APPLIED PROTOCOLS IN SMART CONTROL

By
Ahmed Abdel Kareem Mourad
Ragab Mustafa Abdel Gawad
Mohamed Sayed Mohamed
Asmaa Mohammed Zohri
Asmaa El Sayed Ahmed

A Graduation Project submitted in
partial fulfillment of the requirements
for the degree of

BSc in Electronics and Communication
Engineering

Fayoum University

2010

Approved by **Amr M. Gody**_____

_____

_____

_____

Program
to Offer Degree Bachelor in Electronics and Communication Engineering _____

Date July 2010_____

FAYOUM UNIVERSITY

APPLIED PROTOCOLS IN SMART CONTROL

By
Ahmed Abdel Kareem Mourad
Ragab Mustafa Abdel Gawad
Mohamed Sayed Mohamed
Asmaa Mohammed Zohri
Asmaa El Sayed Ahmed

Supervisory Committee: Associate Professor **Amr M. Gody**
Department of Electrical engineering

A work presented on Smart home and how to automate your home with low cost and high reliability, also representing some technologies and how can using it to utilize the home automation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODES

Executive Summary

Technology in today's world is advancing at a very rapid rate. Once a rare commodity, computers can now be found in hundreds of millions of homes and businesses. A growing trend involving computers is industrial control and home automation, a practice in which electrical devices are controlled with little or no human interaction. Although this may sound like a noble concept, many of these control systems suffer from poor performance in terms of data communications capability. They also require the user to configure them locally, which makes it difficult to check the status of the systems from afar. Also, many systems carry a steep price tag that many potential buyers find Unappealing.

To remedy the first of these issues, a new data communications system will be developed. The system will consist of one or more host units and multiple target units. The host units will initiate all data communications processes to the target units, and a target unit may reply only to the host that hails it. Only one communications process may exist at any given time per host, preventing data communication collisions. Existing electrical wiring will serve as the communications medium, preventing the expense of installing additional wiring in the building.

The access issue will be remedied by designing a software package for a personal computer. The software will allow the host device to connect to a PC, as well as the Internet using TCP/IP. The user will therefore be able to access the host device through a standard Internet connection.

A third issue is the cost of comparable control systems. Presently-available systems that are used in industry cost thousands of dollars. By programming a microcontroller to emulate traditional hardware, less electronic components will be required to build a working system. As a result, overall production costs will be substantially lower than comparable systems.

We will approach this project by dividing it into several key components. An X10 Transmitter will be designed that will allow the host and target units to communicate over the power line. Microcontroller firmware will be created to control the functionality of both host and target units. Circuitry and firmware will be implemented to interface the host unit to a PC. Software will be written to allow user-control over host and target units.

Our design will be superior to presently-available device control systems in that ours will reduce the number of control errors due to corrupted data transmissions, thereby enhancing the reliability of the system. It will also provide an easy-to-use interface that will allow users to render remote control over all host and target units and all of their associated peripherals.

Considering the ongoing growth in popularity of home and industrial control systems, this project has an abundant future. Potential design enhancements include improving the data transfer rate of the modems and enhancing the remote web interface.

ABSTRACT

As the need for control automation systems increases, the number of commercially available systems is broadening every day. Of these systems, the most reliable ones can cost thousands of dollars. The inexpensive ones suffer from poor reliability and must be controlled locally. To solve these issues, a system that is reliable, affordable, and easily accessible is needed. This system is the PLCS Using X10 Protocol, RS485 Protocol, TCP/IP Protocol and finally FBUS Protocol. This automation system will improve the addressed issues by utilizing the capabilities of both customized hardware and software. The system will use communications chips that have costs comparable to Inexpensive systems, but with reliability found in more robust systems. Users will also be able to access the system through a network, such as the Internet, thereby eliminating the constraint of requiring local control.

