

# Getting Started with XBee RF Modules

---

A Tutorial for BASIC Stamp and Propeller Microcontrollers

Version 1.0

by Martin Hebel  
and George Bricker  
with Daniel Harris

PARALLAX 

## WARRANTY

Parallax Inc. warrants its products against defects in materials and workmanship for a period of 90 days from receipt of product. If you discover a defect, Parallax Inc. will, at its option, repair or replace the merchandise, or refund the purchase price. Before returning the product to Parallax, call for a Return Merchandise Authorization (RMA) number. Write the RMA number on the outside of the box used to return the merchandise to Parallax. Please enclose the following along with the returned merchandise: your name, telephone number, shipping address, and a description of the problem. Parallax will return your product or its replacement using the same shipping method used to ship the product to Parallax.

### 14-DAY MONEY BACK GUARANTEE

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. Parallax Inc. will refund the purchase price of the product, excluding shipping/handling costs. This guarantee is void if the product has been altered or damaged. See the Warranty section above for instructions on returning a product to Parallax.

## COPYRIGHTS AND TRADEMARKS

This documentation is copyright © 2010 by Parallax Inc. By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used exclusively with Parallax microcontrollers. Any other uses are not permitted and may represent a violation of Parallax copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by Parallax Inc. Duplication for educational use is permitted, subject to the following Conditions of Duplication: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with Parallax microcontrollers, and the user may recover from the student only the cost of duplication.

This text is available in printed format from Parallax Inc. Because we print the text in volume, the consumer price is often less than typical retail duplication charges.

XBee and XBee Pro are registered trademarks of Digi International Inc. Bluetooth is a registered trademark of Bluetooth Inc. Sig. Propeller and Spin are trademarks of Parallax Inc. BASIC Stamp and Boe-Bot are registered trademarks of Parallax, Inc. If you decide to use any trademarks of Parallax Inc. on your web page or in printed material, you must state that (trademark) is a (registered) trademark of (owner) upon the first appearance of the trademark name in each printed document or web page. Other brand and product names herein are trademarks or registered trademarks of their respective holders.

**ISBN 9781928982562**

### 1.0.1-11.02.07-SCP

## DISCLAIMER OF LIABILITY

Parallax Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax Inc. is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your BASIC Stamp, Propeller, and/or XBee application, no matter how life-threatening it may be.

## PUBLIC FORUMS

We maintain active web-based discussion forums for people interested in Parallax products, at <http://forums.parallax.com>:

- [Propeller chip](#) – This list is specifically for our customers using multicore Propeller chips and products.
- [BASIC Stamp](#) – This list is widely utilized by engineers, hobbyists and students who share their projects and ask questions.
- [Sensors](#) – Discussion relating to Parallax sensors or for interfacing sensors to Parallax processors.
- [Stamps in Class](#)® – Created for educators and students using the Stamps in Class series of tutorials and hardware.
- [Parallax Educators](#) – A private forum exclusively for educators and tutorial developers; resources such as solved test banks, example code, and classroom materials are posted here. Email [education@parallax.com](mailto:education@parallax.com) for enrollment instructions.
- [Robotics](#) – Designed for Parallax robots, this forum is intended to be an open dialogue for robotics enthusiasts.
- [PropScope](#) – Discussion and support for using the Propeller-based PropScope oscilloscope and accessories.
- [Wireless](#) – For customers using XBee, RFID, GSM/GPRS, and other wireless tools with their projects.

## ERRATA

While great effort is made to assure the accuracy of our texts, errors may still exist. If you find an error, please let us know by sending an email to [editor@parallax.com](mailto:editor@parallax.com). We continually strive to improve all of our educational materials and documentation, and frequently revise our texts. Occasionally, an errata sheet with a list of known errors and corrections for a given text will be posted to our web site, [www.parallax.com](http://www.parallax.com). Please check the individual product page's free downloads for an errata file.

---

# Table of Contents

INTRODUCTION .....	4
1 : NETWORKING & XBEE OVERVIEW .....	5
Multi-Node Network Issues .....	5
XBee Overview.....	8
RF Transmission Range Considerations .....	14
Other XBee Features .....	16
Summary .....	17
2 : PARALLAX XBEE ADAPTER BOARDS .....	18
BASIC Stamp and other 5 V Controllers .....	18
Propeller Chip and other 3.3 V Microcontrollers .....	21
Direct PC Interfacing with USB .....	23
Summary .....	24
3 : XBEE TESTING & CONFIGURATION.....	25
XBee Testing.....	25
XBee Configuration .....	30
Overview of Common Configuration Settings.....	34
Common Configuration Settings .....	40
Summary .....	40
4 : PROGRAMMING FOR DATA AND XBEE CONFIGURATIONS .....	42
BASIC Stamp .....	43
Propeller Chip.....	57
Summary .....	70
5 : NETWORK TOPOLOGIES & CONTROL STRATEGIES .....	71
Network Topology Examples .....	71
BASIC Stamp Examples .....	74
Propeller Examples .....	88
Summary .....	100
6 : SLEEP, RESET & API MODE .....	101
Interfacing 5 V Input to the XBee Module .....	101
Using Sleep Modes .....	102
Using Reset.....	103
API Programming and Uses.....	104
Summary .....	127
7 : MONITORING AND CONTROL PROJECTS.....	128
About StampPlot Pro.....	128
StampPlot Configuration and Data Exchange Overview .....	129
BASIC Stamp Project: Monitoring and Controlling a Remote Unit Project .....	133
BASIC Stamp Project: Wireless Joystick Control and Monitoring of a Boe-Bot .....	135
Propeller Project: A Three-Node Tilt-Controlled Robot with Graphical Display .....	143
Conclusion.....	152
8 : XBEE ZB MODULE QUICK START.....	153
Mesh Network Topology .....	153
XBee Series 2 Initial Configuration Instructions.....	153
Testing Your XBee ZB Network .....	161
REVISION HISTORY .....	163

---

## Introduction

As researchers and teachers at Southern Illinois University Carbondale (SIUC) in Electronics Systems Technologies (EST), we have used the XBee 802.15.4 modules extensively in the classroom and in research endeavors. Through national and international collaborations, we have developed systems using the 802.15.4 XBee in real-time biological monitoring, citrus vibration monitoring during harvest, and other monitoring and control systems—such as robotics! With the XBee ZB modules we have collaborated with the USDA on field irrigation monitoring and control. We hope you find them as useful and fun as we have.

### **This tutorial is designed primarily for XBee 802.15.4 modules. What about XBee ZB?**

All of our projects in chapters 1 through 7 have been designed using XBee 802.15.4 modules, sometimes referred to as Series 1. XBee ZB modules, sometimes referred to as Series 2, are not a drop-in replacement for the 802.15.4 modules due to code incompatibility and X-CTU configuration requirements. So you will need to make some significant changes in the XBee setup and microcontroller code to perform these activities with XBee ZB modules.

For example, the following pertains to XBee ZB modules:

- An XBee must have the AT Coordinator firmware downloaded to it using Digi's X-CTU software and the XBee USB Adapter Board (Part number 32400).
- XBee modules used as Endpoints need AT Endpoint firmware downloaded using X-CTU software and the XBee USB Adapter.
- If a Router is needed, the XBee needs to have AT Router firmware downloaded to it using X-CTU software and the USB Adapter.
- Unlike the 802.15.4 XBee, the addresses of XBee ZB modules are NOT configurable by the user. They are assigned by the Coordinator, so it makes duplex communication more troublesome.
- For Remote Endpoints to send data to the Base Coordinator, simply use a destination address (DL) of 0.
- The best suggestion for Coordinator to Remote Endpoint communications is to use the broadcast address (DL) of \$FFFF from the Coordinator to send data to ALL Endpoints.
- If you decide to use API firmware versions, few to none of the API code examples in Chapter 6 or the XBee\_Object.spin operations will be functional.

For these reasons we have chosen to use XBee 802.15.4 modules in the majority of this tutorial. These modules support networking and have proven capable for the vast majority of our projects. If mesh networking with routing is needed, some of the principles in this tutorial may be used, but the code will need to be modified to meet addressing needs. However, you can learn great information from this tutorial to use with XBee ZB modules, and Chapter 8 does include a Getting Started guide for basic configuration of XBee ZB modules.

<b>XBee Modules Available from Parallax Inc.</b>	
<b>802.15.4 Modules</b>	<b>ZB Modules</b>
32404 XB24-ACI-001 XBee 2mW Chip Ant	
32405 XB24-AWI-001 XBee 2mW Wire Ant	32408 XBP24-Z7WIT-004XBEE PRO WIRANT
32406 XBP24-ACI-001 XBee 60mW Chip A	32409 XB24-Z7WIT-004 XBEE 1MW WIRANT
32407 XBP24-AWI-001 XBee 60mW Wire A	

# 1: Networking & XBee Overview

The XBee RF Modem from Digi International is a wireless transceiver. The XBee uses a fully implemented protocol for data communications that provides features needed for robust network communications in a wireless sensor network (WSN). Features such as addressing, acknowledgements and retries help ensure safe delivery of data to the intended node. The XBee also has additional features beyond data communications for use in monitoring and control of remote devices. This chapter will discuss some essential elements of network communications and provide an overview of the XBee module.

## Multi-Node Network Issues

Many wireless modules for microcontrollers simply send data and receive data on the provided frequency. It is up the end user and his (or her) application code to deal with issues such as media access rules, data delivery verification, error checking, and, in multi-node networks, which node will accept and use the data. Devices using a networking protocol can ease the work of the programmer by handling these tasks.

In discussing the problems associated with networking, consider humans and the problems and resolutions in ensuring the proper flow of communications. Certain rules of communication, or protocols, are used in ensuring our message flows properly from sender to receiver across the medium, such as the air gap when talking in person.

### Media Access — Can I talk now?

In conversations between two or more people, it is important that two people don't talk at once or the message from either or both may not get through as their words collide and create confusion to the listener. In networking terms this is media access—how the parties gain access to get their words out to be accepted and understood. People in an informal group tend to wait for an opening before beginning to speak. What happens when two see an opening and both begin to speak at once? They note the problem, both back off and after a little negotiating one begins to speak, allowing that person use of the medium to get his thoughts out.



Figure 1-1: When Words Collide

In more formal settings, other means may be used to control who speaks when. In the classroom, the teacher normally has exclusive access to the medium. By looking around the classroom, the teacher informally polls the students to see if anyone requires access. Should a student raise a hand, the teacher will allow that student to speak. In the classic novel *Lord of the Flies*, to ensure each castaway

# 1: XBee Networking & Overview

---

boy had a chance to speak in council, a conch shell was passed around. Whichever child held the conch had exclusive speaking rights. This ensured each had a chance to speak without others talking over him.

In networking, different network protocols handle access to media in different ways, but are similar to human communications in many ways. Ethernet (IEEE 802.3) over a shared wire uses “CSMA/CD” much like informal human conversations. All nodes (network devices) have an equal right to send data on the network (Multiple Access—MA). First, a node listens to see if the medium (CAT 5 cable) is in use. If it is, it will wait for an opening. Once it sees an opening, it attempts to transmit its data (Carrier Sense—CS). While sending data, the node also monitors what is on the medium and if they sense another node transmitted at the same time, the collision of data is detected and they both stop transmitting, wait a unique amount of time, and try again (Collision Detection—CD).

The WiFi protocol (IEEE 802.11) uses CSMA/CA for access; since these nodes cannot transmit and listen at the same time, they use extra measures to ensure collisions do not occur (Collision Avoidance—CA). Collision avoidance also helps to alleviate the situation where two wireless transmitters are within range of each other, but a third wireless transmitter is outside the range of the first transmitter. The transmitter in the middle hears transmissions from both neighbors, but the transmitters on the ends do not hear each other. They are "hidden" from each other. This is called, in networking, the “Hidden Node Problem.”

The Token Ring protocol (IEEE 802.5) ensures access and prevents collisions by passing a token from node to node in a ring fashion, allowing each to have exclusive access to the medium in turn much like passing the conch shell. The USB protocol uses polling—the USB host (normally the PC) polls each USB device to see if it requires access to communicate, and then provides it a time-slot to communicate with the host.

## Addressing — Are you talking to me?

When talking in groups, informally or in more formal situations such as the classroom, sometimes our words are meant for a specific person and sometimes they are intended for the entire group. When speaking to an entire group, our mannerisms and mode of speech imply the words are meant for the group. To speak to someone individually, we address that person directly by name to identify the intended recipient. Others may hear the message, but understand they are not meant to respond nor perhaps use the information.



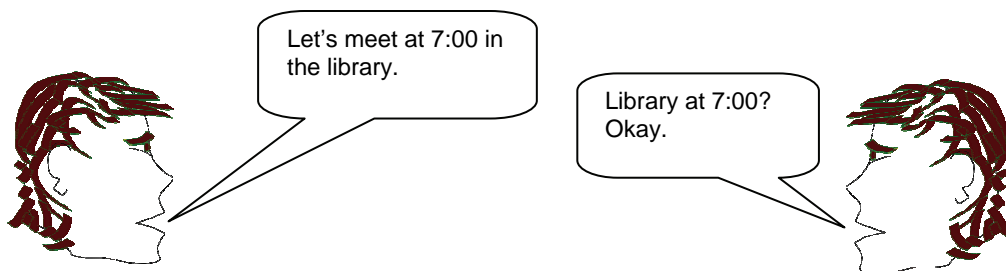
Figure 1-2: Was That Message Meant for Me?

Networking provides the same features—sometimes a message sent is meant for all to accept and sometimes a message is meant for only a specific receiver. While we use names in identification, in networks address numbers are assigned to various devices. Addresses allow the data sent to be used by only one device (point to point) or by an entire group of devices (broadcast message or point to multi-point). In many cases all nodes receive the “data sent” message, but if the data does not contain their address they are programmed to disregard the message—only the device with the correct destination address uses the message. Networks also have specialized addresses to indicate that all the nodes should accept and use the message—the broadcast address.

The source address of the received message can be important too. You are in a group, hear your name and a question, but in order to respond properly you need to know who sent the message. In networking, part of the data sent includes the originator or source address of who sent the message in order to acknowledge and reply accordingly.

### **Error Checking & Acknowledgments — Did you understand me?**

It’s usually important that the words we speak are acknowledged and understood. Whether in informal setting where we are conveying some events in our lives or in more formal settings where instructions and orders are being conveyed, it can be important that we know the recipient received and understood the message. Some form of acknowledgment lets us know through verbal or non-verbal feedback they are receiving the message we convey. Some situations require direct acknowledgement, such as military orders. Other situations where less important information is being sent may not require direct acknowledgement, such as excusing yourself from a group. In some circumstance it can be critically important that the message is not only received, but correctly understood. In the military, orders are often repeated back to ensure the party received the message without error. Many times, we respond in informal situations to ensure we understood the message—“Let’s meet at 7:00 in the library.” “Library at 7:00? Okay.”



**Figure 1-3: Did You Understand Me? Yes!**

When passing data in a network, it is important to verify the message was received and the data contained no errors. A simple form of error checking is to add up all the bytes values to be sent and append that value to the data sent (checksum value). On reception, the same math is performed on the received data and that value is checked against the received checksum value. If there is a mismatch, the data contains errors and is not acknowledged. If the message is not received or contains errors, the receiving node does not acknowledge the reception and the sending node retries the transmission.

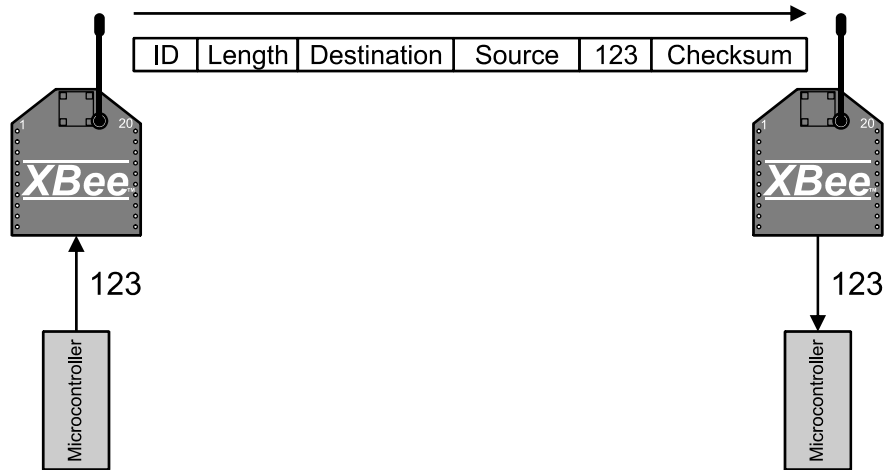
### **Encapsulation — Pack it up and ship it!**

Just as when we send a letter, we don’t simply write the letter and drop it in the mailbox hoping it makes it to the recipient. The letter needs to be placed in an envelope and addressed to ensure that it is received and read by the correct person. The return address allows the recipient to know who it was from (and for return by the post office if required). Data on a network transfers messages in a similar manner. The message (the data we are sending) is packaged with other data the protocol requires as

# 1: XBee Networking & Overview

---

shown in Figure 1-4. This added data typically includes: source address, destination address, error checking values, and other pertinent information needed by the protocol. Our message is encompassed with the other data to help ensure proper delivery to intended node.



**Figure 1-4: Packaging Data for Delivery**

## Application—Got the message, now what?

In networking terms, the packaging of the data, its point-to-point transmission, error checking and acknowledgement is handled by the Data Link protocol. Examples of Data Link protocols include Ethernet, Token Ring and WiFi protocols. These protocols ensure the intended node receives the data error-free. It is up to the application to define what the message is and what to do with it once received. What message is sent and how it will be used is the task of the application engineer—those of us designing systems with nodes that send data between points. How will the software and hardware will interact with one another across the network?

Through this tutorial we look at how the XBee performs the link for our data, how the XBee can be configured as needed by our applications, and how to code our controllers to send and receive data for specific applications.

## XBee Overview

### XBee Benefits

In modern wireless protocols, such as WiFi (IEEE 802.11) and Bluetooth (IEEE 802.15.1), the protocol helps ensure data arrives at the correct destination without errors as discussed. The protocol greatly reduces the work of the programmer in ensuring data delivery. Some key features of protocols in ensuring data delivery and integrity include:

- Media Access: A means to ensure two network nodes do not transmit at the same time causing data collisions and errors in transmission.
- Addressing: A means to ensure only the intended node uses the received data, allowing data to be sent from one point to another point. Or, point to multi-point by sending a broadcast meant for all nodes on the network.
- Error Detection: A means to verify data received at the node correctly.
- Acknowledgements & Retries: A means to inform the transmitting node that the data was delivered successfully. Lacking this, several retries may be performed in an effort to deliver the data.



The XBee utilizes the IEEE 802.15.4 protocol which implements all of the above features. This protocol is known as a Low-Rate, Wireless Personal Area Network (LR-WPAN). It provides up to 250 kbps of data throughput between nodes on a CSMA/CA network. While not intended for large volumes of data, such as image files, it provides a means of moving data quickly between nodes for use in monitoring and control systems commonly referred to as a Wireless Sensor Network (WSN).

In comparison to Bluetooth (IEEE 802.15.1), the LR-WPAN is designed as a much simpler protocol with lower data transfer rates (250 kbps compared to 1 Mbps). Bluetooth was designed as a replacement for peripheral cables and is used in communications between handheld devices, such as phones, requiring access security and high rates of data transfer.

The XBee, using the IEEE 802.15.4 protocol, incorporates the following for communications and control on the WSN (wireless sensor network).

- Clear Channel Assessment (CCA): Before transmitting, an XBee node listens to see if the selected frequency channel is busy.
- Addressing: The XBee has two addressing options: a fixed 64-bit serial number (MAC address) which cannot be changed, and a 16-bit assignable address (which we will use) that allows over 64,000 addresses on a network.
- Error Checking and Acknowledgements: The XBee uses a checksum to help ensure received data contains no errors. Acknowledgements are sent to the transmitting node to indicate proper reception. Up to 3 retries are performed by default if acknowledgements are not received.

## Communication Modes

The XBee supports both an AT and an API (Application Programming Interface) mode for sending and receiving data at your controller. Both have their advantages.

### AT Mode

In AT Mode, also called Transparent Mode, just the message data itself is sent to the module and received by the controller. The protocol link between the two is transparent to the end user and it appears to be a nearly direct serial link between the nodes as illustrated previously in Figure 1-4. This mode allows simple transmission and reception of serial data. AT Commands are used to configure the XBee, such as to send the data to a node with an address of 5: **ATDL 5**.

AT Commands and examples will be explored in more depth later, but the process requires placing the XBee into Command Mode, sending AT codes for configuration, and exiting the Command Mode.

Note that even though the transmission and reception is the raw data, the message itself is passed between nodes encapsulated with needed information such as addressing and error checking bytes.

### API Mode

In API Mode, the programmer packages the data with needed information, such as destination address, type of packet, and checksum value. Also, the receiving node accepts the data with information such as source address, type of packet, signal strength, and checksum value. The advantages are the user can build a packet that includes important data, such as destination address, and that the receiving node can pull from the packet information such as source address of the data. While more programming intensive, API Mode allows the user greater flexibility and increased reliability in some cases.

# 1: XBee Networking & Overview

Note that both sides do not need to be in the same mode. Data may be sent in API Mode and received in AT Mode or vice-versa. The mode defines the communications link between the PC or controller and the XBee modem, and not between XBee modules. Data between XBee modules is always sent using the IEEE 802.15.4 LR-WPAN protocol. Examples using AT and API Modes will be explored in later chapters.

## XBee Module Styles

The XBee module comes in several versions but all have similar pinouts as shown in Figure 1-5. Differences between XBee versions include the power output, antenna style, operating frequency and networking abilities. The XBee is a 20-pin DIP module with pin spacing of 2 mm (0.079 in) as opposed to typical pin spacing of 2.54 mm (0.1 in). The XBee is available in two major versions and variants of those versions. This tutorial uses the XBee and XBee-Pro 802.15.4 versions exclusively through Chapter 7.

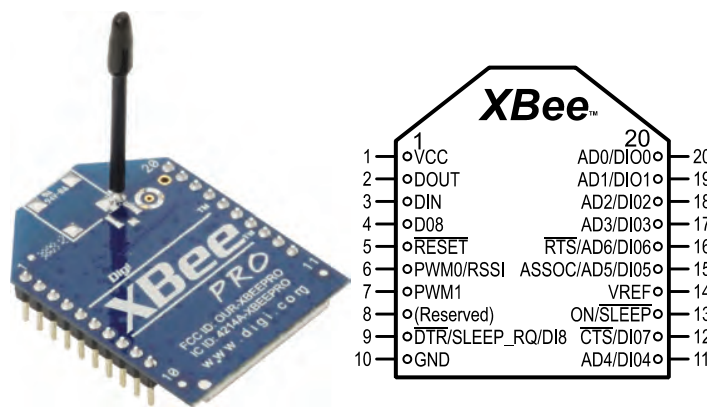


Figure 1-5: XBee-Pro Module and Pinouts

Don't be alarmed by the number of pins—typically very few pins are used in our examples. Figure 1-6 illustrates a typical 3.3 V controller connection to the XBee. Table 1-1 is a brief discussion of the pins and their functions on the XBee.

**! Wait! The XBee is a 3.3 V device! Please see Chapter 2 before connecting your XBee module to a 5 V power source or a 5 V controller, such as the BASIC Stamp.**

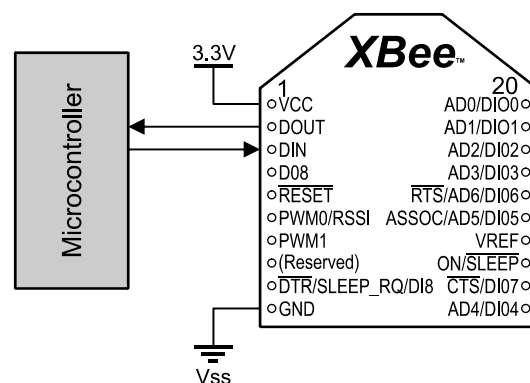


Figure 1-6: Typical Microcontroller Interfacing to the XBee

Table 1-1: XBee 802.15.4 Pin Assignments

Pin	Name	Type	Function
1	VCC	P	2.8 V to 3.4 V
2	DOUT	O	Serial data output from XBee (received data)
3	DIN	I	Serial data input to XBee (data to transmit)
4	DO8	O	Digital data output 8
5	RESET	I	Reset module (low)
6	PWM0/ RSSI	O O	Pulse Width Modulated output Received Signal Strength Indication as PWM signal
7	PWM1	O	Pulse Width Modulated output
8	(Reserved)		
9	DTR SLEEP_RQ DI8	I I I	Data Terminal Ready: handshaking for firmware updates (low) Sleep Request: A high places XBee in sleep mode when configured Digital Output 8
10	GND	G	Ground (Vss)
11	AD4 DIO4	A IO	Analog to Digital Input 4 Digital Input/Output 4
12	CTS DIO7	O IO	Clear to Send output for controller handshaking (low) Digital Input/Output 7
13	ON/SLEEP	O	Digital output, status indication: High = Awake, Low = Sleep
14	VREF	A	Analog to Digital reference voltage
15	ASSOC AD5 DIO5	O A IO	Associated indication when joining a network Analog to Digital Input 5 Digital Input/Output 5
16	RTS AD6 IO6	I A IO	Ready to Send Handshaking input (Low) Analog to Digital Input 6 Digital Input/Output 6
17-20	AD3-AD0 DIO3-DIO0	A IO	Analog to Digital Input 3 to 0 Digital Input/Output 3 to 0

Pin Type: P = Power, G = Ground, I = Input, O = Output, A = Analog Input

A short discussion of pin groups will provide better understanding of use and features of the module:

- **DOUT and DIN:** These are the pins through which serial data is received by our controller or PC (DOUT) and sent to the XBee (Din). This data may be either for transmission between XBee modules or for setting and reading configuration information of the XBee. The default data rate is 9600 baud (bps) using asynchronous serial communications.
- **RESET:** A momentary low on this pin will reset the XBee to the saved configuration settings.
- **CTS/RTS/DTR:** These are used for handshaking between the XBee and your controller or the PC. The XBee will not send data out through the DOUT line to your controller unless the RTS line is held low. This allows the controller to signal to the XBee that it is ready to receive more data. DTR is typically used by the XBee when downloading new firmware, and therefore firmware updates can only be done using XBee adapter boards such as the Parallax USB Adapter Board that implement this connection. When transmitting, the XBee can signal to the controller through the CTS line that it is ready to send more data. CTS is seldom needed because the XBee sends data out by radio much more quickly than it accepts data from the controller.
- **DIO0–DIO7/D08:** These are used as standard 3.3 V digital inputs and outputs. The XBee can be controlled to set the state of the pins. They can also be used in "line passing" so that the state of a pin on one XBee (high or low) is reflected on the corresponding pin of another XBee.

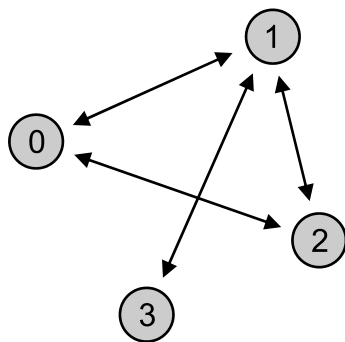
# 1: XBee Networking & Overview

- **AD0 to AD6:** These are 10-bit Analog to Digital Converter (ADC) inputs to the XBee. While we cannot directly read these values, some can also be used in "line passing" so that the amount of voltage on a pin on one XBee is reflected by the amount of voltage (PWM) on the corresponding pin of another XBee.
- **RSSI:** The XBee can report the strength of the received RF signal as PWM output on this pin. This value can also be retrieved using AT commands or as part of a packet in API Mode.
- **PWM0/1:** These pins can be set for 10-bit pulse width modulated output, which can be used directly or filtered for analog output. They can also be controlled using "line passing" by the analog input on another XBee.
- **ASSOC:** When configured, the XBee can be set to join an existing network and assigned certain parameters. In this tutorial we will manually configure the XBee in the network instead of joining networks.

## **XBee & XBee Pro, 802.15.4 Modules**

The 802.15.4 style of XBee (commonly called Series 1) allows point-to-point networking, shown in Figure 1-7, and point-to-multipoint (one node to all nodes) networking. They use the IEEE 802.15.4 data link protocol to move data directly between 2 or more devices. All nodes on the network use the same firmware version though settings on various nodes may vary.

The XBee and XBee-Pro are nearly identical in operation and function with the biggest differences being size and power. While the pinouts are the same, the casing of the XBee Pro is slightly longer. The power ratings and distances, along with other specifications, are listed in Table 1-2.



**Figure 1-7: Point-to-Point Networking**

**Table 1-2: XBee and XBee-Pro 802.15.4 Specifications**

Specification	XBee	XBee-Pro
Supply Voltage	2.8 VDC – 3.4 VDC	2.8 VDC – 3.4 VDC
RF Power	0 dBm, 1 mW	18 dBm, 63 mW
Outdoor Distance (LOS)	300 ft (90 m)	1 mile (1.6 km)
Indoor Distance	100 ft (30 m)	300 ft (90 m)
Current Draw, Receive	45 mA	50 mA
Current Draw, Transmit	50 mA	215 mA
Current Draw, Sleep	< 10 $\mu$ A	<10 $\mu$ A
RF Data Throughput	250 kbps	250 kbps
Operating Frequency, Channels	2.4 GHz, 16 Channels	2.4 GHz, 12 Channels
Receiver Sensitivity	-92 dBm	-100 dBm

Don't get too excited about the outdoor and indoor distances given in in Table 1-2. RF transmissions are affected by many factors, including line-of-sight (LOS) issues. We will discuss RF issues shortly.

## **XBee-Pro 900 and 868**

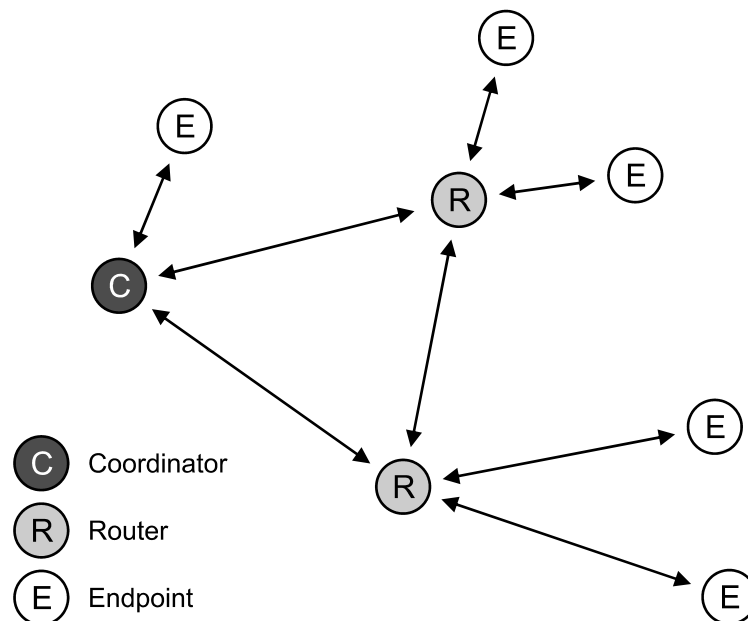
These 900 MHz (U.S.) and 868 MHz (Europe) modems operate at lower frequencies with slower data throughput (156 kbps) but much longer range (6 miles or 10 km). Please visit [www.digi.com](http://www.digi.com) for more information on these modules.

## **XBee ZB and XBee ZB-Pro Modules**

The XBee ZB modules extend the range of the network through routing. They form self-establishing, self-healing networks for moving data across the network as shown in Figure 1-8. While these modems use the IEEE-802.15.4 protocol for point-to-point communications, higher protocols are placed on top of these for network routing. There are two styles of mesh networking protocols available: ZigBee and DigiMesh. Devices can be programmed with either protocol in either AT or API versions. Depending on the function of the device, the correct firmware version must be loaded.

In a mesh network, such as ZigBee, devices have one of three different duties:

- **Coordinator**: Establishes and maintains the network by assigning addresses to joining (associated) devices and assisting with route building.
- **Routers**: move data between nodes that cannot communicate directly due to distances.
- **Endpoints**: The node that collects data and controls devices on the network and is typically connected to our controllers, sensors and other devices for network interfacing.



**Figure 1-8: Nodes in a Mesh Network**



For more information on XBee ZB modules, see Chapter 8 : XBee ZB Module Quick Start

## **RF Transmission Range Considerations**

While distance specifications such as 1 mile sound great, there are many factors that can affect the transmission distance including absorption, reflection and scattering of waves, line-of-sight issues, antenna style and frequency. The sensitivity of the receiver allows it to receive and use signals as low as -100dBm, which is 0.1 picowatts of power. The RF signal uses Direct-Sequence Spread Spectrum, spreading the signal out over the frequency spectrum greatly improving the signal-to-noise ratio (SNR).

The XBee reports the RSSI level (Receiver Signal Strength Indication) in several ways so that you may monitor the strength of your signal:

- The PWM0/RSSI output produces a pulse width providing RSSI level indication.
- In Command Mode, the unit may be polled by using the **ATDB** command.
- When receiving in API Mode, a byte in the packet contains the RSSI level.

We will explore monitoring the strength of your signal later in the tutorial.

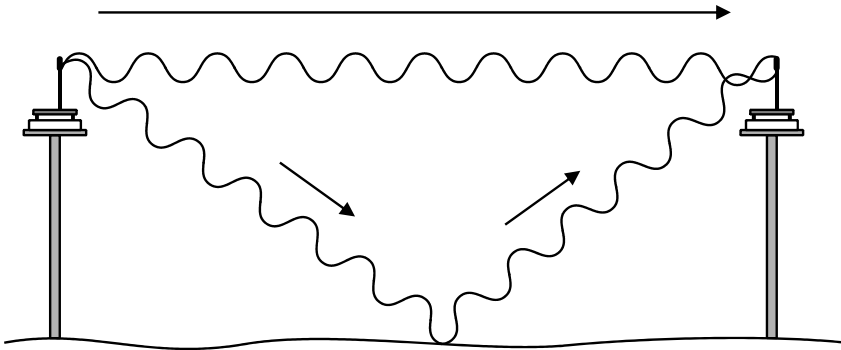
## **Absorption and Reflection**

Radio frequency waves can be absorbed and reflected causing loss of power through objects and over a distance. Metal is a great reflector of RF energy, but almost any surface can cause a reflection of waves causing loss of signal strength and sometimes interference with other waves. Absorption can also be a major factor, sometimes causing power loss, and sometimes useful—A microwave oven heats food by the water absorbing the energy and transferring the RF energy to heat. In fact, microwave ovens operate around 2.4 GHz, showing how susceptible to absorption RF energy at this frequency is. Across a distance water molecules in the air will cause absorption of our signal as will foliage of trees and other plants. While our signal may not be affected by a normal rainfall, heavy downpours can affect the signal when rain is dense enough. Indoors, walls and other objects will cause absorption and reflection as well, limiting the transmission distance to penetrating just a few walls.

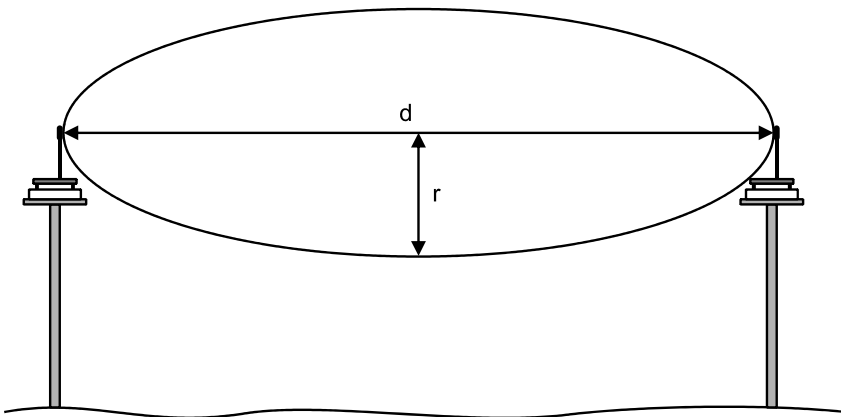
## **Line-of-Sight Issues**

Line-of-Sight (LOS) can help increase reliability of the signal. If there are no obstructions such as trees, buildings, or walls in the way absorbing or reflecting the signal, the power at the receiver will be greater. This is easy to plan for. A more difficult aspect is reflection and destructive interference from wave reflecting due to nearby objects or the ground. Even with line-of-sight, reflections from the ground or other surfaces will cause waves to reflect, join with other waves at the receiver and cause errors in the reception. The XBee's antennas are omni-directional meaning that the radio frequencies are radiated in all directions. A few go straight to the receiver and others go in a variety of directions. While the XBee-Pro may send a transmission of 60 milliwatts, only a very small amount of this power is directed at the receiver. Reflections of some of the other waves will arrive slower and out of phase at the receiver, interfering with the quality of the signal as shown in Figure 1-9.

For best transmission, the clear area needed between the transmitter and receiver can be pictured as a football shaped region called the Fresnel Zone as shown in Figure 1-10. The further the separation distance between the nodes, the wider the radius of this zone needs to be.



**Figure 1-9: Reflected waves causing problems at receiver**



**Figure 1-10: Fresnel Zone between nodes**

The radius of this zone can be calculated using the following equations depending on your units of choice:

$$r_m = 17.32 \sqrt{\frac{d_{Km}}{4f_{Ghz}}} \quad \text{or} \quad r_{ft} = 72.05 \sqrt{\frac{d_{Miles}}{4f_{Ghz}}}$$

For a distance of 300 feet or 0.058 miles (300 ft / 5280 ft), the radius of the zone is approximately 5.6 feet for good signal quality. There are higher order zones further out, but clearance in this zone will typically allow sufficient quality at the receiver.

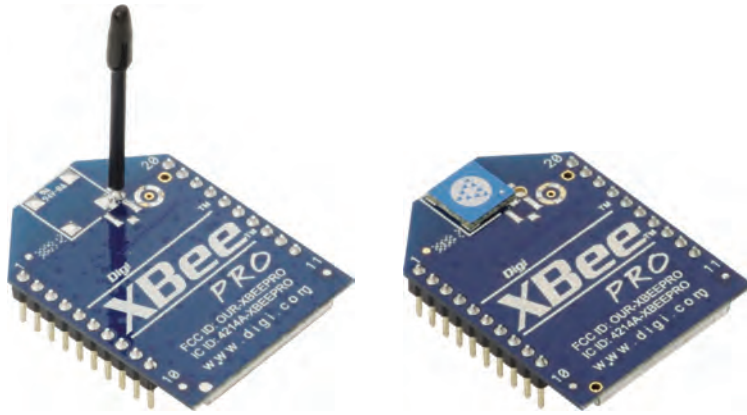
But RF can be a tricky thing. In testing, we have had two units 50 feet apart and 5 feet off the ground and experienced low signal strength. Moving one node a foot back or forward provided excellent transmission. At the initial position a reflection from a higher order zone was just right in damaging the signal at the receiver. A small shift in position cleared up the problem. Of course reflections occur not only off the ground, but any surface in the area.

### Operating Frequency

As was shown in Table 1-2, the XBee has multiple frequency channels it can operate on. This allows multiple networks in the same area to operate on unique frequencies limiting contention of nodes for the same frequency. It also allows the user to select a frequency that may have less noise on it than other channels from other sources in the 2.4 GHz spectrum such as WiFi networks, cordless phones and microwave ovens. Using AT commands, the desired operating channel may be selected. Chapter 1 will provide examples showing channel selection and monitoring.

## Antenna Style

The XBee comes with a variety of antenna styles providing differing amounts of gain. Two of the most popular versions are the chip antenna and whip (wire) antenna styles shown in Figure 1-11. The whip style has about 20% better range than the chip style. Other styles include U.FL and RPSMA connectors for attaching higher-gain antennas.



**Figure 1-11: Whip (left) and Chip (right) Antennae Modules**

## Other XBee Features

Besides simply passing data between nodes for microcontrollers, the XBee has other features that can aid in process monitoring and control. We will explore some of these in the course of the tutorial.

### Digital & Analog I/O

As discussed in the XBee pinout section, the modules have digital I/O, analog inputs, and PWM outputs that may be used in a variety of ways. Later chapters will explore these capabilities in more depth.

- Direct digital output control: Through AT commands, the I/O may be set to be digital outputs and controlled as high or low. For example, the command **ATD0 4** would set D0 to be a low output (0 V) and **ATD0 5** would be a high output (3.3 V).
- Digital and analog input for transmission: The inputs may also be set for digital input or 10-bit analog-to-digital input. Using sampling, the values of the inputs are sent as data to a receiving XBee in API Mode where the digital and ADC data is extracted.
- PWM/analog output: The 10-bit PWM value of an output may be set and optionally filtered for analog output of the unit.
- Line Passing: The digital inputs can control digital outputs on another node, and analog inputs can control PWM outputs on the other node.

### Remote Configuration

Using AT commands, the configuration of a remote module may be changed by sending the command in an API packet. This could be used to control digital or PWM outputs or change other parameters of the Remote XBee.



## **Base and Remote Nodes**

Throughout this tutorial we explore communications between XBee nodes. At times the XBee may be directly connected to a PC for communications and other times the XBee will be connected to a microcontroller. We will use the terms Base node and Remote node to identify the typical function of each.



**Base node:** The Base node will be the node from which an operator would normally monitor or control the wireless network. The Base node will be the one commonly connected to a PC for an operator to monitor, send data and control the system.

**Remote node:** The Remote node will be the node that is typically at remote location to perform the actual data collection or control of actuators in a system. In normal applications, this node would not be connected to a PC, but would send its data to the Base node and receive instructions. In many of our examples we monitor the Remote node as well as the Base node to observe interactions from both sides.

## **Summary**

The XBee is a feature-rich RF module for use on a wireless sensor network. The IEEE 802.15.4 protocol greatly reduces the work of the programming by ensuring data communications. The XBee has many other features for use in a WSN beyond its networking ability. Now that you have a better understanding about the XBee's features and uses, we will look at means of interfacing the RF modem to your microcontroller and showing examples of use.

### 2: Parallax XBee Adapter Boards

The XBee module is a 20-pin DIP package with a 2 mm (0.079 in) pitch between pins. With typical breadboard and solder board hole spacing of 2.54 mm (0.1 in) the XBee requires an adapter for use with these boards. The XBee is a 3.3 V device, as is the Propeller chip, so interfacing between the two can be done through direct connections. Interfacing with the BASIC Stamp and other 5 V controllers requires line conditioning between the controller's 5 V output and the XBee's 3.3 V input pins. However, the XBee's 3.3 V output can directly drive a controller's 5 V input logic. Additionally, a regulated 3.3 V supply is required to power the XBee. Finally, to access the XBee from the PC for communications or configuration a means of serial interfacing to the computer is needed, such as using USB and a serial interface IC. Parallax has developed a series of interface boards for the XBee for ease of physical interfacing, signal conditioning, power requirements, and computer connectivity.


#### BASIC Stamp and other 5 V Controllers

The BASIC Stamp and other 5 V controllers need an adapter that:

- Uses the 5 V supply available to provide regulated to 3.3 V for the XBee supply power.
- Conditions the 5 V logic output to 3.3 V for input to the XBee.

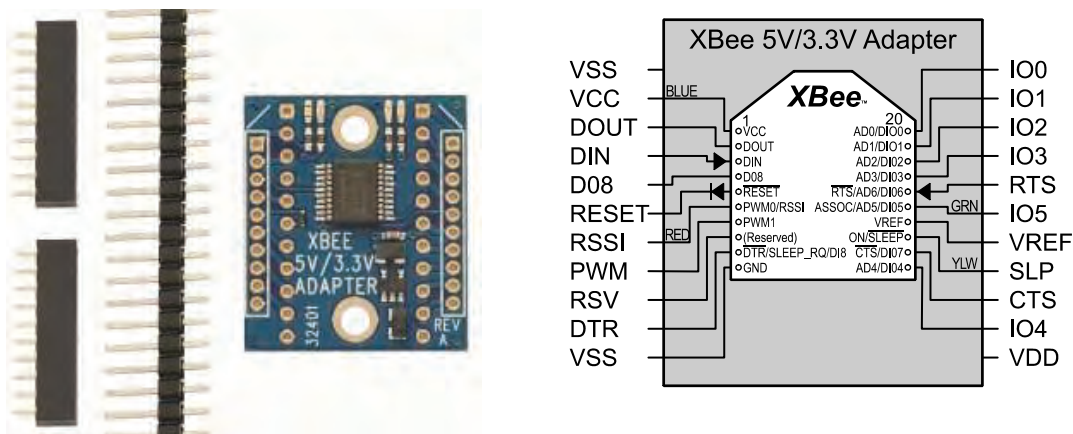
The Parallax XBee 5V/3.3V Adapter Board and the XBee SIP Adapter Board provide both these functions along with pin headers suitable to 2.54 mm (0.1 in) breadboards and solder boards. They contain a 3.3 V voltage regulator and a 5 V to 3.3 V logic buffer.

#### **XBee 5V/3.3V Adapter Board**



This section pertains to the XBee 5V/3.3V Adapter Board Rev A. The Rev B board in development will have I/O buffering and LED configurations similar to the XBee SIP Adapter discussed later in this chapter. Schematics for these boards are available from their product pages at [www.parallax.com](http://www.parallax.com).

The XBee 5V/3.3V Adapter board in Figure 2-1 has two 11-pin headers with 2 buffered inputs to the XBee: DIN (Data into XBee for transmission) and RTS (for flow-control). The XBee's Reset pin is connected through a diode to allow the controller to bring it low but not high. It also allows direct connections to all other XBee I/O pins. The board has 4 LEDs to indicate Power, Association status, RSSI (Received Signal Strength Indication) and Sleep status. The board uses a 3.3 V regulator for power to the XBee from a BASIC Stamp development board's 5 V V<sub>dd</sub> supply. Note that this board comes unassembled; the included male and female headers must be soldered to the board.



**Figure 2-1: XBee 5V/3.3V Adapter Board (#32401, unassembled) & I/O Conditioning**

### I/O Connections

Connections to the directly connected I/O pins should be performed with an understanding of use and signal conditioning needed. Please see Interfacing 5 V Input to the XBee Module on page 101.

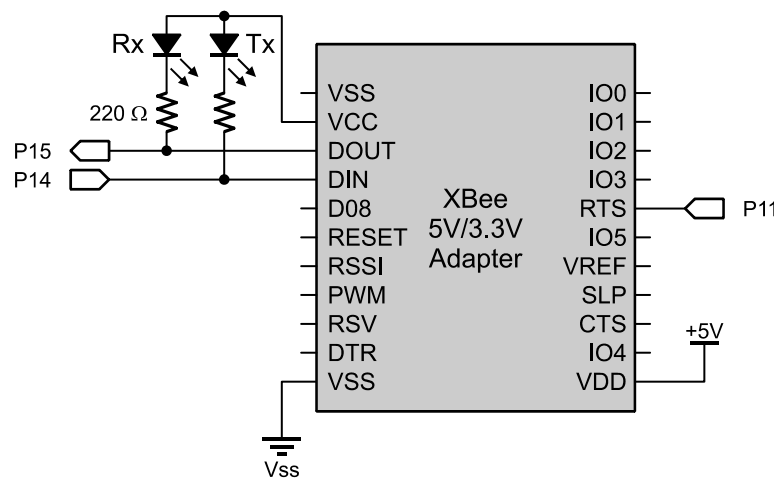
### LED Indication

The connected Tx and Rx LEDs do not show RF data transmitted and received, but data transmitted to and received from the XBee (data in and data out of XBee). In many cases this will be data sent and received via RF but not always.

### Rev A Current Draw Caution!

The FAN2500S33X regulator used on Rev A of this adapter is rated at 100 mA continuous, 300 mA peak current. VCC may be used as a 3.3 V supply output, but be sure that you budget enough current for your XBee module as well as any other device you might supply through this regulator. The regular XBee modules (802.15.4 and ZB) draw less than 55 mA, but XBee Pro modules can draw over 200 mA when in a continuous-transmit state.

Figure 2-2 shows typical connections to the 5V/3.3V adapter for communications, power and optional LEDs to indicate data transmitted to and received by the BASIC Stamp. The XBee, when configured properly, can use the RTS input to wait for a low signal from the BASIC Stamp before sending data to the controller. This allows buffering of received data until the BASIC Stamp is ready to receive.



**Figure 2-2: Typical Connections to XBee 5V/3.3V Adapter with Tx/Rx LEDs**

The LEDs on the adapter board itself indicate:

- **Yellow:** ON/Not Sleeping — This will typically be on unless you configure the XBee for one of the low-power sleep modes (see Chapter 6).
- **Blue:** Power — Indicates 3.3V power is available.
- **Green:** Associate — This LED will typically be blinking unless you are using the associate mode of joining a network (not covered in this tutorial) or by changing the pin configuration. (see Configuring I/O—D0 to D8, P0, P1, M0, M1 on page 39).
- **Red:** RSSI indicator — The XBee PWM output indicates the strength of the received RF signal. This LED will light for several seconds anytime the XBee receives RF data addressed to it. A slight dimming at low power levels may be noticeable.

## 2: Parallax XBee Adapter Boards

### XBee SIP Adapter

The XBee SIP Adapter has features similar to the 5V/3.3V Adapter Board, but conditions more I/O and provides on-board Tx and Rx LEDs. As shown in Figure 2-3, the adapter has 5 V to 3.3 V buffering for the following inputs: DIN, RTS, DTR/Sleep\_RQ. From the XBee, the following outputs are also buffered from 3.3 V to 3.3 V for protection but not level change: DOUT, CTS, ASSOC, and RSSI. Reset has an internal pull-up resistor to keep the input high; the board uses a diode to allow the controller to bring the input low. A solderable header (not shown in drawing) allows direct connections to AD/DIO0 to AD/DIO7 and a connection for 3.3 V. (The XBee SIP Adapter comes fully assembled as shown.)

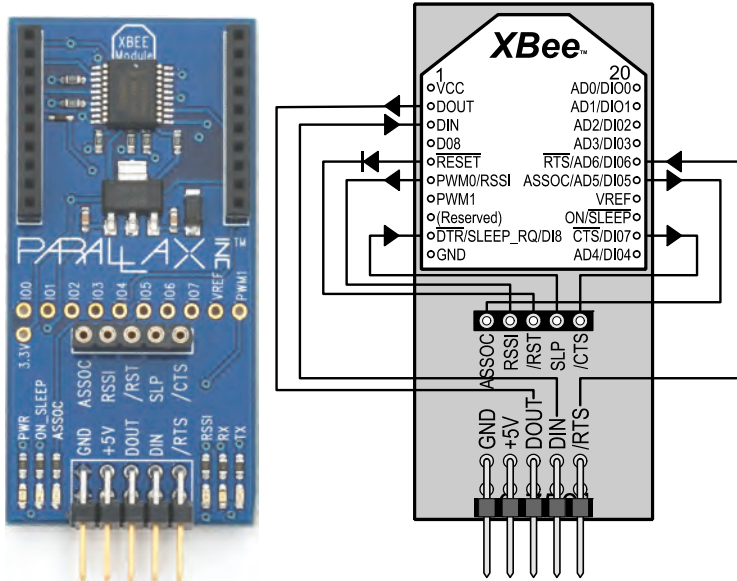


Figure 2-3: XBee SIP Adapter (#32402) & I/O Conditioning

**CAUTION!** Connections to the directly-connected I/O pins should be performed with an understanding of use and signal conditioning needed. Please see Interfacing 5 V Input to the XBee Module on page 101.

Figure 2-4 shows a typical connection the adapter to a BASIC Stamp. RTS is once again used for flow control of data from the XBee to the BASIC Stamp.

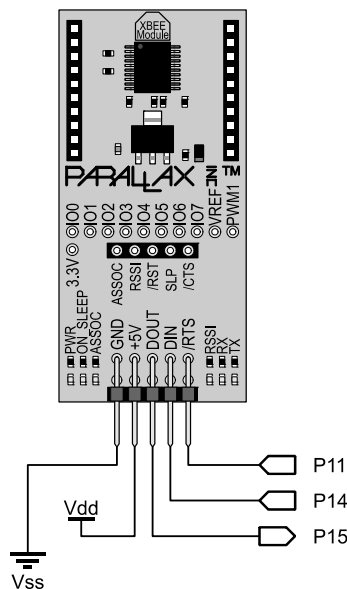


Figure 2-4: Typical Connections to XBee 5V/3.3V SIP Adapter

The LEDs on the SIP Adapter indicate:

- **PWR:** Indicates 3.3 V power available
- **ON\_SLEEP:** Indicates sleep status, so will typically be on indicating the unit is awake unless you configure the XBee for one of the low-power sleep modes.
- **ASSOC:** This LED will typically be blinking unless using the associate mode of joining a network (not covered in this tutorial) or by changing the pin configuration, covered later.
- **RSSI.**(Received Signal Strength Indicator): The XBee PWM output indicates the strength of the received RF signal. This LED will light for several seconds anytime the XBee receives RF data addressed to it. A slight dimming at low power levels may be noticeable.
- **RX:** Indicates data received by the controller from the XBee (DOUT).
- **TX:** Indicates data transmitted by the controller to the XBee (DIN).



### LED Indication

The TX and RX LEDs do not show RF data transmitted and received, but data transmitted to and received from the XBee (data in and data out of XBee). In many cases this will be data sent and received via RF, but not always.

### Propeller Chip and other 3.3 V Microcontrollers

The Propeller chip and other 3.3 V microcontrollers may use the above two boards as well:

- The Propeller board's 3.3 V Vdd supply may be supplied to the VCC pin of the XBee 5V/3.3V Adapter board, and connecting VSS.  
—OR—
- The Propeller board's 5 V supply to the XBee 5V/3.3V Adapter's VDD or the 5 V input of the XBee SIP Adapter and connecting VSS.

The XBee USB Adapter Board (next section) may also be used with USB disconnected:

- By supplying 5 V to the VDD input and connecting VSS.  
—OR—
- By supplying 3.3 V to the VCC pin of the USB Adapter Board and connecting VSS.



### Beware of Dueling Power Supplies!

Take care with all boards that two sources of power are not applied at the same time, such as connecting both 5 V AND 3.3 V supplies. Damaging current will flow between the two sources.

But since both the Propeller chip and the XBee can operate from 3.3 V, all that may be needed is the adapter for 20-pin 2 mm headers of the XBee. The XBee Adapter Board provides this solution. In some mounting situations, no adapter board may be needed.

## 2: Parallax XBee Adapter Boards

### XBee Adapter Board

This board simply converts the 2 mm pin spacing to 2.54 mm (0.1 in) for use on breadboards and solder boards. Figure 2-5 illustrates the connections for this board. Note that this board comes unassembled; the included male and female headers must be soldered to the board.

**Note:** VDD on this board is **NOT** used. VDD on this board is only included for pin compatibility with the other adapter boards. 3.3 V power is connected at VCC.

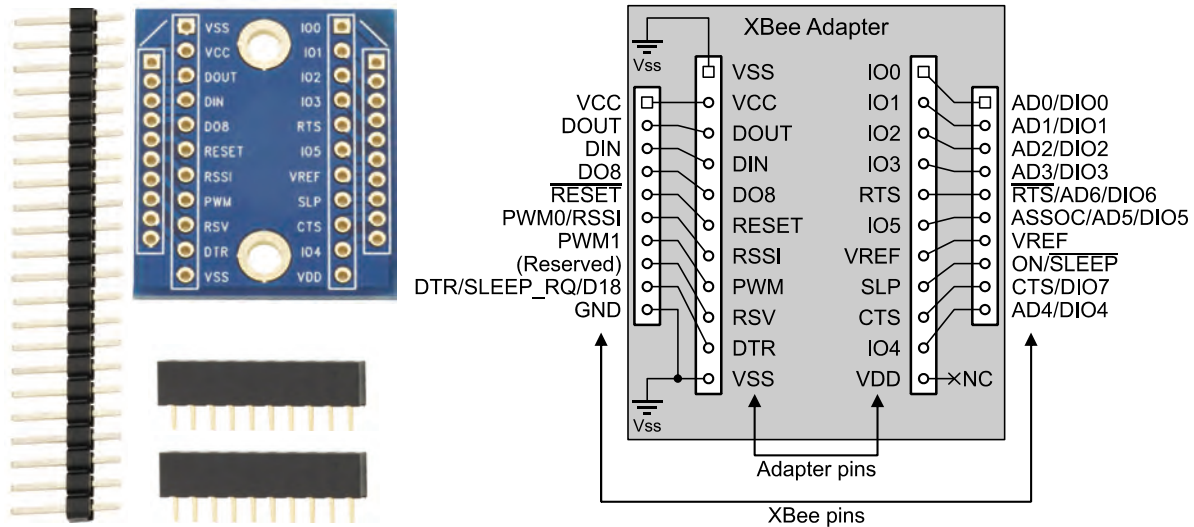


Figure 2-5: XBee Adapter Board (#32403, un-assembled) and Connections

This board has no LEDs. LEDs for communications indication, RSSI, Association and Sleep modes may be optionally connected as shown in Figure 2-6 (some chapters use these LEDs for experiments).

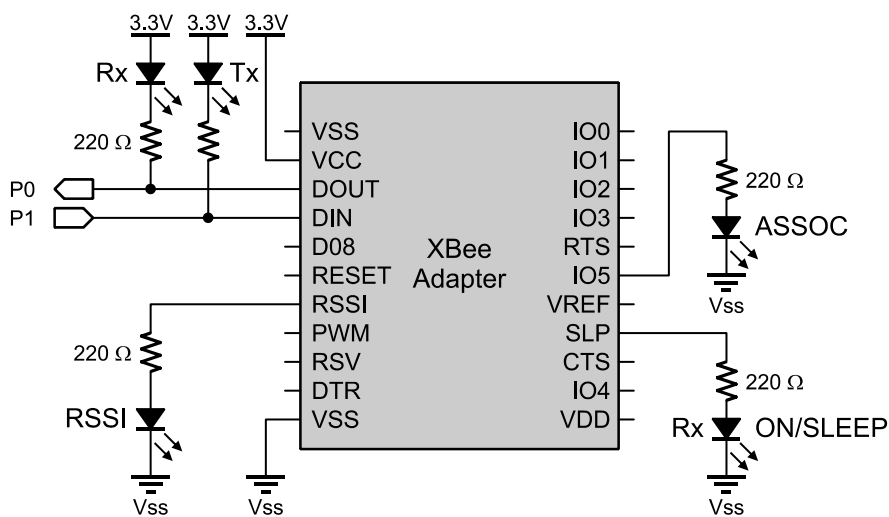


Figure 2-6: Propeller to XBee Adapter with Optional LEDs

### Direct PC Interfacing with USB

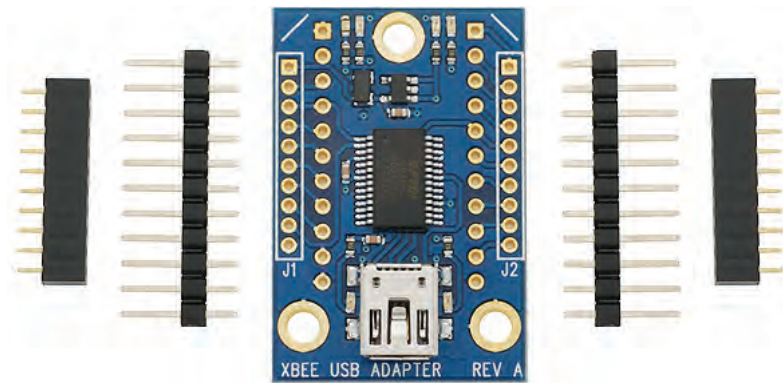
Just as a microcontroller can communicate serially to the XBee, so can a personal computer. Any terminal program, such as Parallax Serial Terminal, HyperTerminal® or the BASIC Stamp Editor's Debug Terminal can send and receive serial data. However, the free X-CTU software from Digi International may be used as a serial terminal, for configuration changes, testing RSSI levels, and to download different firmware to the XBee with the right interface.

