Building a Basic Communication Network using XBee DigiMesh

Keywords: XBee, Networking, Zigbee, Digimesh, Mesh, Python, Smart Home

Abstract:
Using Digi International’s in-house Zigbee module brand XBee and in-house DigiMesh technology a basic ad hoc mesh network can be setup using Xbee DigiMesh 2.4 RF modules to relay data. This capability for basic automated monitoring can be useful for small home automation projects to larger scaled professional needs. Topics discussed are selecting the XBeedigimesh 2.4 RF modules, hardware and software configuration instructions for the XBee and sample code to enable data collection.

Introduction:
Xbee is the in-house brand for Digi International’s Zigbee communication networking modules. Zigbee is an open source networking protocol maintained by IEEE (standard 802.15.4) and is used for mesh networking. It is especially useful for low power, long distance and secure networking with periodic but relatively small amounts of data flow.

The objective of this article is to discuss the technical benefits of using DigiMesh and how an XBeedigimesh network can be configured for basic use. It will also detail needed components, a specific example in monitoring the moisture level of soil around a home garden as well as how to use the XBee Python library to interact with the data.

Steps

Required Items

Hardware:
- (2) XBee DigiMesh 2.4 RF modules [Link]
- XBee Breadboard adaptor [Link]
- XBee USB Explorer [Link]
- Vegetronix Moisture Sensor [Link]
- Protoboard
- 9V Battery
- 3.3V Voltage Regulator

Software:
- Python (version 2.5 or greater) [Link]
- pyserial Library (for Python) [Link]
- XBee Library (for Python) [Link]
- X-CTU [Link]
Parts Selection

The XBee DigiMesh 2.4 RF Modules are distinctly different than other XBee modules offered through Digi given the DigiMesh firmware. This firmware allows for self-healing ad hoc mesh networking with very little configuration by the user. In a DigiMesh network all nodes can talk to all other nodes in their transmission range, there is no user defined path for information to flow. The benefit to this is that the nodes themselves choose the best path between them and the destination for their data and if any node leaves the network for any reason data will automatically be rerouted using a different path to reach the end destination.

These modules also have the series 1 hardware which can take up to 3.3V in their analog in pins where series 2 hardware can only take up to 1.2V, this is useful as many sensors have an output voltage greater than 1.2V so these XBees can be used in conjunction with many more parts such as the Vegetronix sensor we will be using.

The last key benefit to these parts is their sleep self-synchronization. What makes XBee modules in general so low power is their ability to sleep often in a low power state where they consume <50 μA. When the network self synchronizes its sleep then the XBees can spend more time asleep thus conserving more power. The synchronization works by a completely automatic process where the XBees in a network will nominate a “sleep coordinator” that will send out a sleep synchronization packet to everyone in the network during each wake cycle which overwrites whatever sleep settings the XBee might have previously had. This packet contains the information on when to sleep and wake. By doing this it avoids a lot of using error in trying to manually synchronize sleep which is very important because if a node gets too far off of the sleep cycle then it can effect some node data’s’ ability to make it back to the coordinator.

The Breadboard adaptor is necessary since XBee parts do not have protoboard/breadboard spacing as well as the USB explorer to interface the XBees with a computer through a serial port since that can be difficult to wire manually.

X-CTU is a free tool provided by Digi International that allows easy configuration of XBee parts by use of a GUI. The GUI allows for firmware updates to XBees, range testing and parameter changes via a terminal or a tree break out of the individual parameters.

The two additional libraries we will be adding to Python, pyserial and Xbee, allow us to utilize a computer's serial port and recognize XBee data packets respectively.
XBee Configuration

The first step to configuring XBee is to install the X-CTU software and connect an XBee unit to a computer by use of the XBee USB Explorer. You may or may not need to manually install the FDI Driver that allows your USB Explorer to interact with your computer; it should install automatically when you first plug it in but if it does not the driver can be downloaded off Digi International’s website.

When you first start the X-CTU software it will look like:

Here X-CTU shows what serial ports it recognizes, as well as some other settings. The settings you see here in the image are the defaults and we will not need to change them to accomplish the task at hand. You should click the Test/Query button to check that X-CTU can connect with the XBee and recognize it. If it is successful you will see this pop up:
Here you will see the firmware version currently installed on the device as well as the device's serial number. If this is the part you plan on making your coordinator write down the serial number as you will need it later. Note that the serial numbers of the parts are also printed clearly on the backside of the part.

