Global Journal of Engineering Science and Research Management

ZIGBEE WIRELESS MONITORING TECHNOLOGY USING XBEE INTEGRATED CHIPS

Aruna Rai Vadde[*1] Satyanarayana Gaddada[2]
[1,2]Department of Electrical and Computer Engineering, College of Engineering and Technology, Wollega University, Ethiopia.
*Correspondence Author: **Aruna Rai Vadde**

Keywords: Zigbee systems, XBee chips, Xbee-API, Xbee-Arduino.

Abstract

The physical layer convention standard utilized as a part of Zigbee frameworks is IEEE 802.15.4.the remote standard that works, in North America, in the scope of 2400-2483.5 MHz or 902-928 Mhz. Zigbee, and above all our chips, work in the higher 2.4 GHz range, the information transmission rate is dependent upon 250 kilobits every second. The part my exploration that we actualized fundamentally manages the system layer. This research paper Xbee chips and firmware permit us to "black box" the Data Link and Physical layers, and the open source Xbee-API and Xbee-Arduino programming bundles enormously rearranged our work in the system layer.

Introduction

The outlined a framework for remotely controlling transfers and checking current. This is utilized for a home load recreation. By remotely turning transfers on and off by sending charges from a PC to a microcontroller we can change the aggregate burden (present) to our recreated home. For remote correspondence, we utilized Xbee Series and Zigbee RF modules. One of these modules was joined with a microcontroller and the home load reproduction, while an alternate was associated with the PC, which was utilized for gathering and showing information and for transfer checking and control.

Background theory

The transmission framework focused around the Zigbee steering and systems direction convention [1]. This convention and its points of interest are talked about in more prominent detail in the Standards segment (underneath); in this area, we concentrate on fundamental system hypothesis and the part this hypothesis played in our venture. The Data systems (and transmission frameworks) are regularly partitioned into different layers focused around usefulness [2]. This is some of the time called a "convention stack" (for our situation, we are utilizing a "Zigbee stack"). Basically, the bring down the layer, the closer we are to stressing over real physical electrons flying around [3]. On the other hand, the higher the layer, the less we are stressing over physical demands and information structures are that we are managing and controlling.

Physical layer

The physical layer's job is to move individual digital bits from one place to another. The protocols in this layer depend on the actual physical medium [5]. For example, in a wireless system, the actual physical medium is simply the atmosphere.

Link layer

A network's link layer routes a series of bits (sometimes called a datagram) from one node in a network to another. This can happen through a series of intermediate switches (or routers). Protocols at this layer provide more robust and full-featured services than protocols at the physical layer. Wi-Fi is one example of a link-layer protocol [6].

Network and transport layers

Since these layers are higher in the model, conventions at this layer normally are all the more full-emphasized than conventions at the connection or physical layer. These Protocols at these layers utilize the connection layer's steering abilities to move the previously stated datagram's between hubs in a system [7]. The Internet Protocol (IP) is likely the most celebrated system layer convention, while the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) are two cases of well-known and broadly utilized transport-layer conventions [8]. Certain more elevated amount usefulness is more predominant in these two layers than in lower levels. Case in point, stream control – controlling the transmission rate between hubs with a specific end goal to lower blockage on the system (understanding that even simply a two-hub transmission framework can be viewed as a "system") – and solid transmission (guaranteeing that a parcel is really gotten) are two peculiarities regularly actualized in the system and transport layers.

Justification and sources of idea

This project was proposed by a research/project team here at Cornell working on a so-called “Smart Home Energy Monitoring System.”

Global Journal of Engineering Science and Research Management

What this group is working on is an energy-monitoring system for a home using alternative energy sources. A very high-level block diagram of this system is shown below:

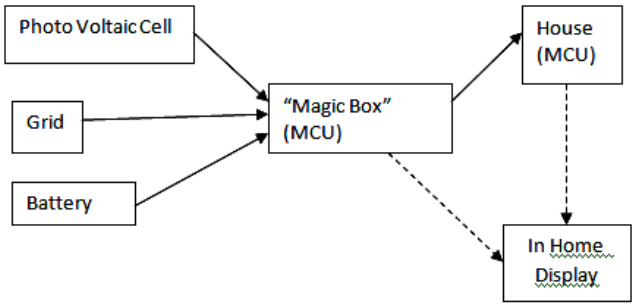

Figure 1. High-level diagram of the "Smart Home Energy Monitoring System."