### **XBee USB Adapter**

The XBee USB Adapter, shown in Figure 2-7, provides a serial interface using the FTDI USB serial interface chip (FT232RL) and virtual COM port drivers to emulate a serial COM port. This board provides the following features:

- Send/receive serial data between the PC and the XBee for communications and configuration.
- Self-powered from USB for 5 V and 3.3 V.
- Incorporates the DTR (Data Terminal Ready) line needed for firmware downloads.
- May be powered from a 5 V source via VDD input or 3.3 V to VCC when operating without USB.

Note that this board comes unassembled; the included female headers must be soldered to the board for mounting the XBee, and the optional male headers may be soldered on for use with a breadboard or through-hole board. Before connecting the board to your computer, download and install the required USB drivers from [www.parallax.com/usbdriers](http://www.parallax.com/usbdriers).



**Figure 2-7: XBee USB Adapter Board (#32400, un-assembled)**



**Beware of Dueling Power Supplies!** Take care with all boards that two sources of powers are not applied, such as connecting both 5V AND 3.3V supplies. Damaging current will flow between the two sources.

**3.3 Volt Supply:** When connected to USB, the 3.3 V pin may be used to supply power, but current draw should be limited to 40 mA.

For use with the PC, simply connect the appropriate USB cable between the PC and the board with an XBee installed. For use with the Propeller, connect per Figure 2-6 as with the XBee Adapter. LEDs may also be added for transmit and receive between the Propeller and XBee. LEDs on the board indicate the following:

- **Green:** ON/Not Sleeping. This will typically be on unless you configure the XBee for one of the low-power sleep modes (see Chapter 6).
- **Yellow:** Power. Indicates 3.3 V power is available.

## 2: Parallax XBee Adapter Boards

---

- **Red:** Associate. This LED will typically be blinking unless using the associate mode of joining a network (not covered in this tutorial) or by changing the pin configuration covered later (see Configuring I/O—D0 to D8, P0, P1, M0, M1 on page 39).
- **Blue:** RSSI indicator. The XBee PWM output indicated strength of received RF signal. This LED will light for several seconds anytime the XBee receives RF data addressed to it. A slight dimming at low power levels may be noticeable.
- **Red:** USB Transmit
- **Green:** USB Receive

### Using a Propeller Chip and Serial\_Pass\_Through.spin

Chapter 4 introduces programming of serial communications. One example program is Serial\_Pass\_Through.spin. This passes serial data between the PC and XBee. A non-USB adapter board and the Propeller can be used for the PC interface. A function this configuration cannot provide is updating firmware on the XBee.

### Summary

Parallax provides a variety of boards for interfacing the XBee to their 5 V and 3.3 V microcontrollers, as well a USB interface version for PC communications and configuration of the XBee. In Chapter 3 we will explore basic communications and configuration of the XBee using the PC, the BASIC Stamp, and the Propeller chip. Table 2-1 is a list of available adapter board options from Parallax.

**Table 2-1: Microcontroller Platforms and Adapter Board Options**

Processor Platform	Adapter Board Options	Board Stock Code
BASIC Stamp and other 5 V microcontrollers	XBee 5V/3.3V Adapter with 5 V supplied to VDD pin.	32401
	XBee SIP Adapter with 5 V supplied to 5 V pin.	32402
Propeller chip and other 3.3 V microcontrollers	XBee Adapter using 3.3 V supplied to VCC pin	32403
	XBee 5V/3.3V Adapter with ONE of these options: <ul style="list-style-type: none"><li>▪ 5 V supplied to VDD pin</li><li>▪ 3.3 V supplied to VCC pin</li></ul>	32401
	XBee SIP Adapter with 5 V supplied to 5 V pin.	32402
	XBee USB Adapter with ONE of these options: <ul style="list-style-type: none"><li>▪ 5 V supplied to VDD pin</li><li>▪ 3.3 V supplied to VCC Pin</li></ul>	32400
PC Interfacing	XBee USB Adapter Board, powered from USB only.	32400
	Interfacing via Propeller using Serial_Pass_Through.spin and any Propeller adapter option per Chapter 4	Any of the above



### 3: XBee Testing & Configuration

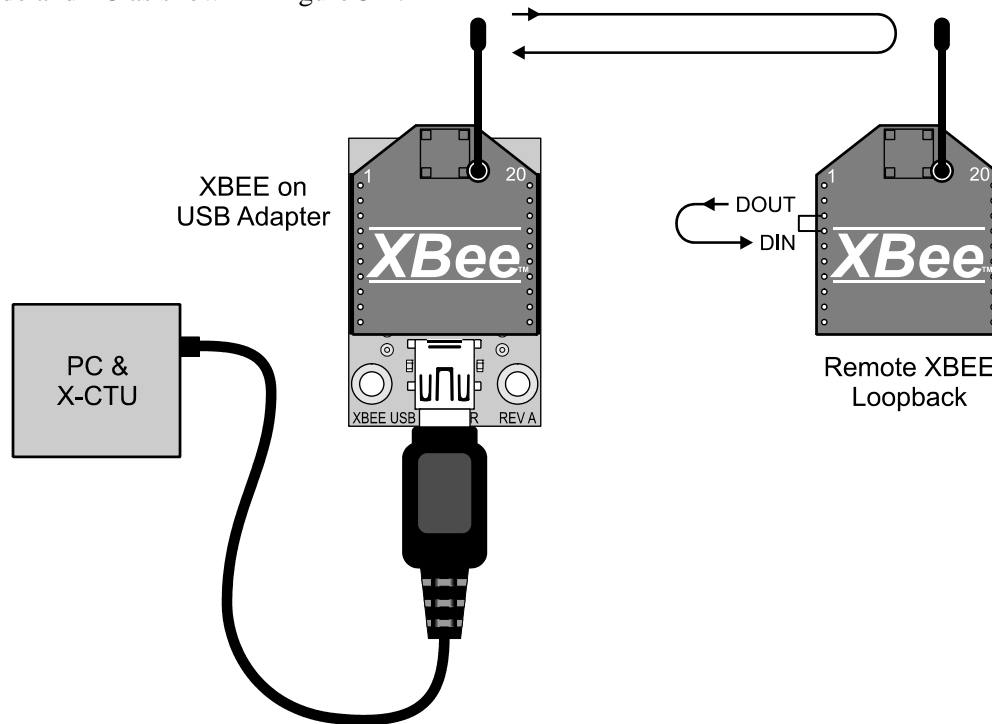
Communications between the PC and XBee can be an integral part of your system whether for XBee configuration, monitoring and control of a device, or simply for testing and feedback,. Using Digi's X-CTU software and Parallax's XBee USB Adapter board, interfacing with the XBee is simple for configuration changes, firmware downloads, testing signal strength, and communications to your remote devices. In this chapter many configuration settings of the XBee will be explored, but not all. Please refer to Digi International's XBee manual for more information.



**We are using XBee 802.15.4 Modules Only!** The term "XBee" from this point through Chapter 7 will refer to any of the 802.15.4 XBee or XBee-Pro, unless otherwise noted.

#### XBee Testing

In this section we will explore Digi International's X-CTU software for communications, signal strength monitoring and configuration of the XBee. The XBee connected to the PC using the XBee USB adapter (Base node) and a Remote node XBee with power and a loopback will be used. For the loopback, DOUT is connected to DIN which will cause any received data to be transmitted back the Base node and PC as shown in Figure 3-1.



**Figure 3-1: XBee Loopback Testing**

#### Hardware Required

- (2 or more) XBee modules
- (1) XBee USB Adapter (#32400)
- (1 or more) XBee adapter boards appropriate for your controller (BASIC Stamp, Propeller chip, etc.)
- (1) 220  $\Omega$  resistor and (1) LED, optional

## 3: XBee Testing and Configuration

---

### Other USB Options:

- Using Propeller chip for PC Communications: The Propeller may be used for USB communication to the XBee using Serial\_Pass\_Through.spin in Chapter 4. This allows all functions except firmware downloads.
- Using the Prop Plug for PC Communications: The Prop Plug (#32201) may also be used for USB communications to the XBee by connecting DOUT to Rx, DIN to Tx, and Vss to VSS with the adapter board powered normally. This allows all functions except firmware downloads.

### Software Required

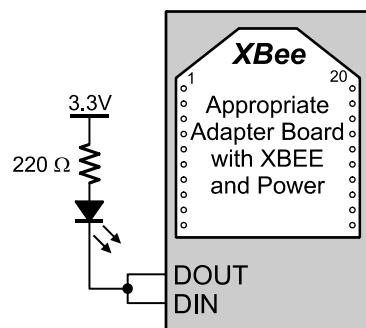
- ✓ Download the X-CTU software for Windows, available from [www.digi.com](http://www.digi.com) or through Parallax's XBee page, [www.parallax.com/go/XBee](http://www.parallax.com/go/XBee). (After installing, it may ask you if you wish to download the latest versions. This is very time consuming and may be skipped initially in most cases.)
- ✓ Download the FTDI drivers for Windows, available from [www.parallax.com/go/usbdrivers](http://www.parallax.com/go/usbdrivers).



**FTDI Drivers:** It's important to install the FTDI drivers before connecting an XBee USB Adapter Board to your computer. If you have installed the BASIC Stamp Editor or the Propeller Tool software, you probably have the Windows FTDI drivers installed already.

### Hardware Connections

- ✓ With an XBee module in place, connect the XBee USB Adapter Board to the PC using an appropriate USB cable.
- ✓ With an appropriate board with an XBee for your BASIC Stamp or Propeller, connect as shown in Figure 3-2:
  - Connect VDD or VCC (as appropriate) and VSS per Chapter 2.
  - Use a jumper wire to connect DOUT to DIN on your breadboard connection.
  - If your adapter board does not have Tx/Rx LEDs, you may optionally connect an LED from DOUT/DIN in series with a 220  $\Omega$  resistor to the VCC/3.3 V supply power.
  - If your adapter board does not have RSSI or ASSOC LEDs, these also may be added per Chapter 2.



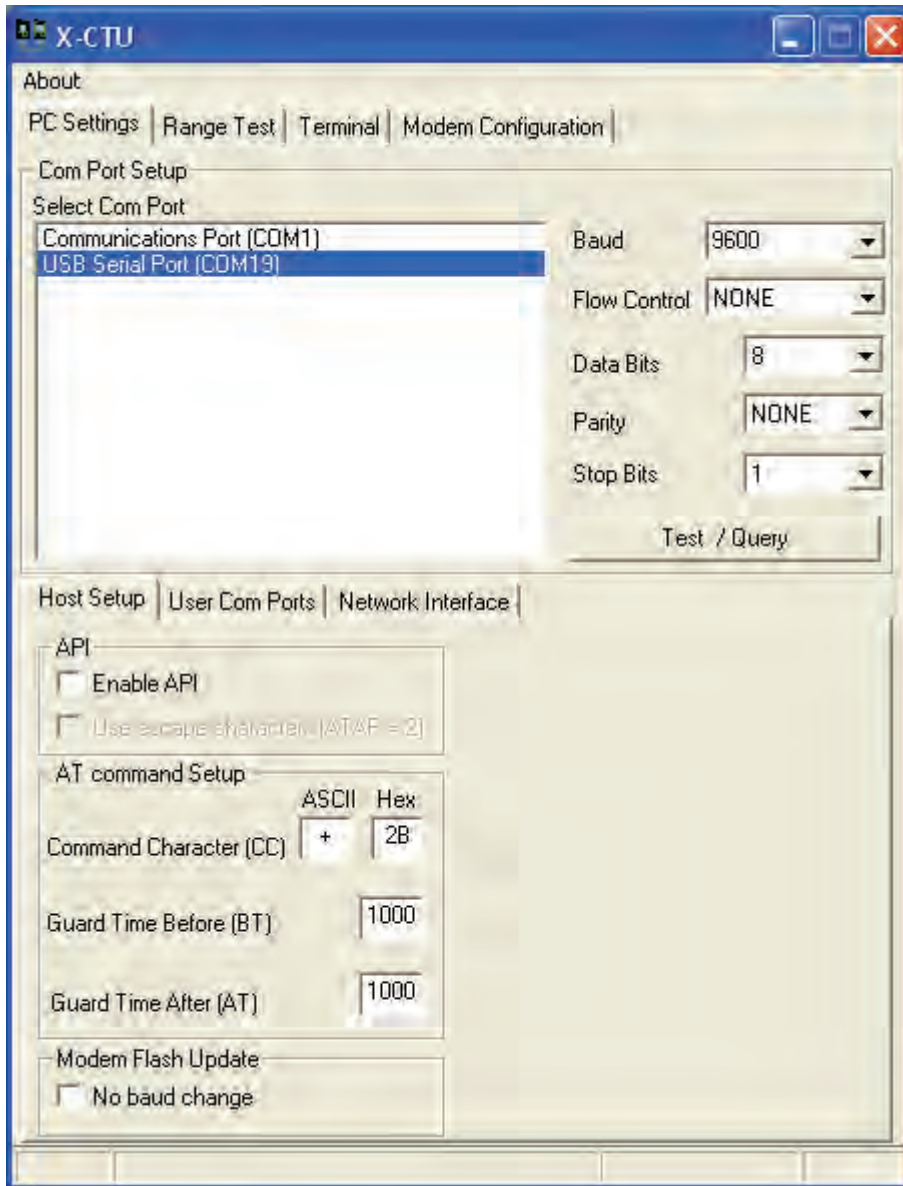
**Figure 3-2: XBee Loopback Testing**

After powering up, both units should have POWER and ON/SLEEP LEDs illuminated, and ASSOC LEDs blinking.

#### Testing Communications

In this section we will test basic communications with the X-CTU software.

- ✓ Ensure the Remote node XBee with loopback is powered.
- ✓ Open the X-CTU software.
- ✓ If your XBee USB Adapter is connected you should have one or more COM ports listed as shown in Figure 3-3.

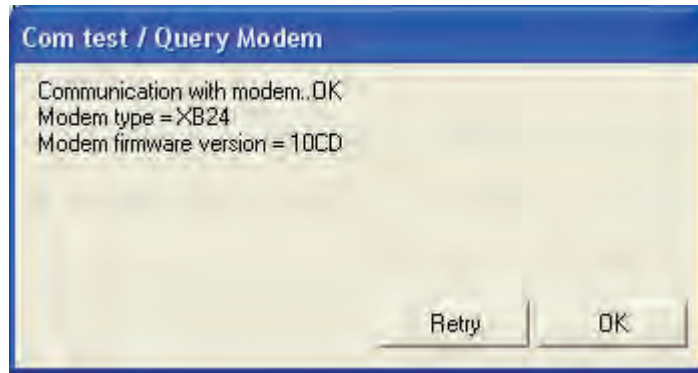


**Figure 3-3: X-CTU COM Port Selection**

- ✓ Select one and click the Test / Query button. The proper selection will show a response similar to Figure 3-4. The message box shows the type of XBee and the firmware version.

### 3: XBee Testing and Configuration

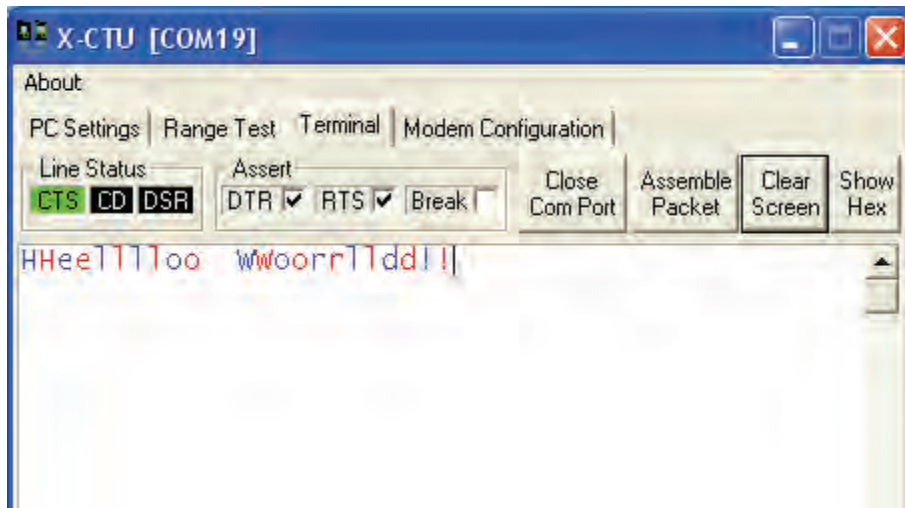
---



**Figure 3-4: X-CTU Test/Query Response**

- ✓ Click the Terminal tab on the software. This tab allows you to send and receive data using a terminal interface. We will also use this for XBee configuration changes shortly.
- ✓ In the window area, type “Hello World!”

If all went well, you should see two sets of each character—a blue followed by a red as shown in Figure 3-5. The blue character is what you sent, the red is what was received.

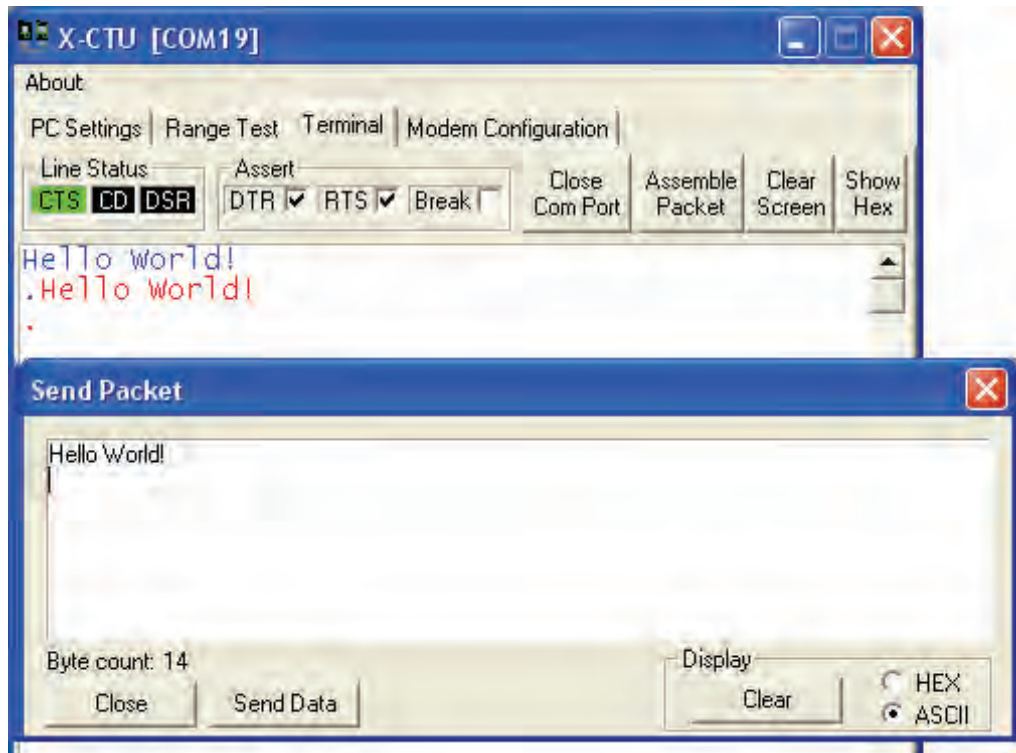


**Figure 3-5: X-CTU Terminal Test**

As each character was typed, it was transmitted, looped back, and returned for reception and display.

- ✓ Type a few more characters and monitor the LEDs on your boards. On both boards you should see the RSSI LED illuminate for several seconds with each character received. If there are Tx/Rx LEDs, these should light on the Remote board, blinking rapidly as data is sent and received.
- ✓ Click the Clear Screen button. In the terminal window, click the Assemble Packet button. This opens a window where you can type text to be sent as a single packet.
- ✓ Type “Hello World!” and press the Enter key in the text box. and then click Send Data.

In the terminal window you should see your text twice as shown in Figure 3-6. The entire text is sent quickly enough to package the data for a single transmission to the Remote XBee and returned via the loopback.



**Figure 3-6: Assembling and Sending a Text Packet**

While it may be interesting at this time to see what would happen if there were two Remote XBee modules with loopback (Do I get two responses back?), it may not be a good idea just yet. One Remote XBee transmitting will cause a repeat on the other XBee, and back again; the one transmission will be repeated over and over between XBee modules essentially causing a broadcast storm. Because all the XBee modules currently have the same address by default, they will be sending to each other as well as the Base and repeating what they get. A better time to test this is when we experiment with addressing.



**Received Decimal Points:** In some tests you will see decimal points/periods. This is typical; they corresponds to unprintable character codes such as carriage returns.

### Range Testing

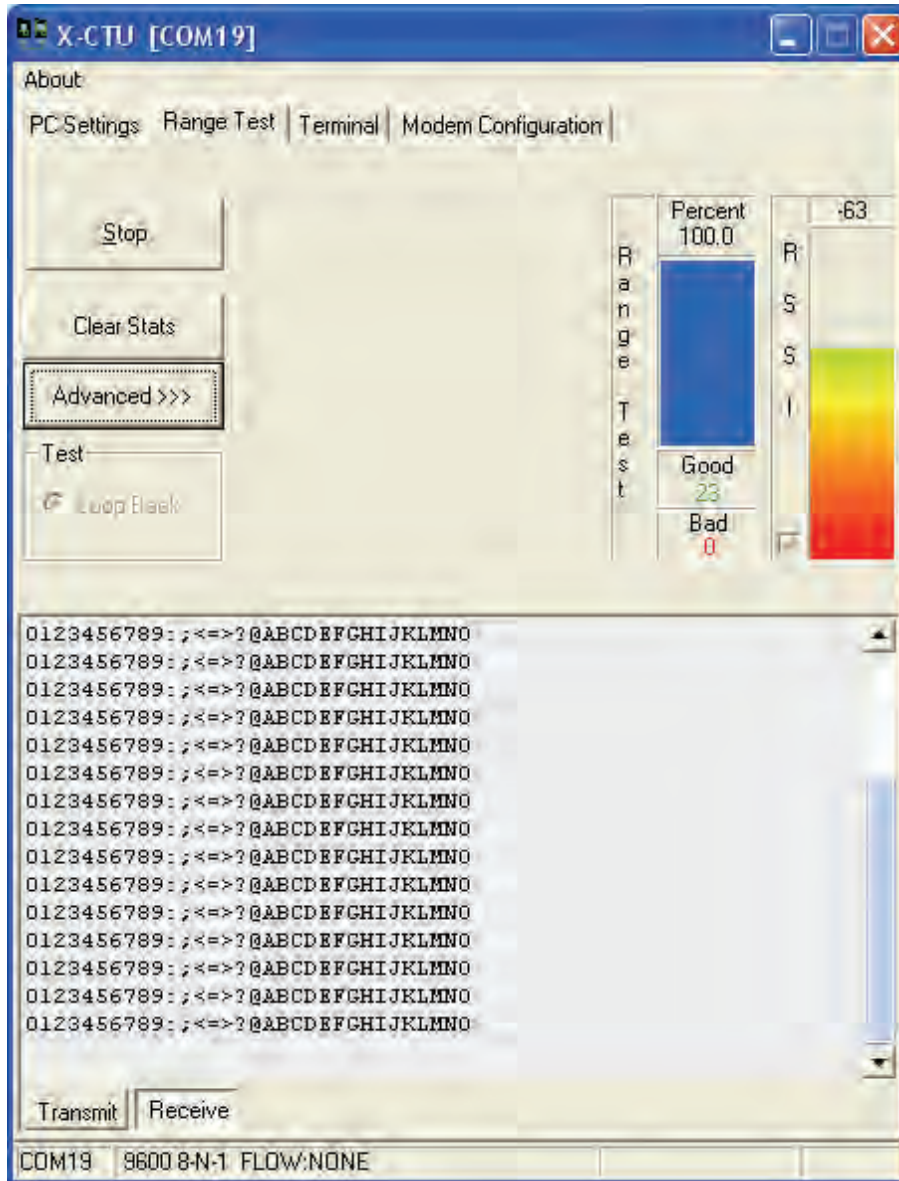
- ✓ Close the Send Packet window and click the Range Test tab on the software.
- ✓ Under the vertical RSSI lettering, check the checkbox to monitor signal strength.
- ✓ Click Start. The software will repeatedly send packets and display the signal strength and percentage of correctly returned packets as well as the packet itself as shown in Figure 3-7. Click Stop once you've seen enough.
- ✓ Note: Some users report this test does not work if they are using the Propeller Serial-Pass-Through method of communications for PC to XBee communications.

RSSI is the Received Signal Strength Indicator value. This value will be between -40 dBm (the highest recorded value) and -100 dBm (the minimum sensitivity of the XBee).

### 3: XBee Testing and Configuration



**Packet Errors:** In theory you should never see a malformed (bad) packet returned. All data is sent/received with error checking. If the packet between XBee modules contained errors, there is a high probability the error was detected and the packet discarded and resent up to 3 times until properly acknowledged. A more likely source of errors is the serial link between the XBee and the PC (or controllers when used).



**Figure 3-7: Range Testing and RSSI Monitoring**

This is a good time to test environmental conditions on your transmission link.

- ✓ Test for interference of the signal by placing your hands over the Base XBee, blocking line-of-sight with your body, placing a metal box over an XBee, or moving the Remote XBee to a further location or a different room or floor in the building.
- ✓ Click Stop when done testing.

#### **XBee Configuration**

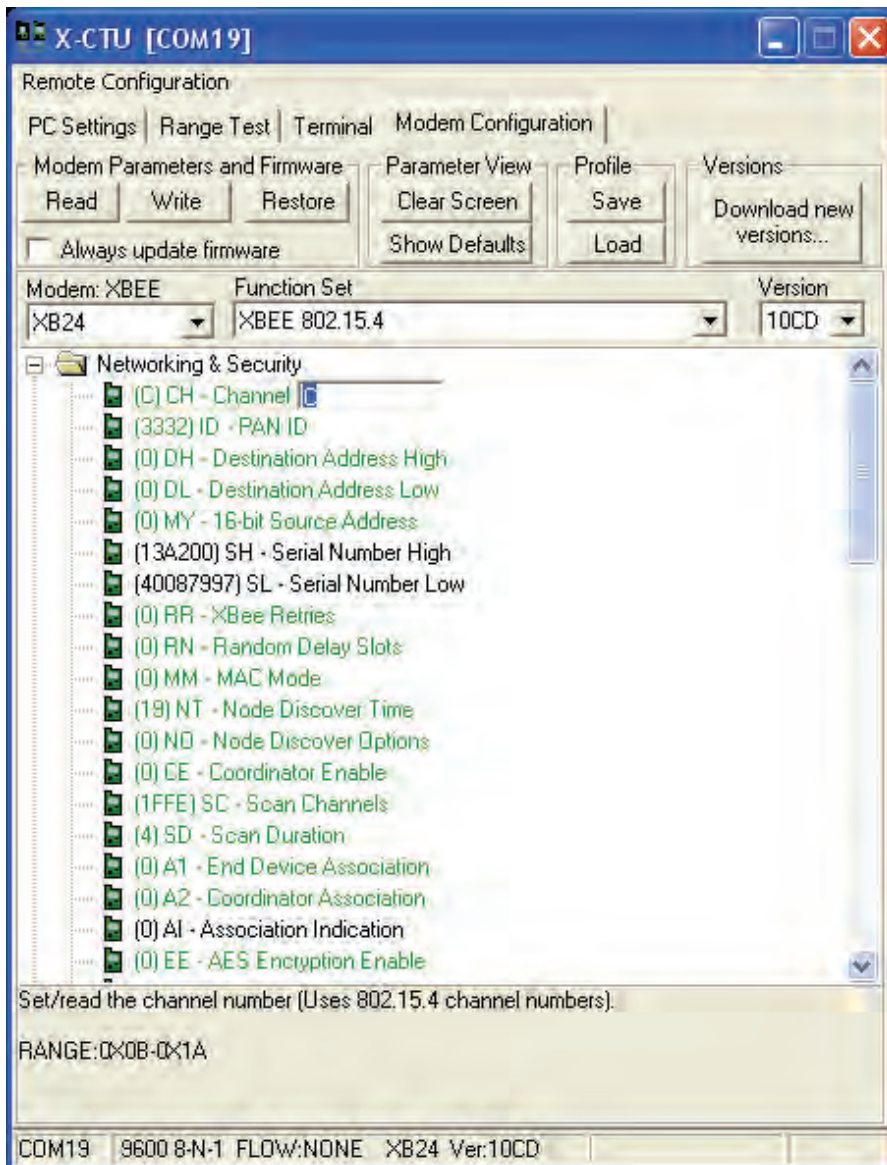
Using X-CTU, the configuration of the XBee may be modified and the firmware upgraded. Through the software, the XBee configuration may be read, changed and written for updates. The XBee may

also be configured through any terminal interface or from the controller by sending the correct character sequences.

#### Using the Modem Configuration Tab

The Modem Configuration tab may be used to read, modify, and write changes the XBee's configuration, and to update the firmware on the module. Notice the drop-down boxes near the top. They list the type of modem found, the firmware type on the module and the firmware version.

- ✓ With an XBee connected, open X-CTU and select the correct COM port.
- ✓ Click the Modem Configuration tab.
- ✓ Click the Read button. If all goes well, you should see the window populated by the current settings as shown in Figure 3-8.
- ✓ Click the Restore button. Once complete as indicated in bottom text area, click Read again. This ensures the XBee has the default configuration in the event it had been modified by you or someone else.



**Figure 3-8: Using “Modem Configuration” to change settings**

## 3: XBee Testing and Configuration

---

### Modifying Settings

To modify settings, click the setting, such as (C) CH – Channel as shown in Figure 3-8. The current setting is C. Note at the bottom of the window a brief description and valid range is provided. The channel (CH) may have a range of C to 1A on the XBee (non-Pro). On XBee-Pro it is C to 17. All parameters are in hexadecimal.

- ✓ Click the CH – Channel setting
- ✓ Modify to “D.”
- ✓ Write then Read and verify the change (current setting is shown in parentheses before the command name)
- ✓ Restore and Read and verify it is back to C.



**Configuration Settings:** Changes to configurations are stored in non-volatile memory, meaning that they are permanent until changed again or restored.

### Updating Firmware

Different firmware versions have different features, though most are simply new revisions with some fixes. In order to use analog/digital features, version 10A1 or higher is needed. To perform remote configuration features, version 10CD or higher is needed. If you don't have these newer versions or if you want to have the latest, use Download new versions... to check for firmware updates online.

Loading updated firmware into the XBee requires more communications than simply Tx and Rx. DTR and CTS are used as well. While configurations may be read and changed with a simple Tx/Rx interface, a full interface is needed for firmware changes, such as with the XBee USB Adapter.



**Updating Warning!** Updating occasionally does not go well. Multiple retries may be required. A failure to update properly could cause your XBee to become inoperable. While most updates go smoothly, there always exists the possibility of a corrupted update.

To change the firmware on your XBee:

- ✓ The XBee module must be in an XBee USB Adapter Board connected to your PC with the USB cable.
- ✓ In X-CTU, perform a Restore and then Read the XBee.
- ✓ Select the firmware Version.
- ✓ Check the Always update firmware checkbox.
- ✓ Click Write. The new firmware and current settings should be downloaded to the XBee module.
- ✓ Uncheck the Always update firmware checkbox.



### Using the Terminal Interface (AT Command Mode)

The XBee may also be configured using AT commands via the X-CTU terminal interface or any terminal program. This concept is important; we will duplicate it when configuring the XBee from our controller. The first step is to place the XBee into a Command Mode. We don't want this to happen accidentally, so a specific sequence is required to enter Command Mode:

- A 2 to 3 second pause since data was last sent
- A string of “+++” (with no Enter key pressed)
- Another 2-second pause
- The XBee will reply with “OK” when ready. The timeout value on Command Mode is fairly short, so too long a delay will cause it to exit the mode requiring you to re-enter the mode if needed.

Let's test it (example shown in **Figure 3-9**).

- ✓ Go to the Terminal tab.
- ✓ Type “Hello” to ensure you are communicating with the Remote loopback XBee.
- ✓ Wait 3 seconds.
- ✓ Type “+++” (With no Enter or Carriage Return).
- ✓ Wait 2 more seconds, and you should see “OK.”
- ✓ Enter **ATCH** & Enter.
  - You should see “C” returned—you requested the current value for channel.
- ✓ Enter **ATCH D** & Enter.
  - You should see “OK.”
- ✓ Enter **ATCN** & Enter to exit the Command Mode.
- ✓ Type “Hello” again.
  - You should NOT see it returned from the Remote XBee. The Base XBee is on Channel D and the Remote XBee is on Channel C so they will not communicate.
- ✓ Repeat the above changing the Base XBee's channel back to C (**ATCH C**) and test communications with the Remote XBee.
- ✓ If you timeout (suddenly your characters are echoed back when trying to enter a command), use **ATCT 500** to increase the timeout value.
- ✓ When complete, use **ATRE** to restore your XBee to default values. Multiple commands may be entered on a single line:  
**ATCH C,CT 500,CN** & Enter



**Note: Changes made to configuration settings using AT Commands are temporary.** The last saved setting will be used if the XBee is reset or powered down & up. To store your settings permanently, use the AT Command **ATWR** before exiting Command Mode. **ATRE** may be used to restore to default, but be sure to use **ATWR** after to save to memory.

### 3: XBee Testing and Configuration

---

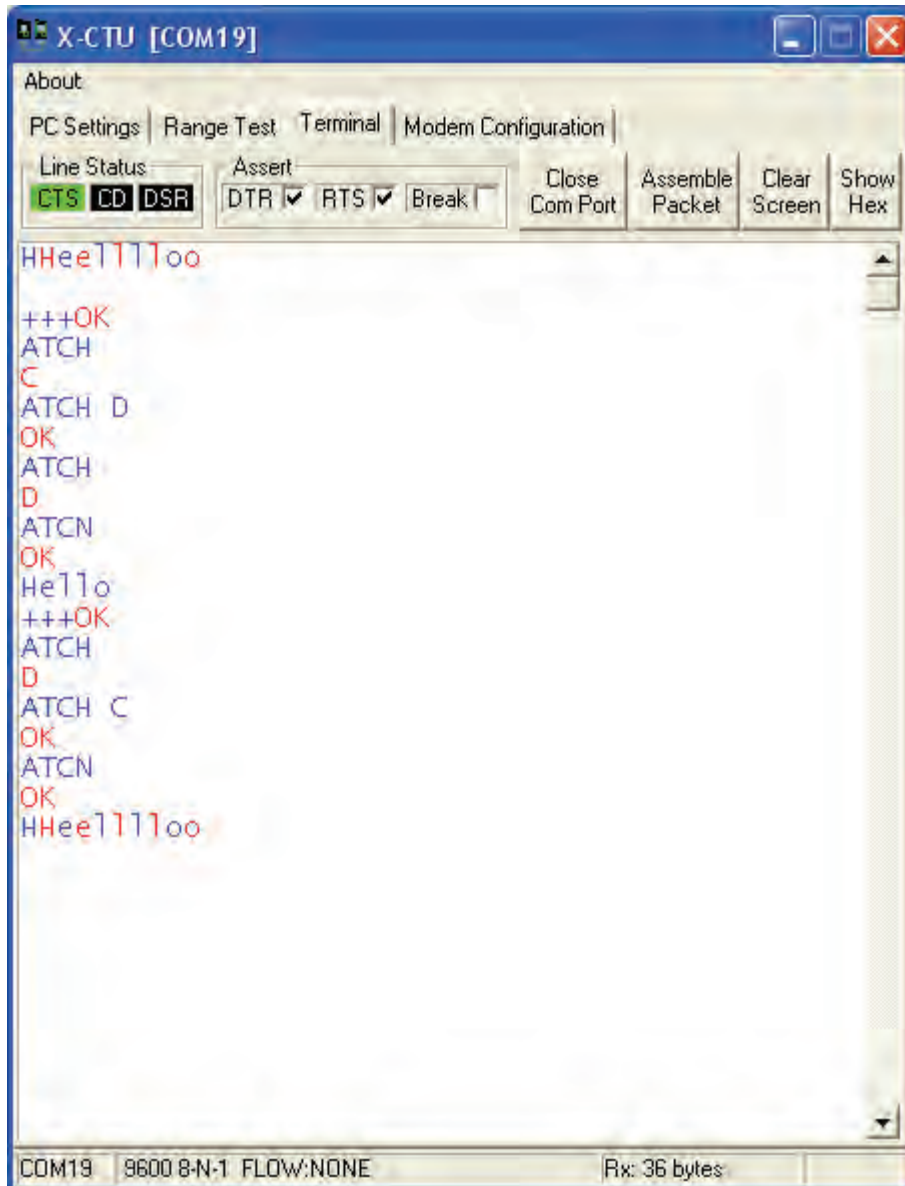


Figure 3-9: Using AT Command Mode in X-CTU Terminal

### Overview of Common Configuration Settings

#### Addressing—MY & DL

The address of the XBee defines its place on the network. It can be used to specify data to a particular node (destination) and the sender's address (source) in some cases. By default, all XBee's have an address of 0 (the **MY** address) and a destination low address of 0 (**DL**). In our loopback testing, the XBee modules used the default configuration and both were sending data from address 0 to address 0 being able to communicate with each other. While this works fine for 2-node networks, in larger networks you may want to send data to a particular node instead of several using the same address. To do this, node address (**MY**) and destination Low (**DL**) address need to be set. A broadcast address (**DL** of FFFF) may be used to send data to all nodes on the same network.

With your loopback configuration up and communicating, let's test some addressing.

- ✓ Using the Terminal tab, type something and ensure it is echoed back to the Base XBee.

- ✓ Notice that on both the Base and Remote XBee, the RSSI LED lights and the TX/RX LEDs will blink (if available).
- ✓ Using the Modem Configuration or terminal AT commands, change **DL** to a value of 1 (setting the destination of the data to address 1: **ATDL 1**).
- ✓ Type a string such as “Hello”. The data should NOT be echoed back.
- ✓ Observe the LEDs on the Remote XBee and loopback XBee.

The RSSI and Rx LEDs should not light. Your data is intended for address 1. Since the Remote XBee is on address 0 it will see the data but ignore it. It will NOT pass it through your DOUT pin (lighting the Rx LED if connected) and returning the data.

- ✓ Change DL back to 0.
- ✓ Change MY to 1 to set the address of your unit to address 1.
- ✓ Send some data.

You should see the RSSI LED and Rx LED as well as the Tx LED light on the Remote XBee. You should NOT see the RSSI or Rx LED light on the Base XBee. The Remote XBee accepted the data addressed to 0, and looped it back to send it to its **DL** address of 0. Since the Base XBee is on address 1, it will not accept the data since the address did not match.

- ✓ Change **MY** back to 0.
- ✓ Change **DL** to FFFF, the broadcast address.
- ✓ Send some data—it should once again be looped back and displayed. The **DL** address of FFFF informs all Remote XBee modules to accept the data regardless of their address (broadcast to all).
- ✓ Return the **DL** address to 0.

Remote XBee modules may be manually configured for different addresses. If you have multiple Remote XBee modules, you can set them up for loopback and the addressing scheme shown in Figure 3-10.

- ✓ On X-CTU, return to the [PC Settings](#) tab.
- ✓ Disconnect the USB cable.
- ✓ Power-down and remove the Remote XBee from its board and place it in the XBee USB Adapter.
- ✓ Reconnect USB.
- ✓ Read and modify **MY** and **DL** addresses as shown in Figure 3-10 using Modem Configuration to ensure it is not temporary.
- ✓ Replace it in the Remote adapter with loopback.
- ✓ Repeat for other Remote XBee modules.
- ✓ Using the Base XBee, modify the **DL** and **MY** addresses to test communications to one and to both (broadcast) Remote units. The Base will need a **MY** of 1 to accept data returned. It will need a **DL** of 2 or 3 to get data to the Remotes. Use a **DL** of FFFF to send and receive from BOTH Remotes.

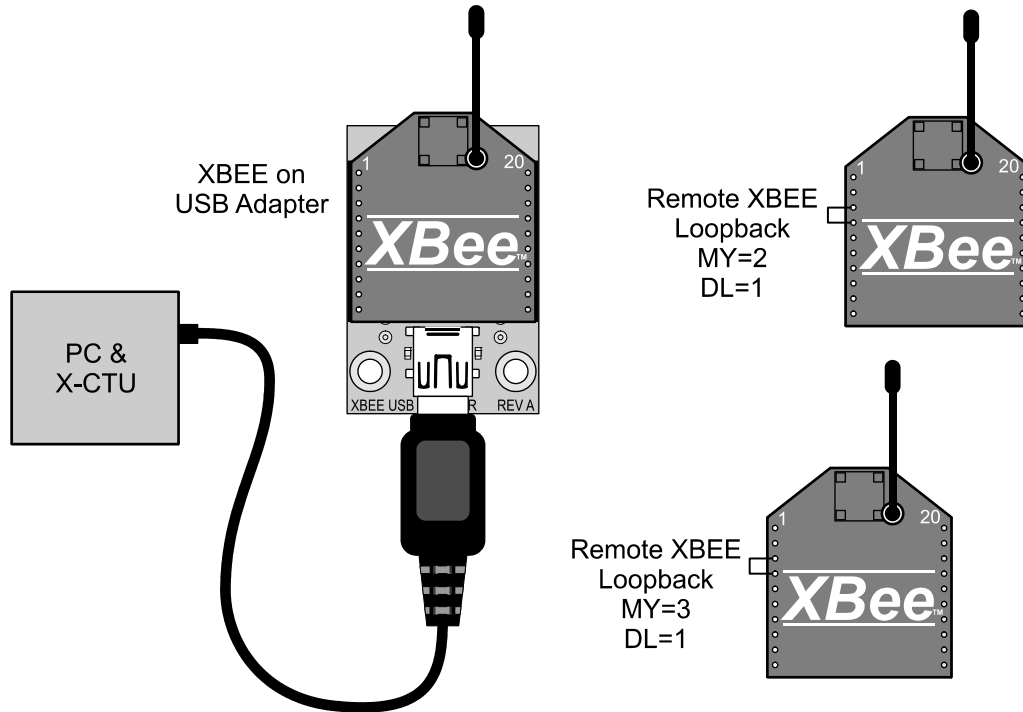


Figure 3-10: Multi-Node Addressing Test



**Try it!** Configure the addresses of all units to send data from Base to 1st Remote, from 1st Remote to 2nd Remote, and from 2nd Remote back to Base.

Hint: You can modify the RP setting to reduce the amount of time the RSSI light is on to better see each character being received in the event you don't have Tx/Rx LEDs connected. Use **ATRP 2** for a 200 ms blink of RSSI light.

✓ Use Restore to return all XBee modules to default configurations when testing is complete.

#### **Node Identifier—NI**

The node identifier allows us to assign a noun name to the XBee node to be seen when a node discovery is performed: **ATNI Node 1**

#### **Node Discovery—ND**

One other interesting test is the Node Discovery (**ND**) command. Using AT Command Mode in the terminal window, enter **ATND** and wait. Other nodes in your network will be listed showing:

- Their **MY** address
- Their serial numbers
- Their noun names if set (**NI** setting).
- The RSSI level (signal strength) of the received information from that node (further on the RSSI level is discussed).

### **Network Separation—ID & CH**

In addition to node addresses, sometimes we may want different group of XBee modules in the same area to be isolated from one another for communications—different networks in the same physical area. This may be done by using either unique Personal Area Network (PAN) IDs (**ID** setting), or by placing them on different channel frequencies (**CH**).

Using unique PAN IDs, even though two XBee modules may have the same address, only the XBee with a matching PAN ID will accept and use the data. This allows multiple networks to operate in the same area where XBee modules may have the same addresses.

- ✓ You may test the lack of communication by changing the ID setting of your Base XBee and test the loopback.

Note that this does not reduce the number of nodes fighting for the use of the frequency medium, which is called contention. Even though IDs are different, the XBee modules are still on the same frequency channel (**CH**) and only one XBee can transmit on that channel at a time if in RF range of each other.

XBee networks may also be separated by putting different networks on different frequency channels (**CH**). This has the benefit of isolating the networks and having them operate on different frequencies, reducing contention. Beyond separation, there may be “better” frequencies to operate at than others in an area. With the plethora of devices in the 2.4 GHz spectrum, some channels in the spectrum may be noisier than others. Using the Energy Detect command, you may monitor the signal strengths of the channels using the **ATED** command.

- ✓ Use the terminal window and enter AT Command Mode.
- ✓ Enter **ATED** to view energy on various channels, from lowest channel (C) to highest (1A, or 17 if using an XBee Pro).

**Figure 3-11** shows the results of two tests. The first was with a nearby microwave oven off, the second with it on. The higher the magnitude it shows, the less noise on that channel. A value such as 4E would convert to a decimal value of 78, so this channel would have a noise level of -78 dBm. We want as little noise in the background as possible. A higher magnitude means a lower noise level. Having the microwave on raised the noise on nearly all channels (there is also WiFi in the area on 2.4 GHz which may be affecting channel readings as well).

### 3: XBee Testing and Configuration

---

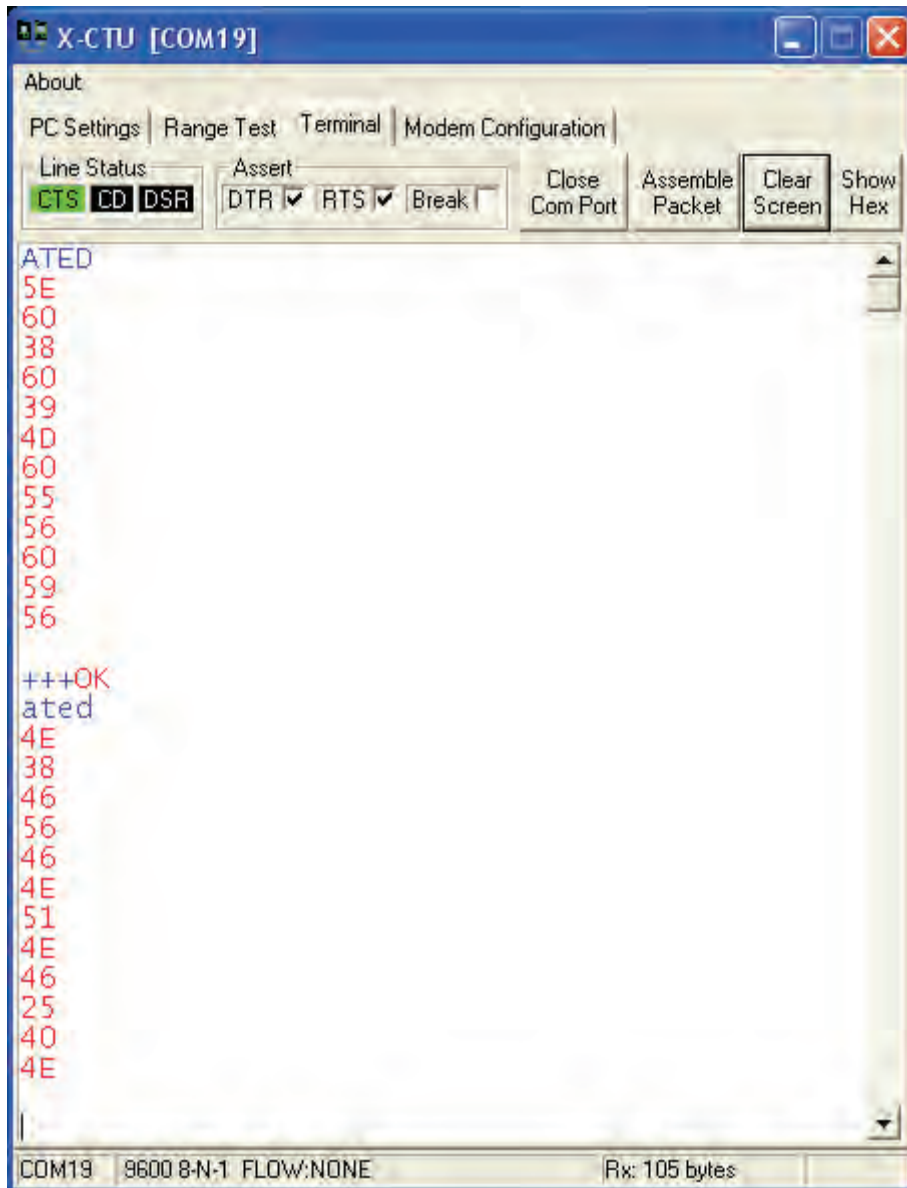


Figure 3-11: Energy Detection Tests

#### Receiver Signal Strength—DB

The XBee can report the receiver signal strength (RSSI) in a number of ways. One way is to poll the unit using the **ATDB** command. The XBee will report RSSI level as a hexadecimal value indicating the magnitude. This value, converted to decimal and made negative, is the signal strength from -40 to -100 dBm. The more negative the signal strength (in dBm), the weaker the signal. Therefore, -50 dBm is better than -60 dBm).

- ✓ Use the Terminal tab to send data to be echoed back.
- ✓ Enter Command Mode and enter **ATDB** for RSSI level.

#### Packetization Timeout—RO

As the XBee accepts serial data, it only waits so long before packetizing and transmitting the data. The packetization timeout sets how long it will wait before transmitting the incoming serial data. If this value is too short, it may cause problems in that data we transmit becomes fragmented. In many

cases the receiving XBee will reconstruct the data and buffer it for our use, but it does lead to more transmissions than needed. Under certain conditions, such as using API Mode on the receiving XBee, this may cause issues.

In a program we may have code that sends some data, does some calculations, and sends more data. We wanted the data to be sent as one transmission, but while doing a calculation, the first chunk of data is transmitted. By increasing the value of **RO** we can help ensure our data stays together and is sent in one transmission. The value of **RO** is the number of character times we want the XBee to wait before sending the collected data. At 9600 bps, each character requires approximately 1 ms (0.1 ms per bit for 10 bits with the start/stop bits). With a default RO value of 3, the XBee will wait approximately 3 ms (3 character times) before packetizing and sending data in the buffer. This value can be increased to FF (255 decimal) for a timeout value of over 255 ms at 9600 baud.

- ✓ Test the loopback by typing “Hello World” quickly. Notice that each character is transmitted and echoed.
- ✓ Change the value of RO to FF using the Command Mode or Modem Configuration (**ATRO FF**).
- ✓ Type “Hello World” once again. If you’re fairly quick you can type the entire phrase before it is packetized, transmitted and echoed.

### **Configuring I/O—D0 to D8, P0, P1, M0, M1**

The I/O on the XBee can be used for a number of uses: digital and analog input (requires API packet parsing covered later), digital output, and communications handshaking (RTS, CTS, etc). Additionally, these and other pins can be used for special functions such as association indication, RSSI indication, Sleep Mode input and PWM output. Additionally, inputs on one XBee can control outputs on another using line-passing (see Chapter 6).

Due to the number of options available, the Modem Configuration tab is useful for seeing the options as well as configuring from there. We’ll test a few out. As shown in **Figure 3-12**, Pin D5 can be used as the association indicator, ADC or digital input, and digital output low and high. Configuration of the pin can be done here or in Command Mode.

- ✓ For example, if you are tired of your association indicator on the XBee board blinking and wish to turn it off, set D5 to 4 (**ATD5 4**).

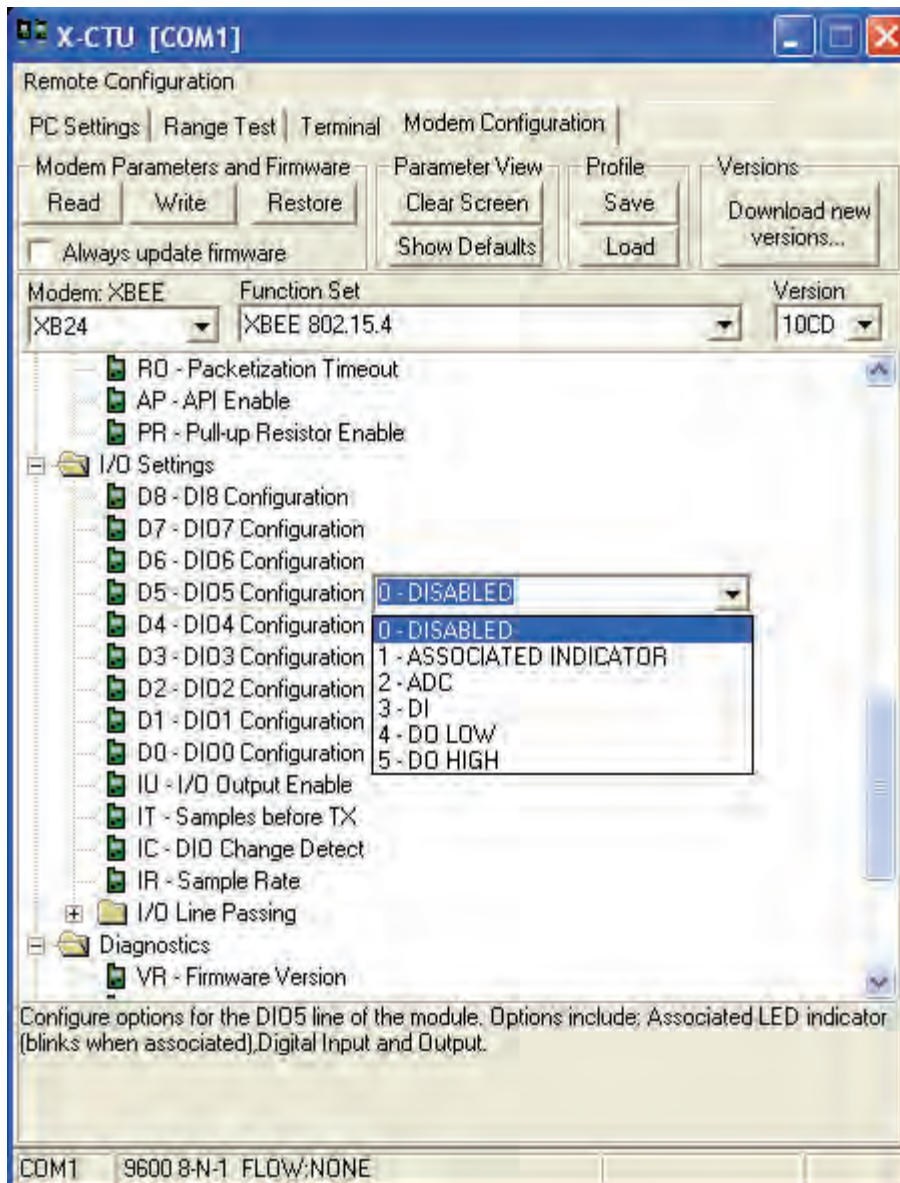
Two I/O can be used to generate Pulse Width Modulated output (PWM), such as the one driving the RSSI LED on your board. The function configuration is in the I/O Line Passing section. To set the PWM value, use Command Mode. To set the RSSI indicator for a certain output level to control LED brightness:

- ✓ In Command Mode, set PWM0 to PWM Output—**ATP0 2**.
- ✓ Set the PWM 10-bit value, in the range of 0 to 3FF (0 to 1023 decimal): **ATM0 5**.
- ✓ Test other PWM values and observe brightness. Your eye is filtering the LED blinking on and off quickly for the dimming effect. The output is still digital, but it may be filtered for analog voltage output.



**Note:** The XBee PWM is not the same type of PWM used for servos. This is a PWM-controlled voltage level as opposed to a PWM signal.

### 3: XBee Testing and Configuration



**Figure 3-12: Digital Input/Output Configuration**

#### **Common Configuration Settings**

Table 3-1 on page 40 is a list of common configuration settings used in this tutorial. For a full list, please see the Command Summary in the XBee Manual. Please see the referenced documentation or XBee X-CTU software for valid ranges and other information on use.

#### **Summary**

The loopback test is an excellent means of testing communications and various configurations. The XBee has a multitude of configuration settings to deal with addressing, interfacing and input and output control. Using the X-CTU interface allows range testing with loopback, modem configuration and a terminal window for communications, testing and configuration. In the next chapter we will explore code for the controllers in sending and receiving data and configuring the XBee.



**Table 3-1: Common Configuration Settings**

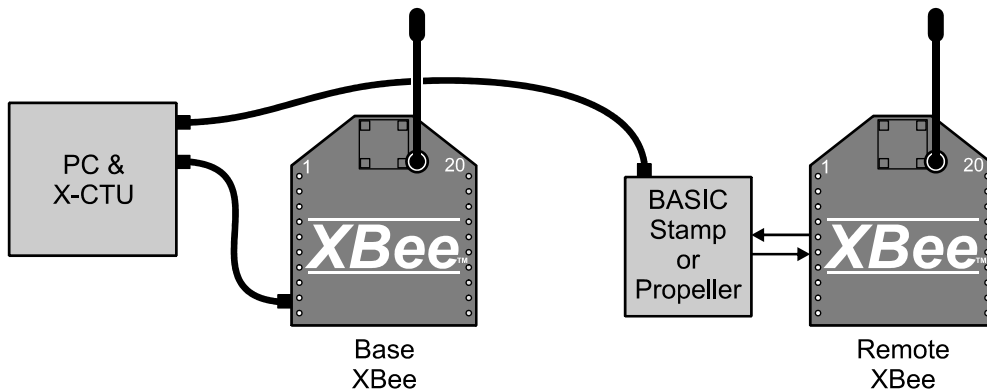
Group	Command	Meaning and Use
<b>Networking &amp; Security</b>	CH	Channel: Sets the operating frequency channel within the 2.4 GHZ band. This may be modified to find a clearer channel to to separate XBee networks. Default value is C.
	ID	PAN ID: Essentially, the network ID. Different groups of XBee networks can be separated by setting up different PANs (Personal Area Networks). Default value is 3332.
	DL	Destination Low Address: The destination address of the XBee for which the transmitted packet is indented. 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.
	NI	Node Identifier: Sets a node's noun name.
<b>Sleep Modes</b>	SM	Sleep Mode: Allows the sleep mode to be selected for low power consumption (<10 $\mu$ A). It is discussed in Chapter 6.
<b>Serial Interfacing</b>	BD	Interface Data Rate: Sets baud rate of the serial data between the XBee and the controller or PC.
	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 very powerful mode will be explored in Chapter 6.
	RO	Packetization Timeout: In building a packet to be transmitted, this sets the length of time the XBee waits for another character before the packet is sent.
<b>I/O Settings</b>	D0–D8	Sets the function of the I/O pins on the XBee, such as digital output, input, ADC, RTS, CTS and other.
	P0, P1	Configures the function of PWM0 and PWM1
	RP	Sets the time length for the RSSI output.
	M0, M1	Sets the PWM value of the PWM outputs
	IR	Sample Rate: The XBee can be configured to automatically send data from digital I/O or ADC's. It requires the receiving node to be in API Mode and the data parsed for the I/O values. See Chapter 6.
<b>Diagnostics</b>	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 2 retries before failure. Additional retries can be added by using RR setting.
<b>AT Command Options</b>	CT	AT Command Timeout: Once in Command Mode, sets the length of the delay before the XBee returns to normal operation.
	GT	Guard Time: When switching into AT Command Mode, defines how long the guard times should be (absence of data before the command line) so that accidental mode change is not performed.
<b>Configuration</b>	WR	Write the configuration to non-volatile memory
	RE	Restore to default configuration in volatile memory

### 4: Programming for Data and XBee Configurations

Data between systems can take several forms depending on need. It may be raw bytes, which can represent a variety of information such as characters, input/output bits, or simply a binary value. While the XBee modules can help ensure the data arrives correctly to the controller, it is up to our code to ensure it is accepted properly and used in the manner desired. Just because our data makes it out of the XBee it doesn't necessarily mean it was accepted properly by the controller.

