

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 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 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 the 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 Topologies and Communication Strategies

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.

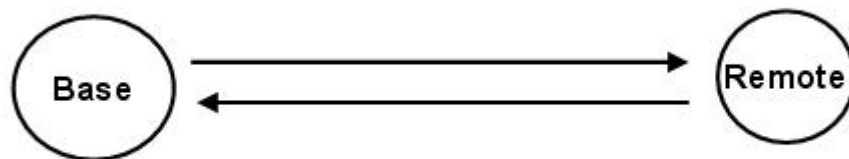


Figure 5-1: Point-to-point Communication Between Two Nodes

Should another pair of XBee 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.

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 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 remote is always sent to the base's address. If needed, the base can manage between nodes and control, such as a remote sensor reading controlling an actuator on another node.

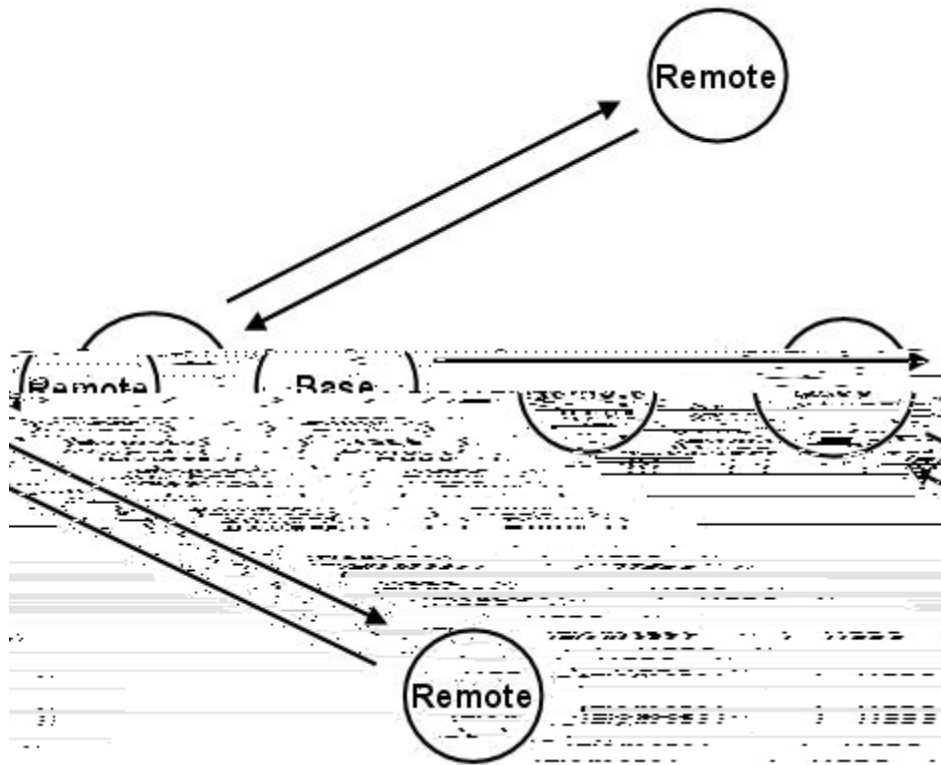


Figure 5-2: Point-to-multipoint—Base Node Communicating Individually to Multiple Remote Nodes

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).