Chapter I: Introduction

1.1 Introduction

The purpose of this project is to design and implement a device control system that utilizes a building's existing electrical wiring as a communications medium. This design will give consumers a more cost-efficient device control solution. The design approach will consist of simulation, construction, and field testing of the system. The significance of the device is that all host and target units may be controlled by any computer connected to the network.

In the early 1970's, Pico Electronics Ltd. was founded by a group of investors who wanted to develop integrated circuits for the handheld calculator market. Each time Pico Electronics started a new project, the project was given an experiment number. Their ninth experiment, "experiment #9", was an integrated circuit for a programmable record changer for a phonograph. Shortly after this, Pico Electronics was asked to build a wireless remote control system for the record changer. This became "experiment #10" for Pico Electronics, or "X-10" for short.

The X-10 systems were designed to use existing household wiring to control devices throughout the household. Pico engineers soon realized that this system had many other uses besides controlling record changers. In 1978, Pico Electronics signed contract with several large retail stores to sell the X-10 system. The system was soon being advertised in 1979, and now, 20 years later, it is still growing in popularity.

In more recent years, several enhanced carrier-current networking solutions have been introduced. CEBus and Lon Works boast improved data transmission rates over X-10, as well as improved error detection, but they also "boast" much higher price tags.

1.2 Statement of Problem

Industrial control and home automation has rapidly been gaining popularity for the past decade. Although many automation and control products are available on the market, many of them suffer from low data rate as X10 Protocol Communication and the other suffer from that has high Cost and not reliable as CEBus and LonWorks.

A second issue encountered with these systems is some of them need cables (LAN) to extend to control in the devices as LonWorks, and that is not preferred in home automation and also for the users.

To remedy the communications issue, our team will develop a more reliable method to communicate with and control in the home devices. This will be high data rate and less cable will extend and low error probability. This done by combining more than one system with them we mean X10 protocol communication, RS485 Protocol communication, TCP/IP protocol communication and Fbus protocol for Nokia Phones.

The accessibility issue will be resolved by designing a user-friendly computer interface for the system. With this interface, users will be able to remotely check the status of their

systems, therefore eliminating the worry of discovering that their "controlled" devices are on the blink. Users will also be able to remotely make changes to the operation of their devices to comply with their schedules.

1.3 Objectives

A. **Data Transfer Rate:** Most inexpensive automation systems have a data transmission rate of only approximately 100 bits-per-second (bps), whereas the more costly ones have a rate of several thousand bps. Our system's rate will be a compromise between the two, with a minimum rate of 600 bps and a desired rate of 1200 bps.

B. **Error Rate:** Low-cost carrier-current data transmission systems can have a less-than-appealing rate of errors. Our system will have a maximum bit error rate of 0.01%.

C. **System Structure:** The most common low-cost automation system can support a maximum of only 256 target units, and in this Project the number of devices increased by using the RS485 Network and the TCP/IP Network.

D. Power Usage: this Project made specifically to utilize the usage of the power because the one of three controlling signal is carried by the power line (signal of X10 Protocol communication), and the other devices use only power source of value +5V, So the user can't worried about power usage.

E. **Cost:** the cost will be low because the basic devices and the protocols are cheaper as example the most expensive device is the XM10 Module with approximately price 40$ and all other devices will be approximately with price 10$.

