

REMOTE TEMPERATURE MONITORING  
SYSTEM

by

David Rubin

Senior Project

New Jersey Institute of Technology

Fall 1999

Abstract

REMOTE TEMPERATURE MONITORING  
SYSTEM

by David Rubin

This project was designed to facilitate the monitoring of the temperature from a remote location. The project is made up of two distinct components, the temperature sensing transmitter and the central receiver.

The temperature sensing transmitter samples the current temperature, then transmits the data over the power line to the central receiver.

The central receiver, a PC, then reads the data off the power line and displays the current temperature on the screen.

## TABLE OF CONTENTS

Cover Page.....	<i>i</i>
Abstract.....	<i>ii</i>
Table of Contents.....	<i>1</i>
Introduction.....	<i>2</i>
Transmitter.....	<i>2</i>
Receiver.....	<i>2</i>
Communication.....	<i>3</i>
X-10 protocol.....	<i>3</i>
Design.....	<i>6</i>
Transmitter.....	<i>6</i>
PL-513.....	<i>6</i>
Power Supply.....	<i>7</i>
Temperature Sensor.....	<i>7</i>
Receiver.....	<i>9</i>
CM11a.....	<i>9</i>
The PC.....	<i>9</i>
Implementation.....	<i>10</i>
Transmitter.....	<i>10</i>
Power Supply.....	<i>10</i>
Temperature Sensor.....	<i>10</i>
Receiver.....	<i>12</i>
Results.....	<i>14</i>
Conclusion.....	<i>15</i>
Bibliography.....	<i>16</i>
Materials.....	<i>17</i>
Appendix.....	

## INTRODUCTION

The need to simply monitor temperature from a remote location has existed for many years. When I decided this is the problem I wanted to solve many different solutions came to mind.

I thought about creating a plug and play web server that would display the temperature. I considered several wireless solutions. I thought about creating a temperature sensing network. After looking into several different ideas I decided to create a temperature sensing transmitter that broadcasts its data over the power lines to a PC which acts as the receiver and presenter of the temperature information.

This project is made up of two distinct component, the transmitter and the receiver.

### **Transmitter**

The transmitter samples the current temperature every second. The transmitter then compares the current temperature to the previous temperature. If the temperature has changed, the transmitter then encodes the temperature data along with the devices ID and transmits it over the power line.

### **Receiver**

The receiver consists of a PC connected to a CM11a. The CM11a is a device that monitors to the power line for X-10 signals. When an X-10 signal is detected the CM11a captures the data. Though a serial port the PC then reads in the data from the CM11a. The PC is then programmed to check that the data received is

error free. If the data is valid the computer then decodes the data and displays the information on the screen.

### **Communication**

The medium that I decided to use to communicate between the transmitter and the receiver is the electrical power line. The reason for this choice was simplicity. All the consumer has to do to get his/her remote temperature monitoring system up and running is to plug the transmitter and receiver into standard AC outlets and install a single piece of software. Within 5 minutes our customer has a completely functioning system.

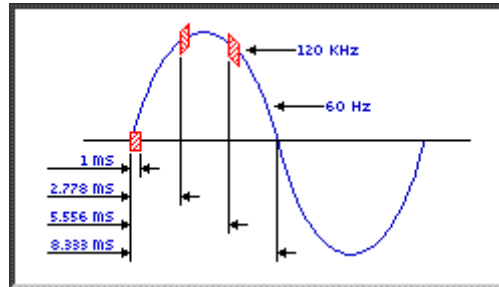
In order to be able to communicate over the power line I decided to make use of the X-10 protocol.

### *X-10 protocol*

X-10 is a communications protocol for remote control of electrical devices. It is designed for communications between X-10 transmitters and X-10 receivers which communicate over standard household wiring. Transmitters and receivers generally plug into standard electrical outlets although some must be hardwired into electrical boxes. Transmitters send commands such as "turn on", "turn off" or "dim" preceded by the identification of the receiver unit to be controlled. This broadcast goes out over the electrical wiring in a building. Each receiver is set to a certain unit ID, and reacts only to commands addressed to it. Receivers ignore commands not addressed to them. X-10 specifies a total of 256 different addresses: 16 unit codes (1-16) for each of 16 house codes (A-P).