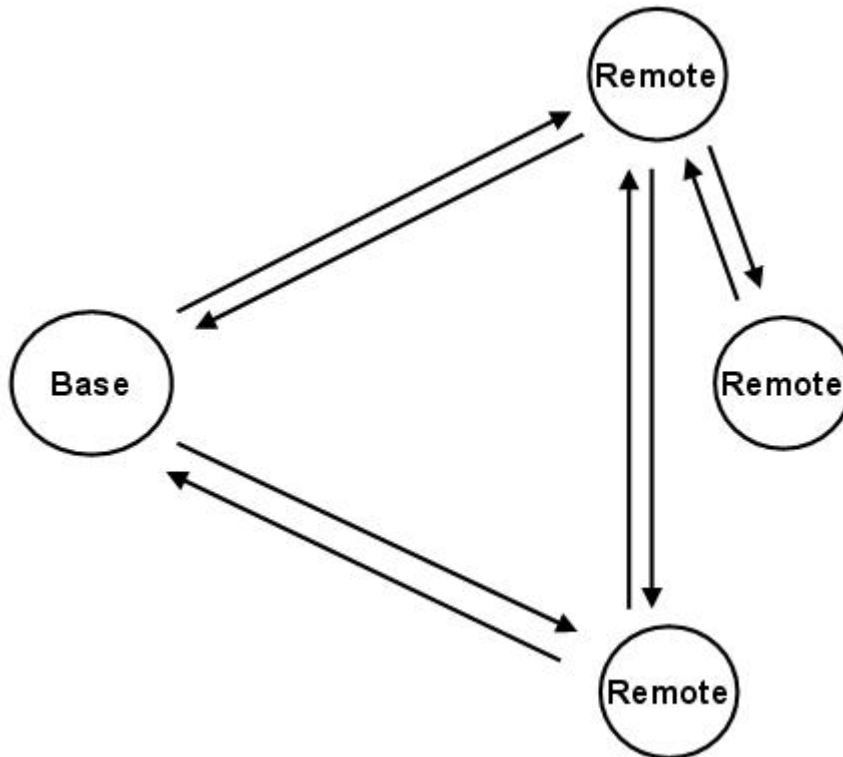


Figure 5-3: Point-to-multipoint—All Nodes Able to Talk to Other Nodes

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 the all destinations. While control actions may work well with a broadcast, such as energizing a remote LED, requesting a value be returned, such as a sensor reading, is problematic since all nodes will attempt to respond.

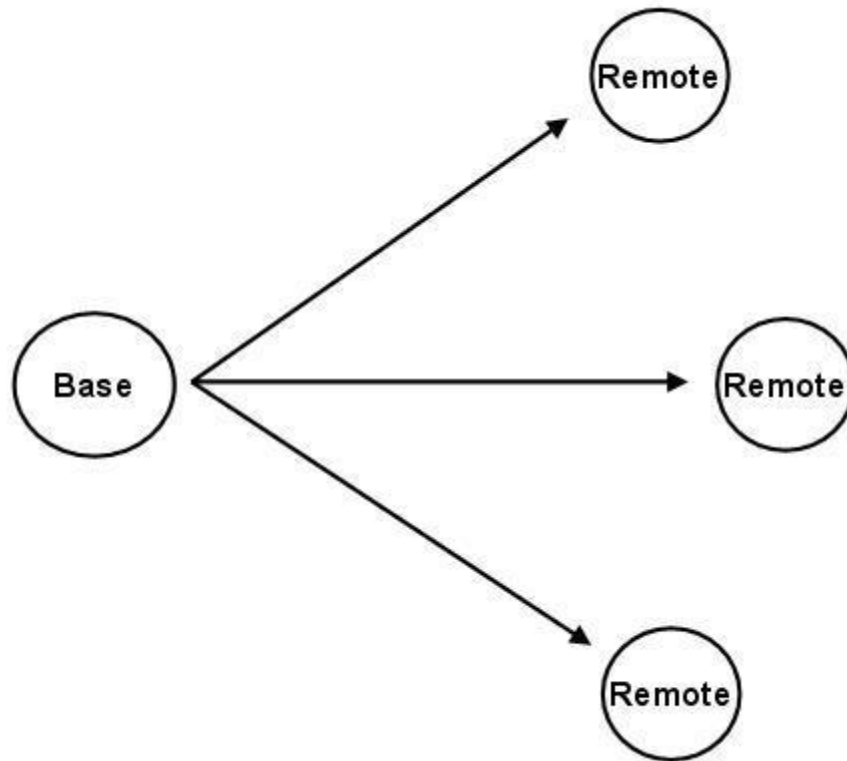


Figure 5-4: Point-to-multipoint—Broadcasting Data to All Nodes

Polling Strategy

In moving data in our network, some control of which 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 goes through a list or range of remote addresses communicating with each in turn 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 controlling communication and remotes having to remain alert, the base may be the passive node waiting for scheduled updates from the remotes. 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 does not initiate the remote's transmission, it needs to have some method of determining the source of the received message. In addition to the remote's data, it also

needs the remote's address so that it may respond to send updates to the active 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

- | | |
|--|---|
| <ul style="list-style-type: none"> (2) BASIC Stamp development boards (2) XBee Adapter Boards (XBee 5V/3.3V Adapter Board or XBee SIP Adapter) (2) XBee modules with default settings (1) Pushbutton (1) LED (1) Buzzer (piezospeaker) | <ul style="list-style-type: none"> (1) Servo (1) Phototransistor (1) 0.01 μF capacitor (1) 10 kΩ Resistor (1) 220 Ω Resistor Optional: XBee USB Adapter and additional 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 5-5 for the base and at least one remote (multiple remotes may be constructed for testing multi-node communications). The base will use the Debug Terminal for control and notifications in most examples.

