WIRELESS ENERGY MONITORING SYSTEM

By

Federico Alejandro Foglia

Approved:

Stephen D. Craven
Assistant Professor of Electrical Engineering
(Director of Thesis)

Raziq Yaqub
Associate Professor of Electrical
Engineering
(Committee Member)

Abdul Ofoli
Assistant Professor of Electrical Engineering
(Committee Member)

William H. Sutton
Dean of the College of Engineering and
Computer Science

Jerald Ainsworth
Dean of the Graduate School

WIRELESS ENERGY MONITORING SYSTEM

By

Federico Alejandro Foglia

A Thesis
Submitted to the College of the
University of Tennessee at Chattanooga
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Engineering,
Concentration in Electrical
Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

May 2012

ABSTRACT

Monitoring, controlling, and managing the energy usage in houses or buildings are tasks that not many electric power companies can do with the exception of a few. This sort of operations may be important to households who want to monitor and possibly even control and manage their own electrical energy usage. For this thesis, an open source Wireless Energy Monitoring System was implemented, primarily based on the OpenEnergyMonitor project. The goal of this thesis is to provide a very simple, scalable, and reliable monitoring system that is capable of reading current and voltage from any electrical panel of any building. The system is based on Arduino Fio boards for reading and processing the data (current and voltage) and Xbee Series 2 radio frequency (RF) modules. The circuit that allows the voltage and current readings is based on the schematics provided by the OpenEnergyMonitor project, but with small modifications.

DEDICATION

I would like to dedicate this thesis to my parents, Luis Foglia and Olga Ghiglione, and to my siblings Andrés Foglia and Julieta Foglia. Thank you for your wholehearted support you have given to me all these years.

ACKNOWLEDGEMENTS

I would like to thank the Electrical Engineering faculty at UTC for their help, experience, and excellent work in each class that I attended. I would specially like to thank Dr. Stephen Craven for his time, help, valuable knowledge, advice, and for giving me the opportunity to study a Master's program under his supervision. You were a great mentor to me.

I would also like to thank my family for their continuous support in good and bad moments. You mean everything to me and I will always be indebted to you.

Finally, I would like to thank to all my friends for their help and support throughout these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API, Application Programming Interface

CSV, Comma Format File

CT, Current Transformer

EPB, Electric Power Board

FPGA, Field Programmable Gate Array

FTDI, Future Technology Devices International

IDE, Integrated Development Environment

PAN, Personal Area Network

POSIX, Portable Operating System Interface

PRET, Precision Timed

RF, Radio Frequency

TX, Transmission

WSN, Wireless Sensor Networks

CHAPTER I

INTRODUCTION

Households find it important to manage and control their use of electrical energy in order to lower their electric bills and help protect the environment. This objective is not always easy to achieve, unless households are given information about their electrical energy usage from their electricity providers. Some power companies provide this kind of service to their customers, but still customers might not receive all the information they might need.

In order to provide a simple and reliable way for households to be able to monitor their electrical energy use a Wireless Energy Monitoring System was implemented in this thesis based on the OpenEnergyMonitor project [1] and using a Wireless Sensor Network (WSN). The WSN is composed by Arduino Fio boards for the sensors and 2.4GHz Xbee RF modules, for the wireless communication. Each node is capable of reading the voltage and current from an electrical panel through a circuit attached to the sensor, a current transformer (CT), and a 9V AC-to-AC power adapter.

Need for the System

Owners of buildings usually need to be informed about the way their energy is being used. Nowadays, there is no easy way to control and manage the usage of the energy (e.g.: read the current and voltage, calculate the real power, apparent power, or power factor) at a building besides using the procedures or tools used by the electric power company, which sometimes are not available for the regular user. This thesis created a Wireless Energy Monitoring System that is reliable, easy to use, and easy to install. Figure 1 shows an overall idea of how the system works.

Some utility companies, such as Chattanooga's Electric Power Board (EPB), provide reliable information to users about energy usage at their homes or buildings. These are normally web-based systems or reports showing the consumption of energy each day or month, but they do not provide readings for each element, room, or place in the building [2]. These reports usually are general readings from the whole building or house, or in other words, these do not differentiate the energy usage at a certain room in a building. This issue does not apply to the system that was developed in this thesis. In using WSN, each room or floor of a building can be monitored separately, as well as the entire building.

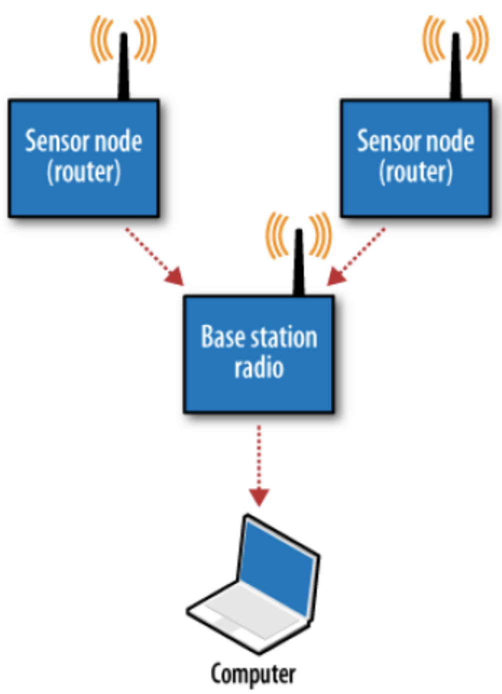

Figure 1. Simple WSN configuration [3]

CHAPTER II

BACKGROUND

An overview of the OpenEnergyMonitor project, WSN, and different types of prototyping platforms, RF modules, and methods of performing readings (voltage and current) is presented in this chapter.

OpenEnergyMonitor Project

As stated in Chapter I, this thesis is based on the OpenEnergyMonitor project [1] that was developed and still supported by a large group of engineers from all around the world. The project provides a reliable, simple, and affordable way to measure current, voltage, and power from any electrical panel of any building. The final goal of this project is to give households a way to control and manage their electrical energy use. The OpenEnergyMonitor system is based on Arduino boards and has three main components, as seen in Figure 2:

1. emonTX: It is a low power wireless energy monitoring node. It is designed to sense data from multiple CT current sensors (3 maximum), optically from a pulse-output utility meter and from multiple one-wire temperature sensors.
2. emonBase: It is a web-connected base-station that receives energy monitoring data from the emonTx and posts to a remote or local server for web-based logging and visualization.
3. emonGLCD: It is an open-source general purpose wireless graphical LCD display unit.

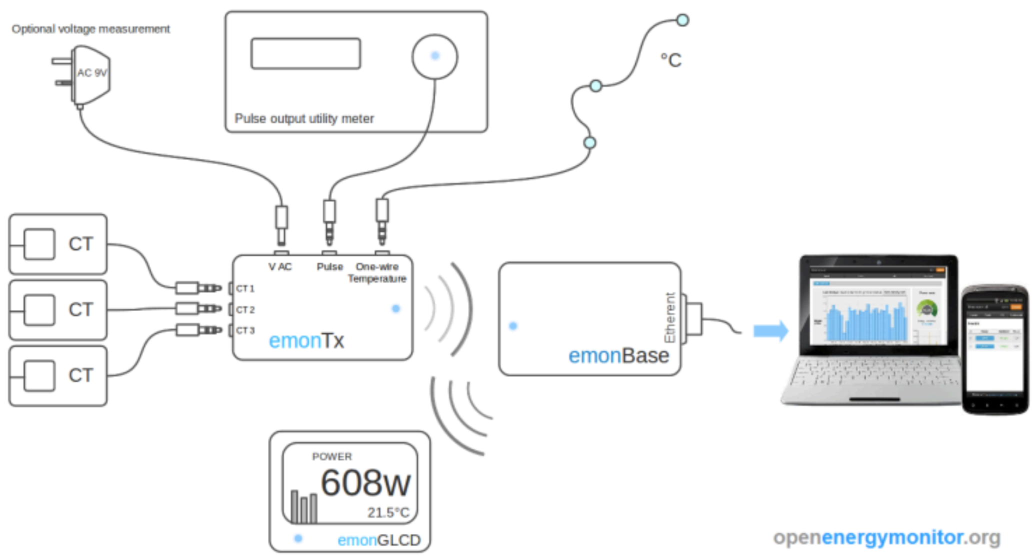

Figure 2. OpenEnergyMonitor overview [1].

This Wireless Energy Monitoring System uses similar components, like the ones used in the OpenEnergyMonitor project. However, there are substantial differences: the central node, which is the analogous of the emonBase device, sends the readings to the computer, which finally shows the information on the command line screen and saves it to a file for future analysis. This information is not available through a web system, like in the emonBase, for simplicity of the system, although this is considered for future work and explained later on in Chapter VI. Also, there is no emonGLCD device in the monitoring system, the information is only shown in the computer and no LCD display is used. Lastly, each sensor node, which is the analogous of the emonTX device, only read current and voltage from one CT and one power adaptor. The sensor nodes do not support multiple CTs, neither temperature nor utility meter readings.

Moreover, this Wireless Energy Monitoring System uses the same schematics for the current and voltage circuit interface that allows the Arduino Fio to perform the readings, but with small modifications: the Arduino boards used in the OpenEneryMonitor project operate at 5V and the Arduino Fio boards used for this thesis operate at 3.3V. Also, the chosen CT for this thesis includes a burden resistor, therefore there is no need to perform the required calculations to select one and include it in the circuit. These small modifications are explained later in Chapter IV.

Wireless Sensor Networks

WSN are composed of small distributed and autonomous sensors used to read different variables, like temperature, humidity, sound, etc., of an environment condition.

These nodes are composed of microcontrollers (with an external limited memory and transceiver to send and receive data) and sensors to monitor the different variables just mentioned. These microcontrollers usually consume very low power and run at a low speed (typically at 4 to 5 MHz). Figure 3 shows how these elements interconnect [3].

The following are the most important or known issues related to WSN. Some of these still do not have a definite solution [3]:

- Power consumption: The sensor nodes are normally powered by batteries. Power conservation and careful use of available energy is something to be taken into account.
- Time synchronization: Some applications for WSN are required to be extremely accurate regarding the reading or monitoring of data through the sensors. Synchronizing the clocks regularly means that the sensor needs to use a lot of energy to relay clock-synchronizing messages. Therefore, there is no perfect solution to this day for this issue.

- Security: The information going across the wireless network is susceptible to different active or passive attacks, such as eavesdropping or altering the readings. Wireless connectivity is less secure than wired connections since data travels everywhere through the wireless space.