In some cases a controller may send a value repeatedly, such as a sensor value. It may not be important that some data is missed by the controller as long as a recent sample is obtained. In other cases it may be more critical that each transmission is accepted and used by the controller. Sometimes data can simply be a single byte. Other times it may be a collection of values that needs to be accepted in the proper order.

This chapter will explore data reception and transmission by the BASIC Stamp and Propeller microcontrollers and explore some ways of handling different tasks for data reception. It will also explore configuring the XBee from the controllers using the AT Command Codes. A Base node will be used connected to the PC to communicate with a Remote node of an XBee interfaced to a microcontroller as shown in Figure 4-1. The Remote node's controller is connected to the PC as well so it can readily display the values it receives from the XBee, so we can rapidly reprogram the controller to reconfigure the XBee, and to provide the XBee with data to send to the Base node.



**Figure 4-1: Base and Remote XBee General Setup**

#### **Base and Remote Nodes Revisited**

Throughout this tutorial we explore communications between XBee nodes. At times the XBee may be directly connected to a PC for communications and other times the XBee will be connected to a microcontroller. We will use the terms Base node and Remote node to identify the typical function of each.



**Base node:** The Base node will be the node from which an operator would normally monitor or control the wireless network. The Base node will be the one commonly connected to a PC for an operator to monitor, send data and control the system.

**Remote node:** The Remote node will be the node that is typically at a remote location to perform the actual data collection or control of actuators in a system. In normal applications, this node would not be connected to a PC, but would send its data to the Base node and receive instructions. In many of our examples we monitor the Remote node as well as the Base node to observe interactions from both sides.

## BASIC Stamp

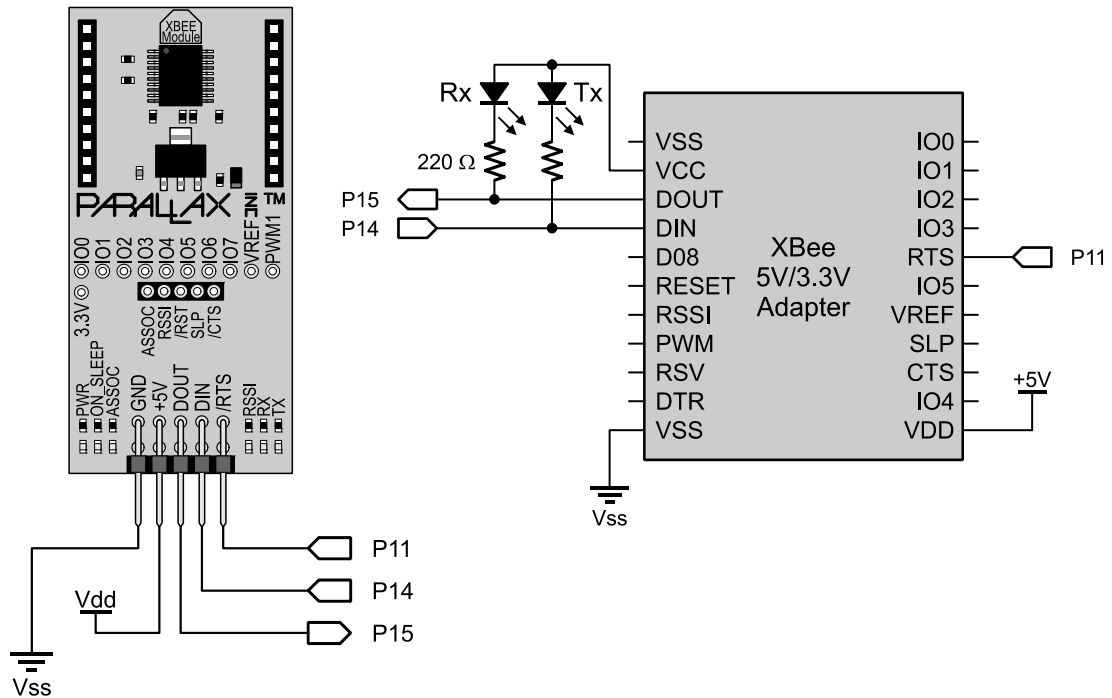
### Hardware & Software

For these tests a single Remote node using the BASIC Stamp will be used for communications with a Base node of an XBee connected to the PC using USB and X-CTU software. Data will be manually entered to be transmitted to the BASIC Stamp to investigate data reception.

#### Hardware:

- (1) BASIC Stamp microcontroller and development board and cable
- (2) XBee modules (Ensure they are restored to default configuration)
- (1) XBee SIP Adapter, or XBee 5V/3.3V Adapter Board
- (1) XBee USB Adapter Board & cable
- (2) 220 Ω resistors and (2) LEDs (optional)

- ✓ Connect the BASIC Stamp to an XBee Adapter Board using one of the options shown in Figure 4-2.. LEDs and resistors are optional but are useful. Note that the DOUT and DIN pins are not jumpered together.



**Figure 4-2: XBee SIP (right) or 5V/3.3V (left) Adapter Connection to BASIC Stamp**

**XBee 5V/3.3V Adapter Rev B:** The XBee 5V/3.3V Adapter Board Rev B has onboard Tx and RX LEDs, so you will not need to add the LED circuits that are shown in the schematic above.

#### Software:

- BASIC Stamp Editor for Windows ([www.parallax.com/basicstampsoftware](http://www.parallax.com/basicstampsoftware))
- Digi's X-CTU Software ([www.parallax.com/go/xbee](http://www.parallax.com/go/xbee))

## 4: Programming for Data and XBee Configurations

---

### Simple DEBUG to Remote Terminal

Being able to monitor a Remote unit is very beneficial, especially with robotics applications or for gathering data from remote sensors. This first code example will display text strings and values that are sent from the Remote node to the Base node for monitoring.

```
' *****
' Simple_Debug.bs2
' Sends data to Remote terminal for monitoring
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}

' ***** Constants & PIN Declarations *****
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
    T9600      CON      84
#CASE BS2SX, BS2P
    T9600      CON      240
#CASE BS2PX
    T9600      CON      396
#ENDSELECT
Baud          CON      T9600

Rx            PIN      15   ' XBee DOUT - Not used in this example
Tx            PIN      14   ' XBee DIN
RTS           PIN      11   ' XBee RTS - Not used in this example

' ***** Variable Declarations *****
Counter       VAR      Byte

' ***** Main LOOP *****
PAUSE 500          ' 1/2 second pause to stabilize comms
SEROUT Tx, Baud, [CLS,"Program Running...",CR]

PAUSE 2000        ' Pause before counting

FOR Counter = 1 TO 20      ' Count and display remotely
    SEROUT Tx, Baud, ["Counter = ", DEC Counter, CR]
    PAUSE 100
NEXT

SEROUT Tx, Baud, ["Loop complete.",CR]
END
```

### Code discussion:

- The **#SELECT-#CASE** conditional compilation structure is used to ensure a baud rate of 9600 no matter the model of BASIC Stamp you may be using.
- DOUT and RTS are not used in this example but will be used in later examples.
- It sends a message, the value of **Counter** from 1 to 20, and a final message “Loop complete.”.

## 4: Programming for Data and XBee Configurations



**Terminals:** While we use the X-CTU in our example, you may also use a BASIC Stamp Editor Debug Terminal selected to the correct COM port. The `CLS` (clear screen) only shows a dot in X-CTU, but will clear the Debug Terminal.

### Test the code:

- ✓ Connect and monitor the Base XBee using X-CTU or other terminal program.
- ✓ Download Simple\_Debug.bs2 to the BASIC Stamp connected to the Remote XBee unit.
- ✓ Monitor the received data in the Base node's terminal window as shown in Figure 4-3.

```
.Program Running...
Counter = 1
Counter = 2
Counter = 3
Counter = 4
Counter = 5
Counter = 6
Counter = 7
Counter = 8
Counter = 9
Counter = 10
Counter = 11
Counter = 12
Counter = 13
Counter = 14
Counter = 15
Counter = 16
Counter = 17
Counter = 18
Counter = 19
Counter = 20
Loop complete.
```

COM15 9600 8-N-1 FLOW:NONE Rx: 286 bytes

**Figure 4-3: Simple Debug to Base Testing**

## 4: Programming for Data and XBee Configurations

---

### Accepting and Displaying Single Bytes

In this next code example, we will use the X-CTU terminal to transmit bytes to the BASIC Stamp on the Remote node to accept, display locally and transmit back to the Base node for display in X-CTU (echoed).

```
' *****
' Simple_Byte_Receive.bs2
' Accepts, displays and echoes back incoming bytes
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}
' ***** Constants & PIN Declarations *****
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
    T9600      CON      84
#CASE BS2SX, BS2P
    T9600      CON      240
#CASE BS2PX
    T9600      CON      396
#ENDSELECT
Baud          CON      T9600

Rx            PIN      15      ' XBee DIN
Tx            PIN      14      ' XBee DOUT
RTS           PIN      11      ' XBee RTS - Not used in this example

' ***** Variable Declarations *****
DataIn        VAR      Byte

' ***** Main LOOP *****
PAUSE 500          ' 1/2 second pause to stabilize comms
DEBUG "Awaiting Byte Data...",CR

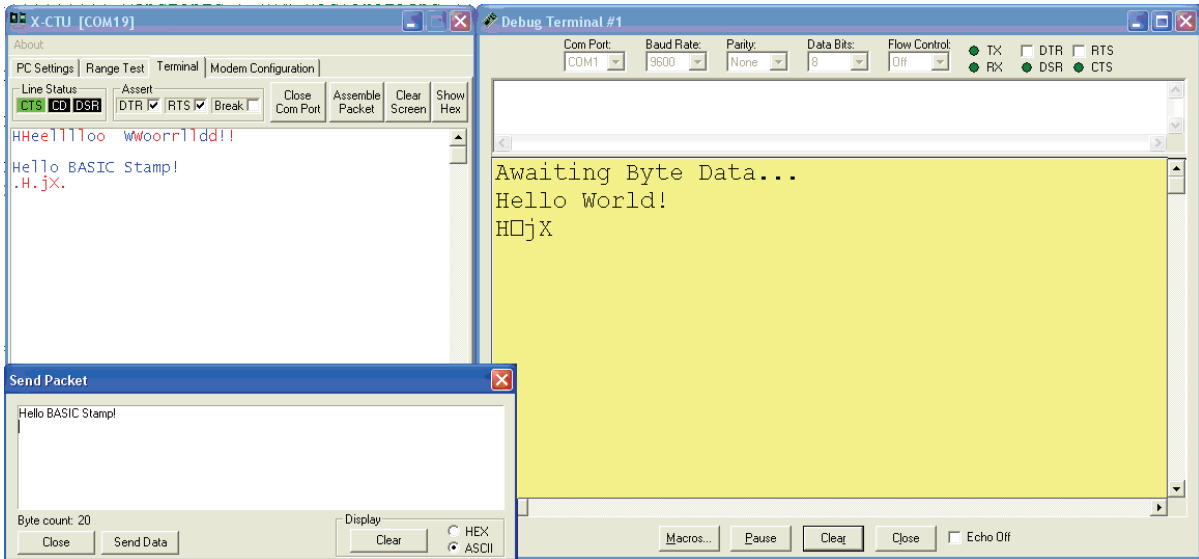
DO
    SERIN Rx, Baud, [DataIn]  ' Accept incoming byte
    SEROUT Tx, Baud, [DataIn] ' Echo byte back
    DEBUG DataIn              ' Display byte as character
LOOP
```

#### Code discussion:

- ✓ Within the **DO-LOOP**, **SERIN** waits until it receives a byte from the XBee, sends it back to the XBee to be transmitted, and sends the character to the BASIC Stamp Editor's Debug Terminal.

#### Test the code:

- ✓ Download Simple\_Byte\_Receive.bs2 to the BASIC Stamp.
- ✓ From the X-CTU terminal window, manually type "Hello World." Note that it displays in the BASIC Stamp Debug Terminal of the Remote node and the X-CTU terminal window of the Base node correctly.
- ✓ Use the Assemble Packet feature of X-CTU to send "Hello BASIC Stamp!" as a single packet. Note that the first character may be correct, but the remaining data is mostly garbage.



**Figure 4-4: Receiving Byte Test Results**

Figure 4-4 is an image of our testing. The first byte was accepted by the BASIC Stamp correctly since it was sitting idle waiting for it. Once the first byte was accepted, the program continued on. In the mean time, data continued to be sent from the XBee even though the BASIC Stamp was not accepting it any longer. When the BASIC Stamp looped back, it caught a byte mid-position and continued collection from there, getting garbage. When we typed the data directly in to the terminal window, we were typing too slowly to get the data out in a single transmission and the BASIC Stamp was fast enough to loopback for our next character.

### Accepting and Displaying Decimal Values

When sending and receiving raw bytes, it is generally more difficult to ensure data is being accepted properly since it could be any byte value, 0 to 255 in binary. Using decimal values can aid in ensuring data is properly received. When data is sent or received as decimal values, using PBASIC's **DEC** modifier, values are sent as characters representing the value. Using bytes, a value such as 123 is sent as a single byte with a value of 123. When 123 is sent and accepted using **DEC**, the value is an ASCII character string as '1', '2' and '3'. The BASIC Stamp will ignore or use as the 'end of string' any non-numeric character, limiting potential garbage and helping to ensure received data is good. Next, Simple\_Decimal\_Receive.BS2 demonstrates accepting, displaying and transmitting decimal values.

```

| *****
| Simple_Decimal_Receive.bs2
| Accepts, displays and echoes back incoming Decimal Value
| *****
| {$STAMP BS2}
| {$PBASIC 2.5}
| ***** Constants & PIN Declarations *****
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
    T9600      CON      84
#CASE BS2SX, BS2P
    T9600      CON      240
#CASE BS2PX
    T9600      CON      396
#ENDSELECT
    
```

## 4: Programming for Data and XBee Configurations

```
Baud          CON      T9600
Rx            PIN      15      ' XBee DOUT
Tx            PIN      14      ' XBee DIN
RTS           PIN      11      ' XBee RTS - Not used in this example

' ***** Variable Declarations *****
DataIn        VAR      Word
' ***** Main LOOP *****
PAUSE 500      ' 1/2 second pause to stabilize comms
DEBUG "Awaiting Decimal Data...",CR

DO
  SERIN Rx, Baud, [DEC DataIn]      ' Accept incoming Decimal Value
  SEROUT Tx, Baud, [DEC DataIn,CR]   ' Echo decimal value back
  DEBUG DEC DataIn,CR               ' Display value
LOOP
```

For testing, the X-CTU terminal is again used:

- ✓ Download Simple\_Decimal\_Receive.bs2.
- ✓ In the terminal window of X-CTU for the Base node, type in a series of numbers, each followed by Enter (Carriage Return), such as 10, 20, 30.
- ✓ Notice that only after Enter (or a non-numeric value) data is processed, displayed and sent back. The BASIC Stamp collects data until the character is no longer numeric.
- ✓ Click Assemble Packet. Using the Send Packet window, enter a series of numbers to be sent at once, again each followed by Enter, such as 40, 50, 60, etc.
- ✓ Send the packet and monitor the results. The first value is used and displayed, several are skipped, and then another is accepted and used as show in Figure 4-5.

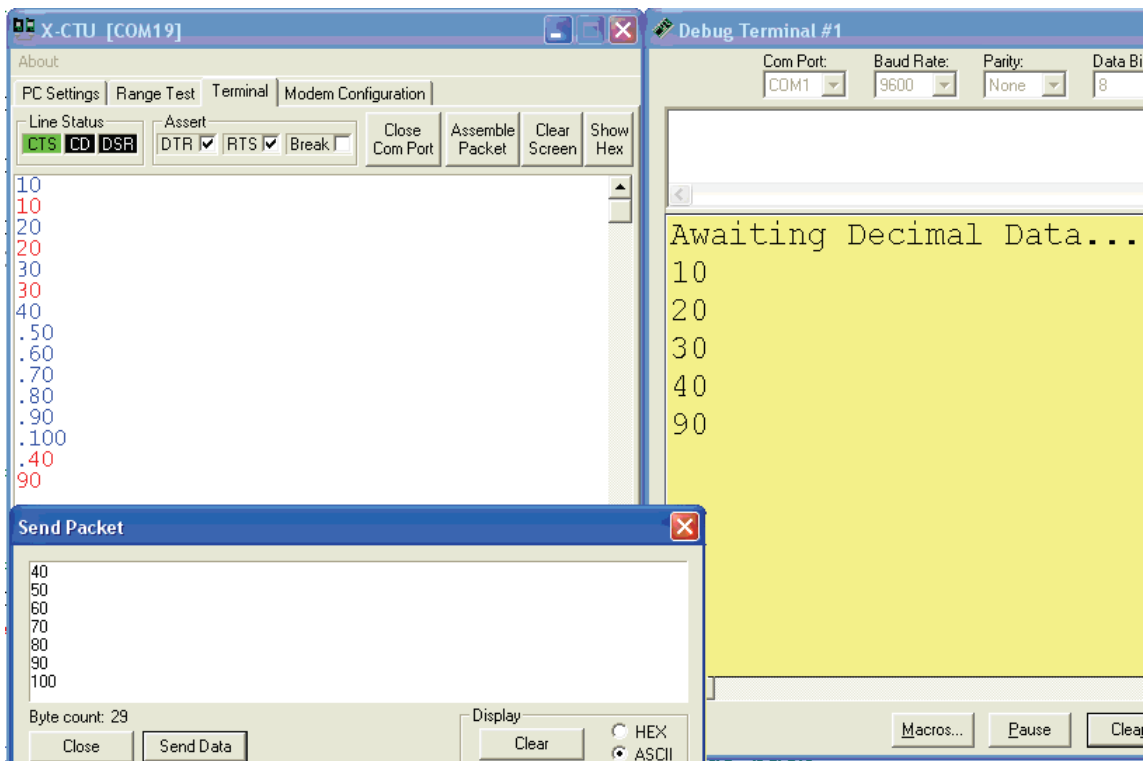


Figure 4-5: Simple Decimal Receive Testing



The XBee again received your data correctly and sent it to the BASIC Stamp, but after getting the first value, the BASIC Stamp was not available to capture much of the remaining data. Since the BASIC Stamp does not buffer incoming serial data, that data is lost. By using flow control, such as RTS, we can help prevent missed data.



### Range & Negative Values?

The BASIC Stamp, using Word-sized variables, can accept values up to 65535 (16-bit values). To use negative values, use the **SDEC** modifier instead for a range of -32768 to +32767.

### Configuring the XBee for using RTS Flow Control & Timeouts

Using the XBee module's RTS (Ready-to-Send) line and the **SERIN** command's /RTS modifier, we can limit lost data by having the XBee send data only when the BASIC Stamp is ready to receive it. Additionally, we can use the *Timeout* feature of **SERIN** so the BASIC Stamp can process other code while waiting for data to arrive. To enable RTS on the XBee, AT command codes will be sent by the BASIC Stamp.

The program `Using_RTS_Flow_Control_for_Bytes.bs2` demonstrates using RTS to collect buffered data from the XBee.

```
' *****
' Using_RTS_Flow_Control_for_Bytes.bs2
' Configures XBee to Use RTS flow control to
' prevent missed DATA
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}
' ***** Constants & PIN Declarations *****
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
#CASE BS2SX, BS2P
  T9600      CON      240
#CASE BS2PX
  T9600      CON      396
#ENDSELECT
Baud          CON      T9600

Rx            PIN      15
Tx            PIN      14
RTS           PIN      11

' ***** Variable Declarations *****
DataIn        VAR      Word

' ***** XBee Configuration *****
PAUSE 500
DEBUG "Configuring XBee...",CR
PAUSE 2000
SEROUT Tx,Baud,["+++"]           ' Guard Time
PAUSE 2000
SEROUT Tx,Baud,["ATD6 1,CN",CR]   ' RTS enable (D6 1)
                                   ' Exit Command Mode (CN)
```

## 4: Programming for Data and XBee Configurations

```
' ***** Main LOOP *****
PAUSE 500
DEBUG "Awaiting Multiple Byte Data... ",CR

DO
  SERIN Rx\RTS,Baud,1000,Timeout,[DataIn] ' Use Timeout to wait for byte
  SEROUT Tx, Baud, [DataIn]              ' Send back to PC
  DEBUG DataIn                           ' Display data
  GOTO Done                               ' Jump to done


  Timeout:                               ' If no data, display
    DEBUG ". "

  Done:

LOOP
```

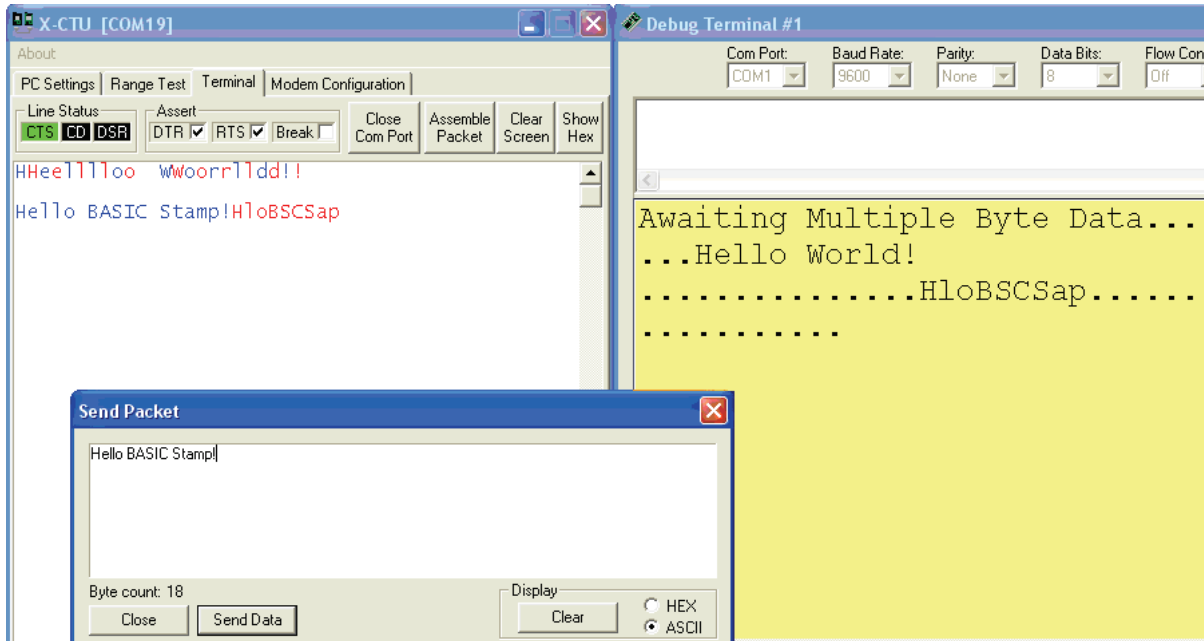
### Analyzing the code:

- In the XBee Configuration section, the identical action we took using the terminal window is used to configure the XBee: A short delay, sending +++, seeing "OK", another delay, sending AT commands and using the CN command to exit Command Mode. The Command Mode will also terminate automatically after a period of inaction.
- The **SERIN** instruction uses the **\RTS** modifier to signal the XBee when it is available to accept new data. The **1000** and **Timeout** label is used to branch if data is not received within 1 second, allowing processing to continue. (See the Timeout units note in the box below.)
- If data is not received within 1 second, the code at the **Timeout:** label will cause a dot to be displayed in the Debug Terminal to indicate that and to show processing by the BASIC Stamp is being performed.

	<b>Syntax Refresher for SEROUT and SERIN</b>
	<b>SEROUT</b> <i>Tpin</i> { \Fpin }, <i>Baudmode</i> , { <i>Pace</i> , } { <i>Timeout</i> , <i>Tlabel</i> , } [ <i>OutputData</i> ]
	<b>SERIN</b> <i>Rpin</i> { \Fpin }, <i>Baudmode</i> , { <i>Plabel</i> , } { <i>Timeout</i> , <i>Tlabel</i> , } [ <i>InputData</i> ]
	For full details on these and any other PBASIC commands or keywords, see the BASIC Stamp Editor Help.
	<b>Timeout units note:</b> The units in <i>Timeout</i> are 1 ms for the BS2, BS2e, and BS2pe, hence a <i>Timeout</i> argument of 1000 would be equivalent to 1 second. If you are using a BS2sx, BS2p, or BS2px, the <i>Timeout</i> units are only 0.4 ms, so the same value of 1000 would be equivalent to 400 ms—something to keep in mind while you read the code explanations throughout the book.

### Testing:

- ✓ Download Using\_RTS\_Flow\_Control\_for\_Byte.bs2 to the Remote node.
- ✓ Using X-CTU for the Base node, send a string in both the terminal window and using the Send Packet window.
- ✓ Notice, as in Figure 4-6, that when sending data as a packet, only every other byte is accepted and processed.



**Figure 4-6: Using RTS Flow Control for Bytes Testing**

The XBee is buffering data, allowing the BASIC Stamp to accept more without garbage, but every other byte is missed. The reason for this is that after collecting one byte, the BASIC Stamp de-asserts the RTS line to stop the XBee from sending more data, but it is too slow. The next byte has left the XBee before it gets notification to stop sending data. This happens every cycle causing every other byte to be missed.

i

**Monitor LEDs**

When the code starts and goes through the configuration sequence, watching the Tx & Rx LEDs blink back and forth is a great indicator that it the XBee is accepting your Command Mode functions.

### Using RTS and String Delimiter with Decimal Values

Using **DEC** for decimal values can help ensure data is received properly. One issue with receiving multiple values is ensuring the order in which they are received, and where to start. For example, let's say we want to send two values in pairs, and send several pairs to be accepted and used, such as:

```

10 & 20
10 & 30
10 & 50
    
```

On reception, the BASIC Stamp gets out of sequence so that the data it receives is:

```

20 & 10
30 & 10
    
```

Because it perhaps missed one, the sequence of how it is accepting data does not match the sequence we intended. Through the use of a start-of-string identifier or delimiter we can help ensure the data is collected starting at the correct value in the buffer. A unique character, one not used in normal data, may be used to identify the start of a string. Using **DEC** values, only 0–9 are valid characters so anything outside of that range would be unique from our data. As data of characters 0 to 9 flow in, a

## 4: Programming for Data and XBee Configurations

---

unique character such as '!' is easily denoted. Using\_RTS\_Flow\_Control\_for\_Decimals.bs2 demonstrates using RTS and a string start delimiter.

```
' *****
' Using_RTS_Flow_Control_for_Decimals.bs2
' Configures XBee to Use RTS flow control to
' prevent missed data using start delimiter
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}
' ***** Constants & PIN Declarations *****
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
    T9600      CON      84
#CASE BS2SX, BS2P
    T9600      CON      240
#CASE BS2PX
    T9600      CON      396
#ENDSELECT
Baud          CON      T9600

Rx            PIN      15
Tx            PIN      14
RTS           PIN      11
' ***** Variable Declarations *****
DataIn        VAR      Byte
Val1          VAR      Word
Val2          VAR      Word
' ***** XBee Configuration *****
PAUSE 500
DEBUG "Configuring XBee...",CR
PAUSE 2000                                ' Guard Time
SEROUT Tx,Baud,["+++"]                    ' Command Mode sequence
PAUSE 2000                                ' Guard Time
SEROUT Tx,Baud,["ATD6 1,CN",CR]          ' RTS enable (D6 1)
                                           ' Exit Command Mode (CN)
' ***** Main LOOP *****
DEBUG "Awaiting Delimiter and Multiple Decimal Data...",CR

DO
    SERIN Rx\RTS,Baud,500,Timeout,[DataIn] ' Briefly wait for delimiter

    IF DataIn = "!" THEN                    ' If delimiter, get data
        SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val1] ' Accept first value
        SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val2] ' Accept next value
                                           ' Display remotely and locally
        SEROUT Tx, Baud, [CR,"Values = ", DEC Val1," ", DEC Val2,CR]
        DEBUG CR,"Values = ", DEC Val1," ", DEC Val2,CR
    ENDIF

    GOTO Done                               ' Jump to Done
Timeout:
    DEBUG "."                               ' If no data, display dots
Done:
LOOP
```

### Analyzing the code:

- Again, RTS flow is configured for the XBee using AT commands.
- A `SERIN` with a 500 ms timeout starts the loop. If no data is received within this time, the timeout dot is shown.
- Next, the code checks to see if the character was the exclamation point `!` and if not, the code is done.
- If it was a `!` then two decimal values are collected, again using timeouts to ensure the code does not lock up while waiting.
- The data values are displayed locally and remotely.

### Testing the code:

- ✓ Download `Using_RTS_Flow_Control_for_Decimals.bs2` to the Remote node.
- ✓ Enter several values, the `!` with an Enter at the Base node's terminal window, then several more values. After entering `!` the next two values should be accepted and displayed.
- ✓ Use the Send Packet window to make a series of numbers and `!`'s to be sent, and test. Notice in Figure 4-7 each set of data following the delimiter was correctly processed.

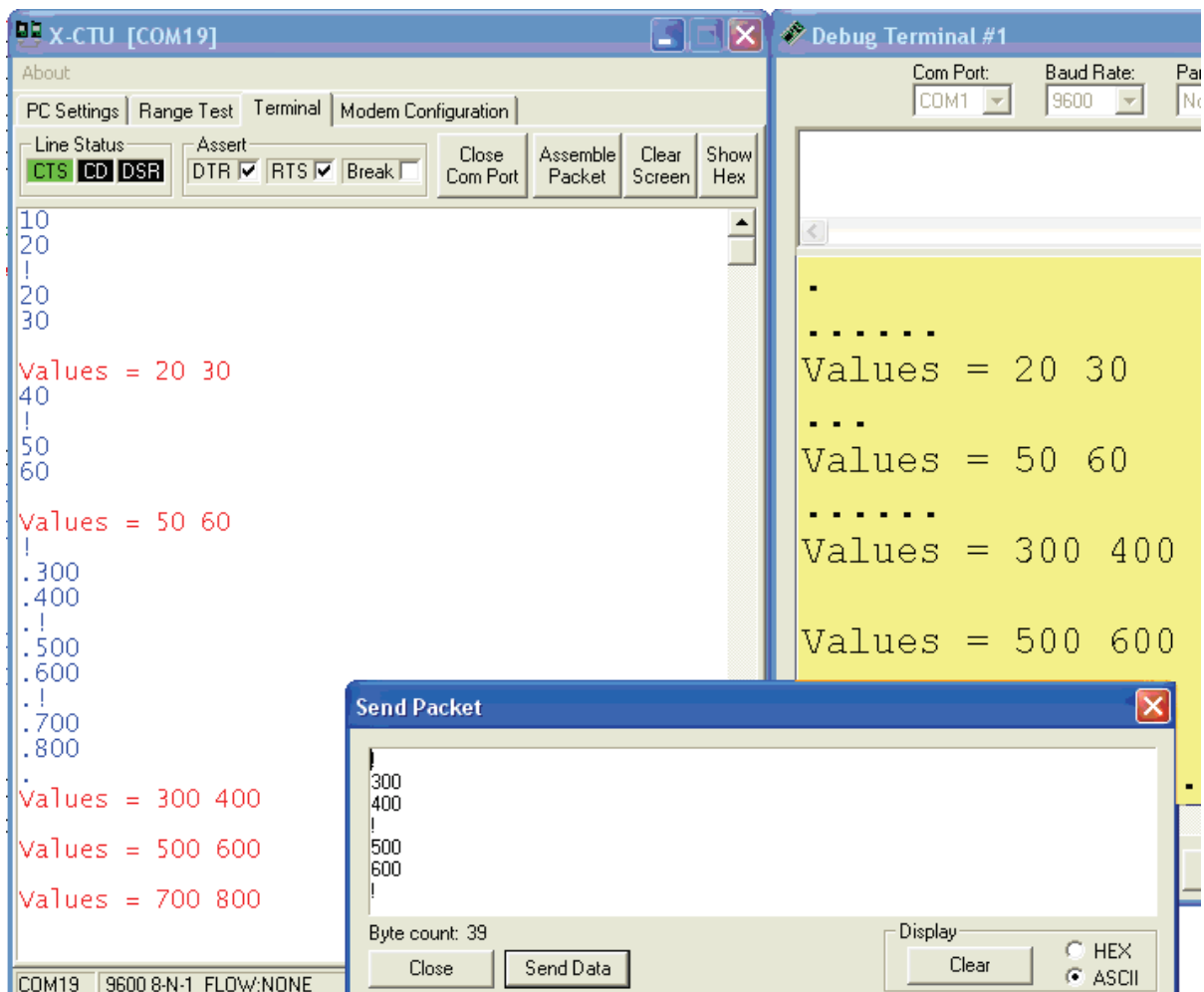


Figure 4-7: Using RTS for Decimal Values Testing

## 4: Programming for Data and XBee Configurations

---

The decimal values were adequately buffered so that no data was lost. What about those missed bytes when RTS de-asserts? Our Enter or carriage return was at the end of each string of characters so that the lost bytes corresponded to those Enter/carriage return characters.

### Reading Data from the XBee – Displaying RSSI Level

The BASIC Stamp can read settings and values from the XBee as well as configuring it. Just as we had done through the X-CTU terminal window in Chapter 3, values can be requested, though it can cause a few issues. Going into Command Mode and performing various AT commands, an “OK” along with data was returned. This “OK” can sometimes cause problems since it will also be sent to the BASIC Stamp. Another problem is to ensure the buffer is empty of other data, especially when using RTS, so that when the value is returned, it is accepted correctly. Finally, when going into AT Command Mode the guard times make this a slow process. By lowering the guard times we can greatly speed up the process of entering Command Mode.

Getting\_dB\_Level.bs2 is optimized to accept values, to request the RSSI dBm level, and display data.

```
' *****
' Getting_dB_Level.bs2
' Receive multiple decimal data and report dBm level
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}
' ***** Constants & PIN Declarations *****
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
#CASE BS2SX, BS2P
  T9600      CON      240
#CASE BS2PX
  T9600      CON      396
#ENDSELECT
Baud          CON      T9600

Rx            PIN      15
Tx            PIN      14
RTS           PIN      11

' ***** Variable Declarations *****
DataIn        VAR      Byte
Val1          VAR      Word
Val2          VAR      Word
' ***** XBee Configuration *****
PAUSE 500
DEBUG "Configuring XBee...",CR
PAUSE 2000                                ' Guard Time
SEROUT Tx,Baud,["+++"]                    ' Command Mode sequence
PAUSE 2000                                ' Guard Time
SEROUT Tx,Baud,["ATD6 1,GT3,CN",CR]      ' RTS enable (D6 1)
                                           ' Very low Guard Time (GT 3)
                                           ' Exit Command Mode (CN)
' ***** Main LOOP *****
PAUSE 500
DEBUG "Awaiting Delimiter and Multiple Decimal Data...",CR
```

```
DO

SERIN Rx\RTS,Baud,5,Timeout,[DataIn] ' Briefly wait for delimiter

IF DataIn = "!" THEN ' If delimiter, get data
SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val1] ' Accept first value
SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val2] ' Accept next value
' Display remotely and locally
SEROUT Tx, Baud, [CR,"Values = ", DEC Val1," ", DEC Val2,CR]
DEBUG CR,"Values = ", DEC Val1," ", DEC Val2,CR

GOSUB Get_dBm ' Retrieve RSSI level
ENDIF

GOTO Done ' Jump to done
Timeout:
DEBUG "." ' If no data, display dots
Done:
LOOP

' ***** Subroutines *****
Get_dBm:
GOSUB Empty_Buffer ' Ensure no data left in XBee
PAUSE 5 ' Short guard time
SEROUT Tx,Baud,["+++"] ' Command Mode
PAUSE 5 ' Short guard time
SEROUT Tx,Baud,["ATDB,CN",CR] ' Request dBm Level (ATDB)& Exit
' Accept returning HEX data with timeout
SERIN Rx\RTS,Baud,50,TimeOut_dB,[HEX DataIn]
SEROUT Tx,Baud,[CR,"RSSI = -",DEC DataIn,CR] ' Display remotely
DEBUG "RSSI = -",DEC DataIn,CR ' Display locally
TimeOut_dB:
RETURN

Empty_Buffer: ' Loop until no more data
SERIN Rx\RTS,Baud,5,Empty,[DataIn] ' Accept, when no data exit
GOTO Empty_Buffer ' Keep accepting until empty
Empty:
RETURN
```

### Code analysis:

- In the XBee Configuration, **GT 3 (ATGT 3)** is used to lower the guard time down to 3 ms (which we will refer to as “Fast Command Mode”).
- After accepting data, the **Get\_dBm** subroutine is executed, which branches to **Empty\_Buffer** to loop accepting data until the buffer is empty.
- **Get\_dBm** then sends the “+++” command sequence with very short delays.
- **ATDB** is sent to request the RSSI level, which the XBee returns and is accepted using **SERIN** as a hexadecimal value. It is then displayed locally and remotely.
- Note that any buffered data will be lost.
- The use of the string delimiter “!” will act also to filter out any unwanted data from being processed as well.

## 4: Programming for Data and XBee Configurations

### Testing:

- ✓ Download Getting\_dB\_Level.bs2 to the Remote node.
- ✓ For the Base node, enter several values, “!” with Enter, then several more values. After entering “!” the next two values should be accepted and displayed along with the RSSI value as shown in Figure 4-8.
- ✓ Use the X-CTU Send Packet window to make a series of numbers and !’s to be sent and test. Notice that data is lost in this instance.

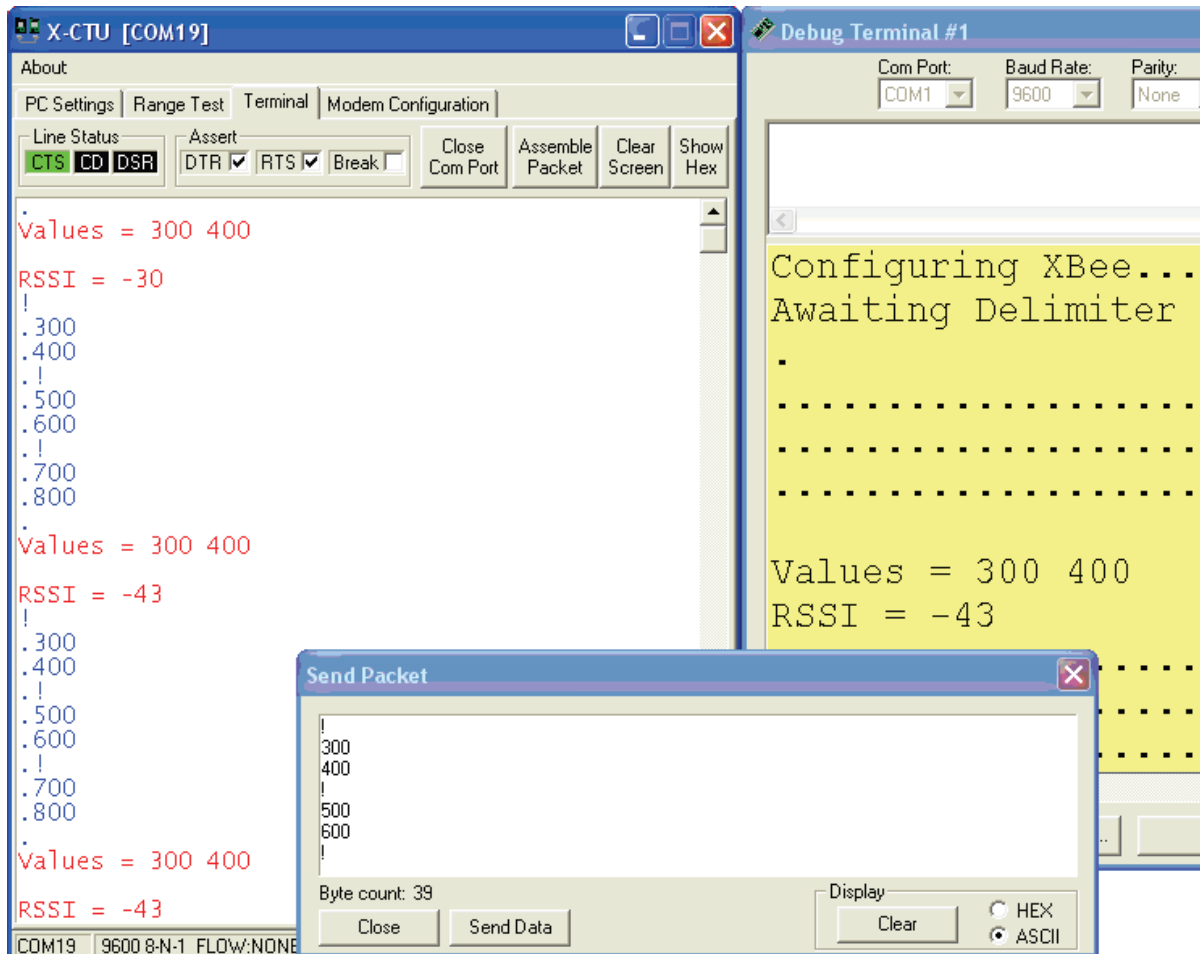


Figure 4-8: Getting dB Level Test

### General BASIC Stamp Notes of Interest

- The X-CTU terminal window is useful for sending data both as individual characters and as an assembled packet, but the BASIC Stamp Editor may also be used to interface to the XBee via USB. Simply open a new Debug Terminal and select the correct COM port, though it only supports individual character entry and not the ability to assemble a packet of data.
- Most any BASIC Stamp program that uses the Debug Terminal for display or interaction may be modified to use a remote Debug Terminal across the XBee transparent link. Replace any **DEBUG**'s with **SEROUT** structures and **DEBUGIN**'s with **SERIN** structures.
- Receiving serial data can be tricky and may require unique "tweaks" to get the data received properly. An option may be to slow down the data by using a slower data rate from the XBee. For example, using **ATBD 0** will set the baud rate to 1200 allowing the BASIC Stamp more



time to process incoming data. The best means is to use X-CTU to manually change the baud rate of the XBee and modify your code accordingly. If done solely in code you will need to start at 9600, change the XBee configuration to the new baud rate, then use the slower baud rate from then on.

- While configuring the XBee in code is nice, it can lead to issues. Let's say you add code to change the baud rate and it begins to communicate at 1200 after configuration code. When you download new code, the XBee will still be at 1200 baud and your configuration information at 9600 will not be understood; this is especially an issue if you changed the configuration! Either manually cycling power on the board to reset or using the XBee Reset line (brought LOW to reset before configuring) to return to default configurations may be needed. While the BASIC Stamp resets with a code download, the XBee does not!
- The **SERIN** instruction has a **WAIT** formatter to idle until a character is received and allows multiple values to be accepted, such as:

```
SERIN Rx\RTS,BAUD [WAIT ("!"), DEC Val1, DEC val2].
```

We found that complex structure was worse at collecting data correctly than our 3-line method, possibly due to using RTS. Also, timeouts cannot be used with the **WAIT** modified. The **SERIN** instruction has many options. Please see the BASIC Stamp Help files for more information on **SERIN**.

### **Propeller Chip**

The Propeller chip is an excellent microcontroller for handling communications. Between the Propeller's speed, multi-core processing, and available objects, serial data communications is nearly effortless.

### **Hardware & Software**

For these tests a single Propeller chip and XBee (the Remote node) will be used for communications with an XBee connected to the PC using USB and X-CTU software as the Base node as shown in Figure 4-1. Data will be manually entered using a terminal program connected for the Base node to be transmitted to the Remote node's controller and displayed to investigate data reception.

#### **Hardware:**

- (1) Propeller P8X32A and development board, using a 5 MHz external crystal oscillator
- (2) XBee modules (Ensure they are restored to default configuration)
- (1) XBee Adapter Board
- (1) XBee USB Adapter Board & Cable
- (5) 220  $\Omega$  resistors and (5) LEDs (optional)



**Alternative to XBee USB Adapter Board:** A Propeller & XBee using Serial\_Pass\_Through.spin discussed in this section or the Prop Plug as discussed in Chapter 2 may be used for X-CTU communications and configuration. These configurations cannot be used for firmware updates to the XBee.

## 4: Programming for Data and XBee Configurations

- ✓ Connect the Propeller to an XBee Adapter Board as shown in Figure 4-9. LEDs and resistors are optional but are useful.

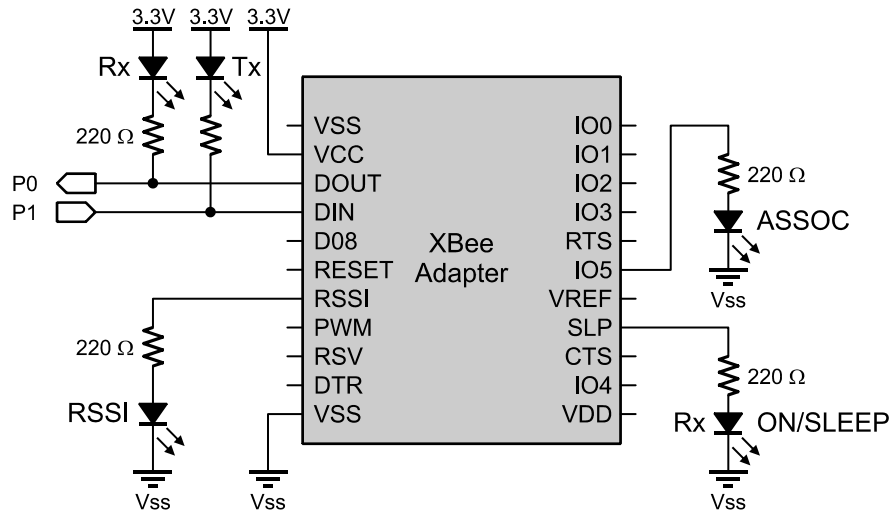


Figure 4-9: Propeller to XBee Adapter Board Interface with Optional LEDs

### Software

- Propeller Tool Software, available from the Downloads link at [www.parallax.com/propeller](http://www.parallax.com/propeller).
- Digi's X-CTU software, available from the X-CTU link at [www.parallax.com/go/xbee](http://www.parallax.com/go/xbee).

### Receiving Byte Values & Chat

The ability to use multiple cogs in multitasking allows the Propeller to accept and process data in one cog while performing operations in another cog. In this example, the Propeller is being used for passing serial data between the PC and XBee. Data from the XBee is accepted and passed to the PC in one cog, and accepted from the PC and passed to the XBee in another cog as illustrated in Figure 4-10.

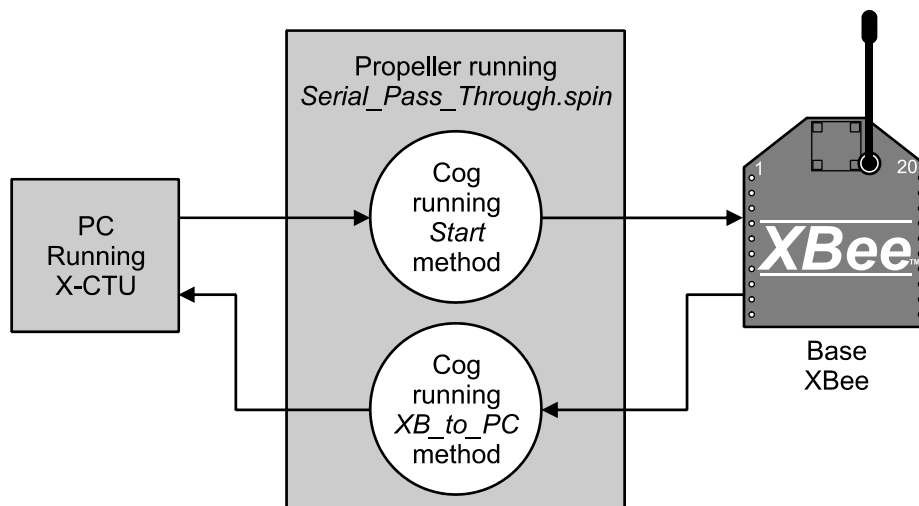


Figure 4-10: Cogs Passing Serial Data

```

{{
*****
* Serial_Pass_Through *
*****
* See end of file for terms of use. *
*****

Provides serial pass through for XBee (or other devices)
from the PC to the device via the Propeller. Baud rate
may differ between units though FullDuplexSerial can
buffer only 16 bytes.
}}

CON
_clkmode = xtall + pll16x
_xinfreq = 5_000_000

' Set pins and Baud rate for XBee comms
XB_Rx    = 0      ' XBee DOUT
XB_Tx    = 1      ' XBee DIN
XB_Baud  = 9600

' Set pins and baud rate for PC comms
PC_Rx    = 31
PC_Tx    = 30
PC_Baud  = 9600

Var
long stack[50]          ' stack space for second cog

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

```

### Analyzing the code:

- As mentioned, two cogs are used with one running the Start method, and one running the XB\_to\_PC method.
- The FullDuplexSerial.spin object is used for serial communications. This object buffers 16 bytes allowing it to receive data before being passed to the code when requested with the rx method. Using this method, execution will wait until a byte is available and returned before continuing.
- In Start, a byte from the PC is requested and passed to the XBee using XB.tx(PC.rx). This is equivalent to accepting a byte to a variable, then sending the byte:

```

DataIn := PC.Rx
XB.Tx(DataIn)

```

## 4: Programming for Data and XBee Configurations

The benefit in not storing it before transmission is the lower processing time needed, which increases the execution speed.

- In the XB\_to\_PC method, data is accepted from the XBee and passed to the PC, allowing data to flow in both directions at the same time using two cogs—full duplex communications.

### Testing:

- ✓ You may use:
  - One XBee on a USB Adapter and one connected to the Propeller running Serial\_Pass\_Through.spin.
  - OR—
  - 2 Propellers connected to XBee modules running Serial\_Pass\_Through.spin.
- ✓ Download the code to controller(s).
- ✓ Open two instances of X-CTU; one using the COM port of the Base node, and one connected to the Remote node (separate PC's may be used).
- ✓ Type in each window to send data to the other.
- ✓ Test using the Send Packet window as well, as shown in Figure 4-11.

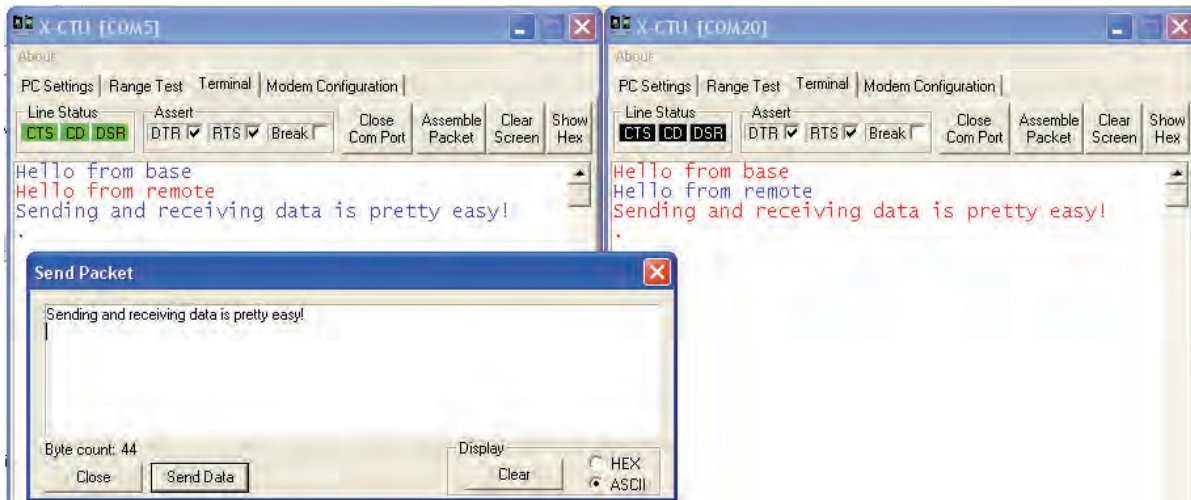


Figure 4-11: Serial Pass Through Chatting

- ✓ Using a Propeller-connected XBee, open the Modem Configuration tab and verify that you can load and save configuration settings via the Propeller.



**COM Port Use:** Only one device can use the Com Port at any time. When programming the Propeller, the X-CTU software must be disconnected by using Close Com Port. The Parallax Serial Terminal (PST) program may also be used.

**Propeller Resets:** When using the terminal window to interface to the Propeller, it is recommended that you use F11 to download instead of F10. This will ensure your code is in memory if the Propeller cycles on terminal window connections.

Note that the serial speed of the XBee and PC do not need to be the same. Different speeds may be used between each side of the Propeller. Communications with the XBee can be handled by the code up to its maximum speed.

Of course, in this example we are simply passing bytes, though code may be written to use the byte values, or as data for processing based on its value:

```
DataIn := XB.rx
If DataIn == "p"
  ' Code to be processed
```

### Debugging Back to Base Unit

Using the XBee is a great way to simply monitor your Remote node for robotics or sensor data. In this example we will use the `str` and `dec` methods of the `FullDuplexSerial` object in `Simple_Debug.spin` to send information back to the Base node for monitoring.

```
{ {
  *****
  * Simple_Debug *
  *****
  * See end of file for terms of use. *
  *****
  Demonstrates debugging to Remote terminal
}}

CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  ' Set pins and Baud rate for XBee comms
  XB_Rx    = 0    ' XBee DOUT
  XB_Tx    = 1    ' XBee DIN
  XB_Baud  = 9600
  CR       = 13   ' Carriage Return value

Var
  word stack[50]

OBJ
  XB      : "FullDuplexSerial"

Pub Start | Counter
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee

Delay(1000) ' one second delay
repeat Counter from 1 to 20 ' count up to 20

  ' Send to Base
  XB.str(string("Count is:")) ' send string
  XB.dec(Counter) ' send decimal value
  XB.Tx(CR) ' send Carriage Return

  Delay(250) ' Short delay

Pub Delay(ms) ' Delay in milliseconds
  waitcnt(clkfreq / 1000 * ms + cnt)
```

### Analyzing the code:

- The `FullDuplexSerial` object is once again used to interface to the XBee, but in this case the local interfacing to the PC is not performed.
- The value of `Counter` is incremented from 1 to 20, a string is sent to the Base using `XB.str` method, and the `Counter` value is sent using `XB.dec` method.
- The line is terminated by sending the byte value of 13 for a carriage return (CR).

## 4: Programming for Data and XBee Configurations

---

### Testing:

- ✓ Connect the Base node to the PC using the XBee USB Adapter board (or a Propeller with Serial\_Pass\_Through.spin) and the X-CTU terminal window.
- ✓ Set up the Remote node with a Propeller running Simple\_Debug.spin.
- ✓ Download and run the program on the Remote node and monitor the Base node's terminal window as shown in Figure 4-12.

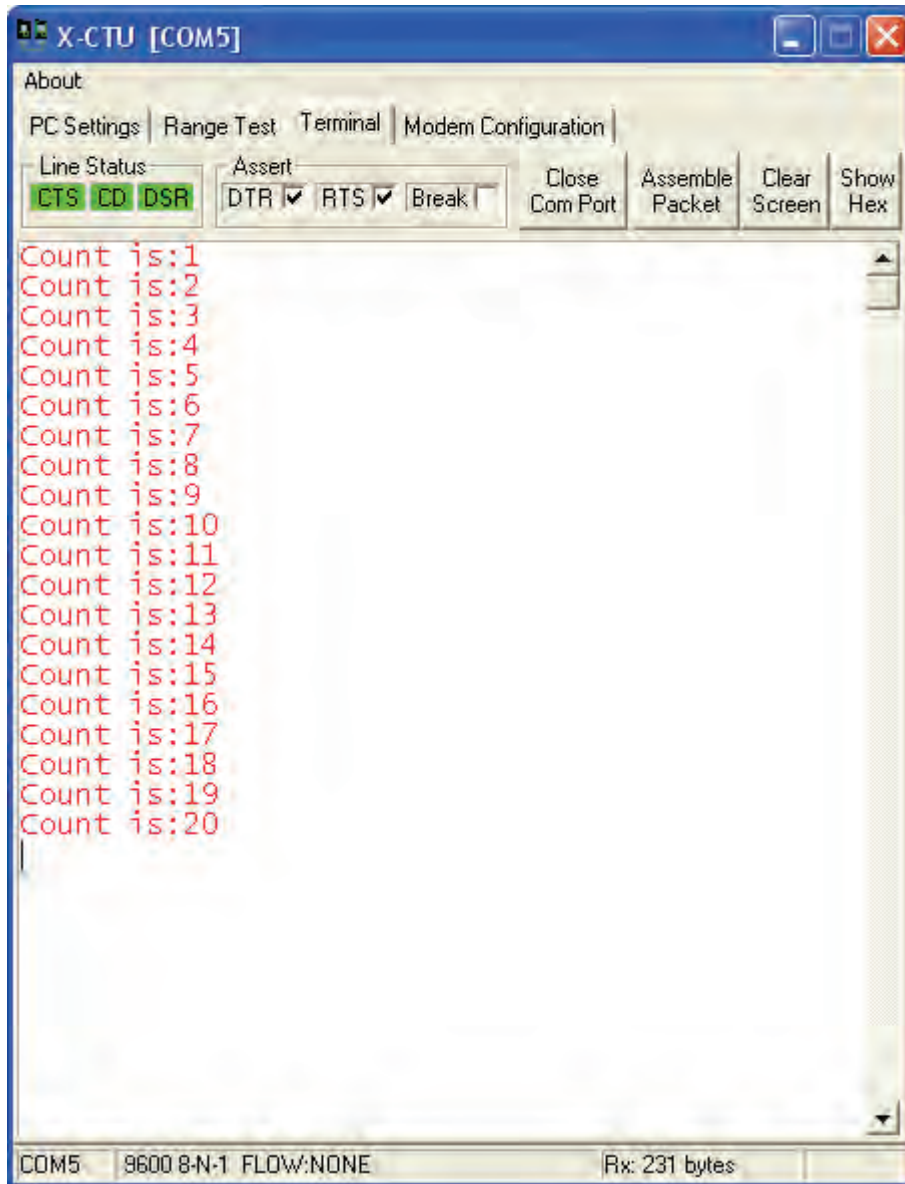


Figure 4-12: Simple Debug Monitoring

### Receiving Simple Decimal Values

In this section we will look at code to accept and return decimal values using the object Simple\_Decimal\_Receieve.spin. This code requires XBee\_Object.spin by Martin Hebel which is available from the Propeller Object Exchange (<http://obex.parallax.com>) or in the distributed files for this tutorial. XBee\_Object.spin uses FullDuplexSerial.spin, but greatly extends it with functions, many specific to XBee interfacing.