Chapter II: Conceptual Framework

2.1 Static class diagram

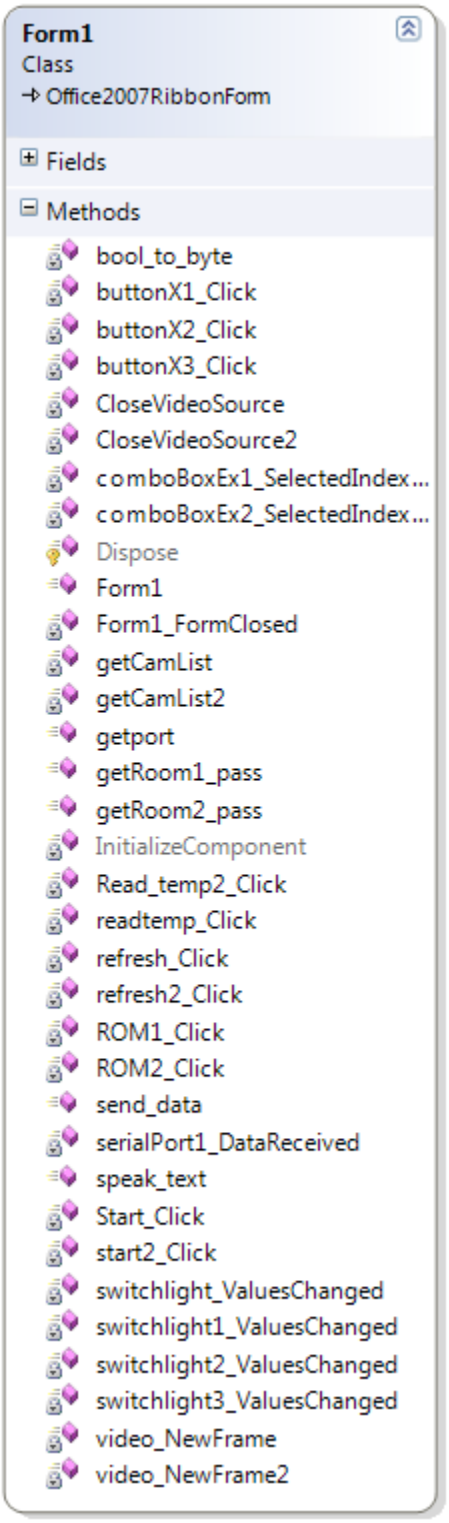


Figure 1 Class diagram for Software interface (Server) 1

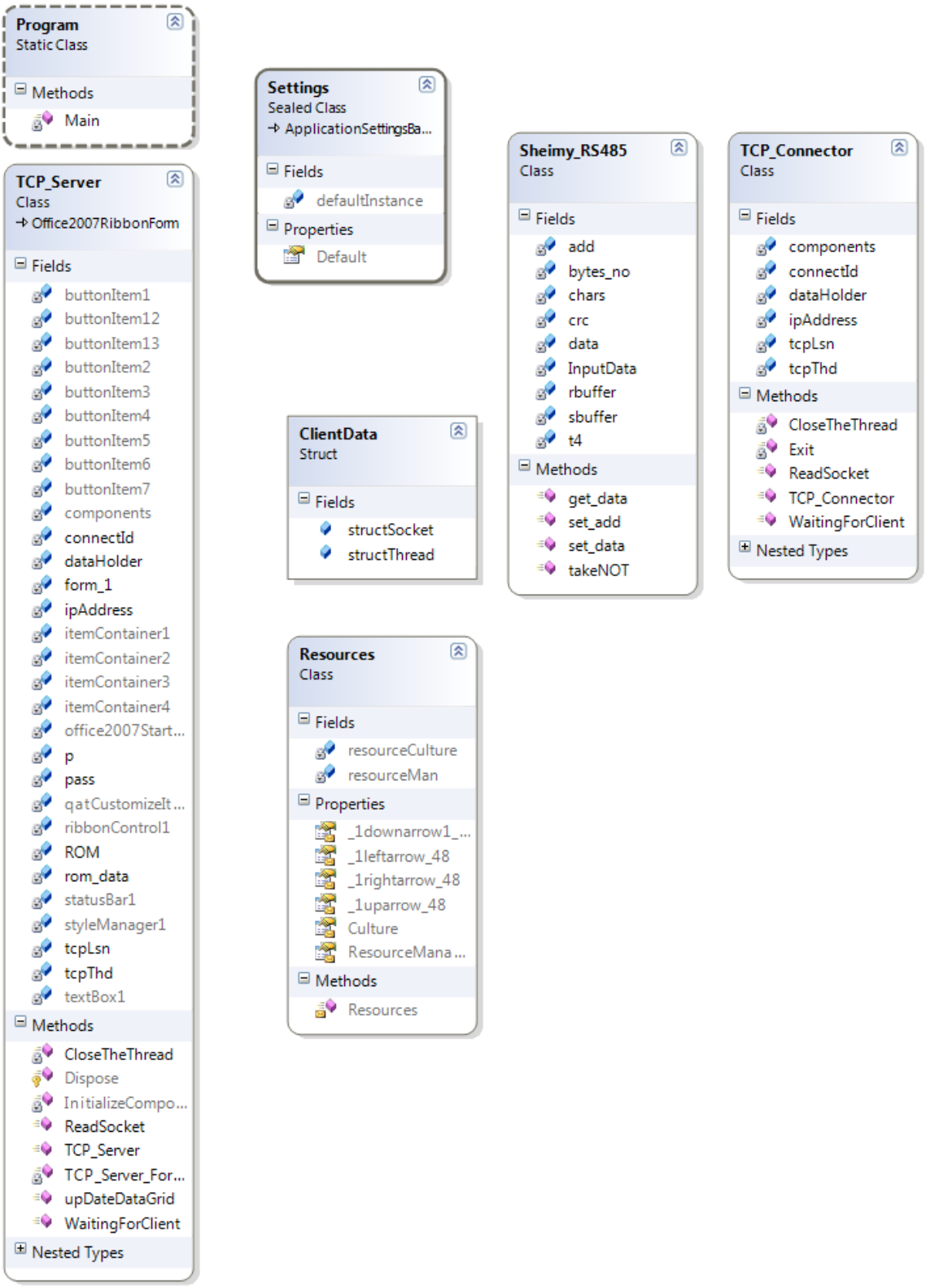


Figure 2 Class diagram for software interface (Sever) 2

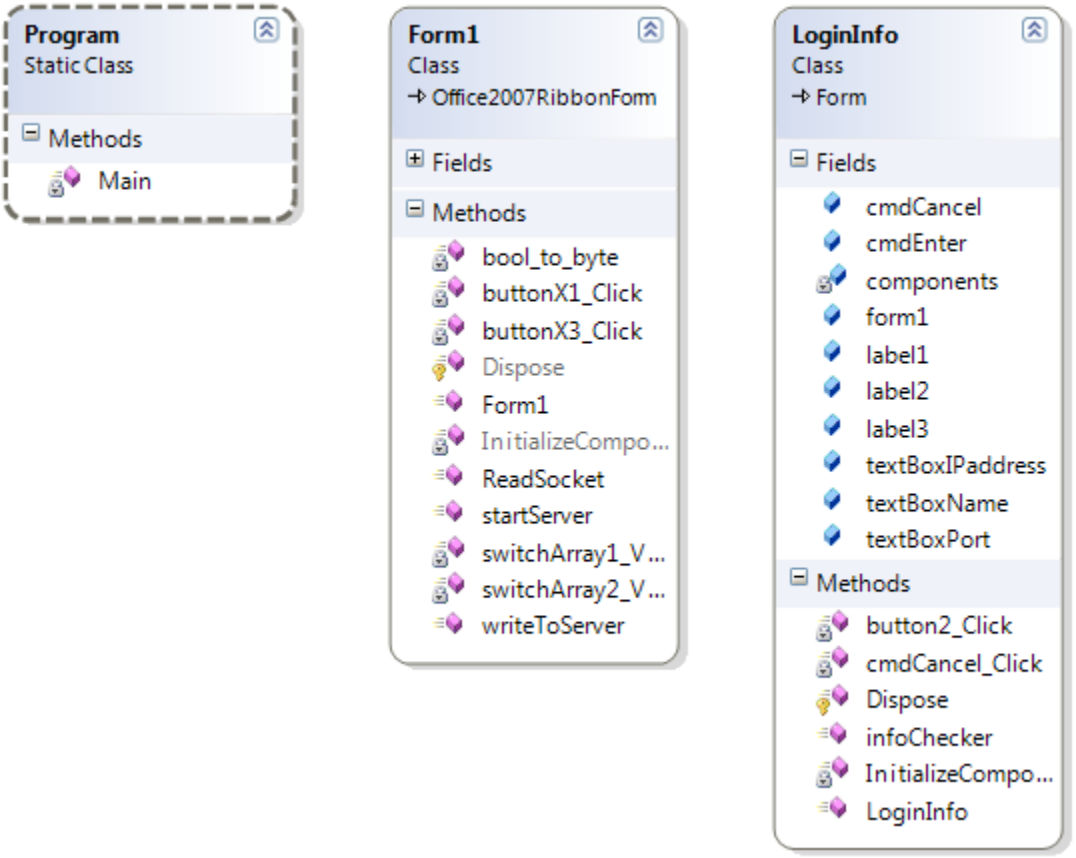


Figure 3 Class diagram for Software interface (Client)

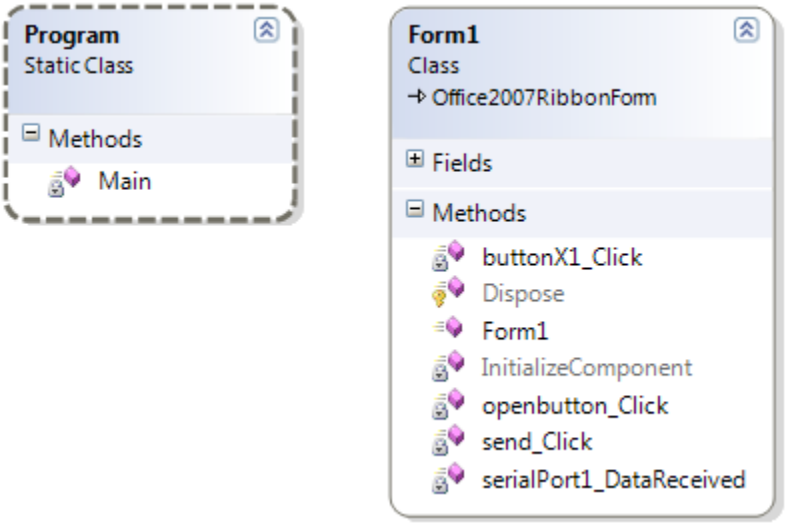


Figure 4 Class diagram of X-10 Transmitter

2.2 Interactivity scenario

2.2.1 X 10 Networks

a. X10 Code required send to the XM10 Module.

b. XM10 Module padding the required code to satisfy the X10 frame and send the frame over the powerline at the Zero Cross.

c. Every receiver in this network receives this frame and read the unit ID if match its unit ID then execute the order in the frame.

d. If the unit ID doesn't match its unit ID then nothing happens.