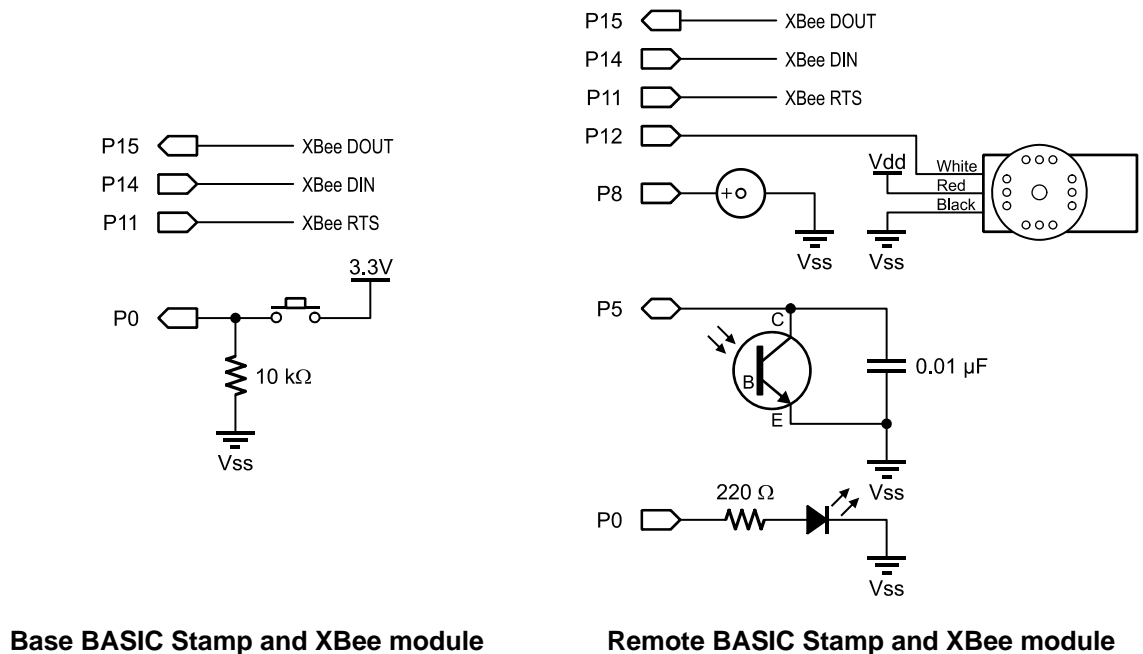


Figure 5-5: BASIC Stamp Base and Remote Schematics

5: Network Topologies & Control Strategies

Using Point-to-Point for Pushbutton Control

In this example a pushbutton on the base is used to control the remote 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 is pressed, the value of variable **Freq** is incremented by 500 and sent as a decimal value to the remote, 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 side, 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}

#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 BASIC Stamp.
- ✓ Download Simple_Control_Remote.bs2 to your remote BASIC Stamp.
- ✓ Press and press and hold the base's pushbutton to test.



Optional Testing and Monitoring

If you have an XBee at address 0 on USB and using X-CTU, you can enter values to be sent to the remote unit 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 to control and monitor the remote nodes. The base unit prompts the user for information to control the remote's LED, buzzer, servo or to request the reading of the photo resistor. The first step is to enter the remote node to be addressed. 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). It requests from the user the address of the node to control. This is based on constants in the remote's code for the node's address. In testing, only a single remote may be used, but you are free to set up as many as you desire.

5: Network Topologies & Control Strategies

After accepting the remote's address, it 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 request 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 reading and repeats.

By setting the guard time low, the BASIC Stamp can configure the XBee within 20 or so milliseconds 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.

```
' *****
' 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

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

```

5: Network Topologies & Control Strategies

```
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'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 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.