In the above diagram, the "Magic Box" essentially exists to route energy between the house, the various energy sources (for example, a solar cell or the electric grid), and an energy storage unit (such as a rechargeable battery). In this project, we will "black box" this Magic-Box system, and instead focus on another portion of this energy monitoring system, which involves sending data regarding energy usage and various energy loads (by wirelessly controlled relays) to a PC in order to display and monitor energy usage within the home.

Our Research focuses on the House to In-Home Display (IHD) part of this system.

- ✓ What our project does is the following: Simulates a home by using resistors to model various appliances or groups of appliances
- ✓ Allows a user to wirelessly turn on and off various sets of appliances by controlling relays Monitors total power consumption and current in this home Wirelessly transmits the above data to a PC, displaying the data on a graph that shows real-time power consumption in the home.
- ✓ Our simulated "home" consists of 7 appliances, each modeled by a resistance (a load).

These 7 loads are all in parallel, and are fed voltage from a 5 V DC source (the microcontroller). In a real house, the source voltage would be 120 V AC (in the United States at least). We considered attempting to implement this, but considering the practical dangers of "playing around" with 120 V AC circuits as well as the fact that a relay control system shouldn't depend on the voltages being fed through the relays led us to decide to stick with 5 V DC circuits for testing and simulation.

Hardware & Software

The Hardware and software an Arduino board instead of the STK500 which we were used to. This required less software for us to write. It seemed much less tedious to write certain tasks in Arduino such as turning on an LED. We were interested in expanding our knowledge of other hardware and software that was similar but not exactly the same as what we had learned throughout the semester. Other than that we did not have to do any other sort of tradeoff between hardware and software.

Standards

The most relevant standard for our project is the Zigbee wireless networking standard, which is IEEE 802.15.4. The reason we chose Zigbee over Wi-Fi (802.11) or another RF standard was twofold. First, it consumes a very low amount of power, which could be very useful on the MCU-end of our transmission system (see Tentative Design section). And second, it is (apparently) of much lower complexity than Wi-Fi, making it easier to implement. It has a lower data rate than Wi-Fi (only up to 250 Kbits/second) but is still easily capable of transmitting the relatively low amounts of data that are necessary in this domain. The main feature of this standard is the necessity of achieving technological simplicity, low operation cost, and low manufacturing cost without sacrificing flexibility or generality.

Software Details

The software portion of this project consists of two main applications: one for an Arduino-based microcontroller unit, and another for the PC. Arduino, as discussed in the hardware section, is an "open-source electronics prototyping platform" based on Atmel microcontrollers (in our case, an ATMega328) and its own Arduino programming language. This language is very similar to C/C++, and the programming paradigm is almost exactly the same as it is when writing in C for AVR-GCC and Atmel MCU's.

Global Journal of Engineering Science and Research Management

On the PC side, we used Java to interact with an XBee chip connected through either a serial or a USB port (in other words, the XBee is recognized as a serial device on the PC).

Evolution of Software Design

The reason for choosing a Java-Arduino based system is simple: we found a pair of very well-documented open source (GPLv3) projects implementing an API for data transmission using XBee chips and the Zigbee protocol. These projects are:

- ✓ XBee-API: This is for the PC side. It is a Java software library with the goal of providing "a flexible and simple to use API to interact with XBee radios.
- ✓ XBee-Arduino: This is very similar to the XBee-API package (and was written by the same person), but is for Arduino-based microcontroller platforms rather than PCs.

Software Design, Microcontroller Side

On the MCU side, we essentially just have a main loop (note, however, that in Arduino, rather than writing an infinite loop in main, you simply write a function called loop() which does the same thing) that checks to see if a new XBee packet has arrived using the XBee-Arduino API.

