashing on both units (if using the AppBee carrier) and text in your Terminal window. You should see two of each character—what you typed in blue and what was echoed back and received in red—as shown in Fig. 5-9.

Tip: Having problems? If you don't see any data returning, be sure the remote XBee is connected properly. If it is not a new XBee, if may have been configured differently. Turn off both units and swap the XBees. After powering up, "Restore", the XBee to default configuration using the X-CTU button, read the second XBee using the X-CTU software, and test again.

Figure 5-8 **Remote XBee connections for loop-back.**

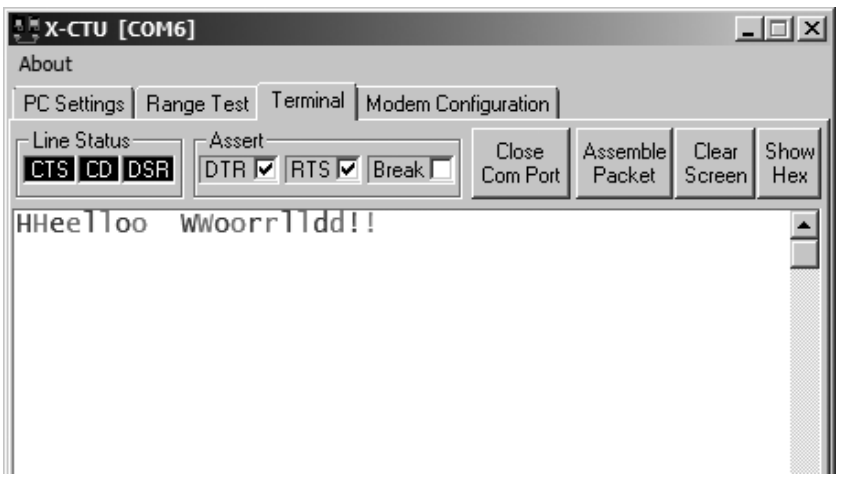


Figure 5-9 **X-CTU Terminal window.**

Tip: Beyond testing purposes, the X-CTU software is not essential, and any terminal program or other serial software package may be used, such as the PST Debug-LITE software used in previous chapters. Just ensure baud rates match between the software and the devices.

As noted, each character is transmitted as it is typed. The XBee can actually send a string of characters at once (up to 100), but it only waits so long before assembling a packet to be transmitted. We type too slowly to get multiple characters quickly enough with the default configuration, but we can assemble a packet of characters that will be kept together:

- ✓ On the X-CTU Terminal window, click Clear screen, and then click Assemble Packet.
- ✓ Type "Hello World!" in the packet box, and click Send Data.

You'll notice your text is returned as a single packet.

One last test is the range test. This allows you to monitor the signal strength from –40 dBm to the XBee's sensitivity limit of around –100 dBm by having the software repeatedly send out a packet to be echoed.

- ✓ Check the check box below the vertical RSSI (receiver signal strength indication).
- ✓ Click Start.
- ✓ Monitor the number of good packets received and signal strength.
- ✓ Block the area between the XBees or move the remote XBee to another room, and test the effect on RSSI level.

Note: In theory, you should never see a bad packet (malformed data) in the received data from the XBee, such as in the Terminal window. All data is error-checked and retried if there is no response or if the error check fails. You should receive either good data or no data at all. The serial-link issue with the XBee is a more probable cause than an RF issue with bad data.

Now that we have an RF link going, it's time to discuss and test some XBee configurations.

XBee CONFIGURATION SETTINGS

As seen, the XBee has numerous settings that can be configured. This configuration can be performed through the Configuration window, through the Terminal window, or through strings sent out from the Propeller. Let's first take a look at some of the more important settings shown in Table 5-1 for this chapter (we will use only a few) and others of interest should you delve deeper with your experiments. Click the Modem Configuration tab of the X-CTU software to view the settings. Clicking any setting will give a brief description and range of values at the bottom of the window.