```
' *****
' 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

```

5: Network Topologies & Control Strategies

```
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                               ' Give base time to set up
    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
```

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 hardware.
- ✓ Repeat steps above for any additional remotes, 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, answer requested information using both the used (as shown in Figure 5-6) and unused remote 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 \$.

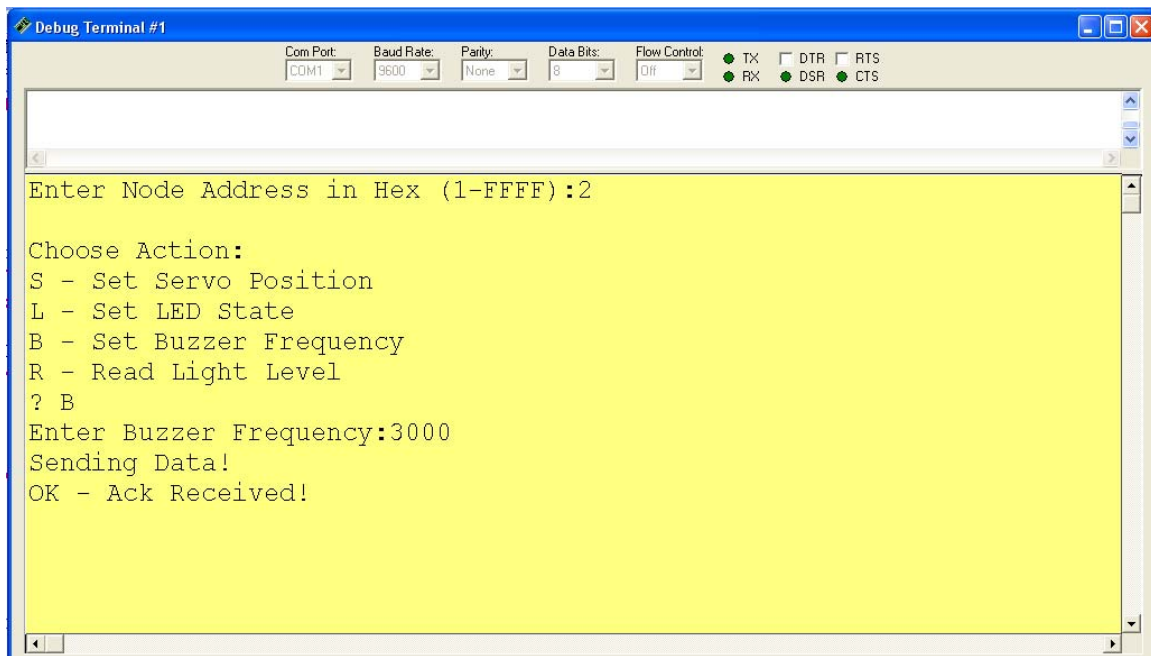


Figure 5-6: Manual Polling Debug Terminal

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 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 to the remote by assigning the XBee the remote's address. It may be used to monitor data from the remote by assigning it the base's address. By assigning the destination address to that of the remote, 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 to the base using the normal base hardware to send data and used the Debug Terminal and base unit to poll remote nodes.

5: Network Topologies & Control Strategies

- ✓ Place the USB connected XBee on a remote's address (ATMY 2) to monitor data from the base to the remote.
- ✓ Place the USB-connected XBee to sent to the remote unit's address (ATDL 2) with the MY address of 0 (ATMY 0).
- ✓ Send data to a remote 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 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
    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 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 remotes – include a range beyond your actual remotes 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: Off
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 unit's address (ATDL 2) to monitor polling data from the base.

Scheduled Updates from Remote Units

Allowing the remote units to make initial contact with the base allows them the ability perform other processes or 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 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 unit will send “C” and all its current values to the base 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 incoming data.
- The base will configure DL for the remote's address using fast Command Mode.
- The base will calculate changes and send out “U” and all updated values to the remote in a single transmission.

5: Network Topologies & Control Strategies

- The remote waits for a short time for “U” and updated data.
- Base is ready for an update from another unit.

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, 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

5: Network Topologies & Control Strategies

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 `Sceduled_Remote.bs2`; please see distributed files for full listing:

- ✓ 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.