If a new packet has arrived, we need to do the following:

- ✓ First, make sure that the packet is a Zigbee packet carrying a payload. If this is the case, proceed.
- ✓ Send an acknowledgement to the sender, letting the code on the PC know that we've received the packet.
- ✓ Wait to see if the sender gets the acknowledgement. If debugging, print whether or not the sender got our acknowledgement.
- ✓ Now, using the String library on Arduino, construct a new string representing the payload that was sent from the PC.
- ✓ Parse this payload, looking for the commands RON and ROFF. Our command format is RON01 for turning pin 1 on, ROFF07 for turning pin 7 off, etc. Each relay is hooked up to a pin, so you can turn on and off relays in this manner. You can send multiple commands per packet, our payload parser simply looks for each of those commands somewhere in the string, and the payload must begin with the String "CMD ".
- ✓ If RON or ROFF commands exist and are valid, simply turn on or off the pins that are signaled. If both RON and ROFF commands for the same pin are in the same payload, turn the pin (relay) off.

Software Design, PC Side

All of the PC-side code was written in Java using Eclipse. The lowest-level overview of our code and software design as it is generated directly from comments written in the source code. However, understanding a piece of software simply by looking at the Javadoc documentation can be difficult, so here we've provided a higher-level overview of our entire software design.

The design of our underlying software (not including the GUI) can be summed up in the following equation:
Xbee Manager + Relay Manager => Xbee Relay Manager => Xbee SWTGui

Xbee configuration

A Brief Primer: The physical aspects of the Xbee chips are introduced in the hardware section (below). Here, we will discuss the software aspects of the configuration.
There are two different modes of operations that Xbees can run in. They are:

- ✓ Transparent mode
- ✓ API mode

In transparent mode, the Xbees essentially act as a dumb serial line replacement. In other words, each Xbee simply forwards everything that comes in through its antenna to its Din port, and sends everything that comes in through its Dout port through its antenna and into the air. This is functionally equivalent to simply having a serial line connected between the two Xbee chips.

In API mode, Xbee operation is not nearly as simple. API stands for Application Programming Interface, and in this mode you can send commands to and from the Xbees to perform various tasks. Further, it is in this mode that the Zigbee protocol is used for Xbee Series 2 chips. In other words, whereas transparent mode acts as a dumb serial line replacement, API mode unleashes the full power of the Xbees and allows us to construct a real wireless network consisting of our two devices.

When in API mode, there are three more modes that each Xbee chip – or any chip on a Zigbee network – can operate in. These modes are:

- ✓ Coordinator mode

Global Journal of Engineering Science and Research Management

- ✓ Router mode
- ✓ End Device mode

These modes are essentially listed in order of decreasing functionality. Coordinators are the most powerful nodes in Zigbee networks, and can coordinate activity within one network while also interacting with the outside world. Other nodes in the network "report to" the coordinator. Router nodes have the capability of forwarding packets from one node to another (acting as an intermediary in the network, or a router). End device nodes cannot act as routers but can communicate with routers or the network coordinator. On series 2 Xbee chips (which we used), it is necessary to change the firmware on the chips to change between API and transparent mode and also between Coordinator, Router, and End Device mode.

In our application, we used the Xbee connected to the PC as a coordinator, and the remote Xbee as an end device. This allowed us to use the Xbee connected to the PC to send commands to a remote Xbee, and also to set up automatic I/O sampling on the remote Xbee. Xbees that are within the network of a certain coordinator are said to be "associated" with that coordinator node.

To configure Xbees, you can use Digi's own X-CTU software, which was created specifically for configuring Xbee and similar chips offered by Digi, or you can simply connect to the Xbee chips via a serial terminal (such as Hyperterminal) and send them a series of commands. You can also use a network coordinator to send commands to be executed on a remote Xbee within that coordinators network. The Xbee chips have various registers that store different parameters. For example, the parameter AP is what determines whether the Xbee is in transparent or API mode. When AP=0, transparent mode is enabled. When AP=2, API mode is enabled.

For changing Xbee firmware, X-CTU must be used (for all intents and purposes, it would be possible to change it without X-CTU, but this wouldn't be practical for most users with a limited amount of time).

Our Xbee configuration

The configuration parameters that we changed on our two Xbees are described below.