OK, let's test out a few things:

- ✓ Test and verify your loop-back setup by sending a string.
- ✓ Under Modem Configuration, change DL to 1.
- ✓ Click Write.
- ✓ The XBee should be updated. Click Read and verify.
- ✓ Go to the Terminal window and type once again. You should get no response, and the remote RX light on the AppBee should not blink.

TABLE 5-1 SUMMARY OF PERTINENT XBee SETTINGS

COMMAND CODE	MEANING & USE
Networking & Security	
CH	Channel: Sets the operating frequency channel within the 2.4 GHz band. This may be modified to find to a clearer channel or to separate XBee networks.
ID	PAN ID: Essentially, the network ID. Different groups of XBee networks can be separated by setting up different PANs (personal area networks).
DL	Destination Low Address: The destination address where the transmitted packet is to be sent. We will use this often to define which node receives data. A hexadecimal value of FFFF performs a broadcast and sends data to all nodes on the PAN. The default value is 0.
MY	Source Address: Sets the address of the node itself. This will be used often in all our configurations. The default value is 0.
Sleep Modes	
SM	Sleep Mode: Allows the sleep mode to be selected for low power consumption (<10 µA). While we won't use it, a good choice is 1—Pin Hibernate. This would allow an output of the Propeller to put the XBee to sleep (using the Sleep Request pin) when it is not sending or expecting data.
Serial Interfacing	
BD	Interface Data Rate: Sets baud rate of the serial data into and out of the XBee.
AP	API Enable: Switches the XBee from transparent mode (AT) to a framed data version where the data must be manually framed with other information, such as address and checksum. This is a powerful mode and will be explored in this chapter.
RO	Packetization Timeout: In building a packet to be transmitted, the XBee waits a set length of time for another character. If not received in the set time, the packet is sent. This is why as we typed characters, each was sent and echoed back. This can be important to change if you have multiple units sending data to one node to ensure that all data sent is received as a single transmission from one unit; otherwise, you may get data from various nodes intermixed.
I/O Settings	
D0 – D8	Sets the function of the I/O pins on the XBee, such as digital output, input, ADC, RTS, CTS, and others.
IR	Sample Rate: The XBee can be configured to automatically send data from digital I/O or ADCs. It requires the receiving node to be in API mode and the data parsed for the I/O values.
Diagnostics	
DB	Received Signal Strength: The XBee can be polled to send back the RSSI level of the last packet received.
EC	CCA Failures: The protocol performs clear channel assessment (CCA)—that is, it listens to the RF levels before it transmits. If it cannot get an opening, the packet will fail and the CCA counter will be incremented.

EA	ACK Failures: If a packet is transmitted but receives no acknowledgement that data reached the destination, EA is incremented. The XBee performs two retries before failure. Additional retries can be added by using the RR setting.
AT Command Options	
CT	AT Command Timeout: Once in command mode, this sets how long of a delay before returning to normal operation.
GT	Guard Time: When switching into AT command mode, this defines how long the guard times should be (absence of data before the command line) so that accidental mode change is not performed.

By changing DL to 1, data is intended for an XBee at address 1. The default settings on XBees are a DL of 0 and an MY of 0. Previously, we were sending data to a node at address 0 from a node at address 0 and vice versa. Be aware, the XBee actually does receive data, sees it is not the intended node, and then dumps it instead of passing it to the DOUT pin (to which the RX LED is connected).

Let's now try configuring using the Terminal window. Due to timeouts, you may have to type a little fast, so you may need a few attempts. Enter the following lines—*do not* type what is in parentheses. `Press enter after each line except for +++.`

- ✓ (Wait three seconds since you typed anything last—this is guard time.)
- ✓ **+++** (*Do not* press ENTER.)
- ✓ (Wait a few more seconds and you should see that it is now in command mode.)
- ✓ `ATDL` (Requests the current DL value; it should return 1)
- ✓ `ATDL 0` (Sets the DL address to 0)
- ✓ `ATDL` (Again requests the DL address, which should be 0)
- ✓ `ATCN` (Exits AT command mode)
- ✓ `Hello World?`

If all went well, you should once again be getting echoes after changing the destination address back to 0. The waiting before and after the +++ is called the *guard time,* and it ensures that if a string containing +++ is sent, the unit won't flip into command mode inadvertently.