```
Unit $2 Reports
Light Reading: 20
LED State: 0
Frequency: 3500
Position: 1000

Setting buzzer to frequency of: 4000
Setting LED to state of: 1
Setting Servo to position of: 500
.....
.....
```

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 remotes 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 | (1) Phototransistor |
| (2) XBee adapter boards | (1) 0.01μF capacitor |
| (2) XBee with default settings | (1) 10 kΩ Resistor |
| (1) Pushbutton | (1) 220 Ω Resistor |
| (1) LED | Optional: XBee USB Adapter and XBee module |
| (1) Buzzer | |



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 and at least one remote (multiple remotes 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).

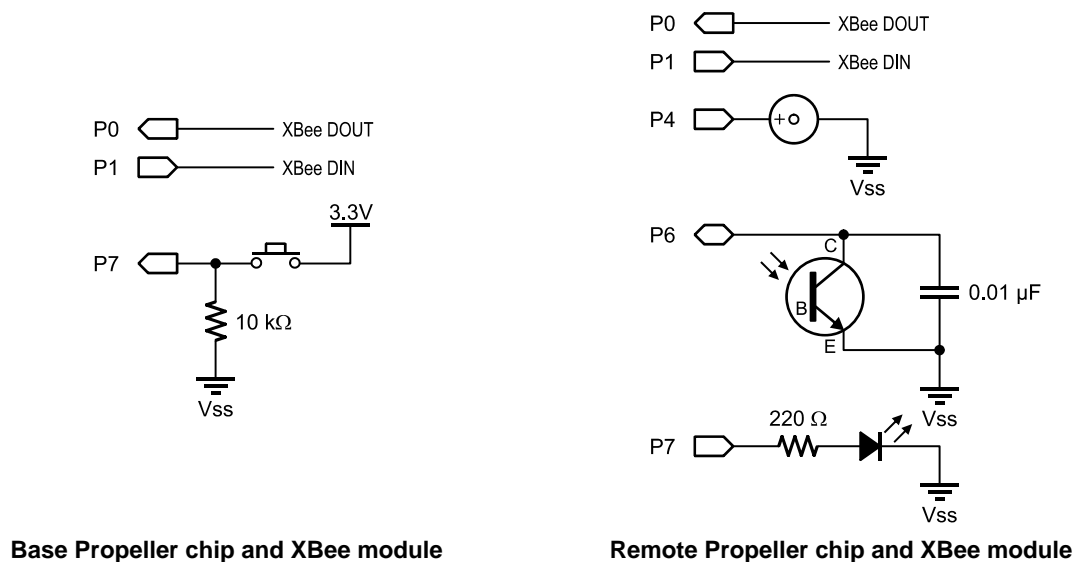


Figure 5-9: Propeller Base and Remote Schematics

5: Network Topologies & Control Strategies

Using Point-to-Point for Pushbutton Control

In this example, the base 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 will increase `State` and `Freq`, not pressing the button will ramp down `State` and `Freq`. The base will continually send “!” and the 2 decimal values to be received and used.

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 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 your remote hardware.
- ✓ Download `Simple_Control_Base.spin` to your base hardware.
- ✓ Press and release the pushbutton on the base while monitoring the remote'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 through the Assemble Packet window such as !200 (Enter) 2000 (Enter). X-CTU may also be used to monitor data from the base to the remote.

Point-to-Multipoint—Manual Polling of Remote Nodes

In this example the Propeller base requests and accepts input from the user via the PC and a terminal window. The base requests the remote node's address to control or read and control action (control LED, control buzzer or read phototransistor). The base also accepts an application acknowledgement from the remote to help ensure actions were accepted.

The base code uses `.AT_INIT` to switch to fast Command Mode for updates. The base address is set to 0. The code uses the `XBee_Object` for the PC communications driver *as well as it is more fully featured to accept decimal input from the terminal*. The code requests the remote's address and uses `.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 by the Propeller `FullDuplexSerial.spin` drivers (up to 15 characters), `.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))
```