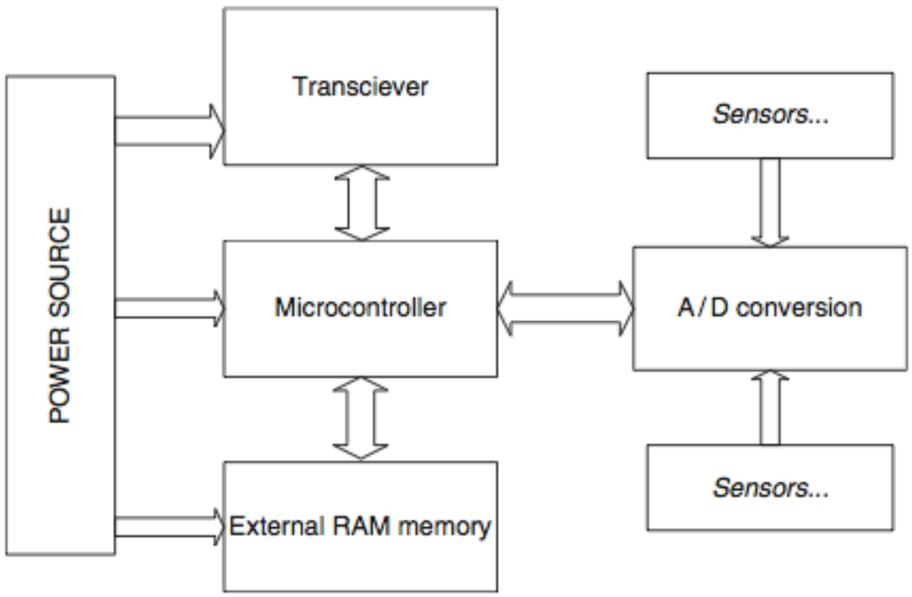

Figure 3. Diagram of a sensor node [3]

Single-hop Wireless Sensor Networks

Most of the applications in WSN are single-hop, where a single node reaches a central node that collects and processes the data that was measured by this single node. These applications are commonly used in the health care industry, where for example, sensors can be embedded into watches that are attached to patients in order to monitor their blood pressure and pulse (as shown in Figure 4). This information can be sent to a central node, which can analyze the data and send alarm messages to the patient or the doctor in case of anomalies [4].

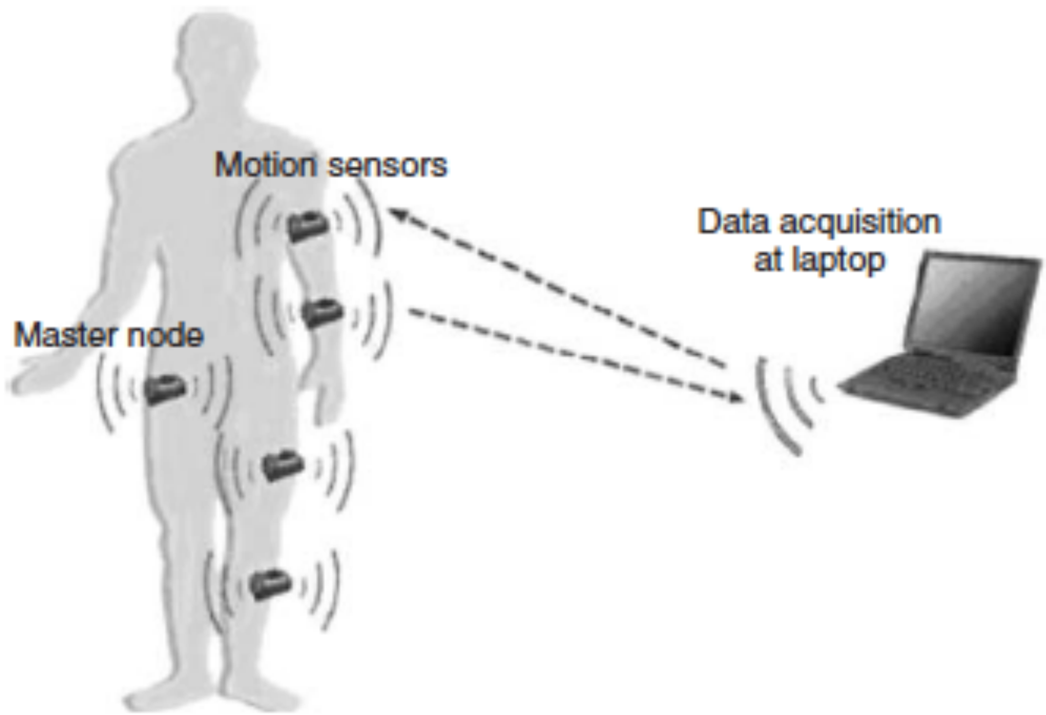

Figure 4. Single-hop sensor network [4].

Multi-hop Wireless Sensor Networks

WSNs normally operate in a multi-hop fashion (see Figure 5), especially when there is a large number of sensor devices in the network. This provides a more powerful and scalable way to "sense" the physical world [4], allowing sensors to retransmit the data that was read and received from other sensors (neighbors or not) to the central node. This data can be just retransmitted, it can be combined with other data received from other sensors, or it can be read from the sensor itself.

This thesis follows an approach where the sensor nodes transmit or route the information to the central node, resulting in a more scalable and reliable system.

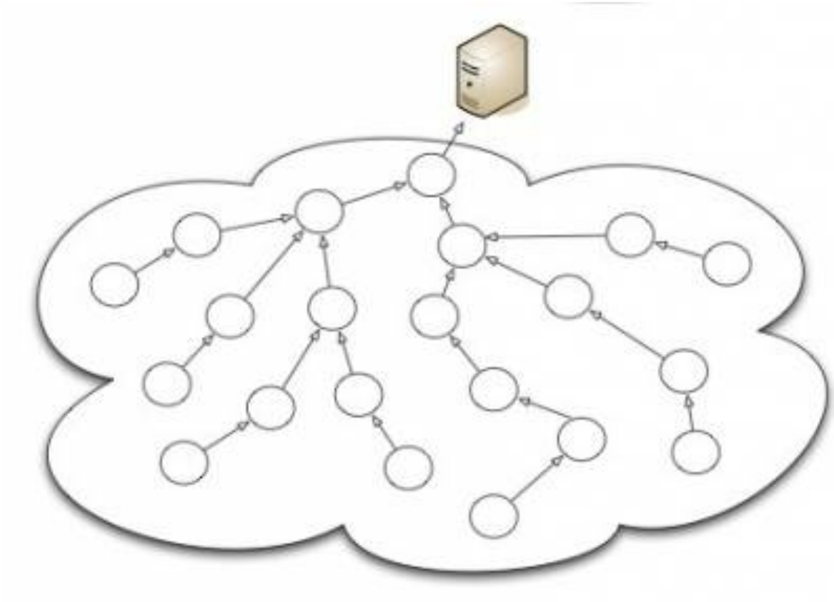

Figure 5. Multi-hop sensor network [4].

Event-driven, Periodic, and On-demand Reporting

The applications of WSN can be placed into three different groups, each having its own data communication mode:

1. Event-driven mode: Sensors report the sensing information to the sink once a specified event has been detected. For example, as shown in Figure 6, a fire detection system will report the fire station as soon as it detects fire in a room. ***Note****: A sink is any node, central node, to which the information must be reported.*
2. Periodic mode: Sensors collect sensing information from the environment at predetermined times and report it periodically to the sink or central node. This mode is used in the WSN of this thesis; each node collects the data from the electrical panels at predetermined times, performs calculations, and then reports it to the central node periodically.

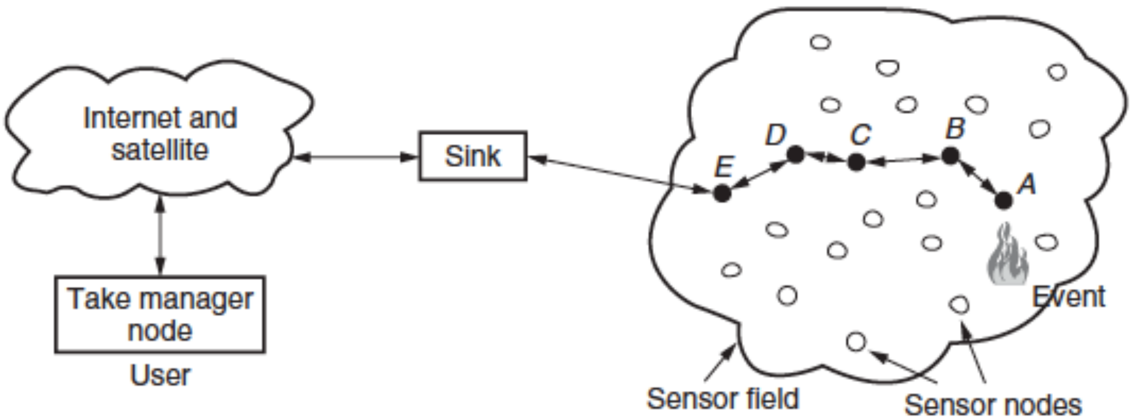

Figure 6. Event-driven mode example [4]

3. On-demand mode: Users decide when to collect the sensing information. They send commands to the WSN indicating they want the data to be reported, and then wait for it to be sent and received. Users may even specify future reporting periods, therefore switching to a periodic mode. An example of the on-demand mode could be when a user sends a report command to a WSN that is collecting the humidity information of a cornfield.

Prototyping Platforms

There are many prototyping boards options that fit well into this system that are available in the market, but the ones considered for this thesis were the Arduino platforms [5], which are provided in a list in Chapter III. These platforms were chosen upon the following main characteristics: size, energy consumption, open source, sufficient number of I/O pins, low cost, and available connectivity to radio modules (or available information on how to interface radio modules and the board). It is also important to mention that the Arduino project provides an Integrated Development Environment (IDE), which uses a C-based language.

Radio Frequency Modules

There are many options for radio modules that could be used for this system. Some of them are well-known, and a lot of information and projects are available online to understand how they work and what their capabilities are. The most often-used RF modules are the ones built by Digi [6], called Xbee. These were previously used for different projects, and provided good range, low energy consumption, reliability, and ease of use. A list of RF modules is provided in Chapter III.

Circuit Interface

A circuit provides the interface between the prototyping board and the electrical panel. It allows the board to perform the voltage and current readings from an electrical panel. It was mentioned in the introduction of Chapter I that this Wireless Energy Monitoring System is based on an open source project called OpenEnergyMonitor [1]. This project provides all the required elements that allow the system to read the voltage and the current. The circuit that is needed is the same one provided in this open source project, but with a small change (this is explained in detail in Chapter IV). It is essentially composed of two voltage dividers (one for the current readings and one for the voltage readings) and two capacitors to get rid of any AC components that might be left and that could create some noise, resulting in less accurate readings. The circuit is connected to a CT that through inductance is capable of reading the current flowing in a wire. Also, it is connected to an AC-AC power adapter that provides step-down (usually to 9VAC) the voltage from the panel in order to be read in a more safe way.

CHAPTER III

WIRELESS ENERGY MONITORING SYSTEM: ANALYSIS

This chapter explains how the monitoring system was designed and the tools or components used to complete it. In order to give a more accurate description of the system, this chapter is divided into five main sections: analysis, hardware design, software design, hardware implementation, and software implementation.

Analysis

Prerequisites

The following prerequisites were needed to successfully build a reliable, simple, and affordable Wireless Energy Monitoring System:

- The nodes in the WSN must provide an accurate reading.
- Low cost components.
- The RF modules must have enough transmission power to send/receive data between adjacent floors.
- The readings or data received need to be shown to the user and at the same time saved in a comma format file (CSV) [7] since it is widely supported by consumer, business, and scientific applications (e.g.: ETAP, Excel, etc.).
- The front-end interface is required to be easy to use.
- The overall system must be reliable, fully scalable, and easy to use.