Tip: Permanent changes? Using the Modem Configuration feature of the X-CTU software, all changes are saved to nonvolatile memory and will still be in place after cycling power. Using the AT commands, the settings will revert to original values after cycling power, unless the ATWR (write) command is sent to write to nonvolatile memory.

The important aspect here is that just as we sent data strings to the Xbee for configuration changes, so can your Propeller configure the XBee through code. Multiple commands can be used in one line by separating them with commas. For example, the following sets DL to 0 and exits command mode: ATDL 0, CN.

TRY THESE!

- ✓ Try changing your MY address to 1 and sending data. You should see the remote unit receive and transmit, but you get nothing back. Why?
- ✓ Change your DL to FFFF. This is the broadcast address—any nodes on your network would receive it. Be sure to set MY back to 0 for the loop-back to work!
- ✓ Use the command ATND (Network Discovery). After a few seconds, you should see a list of other nodes in the network, including their MY address, two lines of the physical address (like a MAC address), and the RSSI level in hexadecimal.
- ✓ Use the command ATED (Energy Detect). You should see a list of about 11 hexadecimal values. This is the energy level seen on the various channels. Higher values are less noisy—a value such as 5A (hexadecimal), for example, converts to a level of –90 dBm.

✓ Use the Configuration tab to restore the XBee to its default values when done testing, or use the AT command ATRE, followed by ATWR, to save to memory.

UPDATING THE XBee VERSIONS

Just a note about the version of the XBees: In the Modem Configuration tab, you can see the version of firmware on your XBee, such as 1083, 10A5, or 10CD. Later versions are more capable. The majority of this chapter requires at least 1083. The firmware on the XBee can be updated by selecting a new version, checking Always update firmware, and clicking Write, but this requires more data lines than we have available with our configurations. A board such as the XBIB-U from Digi International or the WRL-08687, the XBee Explorer, from www.sparkfun.com (which can also double as a carrier board) is recommended. These boards can be used for direct USB access to the XBee as well as changing the firmware, and they supply power to the XBee.

Now that we can send and receive data and configure the XBee, we are ready to start using Spin and the Propeller to communicate via the XBee.

Sending Data from the Propeller to the PC

In this section we will equip a remote Propeller/XBee system with a couple of sensors and then transmit the data from the sensors back to the base XBee to send the data to

the PC for monitoring. The base can be the Propeller using serial pass-through, using the Prop Plug to the XBee, or using a dedicated XBee-to-PC board, as previously mentioned. The sensors used for testing are Parallax's HM55B compass module and the PING))) ultrasonic range finder. These devices will eventually assist in our robot project, but you are free to modify the code to use any of the sensors previously explored in this text.

Additional equipment:

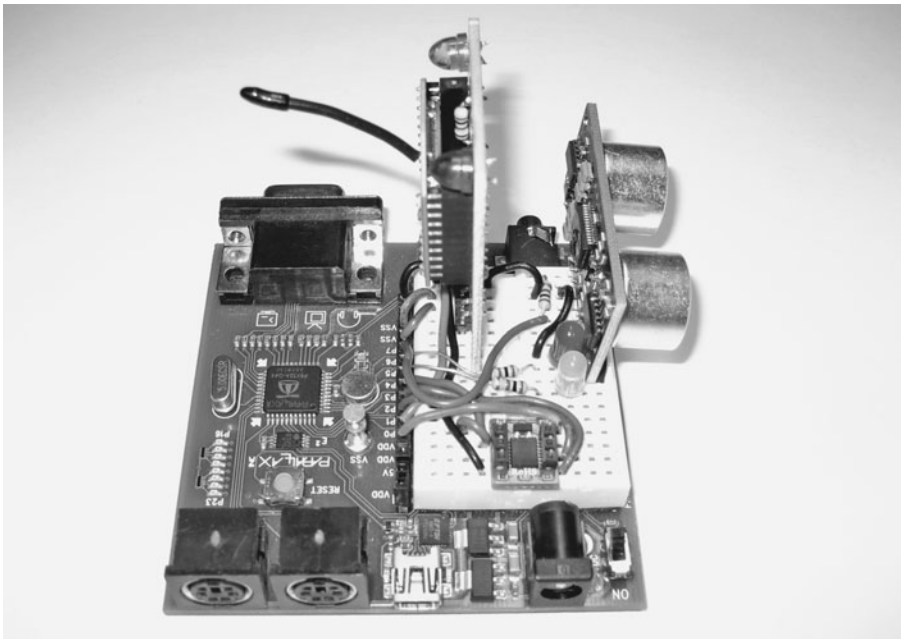
- HM55B Compass Module
- PING))) Ultrasonic Range Finder
- Or other sensors as desired, with appropriate code

Figure 5-10 is an image of the nodes. Even though we don't need to just yet, we will use this opportunity to set the DL address of the remote unit to 0 to ensure it is sending data to the base unit.

- ✓ Connect the PING))) sensor and HM55B compass on the remote unit as shown in Fig. 5-11. If a different I/O pin is used, update the pin numbers accordingly in the `CON` section of the code. Connect the LEDs as well; we will use them shortly.
- ✓ For the base unit XBee, open and clear the X-CTU Terminal window. Open the COM port if closed. Having that port in use will help ensure the correct Propeller is programmed.
- ✓ Download Simple_PC Monitoring_from_Remote.spin to the remote unit.
- ✓ Monitor the remote unit's LEDs—they should blink rapidly a few times after several seconds as the XBee is configured.
- ✓ Monitor the base unit's Terminal window. A "ready" message should be displayed, then the readings of the sensors should be reported every half-second.
- ✓ Test the compass bearing. It should read 0 to 8191 (roughly) as you rotate it, with 0 being approximately magnetic north.
- ✓ Test the range finder by placing an object in front and moving it in and out. The PING))) sensor will report distances from roughly 30 to 3000 mm (3 cm to 3 m).
- ✓ If either sensor fails to respond properly, check your connections and code.

Tip: The range finder has a fairly large angle of emission and detection. Test this by putting an object to the side of range finder and going in and out to determine how wide the angle is at different distances.

After initializing the XBee and compass, there is a three-second delay, +++ is sent followed by another three-second delay and the string of "ATDL 0, CN." Finally, a byte of 13 representing a CR or ENTER key is sent. The destination address is set to 0 and command mode is exited (CN) in exactly the same fashion as you did in the Terminal window.