5: Network Topologies & Control Strategies

```
PC.str(string("B - Control Buzzer",CR))
PC.str(string("R - Read Sensor",CR))
DataIn := PC.Rx          ' Accept action
XB.RxFlush

case DataIn
  "L","l": 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

  "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's code, the program configures for fast Command Mode and configures the MY address of the remote 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 remotes (default is 2).
- ✓ Download the code to the remote hardware.
- ✓ Repeat above if multiple remotes are used.
- ✓ Download Manual_Polling_Base.spin to the base hardware (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 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 addresses.
- ✓ Use an address of FFFF to test a broadcast to all nodes for a control action.

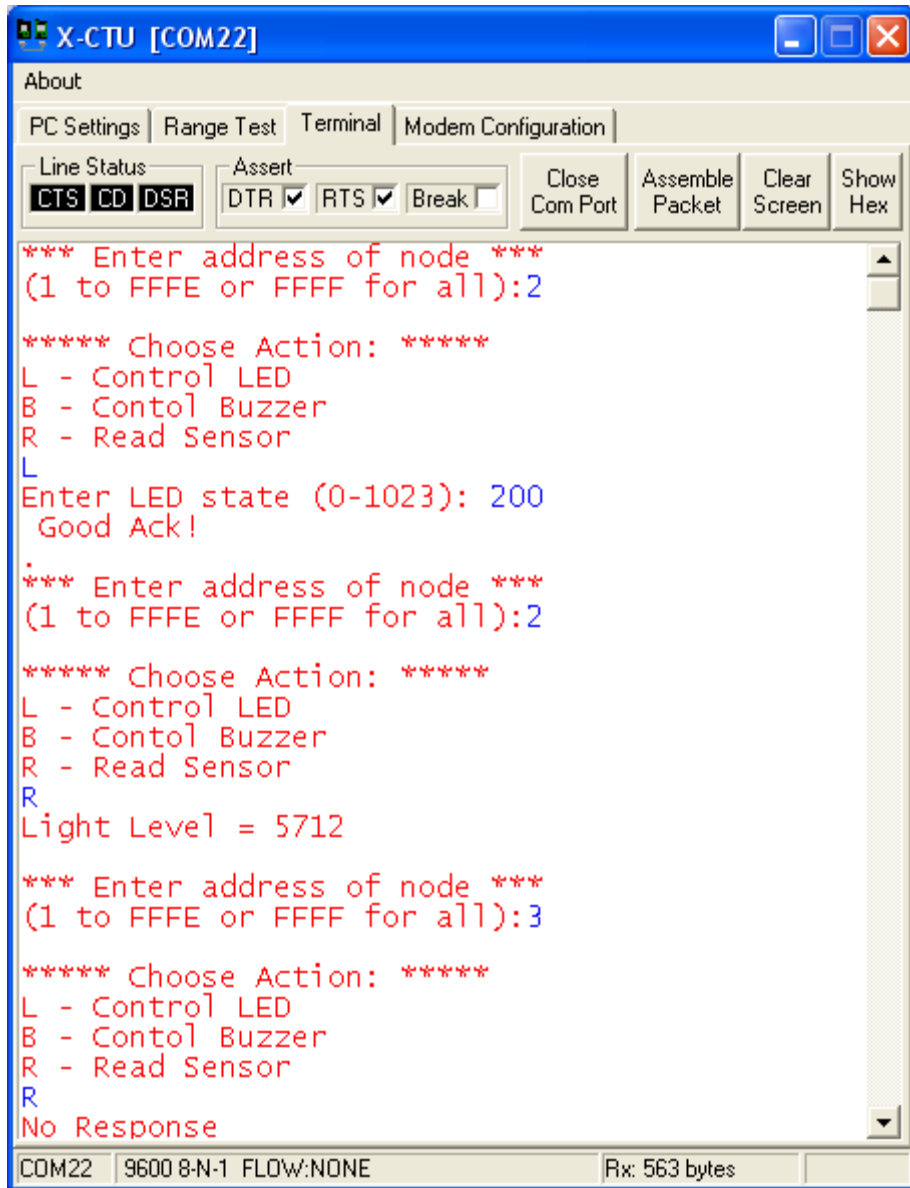


Figure 5-10: Manual Polling Base Terminal

Using a USB XBee for Monitoring and Control

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

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

With automatic polling, the base 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 accepted and the status is displayed. Polling_Remote.spin is used on the remote devices.

The base 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 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)