PC Side: The Coordinator

For this Xbee, we began by flashing the firmware so that we were operating in API mode and as a Coordinator. We did this using X-CTU, as described above.
Next, we ran the following series of commands in order to configure the coordinator:

```
# Reset default settings
ATRE
# set to API mode
ATAP 2
# Set the personal area network (PAN) ID
# Note that the end device needs to have this
# Same PAN ID in order to associate itself
# with this coordinator
ATID 1AAA
# Set this node's string identifier
ATNI PC_COORD
# Write our configuration so it's not lost on reboot
ATWR
# Restart
ATFR
```

MCU Side: The End Device

For this Xbee, we began by flashing the firmware so that we were operating in API mode and as an End Device. Again, we did this using X-CTU, as described above.
Next, we ran the following series of commands in order to configure the end device.

```
# Reset default settings
ATRE
# Set to API mode
ATAP 2
```

Global Journal of Engineering Science and Research Management

```
  # Set PAN ID
  ATID 1AAA
  # Set node's string identifier
  ATNI REMOTE_XBEE
  # Now, configure the Xbee to send IO samples to its coordinator
# every 0xbb8 (roughly 3000) milliseconds
ATIR 0bb8
# Now set D2,D3,D4,D5,D6,D11,D12 to digital inputs
# Each of these is used to monitor one relay
ATD2 3
ATD3 3
ATD4 3
ATD5 3
ATD6 3
ATP1 3 # P1 is D11
ATP2 3 # P2 is D12
# Now set D0 to analog input, using the Xbee's built-in 10-bit
# A/D converter, which reads 1.2 V as 1023 and 0 V as 0.
ATD0 2
# Write config
ATWR
# Restart
ATFR
```

One more note regarding the Xbee's on-board Analog-to-Digital converter: there are 4 pins that can be used for analog input right on the Xbee chips, and each of these has a 10-bit A/D converter that saturates at 1.2 V. Thus, in order to get an accurate reading, you shouldn't attempt to read values much greater than 1.2 V. As described in the hardware section, we set our current-measuring resistor to a value such that when all of the loads (relays) were turned on, we get voltage readings just below 1.2 volts, and thus around 1020 on the 10-bit A/D conversion.

Hardware details

Our system consisted of an Arduino board with an Atmel Mega328 microcontroller, two Series 2 Xbee chips, and seven relays.

Arduino Board

The reason why we chose to use Arduino was because it is an open source software package that can be used with our Xbee chips. It has an Atmel Mega328 MCU so we were able to do all the functions (ISR, configuring ports, etc.) that we learned throughout the four labs beforehand. Since we were familiar with a Mega644 MCU and not a Mega328 MCU we became very proficient at reading datasheets and figuring things out on our own. The Arduino board was also useful in that it had an usb connection to that made it much more convenient to connect to our laptop.