**XBee\_Object & FullDuplexSerial:** The XBee Object.spin uses FullDuplexSerial.spin for communications. The XBee Object duplicates all the methods of FullDuplexSerial, but adds specialized methods for communications with the XBee transceiver for configuration changes and data reception, The XBee Object will be used for the XBee throughout the remainder of this tutorial.

```
{ {
  *****
  * Simple_Decimal_Receive          *
  *****
  * See end of file for terms of use. *
  *****
  Demonstrates receiving and echoing decimal value
}}

CON
  _clkmode = xtall + pll16x
  _xinfreq = 5_000_000

  ' Set pins and Baud rate for XBee comms
  XB_Rx    = 0      ' XBee DOUT
  XB_Tx    = 1      ' XBee DIN
  XB_Baud  = 9600

OBJ
  XB      : "XBee_Object"

Pub Start | Value
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee

XB.Delay(1000)                       ' One second delay

XB.str(string("Awaiting Data...")) ' Notify Base
XB.CR

Repeat
  Value := XB.RxDec                   ' Wait for and accept decimal value
  XB.Dec(Value)                       ' Send value back to Base
  XB.CR                               ' Send carriage return
```

### Analyzing the Code:

- In this example, only one terminal window is needed—the Base node using an XBee USB Adapter or Propeller running Serial\_Pass\_Through.spin for PC communications.
- XBee\_Object.spin is used for XBee communications and interfacing, providing additional functionality over FullDuplexSerial.spin.
- In the Start method, the Remote node informs the Base node's user it is awaiting data and waits for a decimal value to arrive using the XB.RxDec method. The decimal value needs to be terminated with a carriage return (enter key) or by a comma to separate values.
- Once a decimal value is received and stored in Value, it is sent back to the Base node as a decimal value with XB.Dec(Value) along with a carriage return.
- Note that XBee\_Object.spin has methods for both .Delay and .CR to minimize coding for normal needs.

### Testing:

- ✓ Set up the Base node of an XBee connected to the PC using the XBee USB adapter (or Propeller with Serial\_Pass\_Through.spin) and X-CTU terminal Window.

## 4: Programming for Data and XBee Configurations

- ✓ Set up the Remote node with a Propeller running Simple\_Decimal\_Receive.spin.
- ✓ Test sending decimal value in the X-CTU terminal window of the Base node by entering values and using the Enter key.
- ✓ Use the Send Packet window using carriage returns between values as shown in Figure 4-13. Note that no data is lost, due to the speed of the Propeller and the buffering of the drivers.

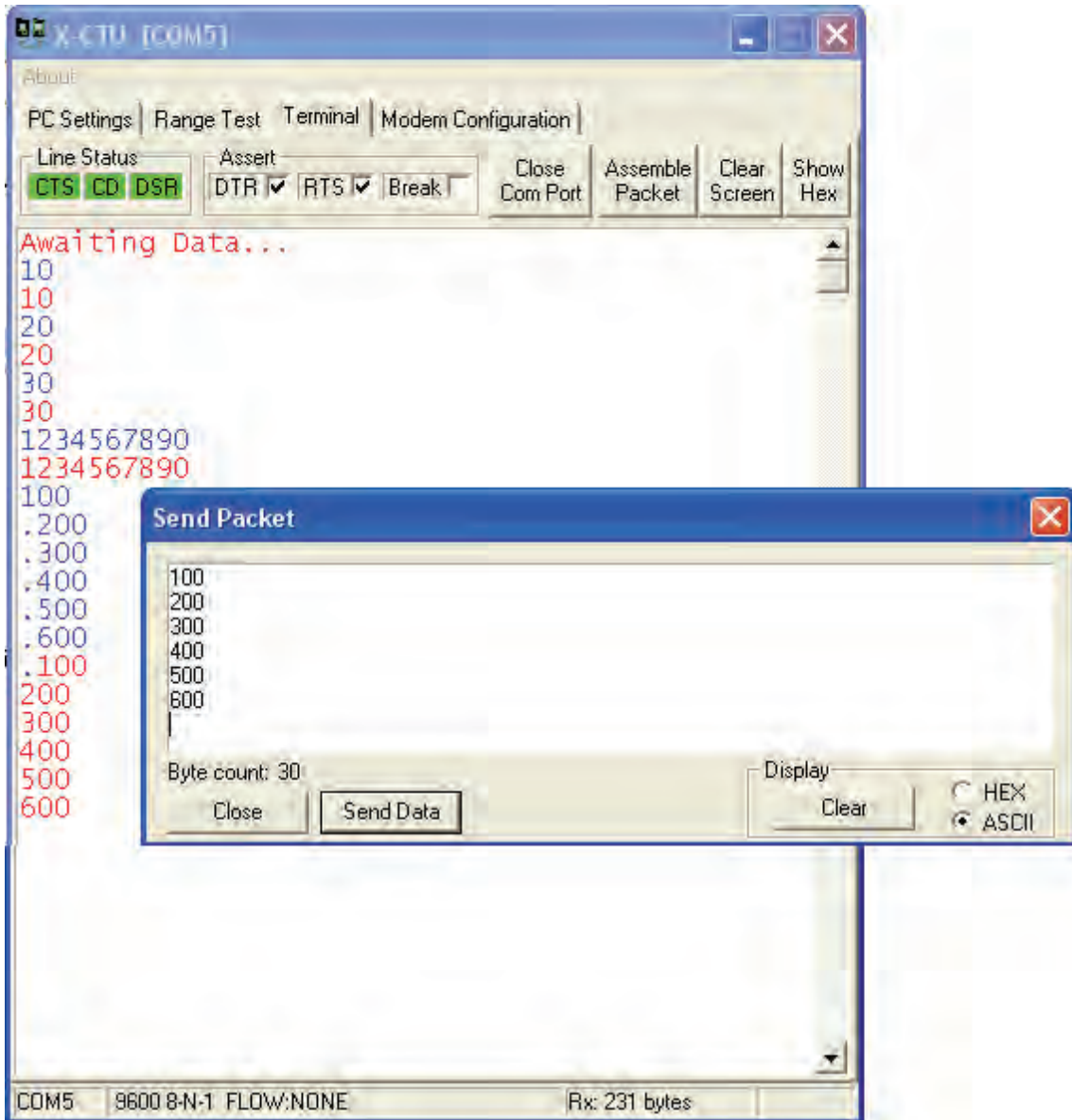


Figure 4-13: Receiving Simple Decimal Values

### Receiving Multiple Decimal Values with Delimiter

In many instances you may need to send multiple values to the Propeller for various operations, such as setting the speed of both motors on a robot. While the Propeller is great at buffering and receiving the data, it may require some extra code to ensure the data is sent in the correct sequence, just as with the BASIC Stamp.



If we send 2 values for val1 and val2, such as:

```
10 & 20
10 & 30
10 & 50
```

...on reception, the Propeller gets out of sequence so that the data it receives is:

```
20 & 10
30 & 10
```

Because it perhaps missed one, the sequence of how it is accepting data does not match the sequence we intended. Through the use of a start-of-string identifier or delimiter we can help ensure the data is collected starting at the correct value in the buffer. The same could be done for bytes but it can cause issues since a byte value can be ANY value typically, 0 to 255. A unique value may not be able to be identified for our communications. Using DEC values, only 0–9 are valid characters so anything outside of that range would be unique from our data. The program `Multiple_Decimal_Receive.spin` illustrates this as well as some other features.

```
{ {
*****
* Multiple_Decimal_Receive *
*****
* See end of file for terms of use. *
*****
Demonstrates receiving multiple decimal
value with start delimiter
} }

CON
_clkmode = xtal1 + pll16x
_xinfreq = 5_000_000

' Set pins and Baud rate for XBee comms
XB_Rx    = 0      ' XBee DOUT
XB_Tx    = 1      ' XBee DIN
XB_Baud  = 9600

' Carriage return value
CR = 13

OBJ
XB      : "XBee_Object"

Pub Start | DataIn, Val1, Val2
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
XB.Delay(1000)                      ' One second delay
XB.str(string("Awaiting Data...")) ' Notify Base
XB.CR

Repeat
DataIn := XB.RxTime(100)             ' Wait for byte with timeout
If DataIn == "!"                    ' Check if delimiter
Val1 := XB.RxDecTime(3000)          ' Wait for 1st decimal value with timeout
Val2 := XB.RxDecTime(3000)          ' Wait for next decimal value with timeout
If Val2 <> -1                        ' If value not received value is -1
XB.CR
XB.Str(string(CR, "Value 1 = "))    ' Display remotely with string
XB.Dec(Val1)                       ' Decimal value
XB.Str(string(CR, "Value 2 = "))    ' Display remotely
XB.Dec(Val2)                       ' Decimal value
XB.CR
Else
XB.Tx(".")                          ' Send dot to show actively waiting
```

## 4: Programming for Data and XBee Configurations



**Value Delimiter:** The XBee\_Object can use carriage returns or commas as delimiters between values, so data may be sent as 30,40 instead of using the Enter key between each.

### Analyzing the code:

- XBee\_Object.spin is used for communications and interfacing.
- In this example, `DataIn := XB.RxTime(100)` reads a byte from the buffer, but only waits 100 ms for data arrive before continuing processing—a receive with timeout. If no data is received, the value in `DataIn` will be -1.
- Upon reception or timeout, the value of `DataIn` is checked to see if it is “!” (the string delimiter). If it is, two decimal values, with 3 second timeouts each, are accepted into `Val1` and `Val2`.
- Identifying strings and decimal values are sent back to the Base unit for display.
- Upon timeout with a value of -1, a dot is sent to the Remote terminal to indicate processing is continuing.

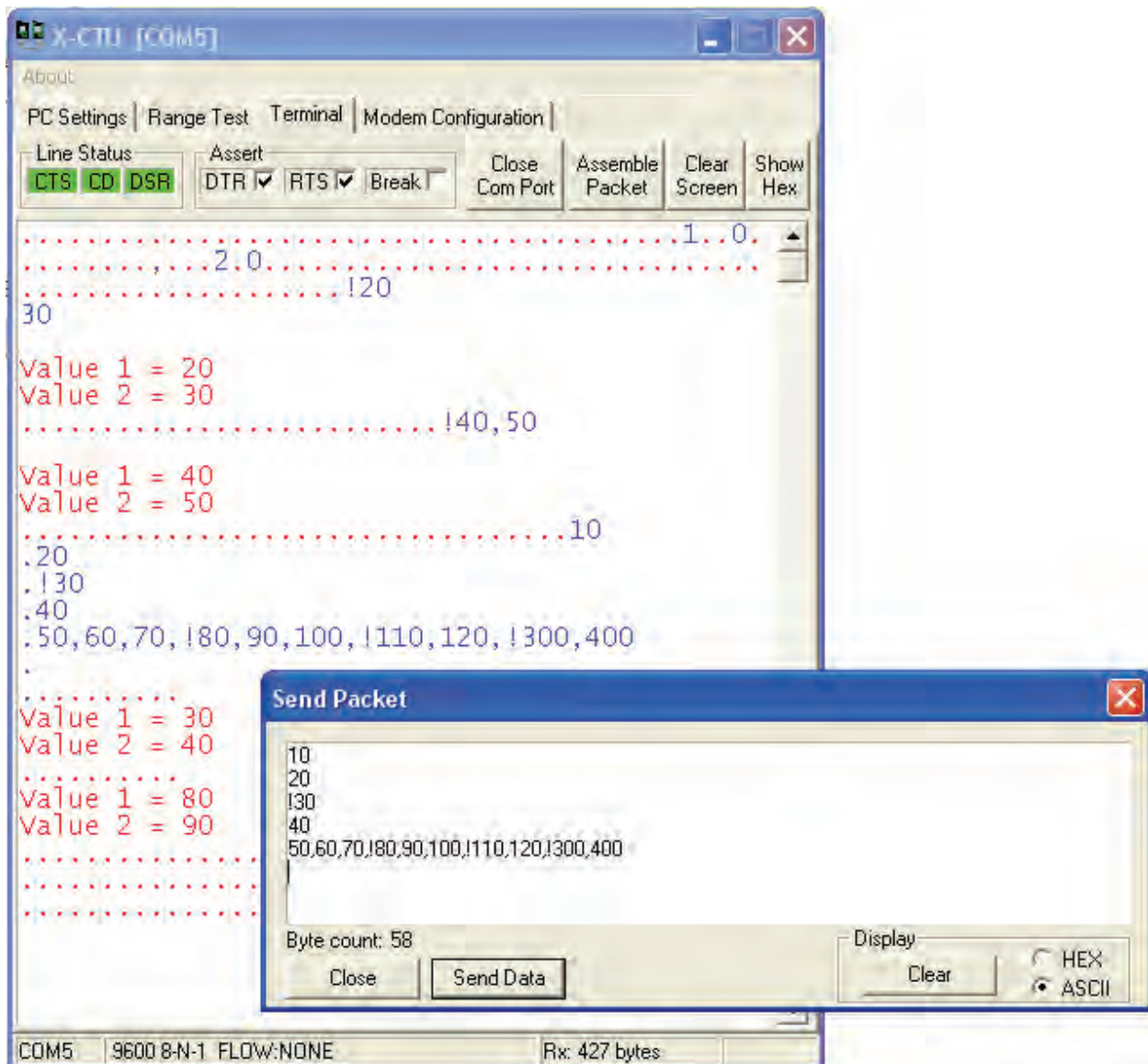


Figure 4-14: Multiple Decimal Receive Testing

### Testing:

- ✓ Set up the Base node XBee connected to the PC via the XBee USB Adapter board (or Propeller with Serial\_Pass\_Through.spin) and X-CTU terminal window.
- ✓ Set up the Remote node with the Propeller running Multiple\_Decimal\_Receive.spin.
- ✓ Test by sending decimal values in the terminal window by entering some values, using “!” and then entering a few more values—DO NOT press Enter after “!”. Test using commas as well.
- ✓ Use the Send Packet window to build a packet of values and using the “!” delimiter. Again, test the use of carriage returns and commas between values as shown in Figure 4-14.

Notice that there was some lost data—the !300, 400 was not displayed. The buffer of the object may have been exceeded while processing. When dealing with serial communications, testing and work-arounds are always needed, but we have a high probability of getting simple packets through successfully at normal update speeds.

### Setting and Reading XBee Configurations

Just as we can configure the XBee through the serial terminal, the Propeller can send and receive data for configuration changes and reading values in the same fashion. Config\_Getting\_dB\_Level.spin demonstrates a simple configuration of the XBee, requesting and accepting dB level for RSSI. Since we don't need to set RTS with the Propeller and we are not ready for address changes yet, we will use a configuration command to turn off the association indicator on the XBee to illustrate sending AT commands.

```
{  
  *****  
  * Config_Getting_dB_Level *  
  *****  
  * See end of file for terms of use. *  
  *****  
  Demonstrates receiving multiple decimal  
  value with start delimiter  
}  
  
CON  
  _clkmode = xtal1 + pll16x  
  _xinfreq = 5_000_000  
  
  ' Set pins and Baud rate for XBee comms  
  XB_Rx    = 0      ' XBee DOUT  
  XB_Tx    = 1      ' XBee DIN  
  XB_Baud  = 9600  
  
  ' Carriage return value  
  CR = 13  
  
OBJ  
  XB      : "XBee_Object"  
  
Pub Start | DataIn, Val1, Val2  
XB.start(XB_Rx, XB_Tx, 0, XB_Baud)      ' Initialize comms for XBee  
XB.Delay(1000)                          ' One second delay  
  
' Configure XBee module  
XB.Str(String("Configuring XBee..."),13)  
XB.AT_Init                               ' Configure for fast AT Command Mode  
  
XB.AT_Config(string("ATD5 4"))          ' Send AT command turn off Association LED
```

## 4: Programming for Data and XBee Configurations

```
XB.str(string("Awaiting Data..."))      ' Notify Base
XB.CR

Repeat
  DataIn := XB.RxTime(100)              ' Wait for byte with timeout
  If DataIn == "!"                      ' Check if delimiter
    Val1 := XB.RxDecTime(3000)          ' Wait for 1st value with timeout
    Val2 := XB.RxDecTime(3000)          ' Wait for next value with timeout
    If Val2 <> -1                        ' If value not received value is -1
      XB.CR
      XB.Str(string(CR,"Value 1 = "))    ' Display remotely with string
      XB.Dec(Val1)                      ' Decimal value
      XB.Str(string(CR,"Value 2 = "))    ' Display remotely
      XB.Dec(Val2)                      ' Decimal value

      XB.RxFlush                        ' Clear buffer
      XB.AT_Config(string("ATDB"))      ' Request dB Level
      DataIn := XB.RxHexTime(200)       ' Accept returning hex value
      XB.Str(string(13,"dB level = "))   ' Display remotely
      XB.Dec(-DataIn)                   ' Value as negative decimal
      XB.CR
    Else
      XB.Tx(".")                        ' Send dot to show actively waiting
```

### Code Analysis:

- Just as when we manually configure the XBee, it requires a 2-second delay since last data sent, the “+++” sequence, and a 2-second delay. This delay may be reduced by setting the guard time lower (**ATGT**). The XBee object has a method called `AT_Init` which will perform this sequence while lowering guard time to allow fast command configurations. Looking at the `XBee_Object` code, we can see the actions taken. The `rxFlush` method is used to clear data from the object buffer along with the OK's that are returned.

```
Pub AT_Init
{{
  Configure for low guard time for AT Mode.
  Requires 5 seconds. Required if AT_Config used.
}}

  delay(3000)
  str(string("+++"))
  delay(2000)
  rxflush
  str(string("ATGT 3,CN"))
  tx(13)
  delay(500)
  rxFlush
```

- After `AT_Config`, the ASSOC indicator LED on the XBee Adapter (if connected) is set to no longer indicate association using `XB.AT_Config(string("ATD5 4"))`. Using fast command times, the configuration of the XBee is quickly updated for D5 to be 4, setting the DIO5 output low.
- After receiving the delimiter and 2 decimal values, the code requests and shows the dB level. `XB.AT_Config(string("ATDB"))` is used to place the XBee into Configuration Mode and send the **ATDB** command (it also sends the **CN** to exit Command Mode). The returned hexadecimal value is returned and saved using :

```
DataIn := XB.RxHexTime(200)
```

## 4: Programming for Data and XBee Configurations

- The dBm level is displayed by sending it back to the Base unit.
- Note that `RxFlush` is used to clear out the buffer when using the Command Mode to prevent buffered data from interfering. This means that not all of our burst data will be processed.

### Testing:

- ✓ Base node XBee connected to PC using USB adapter (or a Propeller with `Serial_Pass_Through.spin`) and X-CTU terminal window.
- ✓ Remote node XBee with Propeller running `Config_Getting_dB_Level.spin`.
- ✓ Test using the `!` delimiter and values. Note that the dBm level is returned after a good set of data and remaining data is lost, as shown in Figure 4-15.

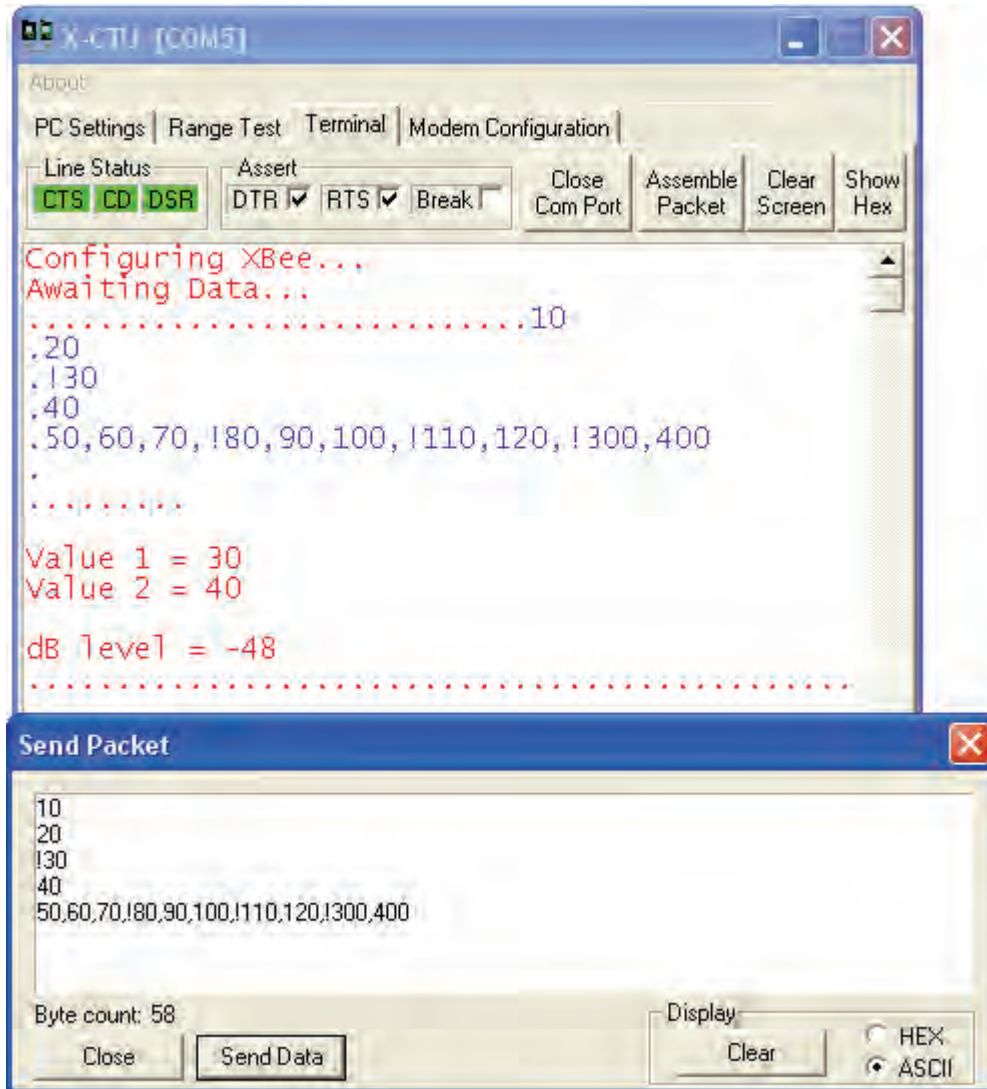


Figure 4-15: Configuring and Reading dBm Level Test



**Config with Variables:** Using `AT_Config`, a complete string is required. If a variable value is needed, use `AT_ConfigVal` method where a string and value are passed:

```
XB.AT_ConfigVal(string("ATDL"), DL_Val)
```

**Monitor LEDs:** When the code starts and goes through the configuration sequence, watching the Tx & Rx LEDs blink back and forth is a great indicator that it the XBee is accepting your Command Mode functions.

## 4: Programming for Data and XBee Configurations

---

While configuring the XBee in code is nice, it can lead to issues. Let's say you add code to change the baud rate and it begins to communicate at 1200 baud after the configuration code. When you download new code, the XBee will still be at 1200 baud and your configuration information at 9600 will not be understood. This is especially an issue if you changed the configuration!. Either manually cycling power on the board to reset or using the XBee Reset line (brought LOW to reset before configuring) to return to default configurations may be needed. While the Propeller resets with a code download, the XBee does not!

### **Summary**

The BASIC Stamp and Propeller can both be programmed to receive data, bytes or decimal values. With the BASIC Stamp, the use of RTS can help ensure data is not missed. XBee configuration may be performed by both controllers using the XBee's AT Command Mode to change settings or to request data such as dBm level for RSSI. While an XBee and terminal window were used, in upcoming chapters two controllers will be used to interact with one another using XBee modules.

## 5: Network Topologies & Control Strategies

While the XBee does a great job of handling the data link layer communications—getting data between the nodes—it is up to the microcontroller coding to handle the communicating meaningfully across the network at the application layer. Determining what data is sent, what will be done with it, what node it is addressed to, and ensuring devices are ready to accept and use the data, are all up to the programmer. The programmer must employ some networking strategy to help ensure application data is sent and received properly. Depending on the complexity of the network and the data, different topologies and schemes may be employed to send the data and interact between nodes. Networking topologies include point-to-point and point-to-multipoint.

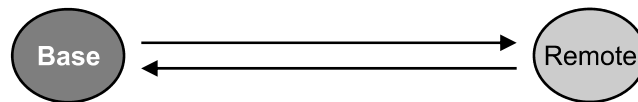
In either topology, but especially in point-to-multipoint, control of the network must be performed to ensure receivers are ready to accept data, the returning node knows where data is to be sent and multiple nodes may not be able to send at once to a common destination. The XBee handles moving the data between nodes, but the application of the network may require some scheme on the programming side to ensure a controlled flow of data for operation, such as polling or scheduling.

In this chapter we will explore some networking strategies and use code to pass data between nodes for control and acquiring data. We will keep the code and hardware fairly simple, but as always, use your knowledge and desires to adapt these to your own needs.

### Network Topology Examples

#### Point-to-Point Network

In a point-to-point network, data is sent between two nodes as shown in Figure 5-1. This is the simplest form of communications to implement and doesn't require any configuration changes to the XBee. Both devices may be left on address 0 (**MY** = 0 and **DL** = 0), their default configuration. Data sent from one unit to address 0 is accepted by and, if data is to be returned, is sent back to address 0.



**Figure 5-1: Point-to-Point Communication Between Two Nodes**

Should another pair of XBee modules wish to communicate in the same area, the address of those nodes may be changed, such as to 1 by setting their **MY** and **DL** addresses, to allow point-to-point communications between them. They may also be placed in different PANs by changing the **ID**, or different frequency channels using **CH**. A good example of point-to-point communication with multiple channels would be a set of radio-controlled cars, each operating on a different set of frequencies.

#### Point-to-Multipoint Networks

In a point-to-multipoint network, a node can communicate with multiple nodes on the network. This requires each node having a unique address on the network.

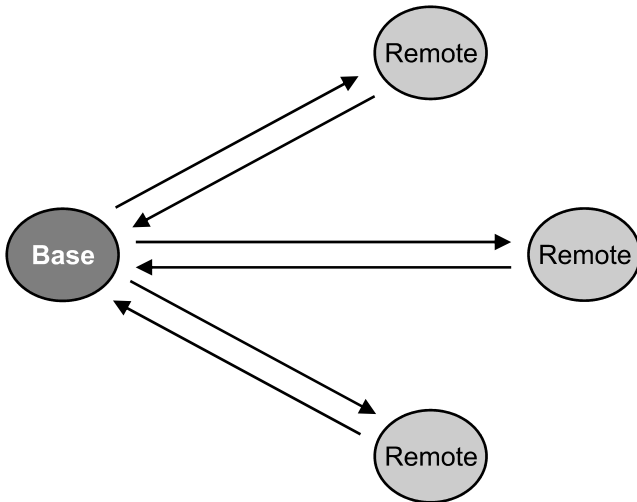
## 5: Network Topologies & Control Strategies

---

### Simple Point-to-Multipoint Network

In a simple network as shown in Figure 5-2, all traffic is *managed* by a central node (a Master, Base or Coordinator) that addresses a Remote node, sends data to that node, and data from the Remote node is sent back to the Base node.

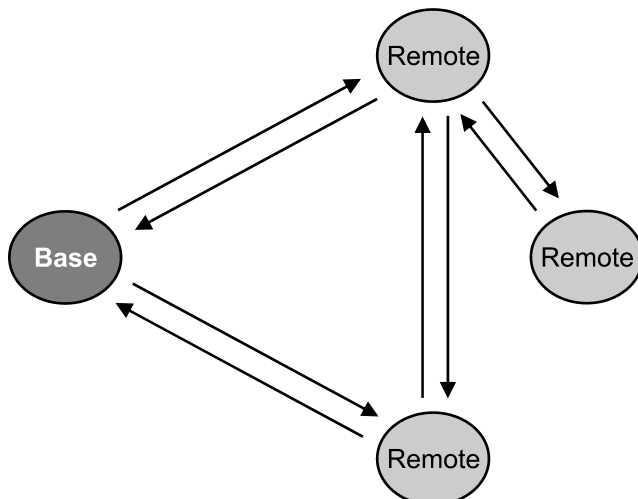
With the XBee, the software at the Base (master) can change the destination address (**DL**) to send to a particular Remote node. Data from a Remote is always sent to the Base's address. If needed, the Base can manage control between nodes, such as receiving a sensor reading from one Remote node and using it to control an actuator on another Remote node. An example could be wirelessly monitoring irrigation needs using one Remote node, and controlling irrigation pumps controlled by another Remote node. The base would centrally monitor the need for irrigation and control the system accordingly.



**Figure 5-2: Point-to-Multipoint Base Node Communicating Individually to Multiple Remote Nodes**

### Complex Point-to-Multipoint Network

In more complex networks, data from any node may send data to any other node, as shown in Figure 5-3. In order for our program to respond with data properly, the receiving node must know the address to return data. The address data may be passed as part of the application data sent or extracted from the frame when using API Mode (Chapter 6).

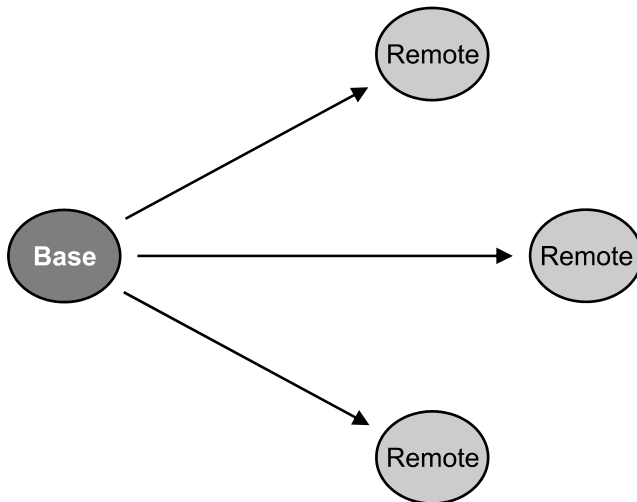


**Figure 5-3: Point-to-Multipoint All Nodes Able to Talk to Other Nodes**



### **Broadcast Point-to-Multipoint Network**

Another form of point-to-multipoint is the network broadcast. Data is sent from one node to all nodes on the network. With the XBee, a **DL** of FFFF is used to send data to all nodes. On the data link side, XBee communications using the broadcast address are NOT acknowledged. Data is simply sent and assumed to have reached all destinations. Control actions may work well with a broadcast, such as energizing a remote LED. Requesting a value be returned, such as a remote sensor reading, is problematic since all nodes will attempt to respond and the processor must be fast enough to process all incoming data.



**Figure 5-4: Point-to-Multipoint Broadcasting Data to All Nodes**

### **Polling Strategy**

In moving data in our network, some control of which node talks when may be required to ensure the data from multiple sources does not cause confusion. In a polling strategy, a central Base, Coordinator or Master is in charge of controlling communications by always talking first. Remote nodes, or slaves, are not allowed to send data until polled by the Base. The Base may send out updates for the Remote nodes or request data from them ensuring a controlled use of the network and data flow. The Base node goes through a list or range of Remote node addresses, communicating with each in turn, thereby maintaining control of communications. Note that the Remote nodes must remain alert and ready to receive data from the Base. The USB protocol uses a polling strategy—the host PC continually polls the USB devices.

### **Scheduling Strategy**

A scheduling strategy may be employed where instead of the Base node controlling communication and remote nodes having to remain alert, the base may be the passive node waiting for scheduled updates from the Remote nodes. The Remote node may perform some task, go into a low-power sleep mode, and wake to send an update to the Base and receive updates. Scheduling allows Remote nodes to sleep or spend time processing, then to periodically send an update to the Base and receive new updates before going back to sleep or doing its business of controlling its system. It also allows a Remote node with urgent data (intruder alert!) to be sent immediately without waiting for its turn to be polled.

With this strategy, since the Base node does not initiate the Remote node's transmission, it needs to have some method of determining the source of the received message. In addition to the Remote node's data, it also needs the Remote's address so that it may respond to send updates to the active

## 5: Network Topologies & Control Strategies

Remote node. There is a problem in that if two or more nodes try to send data at once, the based may not be robust enough to process data coming in from multiple nodes at once.

### BASIC Stamp Examples

With the BASIC Stamp, we will explore a variety of strategies to move data between devices. Due to the BASIC Stamp module's serial communications abilities, careful control of data is required. We will use basic, common hardware to illustrate principles that may aid you in development of your own systems.

#### Hardware

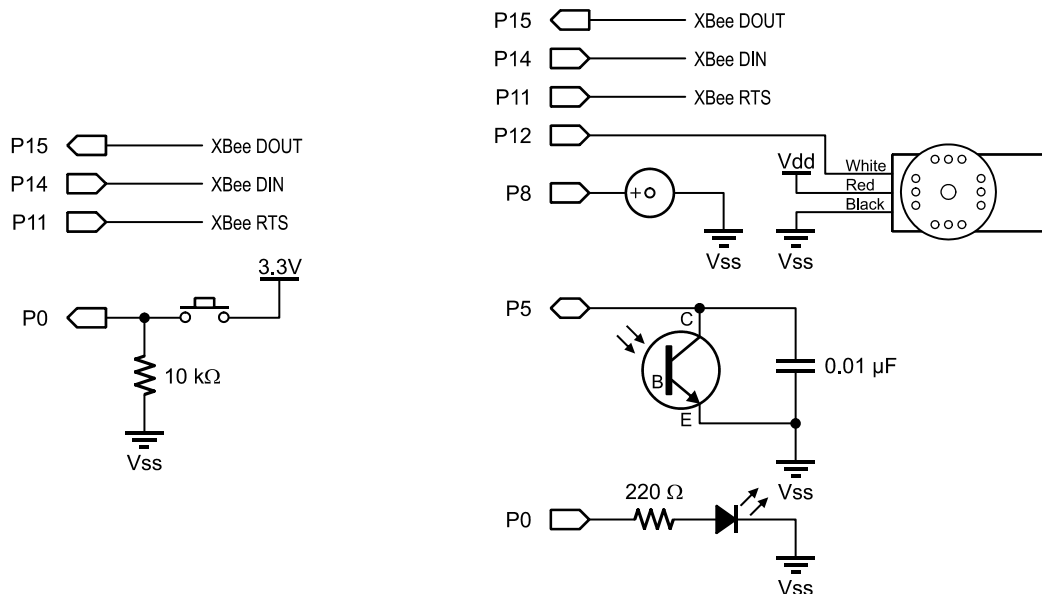
- |  |   |
|--|---|
| (2) BASIC Stamp development boards     | (1) Phototransistor                                   |
| (2) XBee SIP or 5V/3.3V Adapter boards | (1) 0.01 $\mu$ F capacitor                            |
| (2) XBee modules with default settings | (1) 10 k $\Omega$ Resistor                            |
| (1) Pushbutton                         | (1) 220 $\Omega$ Resistor                             |
| (1) LED                                |   |
| (1) Buzzer (piezospeaker)              | Optional: XBee USB Adapter and additional XBee module |
| (1) Servo                              |   |

!

**Board and Power!**

Use an appropriate BASIC Stamp XBee Adapter Board with supplied power per Chapter 2.

- ✓ Assemble the hardware as shown in Figure 5-5 for the Base node and at least one Remote node (multiple Remote nodes may be constructed for testing multi-node communications). The Base will use the Debug Terminal for control and notifications in most examples.



Base node BASIC Stamp and XBee module

Remote node BASIC Stamp and XBee module

**Figure 5-5: BASIC Stamp Base Node and Remote Node Schematics**

## Using Point-to-Point for Pushbutton Control

In this example a pushbutton on the Base node is used to control the Remote node's buzzer and LED. The default settings of the XBee are used to send data between only two nodes, both on address 0.

As the pushbutton on the Base node is pressed, the value of the variable **Freq** is incremented by 500 and sent as a decimal value to the Remote node, increasing the pitch of the buzzer. If the value of **Freq** exceeds 5000, it is reset to 500.

```
' *****
' Simple_Control_Base.bs2
' Sends changing frequency when button pressed
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
    T9600      CON      84
#CASE BS2SX, BS2P
    T9600      CON      240
#CASE BS2PX
    T9600      CON      396
#ENDSELECT

' ***** Variables, Constants and Pins
Baud          CON      T9600  ' Set baud rate
Rx            PIN      15     ' XBee DOUT
Tx            PIN      14     ' XBee DIN
PB            PIN      0      ' Pushbutton
Freq          VAR      Word

' ***** Main Loop
DO
  IF PB = 1 THEN                ' If button pressed...
    Freq = Freq + 500           ' Increment Frequency
    IF Freq > 5000 THEN Freq = 500 ' Limit to 500 to 5000
    SEROUT Tx, Baud,[DEC Freq,CR] ' Send Frequency as decimal
    PAUSE 500                   ' Short delay
  ENDIF
LOOP
```

On the Remote node, the code waits until a decimal value is received, accepts the value, lights the LED, sounds the buzzer at the frequency received, and finally turns off the LED to wait for another decimal value. Note that in this example no flow control (RTS) is used so the BASIC Stamp must be ready to accept incoming data.

```
' *****
' Simple_Control_Remote.bs2
' Receives decimal value to control buzzer and LED
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}
```

## 5: Network Topologies & Control Strategies

---

```
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
#CASE BS2SX, BS2P
  T9600      CON      240
#CASE BS2PX
  T9600      CON      396
#ENDSELECT

' ***** Variables, Constants and Pins
Baud          CON      T9600 ' Set Baud rate

Rx            PIN      15    ' XBee DOUT
Tx            PIN      14    ' XBee DIN
Led           PIN      0
Buzzer        PIN      8
Freq          VAR      Word

' ***** Main Loop
DO
  SERIN Rx, Baud, [DEC Freq]    ' Wait for decimal and accept
  HIGH LED                    ' Turn on LED
  FREQOUT Buzzer, 200, Freq     ' Sound tone
  LOW LED                      ' Turn off LED
LOOP
```

### Testing:

- ✓ Download Simple\_Control\_Base.bs2 to your Base node's BASIC Stamp.
- ✓ Download Simple\_Control\_Remote.bs2 to your Remote node's BASIC Stamp.
- ✓ Press and hold the Base node's pushbutton to test.



#### Optional Testing and Monitoring

If you have an XBee at address 0 on an XBee USB Adapter Board and are using X-CTU, you can enter values to be sent to the Remote node or simply monitor communications passing between nodes.

### Point-to-Multipoint — Manual Polling of Remote Nodes

With manual polling, the user will interact with the Base node to control and monitor the Remote nodes. The Base node prompts the user for information to control the Remote node's LED, buzzer, servo or to request the reading of the phototransistor. The first step is to enter the address of the Remote node to be controlled. It illustrates XBee configurations, flow control, node addressing and application acknowledgements.


In the code, the XBee is configured for fast Command Mode by setting the guard time low (**ATGT**). By setting guard time low, the XBee can be reconfigured in milliseconds instead of requiring several seconds. The code requests from the user the address of the node to control. This is based on constants in the Remote node's code for its address. In testing, only a single Remote may be used, but you are free to set up as many as you desire.

After accepting the Remote node's address, the Base node's program requests what action to perform: whether to control the LED, the buzzer, the servo or to get the phototransistor reading. For control, the value is requested from the user. The data is sent as an action identifier (L, B, S or R) plus a decimal value for control items, such as for servo control:

```
SEROUT Tx,Baud,["S",CR,CR]           ' Send S
SEROUT Tx,Baud,[DEC DataOut,CR,CR]   ' Send Data
GOSUB CheckAck
```

The program waits briefly for an acknowledgement ("1") from the Remote or the Remote node's value of the phototransistor circuit and repeats.

By setting the guard time low, the BASIC Stamp can configure the XBee within 20 milliseconds or so instead of requiring 5 seconds to update the destination address (**DL**). Note that this code does not use RTS flow control – no data will be buffered. Since the Base is controlling the flow of data (Remote's do not send data without being contacted), the Base can be prepared to accept data. Also, the OK's sent from the XBee during configuration are not buffered—they are sent to the BASIC Stamp which does not accept or use them because of how communications are timed.



**Important Note for BS2e, BS2sx, BS2p, BS2pe, and BS2px Users**

The following example programs use conditional compilation to ensure the correct baud rate for your BASIC Stamp model, for communication proof-of-concept. However, other timing-sensitive commands within these programs assume BS2 use; adjustments may be necessary for use with other BASIC Stamp models. Specifically, you may need to make adjustments where (**RCTIME**, **PULSOUT**, and **FREQOUT** are used.

For tips on how to modify BS2 code to work with other BASIC Stamp models, see the BASIC Stamp Editor's Help file (Editor version 2.5 or higher). Under the PBASIC Language Reference section, see the "Adapt BS2 Code to Other Models" article.

```
' *****
' Manual_Polling_Base.bs2
' This program:
'   - Configures XBee for fast AT Command Mode
'   - Using DEBUG Window, User can control Remote
'     L-LED, B-Buzzer, S-Servo or R-Read Remote sensor
'   - Sets address and data to selected node address
'   - Accepts an acknowledgement value
'   - Requires 802.15.4 XBee (Series 1)
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
#CASE BS2SX, BS2P
  T9600      CON      240
#CASE BS2PX
  T9600      CON      396
#ENDSELECT

' ***** Variables, Constants and Pins
Baud          CON      T9600  ' Set Baud rate

Rx            PIN      15      ' XBee DOUT
Tx            PIN      14      ' XBee DIN
```

## 5: Network Topologies & Control Strategies

---

```
DataOut      VAR      Word      ' Frequency to send
DL_Addr      VAR      Word      ' Destination address for data
DataIn       VAR      Byte      ' General variable for data
Light        VAR      Word      ' Returned light level

' ***** Configure XBee in AT Command Mode
PAUSE 500
DEBUG CLS,"Configuring XBee..."

PAUSE 3000                                ' Guard time
SEROUT Tx,Baud,["+++"]                    ' Command Mode Sequence
PAUSE 2000                                ' Guard time
SEROUT Tx,Baud,["ATGT 3,MY 0",CR]         ' Set low guard time and Base address
SEROUT TX,Baud,["ATCN",CR]                ' Exit Command Mode

' ***** Main Loop
DO
  ' Request address and action in DEBUG Window
  DEBUG CLS,"Enter Node Address in Hex (1-FFFF):"
  DEBUGIN HEX DL_Addr                      ' Accept address in Hex
  GOSUB Config_XBee                        ' Set DL address of XBee

  DEBUG CR,"Choose Action:",CR,
    "S - Set Servo Position",CR,
    "L - Set LED State",CR,
    "B - Set Buzzer Frequency",CR,
    "R - Read Light Level",CR,
    "? "
  DEBUGIN DataIn                          ' Accept choice

  ' If Servo Control, get value and send
  SELECT DataIn
  CASE "S", "s"
    DEBUG CR,"Enter Servo Position (500-1000):"
    DEBUGIN DEC DataOut                    ' Accept user data
    DEBUG "Sending Data!",CR
    SEROUT Tx,Baud,["S",CR,CR]             ' Send S
    SEROUT Tx,Baud,[DEC DataOut,CR,CR]     ' Send Data
    GOSUB CheckAck                         ' Get acknowledgement
    GOTO Done

  ' LED control, get state and send
  CASE "L", "l"
    DEBUG CR,"Enter LED State (0/1):"
    DEBUGIN DEC DataOut                    ' Accept user data
    DEBUG "Sending Data!",CR
    SEROUT Tx,Baud,["L",CR,CR]             ' Send L
    SEROUT Tx,Baud,[DEC DataOut,CR,CR]     ' Send LED state
    GOSUB CheckAck                         ' Get Acknowledgement
    GOTO Done

  ' Buzzer control, get value and send
  CASE "B", "b"
    DEBUG CR,"Enter Buzzer Frequency:"
```

```

    DEBUGIN DEC DataOut                ' Accept user data
    DEBUG "Sending Data!",CR
    SEROUT Tx,Baud,["B",CR,CR]        ' Send B
    SEROUT Tx,Baud,[DEC DataOut,CR,CR] ' Send Buzzer Frequency
    GOSUB CheckAck                    ' Get Acknowledgement
    GOTO Done

' Get reading from Remote sensor
CASE "R","r"
    DEBUG CR,"Requesting reading...",CR
    SEROUT Tx,Baud,["R",CR,CR]        ' Send R
    SERIN Rx,Baud,1000,Timeout,[DEC Light] ' Accept returning data
    DEBUG "Light level = ", DEC light,CR ' Display
    GOTO Done
ENDSELECT
Timeout:
    DEBUG "No data received",CR
Done:
    PAUSE 2000
LOOP

Config_XBee:
' Configure XBee for destination node address
PAUSE 10                               ' Short guard time
SEROUT Tx,Baud,["+++"]                 ' Command Mode sequence
PAUSE 10                               ' Short guard time
SEROUT TX,Baud,["ATDL ", HEX DL_Addr,CR] ' Set Destination Node Address
SEROUT Tx,Baud,["ATCN",CR]             ' Exit Command Mode
RETURN

CheckAck:
SERIN Rx,Baud,1000,CheckTimeout,[DEC dataIn] ' Accept incoming byte
IF dataIn = 1 THEN                        ' If 1, then ack'd
    DEBUG BELL,"OK - Ack Received!",CR
ELSE                                       ' If received, but not "1", problem
    DEBUG "Bad Ack!",CR
ENDIF
RETURN

CheckTimeout:
    DEBUG "No ack received!",CR          ' If nothing recieved
RETURN

```

The Remote node's code uses RTS and a short timeout to keep the buzzer sounding and servo positioned. It accepts a letter code and a decimal value for control (if L, S, or B) or sends the phototransistor's reading (if R). For control actions, it sends back a decimal value of 1 as acknowledgment to the Base once data is received and processed.

If no data arrives within 10 ms, the execution branches to the **Control** subroutine to control the LED, drive the servo, and sound the buzzer. By using a timeout, the output devices are continually refreshed. RTS ensures that any data received during control action timing will be buffered for the next **SERIN** operation.

## 5: Network Topologies & Control Strategies

---

```
' *****
' Polling_Remote.bs2
' This program accepts a character and values:
'   - L & 0 or 1 to control state of LED
'   - B & value to control buzzer frequency
'   - S & value to control servo position
'   - R to return value of light sensor
' Return acknowledgements or value to Base
' The address of node may be set by changing MY_Addr value
' Requires 802.15.4 XBee (Series 1)
' *****
' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
    T9600      CON      84
#CASE BS2SX, BS2P
    T9600      CON      240
#CASE BS2PX
    T9600      CON      396
#ENDSELECT

' ***** Variable, Constants and Pins
Baud          CON      T9600

LED           PIN      0
Buzzer        PIN      8
PhotoT        PIN      5
Servo         PIN      12

Rx            PIN      15   ' XBee DOUT
Tx            PIN      14   ' XBee DIN
RTS           PIN      11   ' XBee RTS

Freq          VAR      Word  ' Received frequency for buzzer
State         VAR      Bit   ' Received state of LED
DataIn        VAR      Byte  ' General byte data
Light         VAR      Word  ' Measured light level
Position      VAR      Word  ' Received servo position

My_Addr       CON      $2    ' Set address of node, modify as desired, $1-$FFFE

' ***** Configure XBee to use RTS and set Address
Position = 750

PAUSE 500
DEBUG CLS,"Configuring XBee...",CR
PAUSE 3000                                ' Guard time
SEROUT Tx,Baud,["+++"]                    ' Command Mode Sequence
PAUSE 2000                                ' Guard time
SEROUT Tx,Baud,["ATD6 1",CR]              ' Enable RTS
SEROUT Tx,Baud,["ATMY ", HEX My_Addr,CR]  ' Set node address
SEROUT Tx,Baud,["ATDL 0,CN",CR]          ' Set destination address of Base
                                           ' & Exit Command Mode
```



```

' ***** Main Loop
DO
  GOSUB AcceptData
  GOSUB Control
LOOP

AcceptData:
SERIN Rx\RTS,Baud,10,Timeout,[DataIn]      ' Accept byte
SELECT DataIn
  CASE "L"                                  ' L to control LEF
    SERIN Rx\RTS,Baud,1000,Timeout,[DEC State]' Accept LED state
    PAUSE 200                               ' Give Base time to set up
    SEROUT Tx,Baud,[CR,DEC 1,CR]           ' Return acknowledgment

  CASE "B"                                  ' B to set Buzzer
    SERIN Rx\RTS,Baud,1000,Timeout,[DEC Freq] ' Accept buzzer frequency
    PAUSE 200                               ' Give Base time to set up
    SEROUT Tx,Baud,[CR,DEC 1,CR]           ' Return acknowledgment

  CASE "S"                                  ' S to control Servo
    SERIN Rx\RTS,Baud,1000,Timeout,[DEC Position]' Accept position
    PAUSE 200
    SEROUT Tx,Baud,[CR,DEC 1,CR]           ' Return acknowledgment

  CASE "R"                                  ' R to read light sensor
    HIGH PhotoT                             ' Use RCTime to get value
    PAUSE 5
    RCTIME PhotoT,1,Light
    PAUSE 100                               ' Give Base time to set up
    SEROUT Tx, Baud,[DEC Light,CR]         ' Send value to Base
ENDSELECT

Timeout:
RETURN

Control:
  IF State = 1 THEN                          ' Control LED based on state
    HIGH LED
  ELSE
    LOW LED
  ENDIF

  IF Freq <> 0 THEN                          ' Control Buzzer based on Freq
    FREQOUT Buzzer,50,Freq
  ELSE
    PAUSE 100
  ENDIF

  FOR DataIn = 1 TO 20                       ' Control Servo based on Position
    PULSOUT Servo, Position
    PAUSE 20
  NEXT
RETURN

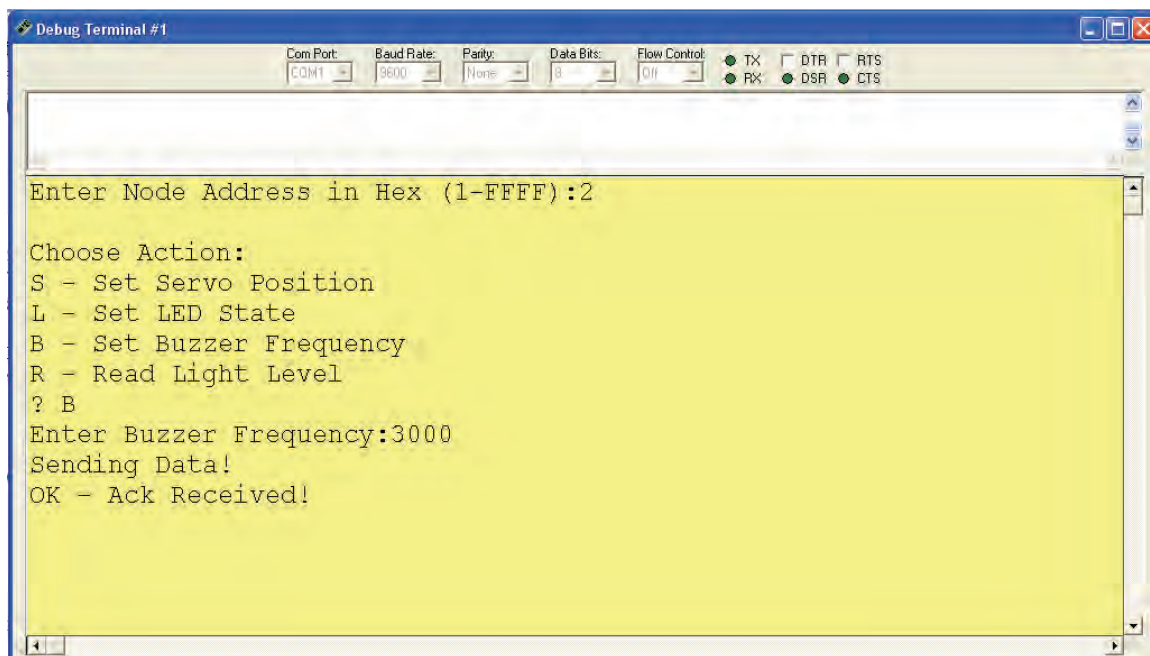
```

## 5: Network Topologies & Control Strategies

---

### **Testing:**

- ✓ Open Polling\_Remote.bs2
- ✓ Modify the `MY_Addr` constant to assign your Remote node's address, \$1 to \$FFFE (code default is \$2).
- ✓ Download to your Remote node hardware.
- ✓ Repeat steps above for any additional Remote nodes, giving each a unique address.
- ✓ Open and download Manual\_Polling\_Base.bs2 and download to your Base hardware.
- ✓ Using the Debug Terminal's transmit pane (as shown in Figure 5-6), answer requested information using both used and unused Remote node addresses and monitor the Remote's actions and responses.
- ✓ Note that the Node Address is entered as a "2" in the Debug Terminal without the hexadecimal identifier of \$.



**Figure 5-6: Manual Polling Debug Terminal**

### **Testing using Broadcast to Multiple Units**

If you have multiple Remotes, perform a broadcast to all units using a node address of FFFF to control an action and monitor Remotes. You may test with a single Remote node to see that it works.

### **Optional: Using XBee & X-CTU for Monitoring and Control**

Using an XBee on USB and Digi's X-CTU software is an effective way of testing data communications, debugging, and monitoring data to or from the Base. It may be used to monitor data being sent from the Base node to the Remote node by assigning the XBee the Remote node's address. It may be used to monitor data from the Remote node by assigning it the Base node's address. By assigning the destination address to that of the Remote node, you may use the terminal to send the control codes and values to the Remote.

- ✓ Place the USB-connected XBee on address 0 (**ATMY 0**) to monitor data from the Remote node to the Base. Use the normal Base hardware to send data, and use the Debug Terminal and the Base node to poll Remote nodes.
- ✓ Place the USB-connected XBee on a Remote's address (**ATMY 2**) to monitor data from the Base node to the Remote node.
- ✓ Place the USB-connected XBee to send to the Remote node's address (**ATDL 2**) with the MY address of 0 (**ATMY 0**).
- ✓ Send data to a Remote node by using the Assemble Packet window, such as:
  - L (Enter),1 (Enter), or ,
  - S (Enter), 750 (Enter), or,
  - B (Enter) 2500 (Enter), or,
  - R to read

### **More on Acknowledgements**

For acknowledgements, we have 3 cases: A "1" returned to the Base means data was accepted and acknowledged (ACK). If no value is received, this is a no-acknowledgement (NACK) meaning the data wasn't received. A value other than "1" would be a bad acknowledgement; this will normally never happen unless there is a problem in our code and data is in the XBee buffer that isn't meant to be there.

While our code doesn't take action on a NACK except to display it, you may program a **DO WHILE... LOOP** to send the data several times (keeping track with a counter) until the retry value is exceeded or an ACK is received. This is very similar to what the XBee does to ensure data delivery between modems, but ours would be for data delivery between the microcontroller applications.

### **Point-to-Multipoint – Automatic Polling of Remote Units with dBm**

With automatic polling, the Base node cycles through the provided range of Remote unit addresses, sending new parameters to each, reading each, and requesting and displaying the RSSI level of the received packet using **ATDB**. For control actions, acknowledgements from Remote units are still accepted and the status is displayed.

In the code, setting **Start\_Addr** and **End\_Addr** establishes the range of addresses to poll. A **FOR-NEXT** loop cycles through the addresses, setting the **DL** parameter for each using fast Command Mode. Frequency, servo position and LED state are set and sent to the individual Remote unit using the same format as in manual polling. The R command is sent to read the Remote unit and the dBm level is obtained and displayed. After a complete cycle, one more set of updates is sent to the broadcast address updating all Remote nodes with the same data.

Partial code listing for `Automatic_Polling_Base.bs2`; please see distributed files for full listing:

```
' ***** Main Loop
DO
  FOR DL_Addr = Start_Addr TO End_Addr ' loop through range of addresses
    DEBUG CR,CR,"***** Starting Control of Address ",
      IHEX DL_Addr," *****"
    GOSUB Config_XBee ' Set DL address
    State = 1 :GOSUB LED_State : PAUSE 200 ' Set Remote LED
    Position = 500 :GOSUB Servo_Pos : PAUSE 200 ' Set Remote servo
    Freq = 5000 :GOSUB Buzzer_Freq : PAUSE 200 ' set Remote buzzer
```

## 5: Network Topologies & Control Strategies

---

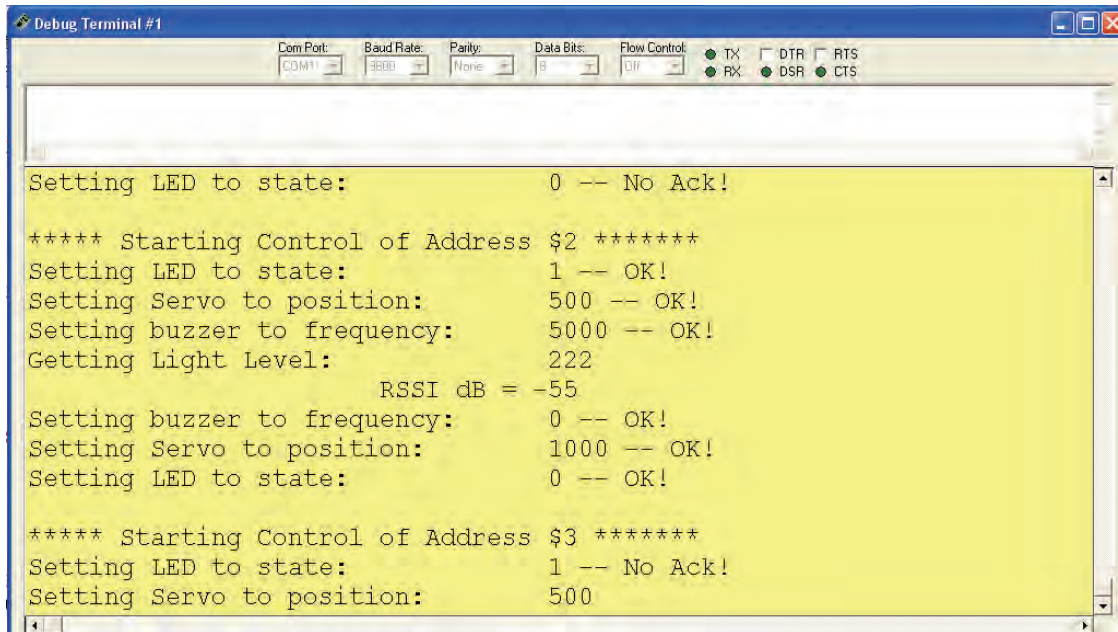
```
GOSUB Read_Light          : PAUSE 200 ' Go read Remote
GOSUB Get_dB              ' Go read & display dB
Freq = 0                  :GOSUB Buzzer_Freq : PAUSE 200 ' Set Remote buzzer
Position = 1000 :GOSUB Servo_Pos   : PAUSE 200 ' Set Remote servo
State = 0                  :GOSUB LED_State   : PAUSE 200 ' Set Remote LED
PAUSE 2000
NEXT

DEBUG CR, CR,"***** Control All Nodes! *****"
DL_Addr = $FFFF          ' Set to control ALL nodes
GOSUB Config_XBee       ' Set DL address
  State = 1              :GOSUB LED_State   : PAUSE 200 ' Set all Remote LEDs
  Position = 500         :GOSUB Servo_Pos   : PAUSE 200 ' Set all servos
  Freq = 5000            :GOSUB Buzzer_Freq : PAUSE 200 ' Set all buzzers
  Freq = 0                :GOSUB Buzzer_Freq : PAUSE 200
  Position = 1000        :GOSUB Servo_Pos   : PAUSE 200 ' Set all servos
  State = 0              :GOSUB LED_State   : PAUSE 200 ' Set all LEDs
  PAUSE 2000
LOOP

Get_dB:
  ' Request, accept, display dB level
  PAUSE 10                    ' Short guard time
  SEROUT Tx,Baud,["+++"]      ' Command Mode sequence
  PAUSE 10                    ' Short guard time
  SEROUT TX,Baud,["ATDB",CR]  ' request dB level
  SERIN Rx,Baud,200,dB_Timeout,[HEX DataIn] ' Accept returning data
  SEROUT Tx,Baud,["ATCN",CR]  ' Exit Command Mode
  DEBUG CR,"                  RSSI dB = -", DEC DataIn
dB_timeout:
RETURN
```

### **Testing:**

- ✓ Open Polling\_Remote.bs2
- ✓ Modify the **MY\_Addr** constant to assign your Remote's address, \$1 to \$FFFE (code default is \$2).
- ✓ Download to your Remote node hardware.
- ✓ Repeat steps above for any additional Remotes, giving each a unique address.
- ✓ Open Automatic\_Polling\_Base.bs2. Modify **Start\_Addr** and **End\_Addr** to encompass a range of addresses included in your Remote nodes—include a range beyond your actual Remotes' node addresses to test what happens when node is not present (by default, range is addresses 1 to 3).
- ✓ Download to your Base hardware.
- ✓ Using the Debug Terminal, monitor the polling as shown in Figure 5-7.



```
Debug Terminal #1
Com Port: COM1 Baud Rate: 9600 Parity: None Data Bits: 8 Flow Control: DTR
TX RX DTR DSR RTS CTS

Setting LED to state: 0 -- No Ack!

***** Starting Control of Address $2 *****
Setting LED to state: 1 -- OK!
Setting Servo to position: 500 -- OK!
Setting buzzer to frequency: 5000 -- OK!
Getting Light Level: 222
RSSI dB = -55
Setting buzzer to frequency: 0 -- OK!
Setting Servo to position: 1000 -- OK!
Setting LED to state: 0 -- OK!

***** Starting Control of Address $3 *****
Setting LED to state: 1 -- No Ack!
Setting Servo to position: 500
```

Figure 5-7: Automatic Polling Debug Terminal

### Optional: Using XBee & X-CTU for Monitoring and Control

Once again, an XBee on USB may be used to monitor data communications—if you did it with manual polling, you can do it with automatic polling as well.