The X-10 protocol sends signals at the zero crossing of the AC line voltage and then at 60 degrees and at 120 degrees after that (in other words, three times every

half cycle corresponding to the zero crossing of the other two phases). This is designed to make X-10 compatible with three phase power situations. See below:



A ONE(1) bit in a legitimate X-10 transmission is a 1 millisecond (mS) pulse code modulated burst of 120KHz on the AC line, and a ZERO(0) is the absence of that burst. The exact length of the burst may not be critical in most applications. The burst is sent three times for each bit, once at each AC zero-crossing (accounting for zero-crossing in 3-phase). That means once each 2.778 mS. The next bit is sent on the following zero-crossing. This is done to get the quietest time on the AC line for the receiver, whatever phase of the AC it's on. The zero crossing gives the best signal-to-noise ratio for data transmission because everything should be shut down then (i.e. the voltage is low).

In addition, each bit is sent both true and complemented, and each code sequence is sent twice. That's a lot of bit redundancy, and just barely enough to make it past the noise on the line, depending on actual conditions.

A single normal command takes eleven cycles of the AC line to finish. All legal commands must first start with the header 1110, a unique code as described below. The header bits take two cycles at one bit per half cycle. The next four cycles are the four-bit House Code, but it takes eight bits total because each bit is sent true then complemented. This is similar to bi-phase encoding, as the bit value changes state half-way through the transmission and improves transmission

reliability. The last five AC cycles are the Unit / Function Code, a five bit code that takes ten bits (again, true then complemented). For any codes except the DIM, BRIGHT and the data following the EXTENDED DATA function, there's a mandatory three cycle pause before sending another command. DIM and BRIGHT don't necessarily need a pause but the data after the EXTENDED DATA command absolutely MUST follow immediately until all bytes have been sent. The EXTENDED DATA code is handy, as any number of eight-bit bytes may follow. The data bytes must follow the true/complement rule, so it will take eight cycles per byte, with no pause between bytes until complete. The only legal sequence that doesn't conform to the true/complement rule are the start bits 1110 that lead the whole thing off because the modules need some way to tell when it's okay to start monitoring again.

A full transmission looks like this(ex. /H8 is the complement of H8):

1 1 1 0	H8 /H8 H4 /H4 H2 /H2 H1 /H1	D8 /D8 D4 /D4 D2 /D2 D1 /D1 F /F
Start	House code	Unit/Function code

So, to activate Unit 12 of House code A, send the following:

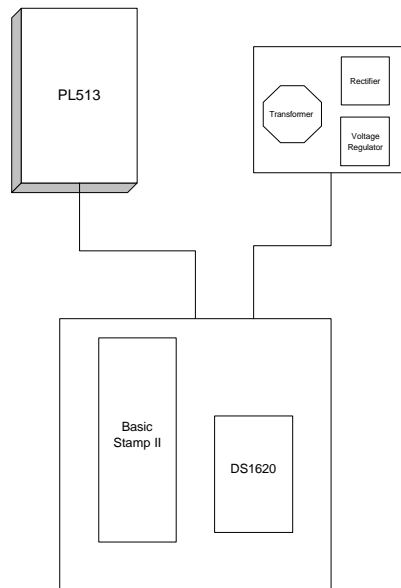
1 1 1 0	0 1 1 0 1 0 0 1	1 0 0 1 1 0 1 0 0 1
Start	House code	Unit/Function code

This sequence would be sent 2 times and at 3 different phases so as to insure that the transmission reaches its destination.