Logical Design

The logical design provides a high level view of the components or parts of which the system is composed. This view needs to be divided into two, hardware and software components:

1. Hardware
    a. Sensor nodes: This is a microcontroller board that provides many digital and analog I/O pins in order to read or write data from or to any device connected to it. The sensor node is capable of establishing a connection to a RF module that will send the data to the central node. Also, it could be battery powered and is low energy-consuming.

       There are two types of sensor nodes: nodes (or frequently called just sensor nodes) and central nodes. This system is composed of many nodes (or sensor nodes) that read the data and then forward it to the central node. The central node collects and saves the data, and also shows it to the user.
    b. RF modules: These are in charge of sending and receiving (or routing) the information to the central node from each node in the network. They have enough transmission power to traverse thin walls or floors.
    c. Circuit interface: This circuit allows the voltage and current readings from an electrical panel. It is directly connected to the sensor node, which performs these readings.
    d. CT: This is used to measure the electrical current of the panel and it must be connected to the circuit in order to allow the readings.
    e. Power adapter: This device is connected to the circuit interface and to the wall. It steps-down the voltage provided so it can be read in safe manner.

2. Software

    a. Front-end interface: This software is in charge of interfacing the central node (the one that receives the information from other sensor nodes in the network). It displays the readings to the user and, at the same time, saves it in a CSV file.

    b. Back-end system: This is the code that controls the sensor nodes and central node. It is in charge of interfacing the RF modules with the sensor itself (prototyping board), so the nodes can receive and send any data. In short, it is in charge of the overall behavior of the sensor, (e.g.: perform readings, prepare a frame to be sent, sent/receive frames).

CHAPTER IV

WIRELESS ENERGY MONITORING SYSTEM: DESIGN

Hardware Design

Radio Frequency Modules

The following list provides the two RF modules that were taken into consideration to be used in the monitoring system:

- Xbee Series 1 [6] (see Figure 7): This module costs less than USD $23. It operates in the 2.4 GHz frequency, its power output is 1mW, it has a maximum range indoor of 30 m and a maximum range outdoor of 90 m, its RF data rate is 250 kbps, it follows the protocol IEEE 802.15.4, and finally the network topology it supports is multipoint (mesh not supported).

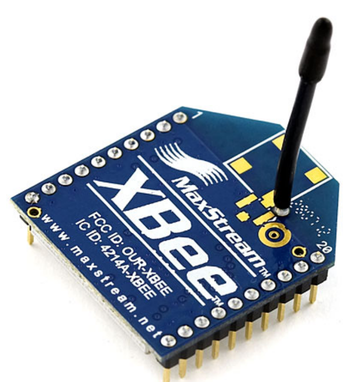

Figure 7. Xbee Series 1 RF module [6]

- Xbee Series 2 [6] (see Figure 8): This module cost less than USD $26. It operates in the 2.4 GHz frequency, its power output is 1.25/2mW, it has a maximum range indoor of 40 m and a maximum range outdoor of 120 m, its RF data rate is 250 kbps, it follows the ZigBee protocol, and finally the network topology it supports is mesh.

Figure 8. Xbee Series 2 RF module [6]

These modules were selected upon the following characteristics: ease of use, available information about their use and configuration, low cost, good range, no operation license needed, high data rate, and low energy consumption.

There are other modules available on the market, for example, the IQRF wireless modules [8]. These modules cost in the range of USD $20, but there is not enough information available on the Internet about their reliability, usage in projects, ease of use, etc.

Finally, from the list of modules given above, the Xbee Series 2 RF modules were chosen as preferred for this thesis. They provide good in-door range (40 meters), good data rate (250 kbps), they are cheap, easy to use, low energy-consuming (3.3V, 40, or 45 mA), and they support mesh (or tree-based) networks. This last characteristic is very important, since one of the prerequisites of this thesis is that the system needs to be scalable and reliable. Therefore, the sensor nodes should be connected in a mesh fashion: if one sensor node cannot connect to the central node because of the distance between them, then the node will be able to connect to any other node in the network and this will forward or route the data to the central node.

Furthermore, these RF modules are supported by the Arduino platform and there exists a large amount of information and projects online on how to use them.

Prototyping Platform

The following list includes the different platforms that were taken into consideration for this system:

- Arduino Pro Mini [9] (see Figure 9): The size of this board is 0.7" x 1.3", its price is less than USD $19 [5], the operating voltage is 3.3V or 5V (depending on model), its input voltage is 3.35 - 12 V (3.3V model) or 5 - 12V (5V model), and the DC current per I/O pin is 40mA.

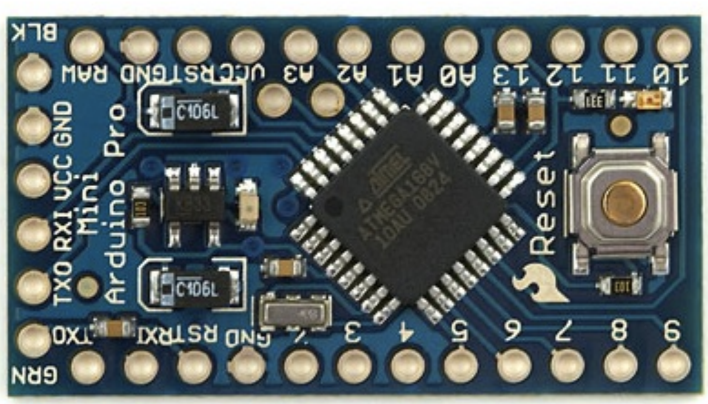

Figure 9. Arduino Pro Mini [9]

- Arduino Mini [9] (see Figure 10): The price of this board is less than USD $34, its size is 0.7" x 1.3", the operating voltage is 5V, the input voltage is 7 - 9V, and its DC current per I/O pin is 40mA.

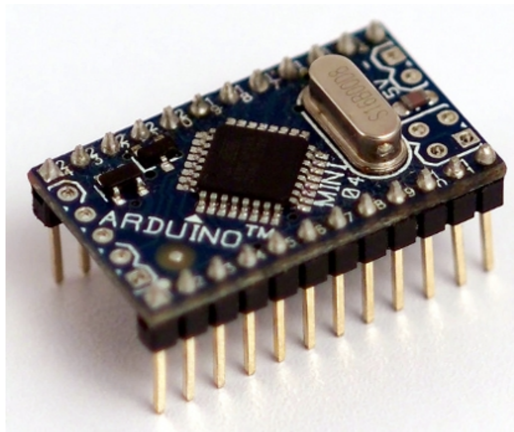

Figure 10. Arduino Mini [9]

- Arduino UNO [9] (see Figure 11): The size of this board is 2.7" x 2.1", its price is less than USD $30, the operating voltage is 5V, the recommended input voltage is 7 - 12V, and its DC current per I/O pin is 40mA and 50mA for 3.3V.

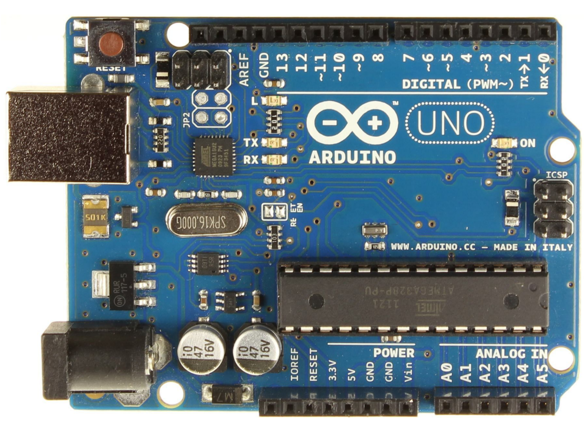

Figure 11. Arduino UNO [9]

- Arduino Fio [9] (see Figure 12): This board has a size of 1.1" x 2.6", its price is under USD $25, it provides a socket for Xbee RF modules on the back of the board, its operating voltage is 3.3V, the input voltage: 3.35 - 12V, and its DC current per I/O pin is 40mA.

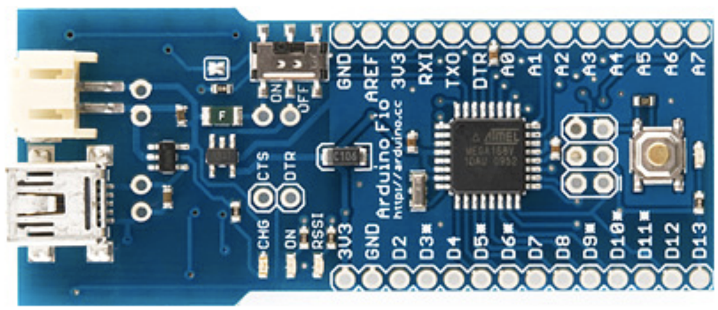

Figure 12. Arduino Fio [9]

All these platforms are from the Arduino open source project. Some of their boards were used in many different projects resulting in very reliable, easy to use, and low cost systems. There are also other prototyping boards, for example, the BeagleBone board (ARM7) from Texas Instruments [10], which provides a lot of computational capabilities (although for this system this is not a need), but it is a much more expensive device compared to the ones listed above.

The chosen board for this system of all the Arduino boards provided in the list above is the Arduino Fio. This board is powered with 3.3VDC, which is perfect since the Xbee RF modules are also powered with 3.3VDC. At the same time, the Fio board physically supports the RF modules since an Xbee socket is available in the back of the board, as shown in the following figure:

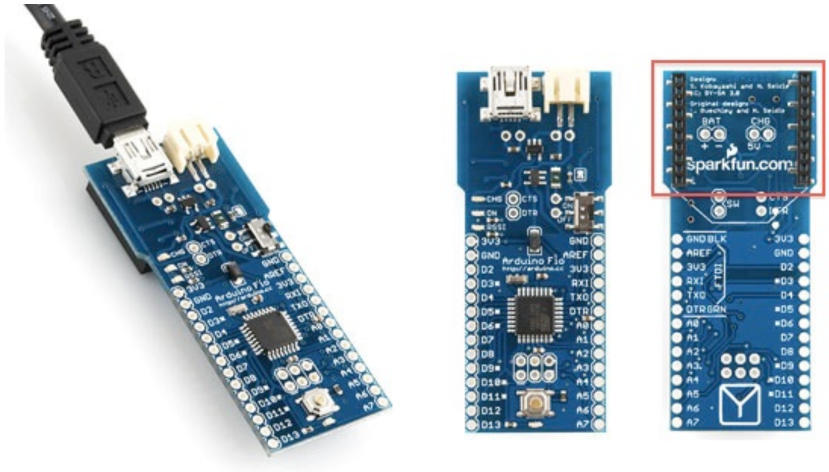

Figure 13. Arduino Fio with socket for Xbee RF modules [9].

The Arduino Fio has a clock speed of 8 MHz, which is enough for the purpose of this system. Also, the Fio board is easy to program due to the C-based language provided by the Arduino Integrated Development Environment (IDE) and it is compact, which makes it suitable for WSN applications, like this system.

Circuit Interface

There are many CTs in the market, like the one shown in Figure 14, that can be used to perform the current readings, some of them come with a burden resistor (which transforms the AC current to AC voltage), and some others do not. The following list gives in detailed the ones are taken into consideration for this system:

- Efergy CT [1] supports a maximum current of 100 Amps and gives an output of approximately 74 mA @ 100A (output type: current).
- SCT-013-000 [1] supports a maximum current of 100 Amps and gives an output of approximately 50 mA @ 100A (output type: current).