- ✓ Using X-CTU, set the USB-connected XBee to a Remote node's address (**ATDL 2**) to monitor polling data from the Base node.

### Scheduled Updates from Remote Units

Allowing the Remote nodes to make initial contact with the Base allows them the ability to perform other processes or to enter reduced power states (Chapter 6) without having to continually monitor for incoming communications from the Base. A down side of this is that with relatively slow processing and limited serial communications handling of the BASIC Stamp, having multiple units attempting to contact the Base simultaneously could affect operation. The use of RTS to allow the Base node's XBee to buffer data (around 100 bytes worth), use of a start-delimiting character, and performing operations quickly will aid in ensuring that the BASIC Stamp will get incoming current values and send updates. Additionally, to limit the back-and-forth communications, application acknowledgements will not be sent for this example.

To make the operation simpler, the communications back and forth will be limited to chunks of data instead of sending control strings for LED, servo and buzzer independently.

- The Remote node will send 'C' and all its current values to the Base node including its address (so the Base knows which address is contacting it) in a single transmission.
- The Base node monitors for the 'C' start delimiter and accepts incoming data.
- The Base node will configure DL for the Remote's address using fast Command Mode.
- The Base node will calculate changes and send out "U" and all updated values to the Remote node in a single transmission.
- The Remote node waits for a short time for 'U' and updated data.
- Base is ready for an update from another unit.

## 5: Network Topologies & Control Strategies

---

Partial code listing for Scheduled\_Base.bs2; please see distributed files for full listing:

```
' ***** Main Loop
DO
SERIN Rx\RTS,Baud,20,UpdateTOut,[DataIn] ' Wait for "C" with timeout
IF DataIn = "C" THEN ' If "C" (Current), collect
  DEBUG CR,"Incoming Data",CR
  ' Accept incoming values with timeout
  SERIN Rx\RTS,Baud,1000,UpdateTOut,[HEX DL_Addr]
  SERIN Rx\RTS,Baud,1000,UpdateTOut,[DEC Light]
  SERIN Rx\RTS,Baud,1000,UpdateTOut,[DEC State]
  SERIN Rx\RTS,Baud,1000,UpdateTOut,[DEC Freq]
  SERIN Rx\RTS,Baud,1000,UpdateTOut,[DEC Position]

  DEBUG CLS, " Unit ", IHEX DL_Addr," Reports",CR,
    "Light Reading: ", DEC Light,CR, ' Display data
    "LED State: ", DEC State,CR,
    "Frequency: ", DEC Freq,CR,
    "Position: ", DEC Position,CR

  GOSUB Config_XBee ' Configure XBee for DL address
  GOSUB ChangeRemote ' Change device values

  SEROUT Tx,Baud,["U",CR,CR, ' Send Update start character
    DEC State,CR,CR, ' Send new LED state
    DEC Freq,CR,CR, ' Send new frequency
    DEC Position,CR] ' Send new position

ENDIF
UpdateTOut:
DEBUG "."
LOOP

ChangeRemote:
Freq = Freq + 500 ' Add 500 to received value
IF Freq > 5000 THEN Freq = 1000 ' limit 1000-5000
DEBUG CR,"Setting buzzer to frequency of:", DEC Freq
IF State = 1 THEN ' Change LED state
  State = 0
ELSE
  State = 1
ENDIF
DEBUG CR,"Setting LED to state of: ", DEC State
Position = Position + 50 ' Add 50 to servo position
IF position > 1000 THEN Position = 500 ' Limit 500 to 1000
DEBUG CR,"Setting Servo to position of: ", DEC Position
RETURN
```

On the Remote node, the code cycles through every 3 seconds to send current values and receive updates. **FlushBuffer** is used to ensure the buffer is empty of data. In accepting data, just in case the buffer held data that wasn't intended, the **GOTO AcceptData** ensures that erroneous characters won't ruin accepting incoming data if **SERIN** gets something other than a "U" delimiter.



### Choosing Start Delimiters

We've used "!", "U", "C" and other characters as delimiters to identify incoming strings and parameters. Typically you want to choose characters that won't likely be transmitted for other reasons. Included in those **not** to use are the letters "O" and "K" since OK's are sent from XBee during configuration changes.

Partial code listing for Scheduled\_Remote.bs2; please see distributed files for full listing:

```

' ***** Main Loop

PAUSE 1000
DO
  GOSUB FlushBuffer          ' Ensure XBee buffer empty
  GOSUB SendUpdate          ' Send current values
  GOSUB AcceptData          ' Receive returned data
  GOSUB Control              ' Control Devices
  PAUSE 3000                 ' Three seconds before sending again
LOOP

FlushBuffer:                 ' Empty XBee buffer
  SERIN Rx\RTS,Baud,50,FlushTime,[DataIn] ' Get data
  GOTO FlushBuffer           ' If data, get more data
FlushTime:                   ' If timed out, done
RETURN

SendUpdate:
  GOSUB ReadLight            ' Read light level
  DEBUG CR,"Sending Updates",CR
  SEROUT Tx,Baud,["C",CR,CR,          ' Send C and values
                HEX My_Addr,CR,CR,    ' Node's address
                DEC Light,CR,CR,      ' Light level
                DEC State,CR,CR,      ' LED state
                DEC Freq,CR,CR,       ' Frequency
                DEC Position,CR]      ' Servo position
RETURN

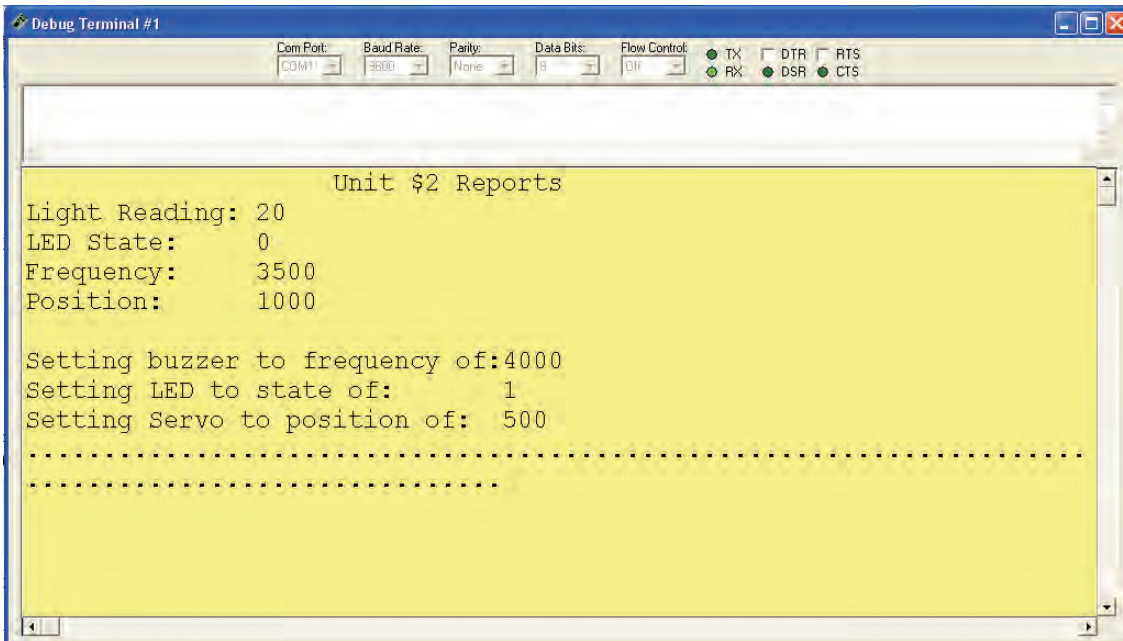
AcceptData:
  SERIN Rx\RTS,Baud,1000,UpdateTout,[DataIn] ' Accept byte
  IF DataIn = "U" THEN          ' If "U", then accept
    SERIN Rx\RTS,Baud,1000,UpdateTout,[DEC State] ' LED State
    SERIN Rx\RTS,Baud,1000,UpdateTout,[DEC Freq] ' Buzzer freq
    SERIN Rx\RTS,Baud,1000,UpdateTout,[DEC Position]' Servo Position
    DEBUG "Updates:",CR        ' Display
    DEBUG "LED State:      ", DEC State,CR
    DEBUG "Frequency:     ", DEC Freq,CR
    DEBUG "Position:     ", DEC Position,CR
  ENDIF
  GOTO AcceptData              ' Ensure not more data
UpdateTout:
RETURN

```

## 5: Network Topologies & Control Strategies

---

- ✓ Open Scheduled\_Remote.bs2.
- ✓ Modify the **MY\_Addr** constant to assign your Remote's address, \$1 to \$FFFE (code default is \$2).
- ✓ Download to your Remote hardware.
- ✓ Repeat steps above for any additional Remotes, giving each a unique address.
- ✓ Download Scheduled\_Base.bs2 to your Base hardware.
- ✓ Using the Debug Terminal, monitor the updates as shown in Figure 5-8.



**Figure 5-8: Scheduled Update Monitoring in Debug Terminal**

### Separating Update Times on Remotes

In our code, we simply wait three seconds between updates. If you have multiple Remote nodes with identical code and they are powered up at once, they will all be on the same schedule and may cause issues with updates as they all send at once. By energizing at separate times, it will help ensure updates are separated. In code you may change the **PAUSE** before the main **DO-LOOP** to be based on something unique if all are powered up at once, such as:

```
PAUSE 1000 + (DL_Addr * 500)
```

This will help offset the transmission of Remote nodes. The XBee itself has a setting, Random Delay Slot (RN) to help randomize its “back-off and retry” algorithm so that the modems themselves are not locked in sync and having problems sending data with the exact same retry times.


### Propeller Examples

The Propeller chip's speed, buffering of the serial data, and use of multiple cogs makes duplex serial communications easy to implement without the use of flow control. We will use basic, common hardware to illustrate principles that may aid you in development of your own systems.



## Hardware

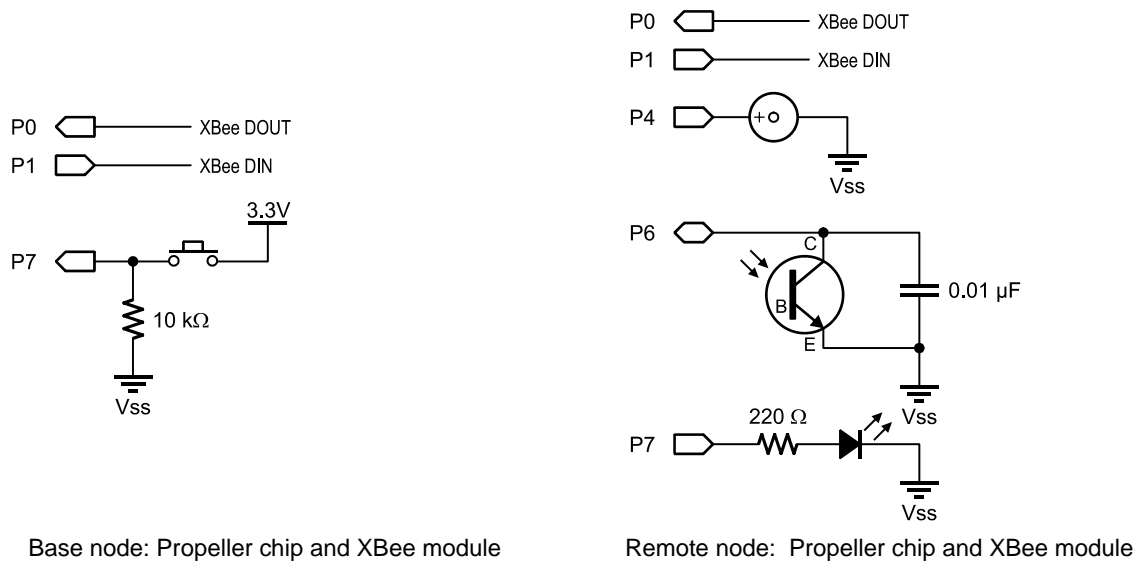
- (2) Propeller development boards
  - (2) XBee adapter boards
  - (2) XBee with default settings
  - (1) Pushbutton
  - (1) LED
  - (1) Buzzer
  - (1) Phototransistor
  - (1) 0.01 $\mu$ F capacitor
  - (1) 10 k $\Omega$  Resistor
  - (1) 220  $\Omega$  Resistor
- Optional: XBee USB Adapter board and XBee module



**Board and Power!**

Use an appropriate BASIC Stamp XBee Adapter Board with supplied power per Chapter 2.

Assemble the hardware as shown in Figure 5-9 for the Base node and at least one Remote node (multiple Remote nodes may be built for testing multi-node communications). The Base will use the X-CTU terminal for control and notifications in most examples (any terminal program may be used such as the Parallax Serial Terminal).



**Figure 5-9: Propeller Base and Remote Schematics**

## Using Point-to-Point for Pushbutton Control

In this example, the Base node unit will send the data to set the state of an LED and the frequency for the buzzer. To exploit the Propeller chip's features, counters are used control the frequency and the brightness of the LED using PWM. *State* is a value from 0 to 1023 for 10-bit PWM control and *Freq* is a value from 0 to 5000. Pressing the button on the Base node will increase *State* and *Freq*, not pressing the button will ramp down *State* and *Freq*. The Base node will continually send “!” and the two decimal values to be received and used.

## 5: Network Topologies & Control Strategies

---

Partial code listing for Simple\_Control\_Base.spin; please see distributed files for full code:

```
Pub Start | State, Freq
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
State := 0                          ' Initial state and frequency
Freq := 0

repeat
  if ina[PB] == 1                    ' If button pressed,
    Freq := Freq + 100 <# 5000      ' increase Freq to 5000 max
    State := State + 10 <# 1020    ' increase State to 1020 max
  else                                ' If released,
    Freq := Freq - 100 #> 0        ' decrease freq to 0 min
    State := State - 10 #> 0      ' decrease state to 0 min

  XB.Tx("!!")                      ' Send start delimiter
  XB.Dec(State)                    ' Send decimal value of State + CR
  XB.CR
  XB.Dec(Freq)                    ' Send decimal value of Freq + CR
  XB.CR
  XB.Delay(100)                   ' Short delay before repeat
```

The Remote node's code waits for the start delimiter ("!!"), then accepts two decimal values for control of the LED PWM and the buzzer frequency, which are used to control the devices.

Partial code listing for Simple\_Control\_Remote.spin; please see distributed files for full code:

```
Pub Start | DataIn
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee

repeat
  DataIn := XB.Rx                    ' Accept incoming byte
  If DataIn == "!!"                 ' If start delimiter
    DataIn := XB.RxDecTime(500)     ' Accept value for LED state
    if DataIn <> -1                  ' If wasn't time out, set PWM
      PWM_Set(LED,DataIn)

  DataIn := XB.RxDecTime(500)       ' Accept value for Frequency
  if DataIn <> -1                    ' If wasn't timeout, set FREQOUT
    Freqout_Set(Buzzer,DataIn)
```

### Testing:

- ✓ Download Simple\_Control\_Remote.spin to the Remote node, and Simple\_Control\_Base.spin to the Base node.
- ✓ Press and release the pushbutton on the Base node while monitoring the Remote node's LED and buzzer.

### Optional: Using an USB XBee and X-CTU for Control and Monitoring

You may use X-CTU with a USB-connected XBee on address 0 to send data to the Remote node through the Assemble Packet window such as, !200 (Enter) 2000 (Enter). X-CTU may also be used to monitor data from the Base node to the Remote node.

### **Point-to-Multipoint—Manual Polling of Remote Nodes**

In this example the Propeller Base node requests and accepts input from the user via the PC and a terminal window. The Base node requests from the user the Remote node's address to control (or read and control) devices connected to the Remote node: turn the LED on and off, play tones on the buzzer, and initiate a phototransistor measurement and read the results). The Base node also accepts an application acknowledgement from the Remote node to help ensure actions were accepted.

The code uses the `XBee_Object` for the PC communications driver; notably, it is more fully featured to accept decimal input from the terminal. The Base node's code calls `XB.AT_INIT` to switch to fast Command Mode for updates allowing XBee configuration changes in milliseconds instead of seconds. The Base node's address is set to 0. The code requests from the user the Remote node's address and uses `XB.AT_ConfigVal` to set the DL to send data to the identified address. The user is prompted for the control action (L,B or R) and a value in the case of LED and buzzer control. The control action code is sent along with a value for the update. It then accepts the phototransistor reading for an "R" action, or obtains an acknowledgement for control actions of the LED and buzzer from the Remote unit.

Due to buffering of serial data (up to 15 characters) by the Propeller `FullDuplexSerial.spin` drivers, used and extended by the `XBee_Object`, `RxFlush` is used to ensure the buffer is empty of "OK"s from the XBee and other data. In the Acknowledgement method, a byte value returned of 1 indicates data accepted, no data returned (a timeout) indicated no-acknowledgment (NACK). Any other value other than 1 indicates a problem with erroneous data in the buffer.

While our code doesn't take action on a NACK except to display it, you may program a REPEAT-WHILE loop to send the data several times (keeping track with a counter) until a retry value is exceeded or an ACK is received. This is very similar to what the XBee does to ensure data delivery between Modems, but ours would be for data delivery between the applications.

Partial code listing for `Manual_Polling_Base.spin`; please see distributed file for full source code:

```
OBJ
  XB   : "XBee_Object"
  PC   : "XBee_Object" ' Using XBee object on PC side for more versatility

Pub Start | Light, DataIn, DL_Addr, State, Freq

  XB.Delay(2000)
  PC.start(PC_Rx, PC_Tx, 0, PC_Baud) ' Initialize comms for PC
  PC.str(string("Configuring XBee..."),CR)
  XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
  XB.AT_Init ' Set up for fast Command Mode
  XB.AT_Config(string("ATMY 0")) ' Set address of Base

  repeat
    PC.CR ' User defines Remote node address
    PC.str(string("*** Enter address of node ***"),CR)
    PC.str(string("(1 to FFFE or FFFF for all):"))
    DL_Addr := PC.RxHex ' Accept address in hex and sets XBee DL
    XB.AT_ConfigVal(string("ATDL "),DL_Addr)
    ' User chooses action to take
    PC.str(string(CR,"***** Choose Action: *****"),CR)
    PC.str(string("L - Control LED"),CR)
    PC.str(string("B - Control Buzzer"),CR)
    PC.str(string("R - Read Sensor"),CR)
    DataIn := PC.Rx ' Accept action
    XB.RxFlush

  case DataIn
    "L","1": PC.str(string(CR,"Enter LED state (0-1023): "))
             State := PC.RxDec ' Accept value for state
             XB.tx("L") ' Transmit L followed by State
             XB.Dec(State)
             XB.CR
             GetAck ' Check for acknowledgement
```

## 5: Network Topologies & Control Strategies

---

```
"B","b": PC.str(string(CR,"Enter buzzer Frequency (0-5000): "))
          Freq := PC.RxDec           ' Accept value for frequency
          XB.tx("B")                ' Transmit F followed by State
          XB.Dec(Freq)
          XB.CR
          GetAck                     ' Check for acknowledgement

"R","r": XB.Tx("R")                 ' Transmit R to Remote
          Light := XB.RxDecTime(500) ' Accept response
          if Light == -1             ' If no data returned,
            PC.str(string(CR,"No Response",CR))
          else                       ' Else, good data
            PC.str(string(CR,"Light Level = "))
            PC.dec(light)
            PC.CR
PC.Delay(2000)

Pub GetAck | Ack
Ack := XB.RxTime(500)               ' wait for response
If Ack == -1                        ' -1, no ack received
  PC.Str(string("-No Ack!",CR))
elseif Ack == 1                    ' 1, good ack
  PC.str(string("-Good Ack! ",CR))
else                                ' any other value - problem
  PC.str(string("-Bad Ack!",CR))
```

In the Remote node's code, the program configures for fast Command Mode and configures the **MY** address of the Remote node based on the value in the CON section of the code for MY\_Addr. The code waits for a byte and checks the action identifier to determine the action, such as accepting and controlling the LED brightness based on the value, accepting and setting the buzzer's frequency, or reading and sending the value of the phototransistor. With control actions, the acknowledgement byte of 1 is returned.

Partial code listing for Polling\_Remote.spin; please see distributed file for full source code:

```
Pub Start | DataIn, Light
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
XB.AT_Init                          ' Set up XBee for fast Command Mode
XB.AT_ConfigVal(string("ATMY "),MY_Addr) ' Configure node's address
XB.AT_Config(string("ATDL 0"))      ' Configure address of Base

XB.RxFlush                          ' Ensure XBee buffer empty
repeat
  DataIn := XB.Rx                   ' Wait for byte
  case DataIn
    "L": DataIn := XB.RxDecTime(500) ' If byte L, accept data from LED PWM
          if DataIn <> -1             ' Ensure wasn't timeout
            XB.Tx(1)                 ' Send ack byte of 1
            PWM_Set(LED,DataIn)     ' Set PWM

    "B": DataIn := XB.RxDecTime(500) ' If byte B, accept data for buzzer
          ' frequency
          if DataIn <> -1             ' Ensure wasn't timeout
            XB.Tx(1)                 ' Send ack byte of 1
            Freqout_Set(Buzzer,DataIn) ' Set buzzer frequency

    "R": Light := RCTime(PhotoT)     ' If R, read RCTime of sensor
          XB.DEC(Light)              ' Send value
          XB.CR
```

### Testing:

- ✓ Open Polling\_Remote.spin and modify the MY\_Addr value as desired, choosing different values for multiple Remote nodes (default is 2).
- ✓ Download the code to the Remote node.
- ✓ Repeat above if multiple Remotes are used.
- ✓ Download Manual\_Polling\_Base.spin to the Base node (Use F11 in the event the terminal resets the Propeller).
- ✓ Open a terminal program (X-CTU or other) on the PC for communications with the Base node and respond to prompts as shown in Figure 5-10. If you used F11 to download your code, you may reset the Propeller to catch initial messages.
- ✓ Test both good and unused Remote node addresses.
- ✓ Use an address of FFFF to test a broadcast to all nodes for a control action.

```
X-CTU [COM22]
About
PC Settings | Range Test | Terminal | Modem Configuration
Line Status: CTS CD DSR
Assert: DTR [x] RTS [x] Break [ ]
Close Com Port Assemble Packet Clear Screen Show Hex
*** Enter address of node ***
(1 to FFFE or FFFF for all):2
***** Choose Action: *****
L - Control LED
B - Control Buzzer
R - Read Sensor
L
Enter LED state (0-1023): 200
Good Ack!
*** Enter address of node ***
(1 to FFFE or FFFF for all):3
***** Choose Action: *****
L - Control LED
B - Control Buzzer
R - Read Sensor
R
Light Level = 5712
*** Enter address of node ***
(1 to FFFE or FFFF for all):3
***** Choose Action: *****
L - Control LED
B - Control Buzzer
R - Read Sensor
R
No Response
COM22 9600 8-N-1 FLOW:NONE Rx: 563 bytes
```

Figure 5-10: Manual Polling Base Terminal

## 5: Network Topologies & Control Strategies

---

### Using a USB XBee for Monitoring and Control

Using an XBee connected to the PC using a XBee USB Adapter Board and a terminal window is good way to test Remote node code and monitor data passed. By setting the USB XBee to a **DL** of a Remote node's address, you can use the Assemble Packet Window of X-CTU to send control actions such as setting the LED (L200 + Enter), buzzer (B3000 + Enter) or reading the phototransistor (R). It may also be used on a **MY** address of 0 to monitor data from the Remote node, or a **MY** of the Remote node's address to monitor data from the Base.

### **Point-to-Multipoint—Automatic Polling of Remote Units with dBm**

With automatic polling, the Base node cycles through the provided range of Remote node addresses sending new parameters to each, reading each, and requesting and displaying the RSSI level of the received packet using **ATDB**. For control actions, acknowledgements from Remote nodes are accepted and the status is displayed. Polling\_Remote.spin is used on the Remote nodes.

The Base node's code uses a `repeat` loop to cycle through the defined range of addresses defined in the CON section, from `DL_Start` to `DL_end` and setting the XBee's **DL** parameter. It uses methods to increment (but limit) the Remote's state value of the LED and the buzzer's frequency, accepting acknowledgements, and reading the Remote's phototransistor. It also uses fast Command Mode to request, receive and display the RSSI dBm level using the **ATDB** command. After control of each node individually, it uses the broadcast address (`$FFFF`) to control the LED and buzzer of all Remote nodes at once.

Partial code listing for `Automatic_Polling_Base.spin`; please see distributed files for full code listing.

```
Pub Start
  XB.Delay(2000)
  PC.start(PC_Rx, PC_Tx, 0, PC_Baud) ' Initialize comms for PC
  PC.str(string("Configuring XBee...",CR))
  XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
  XB.AT_Init ' Set up XBee for fast Command Mode
  XB.AT_Config(string("ATMY 0")) ' Set Base node's address

repeat
  PC.str(string(CR,CR,"*** Individual Polling of Remotes ***"))
  ' Poll nodes from start to end address
  repeat DL_Addr from DL_Start to DL_End
    PC.str(string(CR,CR,"*** Controlling Node:      "))
    PC.DEC(DL_Addr)
    XB.AT_ConfigVal(string("ATDL "),DL_Addr) ' Set Remote address
    XB.Delay(100) ' Allow OK's buffer
    XB.RxFlush ' Empty buffer
    Set_LED ' Send LED settings
    Set_Buzzer ' Send buzzer settings
    GetReading ' Request Light value
    GetdB ' Read RSSI from Remote's data
    XB.Delay(2000) ' 2 second delay
  ' Control all Remotes using broadcast
  PC.str(string(CR,CR,"*** Controlling ALL Nodes ***"))
  DL_Addr := $FFFF ' Set broadcast address
  XB.AT_ConfigVal(string("ATDL "),DL_Addr)
  XB.Delay(100) ' Allow OK's to buffer
  XB.RxFlush ' Flush buffer
  Set_LED ' Control LEDs
  Set_Buzzer ' Control Buzzers
  XB.Delay(4000) ' 4 second delay before repeating
```

```

Pub Set_LED
  State += 100           ' Increase PWM state by 100
  if State > 1000       ' limit 0 to 1000
    State := 0
  PC.str(string(CR,"Setting LED to:      "))
  PC.Dec(State)
  XB.Tx("L")           ' Send L + value
  XB.Dec(State)
  XB.CR
  GetAck               ' Accept acknowledgement

Pub Set_Buzzer
  Freq += 500           ' Increase buzzer freq by 500
  if Freq > 5000        ' limit freq 0 to 5000
    Freq := 0
  PC.str(string("Setting Frequency to:  "))
  PC.Dec(Freq)
  XB.Tx("B")           ' Send B + value
  XB.Dec(Freq)
  XB.CR
  GetAck               ' Accept acknowledgement

Pub GetReading
  PC.str(string("Getting Light Level:   "))
  XB.Tx("R")           ' Send R for light level
  Light := XB.RxDecTime(500) ' Accept returned data
  If Light == -1        ' -1 means timeout
    PC.str(string("No Response"))
  else
    PC.Dec(Light)       ' Display value

Pub GetdB
  PC.str(string(CR,"Getting RSSI dBm:   "))
  XB.RxFlush           ' Empty buffer
  XB.AT_Config(string("ATDB"))       ' Request RSSI dB
  DataIn := XB.RxHexTime(500)         ' Accept returning data in HEX
  If DataIn == -1                     ' -1 means timeout
    PC.str(string("No Response",CR))
  else
    PC.Dec(-DataIn)                   ' Display value in hex

```

### **Testing:**

- ✓ Open Polling\_Remote.spin
- ✓ Modify the MY\_Addr constant to assign your Remote node's address, \$1 to \$FFFE (code default is \$2).
- ✓ Download to your Remote node.
- ✓ Repeat steps above for any additional Remote nodes, giving each a unique address.
- ✓ Open Automatic\_Polling\_Base.spin. Modify DL\_Start and DL\_End to encompass the range of addresses included in your Remote nodes—expand the range beyond your actual Remote node addresses to test what happens when node is not present (By default, range is addresses 1 to 3).
- ✓ Download to your Base node.
- ✓ Using a terminal window, monitor the polling as shown in Figure 5-11.

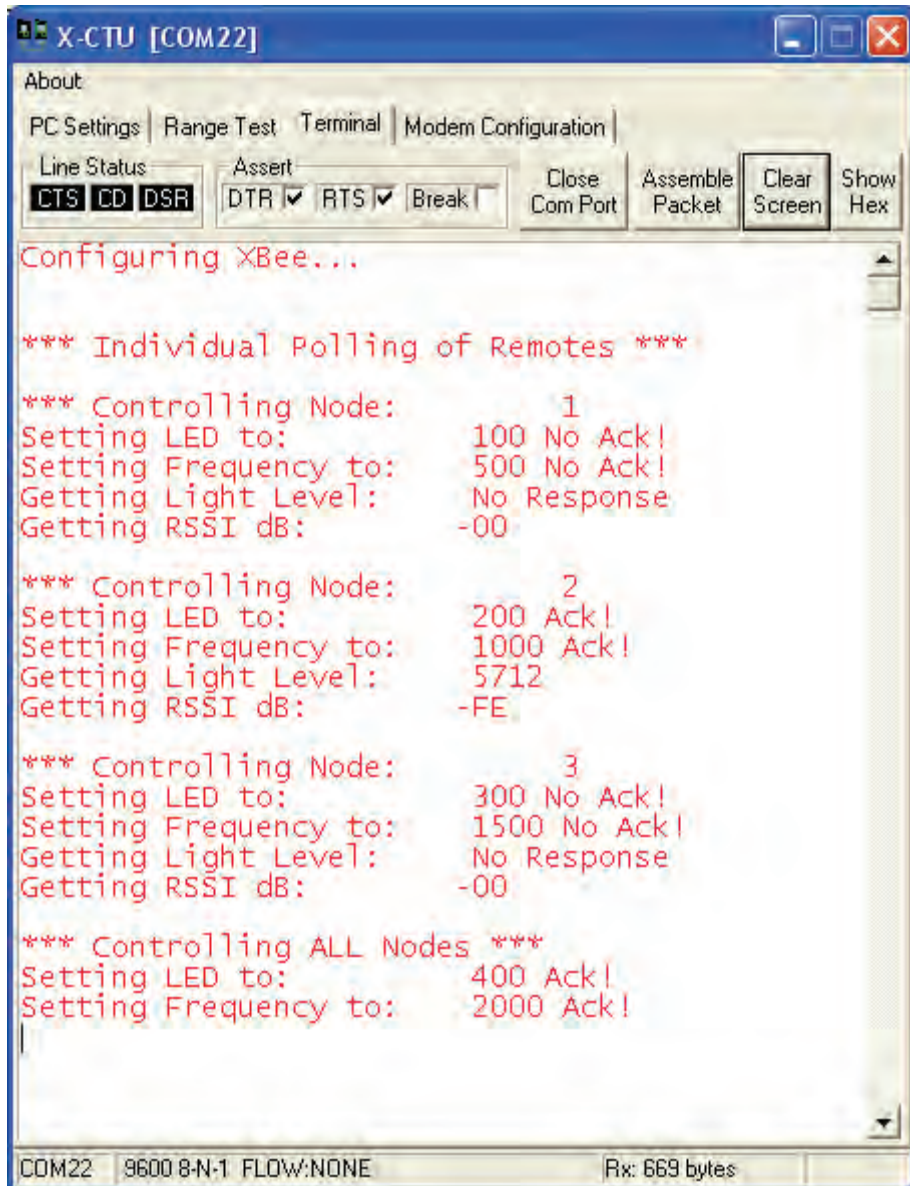


Figure 5-11: Automatic Polling Base Terminal

### Scheduled Updates from Remote Units

Allowing the Remote nodes to make initial contact with the Base node allows them the ability perform other processes or enter reduced power states (Chapter 6) while not having to be continually monitoring for incoming communications from the Base node. It also allows a Remote node with an urgent update to send data without waiting to be polled. To limit the back-and-forth, all data is sent to both the Base node and to the Remote nodes in one burst instead of separate commands being sent. Acknowledgements are also not used to limit data flow and to ensure the Base is ready as quickly as possible for an update from another unit.

To make the operation simple, the communications back and forth will be limited by sending chunks of data instead of sending control strings for LED and buzzer independently.

- The Remote node will send “C” and all its current values to the Base node including its address (so the Base knows which address is contacting it) in a single transmission.



- The Base monitors for the “C” start delimiter and accepts all incoming data.
- The Base will configure **DL** for the Remote node’s address using fast Command Mode used by XB.AT\_Config.
- The Base node will calculate and send out “U” and all updated values to the Remote node in a single transmission.
- The Remote node waits for a short time for “U” and updated data.
- Base is ready for an update from another unit.



### Choosing Start Delimiters

We’ve used “!”, “U”, “C” and other characters as delimiters and to identify incoming strings and parameters. Typically you want to choose characters that won’t typically be transmitted for other reasons. Included in those not to use are the letters “O” and “K” since OK’s are sent from XBee during configuration changes.

Partial code listing for Scheduled\_Base.spin; please see distributed files for complete code listing:

```

Pub Start
  XB.Delay(2000)
  PC.start(PC_Rx, PC_Tx, 0, PC_Baud) ' Initialize comms for PC
  PC.str(string("Configuring XBee...",CR))
  XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
  XB.AT_Init ' Configure for fast Command Mode
  XB.AT_Config(string("ATMY 0")) ' Set address of Base (MY)

  PC.str(string(CR,CR,"*** Waiting for Data"))
  repeat
    DataIn := XB.rx ' Wait for incoming byte character
    If DataIn == "C" ' If C, current update
      DL_Addr := XB.RxDecTime(500) ' Accept Remote address
      PC.str(string(CR," Update from Remote address :"))
      PC.HEX(DL_Addr,2)
      State := XB.RxDecTime(500) ' Accept LED state
      PC.str(string(CR," LED State: "))
      PC.DEC(State)
      Freq := XB.RxDecTime(500) ' Accept frequency
      PC.str(string(CR," Frequency: "))
      PC.DEC(Freq)
      Light := XB.RxDecTime(500) ' Accept light level
      PC.str(string(CR," Light Level: "))
      PC.DEC(Light)
      PC.str(string(CR,"** Updating Remote **"))
      XB.AT_ConfigVal(string("ATDL"),DL_Addr) ' Configure DL for Remote
      Set_LED ' Go send LED update
      Set_Buzzer ' Go send Buzzer update
      PC.str(string(CR,CR,"*** Waiting for Data"))
    else
      PC.tx(".") ' Non-C data

Pub Set_LED
  State += 100 ' Increase PWM state by 100
  if State > 1000 ' limit 0 to 1000
    State := 0
  PC.str(string(CR," Setting LED to: "))
  PC.Dec(State)
  XB.Tx("L") ' Send L + value
  XB.Dec(State)
  XB.CR
  
```

## 5: Network Topologies & Control Strategies

---

```
Pub Set_Buzzer
  Freq += 500          ' Increase buzzer freq by 500
  if Freq > 5000      ' limit freq 0 to 5000
    Freq := 0
  PC.str(string(CR,"   Setting Frequency to:   "))
  PC.Dec(Freq)
  XB.Tx("B")         ' Send B + value
  XB.Dec(Freq)
  XB.CR
```

The Remote node's code uses two cogs: one to send the current data (`Start` method) and one to accept updates (`AcceptData` method). A delay of 5 seconds is used between sending updates (and expecting updates). Due to the Propeller chip's parallel processing abilities, the Remote node's code can still accept updates at anytime should data be sent from the Base node or via an XBee using a USB connection, such as **B2000**.

Partial code listing for `Scheduled_Remote.spin`; please see distributed files for complete code listing:

```
Pub Start | DataIn
  XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
  XB.AT_Init                          ' Set for fast Command Mode
  XB.AT_ConfigVal(string("ATMY "),MY_Addr) ' Set Remote's address
  XB.AT_Config(string("ATDL 0"))      ' Set Base address (destination)

  cognew(AcceptData,@stack)

  XB.Delay(2000)

  repeat
    XB.tx("C")          ' Send C for current data
    XB.Dec(MY_Addr)     ' Send Remote's address
    XB.CR
    XB.Dec(State)      ' Send state of LED PWM
    XB.CR
    XB.Dec(FreqIn)     ' Send Frequency of buzzer
    XB.CR
    Light := RCTime(PhotoT) ' Send Light level
    XB.Dec(Light)
    XB.CR
    XB.Delay(5000)     ' Wait 5 seconds before next update

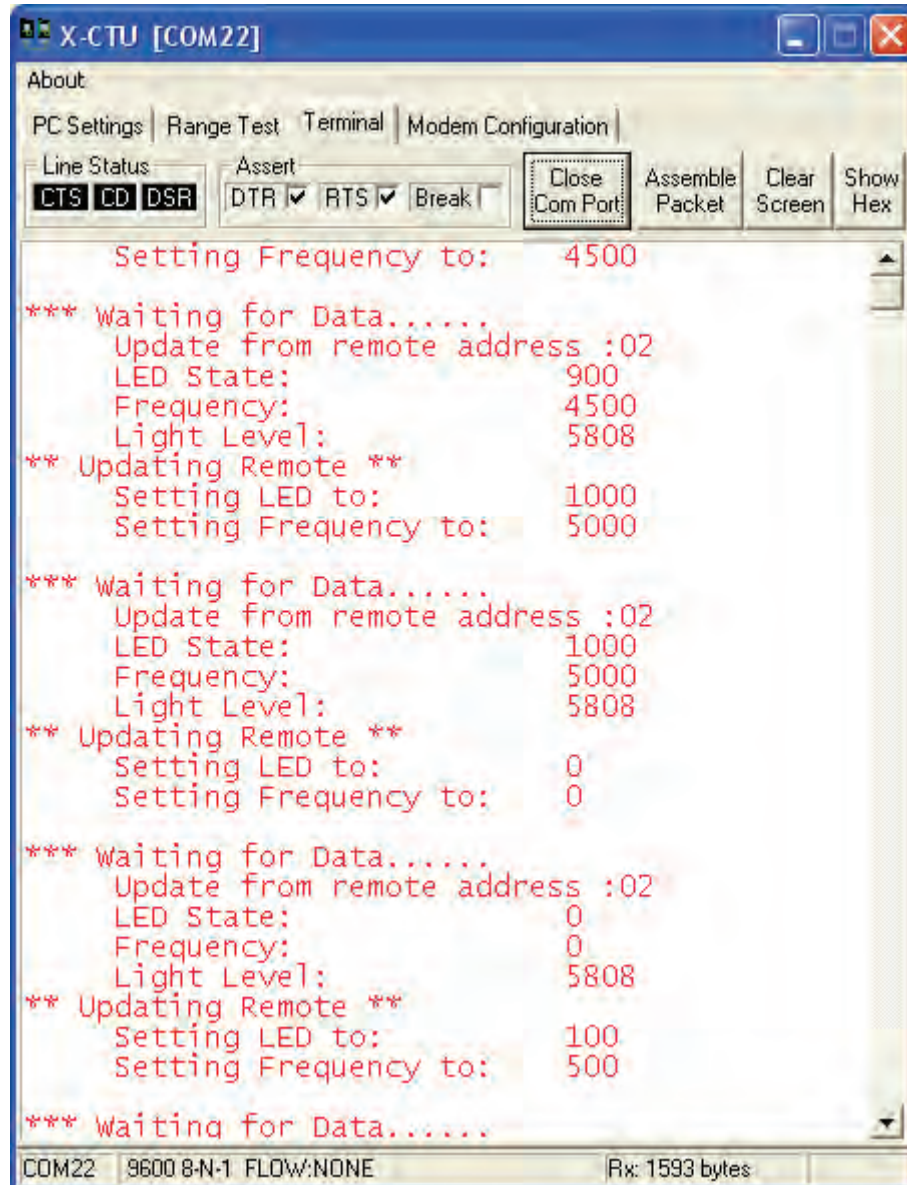
Pub AcceptData | DataIn
  ' Accept incoming settings

  XB.RxFlush          ' Ensure buffer empty
  repeat
    DataIn := XB.Rx   ' Accept incoming byte character
    case DataIn
      "L": DataIn := XB.RxDecTime(500) ' If L, accept data for LED
          if DataIn <> -1              ' Ensure not timeout value, -1
            State := DataIn           ' Accept state
            PWM_Set(LED,DataIn)      ' Set LED State

      "B": DataIn := XB.RxDecTime(500) ' If B, buzzer data
          if DataIn <> -1              ' Ensure not timeout value, -1
            FreqIn := DataIn          ' Accept data for Frequency
            Freqout_Set(Buzzer,DataIn) ' Set buzzer Frequency
```

- ✓ Open `Scheduled_Remote.spin`
- ✓ Modify the `MY_Addr` constant to assign your Remote node's address, \$1 to \$FFFE (code default is \$2).

- ✓ Download to your Remote node.
- ✓ Repeat steps above for any additional Remote nodes, giving each a unique address.
- ✓ Download Scheduled\_Base.spin to your Base nodes.
- ✓ Using a terminal window, monitor the updates as shown in Figure 5-12.



```
X-CTU [COM22]
About
PC Settings | Range Test | Terminal | Modem Configuration
Line Status: CTS CD DSR
Assert: DTR [x] RTS [x] Break [ ]
Close Com Port Assemble Packet Clear Screen Show Hex

Setting Frequency to: 4500
*** Waiting for Data.....
Update from remote address :02
LED State: 900
Frequency: 4500
Light Level: 5808
** Updating Remote **
Setting LED to: 1000
Setting Frequency to: 5000
*** Waiting for Data.....
Update from remote address :02
LED State: 1000
Frequency: 5000
Light Level: 5808
** Updating Remote **
Setting LED to: 0
Setting Frequency to: 0
*** Waiting for Data.....
Update from remote address :02
LED State: 0
Frequency: 0
Light Level: 5808
** Updating Remote **
Setting LED to: 100
Setting Frequency to: 500
*** Waiting for Data.....

COM22 9600 8-N-1 FLOW:NONE Rx: 1593 bytes
```

Figure 5-12: Scheduled Base Monitoring in Terminal

### Separating Update Times on Remotes

In our code, we simply wait five seconds between updates. If you have multiple Remotes with identical code and they are powered up at once, they will all be on the exact same schedule and may cause issues with updates as they all send at once. By energizing at separate times, it will help ensure updates are separated. In the code you may change the delay before the main loop to be based on something unique if all are powered up at once, such as:

```
XB.Delay(1000 + (DL_Addr * 500))
```

## 5: Network Topologies & Control Strategies

---

This will help offset the transmission by Remote nodes. The XBee itself has a setting, Random Delay Slot (RN) to help randomize its “back-off and retry” algorithm so that the modems themselves are not locked in sync and having problems sending data with the exact same retry times. Of course, updates may not be based on time but on some event taking place.

### **Summary**

The XBee, using the IEEE 802.15.4 protocol, ensures data is passed between XBee modules efficiently and correctly at the data link layer. It is up to the programmer to initiate a scheme to ensure that data communications between the applications are also efficient and as robust as possible. Whether performing point-to-point or point-to-multipoint communications, the transmitter and receiver need to have their code coordinated for the application at hand. Some possible schemes include polling and scheduling of communications.

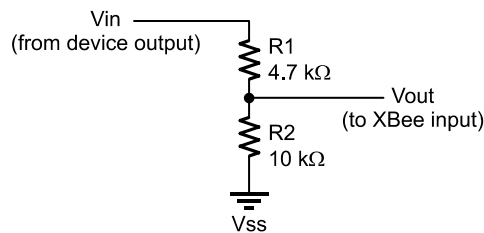
## 6: Sleep, Reset & API Mode

In this chapter we will explore using Sleep and Reset with the XBee, and also dig into API (Application Programming Interface) mode, where data is sent as frames between the controller and the XBee modem. While allowing greater data handling abilities, API Mode can be difficult due to the requirements of data framing.

### Interfacing 5 V Input to the XBee Module

With the XBee being a 3.3 V device, it is important to limit the voltage input to that level or damage may occur to the device. As discussed in Chapter 2, many of the communications and control lines to the XBee are buffered on the XBee 5V/3.3V Adapter and the XBee SIP Adapter. However, the general I/O lines are not. Whether working with the BASIC Stamp or 5 V devices with the Propeller chip, it may be necessary to reduce the input voltage to around the 3.3 V level. With digital I/O, as long as the voltage is near or lower than 3.3 V but high enough to be seen as a digital high, a precise voltage is not important.

The voltage divider in Figure 6-1 may be used to reduce a 5 V digital output to a safe level for an XBee input based on the formula of  $V_{out} = V_{in} \times (R2)/(R1+R2)$ . While this provides a voltage output of 3.4 V, it is within the tolerance of the XBee. The reason for choosing these values is that the 10 k $\Omega$  (R2) and 4.7 k $\Omega$  (R1) resistors are fairly popular sizes. To lower the voltage more, the size of R1 may be increased slightly or R2 decreased.



**Figure 6-1: Voltage Divider for Digital 5 V to 3.3 V interfacing**

The XBee has analog inputs as well, which we will explore. With the analog-to-digital converters (ADC) on the XBee, a reference voltage is used to set the upper limit of the analog resolution. This reference voltage should not exceed the supply voltage of the XBee and is typically set to the supply voltage of 3.3 V. Analog devices that have an output which exceeds 3.3 V should be used with a voltage-limiting circuit, to prevent potential damage to the XBee.

Limiting or dividing analog voltages can be a little tricky, and sensors within the operating limits are preferable. When a suitable sensor is not available, a simple method is to employ the voltage divider as shown in Figure 6-1. However, this may lead to issues of loading down the sensor's output, which is based on the output capability of the device and its output impedance (similar to resistance); the voltage measured at the ADC may be lower than the sensor's actual output. Larger resistors can aid in limiting the effect of loading, such as using 47 k $\Omega$  for R1 and 100 k $\Omega$  for R2 instead, but a point can be reached where the interface between the voltage divider and the ADC's input can cause loading effects.

A simple test to determine if loading is causing issues is to connect the sensor through the voltage divider, and then measure the sensor's output voltage and the voltage into the XBee's analog input. Using the voltage divider formula, calculate and compare the expected voltage to the actual. With devices that have variable resistance, such as a potentiometer, this should be checked at low and high values.

### Using Sleep Modes


With a current draw of around 50 mA while idle, the XBee is a major draw of current, reducing battery life in a battery-powered system. The XBee has available two major sleep modes to reduce the current consumption, but the XBee cannot receive RF or serial data while sleeping. Planning an appropriate network scheme is important, such as using scheduled events controlled by battery powered Remote nodes with a continually powered Base node.

There are two types of sleep modes: the first consists of using the SLEEP\_RQ input pin to control the sleep status, and the second are the cyclic sleep modes where the XBee automatically cycles between being awake and asleep. Table 6-1 is from the XBee manual providing an overview of the sleep modes (**SM** command). By default, the XBee **SM** parameter is 0 for “No Sleep” and the XBee is always awake and ready to receive data. By changing the **SM** parameter, the sleep mode can be changed.

**Table 6-1: XBee Sleep Modes from XBee Manual**

Mode Setting	Transition into Sleep Mode	Transition Out of Sleep Mode (Wake)	Characteristics	Related Commands	Power Consumption
Pin Hibernate (SM = 1)	Assert (high) Sleep_RQ (pin 9)	De-assert (low) Sleep_RQ	Pin/Host-controlled / NonBeacon systems only / Lowest Power	(SM)	<10 $\mu$ A (@3.0 VCC)
Pin Doze (SM = 2)	Assert (High) Sleep_RQ (pin 9)	De-assert (low) Sleep_RQ	Pin/Host-controlled / NonBeacon systems only / Fastest wake-up	(SM)	< 50 $\mu$ A
Cyclic Sleep (SM = 4)	Automatic transition to Sleep Mode as defined by the SM (Sleep Mode) and ST (Time before Sleep) Parameters	Transition occurs after the cyclic sleep time interval elapses. The time interval is defined by the SP (Cyclic Sleep Period) parameter.	RF module wakes in pre-determined time intervals to detect if RF data is present / When SM = 5	(SM), SP, ST	< 50 $\mu$ A when sleeping
Cyclic Sleep (SM = 5)	Automatic transition to Sleep Mode as defined by the SM (Sleep Mode) and ST (Time before Sleep) parameters or on a falling edge transition of the SLEEP_RQ pin	Transition occurs after the cyclic sleep time elapses. The time interval is defined by the SP (Cyclic Sleep Period) parameter.	RF module wakes in pre-determined time intervals to detect if RF data is present. Module also wakes on a falling edge of SLEEP_RQ.	(SM), SP, ST	< 50 $\mu$ A when sleeping

Typically, Pin Hibernate Mode is used to reduce power consumption. This allows the controller to control the sleep status of the XBee, waking it when needs to communicate and leaving it asleep when not. Once in mode 1 or 2 (**SM** = 1 or **SM** = 2), a high on the SLEEP\_RQ pin will cause the XBee to enter sleep once all data has been processed by the XBee. This is important in that the XBee will not enter any sleep mode unless its buffers are empty.

	<p><b>Sleep modes and RTS</b></p> <p>An XBee will not enter any sleep mode unless the serial buffer is empty. If using RTS this could require extra steps, but in testing it appears that using sleep and RTS are not compatible—once having slept and awoken, RTS appears to be disabled. We don't recommend using sleep while requiring RTS for flow control at this time.</p>
---	--

On the BASIC Stamp, use Command Mode to enable a Pin Sleep Mode (**SM** = 1 or **SM** = 2):

```
SEROUT Tx, Baud, [ "ATSM 1",CR]
```

If you are using a Propeller chip, here is a Spin example using `XBee_Object.spin` that does the same thing:

```
XB.AT_Config("ATSM 1")
```

Use a controller output to bring the `SLEEP_RQ` pin high for placing the XBee into a sleep mode, and set the pin low to wake the XBee module. Allow a short pause of around 10 ms after awaking the XBee before sending any data. It is a good idea to bring this pin **LOW** early in your code. Otherwise, when the controller resets, the XBee module will still be in sleep mode and may miss critical configuration changes early in your code. Placing the pin low will ensure the module is awakened early in your code and will catch any configuration changes.



### Adapter Pin Labeling & Signal Conditioning

The `SLEEP_RQ` pin is multifunction as `DTR/SLEEP_RQ/DI8`. On the XBee USB Adapter, XBee Adapter and XBee 5V/3.3V Adapter it is labeled as `DTR`. On the XBee SIP Adapter, it is labeled as `SLP` on the 5-pin female header. For use with the BASIC Stamp, it is conditioned to 5 volts on the XBee SIP Adapter and the XBee 5V/3.3V Adapter Rev B.

## Using Reset

Since the XBee module does not reset when the controller does, the XBee may not be in the correct state to receive configuration changes in your code. For example, you may perform configuration settings before changing baud rate or switching to API Mode. When your controller resets manually (not due to power cycling) or from a PC due to a program download, the XBee would still be in the last configuration. By using the reset pin, you can perform a reset of the XBee so that it is in the default configuration or the last saved change to the configuration.

The XBee resets with a **LOW** on the Reset pin. Bringing this pin **LOW** momentarily at the start of your code ensures the XBee is in the configuration expected. Another option is to tie this pin to the reset pin of your controller so that both reset at the same time.



### Adapter Signal Conditioning

The XBee SIP Adapter and XBee 5V/3.3V Adapter Rev B use a diode to the reset pin, cathode to controller I/O, to allow this pin to be brought low while still allowing the XBee to bring the pin low itself.

### API Programming and Uses

In API Mode (Application Programming Interface), the data between the XBee and the controller is framed with essential information to allow communications, configuration changes, and special XBee functions. Once in API Mode, data is no longer simply sent and received transparently, but must be framed for reception by the XBee and parsed from a frame sent from the XBee. Header fields containing information needed for delivery and use of the data are added to the message sent. The controller data is encompassed, or packaged, with these headers. The entire package is a single frame of data. On transmission the headers are added to the data sent to form the frame. On reception, the headers are read and used by the controller.

Benefits and features of using API Mode include:

- Ensuring data sent from the controller to the XBee is free of errors.
- Allowing controller code to verify data from the XBee to the controller is error free (error checking not currently implemented in XBee\_Object.spin).
- Building a frame for transmission with necessary information such as destination node address.
- Ability to configure the XBee without flipping into Command Mode.
- Reception of data from another node in a frame containing the source node's address and RSSI level of received data.
- Acknowledgement frames to ensure destination nodes received data and configuration information.
- Special XBee functions such as receiving analog and digital data from another node, remotely configuring a node, and line-passing (inputs to one XBee control outputs on another XBee).

The XBee manual illustrates how frames are constructed for transmission to and from the XBee. Each frame type has a unique identifier called the API Identifier. For example, to transmit data to another XBee by its 16-bit address (the **DL** address we normally use), the API identifier is: 0x01 (hexadecimal) or 1. The frame format is shown in Figure 6-2.

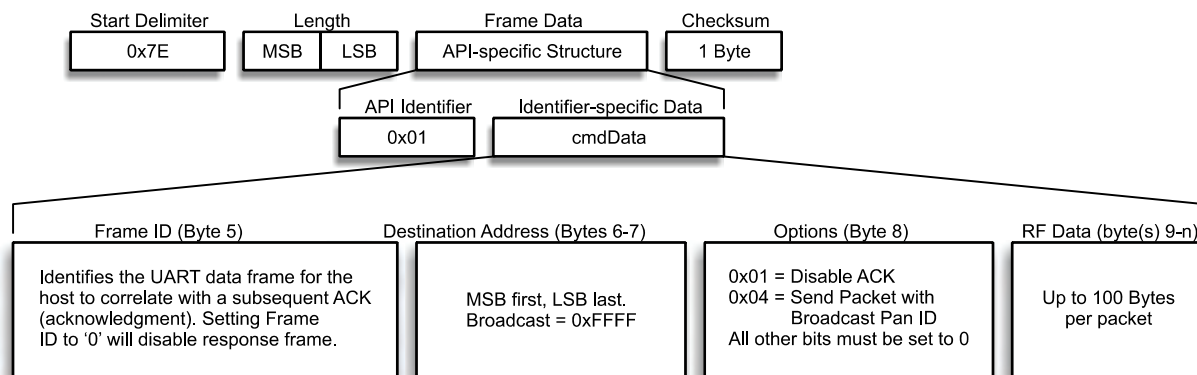


Figure 6-2: API Frame Format for Data Transmission to 16-bit Address



Breaking down the frame:

- **Start Delimiter:** All frames start with 0x7E (\$7E) to mark the start of the frame.
- **Length** (2 bytes) is the number of bytes from the next byte up to but not including the checksum. It is broken up over 2 bytes (MSB and LSB) for lengths over 255 bytes.
- **Frame Data** includes:
  - **API Identifier:** The unique value identifying the function of the frame.
  - **cmdData:** Data to be transmitted made up of:
    - **Frame ID:** Numbers the frame to allow acknowledgements to be correctly identified as belonging to a specific transmission. A value of 0 will return no acknowledgement.
    - **Destination Address** (2 bytes): The 16-bit address to which to send data. Set to \$FFFF for broadcast address.
    - **Options:** Allows special functions and indications on reception such as to not send an acknowledgement.
    - **RF Data** (up to 100 bytes): The actual data to be transmitted.
- **Checksum:** All byte values summed together from after the length byte up to the checksum byte, and subtracted from \$FF.

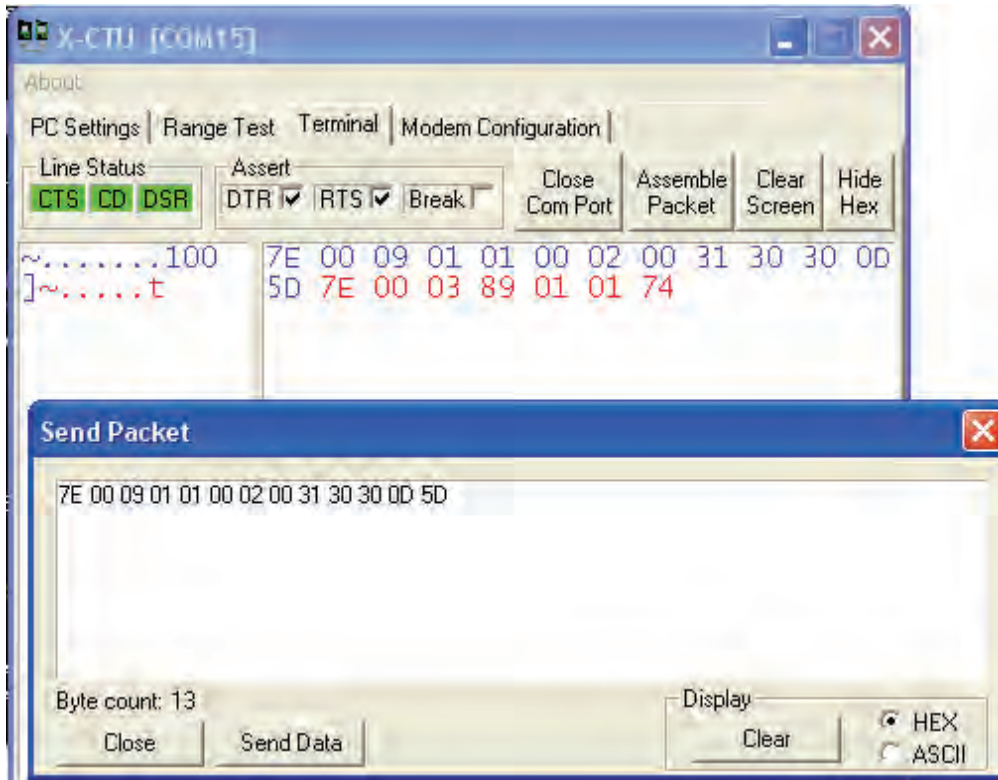
Let's say we wanted to transmit the decimal value of 100 as ASCII values, that is a string of "1", "0" and "0", followed by a carriage return to be sent to a node at address 2. The bytes for the frame would be as shown in Table 6-2.