Global Journal of Engineering Science and Research Management

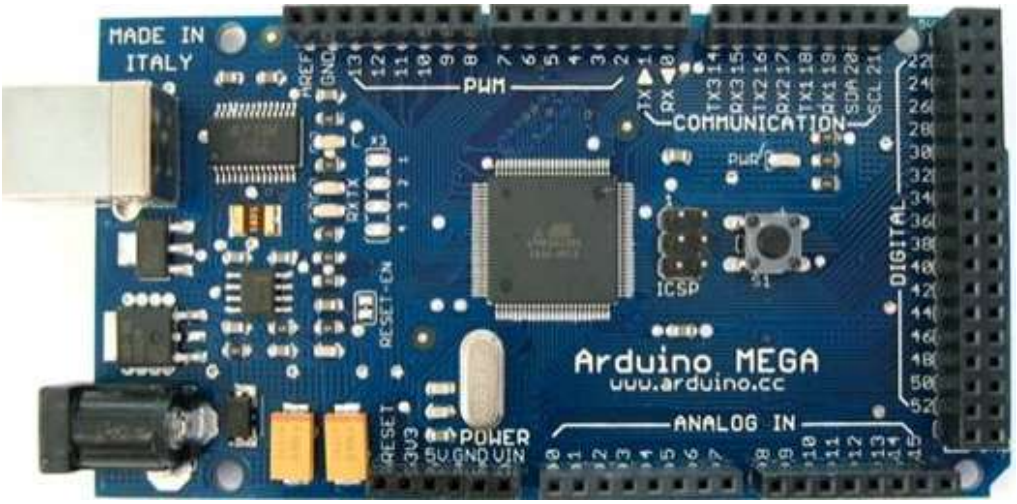

Figure 2: Arduino Board

Xbee Chip

We used two Series 2 Xbee chips, more specifically 802.15.2 Series 2 Chip Antenna Xbee chips. It has an on-chip antenna that has a range of 300 feet indoors. It is also powered by 3.3 Volts. We had originally wanted the much simpler Series 1 chips. Series 1 Xbee chip is based on the IEEE 802.15.4 WPAN specification. It supports point-to-point, and point-to-multipoint communication. It also has 6 analog and 8 digital I/O pins, with 6 pins shared between digital and analog. It is much simpler and does not require a firmware change to switch between Coordinator and End Device, or AT mode. Series 2 is ZigBee compliant. ZigBee introduces mesh networking, where networks can be extended beyond typical wireless range limits through the use of routers. This radio supports 4 analog and 11 digital I/O pins, with 4 pins shared between digital and analog. Some pros of the Series 2 chips are that it consumes less power and we can extend the range of the network beyond the limit of a single transmission by adding routers. Some cons are analog inputs can only measure voltages between 0 and 1.2, requiring a voltage divider to step down from the reference voltage of 3.3V. Another con is that it requires separate firmware for API and transparent (AT) mode. You can't switch between API and transparent mode with the same firmware as with Series 1, instead you have to choose between installing the API firmware, or transparent firmware. Additionally, the firmware is specific for Coordinator, and Router/End Devices. Since we got the Series 2 chip we needed to configure one chip to be a coordinator and another to be an end device. We became familiar on doing this by the X-CTU program. This required some researching and trial and error but we were able to configure each chip.

Figure 3: Antenna Xbee chips

Since each needs 3.3 volts to power it we got a regulated Xbee Explorer board and a regulated USB explorer board. This way we could connect one of the Xbee chips to the PC via the USB explorer board and the other to the Arduino board via the explorer regulated board. The boards needed 5 volts to power it up and were pluggable into our breadboard after soldering on some pin headers. The connection of the Xbee chip to the Arduino board to send and receive packets are as follows: 5V from Xbee chip to 5V on Arduino Board, Gnd from Xbee chip to Gnd on Arduino Board, Dout on Xbee chip to Rx on Arduino board, and finally Din on Xbee chip to Tx on Xbee chip.

Relays

Global Journal of Engineering Science and Research Management

Figure 4: Relays chips

We used 7 mechanical relays and connected them to resistors to serve as our appliances for our simulated home. The way the relays were connected goes as follows: there are 2 control pins. The first one goes to the MCU pin and the second one goes to the MCU gnd. Also there are 2 other pins on the relay and the first of these two goes to the resistor that serves as an appliance and the second goes to VCC (5V). We also attached an LED connected in series with a 330 ohm resistor to the control pins of the relay that go to the MCU. The reason for this was so that we know when we turn on specific relays from the PC whether they actually turn on. At first we used this for debugging purposes but then we decided that it would be better to leave them since they are a good visual aid for people. The resistor value that we have for each appliance, the pins on the arduino board and Xbee chip, as well as the relay that controls it is as follows:

Table: 1

Relay	Resistor Value (ohms)	Appliance	Arduino Board pins	Xbee pins
1	100,000	Base	2	D2
2	68,000	Lights	3	D3
3	36,000	Fridge	4	D4
4	18,000	Water Heater	5	D5
5	14,000	Dishwasher	6	D6
6	3600	Dryer	7	D11
7	3000	Oven	8	D12

We had all the above appliances in parallel and connected them to each other. So in order to measure the current that is going through when various appliances were turned on we needed to add a resistor in series with it and this was sent to pin A0 of the Xbee chip. Also as mentioned above the Series 2 Xbee chip only measures a voltage from 0 to 1.2V so our calculations needed to address that. So we had to find the max current that we would get and that would happen when all appliances were on and the value of the resistance would be 1283 ohms (Vcc= 4.82 V). So we set the equation as follows:

$$4.82/1283+R = Imax$$

$$R = 1.2/I\ max$$

Solving these equations simultaneously you get:

$$I = 2.82mA$$

$$R = 425\ ohms$$

So we needed something a little below 425 ohms and we used 408 ohms.