2.2.2 RS485 Network(Server)

a. Opening the Server Software.

b. Opening the Serial port (COM) connected to the Server Software.

c. Sending the Messages over the RS485 Network Cables.

d. Every receiver in the Network receives those messages and accepts its mine and rejects the others based upon its ID.

2.2.3 RS485 Network(Client)

a. Opening the client software.

b. Opening the serial port (COM) connected to the client software.

c. Sending its ID to the Server.

2.2.4 TCP/IP Network

a. Opening the Server Software.
b. Opening the Serial port (COM) connected to the Server Software.
c. Opening the Server connected to TCP/IP Network.
d. Getting the IPs from the Network.
e. Sending the TCP/IP Packets.
f. Converting TCP/IP Packets into RS485 Frames.
g. Sending the Frames to the clients.

2.2.5 FBUS Connection

a. Establishing the FBUS Connection between Microcontroller and the Phone.
b. Sending the Starting FBUs Message to start the FBUS communication.
c. Refreshing the FBUS Connection by sending the starting message again.
d. Reading the message reached to the phone from the GSM by the microcontroller.

Chapter III: Methodology

3.1 Power line communication and X10

3.1.1 Introduction

Power line communications (PLC) refers to the concept of transmitting information using the electrical power distribution network as a communication channel. This technology allows a flow of information through the same cabling that supplies electrical power. This novel idea of communication helps in bridging the gap existing between the electrical and communication network. It offers the prospect of being able to construct intelligent buildings, which contain many devices in a Local Area Network.

