Simple Sensor Network Activity

* This section borrows heavily from Rob Faludi's *Wireless Sensor Networks, Chapter 5: API and a Sensor Network – thanks Rob!*

This project can serve as a starting point for almost any wireless sensor network you'd like to build. Each member of the class will set up a sensor that sends three analog sensor values to generate Red, Green, and Blue (RGB) values on the screen. When we're done, we'll play a little color matching game.

Each of your sensor boards will act as router, talking to a common base station (acting as the coordinator) connected to a computer running Processing. This base station will parse out the sensor values and display them on screen.

Parts

For each sensor board:

- 1 Solderless Breadboard
- 1 XBee Series 2 Wireless Radio (configured as a ZigBee Router AT mode)
- Jumper wire / Hookup wire
- Power source (3.3v line from your Arduino, or 2AA battery pack)
- 3 Trim Pots (or other analog sensors, though the pots tend to work well for this particular exercise)
- 1 XBee breakout board

For the coordinator radio: (if you want to do this at home)

- 1 Xbee Series 2 Wireless Radio (configured as a ZigBee Coordinator API mode)
- XBee USB Explorer Board (connected to a computer running Processing)

Simple Sensor Network

SparkFun Electronics Summer Semester

Step 1: Prepare Your Coordinator Radio (if doing this at home)

Using X-CTU and your XBee Explorer, configure your Coordinator radio to be a ZigBee Coordinator in API mode (refer to the earlier handout if you get stuck). The coordinator must be in API mode for this exercise because i/o data is only delivered in API mode.

Step 2: Prepare your Router Radio

Using X-CTU and your XBee Explorer, configure your Router radio to be a ZigBee Router in AT mode (refer to the earlier handout if you get stuck).

In X-CTU or CoolTerm, make sure you have the following settings on your router radio:

PAN ID: **7777** (since all the router radios have to be on the same PAN ID as the coordinator)
ATDH: **0** (the 16-bit network address of the coordinator is always 0)

ATDL : **0** (also a 0, because that's the fixed address for the coordinator)

ATJV1: To ensure your router attempts to rejoin the network on startup

ATD02, **ATD12**, **ATD22**: These commands put Pins 0, 1, and 2 in Analog mode

ATIR3E8: Sets the sample rate at 1000 milliseconds (hex 3E8)

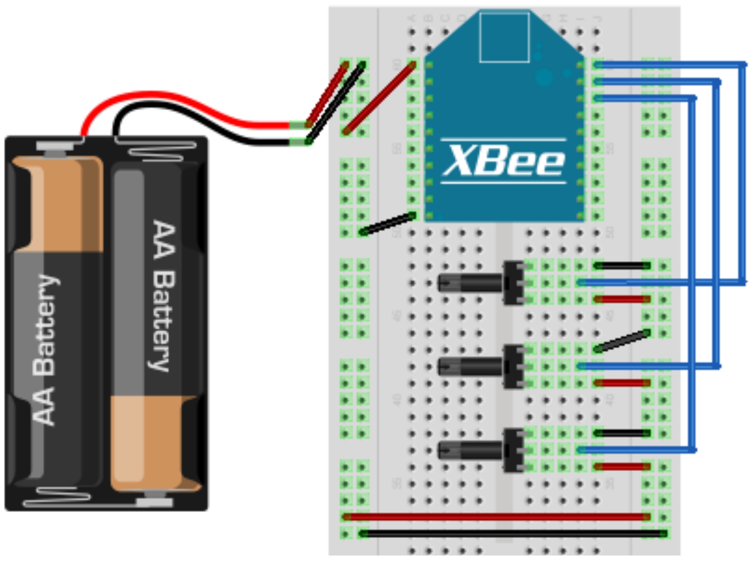
ATWR: Write your settings to non-volatile memory and sets them as the default (otherwise, they'll be reset when you unplug from the USB Explorer)

• It's never a bad idea to check that your settings are correct. To check the current value of a setting on the radio, enter the AT command without the setting number (e.g. ATDH instead of ATDH 0, ATD0 instead of ATD02, ATID instead of ATID7777, etc.) – you should get back a number that corresponds to the setting you entered.

Step 3: Prepare your sensor board

1. Place your configured Router radio in an XBee breakout shield, and place the breakout shield in your breadboard, across the middle divider.

2. Place the three potentiometers (or other analog sensors) on the breadboard, and hook them up to analog pins 0, 1, and 2 on the Xbee (see the wiring diagram below for help):



Remember the pin numbering on the Xbee (Pin 1 is on the upper left, going down the left side, pin 11 is on the bottom right, with numbers going up the right side). Don't forget to hook up power (pin1) and ground (pin10).