- SCT-013-030 [1] supports a maximum current of 30 Amps and gives an output of approximately 1 V @ 30A (output type: voltage).

Furthermore, this circuit interface is entirely based on the OpenEnergyMonitor [1], with two small modifications: first, the chosen CT for this system is the SCT-013-030. It provides a burden resistor; therefore there is no need to perform the calculations to choose a burden resistor and place it in the circuit. And second, the circuit for this system is fed with 3.3VDC instead of 5VDC, since the Arduino Fio provides lower output voltage.

The burden resistor converts the current into voltage, and it is needed since the CT provides a certain amount of current when hooked to the ground wire that is proportional to the current flowing in this wire [11].

Finally, reading the voltage requires a wall transformer (see Figure 15). For this system, a 120VAC to 9V AC-AC adaptor was chosen. This power adaptor steps down the voltage and then feeds it to the circuit, resulting in a safer method of measurement.

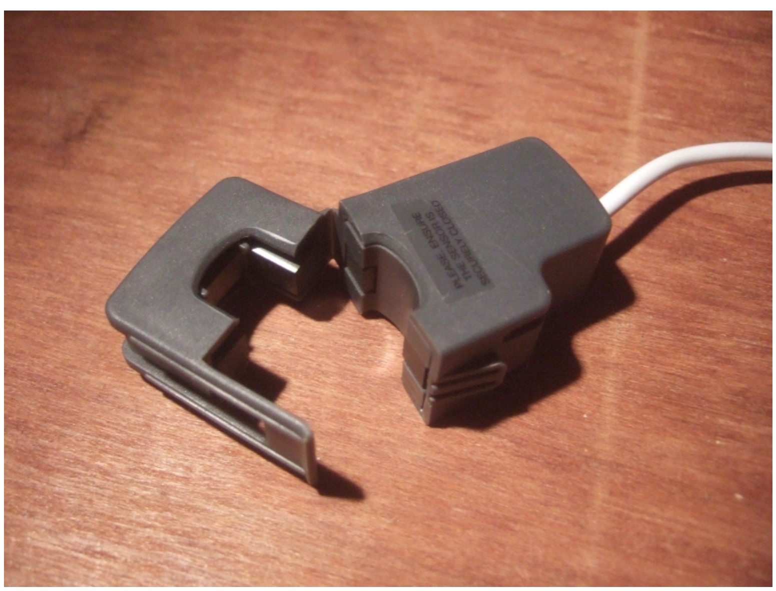

Figure 14. CT [1]

Figure 15. Wall transformer or AC-AC adaptor [1]

The following sections explain how these circuits work, the chosen components, and their schematics [1].

Current Measurement Circuit

In this section, an explanation of the current schematic (see Figure 16) is given. The current circuit is divided into two parts: the CT (SCT-013-030 that includes current sensor and burden resistor of 62 Ohms) and the biasing voltage divider

The sensor produces a current (*Isens*) that is proportional to the instantaneous one (*Iinst*) flowing in the ground wire by:

Isens = CT_turns_ratio x Iinst (calculated in Amps)

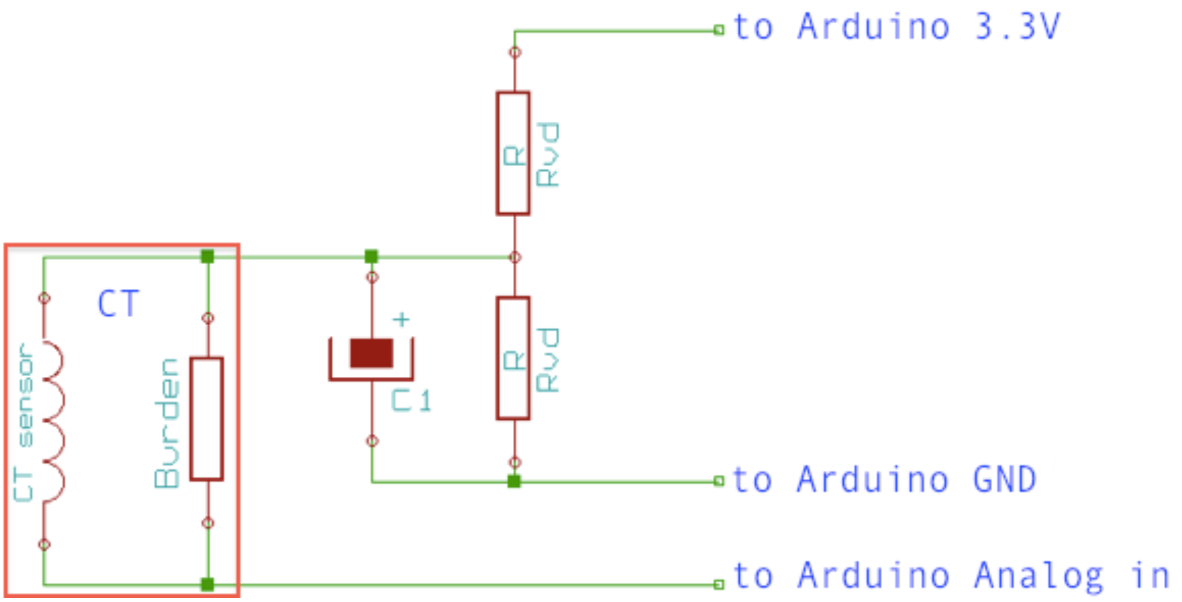

Figure 16. Current measurement circuit [1]

The burden resistor (in the case of the SCT-013-030 CT, this resistor is included in the CT) converts the current *Isens* into a voltage (*VsensI*):

VsensI = burden_resistor x Isens (calculated in Volts)

The two "Rvd" resistors are part of the voltage divider, which outputs a voltage at half the Arduino Fio supply voltage of 3.3V. This voltage (*bias_voltage*) biases the *VsensI* AC voltage (produced by the CT) by 1.65V. This is needed since the analog input of the Arduino requires a positive voltage:

Voltage_to_analog_input = bias_voltage + VsensI (calculated in Volts)

Furthermore, it is important to mention that the capacitor, C1, stabilizes the DC bias, eliminating any possible source of noise that could be fed into the circuit and affect the readings.

Finally, the chosen size for the "Rvd" resistors was 100K ohms (usually between 10K ohms and 100K ohms). Higher resistance lowers energy consumption but generates more noise fed into the circuit. For the C1 capacitor, the chosen value was 10uF.

Voltage Measurement Circuit

In this section, an explanation of the voltage schematic (see Figure 17) is given. The voltage circuit is divided into two main parts: the step-down voltage divider and the biasing voltage divider

The chosen AC-AC power adapter provides the circuit with an AC voltage of 9VAC. The step-down voltage divider scales down this AC voltage from 9VAC to around 1VAC peak to peak (*VsensV*).

The resistors R3 and R4 are part of the voltage divider that outputs a voltage at half the Arduino Fio supply voltage of 3.3V. This voltage (*bias_voltage*) biases the AC voltage produced by the step-down voltage divider (*VsensV*) by 1.65V, needed since the analog inputs of the Fio board requires a positive voltage:

Voltage_at_analog_input = bias_voltage + VsensV (calculated in Volts)

For this circuit, another capacitor was added (C1) in order to stabilize the DC bias and eliminate any source of noise.

Finally, the chosen value for the R3 and R4 resistors was 100K ohms (usually between 10K ohms and 100K ohms). The chosen values for the R1 and R2 resistors were 10K ohms and 100K ohms, respectively. Moreover, the chosen size for the C1 capacitor was 10uF.

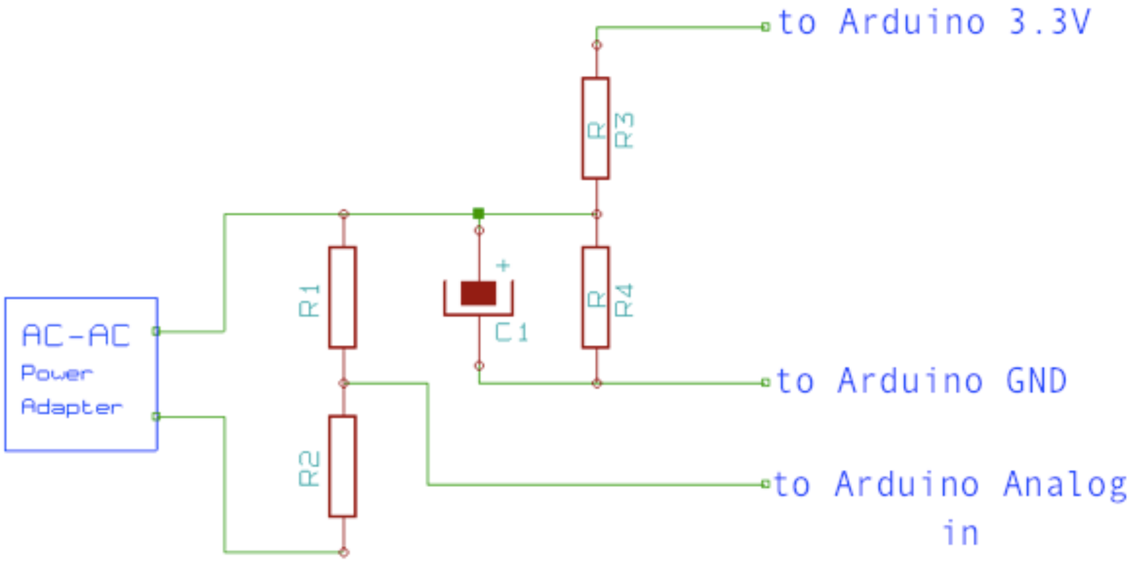

Figure 17. Voltage measurement circuit [1]

Software Design

This section lists and explains the two main software components of this monitoring system: the front-end interface and the back-end system.

Front-end Interface

This is a command line interface that was implemented in C POSIX (Portable Operative System Interface) language and it shows the user the voltage and current readings the central node receives from other sensor nodes in the network. This is achieved due to the serial communication that is established between this front-end interface and the central node.

The interface also saves all the received data in a CSV for further analysis with any other software (e.g.: ETAP or Excel). This file is saved into the folder where the interface is executed from once the user exits the interface.

The information shown to the user is formatted in the following way:

Sensor node, Current (A), Voltage (V), Timestamp: mm-dd-yy(HH:MM:SS)

Where:

- *Sensor node* is the sensor number or sensor name given in the network. This information is important in order to be able to identify where the readings are coming from.
- *Current (A) and Voltage (V)* are the readings performed by that sensor node.
- *Timestamp* (in the given format) is the data and time the readings are registered. This stamp is given to the readings by the interface.

Back-end System

This system is in charge of controlling all the nodes: sensor and central nodes. It provides a serial communication between the Arduino Fio and the Xbee Series 2 RF module. This way, each node can send or receive data using this serial connection. In order to explain how this serial communication works, it is important to understand the different modes the Xbees provide and how these modules work.

The following sections give a detailed explanation on how Xbee Series 2 RF modules work. These sections are divided into: ZigBee, Network topology, Addressing, PAN address, Channels, and API mode.

ZigBee

The Xbee Series 2 RF modules work using a standard communication protocol called ZigBee, which is based on the IEEE 802.15.4. This protocol is for low-power, wireless mesh networking. Xbee is the brand of the RF module or radio that supports these protocols [12].