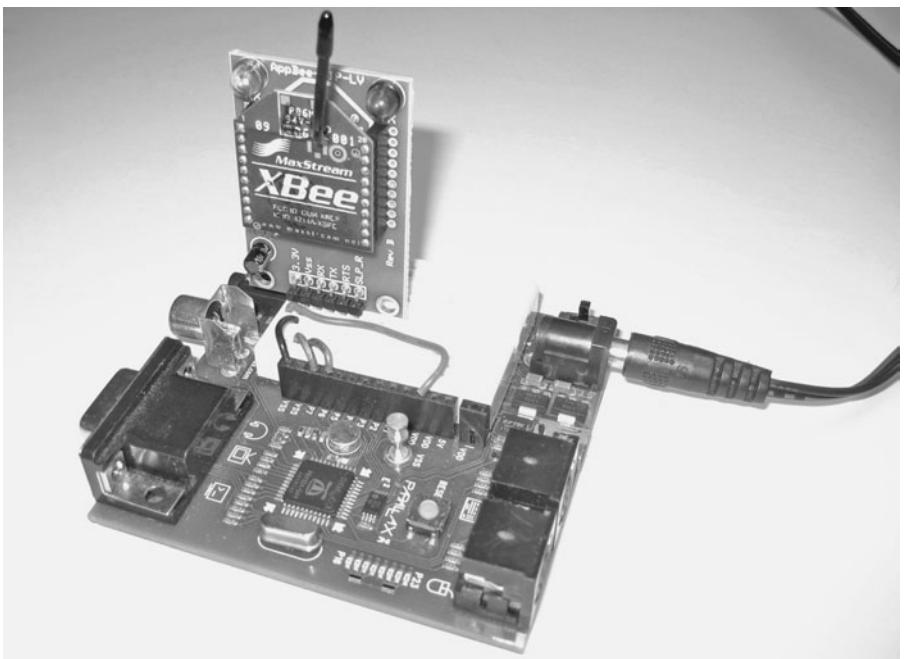


Figure 5-10 Base and remote nodes.

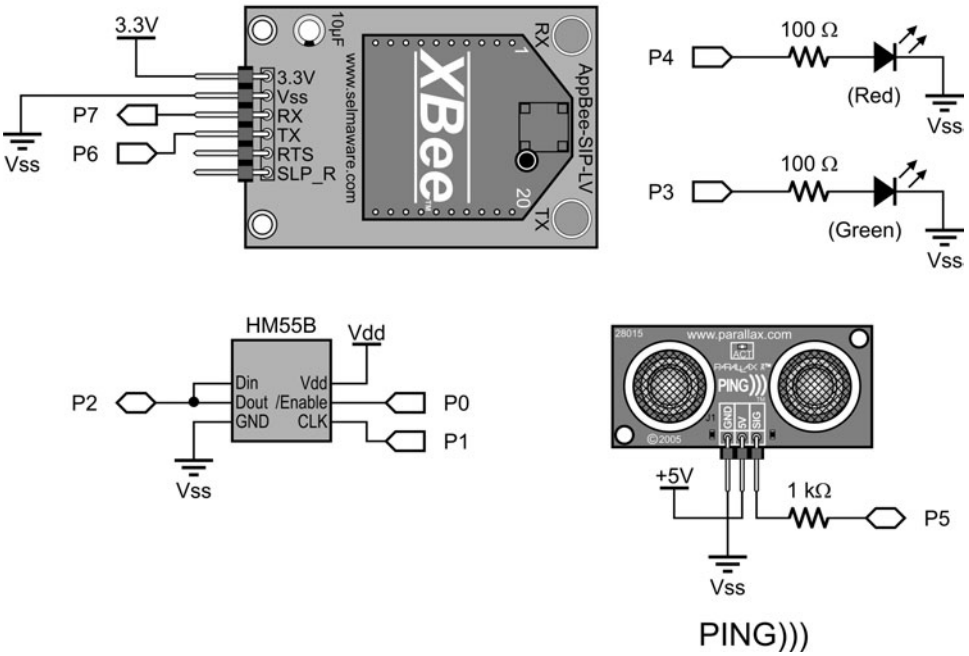


Figure 5-11 Remote unit with PING))) and compass.

```
delay(3000)                          ' Guard time for AT mode
XB.str(string("+++"))                ' Send AT command request
delay(3000)                          ' Guard time
XB.str(string("ATDL 0,CN"))          ' Send code to set DL = 0
XB.tx(13)                            ' Send carriage return
```

The command codes from the Propeller are passed to the XBee using the FullDuplexSerial object duplicating your serial terminal. Through this method any number of commands may be sent to the XBee for configuration changes on initialization or during operation. Those 3-second guard times can cause lag during operation, but we'll deal with that soon.