Global Journal of Engineering Science and Research Management

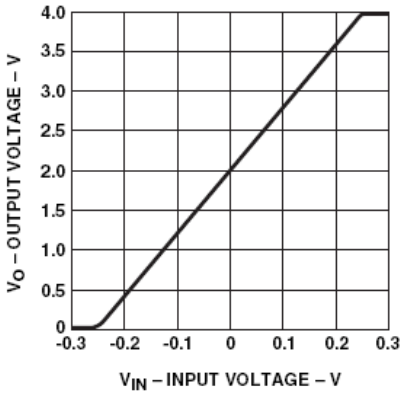

Figure 5. Theoretical transfer curve for the HCPL-7520 optoisolator.

Instead of this our Vout resembled this:

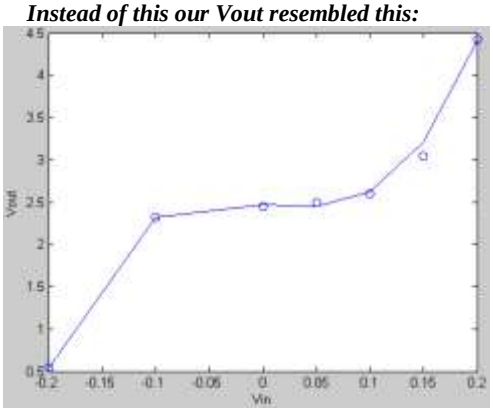

Figure 6. Observed transfer curve for our HCPL-7520 opt isolator.

We used a best fit line with degree=3 and the equation we got

$$Y=272.5018*x^2+ .0513*x^3 – 1.2151*x +2.4733.$$

Nonetheless we decided to go ahead with the implementation with our graph and the points, however when we turned different loads on and off our Vout never changed. We feel that we might have burnt out the chip or that we got a non-functioning chip.

We didn't think that we spent all the hours trying to figure out why the chip wasn't working properly for nothing. We learned a whole lot on how to properly read data sheets, about opto-isolators, and got good practice with designing different circuits with voltage dividers to attain the voltages we wanted. It wasn't necessary for us to have implemented the opto-isolator.

Results of the design

Global Journal of Engineering Science and Research Management

Speed of implementation

Our programs are very responsive. Packets usually arrive with 50 to 70 milliseconds latency. We get I/O samples from the Xbee chips every three seconds and then update the graph on the GUI.

Precision

We tested our relays extensively. When we send the packets to turn on the relays the relays turn on. We tested this by turning a few to all the relays at once and turning them off. We know this because of the LED attached to them. The A/D converter on the microcontroller is also very accurate and stable and gives us the same reading when we turn on the specific appliances.

Safety in the design

We had originally tried to use an opto-isolater to be able to take in a 120 V AC line. However since that wasn't necessary since we are only using 5V it wasn't a problem not to use it. Our design is safe by all standards.

Interference design

Our system does not interfere with other people's designs because we use physical addressing so the packets are only sent to specific Xbee chips that we specified and not all Xbee chips.

Usability

Our system is usable by everyone that is able to use a computer. Our GUI makes it simple to turn on and off appliances. Also our graph is very simple to read and is updated every 3 seconds so it makes it easy for people to see the power being used.

Conclusion

The Overall, our results definitely met our expectations. We set out to construct a wireless transmission system for an in-home display of energy usage and ended up with an entire wireless relay control and power monitoring system with a home-load simulation. Further, whereas we originally proposed using the Zigbee protocol, we thought this would be far too difficult after further reading on the protocol and on Zigbee operation with Xbee chips. Find the Xbee-API and Xbee-Arduino open source software libraries allowed us to use a real networking protocol – and the one we first proposed – rather than using the Xbee chips as a dumb serial line replacement.

References

1. *"ZigBee Cluster Library Specification Download Request". Zigbee.org. April 2010*
2. *"ZigBee Specification ". Zigbee Alliance. June 2013.*
3. *"ZigBee Wireless Networking", Drew Gislason*
4. *ZigBee Document 053474r06, Version 1.0, ZigBee Specification. ZigBee Alliance. 2004.*
5. *"The ZigBee Alliance". Zigbee.org. October 2012.*
6. *"Wireless Sensor Networks Research Group". Sensor-networks.org. October 2012*
7. *"IEEE 802.15.4". Ieee802.org. October 2012*
8. *Bellido-Outeirino, Francisco J. (February 2012). "Building lighting automation through the integration of DALI with wireless sensor networks". IEEE Transactions on Consumer Electronics 58 (1): 47–52. doi:10.1109/TCE.2012.6170054.*