## DESIGN

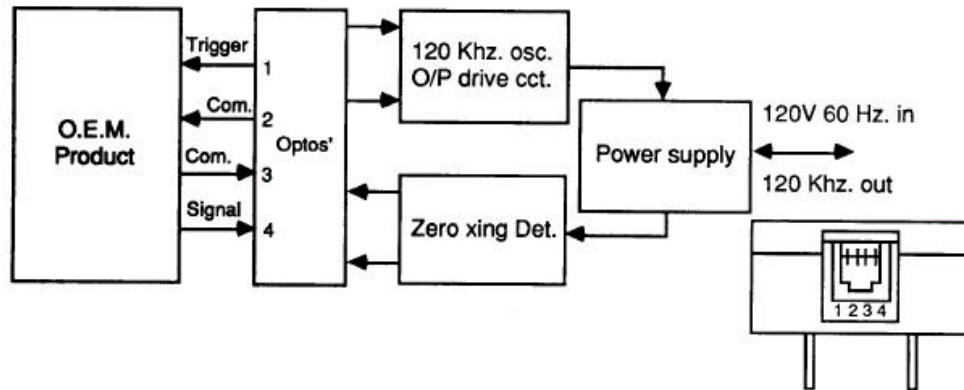
### Transmitter

I decided to the design of the transmitter in a modular fashion. By using modules in the design I was able to build and test each component separately. This decision eased both the construction and testing time enormously. The temperature sensing transmitter was split into three component:



### *PL-513*

The PL-513 is a power line interface for X-10. It provides a 4 wire interface for all OEM products that wish to make use of an X-10 to communicate over the power line. The diagram on the top of the next page demonstrates how the interface works.



### *Power Supply*

The power supply module was designed to keep noise off of the power line. It consists of a 110 V to 12.6 V transformer, a full-wave bridge rectifier, an electrolytic capacitor and a linear voltage regulator.

It is common today to use switching regulators instead of linear regulators. My decision to go with a linear regulator was driven by the fact that switching regulators produce lots of high frequency noise. It was possible that this high frequency noise would disrupt the X-10 transmissions.

### *Temperature Sensor*

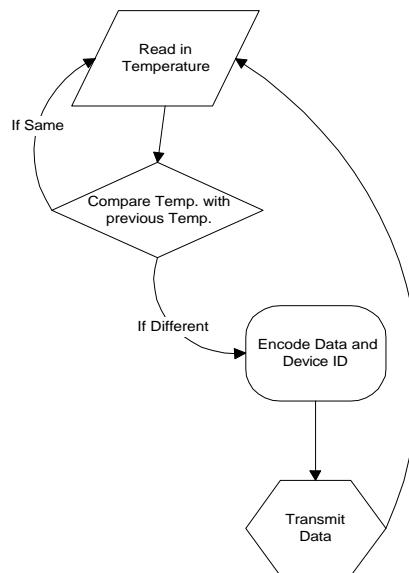
The temperature sensor module consists of the Basic Stamp II micro-controller and a DS1620, a Dallas Semiconductor digital thermometer.

The decision to go with the DS1620 was a relatively easy one. The DS1620 has almost become the standard of the embedded systems engineer. The DS1620 measures temperatures in units of 0.5 degrees from -55° to +125° C. The DS1620 makes its temperature data available in a 9-bit, two's complement format. Each unit represents a 0.5° C, so a reading of 50 translates into +25° C.

Negative numbers are expressed as two's complement numbers with a leading 1 representing a negative number.

The decision of which micro-controller to go with was not as easy. Timing is extremely important when it comes to X-10 and, therefore, the choice of micro-controllers is of great significance. I looked at several dozen different micro-controllers. There were many issues that had to be weighed when choosing the micro-controller for this project. Cost, access to a micro-controller programmer, ease of use, as well as a proper environment to testing and debugging were some of the issues that I considered. In the end I decided to go with the Basic Stamp II.

Below you will find a flow chart of the micro-controllers function in the temperature sensor and transmitter:



**Receiver**

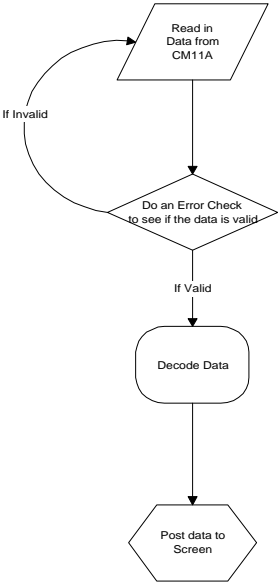
The receiver consists of a PC connected to a CM11a via a serial port.

*CM11a*

The CM11a is a device that monitors the power line for X-10 signal. When the CM11a detects an X-10 signal it copies the transmission into memory. The CM11a device can store up to ten X-10 transmissions. In order to read the X-10 data into the PC you must communicate with the CM11a using the proper protocol. The CM11a communicates with the PC at 4800 baud. A reference to the full CM11a protocol can be found in the bibliography.