**Table 6-2: API Frame Values to Transmit Decimal 100**

Frame Element	Value	Function
Start Delimiter	\$7E	
Length MSB	\$00	Number of bytes from API Identifier up to Checksum
Length LSB	\$09	
API Identifier	\$01	
Frame ID:	\$01	
Dest. Addr. MSB	\$00	Destination node address (\$0002)
Dest. Addr. LSB	\$02	
Options	\$00	Send an acknowledgement
Data	\$31	ASCII character "1"
Data	\$30	ASCII character "0"
Data	\$30	ASCII character "0"
Data	\$0D	ASCII Carriage Return (decimal 13)
Checksum	\$5D	(\$FF - (\$01+\$01+\$00+\$02+\$00+\$31+\$30+\$30+\$0D))

## 6: Sleep, Reset & API Modes

If any byte in the frame sent to the XBee is incorrect (the checksum doesn't pass verification), it will not be accepted. If correct, the XBee will repackage the data for the 802.15.4 protocol and transmit to node \$02. Figure 6-3 shows manually framing the packet for transmission using the Assemble Packet window of the X-CTU terminal.



**Figure 6-3: Manually Constructing an API Frame**

Notice that after the last byte, more data was returned by the XBee. This is a transmit delivery acknowledgement frame (API Identifier \$89). The last 01 indicates that delivery failed—we had no node listening at address \$2. If you wish to test this yourself, try to send some data manually in API Mode:

- ✓ With an XBee connected to USB, place the XBee in API Mode (**ATAP 1**) using the Modem Configuration tab or using Command Mode.
- ✓ On the PC Settings tab, select Enable API to ensure you can change configurations when you need to.
- ✓ In the Terminal tab, select Show Hex and open the Assemble Packet window.
- ✓ Select the HEX button.
- ✓ Enter the hexadecimal values and press Send Data.
- ✓ You should get an acknowledgement—if you do have another XBee at address 2, it should receive the data of 100.
- ✓ Modify the values to send data to node 0 instead.
- ✓ Try bad values—you should get no acknowledgement frame back because the packet was not accepted and therefore the acknowledgement not transmitted.
- ✓ Return the XBee to AT Mode by setting **ATAP** to 0 and deselecting Enable API on PC Settings.



### AT & API Communications

Note the XBee modules DO NOT need to be in the same mode to communicate. An XBee at address \$2 in AT Mode will simply show the text of 100 was received. API Mode is for communications between the controller and the XBee, not between two XBee modules. XBee to XBee communications are also framed, using the IEEE-802.15.4 protocol, but we never see it or work directly in it.

When in API Mode, data received is similarly framed for reception by the controller to error check (if desired), parse the fields, and pull out data. Figure 6-4 is the API frame for Receive Packet frame showing the source's address, RSSI level and other fields that may be used by the receiving unit.

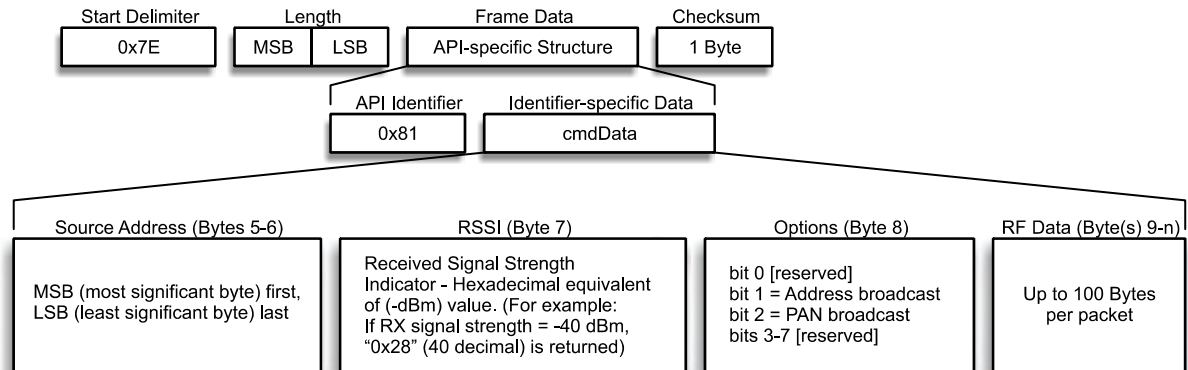


Figure 6-4: API Frame Format for Data Reception by 16-bit Address

Please see the XBee manual for other frame types discussed in this chapter.

### Programming for API Mode

For the Propeller, the XBee\_Object.spin file provides the ability to send framed data of various types and to accept and parse framed data of various types. The following code demonstrates the method to send a string such as "100" from the Propeller using API Mode. The array of dataSet[] is constructed of the required bytes and data, the checksum calculated, and the entire frame is transmitted.

```

Pub API_Str (addy16,stringptr) | Length, chars, csum,ptr
{{
  Transmit a string to a unit using API Mode - 16 bit addressing
  XB.API_Str(2,string("Hello number 2"))      ' Send data to address 2
  TX response of acknowledgement will be returned if FrameID not 0
  XB.API_RX
  If XB.Status == 0 '0 = Acc, 1 = No Ack
}}

ptr := 0
dataSet[ptr++] := $7E
Length := strlen(stringptr) + 5 ' API Ident + FrameID + API TX cmd +
                                ' AddrHigh + AddrLow + Options

dataSet[ptr++] := Length >> 8   ' Length MSB
dataSet[ptr++] := Length        ' Length LSB
dataSet[ptr++] := $01           ' API Ident for 16-bit TX
dataSet[ptr++] := _FrameID     ' Frame ID
dataSet[ptr++] := addy16 >>8   ' Dest Address MSB
dataSet[ptr++] := addy16       ' Dest Address LSB
dataSet[ptr++] := $00         ' Options ' $01 = disable ack,
                                ' $04 = Broadcast PAN ID

```

## 6: Sleep, Reset & API Modes

---

```
Repeat strsize(stringptr)          ' Add string to packet
  dataSet[ptr++] := byte[stringptr++]
csum := $FF                          ' Calculate checksum
Repeat chars from 3 to ptr-1
  csum := csum - dataSet[chars]
dataSet[ptr] := csum

Repeat chars from 0 to ptr
  tx(dataSet[chars])                ' Send bytes to XBee
```

On reception of data for the Propeller, the API Identification number of the received data is checked and based on it the fields and data for the received frame are parsed.

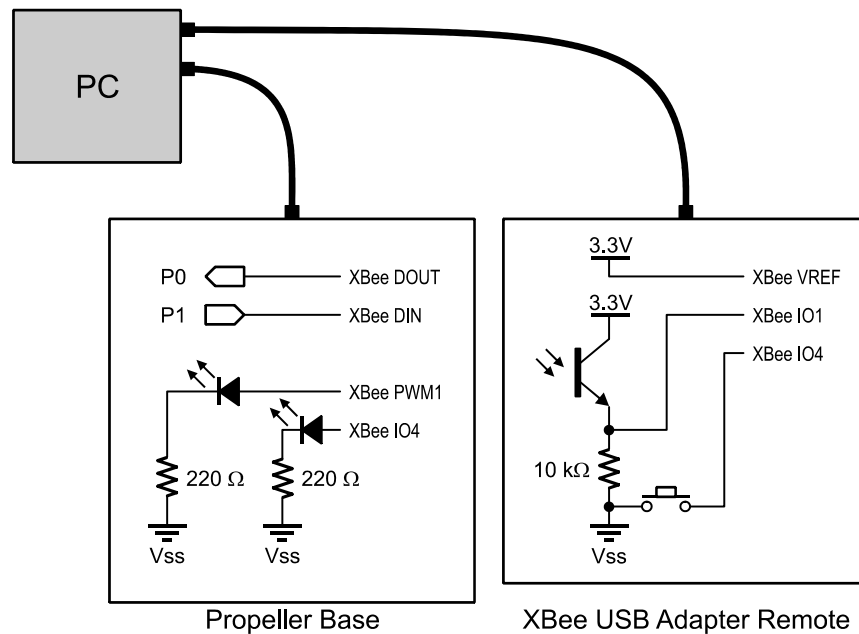
```
Pri RxPacketNow | char, ptr, chan
{{
  Process incoming frame based on Identifier
  See individual cases for data returned.
  Check ident with :
  IF XB.rxIdent == value
    and process data accordingly as shown below
}}

ptr := 0
Repeat
  Char := rxTime(1)                  ' accept remainder of data
  dataSet[ptr++] := Char
while Char <> -1
  ptr := 0
  _RxFlag := 1
  _RxLen := dataSet[ptr++] << 8 + dataSet[ptr++]
  _RxIdent := dataSet[ptr++]
  case _RxIdent
    $81:  ' ***** Rx data from another unit packet
          ' Returns:
          ' XB.scrAddr      16bit addr of sender
          ' XB.RxRSSI      RSSI level of reception
          ' XB.RXopt       See XB manual
          ' XB.RxData      Pointer to data string
          ' XB.RXLen       Length of actual data
          _srcAddr16 := dataSet[ptr++] << 8 + dataSet[ptr++]
          _RSSI := dataSet[ptr++]
          _RXopt := dataSet[ptr++]
          bytefill(@_RxData, 0, 105)
          bytemove(@_RxData, @dataSet+ptr, _RxLen-5)
          _RxLen := _RxLen - 5
```

The BASIC Stamp doesn't have the large amount of RAM or processing power of the Propeller chip. We will show some sample code that can be used for some API testing, but it requires manually doing much of the work and filling in some values for use. In general though, for basic transmission and reception of data, AT Mode where the destination address is set and the RSSI level polled using **ATDB**, as prior examples have used, is perfectly fine for use.

### Propeller XBee Object API Interfacing

In this section we will explore API interfacing using `XBee_Object.spin` (version 1.6 or higher). The top object, `API_Base.spin`, will be used to communicate to a local XBee using API Mode for a variety of tasks. The initial configuration shown in Figure 6-5 has a Propeller-interfaced XBee as the Base node, and a USB-connected XBee as the Remote node assigned to address 2. USB is used to both configure and monitor the Remote node's XBee and to power the unit. The program `API_Base.spin` will illustrate how the Base node's `XBee_Object` code is sending, receiving and using data from the Remote node's XBee.



**Figure 6-5: Propeller API Testing Configuration**

- ✓ Connect the hardware as shown (the I/O devices will be used later).
- ✓ Using an open instance of X-CTU, set Remote node, the USB-Adapter connected XBee, to a **MY** of 2.
- ✓ Have an instance of the X-CTU interface open for monitoring data communications with the Base node, the Propeller connected XBee.
- ✓ Using F11, download API\_Base.spin to your Propeller Base node, then open its terminal for display.
- ✓ A menu should be shown in the terminal. If not shown, reset your Propeller.

### Transmitting a Byte

The first task we want to explore is simply sending and receiving characters that could represent individual byte values sent from the Base node to the Remote node. Keep in mind that when transmitting individual bytes, each byte is sent as an individual packet for reception. Sending a collection of bytes in a single transmission takes a little more work that we will explore later.

- ✓ Choice 1 in the API\_Base.spin menu is to Send Character. Press 1 to select.
- ✓ Set the number of the destination node for the character—Enter 2.
- ✓ Type a character to be transmitted.

If all goes well, the results should look similar to Figure 6-6. In the Remote nodes terminal window (right), we can see the character displayed. In the Base node’s terminal, we have received notification that the transmission was successful. A transmission acknowledgement frame was received, indicating that our data was accepted with a status of “acknowledged.”

## 6: Sleep, Reset & API Modes

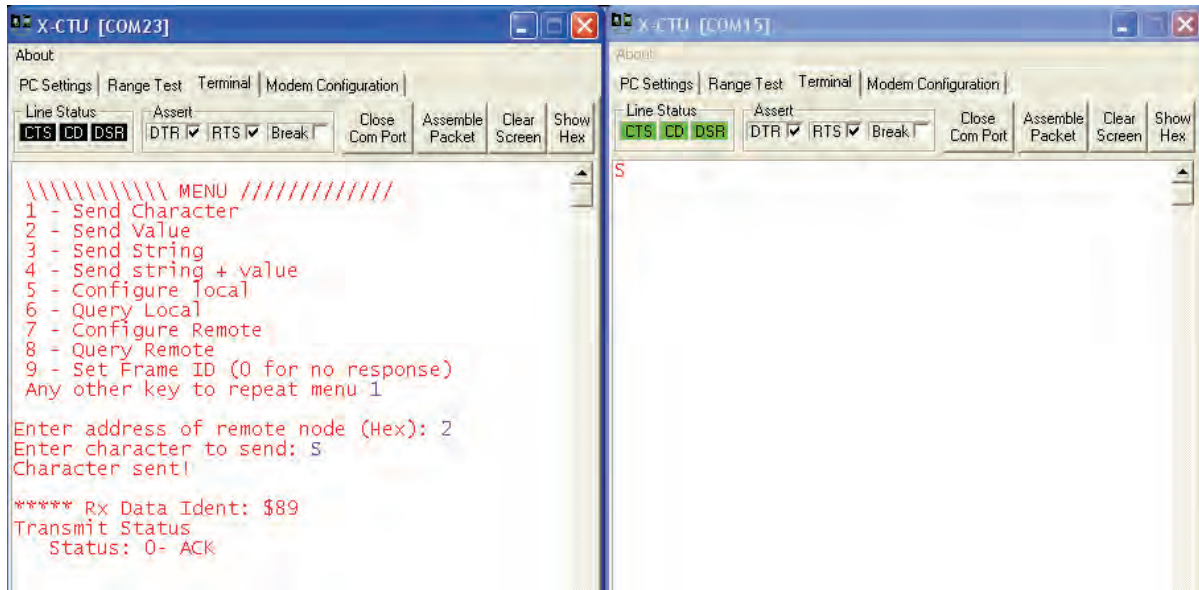


Figure 6-6: Using API\_Base.spin to Transmit a Character

- ✓ Send another character to a node that is not present and see that you receive notification that it was not acknowledged.

The code to accept and send your menu choice and data for transmission is as follows:

```
Case choice
  "1":
    PC.str(string(13,13,"Enter address of Remote node (Hex): "))
    Addr := PC.RxHex
    PC.str(string("Enter character to send: "))
    Value := PC.Rx
    XB.API_Tx(Addr, Value)
    PC.str(string(13,"Character sent!"))
    XB.Delay(2000)
```

The method call `XB.API_Tx(addr, value)` sends the address and byte value to the `XBee_Object`, so it can create a transmit packet properly framed for reception and use by the Base XBee, which will then repackage and send the data to the Remote XBee.

In a separate cog, the Propeller using the XBee object monitors for incoming data frames. Based on any incoming frames, the API Identification value (`XB.RxIdent`) defines what type of frame was received.

```
repeat
  XB.API_Rx
  PC.Str(string(13,13,"***** Rx Data Ident: $"))
  PC.Hex(XB.RxIdent,2)
  case XB.RxIdent
```

When the Remote node's XBee acknowledges reception to the Base node's XBee, the Base XBee sends a transmit status frame to the Propeller. `XBee_Object` accepts the frame and parses it, and then `API_Base.spin` displays the information.

```

s89:          Tx Status
PC.str(string(13,"Transmit Status"))
PC.str(string(13," Status: "))
PC.Dec(XB.Status)
Case XB.Status
0: PC.str(string("- ACK"))
1: PC.str(string("- NO ACK"))
2: PC.str(string("- CCA Failure"))
0: PC.str(string("- Purged"))

```

The status of a transmission can be: an acknowledgement; no acknowledgement; CCA failure when the RF levels are too high to transmit; or it could be purged in certain cases.

- ✓ Use Set Frame ID (choice 9) to change the frame to 0 and send another character. Note that using a frame of 0 disables acknowledgement frames from being used, which can minimize the data we need to process.
- ✓ Enter a Frame value of a higher value to turn acknowledgements back on.

### Transmitting a Decimal Value

In order to send a decimal value such as “1234” it must be converted to a string using the number.spin object. This allows the XBee to accept the string using the XB.API\_Str method.

- ✓ Use choice 2 of the menu to Send Value.
- ✓ Enter the address of the Remote node: 2.
- ✓ Enter a decimal value and press Enter.
- ✓ Verify the value is shown on the Remote’s terminal and an “Ack” was received.

In the code, the address and address are accepted. The value is converted to a string (using the object “numbers.spin”) and XB.API\_Str is used to send the values for framing and transmission.

```

"2":
PC.str(string(13,13,"Enter address of Remote node (Hex): "))
Addr := PC.RxHex
PC.str(string("Enter Value to send (Decimal): "))
Value := PC.RxDec
XB.API_str(Addr, num.toString(Value,num#dec))
PC.str(string("Value sent!"))
XB.Delay(2000)

```

### Sending String Data

To send a string of text (choice 3), the code builds an array to transmit, consisting of up to 100 characters, or terminating with a carriage return (Enter) and sent by calling the XB.API\_Str method as well.

```

"3":
PC.str(string(13,13,"Enter address of Remote node (Hex): "))
Addr := PC.RxHex
PC.str(string("Enter string to send: "))
ptr := 0
repeat
  strIn[ptr++] := PC.Rx
while (strIn[ptr-1] <> 13) and (ptr < 100)
  strIn[ptr-1] := 0
XB.API_str(Addr, @strIn)
PC.str(string("String Sent!"))
XB.Delay(2000)

```

## 6: Sleep, Reset & API Modes

---

Of course, a constant string can be sent using the Remote address and string function:

```
XB.API_Str(2, string("Hello World!"),13))
```

### **Transmitting Multiple Data**

Choice 4 allows us to combine text and decimal values. You may test by entering the necessary information. Remember, if we want this data to be sent in a single transmission, the individual data must be assembled before the packet is ready to be framed and transmitted. This requires adding individual data that will be used to construct an array to eventually be transmitted. The code for choice 4 demonstrates assembling multiple data for one transmission.

```
"4":
PC.str(string(13,13,"Enter address of Remote node (Hex): "))
Addr := PC.RxHex
PC.str(string("Enter string to send: "))
ptr := 0
repeat
  strIn[ptr++] := PC.Rx
while (strIn[ptr-1] <> 13) and (ptr < 100)
strIn[ptr-1] := 0
PC.RxFlush
PC.str(string("Enter Value to Send (Decimal): "))
Value := PC.RxDec
XB.API_NewPacket
XB.API_AddStr(@strIn)
XB.API_AddStr(num.toString(Value,num#dec))
XB.API_AddByte(13)
XB.API_str(Addr,XB.API_Packet)
PC.str(string("String & Value Sent!"))
XB.Delay(2000)
```

After the data is accepted, `XB.API_NewPacket` is called to create a new packet of data. An array of data is assembled using both `XB.API_AddStr` and `XB.API_AddByte`. Once the packet is constructed, `XB.API_Str` is called to transmit the data, using `XB.API_Packet` as a pointer to the data.

Note that `API_Str` expects valid string data ending with a terminating 0 (done by filling the `API_Packet` with all 0's before use when `XB.NewPacket` is called). If data containing a 0 needs to be sent, the `XB.API_txPacket` method must be used instead providing the number of bytes to send (size):

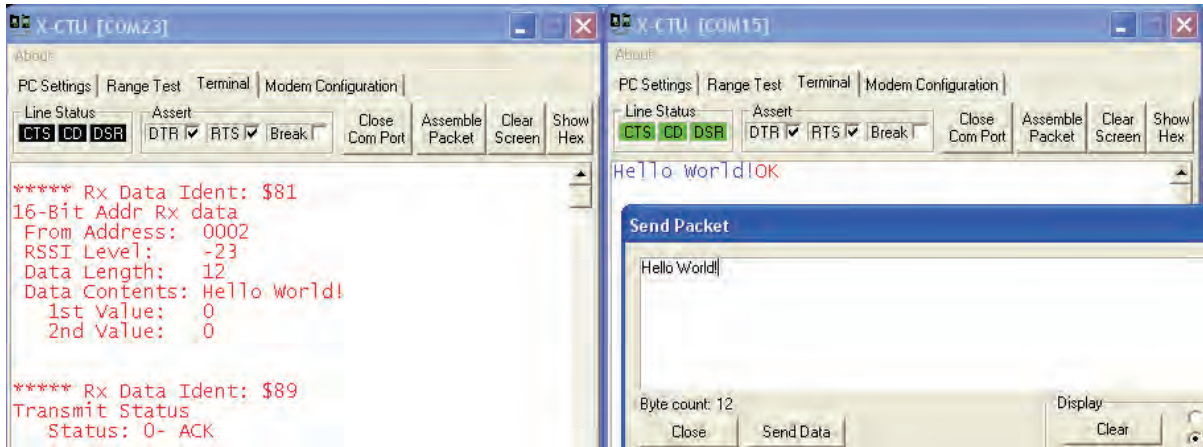
```
XB.API_TxPacket(addr, pointer, size)
```

### **Receiving Data**

As data is received by the Base node's XBee, it is placed into a frame for delivery to the Propeller. `API_Base.spin` will demonstrate ways of pulling out the incoming data for use, as well as use of other information in the received frame.

- ✓ In the Remote node's terminal, select Assemble Packet to open the Send Packet window, enter "Hello World!" and send the data. The Send Packet window is used to ensure the entire string is sent in one transmission instead of a transmission for each character.
- ✓ Your results should look similar to Figure 6-7.





**Figure 6-7: Receiving String of Data by API\_Base.spin**

On receiving a data packet, the frame is broken down by code in `XBee_Object`. The `API_Base` object shows pertinent data received, including the API identification, the source address, the RSSI level of the received packet, the length of the actual data, the data itself, and any values sent (shown in the next example). In this example, the most important piece of data is `XB.RxData` which points to an array holding the incoming data.

```

$81:
      ' 16-Bit Rx Data
      PC.str(string(13,"16-Bit Addr Rx data"))
      PC.str(string(13," From Address:  "))
      PC.hex(XB.srcAddr,4)
      PC.str(string(13," RSSI Level:  "))
      PC.Dec(-XB.RxRSSI)
      PC.str(string(13," Data Length:  "))
      PC.dec(XB.RxLen)
      PC.str(string(13," Data Contents: "))
      PC.Str(XB.RxData)
      PC.Str(string(13," 1st Value:  "))
      Value := XB.ParseDec(XB.RxData,1)
      PC.Dec(Value)
      PC.Str(string(13," 2nd Value:  "))
      Value := XB.ParseDec(Xb.RxData,2)
      PC.Dec(Value)
      XB.API_str(XB.srcAddr,string("OK"))

```

Also note that an “OK” is shown on the Remote node’s terminal and the Base node’s terminal shows the status of transmission of the “OK.” The address parsed from the received data is used to reply back to the Remote node. Now let’s send some numeric values as data:

- ✓ Clear the old data and enter 100,200 in the Remote node’s Send Packet window and send data.
- ✓ On the Base node, you should see 100 and 200 for 1<sup>st</sup> value and 2<sup>nd</sup> value respectively, along with the other data.
- ✓ The XBee object can take a string, and using commas and carriage returns as delimiters, parse decimal values from strings. For example, this line pulls the 2<sup>nd</sup> decimal value from a string of text held in `XB.RxData`:

```
XB.ParseDec(XB.RxData,2)
```

## 6: Sleep, Reset & API Modes

---

### Local Configuration

While in API Mode, configuration information is also sent for use by the XBee and it will be rejected if not properly framed. While AT Mode may still be used, using API Mode eliminates the need of popping in and out of Command Mode. Choice 5, Configure Local, allows entering the parameter name and value for configuring the local XBee.

- ✓ Select choice 5.
- ✓ Enter D5 for configuration parameter (association indication output).
- ✓ Enter 0 for value (disable indicator).
- ✓ If you have the LED connected to IO4, use D4 and values of 5 and 4 to turn on and off.

If available, the association LED should stop blinking on the Base node's XBee adapter. The code of `XB.API_Config(str, value)` is used to have `XBee_Object` package a frame for delivery to the local XBee.

```
"5":
    PC.str(string(13,13,"Enter 2-letter Command Code: "))
    strIn[0] := PC.rx
    strIn[1] := PC.rx
    strIn[2] := 0
    PC.str(string(13,"Enter Value (Hex): "))
    Value := PC.RxHex
    XB.API_Config(@StrIn, value)
    PC.str(string("Configuration Sent!"))
    XB.Delay(2000)
```

Notice in the terminal window an acknowledgement is received from the XBee configuration. The API Identification code of \$88 shows the frame was received with data of the command sent (D5). The value returned is valid on a query only, not a configuration change. In normal code (non-interactive), the following code may be used to send configuration information:

```
XB.API_Config(string("D5"),0)
```

### Local Query of Configuration Value

You may also have the XBee Object query a setting on the local XBee by using choice 6 and entering a parameter to query, such as ID for PAN ID. The `API_Base` object will send the query and display the returned data parsed by `XBee_Object` using the API frame identification of \$88 once again. Code to send data:

```
"6":
    PC.str(string(13,13,"Enter 2-letter Command Code: "))
    strIn[0] := PC.rx
    strIn[1] := PC.rx
    strIn[2] := 0
    XB.API_Query(@StrIn)
    PC.str(string(13,"Query Sent!"))
    XB.Delay(2000)
```

Returned frame data from XBee Object:

```
$88:
      Command Response
    PC.str(string(13,"Local Config/Query Response"))
    PC.str(string(13," Status:          "))
    PC.Dec(XB.Status)
    Case XB.Status
      0: PC.str(string(" - OK"))
      1: PC.str(string(" - Error"))
```

```

2: PC.str(string(" - Invalid Command"))
3: PC.Str(string(" - Invalid Parameter"))
PC.str(string(13," Command Response"))
PC.str(string(13," Command Sent: "))
PC.str(XB.RxCmd)
PC.str(string(13," Data (valid on Query):"))
PC.hex(XB.RxValue,4)

```

### **Remote XBee Configuration**

If you have firmware 10CD or higher installed on the XBee modules, you can also configure the parameters on a Remote node! This can be used to change normal parameters, or to control an output on a Remote node—such as D5, the association LED, or any other digital output you desire along with the PWM outputs.

- ✓ Monitor the Association LED on the Remote node.
- ✓ Enter choice 7.
- ✓ Enter 2 for the Remote's address.
- ✓ Enter D5 for the association LED.
- ✓ Enter 4 to turn it off and verify.
- ✓ Repeat, using a value of 5 to turn it on.

```

"7":
PC.str(string(13,13,"Enter address of Remote node (Hex): "))
Addr := PC.RxHex
PC.str(string(13,"Enter 2-letter Command Code: "))
strIn[0] := PC.rx
strIn[1] := PC.rx
PC.str(string(13,"Enter Value (Hex): "))
Value := PC.RxHex
XB.API_RemConfig(Addr,@StrIn, value)
PC.str(string(13,"Remote Configuration Sent!"))
XB.Delay(2000)

```

Notice that confirmation of configuration information is returned and parsed including the address, status of configuration change, and parameter set.

### **Remote XBee Query**

Using choice 8, the Remote node's XBee may be queried for current settings and values. Note that you cannot read the value of a digital or analog input, only its configuration setting. Use choice 8 to enter a Remote address (2) and a value to query.

Code to send query:

```

"8":
PC.str(string(13,13,"Enter address of Remote node (Hex): "))
Addr := PC.RxHex
PC.str(string(13,"Enter 2-letter Command Code: "))
strIn[0] := PC.rx
strIn[1] := PC.rx
XB.API_RemQuery(Addr, @StrIn)
PC.str(string(13,"Remote Query Sent!"))
XB.Delay(2000)

```

## 6: Sleep, Reset & API Modes

---

Code to show parsed data:

```
§97:
PC.str(string(13,"Remote Query/Configuration"))
PC.str(string(13," From Address:   "))
PC.hex(XB.srcAddr,4)
PC.str(string(13," Status:       "))
PC.Dec(XB.Status)
Case XB.Status
  0: PC.str(string(" - OK"))
  1: PC.str(string(" - Error"))
  2: PC.str(string(" - Invalid Command"))
  3: PC.Str(string(" - Invalid Parameter"))
  4: PC.str(string(" - No Response"))
PC.str(string(13," Command Response"))
PC.str(string(13,"  Command Sent:   "))
PC.str(XB.RxCmd)
PC.str(string(13,"  Data (valid on Query): "))
PC.hex(XB.RxValue,4)
```

### **Analog and Digital Data**

Analog and digital data may be sent as framed data from a one XBee to another reading the API frames if you are using firmware version 10A1 or higher. On the Remote node, the destination address (**DL**) is set, the various I/O's are configured for digital inputs or ADC, and **IR** is set for the sample rate. The Remote node will send data of the configured I/O at the set interval as a framed packet with bits and data identifying the transmitted data. The number of samples per transmission may be set using the **IT** setting (samples before transmit). XBee\_Object.spin only accepts one sample per transmission at this time.

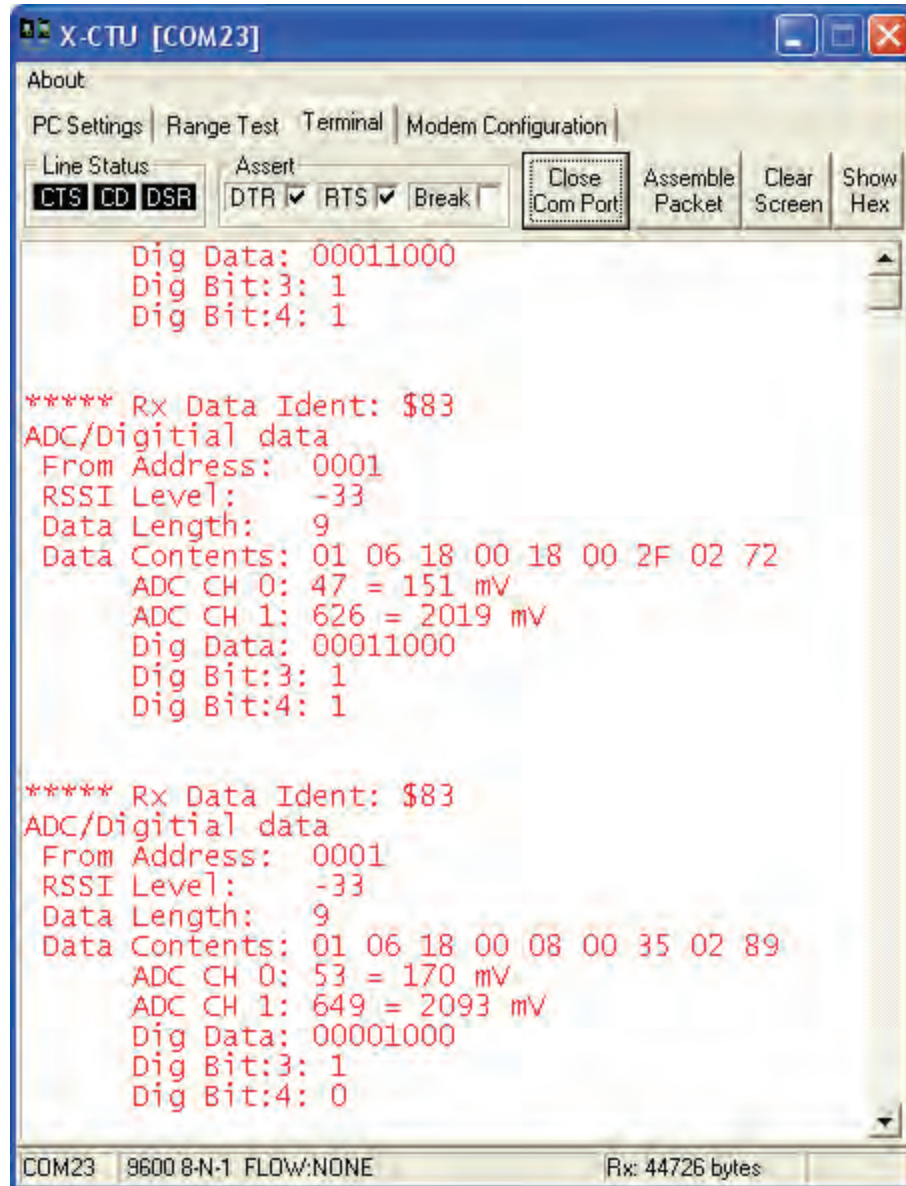
The XBee needs to be configured with appropriate I/O devices connected. Figure 6-5 on page 109 shows the XBee USB Adapter being used on a breadboard, and powered for convenience of configuration changes from USB. It is using the phototransistor as an analog input and a pushbutton as digital input. Note that the inputs on the XBee have internal pull-up resistors eliminating the need for pull-up or pull-down resistors in the configuration.

- ✓ Through the X-CTU Modem Configuration tab, configure the **Remote node's** USB XBee for the following:
  - **MY** to 1 (Remote XBee to address 1)
  - I/O Setting **D0** to ADC (2) (to read analog input)
  - I/O Setting **D1** to ADC (2) (to read analog input)
  - I/O Setting **D3** to DI (3) (to read digital input)
  - I/O Setting **D4** to DI (3) (to read digital input)
  - **IR** Setting to \$7D0 (2000 decimal to sample once every two seconds)
- ✓ Monitor the Propeller-connected Base node in X-CTU.
- ✓ Information similar to Figure 6-8 should be displayed.



#### **Using Remote Configuration for non-USB connected XBee**

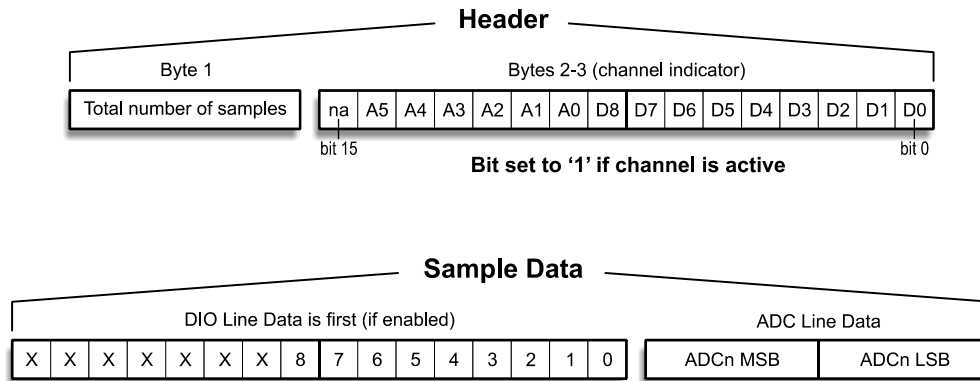
The Base XBee may also be used for configuration using menu Choice 7, Remote configuration to address 1. Use **WR** to store data in non-volatile memory of the Remote unit.



**Figure 6-8: Analog and Digital Testing Results**

Figure 6-8 shows the data sent from the Remote XBee module. The frame identification indicates it is analog and digital frame type (\$83) from address 0001. The RSSI level is displayed as well as the raw data contents in hexadecimal. Any active channel data is shown: the analog inputs on D0 and D1 as both decimal and converted to millivolts based on the 3.3 V reference. The full digital data is shown and the active digital inputs on D3 and D4. Note that per our hardware, we only have an analog input on D1 and a digital input on D4. The other data is sent simply for testing reading the data. From the XBee manual, the data sent for the frame is constructed as shown as in Figure 6-9.

## 6: Sleep, Reset & API Modes



**Figure 6-9: ADC and Digital Data Format**

The Header defines the number of samples in the transmission and sets any bit high for which analog or digital data is sent. The Sample Data contains the actual digital data as two bytes, and the MSB and LSB bytes (for 10-bit data, maximum value 1023) of each analog channel sent.

Looking at our actual data contents received for the last transmission:

- 01 = Number of samples: 1
- 06 = Analog channels A1 and A2 are active (Binary 0000\_0110)
- 18 = Digital channels D4 and D3 are active (Binary 0001\_1000)
- 00 = DIO line data is all 0's for upper byte (Only D8 and rest unused)
- 08 = DIO line data for D7 to D0, only D3 is high (Binary 0000\_1000)
- 00 & 35 = Analog channel 0 data (0035 converted to decimal = 53)
- 02 & 89 = Analog channel 1 data (0289 converted to decimal = 649)

The analog channels are converted to a voltage with the following equation to multiply received value by 3300 mV (3.3V = reference) and divide by 1023 (the maximum analog value):

$$\text{Data value} \times 3300 / 1023.$$

The XBee object reads this fairly complex data, analyzes it, and allows the top object to access the data in a manageable format for use. Should any channel read not contain enabled data, a value of -1 is returned.

```

s83:                                     ' ADC / Digital I/O Data
PC.str(string(13,"ADC/Digital data"))
PC.str(string(13," From Address:  "))
PC.hex(XB.srcAddr,4)
PC.str(string(13," RSSI Level:  "))
PC.Dec(-XB.RxRSSI)
PC.str(string(13," Data Length:  "))
PC.dec(XB.RxLen)
PC.str(string(13," Data Contents:  "))
repeat ch from 0 to XB.RxLen - 1         ' Loop thru each data byte
  PC.Hex(byte[XB.RxData][ch],2)         ' Show hex of data byte
  PC.Tx(" ")

```

```

repeat ch from 0 to 5
  If XB.rxADC(ch) <> -1
    PC.str(string(13,"      ADC CH "))
    PC.Dec(ch)
    PC.str(string(": "))
    PC.Dec(XB.rxADC(Ch))
    PC.str(string(" = "))
    PC.Dec(XB.rxADC(Ch) * 3300 / 1023)
    PC.str(string(" mV"))
    ' loop thru ADC Channels
    ' If not -1,
    ' data on channel
    ' display channel
    ' display channel data
    ' display in millivolts

```



#### Transmit LED?

The Tx LED on the USB Remote will not light when sending the analog/digital data. This LED is actually indicating data sent to the XBee for transmission (DIN), not that it is transmitting.

### Line-Passing

In line passing, the analog and digital inputs of one XBee are passed to another XBee to be directly used to control devices (or have the outputs read by controller) instead of the controller reading the packet. The same Analog/Digital packet is sent from the Remote node, but the values are used to control the Base node's I/O instead. D0 to D7 on the Remote will control D0 to D7 on the Base respectively. If analog values are read instead, an analog value on D0 or D1 can control PWM outputs on PWM0 (normally used for RSSI) and PWM1 respectively.

To configure the **Base node** for accepting Line Passing, use menu choice 5 for the following setting where specified on the Base node.

- ✓ Ensure the Remote node is configured for sampling analog and digital data from the previous section.
- ✓ Unplug the Remote from USB to stop data from being sent.
- ✓ Set IA to 1: Sets the address from which data will be used, for our example, the Remote node is at an address of 1 (FFFF may be used to accept data from any device).
- ✓ Set D4 to 4: Sets the direction of the I/O to be controlled to be an output low.
- ✓ Set P1 to 2: Sets PWM1 to output PWM.
- ✓ Set IU to 0: Disables the received analog/digital data frame from being sent to the Propeller chip (data is used for I/O control only). This value is normally 1.
- ✓ Connect the Remote node XBee to USB. Every 2 seconds, data should be sent from the remote node updating the hardware on the Base node. The button on the remote node should control the state of one LED on the Base node, and the light level at the Remote node's phototransistor should control the brightness of the other LED on the Base node using PWM.

All I/O controlled can use timeout values. If data is not received in a set amount of time, the output will turn off. This value is in 100 ms increments. **T0** to **T7** (timeout for D0 to D1 outputs) is by default \$FF, 255 or 25.5 seconds. **PT** controls the PWM timeout. Setting these to 0 will disable timeouts.

For more rapid updates, set the sample rate, **IR**, of the **Remote node** XBee to a lower value. Also, **IC** may be set to send data should digital data change. The binary value describes which I/O to use; FF means any, D0–D7. A value of \$10 (0001\_0000) would send data only if D4 changes. Only digital data is sent for a state change.

## 6: Sleep, Reset & API Modes

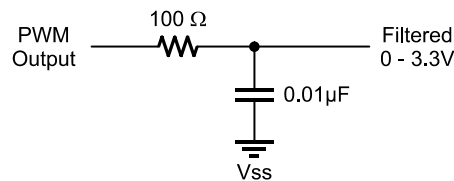
---

- ✓ Set the Remote node XBee's IR value using menu choice 7 and address 1 to configure **IR** to 50.

The digital output may be read directly by the Propeller to determine Remote node I/O states. The PWM may be read directly by measuring pulse width. It is very fast, from around 3 to 61  $\mu\text{s}$  (high to low PWM), and read by a cog counter (such as using `PulsIn_us` or using `PULSIN_C1k` for direct counter value from `BS2_Functions.Spin`). Note that a constant high or low value will cause the cog to lock up while waiting for a pin state change.

- ✓ The `PULSIN_C1k` method has been included in your code for menu choice 0. Monitor the clock counts on P7 as you adjust light levels at the Remote node. Note that maximum or minimum values may cause the cog to lock up waiting for pin change.
- ✓ Press any key while reading data to exit.

The PWM output may also be filtered to create an analog voltage using a low-pass filter shown in Figure 6-10.

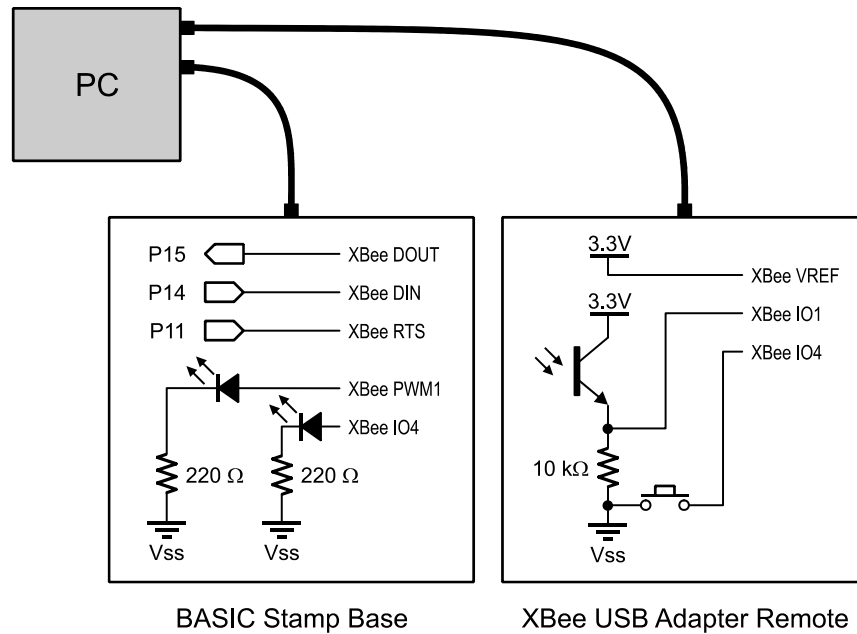


**Figure 6-10: PWM Filtering for Analog Voltage**

### BASIC Stamp and API Mode

While using API Mode has clear advantages, the amount of RAM memory (26 bytes) on the BASIC Stamp can make working with frames cumbersome and memory intensive. We'll demonstrate sending and receiving frames, but if you simply need to send and receive data, we don't recommend API Mode. Each unique use would require work to make it function for that use. Remember, both sides of a transmission DO NOT need to be in API Mode. It is simply a means of packaging data to and from the controller and using higher-level functions.





**Figure 6-11: BASIC Stamp Testing Hardware Configuration**

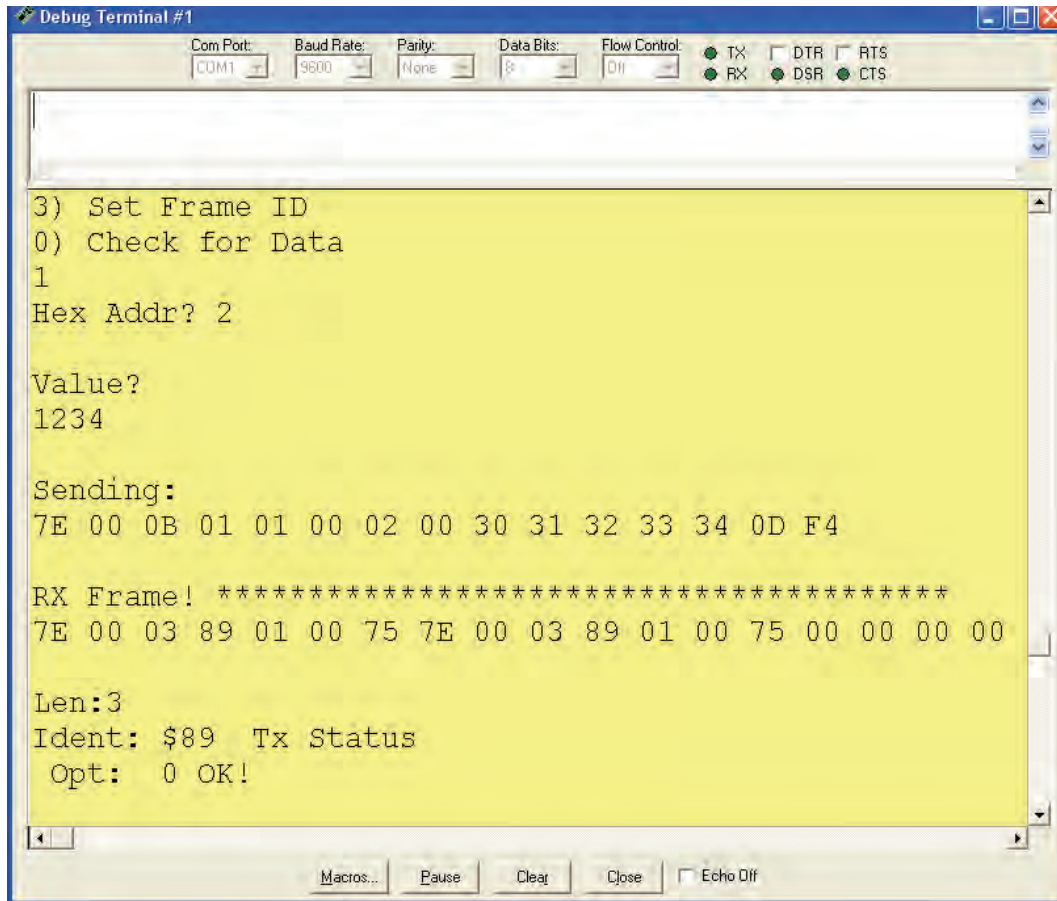
The Base node consists of a BASIC Stamp connected to an XBee, and LED and resistors directly connected to the XBee that it will control. The Remote node consists of a XBee on a USB Adapter, powered from USB, and having input devices of a pushbutton and phototransistor directly connected to XBee inputs.

The code, `API_Base.BS2`, will be used to send and receive framed data in API Mode to highlight some of the nice features of API Mode and the XBee module in this mode. The code is very long and memory intensive. Further on we will discuss some tricks to use less memory, but generally these tricks reduce the amount of data received and nearly eliminate the need to use API Mode in the first place. The configuration in Figure 6-11 is used to test `API_Base.bs2`. The Base node is controlled and monitored via the BASIC Stamp Debug Terminal and the Remote node is on USB for power and is using X-CTU for a terminal. An array of 18 bytes called `DataSet` is used to hold the incoming and outgoing frame.

#### Sending a Decimal Value

- ✓ Configure the hardware as shown in Figure 6-11.
- ✓ Configure the Remote node's XBee on USB for an address of 2 (**MY=2**) and monitor the terminal window.
- ✓ Download `API_Base.bs2` to the Base.
- ✓ Use menu option 1.
- ✓ Enter a Remote address of 2 and a value.
- ✓ Note the value should be shown in the Remote node's terminal window.
- ✓ On the Base, you should see the framed data that was sent, and a reply frame similar to Figure 6-12.

## 6: Sleep, Reset & API Modes



**Figure 6-12: API\_Base.bs2 Sending Value with Response**

You can compare the data sent to the API frame format shown in Figure 6-2 on page 104 and identify individual bytes of the frame. The Rx Frame data is a response from the XBee on the status of the transmission. The option field identifies the result of the transmission.

- ✓ Send a value to a node that is NOT present and view the status results.

Looking at the code to send our value, each field is assigned, the value is converted to decimal, the checksum is calculated by subtracting all bytes from \$FF, the data is transmitted, and the response frame is checked.

```
SELECT DataSet(0)
CASE "1"
    DataSet(0) = $7E          ' Send value to address
    DataSet(1) = 0           ' Start Delimiter
    DataSet(2) = 11         ' Length MSB
    DataSet(3) = $01        ' Length LSB
    DataSet(4) = FrameID    ' API Ident
    DEBUG CR, "Hex Addr? "  ' FrameID
    DEBUGIN HEX Value       ' Enter address
    DataSet(5) = Value.HIGHBYTE ' Parse address into frame
    DataSet(6) = Value.LOWBYTE
    DataSet(7) = 0          ' Options - request Ack
    DEBUG CR, "Value? ", CR ' Enter value to send
    DEBUGIN DEC Value
```

```

FOR Idx = 0 TO 4           ' Save value in frame as ASCII
  DataSet(8+Idx) = Value DIG (4-Idx) + $30
NEXT
DataSet(13) = CR          ' End with CR
DataSet(14) = $FF         ' Checksum start value
FOR idx = 3 TO 13        ' Calculate checksum
  DataSet(14) = DataSet(14) - DataSet(idx)
NEXT
DEBUG CR
DEBUG "Sending:",CR
FOR idx = 0 TO 14        ' Show data frame
  DEBUG HEX2 DataSet(Idx), " "
NEXT
FOR idx = 0 TO 14        ' Send data frame
  SEROUT Tx,Baud,[DataSet(Idx)]
NEXT
GOSUB GetFrame           ' accept returning frame

```

Upon calling **GetFrame**, incoming data is checked for and, based on the API Identifier, pertinent data is displayed. (More could be shown for data, but memory limitations of the BS2 required a tightening of the code.) Once data is received, the first byte is checked for the value of \$7E (Start Delimiter), and then based on the value of API Identifier, the data is parsed, pulling out appropriate data.

```

GetFrame:
  GOSUB ClearDataSet     ' Clear frame
                          ' accept frame data with RTS
  SERIN Rx\RTS,Baud,50,Timeout,[STR DataSet\18]
Timeout:
  IF DataSet(0) = $7E THEN ' Check for start delimiter
    DEBUG CR,CR,"RX Frame! ",REP "*" \40,CR
    FOR IDX = 0 TO 17     ' Show frame
      DEBUG HEX2 DataSet(IDX), " "
    NEXT
    GOSUB ParseData      ' Break down frame
  ENDIF
RETURN

ParseData:
  DEBUG CR,              ' Display length & API Ident
    "Len:",DEC DataSet(1)<<8 + DataSet(2),CR,
    "Ident: ",IHEX2 DataSet(3)
  SELECT DataSet(3)

```

Based upon an API Identifier value of \$89, it is identified as a Transmit Status Frame, and the field values are used as needed:

```

CASE $89                 ' Transmit status frame
  DEBUG " Tx Status",CR,
    " Opt: ",DEC DataSet(5)
  IF DataSet(5) = 0 THEN DEBUG " OK!"
  IF DataSet(5) = 1 THEN DEBUG " No Ack"
  IF DataSet(5) = 2 THEN DEBUG " CCA Fail"
  IF DataSet(5) = 3 THEN DEBUG " Purged"
  DEBUG CR

```

## 6: Sleep, Reset & API Modes

---

### Setting Frame ID

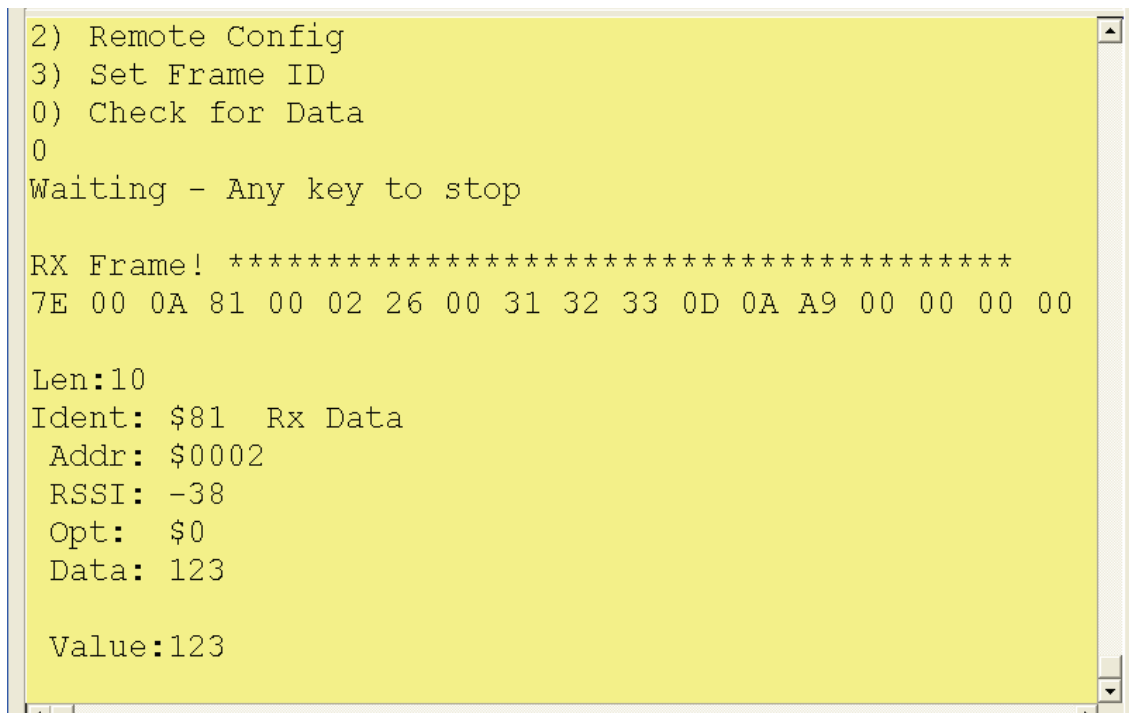
If the frame ID is set to 0, a status frame will not be sent to the controller. This may be useful at times to limit the amount of data coming in. Use menu option 3 to set the frame ID to 0, test sending a value, and then returning frame ID to a higher value.

### Receiving a Decimal Value

API\_Base.bs2 can receive a decimal value as well, and from the data packet pull additional information—this is the true power of using API Mode.

- ✓ Select menu option 0 to continually monitor for incoming frames.
- ✓ On the Remote node, use X-CTU's Assemble Packet to send a value, such as 123, to the Base.
- ✓ Monitor the Debug Terminal.
- ✓ Press any key to return to the menu (which may take several presses).

As shown in Figure 6-13, a frame with an API Identifier of \$81 is accepted and parsed showing the sender's address, the RSSI level of the received data, and the content of the data.



```
2) Remote Config
3) Set Frame ID
0) Check for Data
0
Waiting - Any key to stop

RX Frame! *****
7E 00 0A 81 00 02 26 00 31 32 33 0D 0A A9 00 00 00 00

Len:10
Ident: $81  Rx Data
Addr: $0002
RSSI: -38
Opt:  $0
Data: 123

Value:123
```

**Figure 6-13: API Base Parsing Received Frame of Data**

In the code, the frame bytes are parsed, displayed, and the data bytes are converted to decimal values for use.

```
CASE $81                                ' Received data frame
  DEBUG "  Rx Data",CR,
      " Addr: ",IHEX4 DataSet(4)<<8 + DataSet(5),CR,
      " RSSI: -",DEC DataSet(6),CR,
      " Opt:  ",IHEX DataSet(7),CR,
      " Data: "
  Value = 0                               ' show value & convert to decimal
```

```

FOR IDX = 8 TO (8 + DataSet(2)-6)
  DEBUG DataSet(IDX)
  IF (DataSet(Idx) > $29) AND (DataSet(Idx) < $40) THEN
    Value = Value * 10 + (DataSet(Idx)- $30)
  ENDIF
NEXT
DEBUG " Value:", DEC Value,CR ' Display data

```

### Receiving ADC and Digital Data

One of the great features of API Mode is the ability to receive a frame of data from a Remote node's XBee containing digital and ADC data from signals applied to the Remote XBee. Inputs D0 to D8 may be configured for ADC input (not all inputs allow ADC data) or for digital inputs. Based upon the sampling rate, a frame of data with the values is sent for use in API Mode. Please see the Propeller section for a fuller discussion of this operation.

- ✓ On the Remote node's USB XBee, ensure **DL** is set to 0 (address of Base).
- ✓ In I/O Settings, set:
  - **D0** to ADC
  - **D1** to ADC
  - **D3** to DI (Digital Input)
  - **D4** to DI (Digital Input)
- ✓ Set sampling rate (**IR**) to 7D0 (2000 decimal, every 2 seconds).
- ✓ On the Base's Debug Terminal, monitor frames using option 0.

You should see framed data arrive as shown in Figure 6-14.

```

DIO Ch3=1
DIO Ch4=0

RX Frame! *****
7E 00 0E 83 00 02 26 00 01 06 18 00 08 00 5A 00 4D 86

Len:14
Ident: $83  Rx  ADC/DIO
Addr: $0002
RSSI: -38
Opt:  $0
Data:
  ADC Ch0=90
  ADC Ch1=77
  DIO Ch3=1
  DIO Ch4=0

```

Figure 6-14: API Base Receiving Analog and Digital Data

## 6: Sleep, Reset & API Modes

---

Pressing the button on the Remote should change the value of DIO Ch4, and changing the light level to the phototransistor should change the ADC Ch1 value. Ch0 and Ch3 are not used in our hardware, but shown as additional data. The ADC is 10-bit, having a value of 0 to 1023. Please see the Propeller section for a full discussion on how the data is arranged and what analysis is required.

- ✓ Set **IR** to 0 on the Remote node to stop transmissions.

On the Remote, you may have it send only digital data when there is a state change on the digital inputs by setting **IC** to FF (monitor all 8 bits, DI0 to DI7). Note that the Base code can only accept up to 2 channels of ADC Data.

### Remote Configuration

Another great use of API Mode is configuring a Remote node's XBee (firmware version 10CD or higher required). In this mode, the 2-letter command configuration and a value are sent to a Remote XBee providing the address. This allows one XBee to manually control digital or PWM outputs on another XBee or to change its configuration.

- ✓ Using Base node option 2, enter the address for the Remote node (2).
- ✓ Control the Association LED on the Remote by entering **D5** for the command code.
- ✓ Enter 4 to turn off the Association LED.
- ✓ Repeat using 5 to turn the Association LED on.
- ✓ Connect an LED and resistor to IO2 and control it using command **D2**.