In the `SendData` method, you can see that the range and direction (`theta`) are read from the devices. Using a combination of text strings and `XB.dec` (decimal) methods, the data is sent to the base XBee, where it is passed through to the PC for monitoring in the Terminal window. Figure 5-12 is a sample output of received data.

```
repeat
  range := Ping.Millimeters(PING_Pin)      ' Get range in mm
  theta := HM55B.theta                     ' Get bearing (0-8191)
  XB.str(string(13,13,"Ping Range(mm): "))' Send string to base
  XB.dec(range)                            ' Send range as decimal
  XB.str(string(13,"Direction(0-8192): "))' Send string to base
  XB.dec(theta)                            ' Send bearing as decimal
  delay(500)                               ' Short delay before repeat
```

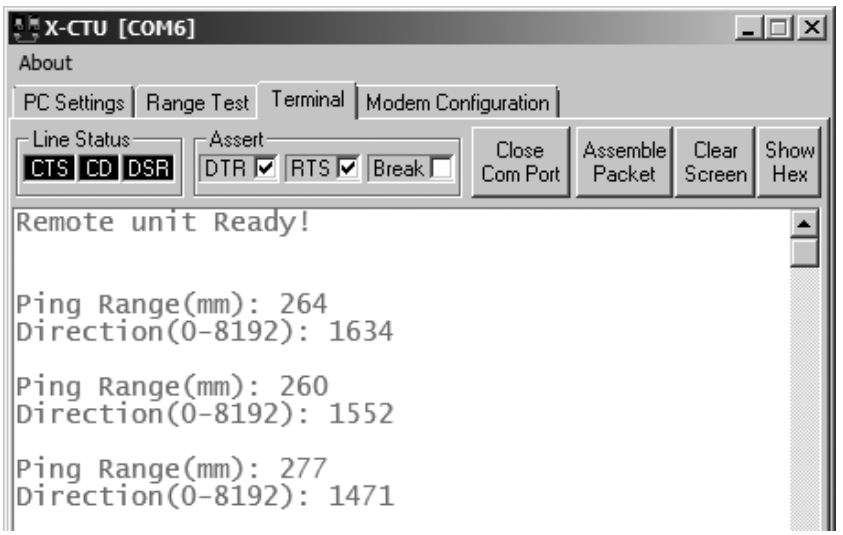


Figure 5-12 Sample output in Terminal window of range and bearing.

TRY IT!

✓ Try adding a simple device, such as a pushbutton, and reporting its state back to the PC. If you are out of I/O, you may remove the LEDs.

Polling Remote Nodes

In an LR-PAN, nodes typically come in one of three flavors:

- *Coordinators* help manage the network, from controlling communications to assigning information to devices.
- *End devices* are used to read and control devices on the network.
- *Routers* are used to pass data between nodes at distances too far to reach directly.

There is nothing prohibiting end devices from talking to one another, and once a network is established, the coordinator's job may come to an end. In this chapter we will refer to the base unit, the one at the PC, as a coordinator because it will help control communications and be a common collection point. Our remote nodes will be end devices that we will monitor and control.

Multinode communications can be tricky. Aspects to be dealt with include: Which node can send data when? When data arrives, who is it from? Do nodes need permission to talk or can they do so at any time? We need to ensure that nodes don't talk over one another

(causing collisions on the network) and that the receiving units know who the data is from in order to respond appropriately or take some other action. XBee, using IEEE 802.15.4, works similar to Wi-Fi. A node listens before it transmits to help ensure that no other node is transmitting at the time (this is *Clear Channel Assessment,* or CCA). Delivery of data is verified through acknowledgements. If the sender does not get a response, it tries again. This method is known as CSMA/CA or *Carrier Sense, Multiple Access/Collision Avoidance*. Unlike Ethernet, which uses collision detection (CSMA/CD), a node cannot listen once it starts transmitting so it cannot detect collisions.