The configuration you’ll need for the end device is:

```
Here you type “+++” and await the XBee’s response of “OK” to enter command mode. During command mode you can give commands to the XBee to either change parameters or to ask the XBee what values it currently has set for a parameter. If you do not give the XBee any commands for longer than 10 seconds the XBee will automatically exit the command mode. If you type any commands that the XBee responds with “ERROR” instead of “OK” then you have either given it an invalid command or it did not process it correctly.

For a collection of XBees to belong to the same network they must have the same PAN ID and Channel setting, here in this tutorial we give PAN ID (ID) being 8 and the channel (CH) being B. Of the 2.4GHz these XBees operate at the bandwidth spectrum is split into 16 different channels,
which channel you use is what you are setting with the channel number, on other XBee units the XBees will scan all the channels to pick the one that is most suitable for use whereas these you choose for the XBee.

All units must also have compatible sleep modes. The possible sleeping modes for DigiMesh are to be in the default “Normal Mode” where the XBee will not sleep and will not generate sleep sync messages but will rely them if sent from someone else, “Cyclic Sleep Mode” where the XBee will sleep cyclically as determined by a sleep coordinator or if there is no sleep coordinator as determined by their internal sleep settings (set by the user) and lastly “Sleep Support Mode” where they do not sleep but will generate sleep sync messages. In our example we will set a sleeping network where the coordinator is also the sleep coordinator that will instruct the other remote devices to sleep cyclically so they are awake for 30 seconds and asleep for 30 seconds.

On the remote device we also set its destination address to be the address we wrote down earlier as it is the serial address of our desired coordinator. Each XBee has a unique 64-bit address so the first 8 characters of the address are the DH (destination high) and the last 8 are the DL (destination low). An enable an analog to digital conversion register as they are disabled by default. We will also use the IR command to set a sampling rate which is set in milliseconds.

If you so desired to have more than one remote node in your network you need only have an additional XBee DigiMesh 2.4 RF Module unit and configure it in the exact same way.

For the coordinator you’ll need to configure it as in the image below:
The coordinator node requires less configuration by the user since it does not have to collect data samples and send them, it merely has to be ready to receive everyone else’s information. In order to achieve this we will make sure it has the same PAN ID and Channel that we set on the remote node. We will also set this node to sleep in the “Sleep Support Node” so although it will never sleep itself it does send sleep synchronization messages out so the remote nodes will follow the wake 30 seconds, sleep 30 seconds cycle we have set here on the coordinator.

On the coordinator we also set the command ATAP1 which puts the XBee in API mode. XBees by default operate in a “Transparent Mode” where they blindly send out any data they receive. This is the mode you would want to use if you were using XBee units solely for their networking capabilities such as if you were using them in conjunction with an arduino to send data from an arduino to a computer or other arduino. API mode is meant for when you are using XBees for their ability to take analog samples or detect analog changes as well. API mode allows the coordinator to see incoming data in packet form with sampled data information and source addresses so it is clear which data came from where/which node.

**Hardware Setup**
The hardware setup will include powering the XBee and the moisture sensor we are going to use. Below is a pin-out of the XBee model we are using with the needed pins highlighted:

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>-</td>
<td>Power supply</td>
</tr>
<tr>
<td>2</td>
<td>DOUT</td>
<td>Output</td>
<td>UART Data Out</td>
</tr>
<tr>
<td>3</td>
<td>DIN / CONFIG</td>
<td>Input</td>
<td>UART Data In</td>
</tr>
<tr>
<td>4</td>
<td>DO8*</td>
<td>Output</td>
<td>Digital Output 8</td>
</tr>
<tr>
<td>5</td>
<td>RESET</td>
<td>Input</td>
<td>Module Reset (reset pulse must be at least 200 ns)</td>
</tr>
<tr>
<td>6</td>
<td>PWM0 / RSSI</td>
<td>Output</td>
<td>PWM Output 0 / RX Signal Strength Indicator</td>
</tr>
<tr>
<td>7</td>
<td>PWM1</td>
<td>Output</td>
<td>PWM Output 1</td>
</tr>
<tr>
<td>8</td>
<td>[reserved]</td>
<td>-</td>
<td>Do not connect</td>
</tr>
<tr>
<td>9</td>
<td>DTR / SLEEP, RO / D18</td>
<td>Input</td>
<td>Pin Sleep Control Line or Digital Input 8</td>
</tr>
<tr>
<td>10</td>
<td>GND</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>11</td>
<td>AD4 / DI04</td>
<td>Either</td>
<td>Analog Input 4 or Digital I/O 4</td>
</tr>
<tr>
<td>12</td>
<td>CTS / DI07</td>
<td>Either</td>
<td>Clear-to-Send Flow Control or Digital I/O 7</td>
</tr>
<tr>
<td>13</td>
<td>ON / SLEEP</td>
<td>Output</td>
<td>Module Status Indicator</td>
</tr>
<tr>
<td>14</td>
<td>VREF</td>
<td>Input</td>
<td>Voltage Reference for A/D Inputs</td>
</tr>
<tr>
<td>15</td>
<td>Associated / AD5 / DI05</td>
<td>Either</td>
<td>Associated Indicator, Analog Input 5 or Digital I/O 5</td>
</tr>
<tr>
<td>16</td>
<td>RTS / AD6 / DI06</td>
<td>Either</td>
<td>Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6</td>
</tr>
<tr>
<td>17</td>
<td>AD3 / DI03</td>
<td>Either</td>
<td>Analog Input 3 or Digital I/O 3</td>
</tr>
<tr>
<td>18</td>
<td>AD2 / DI02</td>
<td>Either</td>
<td>Analog Input 2 or Digital I/O 2</td>
</tr>
<tr>
<td>19</td>
<td>AD1 / DI01</td>
<td>Either</td>
<td>Analog Input 1 or Digital I/O 1</td>
</tr>
<tr>
<td>20</td>
<td>AD0 / DI00</td>
<td>Either</td>
<td>Analog Input 0 or Digital I/O 0</td>
</tr>
</tbody>
</table>

On a protoboard you’ll supply the power rail using a 9v battery that runs through a 3.3V voltage regulator. The 3.3V then in the actual power rail itself is good for powering the XBee and the moisture sensor. The final setup will look like:

In this schematic the moisture sensor is modeled using a potentiometer.

Software Setup (Python Scripting)
The needed software for this endeavor other than X-CTU would be Python (version 2.5 or later). For Python you will need to add the pyserial and XBee libraries in order to interact with your serial ports and XBee respectively. The most minimal code you could use to grab data packets and display them in the Python shell would be:

```python
import serial
from xbee import ZigBee

serial_port = serial.Serial('COM3', 9600)
zbo = ZigBee(serial_port)

while True:
    try:
        print(zbo.wait_read_frame())
    except KeyboardInterrupt:
        break

serial_port.close()
```

Here you will note in line 14 we “import xbee from ZigBee” and not from XBee as is mentioned in many other documentations because of the specific XBee module we are using which is compatible with the Zigbee module and not the XBee one. In line 16 we also use the COM3 port which is the port our computer recognized the XBee as being attached to if you look back to when we introduced the X-CTU software. We also did not change the baud rate which is why it still reads 9600 in line 16 as well.

This code will print out the whole raw packet and look like the image below:

```
{'source_addr_long': 0x0012345600000000, 'source_addr': 0x00000000, 'id': 0, 'tx_cnt_long': 0, 'ts_local': 0, 'ts_global': 0, 'reserved': 0, 'data': {'adc-0': 1234, 'adc-1': 5678, 'adc-2': 9012, 'adc-3': 3456}, 'sample_type': 'none', 'sample_num': 0, 'res': 0}
```

You will notice in these data packets that they have the information “samples” and “source_addr_long” which are the two pieces of information we’d really like to have. The data in the “samples” category is in the format {'adc-0': [a number value between 0-1023]}, the adc-0 means the data was taken from the ADC register 0 which is what we configured earlier, the data being on a scale of 0 to 1023 is because the XBee has a 10-bit register so if we want our analog values we’ll have to convert these numbers from 0-1023 to 0-3 (our sensor gives readings from 0 to 3 volts). It would be nice to know when the sample was taken so we will add a date/time stamp to it as well as storing the data in a file so it is...
saved somewhere for later use or reference. We will expand out code to
the following in order to grab the address information, a converted analog
sample, a date/time stamp and store all this data to a .txt file:

```python
import serial
from xbee import ZigBee
import time, sys, datetime

#Open port and read from it
serial_port = serial.Serial('COM3', 9600)
zb = ZigBee(serial_port)

#Continuously read from port
while True:
    try:
        data = zb.wait_read_frame()  #Grab data
        addr = data['source_addr_long']

        #Stores just the voltage reading converted to 0-3.3V
        file = open('/home/cj/Documents/data.txt', 'a')
        value = float(data['samples'][0][data['adc-0']])
        num = (value*3.0)/1023.0
        file.write(datetime.datetime.now().strftime('%Y-%m-%d-%H-%M-%S')+ ' '+str(addr)+ ' '+str(num)+'
')
    except KeyboardInterrupt:
        break

serial_port.close()
```

Results

Upon completion of this project you should be able to form a network consisting
of at minimum one coordinator node and one remote node where the remote node
will send moisture data wirelessly to the coordinator node that saves the results in
a .txt file to be analyzed and or viewed by the user.

Recommendations

It is better for XBee’s range and transmission quality to have the antenna, if you
are using the module with a wire antenna, straight up since this allows more of the
wire antennas radiation field to be exposed. If you are going to enclose the XBee
circuit in an enclosure you should not choose a metal enclosure or one made of
any such material that would absorb or block the RF transmission. RF
transmission is also very sensitive to “line-of-sight” issues. This means in there is
a large metal object or any object of a decent size that will block or absorb RF
waves in the path of desired transmission the transmission will be compromised
or may not reach its destination at all. This issue is something to consider when
choosing where to place your XBee units.
References

1. XBee DigiMesh 2.4 RF Module Datasheet [Link]
2. Using XBee Radios for Wireless Acceleration Measurements [Link]
3. Tweet-A-Watt [Link]
4. Getting Started with XBee RF Modules, A Tutorial for BASIC Stamp and Propeller Microcontrollers [Link]
5. Difference between Zigbee and Digimesh [Link]
6. Basic XBee ZB ZigBee (Series 2) Chat [Link]
7. Why don’t my XBee’s talk to each other? [Link]
8. Python - Add Date Stamp to Text File [Link]