ZigBee is a set of layers that work on top of the IEEE 802.15.4 protocol. These layers add three very important features to the protocol [12], which are: **Routing** - the routing tables define how a radio passes messages through a series of other radios along the way to their final destination, **Ad-Hoc network creation** - this automated process creates an entire network of RF radios (or modules) on the fly, without any human intervention, and **self-healing mesh** - if one or more RF modules are missing or fail, the network is automatically reconfigured in order to repair any broken routes.

Moreover, there are three types of Xbee Series 2 RF modules that have different roles in the network. These are the **coordinators**: Each ZigBee network must have a single (always one-per-network) coordinator device that is responsible of forming the network, handing out the addresses, and managing other functions that define the network. The **routers**: ZigBee networks can also have routers connected to them. These can join existing networks, send and/or receive information, and route information. Since these modules are routers, they are usually powered from an electrical outlet because they must be turned on all the time. And finally, the **end devices**: these are stripped-down versions of a router. They have fewer capabilities, and they are normally used for sensing the information, sending and/or receiving this information, and joining existing ZigBee networks.

Network Topology

ZigBee network topologies indicate how the radios are logically connected to each other. There are four major ZigBee topologies [12], illustrated in Figure 18:

1. Pair: This is the simplest network possible. It requires just two radios or nodes. One of these nodes must be the coordinator so the network can be formed. The other node can be configured as a router or an end device.
2. Star: In this topology, a coordinator is placed in the center of the topology and connects to a circle of end devices. Every message in the network passes through the coordinator

(which routes them as needed) to reach destination; therefore, end devices do not communicate with each other directly.

3. Mesh: This configuration required router nodes in addition of the coordinator RF module. These radios can pass messages along to other routers and end devices as needed. The coordinator manages the network and also routes messages between devices. End devices may be attached to any router or to the coordinator. They send and/or receive messages to/from other end devices, but they always need their parents' help to establish this communication.
4. Cluster tree: This topology is very similar to the mesh one. Here, routers form a backbone of sorts, with end devices clustered around each router.

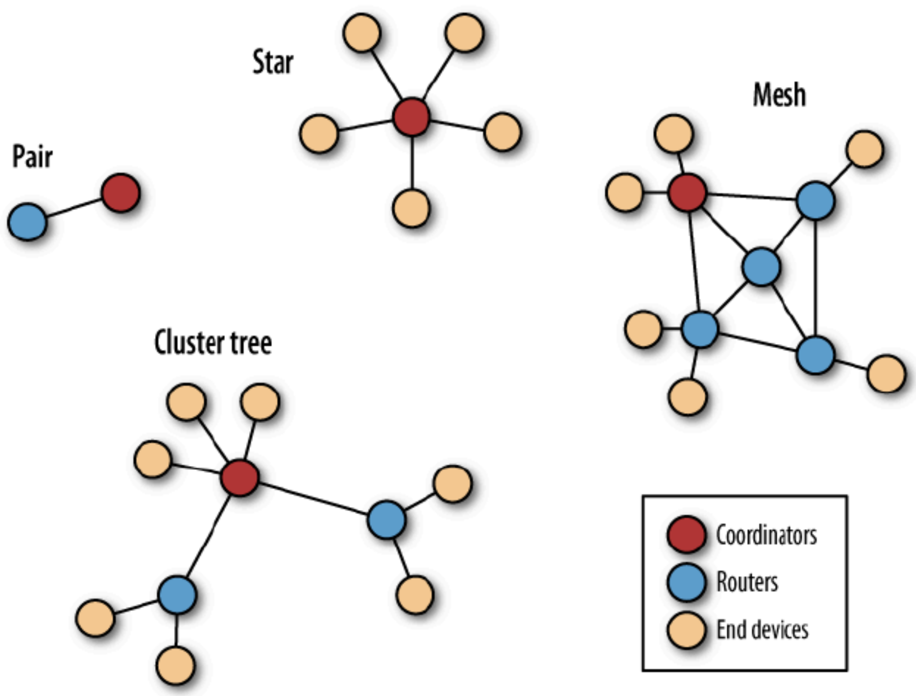

Figure 18. ZigBee network topologies [12].

Addressing

In order to send a ZigBee message from one radio to another, the address of the destination is needed. Each radio has a unique and permanently assigned 64-bit serial address (there are no other ZigBee radio with the same address). However, radios also have a 16-bit address that is dynamically assigned by the coordinator when it sets up the network. This 16-bit address is unique within the network created by this coordinator and since it is shorter, more of them can be manipulated in the limited memory available in the Xbee chip. Finally, each Xbee radio can be assigned a short string of text called "node identifier". This allows the radio to be identifier with a more human-friendly name.

PAN Address

Each ZigBee network that is created has another 16-bit address called Personal Area Network (PAN). There are 65.536 different PAN addresses available, each having the capability to generate another 16-bit radio address below it. In theory, there is room for more than 4 billion radios [12].

Channels

All radios in the network need to be turned to the same frequency (or channel) in order to be able to communicate with other devices. When the coordinator selects a network PAN address, it also scans for all the available channels, typically 12 different ones, and picks one of them for the network's communications. The coordinator tells the rest of the radios in the network what channel to use when they join the network, otherwise they will not be able to communicate with each other.

API Mode

The Application Programming Interface (API) mode in the Xbee modules is used to transmit highly structured data quickly, predictably, and reliably [12]. The Xbee API consists of a series of bytes, each new one building on the information already transmitted. The following table shows the basic structure of an API frame:

Table 1. Basic API frame structure [12].

Start delimiter	Length		Frame data	Checksum
Byte 1	Byte 2	Byte 3	Byte 4 ... Byte n	Byte n + 1
0x7E	MSB	LSB	API-specified structure	Single byte

- Start delimiter: This byte indicates where or when a frame starts, so Xbee RF modules wait for this byte (0x7E) to know when the frame start.
- Length: These two bytes (MSB and LSB) that are received after the start delimiter indicate the overall length of the data frame. Usually, the MSB byte is zero and the LSB contains the entire length.
- Frame data: This is specific to each type of message received in the Xbee radio. Some messages will contain a lot of data, while others will contain just two bytes of data.
- Checksum: This is always the very last byte of the frame. The checksum is the sum of all the bytes that made up the frame, used at the receiving end to check for errors in the transmission of the message. To calculate the checksum, not including the frame delimiter and length, add all bytes, keeping only the lowest 8 bits of the result, and subtract this result from 0xFF. To verify the checksum add all bytes, including the checksum but not the delimiter and the length bytes. If the checksum is correct, the sum will equal 0xFF.

Inside this general frame structure, there are also substructures that cover all the different kinds of data that an Xbee radio is able to send or receive. There more than a dozen of different API frame types currently defined for the Xbee ZigBee modules.

In the first four bytes of an Xbee API frame are the most important ones, since they describe the following information [12]:

- Where the frame begins (start byte)
- How long the frame is going to be (length bytes)
- What kind of frame the radio is looking at (frame type)

Although there are more than a dozen API frame types, the Xbee ZigBee radios used for this system will only use the transmission (TX) request frame (represented by the code 0x10 in hexadecimal). Therefore, no explanation of the other frame types is given.

The TX request frame is used whenever an Xbee RF radio wants to send data to another Xbee RF radio. The frame is composed by the bytes described in Table 2.

Using the 64-bit address requires broadcast transmissions to discover the 16-bit address of the device [12]. Therefore, the system has a limit when using 64-bit address to send information in a network with more than 10 nodes (radios). It is important to migrate the application toward either discovering and using the 16-bit addresses in advance via the API or saving them off-board on the computer or device when it receives incoming data from this remote node. However, the network formed by this Wireless Energy Monitoring System is generally small (less than 10 nodes); therefore the system is still scalable, but it would be required to migrate to one of the solutions given above if more than 10 nodes are used in the network.

Table 2. API format for ZigBee TX request [12].

Frame fields		Offset (Bytes)	Description
Start delimiter		0	Indicates beginning of the frame.
Length		MSB 1 LSB 2	Number of bytes between the length and the checksum.
Frame-specific data	Frame type	3	This is the type of frame. In this case, this would be 0x10 since it is a TX request frame.
	Frame ID	4	Identifies the UART data frame for the host to correlate with a subsequent acknowledgment. If set to 0, no response is sent.
	64-bit destination address	MSB 5 ... LSB 12	Sets the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 – Reserved 64-bit address for the coordinator. 0x000000000000FFFF – Broadcast address.
	16-bit destination network address	MSB 13 LSB 14	Sets the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
	Broadcast radius	15	Sets maximum number of hops a broadcast transmission can take. If set to 0, the broadcast radius will be set to the maximum hops value.
	Options	16	Supported transmission options, which include: 0x01 – Disable ACK 0x20 – Enable APS encryption (if EE=1) 0x40 – Use the extended transmission timeout for this destination Enabling APS encryption decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address. All unused and unsupported bits must be set to 0.
	RF data	17 ... 24	Data that is sent to the destination RF module.
Checksum		25	0xFF – the 8-bit sum of bytes from offset 3 to this byte.

CHAPTER V

WIRELESS ENERGY MONITORING SYSTEM: IMPLEMENTATION AND TESTS

Hardware Implementation

The hardware for this system was relatively easy to implement, since the researcher had previous experience with the components of the system. The hardware implementation costs are explained in the following table.

Table 3. Costs of the hardware implementation.

Part	Cost (in USD dollars)
Arduino Fio Platform	25
Xbee Series 2	26
Power Adapter	6
CT	10
FTDI USB Wire	18
1 Cell Li-Po Battery	8
Circuit Interface	7

In the following section, a detailed explanation on how the system works and how it is connected is given.

System Integration

The connection between all the components is very simple. The Xbee Series RF modules are connected to the backside of the Arduino Fio boads. The RF modules should be facing down.

The circuit interface that allows the Fio board to perform the readings is plugged to the front of the Fio board (pins: GND, 3.3V, A0, and A1). At the same time, from the circuit, both the CT and the AC-AC power adaptor are connected to the electrical panel. The CT is connected to ground wire of the panel; the other end of it is connected to the circuit. The power adaptor is also connected to the panel and the end of it is connected to the circuit.

Finally, the coordinator or central node is connected to the FTDI (3.3V) cable on one end (black wire of FTDI wire goes to GND pin of the Fio board); the other end of it is connected to the computer that executes the front-end interface, as shown in Figure 19.