*The PC*

The PC must communicate with the CM11a with the proper protocol in order to receive the X-10 transmission. A flow chart for the operation of the software can be found below:



## IMPLEMENTATION

### **Transmitter**

As discussed earlier the transmitter was designed as 3 separate modules. After studying the function of the PL-513 it was fairly easy to communicate with it through its 4 wire interface.

### *Power Supply*

Once I had decided to go with a linear voltage regulator I thought the rest of the design would be straight forward. After constructing my power supply I discovered that not all of the AC was being filtered out. Because of this, the voltage regulator was not functioning. After further study into the construction of power supplies I discovered that I was not using the proper capacitor. An electrolytic capacitor was necessary, in order to deal with the constantly fluctuating AC waveform. After tweaking the capacitance values I was finally able to generate a clean 5 volts.

### *Temperature Sensor*

The temperature sensor, consisted of the DS1620 the Basic Stamp II micro-controller and assorted capacitors and resistors.

The DS1620 makes its temperature data available in a 9-bit, twos complement format. Each unit represents a  $0.5^{\circ}$  C, so a reading of 50 translates into  $+25^{\circ}$  C. Negative numbers are expressed as twos complement numbers with a leading 1 representing a negative number. Being that I am only concerned with positive values I have decided to write my program to ignore values in which the ninth bit is a 1.

The DS1620 can measure temperatures from  $-55^{\circ}$  to  $+125^{\circ}$  C. The operating temperature range for the Basic Stamp is 0 to  $+70^{\circ}$  C. Therefore, I would have liked to recognize all the temperatures in the range from 0 to  $+70^{\circ}$  C. I will discuss later why I was unable to do this.

The software in the Basic Stamp chip was to perform 3 functions. First the chip reads in the current temperature from the DS1620. If the temperature is the same as the previous temperature, it loops back to read the temperature again. This design decision was made in order to minimize the number of transmissions made and thereby minimize the noise on the power line. Once we have a new temperature reading the micro-controller encodes the temperature data along with the device ID into an X-10 format. Then the micro-controller transmits the data to the PL-513 for transmission over the power line.

X-10 was never intended to be used to transmit data and for the most part it has not been used for that purpose. X-10 is used primarily in home automation products to turn on, off and dim lights.

In 1978 the designers of this protocol did put in the ability to transmit data. Although it has gone mostly unused, it was my intention to make use of this mostly unused feature in this project.

The ability to send data using the X-10 protocol is made available through the use of the EXTENDED DATA function code. Usually after you send a command to activate house code A, unit 12 you would send a function code such as 00101 which would turn on that light. When the function code for EXTENDED DATA(11001) is used, immediately followed by a sequence of 8 bits, that 8 bits can be any data that you might want to send, which in my case is the temperature. The key is that there must not be any delay between the EXTENDED DATA code and the actual data.

As part of my project I came up with an encoding method that would allow me to encode my temperature information and the unit ID in those 8-bits. In addition my system was going to be completely compliant with the X-10 protocol. This would allow other X-10 devices to operate on the same system.

The Basic Stamp II did not provide support for the EXTEND DATA function. I would have to create the functionality on my own. Unfortunately when I set out to design this functionality I did not realize how sensitive to timing it was. After a few weeks I came to realize that this functionality could not be built with the Basic Stamp. The Basic Stamp runs Basic as an interpreted language. Without access to the assembly code, this functionality could not be created.

To every problem there is a solution. It did not take me long to come up with a work around. I decided that I would just encode my temperature information in the header data. That was a great workaround because it meant that my X-10 transmitter would still adhere to the X-10 protocol and thereby not interfere with any other possible X-10 transmissions on the same power line. The only problem with this solution was that I now had much less space to encode my temperature data. I could now only transmit 32 different temperatures. So I decided to transmit temperatures in the 10 – 41 degree Celsius range (in 1 degree increments). This corresponds to between 50 and 105 degrees Fahrenheit, still a very nice range.