### Line Passing

In Line Passing, the outputs on one XBee are controlled by the digital and analog inputs on another XBee. Please see the Propeller section on this topic for controlling the LEDs on the Base XBee. Base configuration needs to be done in code locally, such as:

```
SEROUT Tx,Baud,[ "ATAP1,D61,IA1, D44, P12,UI0,CN",CR]
```

- ✓ Set the Remote node USB XBee to send data every 50 ms (**IR** to 32).
- ✓ Press the Remote node's pushbutton and cover the phototransistor while monitoring the LEDs on the Base.

### Shortcuts for API Mode

There are several good examples on the Parallax Forums for API Mode on the BASIC Stamp. These generally demonstrate very specific use while not gobbling up the majority of the free memory. The basic method is to do much of the frame assembly manually, parse in the needed data and calculate the final checksum value. If, for example, you wanted to send 2 bytes manually, the length and other fields can be manually configured and the last few bytes calculated to be transmitted. For our illustration, the following code is a shortcut method for remotely configuring an XBee using very little RAM and ROM:

```
DO
  DEBUG CR,"Hex Address of Remote? "      ' Enter address
  DEBUGIN HEX Addr
  DEBUG "2-letter Command? "              ' Enter config code
  DEBUGIN Code(0)
  DEBUGIN Code(1)
  DEBUG CR,"Hex Value? "                  ' Enter value for config
```

```
DEBUGIN HEX Value
Checksum = $FF-$17-2-Addr-code(0)-code(1)-value ' Calc checksum
DEBUG "Sending:",CR
SEROUT Tx,Baud,[$7E,0,17,$17,          ' start, length, ident
          0,REP 0\8,                    ' Frame ID 0, 0 64-bit
          Addr.HIGHBYTE, Addr.LOWBYTE,  ' address
          2,                            ' option - apply now
          code(0),code(1),              ' Config code
          Value.HIGHBYTE,value.LOWBYTE,' Value to send
          checksum]                    ' Checksum value

LOOP
```

On receiving data, you may take advantage of special formatters within the **SERIN** *Inputdata* argument. **WAIT** can be used to wait for the start delimiter, and **SKIP** can be used to skip over data to the pertinent data:

```
SERIN Rx,Baud,[WAIT($7E), SKIP 7, DEC Value]
```

### **Summary**

When working with the XBee individual I/O, it's important to ensure that proper voltages are applied to the 3.3 V device to prevent damage. Sleep mode allows placing the XBee module into a low-current state. Using API, data can be framed for transmission to a particular node and received data provides additional information and uses. The sending node's address and the RSSI level can often be pulled from the frame, and options such as remote configuration and transmitting ADC and digital data are available. While the coding for frames can be complex, the XBee object can assist when working in API Mode for the Propeller. Use of API Mode on the BASIC Stamp should be weighed against the need for the additional features it provides.

# 7: Monitoring and Control Projects

In this chapter we will explore projects for monitoring and control of devices via the XBee Modules.

With the BASIC Stamp, we will use StampPlot Pro, by Martin Hebel, for a monitoring and control project then use it in a joystick-controlled wireless Boe-Bot project for monitoring the bot graphically. As both projects rely heavily on using StampPlot Pro, we will begin with a short discussion of its use.

For the Parallax Propeller, the project from *Programming and Customizing the Multicore Propeller Microcontroller* (McGraw-Hill, 2010) will be used. When this chapter was written, Parallax was not carrying the XBee with adapter boards, so a board similar to the XBee SIP Adapter was used, but the purpose is the same. The video capabilities of the Propeller chip are used to visually monitor a remotely controlled bot.

### **About StampPlot Pro**

StampPlot Pro is a graphical data acquisition and control software tool that accepts and sends data in a way very similar to a terminal program. Using **DEBUG** statements in PBASIC or serial communications objects with the Propeller chip, data is sent to StampPlot in order to plot data, update interface objects, or to read interface objects. It also supports full configuration through text sent to it and provides a means of drawing on the plot area. We will look at some key features of StampPlot Pro related to the projects in this section. More information may be found in the package's help files and in primer tutorials at [www.selmaware.com/stampplot/download.htm](http://www.selmaware.com/stampplot/download.htm), including how to develop your own drag-and-drop-interfaces. For questions on StampPlot, please use the Yahoo forum listed on the StampPlot web pages.

- ✓ Download and install StampPlot Pro: [www.selmaware.com/stampplot/download.htm](http://www.selmaware.com/stampplot/download.htm).

### **Connecting a Device to StampPlot**

StampPlot Pro may be used by directly connecting the controller to the PC running StampPlot. Or, using Transparent AT mode, the data for StampPlot can be sent from a remote node to an XBee USB connected base on the PC.

The first step in connecting a BASIC Stamp or XBee module on an XBee USB Adapter board to StampPlot is to ensure the correct COM port is selected (either actual or virtual using the USB interface). You will take these steps later when you complete the two BASIC Stamp projects that follow the StampPlot Configuration and Data Exchange Overview.

- ✓ From the menu bar, select View → Configuration.
- ✓ Select the correct COM port for your device. If you have Version 3.8.4 or higher, use the “?” button to see only active COM ports.
- ✓ Select Set as Default to ensure it is active the next time you open StampPlot.
- ✓ From the menu bar, select Plot → Connected (F6) and Plot Data (F7). You will notice corresponding button bar options shown in down state.



**Only one application at a time!** As always, only one application on your computer can have access to a COM port at a time. Programming and use of Debug Terminals will require StampPlot be disconnected (F6).

**Valid COM Port Range: 1–15:** StampPlot, using legacy code, can only connect to COM ports 1–15. If your USB board is assigned a higher value, use your Windows Device Manager to select the COM port, go to advanced settings, and select a valid COM port number.



StampPlot comes with several pre-defined graphical interface options. Interfaces can also be custom-built with text-based macro files (.spm files) created by StampPlot, like the one provided for this text: Open Base Interface.spm. An interface can also be defined through information sent from a microcontroller, as we will do later with the BASIC Stamp. Data for StampPlot may come from a directly connected controller, such as the BASIC Stamp using the **DEBUG** instruction, or via a USB-connected XBee using data sent from a controller using the **SEROUT** instruction.

### StampPlot Configuration and Data Exchange Overview

This section gives an overview of StampPlot instructions for various types of tasks, as well as example code snippets for both the BASIC Stamp 2 and Propeller P8X32A. It is not meant to be a hands-on tutorial, but a summary of some of the software's features.

#### Plotting Analog Data

Once connected, serial data may be sent to StampPlot for plotting up to 10 analog values. The general syntax for plotting either sending single value or multiple comma-separated values (each plotted in a different color) is:

10,50,90



#### End with Carriage Return!

All strings sent to StampPlot must end with a carriage return in order to be processed.

For the BASIC Stamp, single or multiple values may be sent from the BASIC Stamp, such as:

```
DEBUG DEC Val, CR
DEBUG DEC Val1, ",", DEC Val2, ",", DEC Val3, CR ' multiple values
' with text-commas.
```

Using a Base node using the XBee USB Adapter, values may be sent to the computer from a Remote node using **SEROUT**, which will then send the serial data to StampPlot:

```
SEROUT Tx, Baud, [DEC Val,CR]
SEROUT Tx, Baud, [ DEC Val1, ",", DEC Val2, ",", DEC Val3, CR]
```

With the Propeller chip, the XBee object or similar serial objects may be used for communicating with either a directly connected Propeller or through a USB Base XBee:

```
XB.Dec(val)
XB.Tx(13)
```

...or:

```
XB.Dec(Val1)
XB.Tx(",")
XB.Dec(Val2)
XB.Tx(13)
```

## 7: Monitoring and Control Projects

---

### Plotting Analog Values Manually

When comma-separated values are sent, each value is assigned a separate channel, from 0 to 9, for plotting and a color. The Analog Channel (!ACHN) command instruction may also be used to manually define the channel and color:

```
!ACHN Channel, Value, Color
!ACHN 1,310,(RED)
```

If you have multiple units sending data to one PC for plotting, these different units may send data for different channels, allowing data from multiple units to be plotted on the same graph.

BASIC Stamp:

```
DEBUG "!ACHN 1, ", DEC val, ",(RED)", CR
SEROUT Tx,Baud,[ "!ACHN 1, ", DEC val, ",(RED)", CR]
```

Propeller:

```
XB.Str(string("!ACHN 1, "))
XB.Dec(val)
XB.str(string(", (RED)", 13))
```



**View your data for format:** A good check is to view your data with a serial terminal to ensure that your data is formatted properly. The Immediate/DEBUG window in StampPlot can also be used to monitor data and errors. It can also be used for directly entering text for testing or configuration of StampPlot.

**Multiple Units Sending Data:** If you have multiple units sending data to a single application, it may be necessary to increase the XBee Packetization timeout (RO) value. This will help ensure strings are not sent a portion at a time, possibly mixing together data from multiple transmitters. Ideally, a string should only be sent if it contains a full set of data.

### StampPlot Instructions

Instructions to control StampPlot, such as !ACHN above, may also be sent. These commands can configure nearly all aspects of StampPlot as well as creating interfaces directly from the controller. Most of the instructions being with an exclamation point “!” (drawing instructions can use others) and use a 4-letter instruction followed by data. Some examples:

```
ISPAN 0,2000    — Sets the span of Y-axis
ITMAX 360      — Set maximum time on X-axis in seconds
!PLOT ON       — Enables plotting
!RSET          — Reset the plot (clear plot)
!NEWP         — Start a new plot, resetting all values to default
```

These commands can be sent from the microcontroller, entered into the Immediate/DEBUG Window of StampPlot, or be used in the code of StampPlot objects such as a button on the interface when clicked. As always, all strings must end with a carriage return.

BASIC Stamp:

```
DEBUG "!RSET",CR
SEROUT Tx, Baud, ["!RSET",CR]
```

Propeller:

```
XB.Str(string("!RSET", 13))
```

### Creating and Using StampPlot Objects

In StampPlot, “objects” are buttons, gauges, meters and the like, that a user normally interacts with for viewing or control. This allows an interface to be developed for unique monitoring and control applications.

Objects may be created in StampPlot via drag-and-drop using the Object Editor to develop the interface, and the Macro Editor to generate the interface macro file. Objects may also be created in code to be sent from the controller. The plotting area is reduced using the !PPER (Plot Percentage) instruction to provide room on the background. Then, objects are created, defining object names and the parameters. This information may be sent as strings from the controller.

!PPER 70,100 — Plotting area is 70% wide and 100% high of StampPlot

For example, here is code to create a meter object called Meter1 at position 75, 90, width and height of 20, range of 0 to 2000 with alarms at 0, 1500. When reading in the alarm area, the code for the meter would be processed (not shown).

```
!POBJ oMeter.Meter1=75.,90.,20.,20.,0,2000,0,1500
```

Once created, objects may be updated by using the !POBJ instruction (!O for short) and the object’s name:

```
!O Meter1=500
```

#### BASIC Stamp:

```
DEBUG "!O Meter1=", Dec val, CR
SEROUT Tx,Baud, ["!O Meter1=", DEC val,CR]
```

#### Propeller:

```
XB.str(string("!O Meter1="))
XB.Dec(val1)
XB.Tx(13)
```

StampPlot interface objects may be referenced by placing the name in parentheses. One example of this is requesting the value be returned to the controller via a !READ instruction, then accepting the data by the controller:

```
DEBUG "!READ (Slider1)",CR      ' request object value
DEBUGIN DEC val                 ' accept object value
```

...or:

```
SEROUT Tx, Baud, ["!READ (Slider1)",CR]      ' request object value
SERIN Rx\RTS, Baud, 100, Timeout, [DEC Val]  ' accept value with timeout
```

#### Propeller:

```
XB.Str(string("!READ (Slider1)", 13))        ' request object value
Val := XB.RxDecTime(100)                    ' accept value with timeout
```

Objects on the StampPlot interface may be edited by holding the Shift key and right-clicking an object.

## 7: Monitoring and Control Projects

---

### Drawing Instructions

Drawing instructions allow drawing to occur within the plot area (or background). Using drawing instructions allows lines, circles, arcs, points, text, and images to be placed on the plot. Coordinates may be given to either reference the X (time) and Y (analog value) axis values, or as absolute coordinates of 0,0 (lower-left) to 100,100 (upper-right) regardless of axis values by appending an “a” to the value.

Here is an example for drawing a line from 20,20 to 80,80, in red, using absolute coordinates:

```
!LINE 20a,20a,80a,80a,(RED)
```

Using current data, the controller may send properly-formatted strings to draw in real time. Angles of the arcs may be defined using the DARC (draw arc) and DPIE (draw pie) instructions, . The width of the drawing may also be defined using the DPTH instruction.

Drawing instructions may also begin with @ for constant or ~ for temporary drawing commands. Normally, when a drawing command such as !CIRC is executed, the command is performed again when the plot redraws or is refreshed. A reset (!RSET) clears all drawings. Temporary commands are not saved or redrawn during a refresh. Constant drawing commands persist through resets. Using constant drawings, drawings may be performed once and will be redrawn through resets until a new plot is started or the !CLRC (clear constant) instruction is issued.

Actually, two plot areas, called pages, exist for drawing: page 0 and page 1. This allows a drawing to be performed on a hidden page and then copied over to the viewed page, providing “instant” updates instead of seeing each component drawn individually. Either page may be configured to be the drawing and viewed page. The command DPG1 (draw on page 1) can be used to draw to the hidden page, and CPY1 to copy the image to the viewed page 0. Drawing (and plotting) to non-viewed pages can dramatically increase the speed of drawing operations as well.

### Math Operations

While not used in our examples, StampPlot may also perform floating point, trigonometric and many more operations on data before use. For example, to convert a digital value from a 10-bit ADC (analog to digital converter) with a span of 5 V back into an equivalent voltage before plotting, inform StampPlot to perform math using brackets on the value and formatted for 2 decimal places:

```
[ val * 5 / 1023 FORMAT 0.00 ]
```

In PBASIC:

```
DEBUG "[", DEC val, " * 5 / 1023 FORMAT 0.00]",CR
```

Note that all operations require spaces between values and operations. For equations, the order of operations is left to right unless nested brackets are used.

### XBee Interfacing

Typically, the XBee is used simply as a transparent device for receiving and sending data between StampPlot and the Remote units. However, recent changes to StampPlot make it more XBee friendly for control and configuration purposes though API Mode is not yet supported. For example, placing the XBee into Command Mode and changing the destination address, such as to a DL of 2, can be

performed. The general sequence of events for configuring the DL setting with guard times while grabbing the returned OK (or values) is as follows:

```

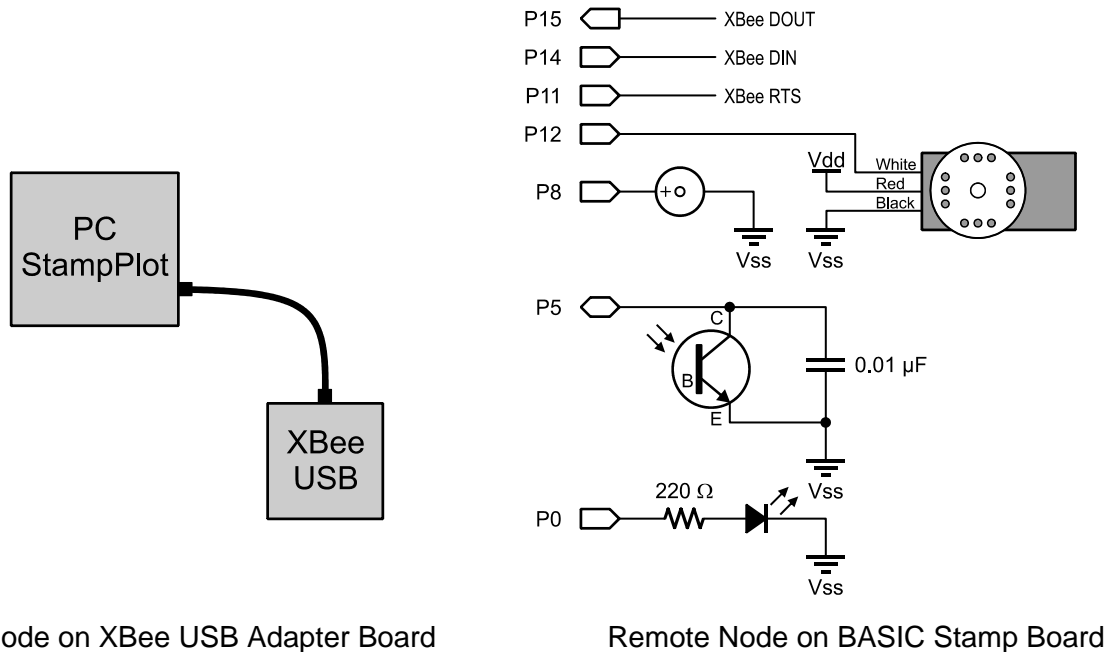
!WAIT 2
ISRAW +++
IGRAB 3
ISTAT (GRAB)
ISEND ATDL 2
IGRAB
ISTAT (GRAB)
ISEND ATCN
IGRAB
ISTAT (GRAB)
    
```

### **BASIC Stamp Project: Monitoring and Controlling a Remote Unit Project**

In this project, we will use many of the features of StampPlot across an XBee network in order to monitor data from a Remote unit and update parameters on the Remote unit. Figure 7-1 is the hardware configuration of the project.

#### **Hardware**

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>(1) BASIC Stamp development board</li> <li>(1) XBee SIP or 5V/3.3V Adapter board</li> <li>(1) XBee USB Adapter Board</li> <li>(2) XBee modules with default settings</li> <li>(1) Pushbutton</li> <li>(1) LED</li> </ul> | <ul style="list-style-type: none"> <li>(1) Buzzer (piezospeaker)</li> <li>(1) Servo</li> <li>(1) Phototransistor</li> <li>(1) 0.01<math>\mu</math>F capacitor</li> <li>(1) 10 k<math>\Omega</math> resistor</li> <li>(1) 220 <math>\Omega</math> resistor</li> </ul> |
|---|--|



**Figure 7-1: Base and Remote Plotting and Control Hardware**

## 7: Monitoring and Control Projects

In this example, the Remote BASIC Stamp reads the phototransistor and sends the data to the Base node to be plotted in StampPlot (Figure 7-2). The Remote node then uses StampPlot's !READ instruction to read the value of the interface switch and sliders. The switch controls the LED and the slider controls the pitch of the buzzer and the servo position. The StampPlot interface objects are accessed by their names: Meter1, swLED, sldTone and sldServo.

- ✓ Ensure your XBee modules are restored to their default configuration.
- ✓ Open the interface macro provided in the distributed files by double-clicking [Base Interface.spm](#).
- ✓ Connect StampPlot to the COM port defined for the XBee USB Adapter Board by using [View](#) → [Configuration](#) in StampPlot and selecting the correct COM port. Use the “?” to view current available COM ports.
- ✓ Download Remote\_Monitor\_Control.bs2 to the Remote node.
- ✓ Monitor StampPlot while changing the light level at the Remote node's phototransistor.
- ✓ Change the virtual button and sliders on StampPlot and monitor the Remote node.

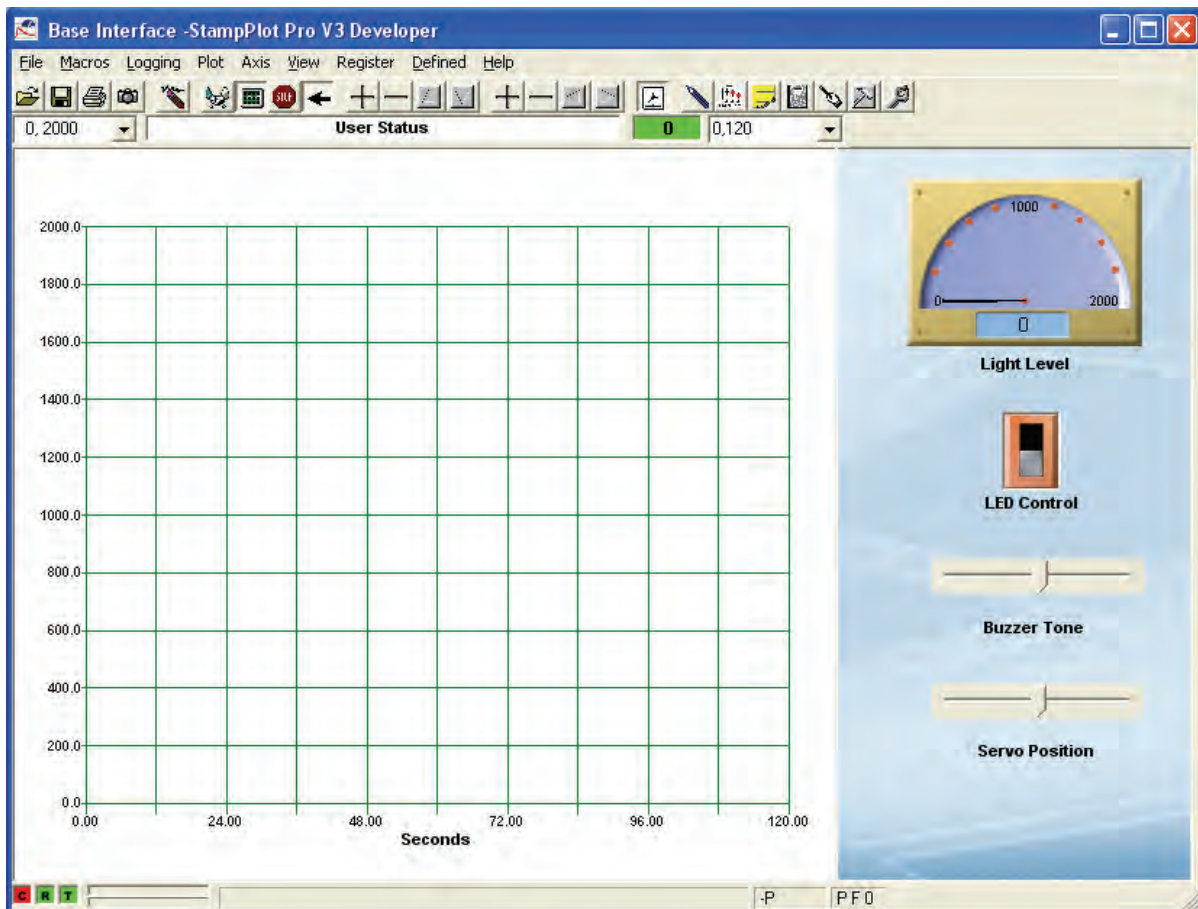


Figure 7-2: StampPlot Interface for Base

The following code demonstrates sending data to the USB Adapter for plotting by StampPlot.

```
StampPlot_Update:
  HIGH PhotoT          ' Use RCTime to get value
  PAUSE 5
  RCTIME PhotoT,1,Light
  ' Send value to be plotted on plot channel
  ' !ACHN Channel (0-9), value, color (0-15 or color name)
  SEROUT Tx,Baud,["!ACHN 0,",DEC Light,"(BLACK)",CR]
  ' Update meter
  ' !O Meter1 = value
  SEROUT TX,Baud,["!O Meter1","=",DEC Light,CR]
RETURN
```

When reading the interface, the Remote unit requests the values of swLED, sldTone and sldServo to control the Remote node's hardware by requesting their values and accepting the returned data.

```
StampPlot_Read:
  GOSUB EmptyBuffer          ' ensure buffer empty
  ' request switch value for LED state
  ' !READ (swLED) for LED data
  SEROUT TX,Baud,["!READ (swLED)",CR]
  ' Accept returned data
  SERIN Rx\RTS,Baud,100,Timeout,[DEC State]
  ' Request slider for buzzer
  ' !READ (sldTone) for buzzer frequency
  SEROUT TX,Baud,["!READ (sldTone)",CR]
  ' Accept returned value (0 to 5000)
  SERIN Rx\RTS,Baud,100,Timeout,[DEC Freq]
  ' !READ (sldServo) to control servo position
  SEROUT TX,Baud,["!READ (sldServo)",CR]
  ' Accept returned value (500 to 1000)
  SERIN Rx\RTS,Baud,100,Timeout,[DEC Position]
Timeout:
RETURN
```

### **BASIC Stamp Project: Wireless Joystick Control and Monitoring of a Boe-Bot**

For this project we will develop a wirelessly controlled Boe-Bot (our Remote node) using the 2-Axis Joystick module on a second BASIC Stamp board (our Base node), and will monitor the robot using StampPlot. The Base node is connected to the PC for control and sends data to the PC for StampPlot updates.

#### **Hardware Required**

##### **For Boe-Bot (Remote node)**

- (1) Boe-Bot Robot, assembled and tested\*  
(#28132 or #28832)
- (1) Ping))) Ultrasonic Distance Sensor (#28015)
- (1) Ping))) Mounting Bracket Kit (#570-28015)
- (1) HM55B Compass Module (#29123)
- (1) ServoPal (#28824)
- (1) XBee SIP Adapter
- (1) XBee 802.15.4 Module

##### **For Joystick (Base node)**

- (1) Board of Education and  
BASIC Stamp 2
- (1) 2-Axis Joystick (#27800)
- (1) XBee SIP Adapter
- (1) XBee 802.15.4 module
- (1) Piezospeaker
- (3) 220  $\Omega$  resistors
- (1) 1 k $\Omega$  resistor
- (2) 0.01  $\mu$ F capacitors

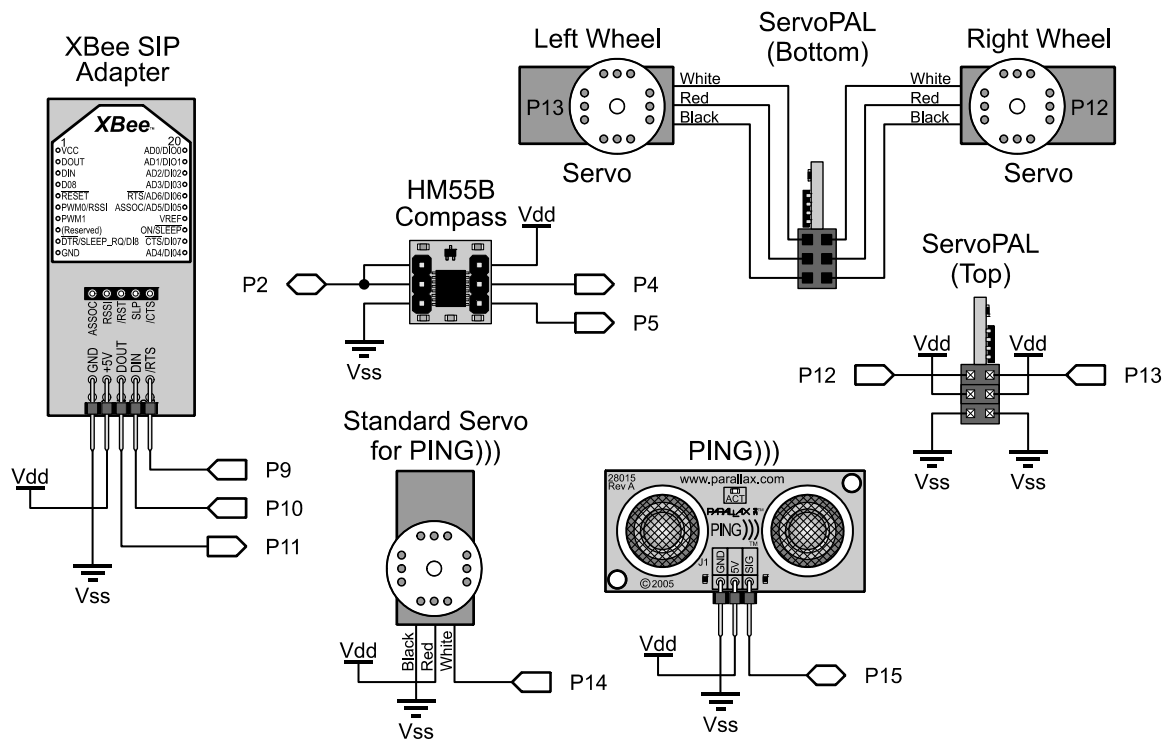


**Note:** you may wish to construct your Boe-Bot so that the servos are positioned with the access port facing away from the battery pack. This will allow easy access to the servos' adjustment screws from the back of the chassis for re-centering the servos later in the activity, if needed.

## Building the Robot (Remote Node)

Figure 7-3 shows the Boe-Bot hardware configuration used in this project. The Boe-Bot drive is controlled from wirelessly received data, using the ServoPal to allow smooth driving while sending or receiving data. A Ping))) sensor on the servo mount is used to look directly ahead or at an angle to the Boe-Bot. The HM55B Compass Module monitors the direction of the Boe-Bot. The Ping))) sensor distance in inches and the mount servo direction are sent back to the Base for monitoring with StampPlot. Figure 7-4 is an image of our assembled Boe-Bot.

- ✓ Construct the Boe-Bot and download `Wireless_Joystick_BotBot.bs2`. Be sure your board's servo voltage jumper is set to Vdd to prevent damage to the ServoPal.



**Figure 7-3: Boe-Bot Hardware**



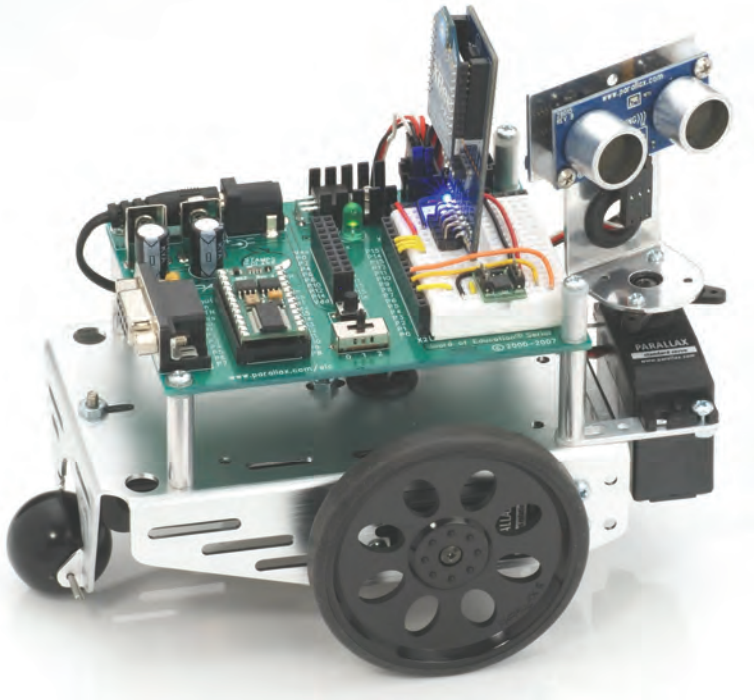


Figure 7-4: Assembled Boe-Bot

### Building the Joystick (Base Node)

The Base hardware, shown in Figure 7-5, uses the joystick to control the direction of the Boe-Bot. The button, when pressed, will cause the Base node to send data to control the Boe-Bot's Ping))) servo direction. The buzzer uses the returned Ping))) distance to sound a tone. The closer the object, the higher the pitch of the buzzer will be, providing audible feedback for remote driving. Figure 7-6 shows the completed unit. Figure 7-7 shows the placement of components under the joystick. Note that the button is hanging off the board; you may use a heavy gauge wire to help hold it in place.

- ✓ Construct the Base and download BoeBot\_Control.bs2. Be sure to close the BASIC Stamp Editor's Debug Terminal to free up the COM port for StampPlot.

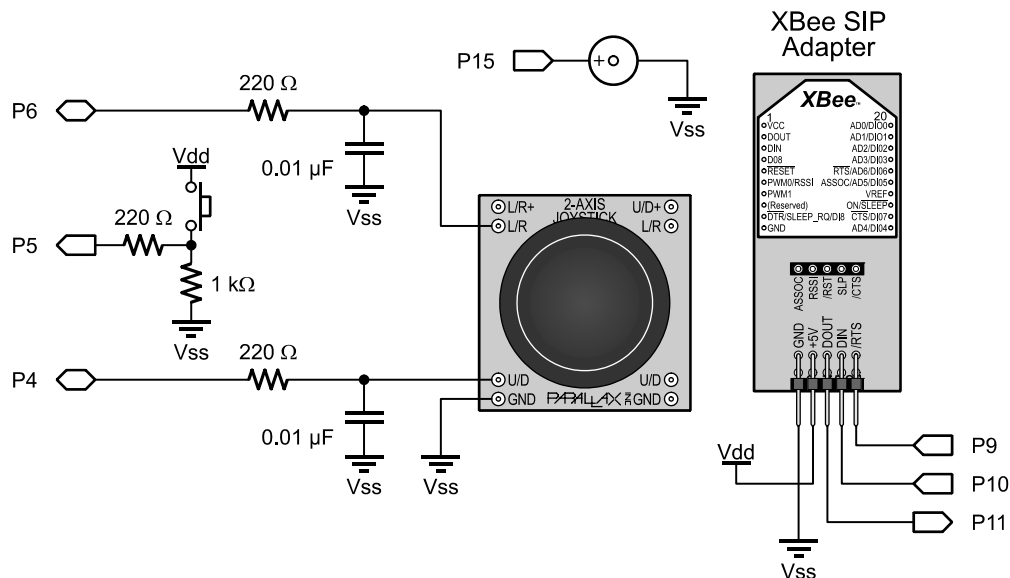
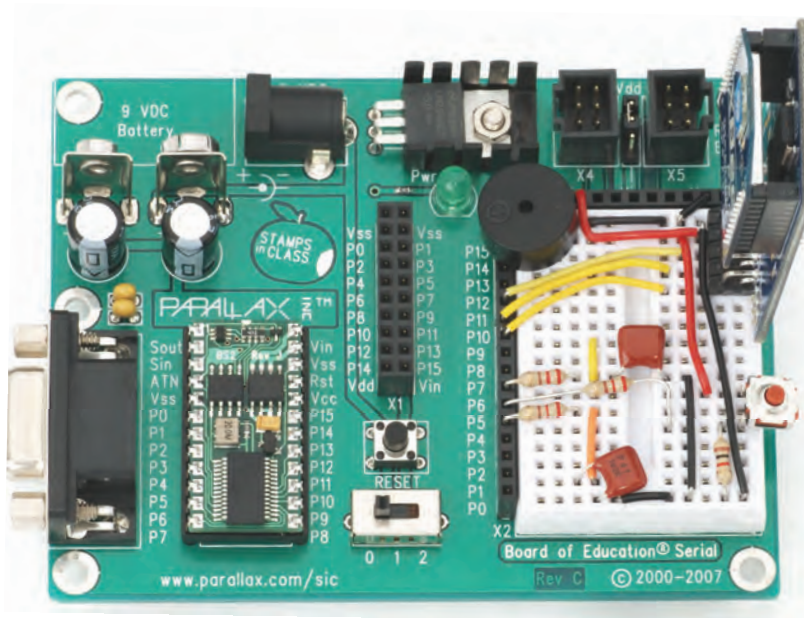


Figure 7-5: Base Control Hardware



**Figure 7-6: Base Controller with Joystick**



**Figure 7-7: Base Controller without Joystick**

On connecting the Base unit to StampPlot, the BASIC Stamp will configure the graphical interface and send drawing instructions to StampPlot for monitoring the Remote Boe-Bot, as shown in Figure 7-8. When driving normally, the direction of the Boe-Bot will be displayed by the pie-shaped wedge in the center, along with a red dot to indicate direction and distance value returned from the Ping))) sensor. When the pushbutton on the BOE breadboard is pressed, the display will not clear, allowing an area to be mapped.

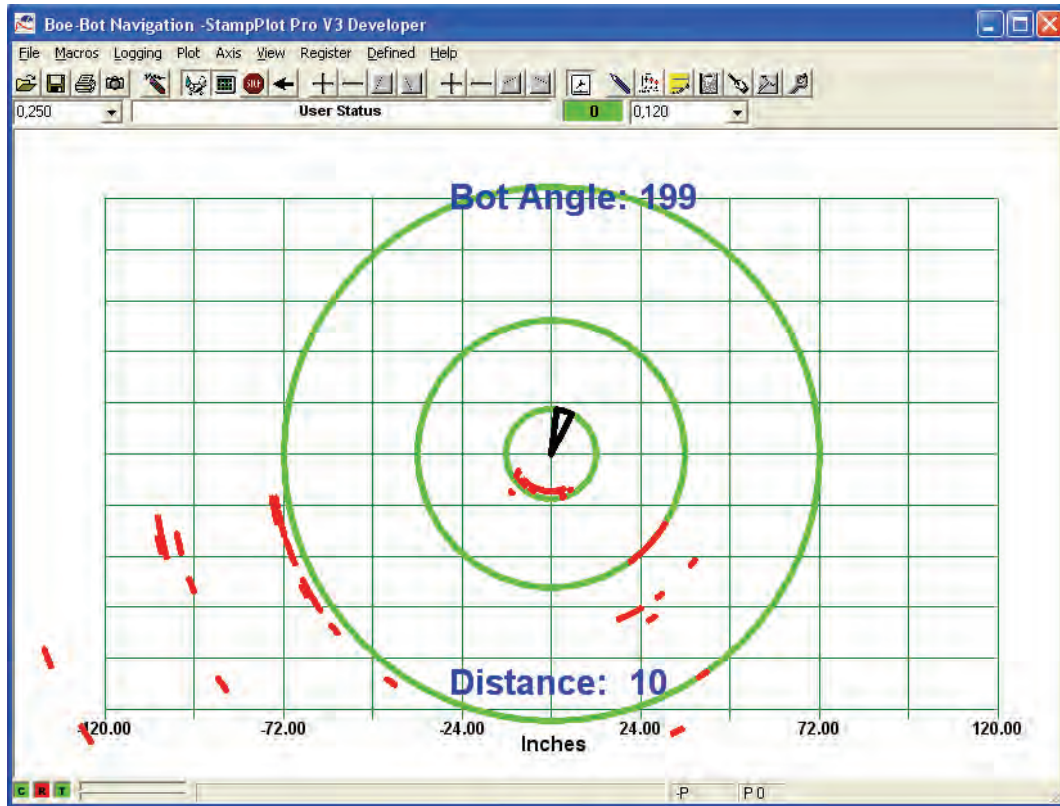


Figure 7-8: StampPlot Interface for Boe-Bot Monitoring

### Joystick (Base Node) Code Discussion

Upon connecting with StampPlot the BASIC Stamp will be reset, which restarts the code and allows initial configuration for StampPlot to be sent to the PC. This starts a new plot, clears any objects, configures the plot, and uses constant drawing instructions to lay down the persistent background with concentric circles at 1-foot, 3-foot and 6-foot distances.

```

PAUSE 500
DEBUG "!NEWP",CR                               ' New plot
DEBUG "!POBJ Clear",CR                         ' Clear object
DEBUG "!TITL Boe-Bot Navigation",CR           ' Title plot
DEBUG "!PLOT ON",CR                            ' Enable plotting
DEBUG "!TMAX 120",CR                           ' set up max time (X axis)
DEBUG "!TMIN -120",CR                          ' Set min time
DEBUG "!XLBL Inches",CR                        ' Label Y Axis
DEBUG "@VPG1",CR                               ' View page 1
    ' **** Constant Drawings ****
DEBUG "@DWTH 5",CR                             ' Draw width
DEBUG "@CIRC 50a,50a,12,(Green)",CR           ' Draw circles - 1 foot
DEBUG "@CIRC 50a,50a,36,(Green)",CR          '                               3 foot
DEBUG "@CIRC 50a,50a,72,(Green)",CR          '                               6 foot
DEBUG "@FREC -140,-1,-121,252,(white)",CR    ' Cover Y Axis

PAUSE 2000                                     ' Guard Time
SEROUT Tx,Baud,["+++"]                        ' Command mode sequence
PAUSE 2000                                     ' Guard Time
SEROUT Tx,Baud,["ATD6 1,CN",CR]              ' RTS enable (D6 1)
                                             ' Exit Command Mode (CN)

```

## 7: Monitoring and Control Projects

---

The directional values of the joystick are read based on the initial center position, the drive values for right and left servos are calculated, combining the directions, and limited to values in the range of 300 to 1200. The first time the routine is called, the center position of the joystick is saved.

```
Read_Stick:
HIGH UD_Pin           ' Charge cap
PAUSE 2
RCTIME UD_Pin, 1, UD   ' Read stick

HIGH LR_Pin           ' Charge Cap
PAUSE 2
RCTIME LR_Pin, 1, LR   ' Read stick
IF First_Read = 0 THEN ' If first read of joystick
  UD_C = UD           ' store center positions
  LR_C = LR
  First_Read = 1      ' mark as centers read
ENDIF
' Mix Up-down and Left-right to determine drive values, limit ranges
R_Drive = 750 - ((UD - UD_C) * 3) + (LR - LR_C) MIN 300 MAX 1200
L_Drive = 750 + ((UD - UD_C) * 3) + (LR - LR_C) MIN 300 MAX 1200
Ping_New = 750 - ((LR - LR_C)*10) MIN 300 MAX 1200      ' Calculate position
```

The drive data is sent to the Remote unit with a start delimiter of “D” (drive). If the button is pressed, the left-right joystick value is sent with a start delimiter of “P” (pan).

```
IF PB = 1 THEN          ' If pressed, send PING position
  SEROUT Tx,Baud,["P",CR,CR,DEC Ping_New,CR,CR]
ELSE                    ' Not pressed, send drive
  SEROUT Tx,Baud,["D",CR,CR,DEC R_Drive,CR,CR,DEC L_Drive,CR,CR]
ENDIF
```

After sending data, the BASIC Stamp waits for 10 ms for returned data from the Boe-Bot with distance in inches, direction angle and the position of the Ping))) servo. A counter is used to attempt to read data 50 times, ensuring sufficient time and checking to see if the start delimiter is in the buffer. You may elect to comment out the code to update StampPlot and un-comment out the line **GOSUB Show\_Debug** to simply show data in the BASIC Stamp Editor’s Debug Terminal.

```
Counter = 0
ReadAgain:
  SERIN Rx\RTS,Baud,10,Timeout,[DataIn]      ' Get returned data
  IF DataIn = "!" THEN                        ' IF good start, accept rest
    SERIN Rx\RTS,Baud,20,Timeout,[DEC Inches] ' Get inches
    SERIN Rx\RTS,Baud,20,Timeout,[DEC Angle]  ' Get Angle
    SERIN Rx\RTS,Baud,20,Timeout,[DEC Ping_Pos]' Get Ping Position
    FREQOUT Buzzer,50,(146-Inches) * 35      ' Sound buzzer
    GOSUB Update_StampPlot                    ' Update StampPlot
    ' GOSUB Show_Debug                        ' If no StampPlot, update
  DEBUG
  GOTO ReadDone
ENDIF
Timeout:
  Counter = Counter + 1                       ' In case not delimiter,
  IF Counter < 50 THEN GOTO ReadAgain         ' read 50 characters to find
  IF Counter < 50 THEN GOTO ReadAgain         ' if delimiter in buffer
ReadDone:
```

In the `Update_stampPlot` subroutine, if the button is not pressed, the plot will be reset, clearing the plot of current data (constant drawings will be kept). The direction of the bot is calculated for plotting and drawn as a pie shape along with text information. If the button is not pressed, the plot will not be cleared; this allows distance measurements to accumulate on the plot, effectively mapping the area while panning the Ping))) sensor.

In either case, button pressed or not, the direction of the bot and the Ping))) position are added together and a small arc is drawn at that angle with a distance of the Ping))) data in inches. Once the image is fully assembled, the unviewed page 0 is copied to page 1 for viewing.

```

IF PB = 0 THEN                                ' If PB not pressed, clear
  DEBUG "!RSET",CR                             ' reset plot
                                              ' Place text
  DEBUG "~TEXT 40a,100a,2a,(Blue),Bot Angle: ",DEC Angle,CR
  DEBUG "~TEXT 40a,5a,2a,(Blue),Distance:  ",DEC Inches,CR

  D_Angle = 360 - Angle -90                    ' Calculate bot drawing angle
                                              ' draw pie based on angle
  DEBUG "~DPIE 50a, 50a, 5a,",SDEC D_Angle-10,
        ",",SDEC D_Angle+10,",(Black)",CR
ENDIF
IF Ping_Pos > 750 THEN                        ' Calculate angle of bot + ping
  D_Angle = 360- Angle + 90 + (Ping_Pos - 750 / 5)
ELSE
  D_Angle = 360- Angle + 90 - (750 - Ping_Pos / 5)
ENDIF
                                              ' Draw arc for ping point
DEBUG "~DARC 50a, 50a,", DEC Inches,"",
      SDEC D_Angle-1,"",SDEC D_Angle+1,",(Red)",CR
DEBUG "~CPY0",CR                             ' copy image to page 1 for viewing
RETURN

```

### Boe-Bot (Remote Node) Code Discussion

On the bot, the code receives drive and Ping)) position data from the Remote node. When data is received starting with a “D” (for driving) delimiter, the drive values are accepted and used to set the ServoPal. Code not shown uses the variable `Counter` to turn off the drive if 50 loops are performed without receiving new data.

```

SERIN Rx\RTS,Baud,10,TimeOut,[DataIn]        ' Briefly wait for delimiter
IF DataIn = "D" THEN                          ' If ! delimiter, get data
  DEBUG "Here",CR
  SERIN Rx\RTS,Baud,10,TimeOut,[DEC UD]      ' Accept Up-Down value
  SERIN Rx\RTS,Baud,10,TimeOut,[DEC LR]      ' Accept Left-Right value
  ' mix UD and LR channels for drives.
  ' Normal range for UD/LR is 1 to 70. Maybe need to adjust based
  ' on your values.
  R_Drive = 750 - ((UD - UD_C) * 2) + (LR - LR_C) ' Calc right drive
  L_Drive = 750 + ((UD - UD_C) * 2) + (LR - LR_C) ' calc Left drive

  PULSOUT nInp,R_Drive                       ' Set ServoPals
  PULSOUT nInp,L_Drive
  counter = 0                                 ' Clear counter for no-data timeout
  GOSUB SendData                              ' Send data to control
ENDIF

```

## 7: Monitoring and Control Projects

---

If data is sent with a P (button pressed on Base control), the data is accepted for the Ping))) servo position and the bot is stopped by sending 2000 to the ServoPal. Free to modify the code to be able to span while driving by commenting out the `PULSOUT` commands. Keep in mind that once set, the Ping))) servo will keep its position even while driving.

```
IF DataIn = "P" THEN                                ' If p delimiter, get data
  SERIN Rx\RTS,Baud,10,Timeout,[DEC Ping_Pos]      ' Accept Ping servo value
  PULSOUT nInp,2000                                ' Stop bot
  PULSOUT nInp,2000
  GOSUB SendData                                  ' Send data to base
ENDIF
```

The data of **Inches**, **Angle** and **Ping\_Pos** (Ping))) position) position are sent back to the Base controller.

```
SendData:
  GOSUB Get_Sonar                                ' get sensor value
  GOSUB Get_Compass                              ' Get compass Angle
                                                    ' Send data to control
  SEROUT Tx,Baud,["!",CR,CR,DEC Inches,CR,CR,
  DEC Angle,CR,CR,DEC Ping_Pos,CR,CR]
RETURN
```

### Testing the Code

Let's try the project:

- ✓ Open StampPlot, select the correct COM port and connect.
- ✓ Allow time for the XBee modules to initialize.
- ✓ You should hear the buzzer sounding and StampPlot updating once ready.
- ✓ Drive the bot with the joystick.
- ✓ Use the button to pan the Ping))) sensor left-right.
- ✓ Try following a wall by panning the Ping))) sensor to the side, and then driving while trying to maintain a set distance from the wall while monitoring StampPlot or listening to the tone.



**Note:** If your bot moves when the joystick is centered, you may need to adjust the 'center' position on the servos by turning the servo's adjustment screw slightly.

### Things to Think About

Ready for a challenge? Consider using the analog and digital inputs of the XBee for direct control. The joystick can supply two analog values to the XBee, and the button a digital input. The XBee would transmit the analog and digital data directly to the XBee on the bot. The bot would read the API analog/digital frame for direct control for drive. Getting data back to the base in API Mode is a little problematic, but code could switch the XBee back to AT Mode (**AP 0**), and send data directly back for plotting by StampPlot or use in a terminal window. Once sent, the bot's XBee could be flipped back into API Mode (**AP 1**) to receive more analog/digital data.

Another neat idea would be to have the remote bot read sliders on StampPlot which would be configured for 0-360 degrees and servo speed. The bot would read these values via the wireless link. The bot would then use the speed slider value for drive speed and would adjust position to match the

bearing defined by the StampPlot slider. As you update sliders on the StampPlot interface, the bot would read them and respond accordingly to drive at the speed and direction defined by the interface sliders.

### **Propeller Project: A Three-Node Tilt-Controlled Robot with Graphical Display**

A 3-node network for controlling and monitoring a robot will be explored for the last project in this chapter. The system as shown in Figure 7-9 has:

- A Propeller Demo Board network node (address 0) with an accelerometer to measure angle of inclination on two axes for the tilt controller
- A robot on the network (address 1) using a Propeller Proto Board on a Boe-Bot robot chassis with HM55B compass, Ping))) range finder on a servo bracket, and LEDs
- A Propeller Demo Board on the network (address 2) driving a TV for video display of the graphical display



#### ***Programming and Customizing the Multicore Propeller Microcontroller: The Official Guide***

The Propeller project in this chapter is an excerpt from the book named above, in “Chapter 5: Wirelessly Networking Propeller Chips” by Martin Hebel. Many thanks to McGraw-Hill for generously allowing the reprint of this material. The complete example code for this section can be found in the Chapter\_05 subdirectory of:

<ftp://ftp.propeller-chip.com/PCMProp>.

This Official Guide is available for purchase from Parallax.com, Amazon.com, and other book retailers.

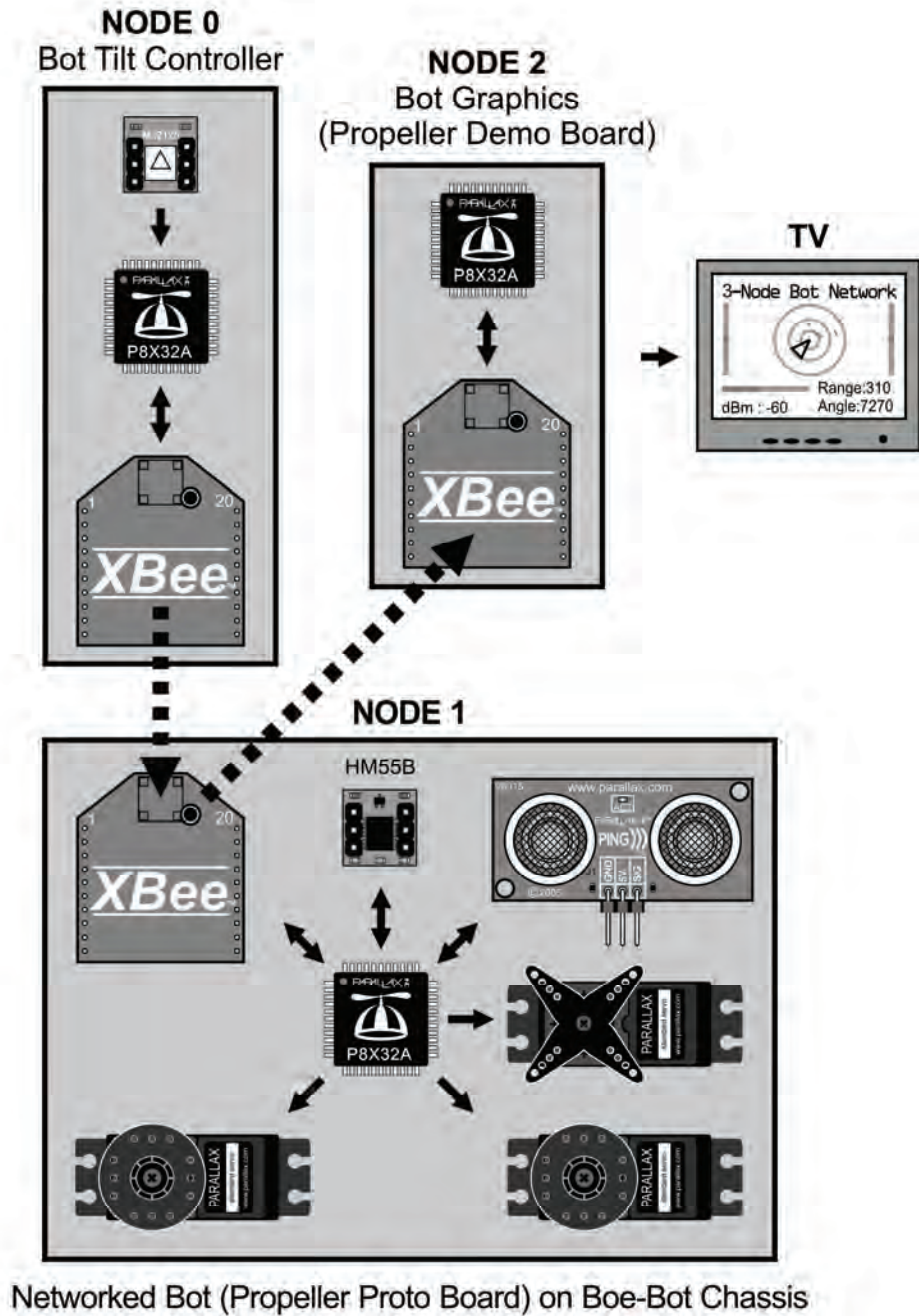
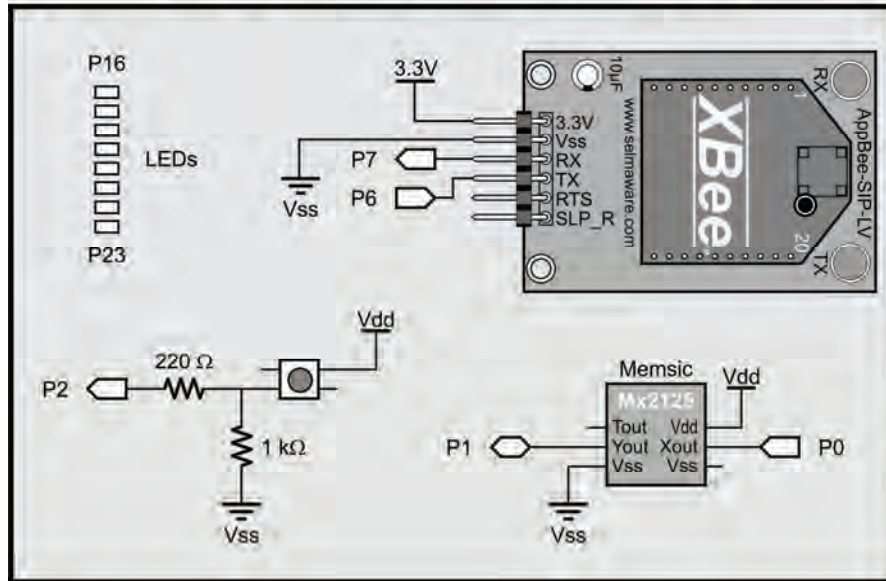


Figure 7-9: 3-Node Bot Network Diagram

Figure 7-10 shows the wiring connection diagram for each of the nodes. Note that in switching to the Proto Board we will change the I/O pins used for the XBee.



**NODE 0**  
Bot Tilt Controller (Propeller Demo Board)



**NODE 1**  
Bot (Propeller Proto Board on BOE-Bot Chassis)

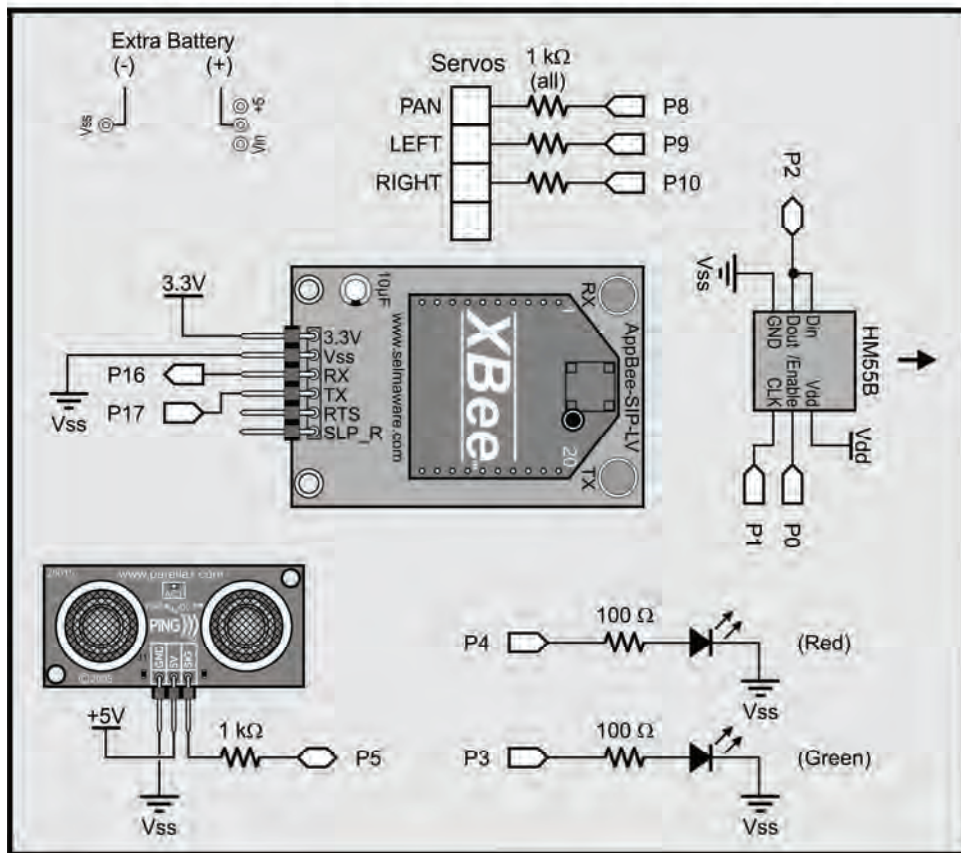
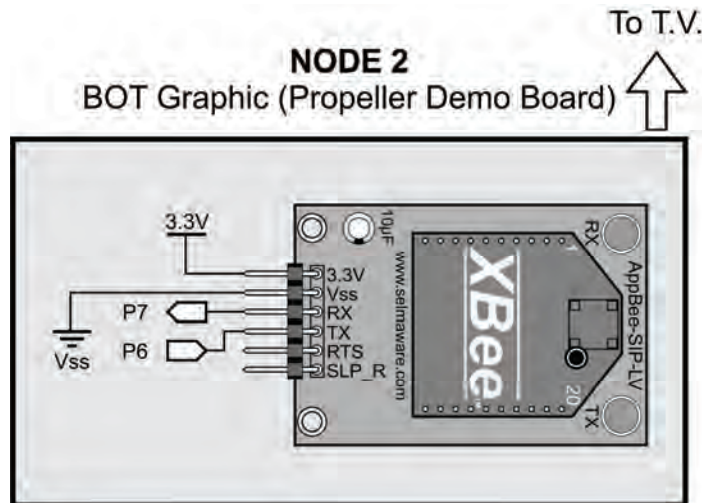


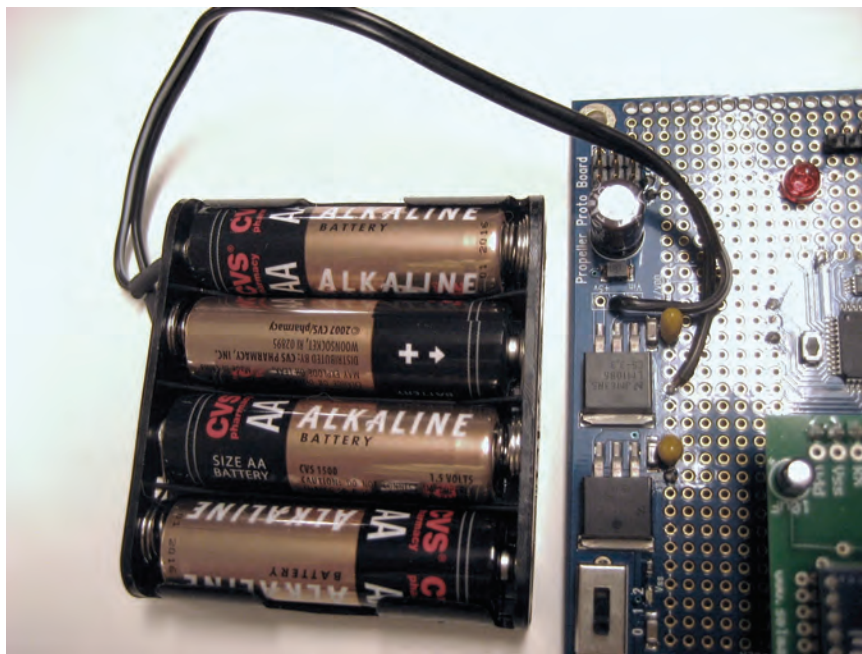
Figure 7-10: Hardware Wiring Diagram for Bot System



**Figure 7-10: Hardware Wiring Diagram for Bot System (continued)**

Hardware construction tips for bot:

- ✓ If you are not familiar with the Boe-Bot robot, you may want to look through *Robotics with the Boe-Bot* by Andy Lindsay, available for download at [www.parallax.com/education](http://www.parallax.com/education), to familiarize yourself with the basic hardware and servo operation.
- ✓ For the bot, two 4-AA packs were used with the spare tied under the normal battery pack. The second battery pack is used only for servo power. Attempts at using a single supply caused voltage and current spikes affecting the Propeller chip's operation. The connector of the battery pack was cut off and soldered to the servo header power and Vss (See Figure). Other sources may be used, but supply voltage should not exceed 7.5 V or the servos can be damaged. Use coated wire to strap the batteries under the bot.