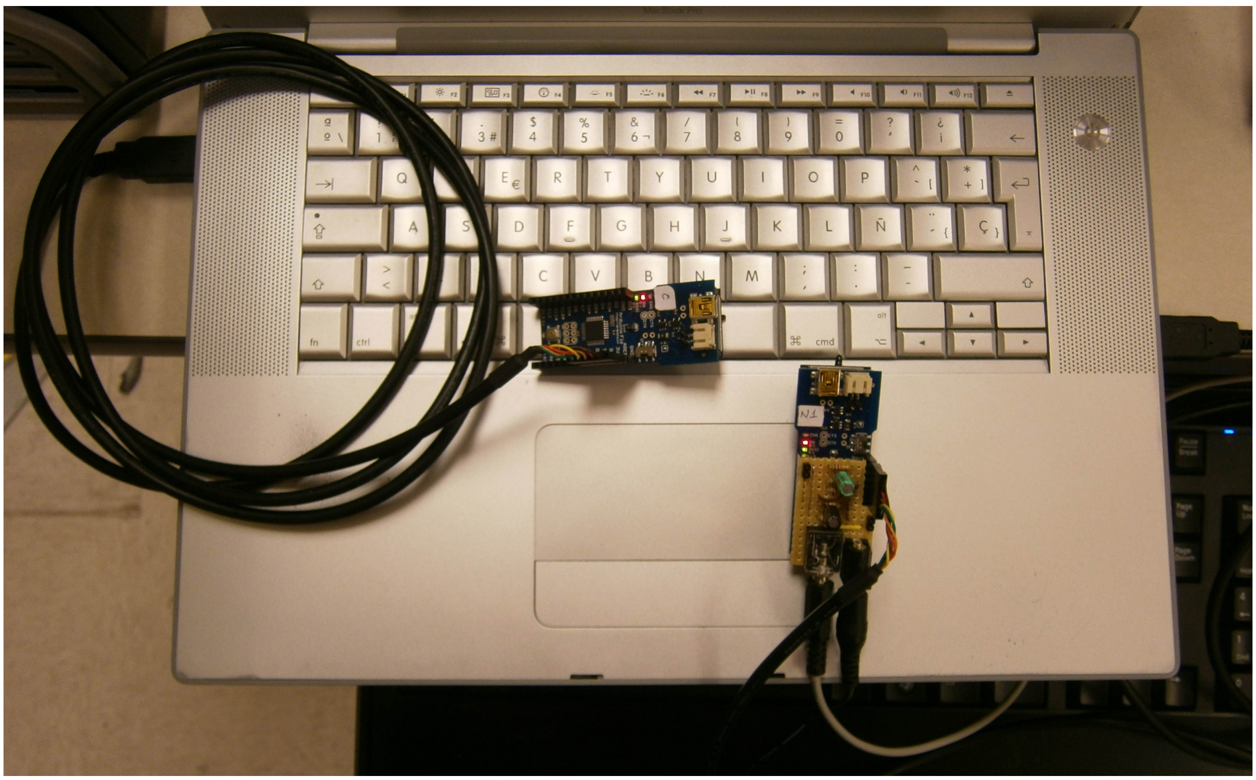

Figure 19. Coordinator node connected to the FTDI USB wire (left) and sensor node with the circuit interface.

Furthermore, Figure 19 also shows the system up and running. The CT of sensor node 1 (on the right) is connected to the Omicron CMC 256-6 relay test set current output wire (output A-1 to N), while the power adaptor is connected to a wall plug. The Omicron CMC 256-6 is used for testing relays and provides electrical energy in a safe and easy way through its outputs. Therefore, this device was ideal for testing the sensor nodes and verifying that the current and voltage readings were correct.

All the Arduino Fio boards can be powered from a 1-cell Li-Po battery (with a nominal voltage of 3.7V per cell). Any other source of power (e.g.: another FTDI 3.3V wire) that does not exceed this voltage is also suitable to power the Arduino Fio boards with the Xbees RF modules.

Software Implementation

Sections of the most important parts of the code for both the front-end interface and the back-end system are shown and explained. In order to have full access to the source code, please refer to Appendix A.

Front-end Interface

This interface is implemented in the "main.c" file. As indicated, it is programmed in C language and is in charge of the serial communication between the central node and the computer. Also, this program saves all the readings and gives them a timestamp before it is exited. The functions used by the interface are:

- int serialport_init(const char* serialport, int baud): This function is in charge of opening the specific port (serialport) in fully raw mode and at a certain baud rate (baud) in order to receive and/or send data using this serial port. It returns a valid serial connection or -1 if an error is encountered.
- int serialport_read(): This function allows the user to read the data available in the serial port in a "char by char" (character by character) fashion. The data is displayed to the user in the command line and saved in a CSV file at the same time. The function

returns -1 if it encounters any problem whenever it attempts to read, otherwise it returns 0;

- void exiting_program(int sig): This function is executed whenever a user (using the command line) wishes to terminate the process that is being executed. Therefore, if the user presses CTRL + C, it will mean he/she wishes to stop the execution of the interface, for which the function will close the serial port and save the information in the CSV file.

In order to execute the interface, the user must first compile the C code using any C compiler and then execute it with the default options, using different ones, or if needed he/she can execute the help option. This process is covered in the following example (Mac OS X v. 10.6 operative system was used in this example):

1. Compile interface: *sudo gcc -o interface ./main.c*
2. Execute default port interface, different port interface, or help:
    a. Execute interface with default options (which is the default port named "/dev/tty.usbserial-A800I8KA"):

       ./interface

    b. Execute interface with different serial port:

       ./interface <serial_port_name>

    c. Execute help option of the interface:

       ./interface -help

3. Exit interface: *CTRL + C*. This will stop the executing of the interface, close the port and save all the information shown in the command line in a CSV file. This file will be available in the same path from where the interface was executed.

Back-end System

This system controls the Arduino boards and the Xbee Series 2 modules. It is written in a C-based language supported by the Arduino IDE. There are two files that compose this system:

- "EnergyMonitor_Coordinator.ino" (it controls the central node or coordinator). This file includes two functions:
  - void setup(): This function is specific to the Arduino language. It is used to initialize pins, create serial communications, call any function before running the main (loop) function, etc. It is only executed once, and as stated before, it is used for initialization processes. In this case, the setup function for the coordinator initializes a serial communication between the Arduino, the Xbee modules, and the computer at a baud rate of 57600 bps.
  - void loop(): This is the main function of the program. It executes the part of the code that is in charge of reading from the serial port all the data that is received with the Xbee radio module. This data is also sent through the serial communication to the computer, with the following format: "node,current,voltage,\n". This will allow the interface running at the computer to know when to separate each field and each reading. This main function is executed indefinitely, until the board is powered off.
- "EnergyMonitor_Router.ino" (it controls the sensor nodes). This file includes three functions:
  - void setup(): This function also initializes a serial communication between the Fio board and the Xbee Series 2 radio. It also initializes the "EnergyMonitor" library that is used to perform the readings in the following way: it indicates the pins that

are used for reading the voltage and the current; and it calibrates the energy monitor to give more accurate readings.

- uint8_t calculate_checksum(uint8_t *frame): This function receives a pointer to a frame (or package that is sent between the nodes of the network) and returns the checksum number of it.
- void loop(): This main function executes indefinitely the code that performs the reading using the function (available at the "EnergyMonitor" library) and the circuit interface that is connected to the Arduino Fio board. The function creates a frame with the readings and sends it to the destination, which is the coordinator or central node. However, this frame could be first sent to another router node/s, in case that the sending node is far away from the coordinator. The time between the readings and the transmission of a frame is 1750ms (1000ms for the reading to be performed and 750ms between transmissions).

Tests

The tests were carried out using an Omicron CMC 256-6 test set. Through this test set's software, a current of 10A from output A-1 to N (ground or GND) was set up; then the front-end interface was executed in the computer in order to start receiving the readings from this sensor node.

Figure 20 shows the connection between the Omicron CMC 256-6 and the sensor node 1.

It is important to mention that the readings had to be calibrated in order to obtain more accurate values. This process is done by changing the values that are passed to the calibration function of the library available at the OpenEnergyMonitor project website [1]. The values were changed and a test was done until accurate readings were obtained. The values used for the final calibration are 0.54643 for the voltage, 0.13399 for the current, and 2.3 for the phase.

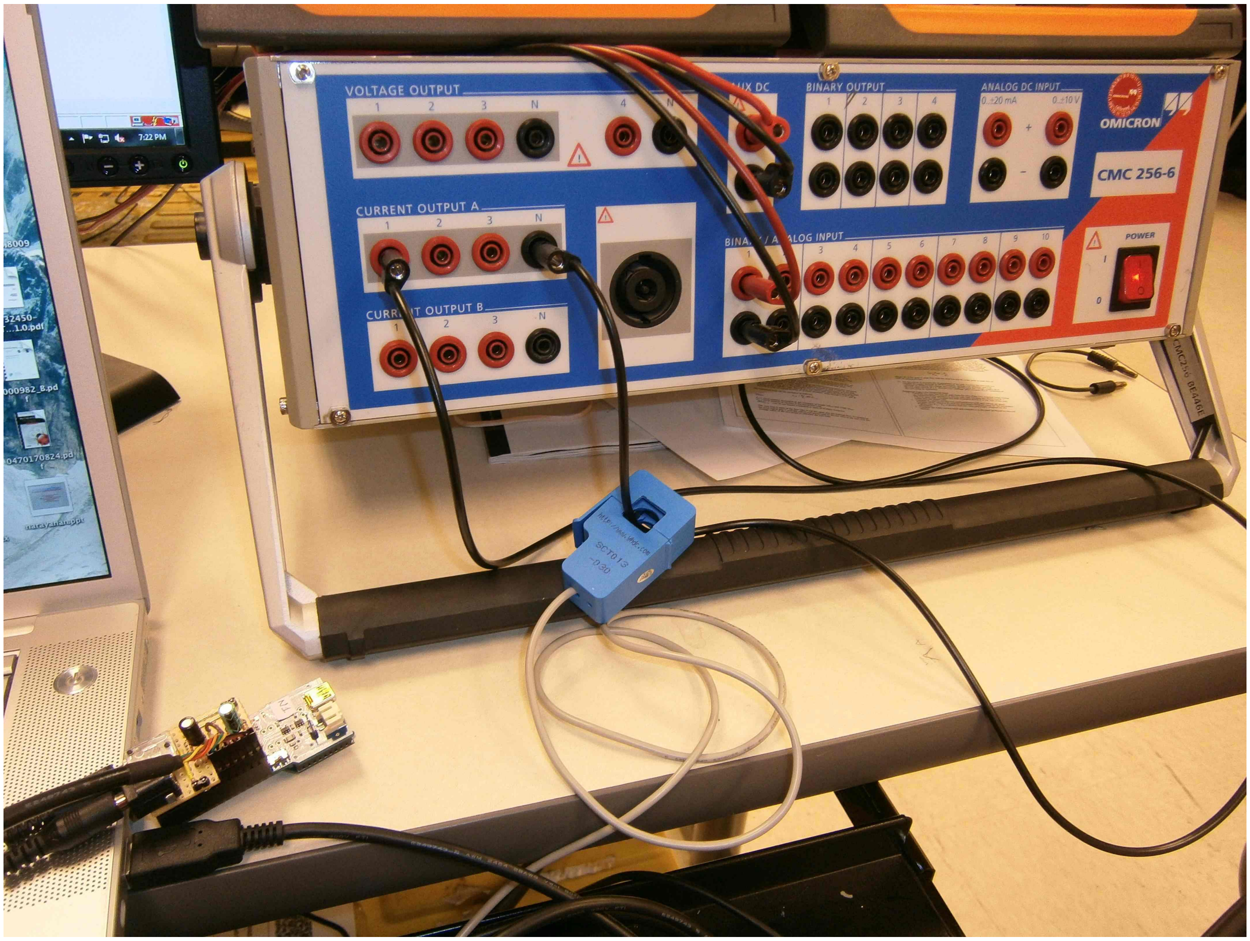

Figure 20. CT connected from the circuit interface to ground wire on the OMICRON 256-6.