```

5: Network Topologies & Control Strategies

```
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 dB:  "))
  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'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.
- ✓ Open Automatic_Polling_Base.spin. Modify DL_Start and DL_End to encompass a range of addresses included in your remotes – include a range beyond your actual remotes to test what happens when node is not present (By default, range is addresses 1 to 3).
- ✓ Download to your base hardware.
- ✓ Using a terminal window, monitor the polling as shown in Figure 5-11.

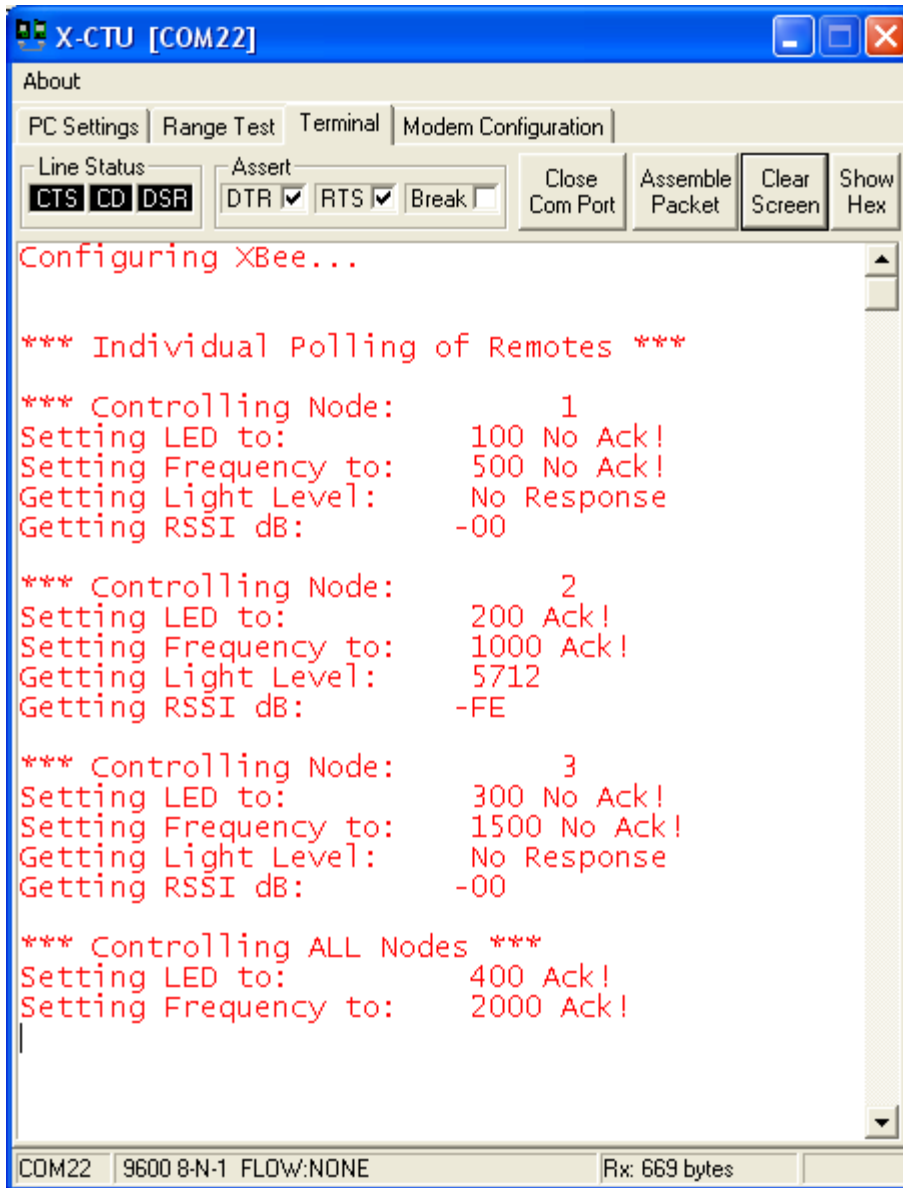


Figure 5-11: Automatic Polling Base Terminal

Scheduled Updates from Remote Units

Allowing the remote units to make initial contact with the base 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. It also allows a remote 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 and to the remotes 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.

5: Network Topologies & Control Strategies

- The remote unit will send “C” and all its current values to the base 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’s address using fast Command Mode used by .AT_Config.