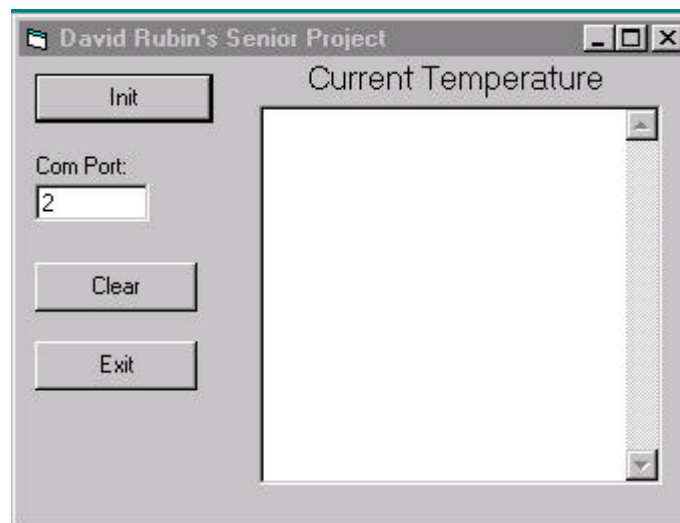
### **Receiver**

I first wrote the software to communicate with CM11a in the Linux operating system. I wrote the software in Python, a very portable language. My plan was to write the first draft of the software under Linux and then to port it to Windows.

I had a fairly decent implementation completed under Linux when I decided to port it to Windows.

Although my Python was quite portable, it was not easy to change Linux system calls to Windows system calls.

After much frustration I decided to start from scratch and reimplement my code in Visual Basic 6.0. I now have a very nice Windows graphical user interface to my program.



## RESULTS

This project works very much like my original proposal. We have one module that when plugged in to a standard AC outlet broadcast the temperature data over the power line. We have another module that reads the data off of the power line and displays the current temperature.

The one place where the implementation varies with the proposal is in the display. In my proposal I stated that I would display the current temperature on a web page. In my Linux implementation this was easy to do with one simple CGI program.

I realized that a web-browser based display would not be the best method of demonstrating my project. The nature of the web-browser is that the pages are static. I want to demonstrate a temperature that is constantly changing. Data which is constantly changing and web pages which are static are a poor combination.

A logistical problem also exists with installing a web-server on NJIT PC's. Therefore, demonstrating the project through a Windows GUI seemed like the most logical option. A combination Linux/web-server option is in the works (see conclusion for more information).

## CONCLUSION

This project as it stands now is a fully functional and useful product. While doing research for this project I discovered that there are many X-10 hobbyist looking for a product such as this.

The cost of the parts to complete a product such as this is approximately \$160. That would make this product out of reach of most X-10 hobbyist. A large part of the expense for this project was the Basic Stamp II micro-controller at \$50. The Basic Stamp was a great chip for experimentation, but it is prohibitively expensive for commercial production in a product such as this. It would not be to difficult to port the functionality of the Basic Stamp to a much more economical PIC micro-controller. Most X-10 hobbyist already own a CM11a. Eliminating the cost of the Basic Stamp along with the cost of the CM11a, the temperature sensing transmitter can now be constructed for a much more reasonable \$50.

The original inspiration for this project came from my father's need to monitor the temperature of an animal room for mice in medical experiments. My father travels significantly and, therefore, he must be able to monitor the temperature from a remote location. With the Internet accessible from almost anywhere in the world, a web browser is the perfect method for remote modeling.

It is my intension to complete the Linux\web-server version of the receiver in order to give my father the ability to monitor the temperature of his animal room from anywhere in the world.

## BIBLIOGRAPHY

Dietz, Paul H. *A Pragmatic Introduction to the Art of Electrical Engineering*, 1st ed. Hopkinton, MA: Paul Henry Dietz, 1998.

BASIC Stamp Programming Manual  
Version 1.9  
<http://www.parallex.com>

*X-10.org*  
<http://www.x10.org/>

*X-10 FAQ*  
[ftp://ftp.scruz.net/users/cichlid/  
public/X-10faq.txt](ftp://ftp.scruz.net/users/cichlid/public/X-10faq.txt)

*X-10.com*  
<http://www.x10.com/>