The following figure illustrate the process in which the readings are received after executing the front-end interface and when this information is saved in a file after closing the program.

```
Execute program with -help, for more information
Press Control+C to exit the program

Sensor node, Current (A), Voltage (V), Timestamp: mm-dd-yy(HH:MM:SS)
1,  9.99,  120.10,  02-29-12(19:15:47)

Sensor node, Current (A), Voltage (V), Timestamp: mm-dd-yy(HH:MM:SS)
1,  9.99,  120.10,  02-29-12(19:15:48)

Sensor node, Current (A), Voltage (V), Timestamp: mm-dd-yy(HH:MM:SS)
1,  9.99,  120.12,  02-29-12(19:15:50)

Sensor node, Current (A), Voltage (V), Timestamp: mm-dd-yy(HH:MM:SS)
1,  9.99,  120.98,  02-29-12(19:15:51)

Sensor node, Current (A), Voltage (V), Timestamp: mm-dd-yy(HH:MM:SS)
1,  9.98,  120.96,  02-29-12(19:15:52)
```

Figure 21. Execution of the user interface and receive data.

```
Program interrupted. Closing serial connection and log file...
```

Figure 22. Closing program and saving readings in a CSV file.

Figure 21 shows the execution of the program and the information that is received from node 1, which contains the node number, the current in Amps, the voltage in Volts, and the timestamp.

Figure 22, shows the process of closing the program by pressing CONTROL+C. This will stop the execution immediately and save the information in a CSV file.

For the tests shown above, only one sensor node (sensor node 1) was used. Only one implementation of the circuit interface was done since all of them work in the same way. However, the system was also tested with another two sensor nodes (Arduino Fio boards + Xbee

Series 2 radios) attached to the mesh network in order to test connectivity and reliability. The results were positive since no rare behavior was encountered or experienced for long time executions of the system. The readings from the sensor nodes that do not have a circuit interface, nor use CT or power adapters, were obviously incorrect since no circuit interface was attached to them (as mentioned above).

Finally, after the information is saved in the CSV file, it can be opened with different software for further analysis, like for example: Microsoft Excel, Etap, etc. This file is located in the working directory path.

The last step of this test was to calculate the different errors of the readings and determine how accurate the readings were. The following Table 4 shows the 29 readings; Table 5 shows the average of the current and voltage readings and the systematic and maximum random errors.

Table 4. Readings from sensor node and error calculations.

Sensor	Current (A)	Voltage (V)	Timestamp
1	9.99	120.1	02-29-12(19:15:47)
1	9.98	120.1	02-29-12(19:15:48)
1	9.99	119.98	02-29-12(19:15:50)
1	10	119.78	02-29-12(19:15:51)
1	9.99	119.77	02-29-12(19:15:52)
1	9.99	119.64	02-29-12(19:15:53)
1	9.98	119.78	02-29-12(19:15:54)
1	9.97	119.92	02-29-12(19:15:55)
1	9.98	119.97	02-29-12(19:16:56)
1	9.99	119.98	02-29-12(19:16:57)
1	9.99	120.3	02-29-12(19:16:58)
1	9.99	120.6	02-29-12(19:16:59)
1	9.98	120.84	02-29-12(19:16:00)
1	9.98	119.97	02-29-12(19:16:01)
1	9.98	119.97	02-29-12(19:16:02)

Table 4. Continuation

Sensor	Current (A)	Voltage (V)	Timestamp
1	9.98	119.92	02-29-12(19:16:03)
1	9.99	119.78	02-29-12(19:16:05)
1	9.99	119.76	02-29-12(19:16:06)
1	9.99	119.86	02-29-12(19:16:07)
1	9.99	119.82	02-29-12(19:16:08)
1	9.99	119.97	02-29-12(19:16:10)
1	9.99	120.6	02-29-12(19:16:11)
1	9.98	120.4	02-29-12(19:16:12)
1	9.98	119.99	02-29-12(19:16:13)
1	10	120.8	02-29-12(19:16:15)
1	9.99	119.92	02-29-12(19:16:16)
1	9.98	119.97	02-29-12(19:16:17)
1	9.98	120.3	02-29-12(19:16:18)
1	9.99	119.82	02-29-12(19:16:19)

Table 5. Average of the readings and their errors.

	Current (A)	Voltage (V)
Average reading	9.985	120.088
Systematic error	-0.015	0.088
Maximum random error	-0.015	0.812

Errors in experiments or tests can be classified in two different types [13]:

- Systematic errors: These are consistent, repeatable errors. For example, a multi-meter might give 10% of high readings, no matter the input. The first major source of systematic error is that resulting from calibration of the measurement system. The systematic error can be estimated using the following equation:

  Systematic error = average of readings - true value

- Random errors: These errors are caused by a lack of repeatability in the output of the measurement system. The equation to estimate this error is the following:

  Random error = reading - average of readings

To estimate the maximum random error, the reading that deviates the most from the average readings needs to be calculated first. Then, the average of readings is subtracted from it.

Finally, the accuracy of this Wireless Energy Monitoring System is defined by estimating the closeness of agreement between the measured values and the true value. Therefore, it can be determined from the values in table 4 that the system has an accuracy of -0.15% for current readings and +0.75/-0.3% for voltage readings. Therefore, the test and its results show a very accurate energy monitoring system, with low systematic and random errors.

Problems Encountered

The first challenge or problem that was faced when designing the system was using the Arduino Fio platform. In the OpenEnergyMonitor project, a 5V-powered Arduino was used. However, the Fio boards used in this thesis are powered with 3.3V since these also have to power the Xbee radios (Fio boards do not need a breakout board to talk to the Xbees since they work with the same voltage range). This power difference between boards did not make any difference in the circuit interface, but the back-end system had to be calibrated to get a more accurate reading. The calibration values provided by the library used in the OpenEnergyMonitor were specific for a 5V board, so they were changed at the same time a test was being conducted using the Omron CMC 256-6 relay test set.

Another problem that was encountered was that in the beginning of the project was that only Xbee Series 1 radios were available to be used. These radios do not support mesh networks, therefore the system could not be scalable because the end nodes do not have routing capabilities. Although the range tests on this radios were positive, Xbee Series 2 radios were used in the end so the system could be scalable and the end nodes could route data coming from nodes that could not reach the central node.

Finally, the last important problem to be mentioned was encountered when the energy monitoring system was calibrated. Although the system is very accurate, it still needs a better calibration process in other to ensure better accuracy. During the test process, only simple calibration was carried out (change calibration values and check readings against real values). This gave good accuracy to the system (resulting in very low error readings) but probably not good enough if a more intense calibration was done (for example: Taking into account multiple readings through many cycles).

CHAPTER VI

CONCLUSIONS

The Wireless Energy Monitoring System meets all the requirements or prerequisites given in Chapter III. For example, one of the requirements states that the system should be fully scalable, but this is not met in the case of the system having more than 10 nodes. As mentioned in this chapter, using 64-bit addressing could create a bottleneck in the network when it has more than 10 nodes. The solution to this problem is going to be covered in one of the following sections, as well as other improvements that could be taken into consideration.

Finally, the Wireless Energy Monitoring System that was built for this thesis provides an accurate, easy, reliable, and affordable way to measure and control the electrical energy use of any building. These capabilities give extra control to the user that is not commonly provided by power companies.

Future Improvements

It was mentioned in Chapter III that in order for this system to be fully scalable, some changes in the addressing method would need to be done. The system is scalable if there are no more than 10 nodes in the network. However, if more nodes are added and the 64-bit address method is used, a bottleneck in the network is created, resulting in lower data rates, scalability, and performance. To solve or improve this situation, two methods could be used:

1. Node discovery and usage of the 16-bit addressing advanced via the API.

2. Saving the 64-bit addresses off-board on a computer or device when it receives incoming data.

Any of these two solutions would fix the scalability issue, although the system is still scalable when no more than 10 nodes are used in the network.

Another improvement for this system would be designing a new front-end interface since the one that was implemented for this monitoring system is simple and command line-based. It would be a major improvement to design and implement a web-based service that could hold the readings and shown them on a web browser. Therefore, there would be no need of connecting the central node to a computer using a serial port and accessing the data (voltage and current readings) only using this computer. The data would be available from anywhere in the network by just using a web browser, and at the same time, the information could be displayed in a more readable way for the human eye (e.g.: use of graphs, etc.).

Furthermore, improving the calibration process for this monitoring system would be another important task to be considered. In order to do this, a script that logs all the readings and perform the calculations to obtain the calibrations values required by the OpenEnergyMonitor library should be created. As listed in the previous section, the calibration process that was carried out for this monitoring system was simple, although resulting in accurate readings. However, a better and more intense calibration process would give the system more accuracy and lower systematic errors.

Another thing to consider improving is the timestamp that is given to the readings. Normally, all sensor nodes give this timestamp before sending the information to the central node. In order to do this, a GPS unit and some synchronization method needs to be implemented in the system.

Moreover, adding the functions call of the OpenEnergyMonitor library that perform the calculations of the phase, energy, and power, followed by another calibration process, is another improvement that needs to be considered as a future work for this system.

Finally, the last improvement to be considered is to migrate the central node to the "Multi-Fire" microprocessor that is being designed at the University of Tennessee at Chattanooga. This microprocessor is being designed for a Field Programmable Array Gate (FPGA) for a Cyber Physical Systems project. This migration process is ongoing, however this task is not yet completed.

REFERENCES

[1] Trystan Lea, Glyn Hudson, Suneil Tagore, Carlos Alonso Gabizó, Ken Boak, Amin Zayani. (2012, Jan. 5). *OpenEnergyMonitor* [Online]. Available: http://openenergymonitor.org/

[2] EPB (2012, Jan 6). *Calculators Electric* [Online]. Available: Powerhttps://www.epb.net/power/home/energy-efficiency/calculators/

[3] Liljana Gavrilovska, Srdjan Krco, Veljko Milutinovic, Ivan Stojmenovic, Roman Trobec, *Application and Multidisciplinary Aspects of Wireless Sensor Network*, 1st. ed., Springer, 2011.

[4] Amiya Nayak, Ivan Stojmenovic, *Wireless Sensor and Actuator Networks*, 1st. ed., Wiley, 2010.

[5] Arduino platforms (2012, Jan. 10). *Sparkfun* [Online]. Available: http://www.sparkfun.com/search/results?term=arduino&what=products

[6] Xbee radio frequency modules (2012, Feb. 1). *Digi* [Online]. Available: http://www.digi.com/xbee/

[7] CSV file specification (2012, Feb 13) [Online]. Available: http://creativyst.com/Doc/Articles/CSV/CSV01.htm

[8] IQRF wireless modules (2012, Feb. 16). *IQRF* [Online]. Available: http://www.iqrf.org/weben/index.php

[9] Arduino hardware (2012, Jan. 10). *Arduino* [Online]. Available: http://arduino.cc/en/Main/Hardware

[10] BeagleBone board (2012, Feb. 3). *Texas Instruments* [Online]. Available: http://beagleboard.org/

[11] Current Transformer (2012, Feb 8). *Kappaelectricals* [Online]. Available: http://www.kappaelectricals.com/technical.html

[12] Robert Faludi, *Building Wireless Sensor Networks, 1st. ed.,* O'Reilly, 2011.