**Figure 7-11: Supplying Separate Power for Bot's Servo Drive**

- ✓ Ensure the MH55B compass is mounted facing forward and that it is away from large current loads such as batteries and servos. The magnetic fields will cause problems with proper compass bearing.
- ✓ The Ping))) sensor is mounted using the Ping))) Mounting Bracket Kit. Manually rotate the servo to find the center prior to mounting the bracket. Mount the cable header so that servo rotation does not hit it (the servo turns further manually than with the code—about 45 degrees each way).
- ✓ A battery supply was used on the tilt controller as well for unfettered operation.

### Overview of System Operation

**Tilt Controller:** This is used to read the Memsic 2125 Accelerometer Module, calculate right and left motor drives based on inclination, and then send drive values to the bot. The tilt controller, shown in Figure 7-12, also receives data for the bot and uses the PING))) range measurements to light the eight Demo Board LEDs, showing distances of 100 mm for local range indication. Pressing the pushbutton will send a panning map instruction to the bot to map the area in front of it for display by the TV graphics node. This node has a MY address of 0 and a DL address of 1 (the bot).

**Bot:** This controls all functions of the networked bot shown in figure, including:

- Accepting drive values for motor drive. Should no data be received for 1.5 s, the bot will stop and blink the red LED.
- Accepting panning map instruction (p) to perform mapping operation. When this instruction is received, the bot will stop and pan the PING))) sensor from right to left, measuring distances and sending map data (m) to the TV graphics node and the tilt controller. When mapping is complete, the bot will remain steady and blink the green LED until the tilt controller releases it from mapping mode (user presses button again). The bot will send a clear (c) code to the video to clear the display when map mode exits.
- While driving, the bot will transmit updates (u) of right and left drive values, PING))) range, and direction of travel from the HM55B compass (0-8191).
- This node has a MY address of 1 and sends data to \$FFFF—all nodes on the network for a broadcast.

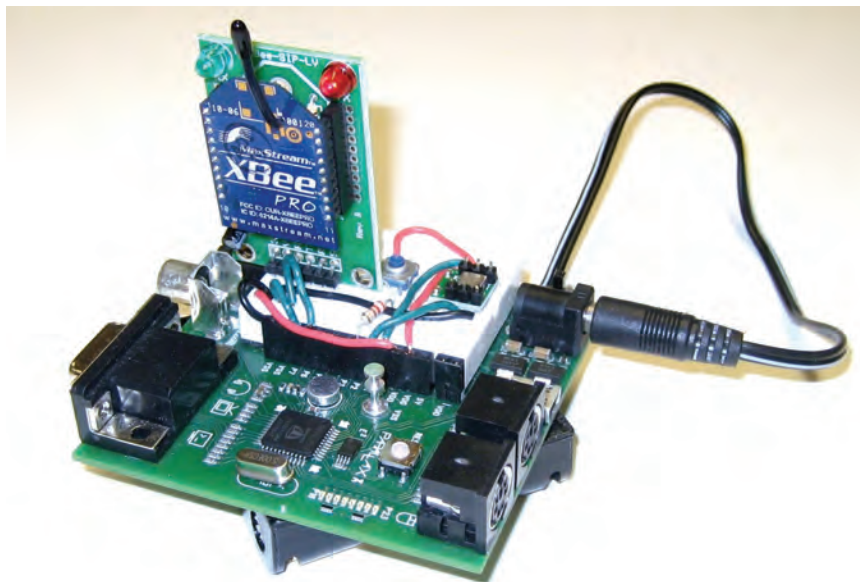
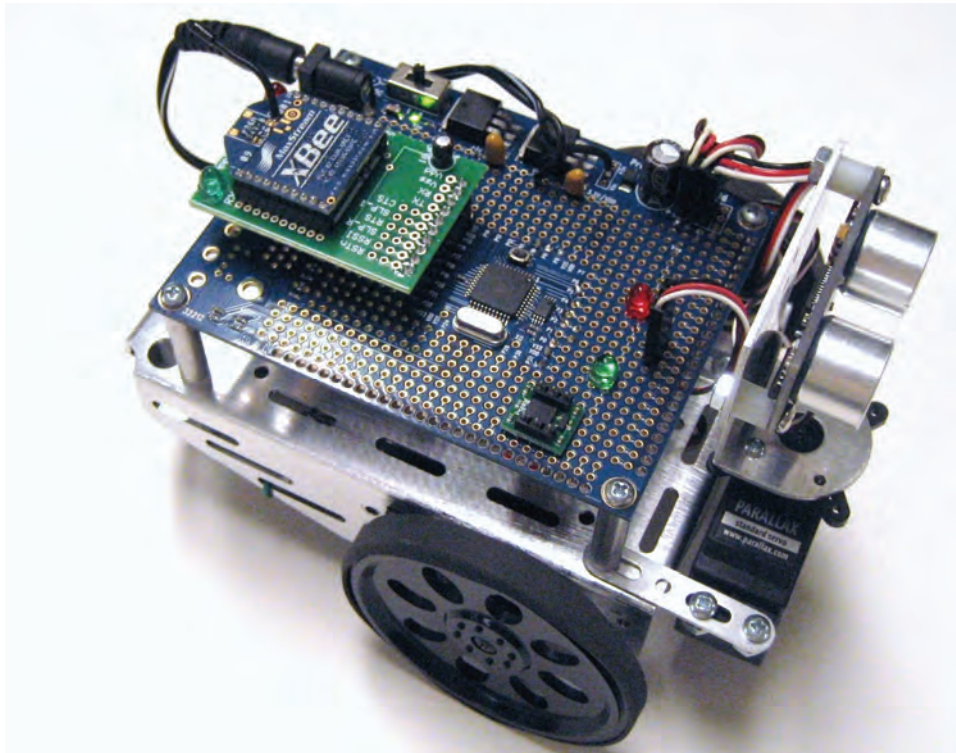
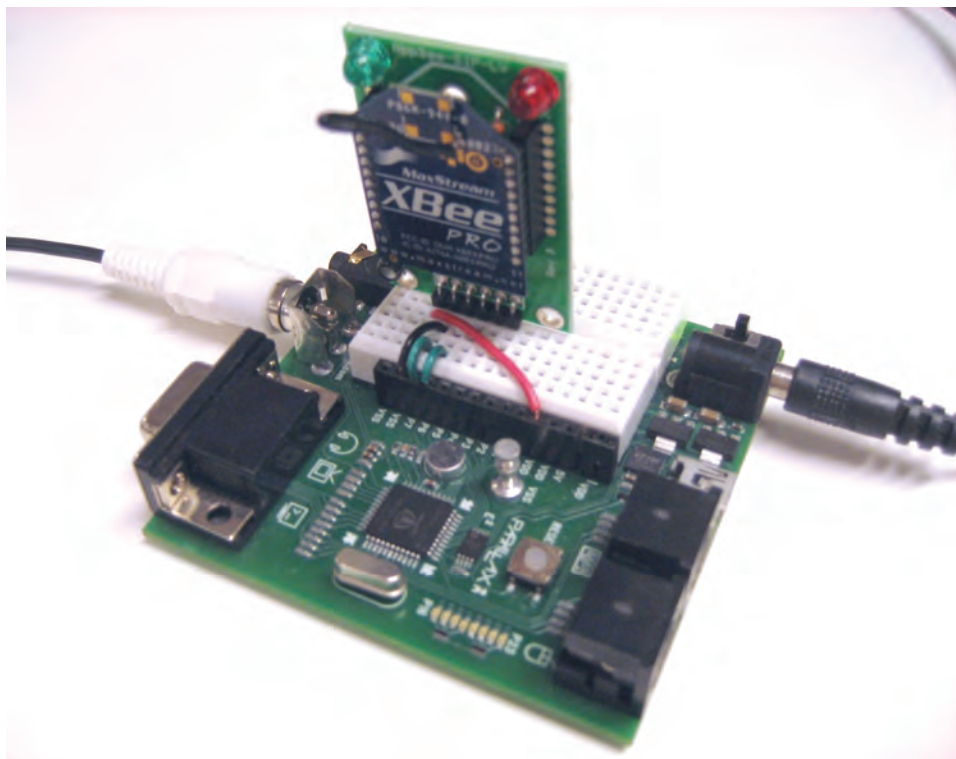


Figure 7-12: Tilt-controller Board with Memsic 2125 Accelerometer



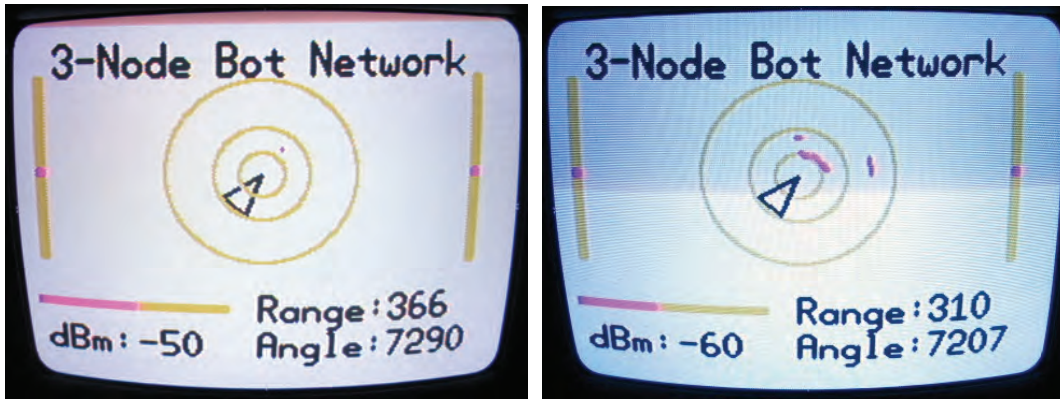
**Figure 7-13: Networked Bot with Range Finder, Compass, and XBee**



**Figure 7-14: Bot TV Graphics Controller**

**Bot Graphics:** shown in Figure 7-14, drives the graphics TV display showing:

- The bot bearing as a rotating triangle and text
- Distance to object as a red point in front of the bot and text
- Yellow range marker circles at 0.25 m, 0.5 m, and 1 m
- Left and right drives as bar indicators
- Signal strength for RSSI (dBm) as bar and text



**Figure 7-15: TV Displays for Normal and Panning Map Data**

The update packets contain information for much of the display, and the RSSI is pulled from the received frame through API Mode. When the bot is in mapping mode, mapping packets contain data to plot the range map. It also accepts clear codes from the bot to clear the mapped display. The node has a MY address of 2 and sends out no data. The output display of the graphics controller is shown in Figure 7-15 in both normal driving and with the bot performing a panning map operation.



**Note:** The Ping))) range finder has a wide angle of emission and reception. Do not expect pinpoint accuracy when mapping.

## Bot Network Code

### Bot Tilt Controller

For the tilt controller, in the `SendControl` method, if the button is pressed, a series of p's is transmitted. With the amount of data flying, some missed bytes on reception are normal, and this helps ensure the bot gets a p-instruction for a panning map operation. If not pressed, the accelerometer is read for the X- and Y-axis (-90 to 90 degrees, 0 level) and the drive for each servo is calculated by mixing the two axes of tilt for a final servo value of 1000 to 2000 (the range of servo control) for each. The data is sent as a "d" packet for drive.

```
Forward := (accel.x*90-offset)/scale * -1
' Read and calculate -90 to 90 degree for turn
Turn := (accel.y*90-offset)/scale
' scale and mix channels for drive, 1500 = stopped
Left_Dr := 1500 + Forward * 3 + Turn * 3
Right_Dr := 1500 - Forward * 3 + Turn * 3
```

## 7: Monitoring and Control Projects

---

In the `AcceptData` method (which is running in a separate cog), incoming packets are analyzed if update data (u) or map data (m) and the local LEDs are updated. Based on the range, eight 1s are shifted to the left eight positions, then shifted right again based on the range/100. This allows one LED to light for every 100 mm or 0.1 m, out to 800 mm or 0.8 m.

```
outa[16..23] := %11111111 << 8 >> (8 - range/100)
```

### Bot Code

For the bot controller, in `Start`, received bytes are analyzed with a timeout. If any data is not received for 1500 ms, the red LED will begin to blink. Received data is analyzed for either “d” for drive data or “p” to begin a mapping scan.

```
case DataIn
    ' test accepted data
    "d":
        Right_dr := XB.RxDEC
        Left_dr := XB.RxDEC
        SERVO.Set(Right, Right_dr)
        SERVO.Set(Left, Left_Dr)
        ' if drive data
        ' get right and left drive
        ' drive servos based on data
    "p":
        mapping := true
        outa[grnLED]~~
        Map
        ' p = pan and map command
        ' set flag for mapping
        ' turn on green LED
        ' go map
```

The `SendUpdate` method is run in a separate cog to continually send out the status of the range, direction, and drive values led by “u.” The value of theta is subtracted from 8191 to allow the direction of rotation to be correct in the graphics display. If a panning map is in progress, updates are suspended due to mapping being true.

```
Repeat
    if mapping == false
        XB.Delay(250)
        Range := Ping.Millimeters(PING_Pin)
        theta := HM55B.theta
        XB.TX("u")
        XB.DEC(Range)
        XB.CR
        XB.DEC(8191-theta)
        XB.CR
        xb.DEC(Right_Dr)
        XB.CR
        XB.DEC(Left_Dr)
        XB.CR
        ' if not mapping
        ' read range
        ' Read Compass
        ' send "update" command
        ' Send range as decimal string
        ' Send theta of bearing (0-8191)
        ' send right drive
        ' send left drive
```

When mapping, the value of `pan` is looped from 1000 to 2000, the range of allowable servo values. The range is measured, and the `PanOffset` is calculated. The value of the “pan” has 1500 subtracted (recall that 1500 is a centered servo). The result is multiplied by 2047 (90 degrees, with 8191 being a full 360 degrees) and divided by the full range of `pan`. Finally, an “m” is sent followed by range and angle of the servo plus the `pan` offset. This repeats for each value of `pan`, from 1000 to 2000, in increments of 15 steps or 1.35 degrees (15 • 90 degrees / 1000 steps = 1.35 degrees). Once mapping is complete, the system will wait until another “p” is received to exit pan mapping mode while sending a “c” to clear the video display. The variable “mapping” is used as a flag to prevent the `SendUpdates` code running in a separate cog from sending updates while mapping.

```
Pub Map | panValue
'' turns servo from -45 to + 45 degrees from center in increments
'' gets ping range and returns m value at each
```

```

SERVO.Set(Right, 1500)      ' stop servos
SERVO.Set(Left, 1500)

SERVO.Set(Pan, 1000)      ' pan full right
XB.Delay(1000)

                                ' pan right to left
repeat pan from 1000 to 2000 step 15
  SERVO.Set(Pan, panValue)
  Range := Ping.Millimeters(PING_Pin)      ' get range calculate
                                           ' based on compass
                                           ' and pan
  PanOffset := ((panValue-1500) * 2047/1000)
  XB.TX("m")                          ' send map data command
  XB.DEC(Range)                         ' Send range as decimal
  XB.CR
  XB.DEC((8191-Theta) + PanOffset)      ' Send theta of bearing
  XB.CR
  XB.delay(50)
  XB.delay(1000)

SERVO.SET(Pan,1500)      ' re-center pan servo

```

**Try it!**

Add another device, such as speaker, to your bot. Add a button on the tilt controller and modify code to control the device from the tilt controller.

**Bot Graphics**

The bot graphics code is responsible for accepting the data and displaying it graphically on a TV screen. Note that this XBee is in API Mode so that the RSSI level may be pulled out of the received frame. The code looks for one of three incoming byte instructions: “u,” “m,” and “c.” Updates, “u,” are messages with update data as the data moves, with range, bearing, and drive values (limited between 1000 and 2000), and it retrieves RSSI level for display creation.

```

Repeat
  XB.API_rx      ' Accept data
  If XB.RxIdent == $81      ' If msg packet...
    if byte[XB.RxData] == "u"      ' If updates, pull out data
      get DEC data skipping 1st byte (u)
      range := XB.ParseDEC(XB.RxData+1,1)
      bearing := XB.ParseDEC(XB.RxData+1,2)
      rDrive := XB.ParseDEC(XB.RxData+1,3) <#2000 #>1000
      lDrive := XB.ParseDEC(XB.RxData+1,4) <#2000 #>1000
      RSSI := XB.RxRSSI
      Update

```

Mapping (m) strings are used to map what the bot “sees” without clearing off old data while a mapping pan is in progress. Clear, “c,” is received once the bot switches back into drive mode after mapping.

We aren’t going to delve too deeply into the graphics creation here, as it’s not a major subject for this chapter. One point of interest is in that many graphic programs the video data is written into one part of memory (such as `bitmap_base`), and when the complete display change is ready, it is copied into the section of memory that the graphics driver uses to display the actual display. It is effectively double-buffered to prevent flicker on the screen. We do not have the luxury of the memory needed for that operation. Instead, to reduce flicker, values of the old data are saved. When updating, the graphics are redrawn in the background color to “erase” them, then the new data is used to draw the graphics in the correct color, such as in this code:

## 7: Monitoring and Control Projects

---

```
' Draw bot vector image
gr.width(2)
gr.color(0)
gr.vec(120,120, 100, (bearing_1), @bot)
gr.color(1)
gr.vec(120,120, 100, (bearing), @bot)
' white
' erase last image
' black
' Draw new image
```

Many features of `graphics.spin` are used, including text, lines, arcs, and vector-based graphics. The code is fairly well commented for adaptation.



### Try it!

Add another sensor to your bot. Modify both the bot and graphics code to send and display the value.

## Conclusion

In this chapter, and throughout the tutorial, we explored interfacing with the XBee modules. From Base control to multiple controllers communicating, the use of the XBee with the BASIC Stamp and Propeller chip open many doors in the area of monitoring and control. We provided information and examples; the rest is up to you and your needs!



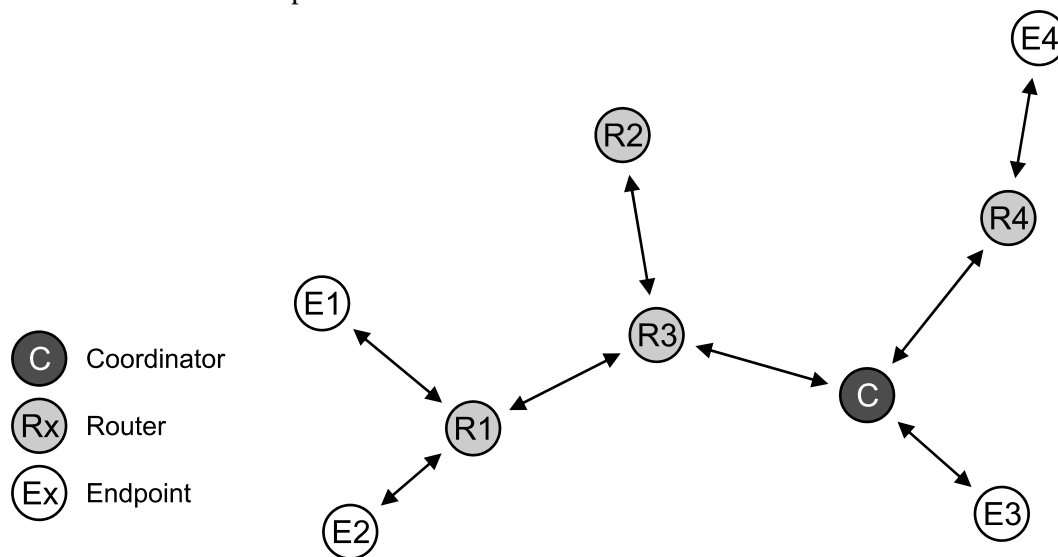
## 8: XBee ZB Module Quick Start

by Daniel Harris

Digi International's XBee ZB modules (series 2) offer a significant set of features that enable point-to-point or point-to-multi-point communications in your project. If you have not previously used XBee modules and are simply looking for basic wireless communication, then I would recommend sticking with the regular XBee 802.15.4 (Series 1) modules—XBee ZB modules require much more initial configuration before they can be used for communication.

### Mesh Network Topology

For those who are looking to take advantage of the mesh network topology that an XBee ZB network provides, these quick start instructions will get you started. Figure 8-1 is an example of a possible ZB Personal Area Network map.



**Figure 8-1: XBee ZB Personal Area Network Map Example**

In the example above, End-point 1 can communicate directly with End-point 4 by communicating through other Routers and the Coordinator. This communication is made possible by the mesh network infrastructure that an XBee Series 2 network provides. These instructions, outline how to set up point-to-point communication between two Series 2 modules in the simplest possible network configuration, a Coordinator and a single Router.

### XBee Series 2 Initial Configuration Instructions

#### Hardware Required

- (2) XBee ZB Modules
- (2) Parallax XBee USB Adapters

### Hardware and Connections

- ✓ Download and install Digi's X-CTU software so that you can configure the XBee ZB module: <http://www.digi.com/support/kbase/kbaseresultdetl.jsp?kb=125>
- ✓ Connect your XBee ZB module to the computer, using an XBee USB Adapter board (Parallax stock #32400).
- ✓ Once your XBee is connected to the computer, start the X-CTU software. You should see available COM ports enumerated in the "Com Port Setup" pane. Select the COM port your XBee is connected to and click the Test/Query button at the right to test communications between the computer and XBee ZB module.
- ✓ If the test fails, try another COM port. If all ports fail, check your connections.

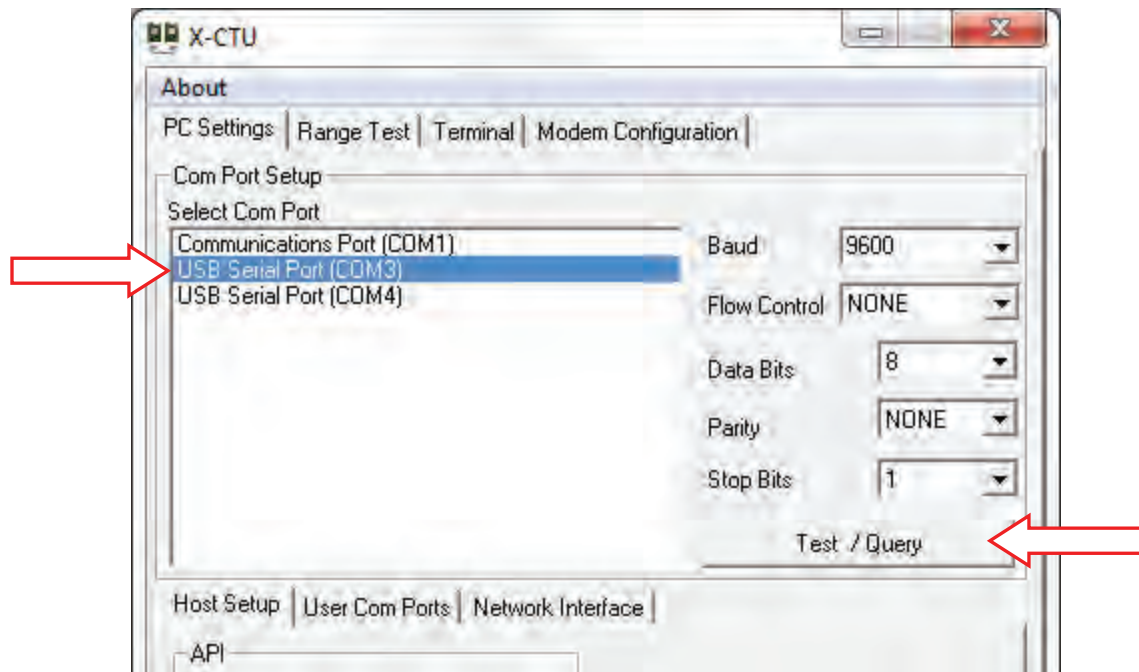


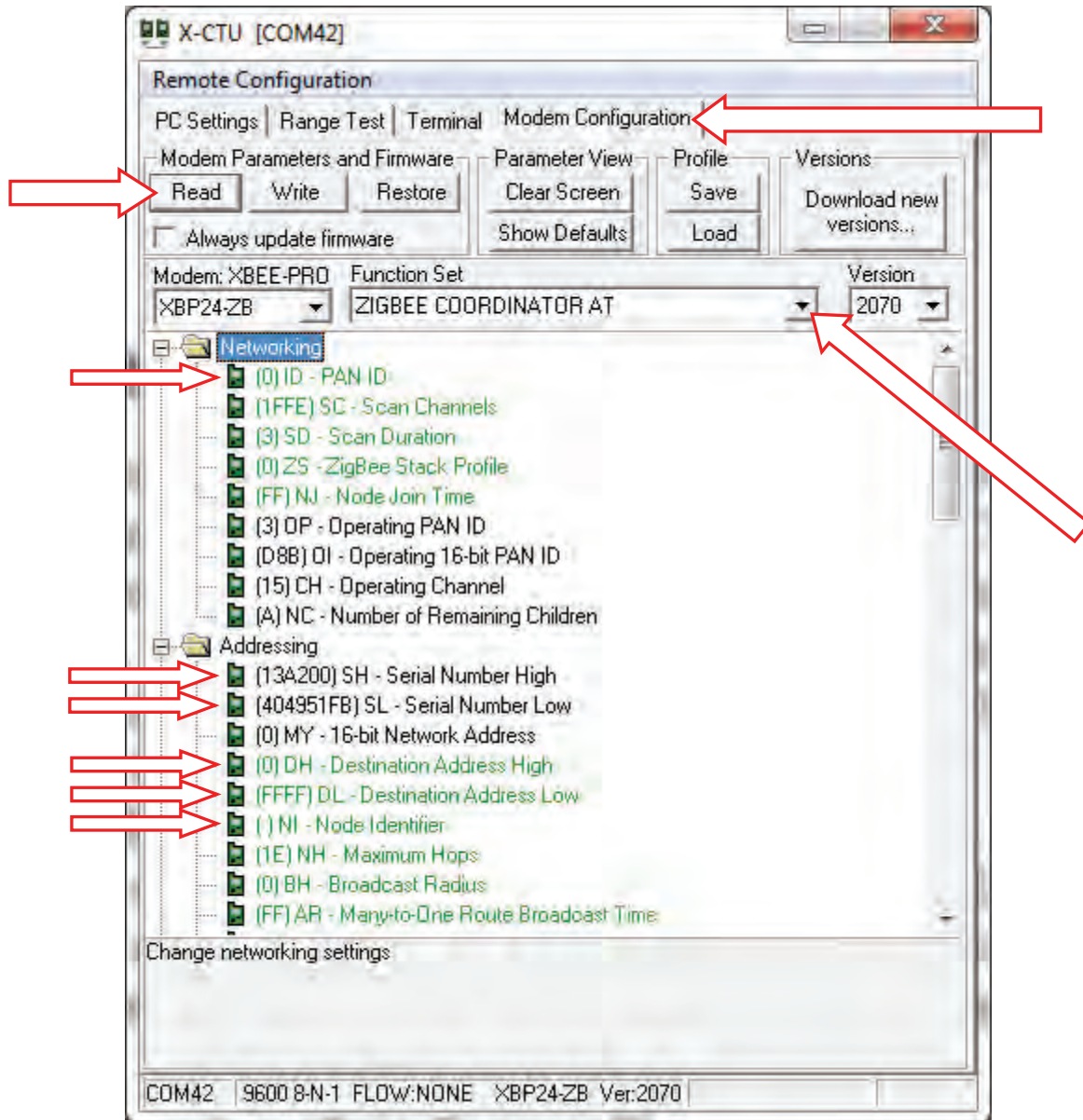
Figure 8-2: X-CTU PC Settings Options

### Configuring an XBee ZB module as the Coordinator—Part 1 of 2

You are going to need to configure BOTH XBee ZB modules separately. One module is going to be configured as a "Coordinator" and the other as a "Router". We will start by configuring the Coordinator.

- ✓ Once your computer is able to talk to the XBee Series 2 module, click the Modem Configuration tab.
- ✓ Press the Read button to read the modules current configuration.

The X-CTU software should display a great deal of configuration information about the XBee ZB module. Your screen will look something like the image in Figure 8-3. Some important configuration options are denoted with a red arrow.

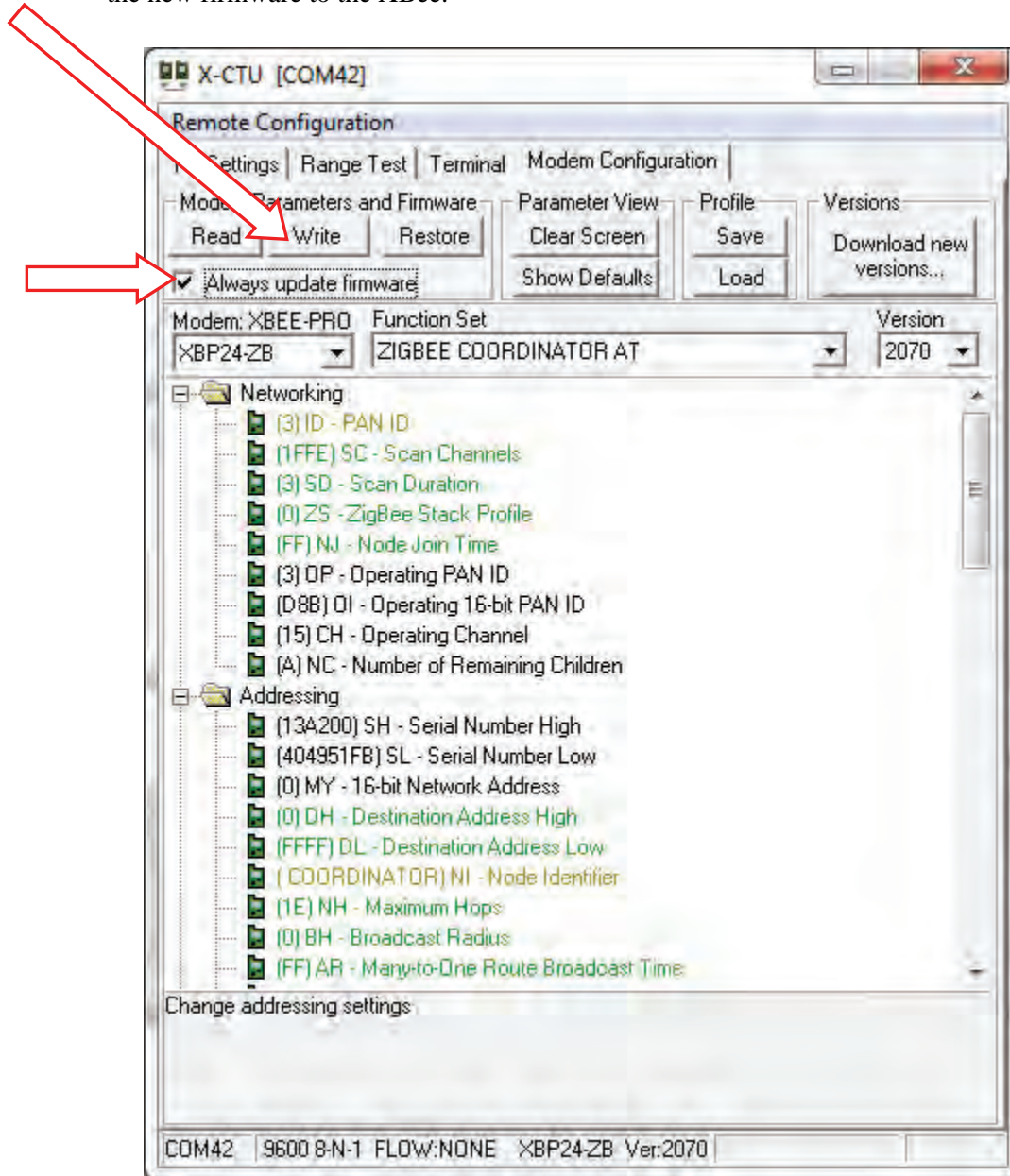


**Figure 8-3: X-CTU Modem Configuration Read Pane**

- ✓ It is desirable to set a specific Personal Area Network (PAN) ID number to differentiate your XBee network from others. You can find this option under the Networking section of the configuration pane. Chose a PAN ID number for your network and enter it into the PAN ID field. Make sure that you write this PAN ID number down—later we will need to set the Router with this number so that it joins our Coordinator’s network. I set my Coordinator’s PAN ID arbitrarily to “3.”
- ✓ Under the Addressing option, write down the values for Serial Number High and Serial Number Low—we will need them later when configuring the Router. The value that these options are set to is located within the parentheses preceding the option name. For example, my module’s Serial Number Low (in the picture above) is 404951FB. An XBee module’s serial number is NOT user configurable.

## 8: XBee Series 2 Quick Start

- ✓ It is recommended, but not required, that you set the Node Identifier option. The Node Identifier is a user configurable text name that a user can set to easily identify a module. To keep things simple, I named my coordinator XBee module “COORDINATOR”.
- ✓ Now we will write these changes to the module. Check the Always update firmware check box and click the Write button. Checking that box ensures that the X-CTU utility will write the new firmware to the XBee.



**Figure 8-4: X-CTU Modem Configuration Read Pane**

- ✓ Once the write process has successfully completed, click the Read button to get the module’s current settings. Double check that everything was written correctly. Look at the PAN ID and Node Identifier and verify them with the information you recorded.
- ✓ If everything checks out, then remove the newly programmed Coordinator from the XBee Adapter Board and set the XBee ZB module aside—we will need to write some additional

information to it a bit later. Make sure that you properly disconnect power to the module before you remove it from the adapter.

### Configuring an XBee ZB module as the Router

- ✓ Insert the second XBee ZB module into the adapter board for configuration. Power your XBee on (plug in the USB and/or apply power).
- ✓ Using the X-CTU utility, click on the Read button to read the module's current configuration. Some important configuration options are denoted with a red arrow.

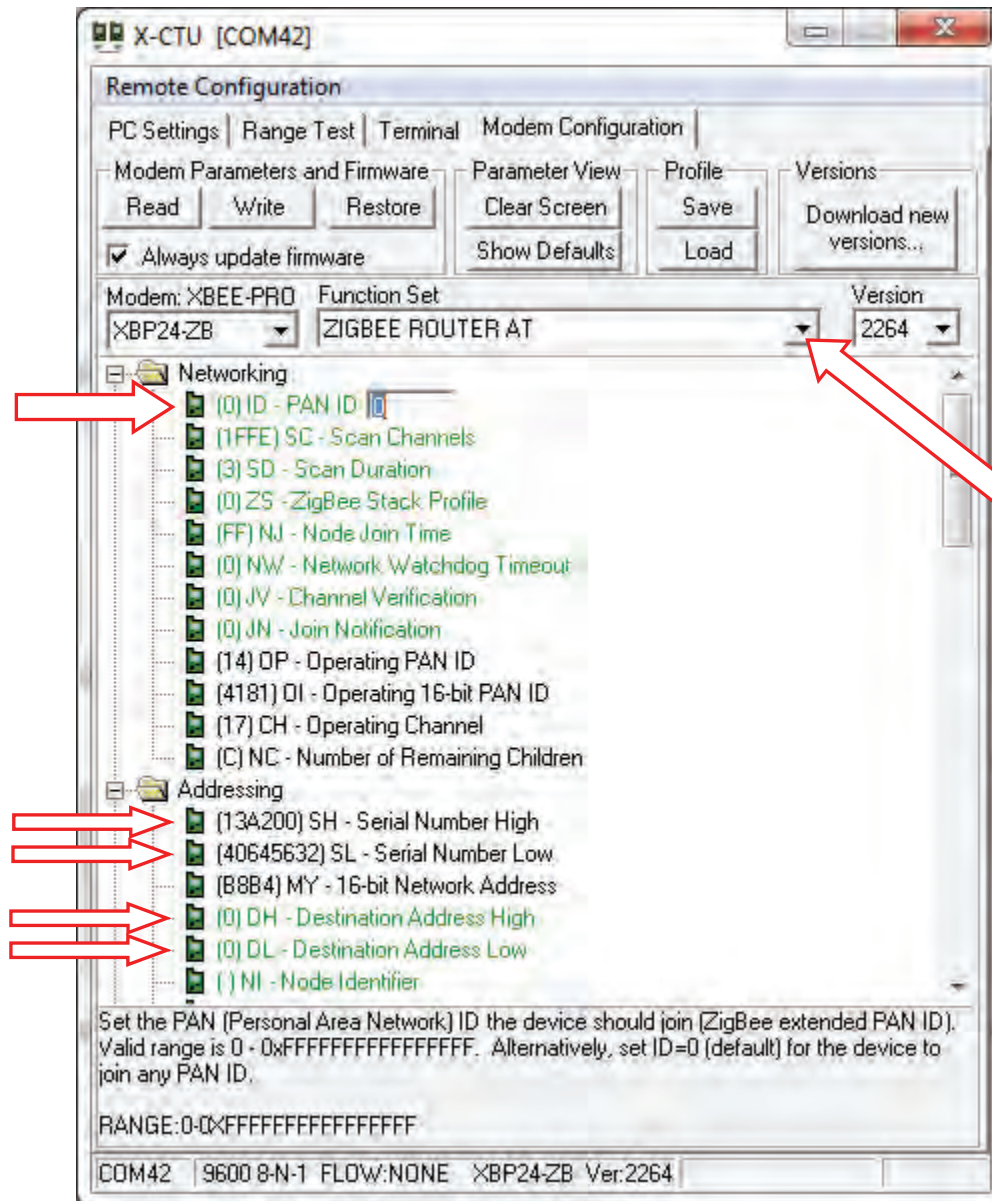


Figure 8-5: X-CTU Modem Configuration Read Pane

- ✓ Set the Function Set option to **ZIGBEE ROUTER AT**. XBee ZB modules purchased from Parallax are distributed with the ZIGBEE ROUTER AT firmware pre-installed by default so setting this may not be necessary.

## 8: XBee Series 2 Quick Start

- ✓ Set the PAN ID of this Router module to match the value that you entered into the Coordinator.
- ✓ Write down both the Serial Number High and Serial Number Low values. These values are represented as hex numbers within the parentheses preceding the option name. You might make a note indicating that the address is for the Router. Later, we will need these values in order to program the Coordinator to only talk with this Router.
- ✓ Change both the Router's Destination High and Low Address values to the Coordinator's High and Low addresses respectively. The Coordinator's Serial Number High should be entered into the Router's Destination Address High setting. Next, the Coordinator's Serial Number Low should be entered into the Router's Destination Address Low setting. This ensures that the Router only talks to the Coordinator.

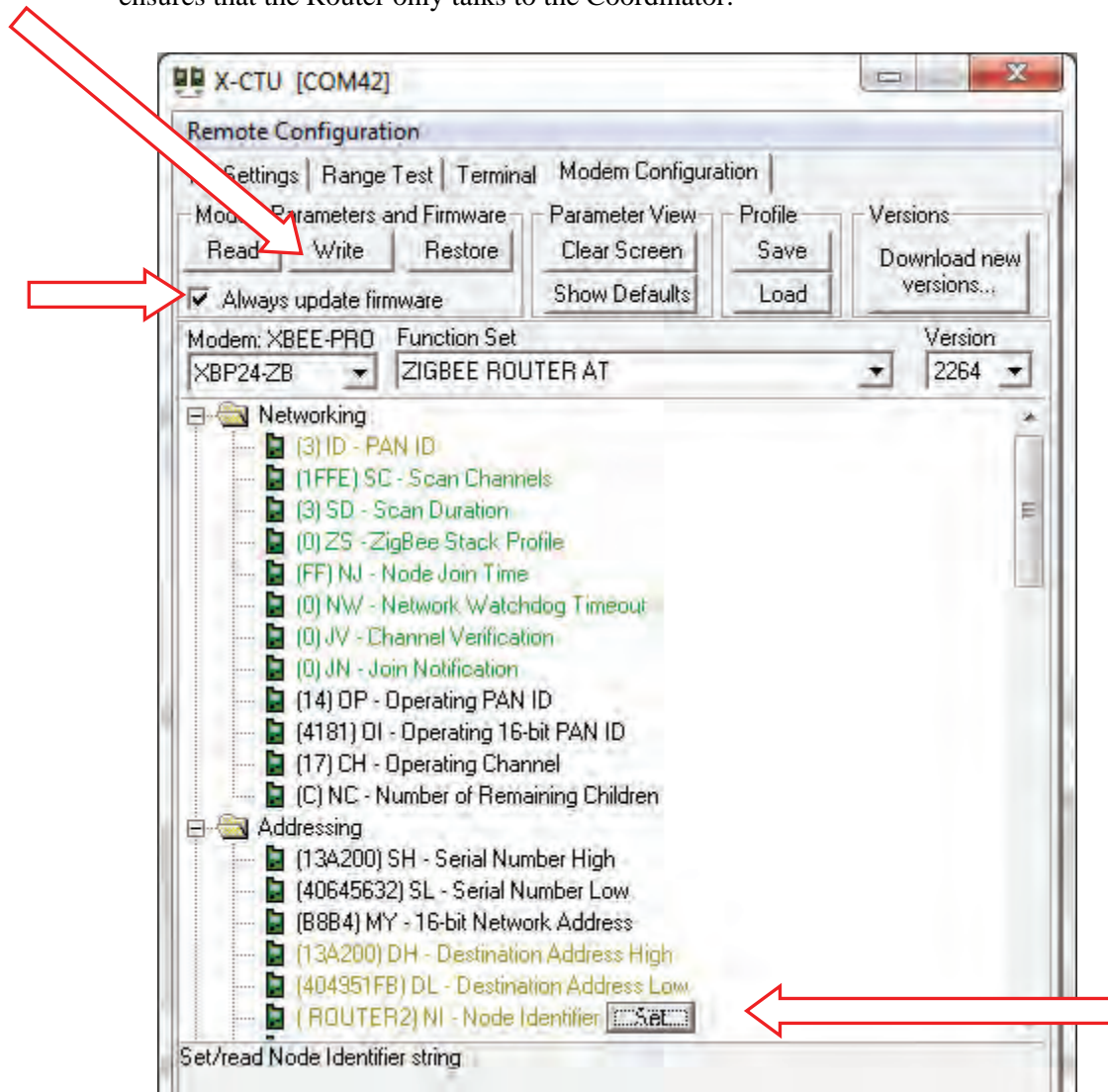


Figure 8-6: X-CTU Modem Configuration Read Pane

- ✓ It is recommended that you give the Router a text name so that you can easily identify the node. Do this by changing the Node Identifier field. In my example, I named my Router "ROUTER2".
- ✓ Check the Always update firmware check box to force X-CTU to update the XBee module's firmware.

- ✓ Click the Write button on the X-CTU utility to write our changes to the module.
- ✓ Once the write process has successfully completed, click the “Read” button to verify the programmed settings. Double check that the PAN ID, Node Identifier, and Destination High and Destination Low addresses all have been written. Verify that the Router’s Destination High and Destination Low addresses match the Serial Number High and Low values of the Coordinator.
- ✓ Disconnect power to and remove the newly programmed Router from the XBee Adapter Board and set the XBee ZB module aside—we will need to write some additional information to Coordinator.

### Configuring an XBee ZB module as the Coordinator—Part 2 of 2

- ✓ Insert the Coordinator into the adapter board for configuration. Power your XBee on (plug in the USB and/or apply power).
- ✓ Click the Read button to get the configuration of the Coordinator.
- ✓ Set the Coordinator’s Destination Address High and Destination Address Low to the recorded values for the Router’s Serial Number High and Serial Number Low respectively. This ensures that the Coordinator only talks with the Router.

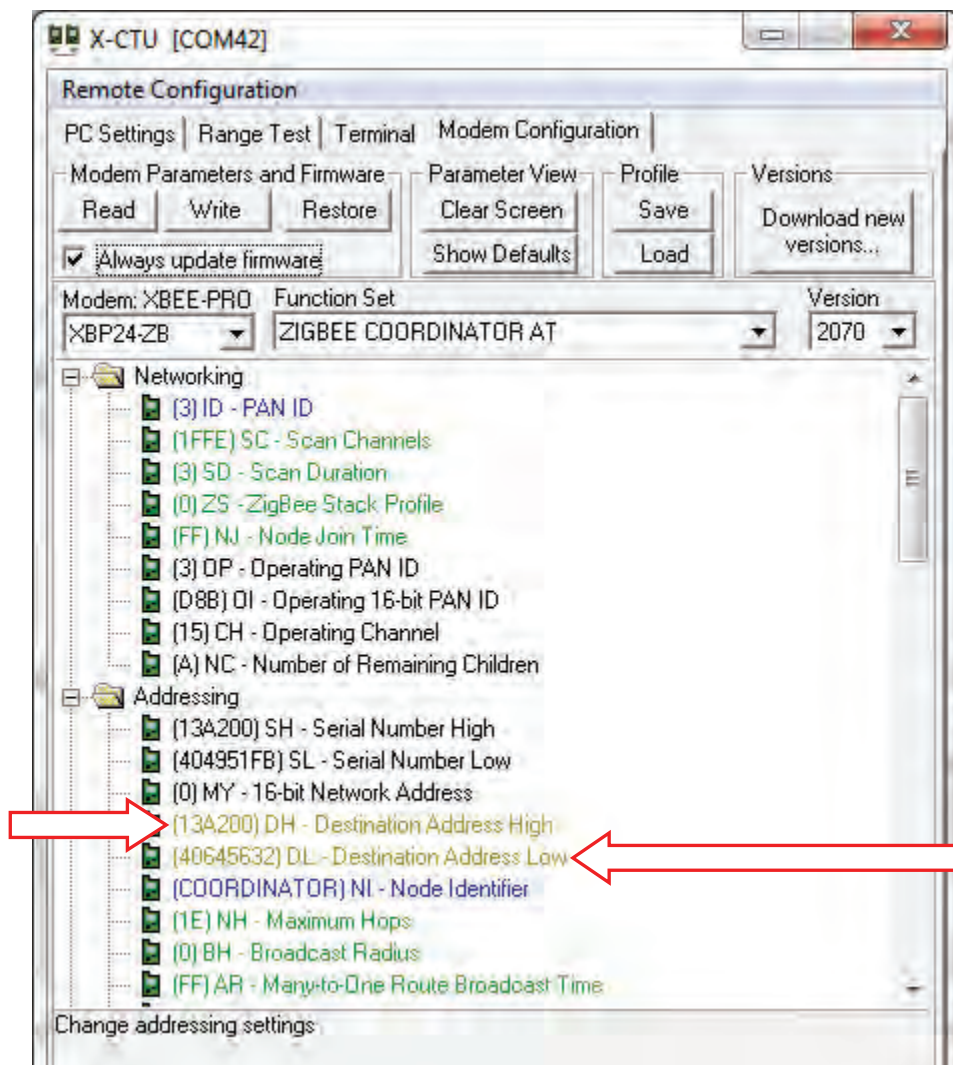
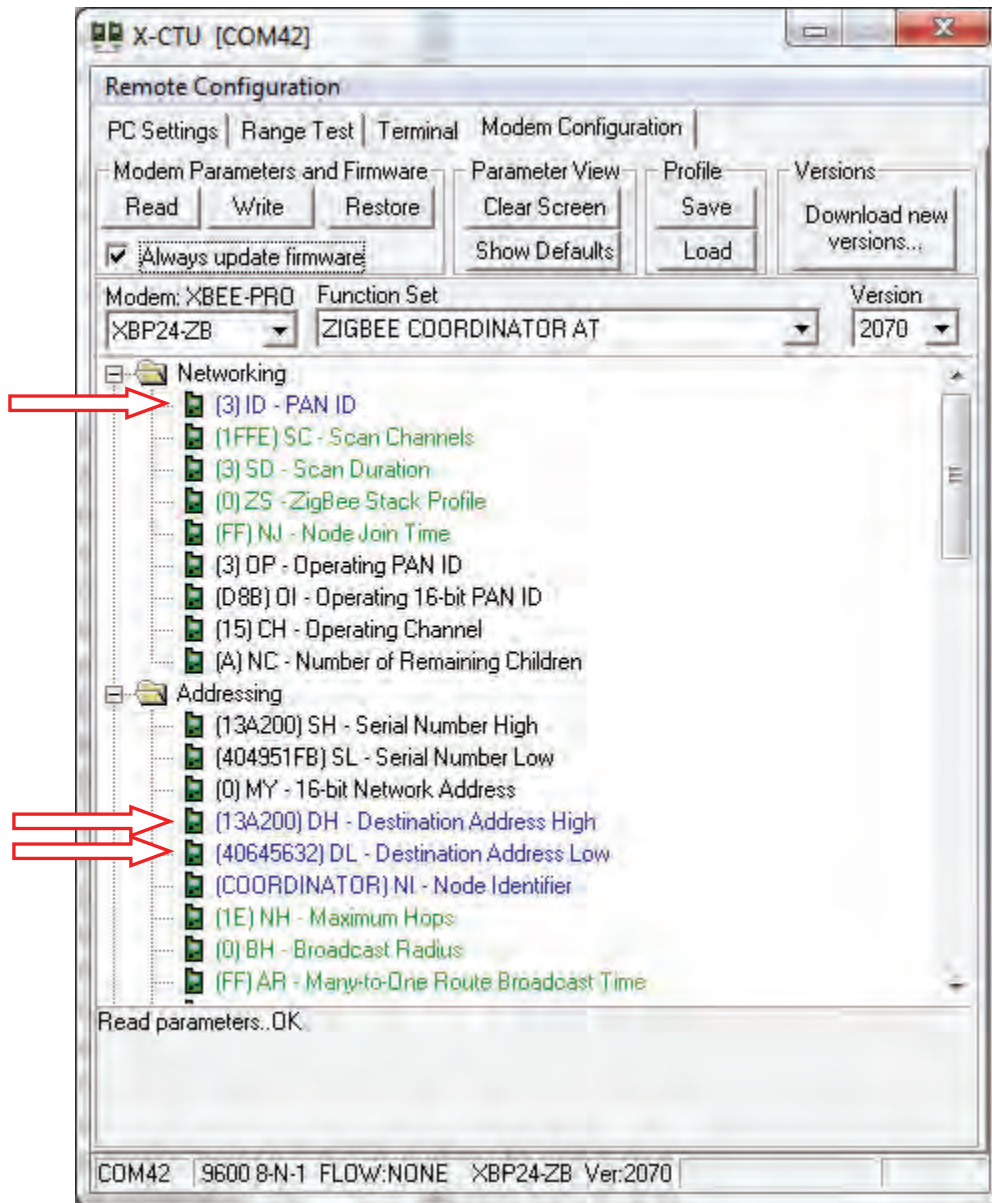


Figure 8-7: Update Coordinator’s Destination Addresses with Router’s Serial Numbers

## 8: XBee Series 2 Quick Start

- ✓ Click the Write button on the X-CTU tool to write the changes to the Coordinator.
- ✓ Finally, click the Read button again and verify that your settings have been properly written. Double check that the PAN ID, Node Identifier, and Destination High and Destination Low addresses all are correct. Verify that the Coordinator's Destination High and Destination Low addresses match the Serial Number High and Low values of the Router.



**Figure 8-8: Update Coordinator's Destination Addresses with Router's Serial Numbers**

That's it! You are done configuring your XBee ZB modules. By this point, you should have done the following:

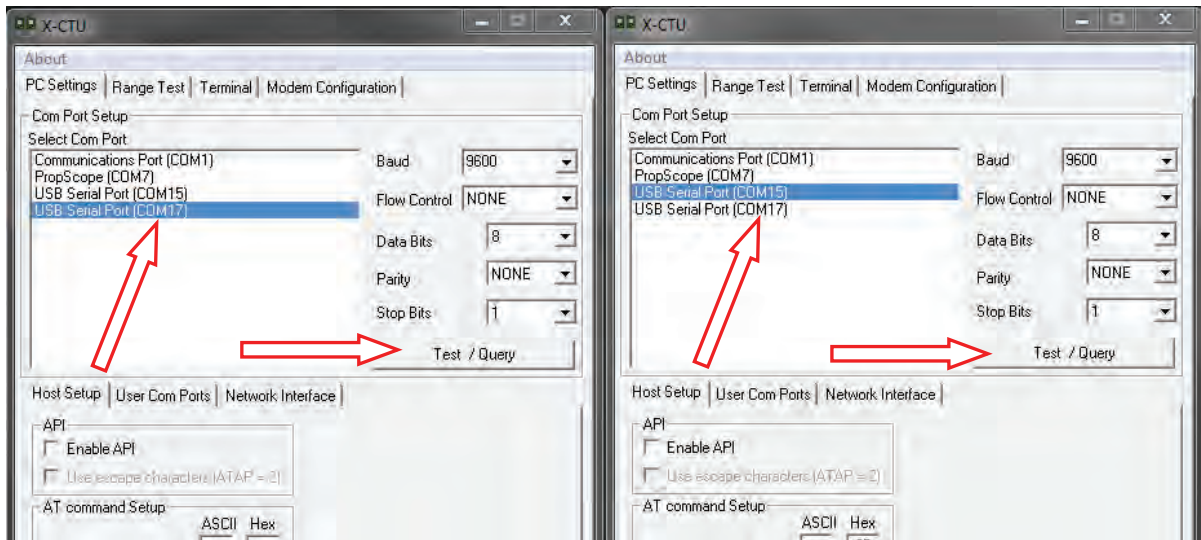
- Configured an XBee ZB module as a Coordinator
- Configured an XBee ZB module as a Router
- Configured the Coordinator to only send data to the Router
- Configured the Router to only send data to the Coordinator



## Testing Your XBee ZB Network

To test your new XBee ZB wireless network follow these instructions:

- ✓ Using the second USB XBee Adapter board, insert both XBee modules, plug in both USB cables, and open up two instances of the X-CTU utility software.
- ✓ In one X-CTU window, select one COM port for one module and in the other X-CTU window, select the COM port for the other module. To verify your connection, click Test/Query in both windows to test communication with both modules.

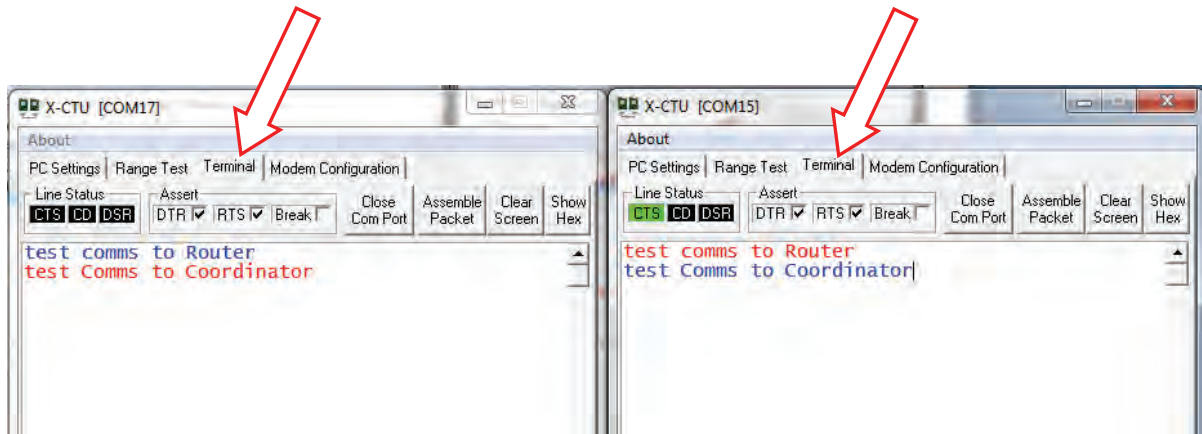


**Figure 8-9: Update Coordinator's Destination Addresses with Router's Serial Numbers**

- ✓ Once you have communication with both XBee ZB modules, click on the Terminal tab in both X-CTU windows as shown in Figure 8-10.
- ✓ If you have oriented your X-CTU windows side-by-side like I have, type in the blank space in the left window. You should see your text in blue on the left, indicating that it was sent to the COM port for transmission. On the right, you should see your text echoed in red, indicating that it was received by the XBee and sent back to the computer.

## 8: XBee Series 2 Quick Start

---



**Figure 8-10: Entering and Viewing Text in Terminal Tabs**

If your test was successful, then congratulations! You have completely configured your XBee modules! They are ready for use in whatever project you have that requires wireless serial communication. Remember, these two modules are configured to communicate only with each other. If you expand your network, you will need to configure your modules differently to allow for addition communication within the network. I would highly recommend reading the datasheet for Digi International's XBee ZB module. Have fun making things wireless!

## Revision History

### Version 1.0.1

Global: Series 1 references changed to 802.15.4; Series 2 references changed to ZB.

Page 31: corrected text of Fixed text of Figure 3-8.

Page 34: replaced Figure 3-9.

Page 109: replaced image Figure 6-5.

Page 134: Deleted last paragraph under figure..

Page 161: deleted information box..