Note: if you're using different analog sensors, remember to hook them up properly (don't forget necessary power requirements, capacitors, resistors, voltage regulators, etc.)

Great! You should be ready to start sending your values to the controller!

Step 4 (Optional): Programming your Processing Sketch (for at-home use)

The Processing sketch for the controller XBee is a bit more complex than normal in terms of interfacing with sensors, since we have to deal with receiving them wirelessly from the router XBee's, not just through the Serial port from an Arduino. If you want to run this code at home, you'll also need the Xbee-Api Java Libraries from Andrew Rapp, which you can download here: http://code.google.com/p/xbee-api/downloads/list

You'll want to put the log4j.jar and xbee-api-version#.jar files in a 'code' folder inside your Processing sketch folder (further instructions in the comments at the beginning of the code).

```
/*
 * Matching color game from sets of Analog sensors
 * by Ben Leduc-Mills
 * based off code by Rob Faludi http://faludi.com
 */

// used for communication via xbee api
import processing.serial.*;

// xbee api libraries available at http://code.google.com/p/xbee-api/
// Download the zip file, extract it, and copy the xbee-api jar file
// and the log4j.jar file (located in the lib folder) inside a "code"
// folder under this Processing sketch's folder (save this sketch, then
// click the Sketch menu and choose Show Sketch Folder).
import com.rapplogic.xbee.api.ApiId;
import com.rapplogic.xbee.api.PacketListener;
import com.rapplogic.xbee.api.XBee;
import com.rapplogic.xbee.api.XBeeResponse;
import com.rapplogic.xbee.api.zigbee.ZNetRxIoSampleResponse;

String version = "1.1";

// *** REPLACE WITH THE SERIAL PORT (COM PORT) FOR YOUR LOCAL XBEE ***
String mySerialPort = "/dev/tty.usbserial-A40084zG";

// create and initialize a new xbee object
XBee xbee = new XBee();

int error=0;

// make an array list of thermometer objects for display
ArrayList colorBoxes = new ArrayList();
// create a font for display
PFont font;
int R, G, B;
float value1, value2, value3;  //values from the 3 sensors
int tolerance = 20;  //how close do we need to get
```

```
void setup() {
  size(850, 850); // screen size
  smooth(); // anti-aliasing for graphic display

  //set up target colors
  R = round(random(0, 255));
  G = round(random(0, 255));
  B = round(random(0, 255));

  // You'll need to generate a font before you can run this sketch.
  // Click the Tools menu and choose Create Font. Click Sans Serif,
  // choose a size of 10, and click OK.
  font =  loadFont("SansSerif-10.vlw");
  textFont(font); // use the font for text

    // The log4j.properties file is required by the xbee api library, and
  // needs to be in your data folder. You can find this file in the xbee
  // api library you downloaded earlier
  PropertyConfigurator.configure(dataPath("")+"log4j.properties");
  // Print a list in case the selected one doesn't work out
  println("Available serial ports:");
  println(Serial.list());
  try {
    // opens your serial port defined above, at 9600 baud
    xbee.open(mySerialPort, 9600);
  }
  catch (XBeeException e) {
    println("** Error opening XBee port: " + e + " **");
    println("Is your XBee plugged in to your computer?");
    println("Did you set your COM port in the code near line 20?");
    error=1;
  }
}

// draw loop executes continuously
void draw() {
  background(224); // draw a light gray background

  fill(R, G, B);  //set the fill to our randomly generated color
  rect(width/2-125, 10, 250, height-20);  //and draw a nice big rect for people to
match

  // report any serial port problems in the main window
  if (error == 1) {
    fill(0);
    text("** Error opening XBee port: **\n"+
      "Is your XBee plugged in to your computer?\n" +
      "Did you set your COM port in the code near line 20?", width/3, height/2);
  }
  SensorData data = new SensorData(); // create a data object
  data = getData(); // put data into the data object
  //data = getSimulatedData(); // uncomment this to use random data for testing
```

```
 // check that actual data came in:
  if (data.value1 >=0 && data.address != null) {

    int i;
    boolean foundIt = false;
    for (i = 0; i < colorBoxes.size(); i++) {
      if ( ((ColorBox) colorBoxes.get(i)).address.equals(data.address) ) {
        foundIt = true;
        break;
      }
    }

    value1 = data.value1;
    value2 = data.value2;
    value3 = data.value3;

    //if the box exists already, update the color values
    if (foundIt) {
      ((ColorBox) colorBoxes.get(i)).value1 =  value1;