So the data link layer of communications helps ensure data gets passed properly, but it still doesn't assist in higher-level functions controlling the who and when of communications. In the next section we will look at a method of using a Propeller acting as a coordinator to poll end devices for their data. USB works in much the same way—each device is polled one at a time to see if they need access or have data to send.

COORDINATOR MANUALLY POLLING REMOTE END DEVICES

A hardware configuration similar to the one from the previous section will be used, but this time, the Propeller needs to be in the communications chain at the base instead of simply using a Prop Plug for XBee communications. Also, to demonstrate control action, the two LEDs on the remote end device provide control action. You are welcome to have as many end points as you desire (well, up to 65,000), or just use one and change the end point's address to test. Figure 5-13 is a diagram of our network and hardware.

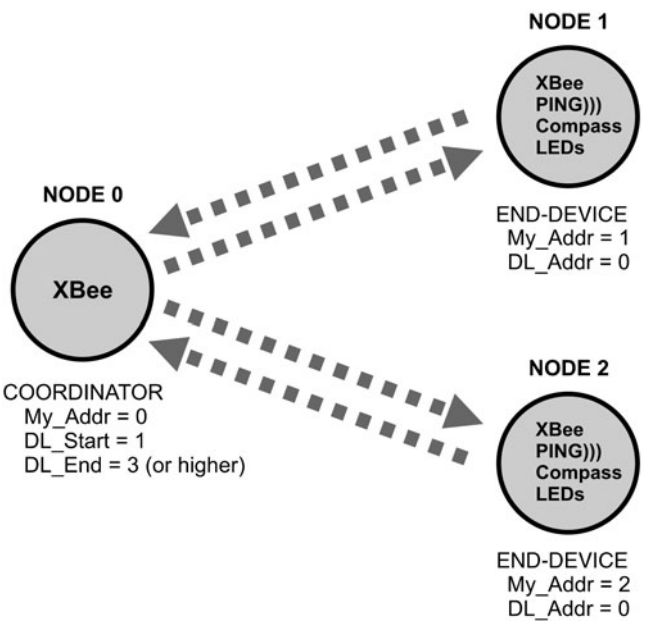


Figure 5-13 Hardware for coordinator polling.

In this example, the coordinator cycles through a range of end-point addresses by changing the DL value of the coordinator's XBee. It sends out codes and values to request data from each end point and to control the LEDs on each. Before allowing the coordinator to have control, we are going to manually test the control and responses.

- ✓ Add the LEDs to the remote end device.
- ✓ Open Acquisition_with_Control_End.spin.
- ✓ For each end device, number the constant `MY_Addr` in the `CON` section of the code sequentially from 1 up, skipping a few numbers to test "unresponsive nodes."
- ✓ Download Acquisition_with_Control_End.spin to each remote end device.
- ✓ Use the Propeller for serial pass-through or another PC-to-XBee configuration at the PC.
- ✓ Change the DL of the coordinator/base XBee to 1.
- ✓ In the Terminal window, type some p's and c's. If your end point at address 1 is awake, you should get values back for compass bearing and range finder distance.
- ✓ For this next test, use the "Assemble Packet" window. Type and send the following:
  - Type i3 and then hit Enter.
  - Type 1 and then hit Enter.
  - Click Send.
- ✓ Change the 1 to a 0 and send again.
- ✓ Test again by using 4 instead of 3.
- ✓ What you should see is LEDs on P3 and P4 turning on with 1 and off with 0.

Figure 5-14 is an image of our communications test.