- The base will calculate and send out “U” and all updated values to the remote in a single transmission.
- The remote waits for a short time for “U” and updated data.
- Base is ready from 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

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 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 code can still accept updates at anytime should data be sent from the base or USB, 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

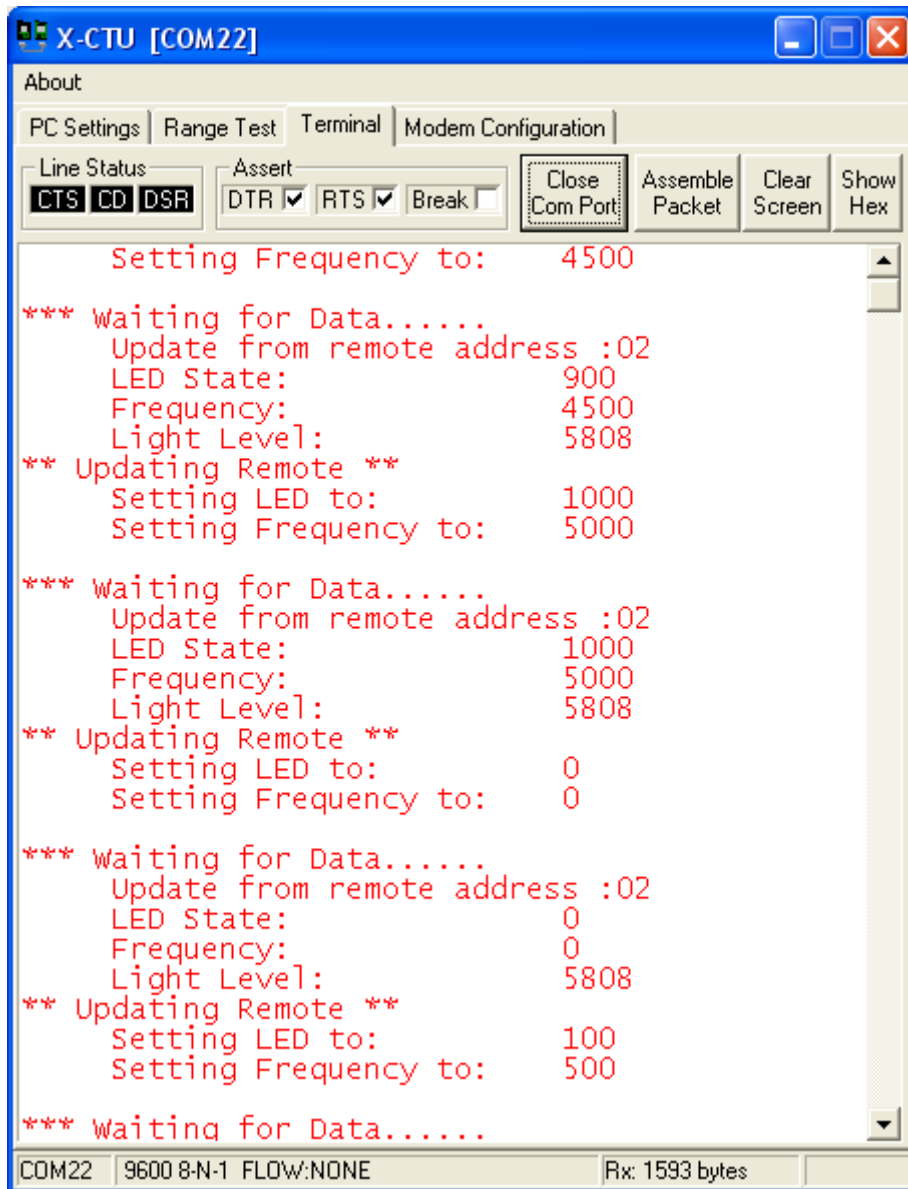
  XB.RxFlush           ' Ensure buffer empty
  repeat
    DataIn := XB.Rx    ' Accept incoming byte character
    case DataIn
      "L": DataIn := XB.RxDecTime(500) ' If L, accept data for LED
          if DataIn <> -1             ' Ensure not timeout value, -1
            State := DataIn          ' Accept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

```

5: Network Topologies & Control Strategies

- ✓ Open Scheduled_Remote.spin
- ✓ 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.spin to your base hardware.
- ✓ 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 powered up at once, such as:

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

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 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.