      ((ColorBox) colorBoxes.get(i)).value2 =  value2;
      ((ColorBox) colorBoxes.get(i)).value3 =  value3;
    }

    //if there's room in the array, create a new box
    else if (colorBoxes.size() < 12) {
      // ColorBox(address, sizeX, sizeY, posX, posY)
      colorBoxes.add(new ColorBox(data.address, 260, 40, 20, (colorBoxes.size()) * 61
+ 10));
      ((ColorBox) colorBoxes.get(i)).value1 =  value1;
      ((ColorBox) colorBoxes.get(i)).value2 =  value2;
      ((ColorBox) colorBoxes.get(i)).value3 =  value3;
    }

    //draw the color boxes
    for (int j = 0; j<colorBoxes.size(); j++) {
      ((ColorBox) colorBoxes.get(j)).render();
    }

    //if the R, G and B of the box are close enough, declare a match!
    if ((value1 < R + tolerance && value1 > R - tolerance) &&
      (value2 < G + tolerance && value2 > G - tolerance) &&
      (value3 < B + tolerance && value3 > B - tolerance)) {
      println("Close?  " + R + "  " + value1 + "    " + G + "  " + value2 + "   " + B + "
" + value3);
      //text(((ColorBox) colorBoxes.get(i)).address + " WINS!!!", width/2-125,
height/2-125);
      fill(200, 20, 20);
      textSize(20);
      text(" WINS!!!", ((ColorBox) colorBoxes.get(i)).posX,
      ((ColorBox) colorBoxes.get(i)).posY+20);
    }
    println("Close?  " + R + " " + value1 + "    " + G + " " + value2 + "  " + B + " "
+ value3);
  }
} //end of draw loop


// defines the data object
class SensorData {
  float value1;
  float value2;
  float value3;
  String address;
}

// used only if getSimulatedData is uncommented in draw loop
//
SensorData getSimulatedData() {
  SensorData data = new SensorData();
  int value = int(random(750, 890));
  String address = "00:13:A2:00:12:34:AB:C" + str( round(random(0, 9)) );
  //data.value = value;
  data.address = address;
  delay(200);
  return data;
}

// queries the XBee for incoming I/O data frames
// and parses them into a data object
SensorData getData() {

  SensorData data = new SensorData();
  //int value = -1;      // returns an impossible value if there's an error
  float value1 = -1; //sensor values
  float value2 = -1;
  float value3 = -1;
  String address = ""; // returns a null value if there's an error

  try {
    // we wait here until a packet is received.
    XBeeResponse response = xbee.getResponse();
    // uncomment next line for additional debugging information
    //println("Received response " + response.toString());

    // check that this frame is a valid I/O sample, then parse it as such
    if (response.getApiId() == ApiId.ZNET_IO_SAMPLE_RESPONSE
      && !response.isError()) {
      ZNetRxIoSampleResponse ioSample =
        (ZNetRxIoSampleResponse)(XBeeResponse) response;

      // get the sender's 64-bit address
      int[] addressArray = ioSample.getRemoteAddress64().getAddress();
      // parse the address int array into a formatted string
      String[] hexAddress = new String[addressArray.length];
      for (int i=0; i<addressArray.length;i++) {
        // format each address byte with leading zeros:
```

```
        hexAddress[i] = String.format("%02x", addressArray[i]);
      }

      // join the array together with colons for readability:
      String senderAddress = join(hexAddress, ":");
      //print("Sender address: " + senderAddress);
      data.address = senderAddress;

      // get the value of the first input pin
      value1 = ioSample.getAnalog0();
      //print(" analog value1: " + value1 );

      value2 = ioSample.getAnalog1();
      //print(" analog value2: " + value2 );

      value3 = ioSample.getAnalog2();
      //print(" analog value3: " + value3 );

      data.value1 = value1;
      data.value2 = value2;
      data.value3 = value3;
    }
    else if (!response.isError()) {
      println("Got error in data frame");
    }
    else {
      println("Got non-i/o data frame");
    }
  }
  catch (XBeeException e) {
    println("Error receiving response: " + e);
  }
  return data; // sends the data back to the calling function
}


//Class for color boxes (takes R,G,B values from 3 different sensors)

class ColorBox {

  int sizeX, sizeY, posX, posY;
  float value1, value2, value3;
  String address;

  ColorBox(String _address, int _sizeX, int _sizeY, int _posX, int _posY) {
//initialize ColorBox object

    address = _address;
    sizeX = _sizeX;
    sizeY = _sizeY;
    posX = _posX;
    posY = _posY;
  }
  void render() {

    noStroke();
    //fill(value1, value2);
    fill(value1, value2, value3);
    rect(posX, posY, sizeX, sizeY);

    textAlign(LEFT);
    fill(0);
    textSize(10);

    text(address, posX, posY + sizeY + 10);
  }
}
```