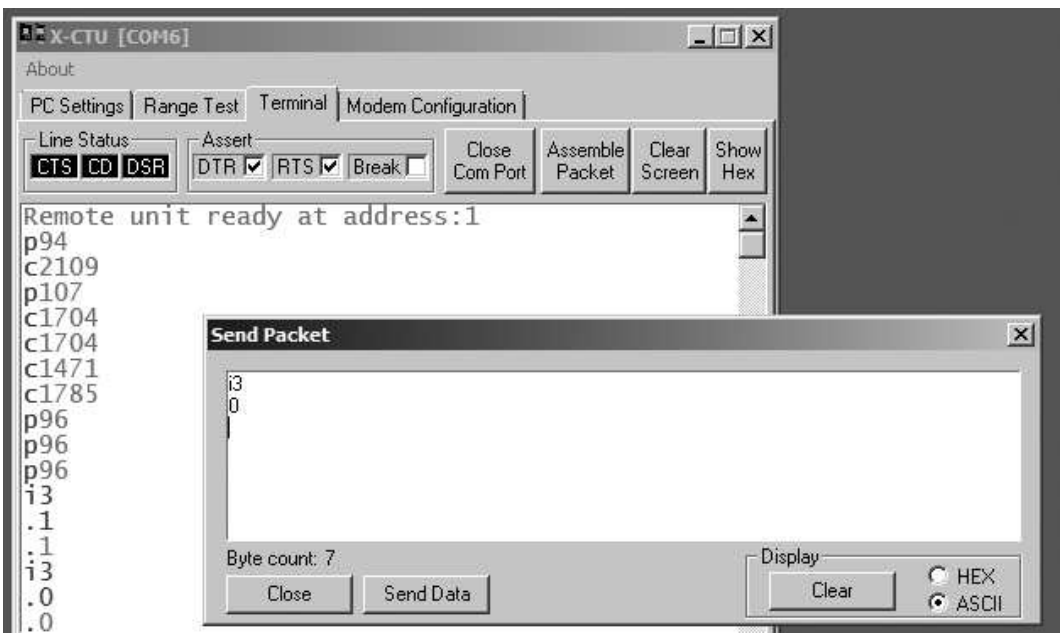


Figure 5-14 End-device responses to requests.

Looking at the end-device's code, data communications with the XBee is now through the XBee_Object. This is an object I wrote for easing some data communication and configuration issues. It uses FullDuplexSerial but greatly extends it.

Tip: The "XBee_Object" can be downloaded from Parallax's Object Exchange (http://obex.parallax.com). If you have previously downloaded it, be sure it is version 2 or higher. It is also included in the book's distributed files.

XB.AT_Init initialized the XBee to AT mode, allowing for short guard times (using ATGT), so instead of six seconds to modify a configuration, it can be done quickly in code. XB.AT_ConfigVal allows passing an AT command and a value to set configurations, such as the DL and MY addresses. The underlying code switches the XBee to command mode, sends data, and exits using the short guard times.

```
'' Enable XBee for fast configuration changes
XB.AT_Init

'' Set MY and DL (destination) address.
XB.AT_ConfigVal(string("ATMY"), MY_Addr)
XB.AT_ConfigVal(string("ATDL"), DL_Addr)
```

In the ProcessData method, XB.rx is used to tell the Propeller to wait for one character or byte of data. It then tests this character to determine what set of actions to take:

```
dataIn = XB.rx
Case dataIn
   "p":                                    ' p = PING distance
      range := Ping.Millimeters(PING_Pin)   ' Read PING in mm
      XB.dec(range)                 ' Send range as ASCII decimal value
      XB.cr                         ' End decimal string with CR

   "c":                             ' c = Compass
      theta := HM55B.theta          ' Read Compass
      XB.dec(theta)                 ' Send theta of bearing as decimal
      XB.cr                         ' End with carriage return

   "i":                                ' i = I/O control
       IO := XB.rxDecTime(timeout)     ' Accept IO number w/timeout
       state := XB.rxDecTime(timeout)  ' Accept state (1/0) w/timeout
       if state <> -1
         dira[IO]~~                 ' Set direction of pin
         outa[IO] := state          ' Set state of pin
         XB.dec(outa[IO])           ' Send state back for verification
         XB.cr                      ' End decimal string with CR
```

If p, send back the decimal value of the range finder.
If c, send back the decimal value of the compass bearing.