[13] Anthony J. Wheeler, Ahmed R. Ganji, *Introduction to Engineering Instrumentation*, 3rd. ed., Pearson, 2010.

APPENDIX A

FRONT-END INTERFACE AND BACK-END SYSTEM SOURCE CODES "main.c"

```c
/*
Author: Federico Alejandro Foglia
Master of Science in Engineering: Concentration in Electrical Engineering (Specialty: Computer Hardware)
Thesis: Wireless Energy Monitor
University of Tennessee At Chattanooga
Director: Dr. Stephen Craven
Description: This is a serial program is part of the Energy Monitor System. It is used to receive the readings sent from the Arduino Fio to the PC in order to present it to the user for further analysis. This data is also saved in the PC to be used on other software.

For serial programming on POSIX, Termios info:
http://unixwiz.net/techtips/termios-vmin-vtime.html
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>    // File control definitions
#include <errno.h>    // Error number definitions
#include <termios.h>  // POSIX terminal control definitions
#include <getopt.h>
#include <time.h>
#include <signal.h>
#include <stdbool.h>
#include <string.h>

int serialport_init(const char* serialport, int baud);
int serialport_read();
void exiting_program(int sig);

FILE *logFile;
int serialConn;

int main(int argc, char *argv[])
{
	char *serialport;

	system("clear");
	printf("Execute program with -help, for more information\nPress  Control+C to exit the program\n");
	if(argc > 1)
	{
	  serialport = argv[1];
```

```
          if(strcmp(serialport, "-help") == 0)
          {
                    printf("Execution of the program:\n./interface (by default    /dev/tty.usbserial-
          A800I8KA will be used)\nOR\n./interface    <serial_port>\n");
                    exit(EXIT_SUCCESS);
          }
        }else{
          serialport = "/dev/tty.usbserial-A800I8KA";
        }

        int baudrate = B57600;
        time_t ttime = time(0);
        struct tm *localTime = localtime(&ttime);
        char output[25];

        strftime(output, 26, "log-%m-%d-%y(%H.%M.%S).csv", localTime);
        if((logFile = fopen(output, "w")) == NULL)
        {
          printf("Cannot open file");
          exit(1);
        }
        fprintf(logFile, "sensor,current(A),voltage(V),timestamp\n");
        serialConn = serialport_init(serialport, baudrate);
        (void)signal(SIGINT, (void *)exiting_program);
        serialport_read(serialConn, logFile);

        fclose(logFile);
        close(serialConn);

    exit(EXIT_SUCCESS);
}

/*
 Opens the port in fully raw mode in order to receive and send data.
 Returns a valid fd or -1 on error
*/
int serialport_init(const char* serialport, int baud)
{
    struct termios toptions;
    int serial;
        serial = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
        if(serial == -1)
        {
       perror("Unable to open port");
       return -1;
```

```
    }

    if(tcgetattr(serial, &toptions) < 0)
        {
      perror("Unable to get the term attributes");
      return -1;
    }

    speed_t brate = baud;
        cfsetispeed(&toptions, brate);
    cfsetospeed(&toptions, brate);

    // 8N1
    toptions.c_cflag &= ~PARENB;
    toptions.c_cflag &= ~CSTOPB;
    toptions.c_cflag &= ~CSIZE;
    toptions.c_cflag |= CS8;

    // No flow control
    toptions.c_cflag &= ~CRTSCTS;

    // Turn on READ & ignore control lines
    toptions.c_cflag |= CREAD | CLOCAL;
    // Turn off software flow control
    toptions.c_iflag &= ~(IXON | IXOFF | IXANY);
    // Make it raw
    toptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
    // Make it raw
    toptions.c_oflag &= ~OPOST;
    toptions.c_cc[VMIN]  = 0;
    toptions.c_cc[VTIME] = 20;
    if(tcsetattr(serial, TCSANOW, &toptions) < 0)
    {
      perror("Unable to set term attributes");
      return -1;
    }

    return serial;
}

// Read data from serial port
int serialport_read()
{
    char b[1];
        // Returned value from reading function from serial connection
```

```
        int n;
        time_t ttime;
        struct tm *localTime;
        char output[17];
        bool first_exec = true;

    do{
        n = read(serialConn, b, 1);  // Read one char at a time

        if(n == -1)
            {
                    return -1;    // Error, not able to read
            }
            if(n == 0)
            {
                    usleep(10 * 1000); // Wait 10msec and try again
                    continue;
            }

            // Save information in the log file
            if(b[0] == '\n' && !first_exec)
            {
                    // Last line of each row
                    ttime = time(0);
                    localTime = localtime(&ttime);
                    strftime(output, 18, "%m-%d-%y(%H:%M:%S)", localTime);
                    fprintf(logFile, "%s\n", output);
            }else{
                    if(b[0] != '\n') fprintf(logFile, "%c", b[0]);
            }
            if(!first_exec && b[0] == '\n') printf("%s", output);
            if(first_exec || b[0] == '\n')
                    printf("\n\nSensor node, Current (A), Voltage (V), Timestamp: mm-
                    dd-yy(HH:MM:SS)\n");
            // Print the reading sent form Arduino Fio
            if(b[0] != '\n') printf("%c", b[0]);
            if(b[0] == ',') printf("  ");

            if(first_exec) first_exec = false;
    }while(1);

    return 0;
}

void exiting_program(int sig)
```

```
{
        system("clear");
        printf("Program interrupted. Closing serial connection and log file...\n");

        fclose(logFile);
        close(serialConn);
        exit(sig);
}
```

"EnergyMonitor_Coordinator.ino"

```
/*
Author: Federico Alejandro Foglia
Master of Science in Engineering
Concentration in Electrical Engineering (Specialty: Computer Hardware)
Thesis: Wireless Energy Monitor
University of Tennessee At Chattanooga
Director: Dr. Stephen Craven
Description: Code for coordinator node. It receives the readings (current and voltage) from all
the nodes in the network and forwards it to a PC through serial connection for further analysis.
*/

#define SIZE_TX_REQ 20
void setup()
{
  Serial.begin(57600);
}
void loop()
{
  uint8_t discard; // Bytes to be discarded
  uint8_t node_number, currentIntPart,currentDecPart;
  uint8_t checksum, voltageIntPart, voltageDecPart;
  String current, voltage;
  // Make sure everything we need is in the buffer
  if(Serial.available() >= 21)
  {
    // Look for the start byte
    if(Serial.read() == 0x7E)
    {
      // Read the variables that are not needed out of the buffer
      for(uint8_t i = 0; i < 14; i++)
      {
        discard = Serial.read();
      }
```

```
      // Reading information sent from the nodes
      node_number = Serial.read();

      currentIntPart = Serial.read();
      currentDecPart = Serial.read();
      current = (String)(currentIntPart) + "." + (String)(currentDecPart);

      voltageIntPart = Serial.read();
      voltageDecPart = Serial.read();
      voltage = (String)(voltageIntPart) + "." + (String)(voltageDecPart);

      checksum = Serial.read();

      // Send the data through the serial port to a PC
      Serial.print(node_number, DEC);
      Serial.print(",");
      Serial.print(current);
      Serial.print(",");
      Serial.print(voltage);
      Serial.print(",");
      Serial.print("\n");
    }
  }
}
```

"EnergyMonitor_Router.ino"

```
/*
 Author: Federico Alejandro Foglia
 Master of Science in Engineering
 Concentration in Electrical Engineering (Specialty: Computer Hardware)
 Thesis: Wireless Energy Monitor
 University of Tennessee At Chattanooga
 Director: Dr. Stephen Craven
 Description: Code for router nodes. These read the current and the voltage and send them to the
coordinator for further analysis.
 Emon.h is a library for openenergymonitor
 Created by Trystan Lea, April 27 2010
 GNU GPL
 */
#include "Emon.h" // Load library for energy monitor
#define SIZE_TX_REQ 23
#define NODE_IDENTIFIER 0x01 // Example: Floor where this node is located
```

```
EnergyMonitor emon;

void setup()
{
 Serial.begin(57600);
 // Energy monitor analog pins: 1 for voltage, 2 for current
 emon.setPins(1, 2);
 // Energy monitor calibration: Voltage, current, and and phase calibrations
 emon.calibration(0.54643, 0.13399, 2.3);
}

// Calculate the checksum of a frame
uint8_t calculate_checksum(uint8_t *frame)
{
 long partial_result = 0;

 for(uint8_t i = 3; i < (SIZE_TX_REQ - 1); i++)
 {
  partial_result += frame[i];
 }

 return 0xFF - (partial_result & 0xFF);
}
void loop()
{
 uint8_t *TX_frame = (uint8_t *) malloc(SIZE_TX_REQ); // Frame to be sent
 int integerPart; // Integer part of the readings
 int decimalPart; // Decima part of the readings

 // Frame to be sent to the coordinator with the readings
 TX_frame[0] = 0x7E; // Start delimiter
 TX_frame[1] = 0x00; // Length - MSB: always 0
 TX_frame[2] = 0x13; // Length - LSB: number of bytes that follow, not including checksum
 // Frame specific data
 TX_frame[3] = 0x10; // Frame type: TX request
 TX_frame[4] = 0x01; // Frame ID
 TX_frame[5] = 0x00; // 64-bit destination address (MSB)
 TX_frame[6] = 0x13;
 TX_frame[7] = 0xA2;
 TX_frame[8] = 0x00;
 TX_frame[9] = 0x40;
 TX_frame[10] = 0x3B;
 TX_frame[11] = 0x8E;
 TX_frame[12] = 0xCB; // 64-bit destination address (LSB)
 TX_frame[13] = 0xFF; // 16-bit destination address (MSB), not known
```

```
  TX_frame[14] = 0xFE; // 16-bit destination address (LSB), not known
  TX_frame[15] = 0x00; // Broadcast radius: 0x00 for max. number of hops
  TX_frame[16] = 0x00; // Options: 0x00, all unused
  TX_frame[17] = NODE_IDENTIFIER; // RF data: node identification

  while(true)
  {
   // Energy Monitor calculation function: wavelengths and timeout
   emon.calc(60, 1000);
   integerPart = (int)emon.Irms;
   decimalPart = (int)((emon.Irms - integerPart) * 100.00);
   TX_frame[18] = integerPart;  // RF data: current reading (I)
   TX_frame[19] = decimalPart;
   integerPart = (int)emon.Vrms;
   decimalPart = (int)((emon.Vrms - integerPart) * 100.00);
   TX_frame[20] = integerPart;  // RF data: voltage reading (V)
   TX_frame[21] = decimalPart;
   TX_frame[22] = calculate_checksum(TX_frame); // Checksum
   Serial.write(TX_frame, SIZE_TX_REQ); // Send the frame to the coordinator
   delay(750); // Wait 750ms for coordinator to process data
  }
}
```

VITA

Federico Alejandro Foglia was born in Buenos Aires, Argentina. He moved to Spain in 2002 with his family, where he attended The University of Vigo and graduated with a Bachelors of Science degree in Computer Engineering in May 2010. Alejandro started an international exchange program (ISEP) with The University of Tennessee at Chattanooga, which was concluded in May 2010. Alejandro returned to this university through the same international exchange program to start working as a graduated research assistant for the Electrical Engineering department and to start a Master's program. Lastly, Alejandro graduated with a Master of Science degree in Electrical Engineering (Specialty: Computer Hardware) in May 2012.