There are two main applications for power line communication - one for broadband Internet access to the home and the other for home and office networking. This work focuses on using power lines for home networking. Home networks typically use Ethernet or wireless devices. Ethernet provides high speed networking, but requires dedicated category 5 (CAT5) cabling which would need to be installed in the home. Wireless devices are now becoming more popular and work quite well. One major attraction of power line communication is the high availability of power outlets. "As long as there is a power socket, there is a connection to the network". The high node availability is why this technology has tremendous market potential. Power line communication technology has been slow to evolve because the lines were designed solely for the purpose of 50Hz main power distribution. But after development of X-10 protocol for convenient transmission over power line, it became easy.

3.1.2 Advantages and Disadvantages of PLC

3.1.2.1 Advantages of PLC

A. PLC integrates the transmission of communication signal and 50/60 Hz power signal through the same electric power cable.
B. The data link appears 'transparent' to the user. Although the devices are connected through the power line, consumers perceive that there is a "separated" link available for data communications.
C. Since the existing power lines are used for signal transmission, the initial heavy cost and investment for setting up a data communications system is avoided.

3.1.2.2 Disadvantages of PLC

A. Minimum-security levels: power lines do not necessarily provide a secure media.
B. Data attenuation: due to the presence of numerous elements on a power line network, data Attenuation is like issue.
C. High costs of residential appliances: the cost of a power line network modem is not always competitive with the cost of a standard modem used to connect to a phone line network.
D. lack of global standards: there are several different standards for power line

E. communication, and the development of a global standard for distributing data over existing in-home power line systems does not seem to be the trend of the international market
F. Noise: the greater amount of electrical noise on the line limits practical transmission speed (vacuum cleaners, light dimmers, kitchen appliances and drills are examples of noise sources that affect the performance of a power line-based home network).

3.1.3 The Challenge

Since the power line was devised for transmission of power at 50/60 Hz and at most 400 Hz, the use of this medium for data transmission (especially at high frequencies) presents some technically challenging problems. It is one of the most electrically contaminated environments, which makes it very hostile for transmission of data signals.

The channel is characterized by high noise levels and uncertain (or varying) levels of impedance and attenuation. In addition, the line offers limited bandwidth in comparison to cable or fiber-optic links.

Power line networks are usually made of a variety of conductor types and cross sections joined almost at random. Therefore a wide variety of characteristic impedances are encountered in the network .This imposes interesting difficulties in designing the filters for these communication networks.

3.2 X10 Protocol

3.2.1 What is the X10?

X10 is a remote-control system used for home automation. Its chief benefit is that it requires no additional wiring - it uses the electric power wiring in your house to send control signals.

This system was originally offered by BSR, a company that made audio equipment. Over the years, the product line spun off into a separate company, the local manifestation of which is called "X10 USA".

The X-10 technology is one of the oldest power line communications protocol and uses a Form of Amplitude Modulation (ASK Modulation) to transmit information. Although it was originally unidirectional (controller to controlled modules) recent developments indicate that some bi-directional products are being implemented. X-10 controllers send their signals over the power line to simple receivers that are used mainly to control lighting and other appliances.

Some controllers available today implement some sort of gateway between the power line and other medium such as RF and infrared.

A 120 kHz AM carrier, 0.5 watts signal is superimposed into the ac power line at zero Crossing to minimize the noise interference. Information is coded by way of bursts of this high frequency signal. To increase communications reliability, every bit of information is sent twice, requiring a full line cycle, which limits the transmission rate to 60 BPS (in a 60 Hz line). A normal X-10 command consists of two packets with a 3-cycle gap between packets. As mentioned, each packet contains two identical messages of 11 bits each, which yields a 48-cycle command length of about 0.8 second. This represents a poor bandwidth while the reliability of the transmission is severely compromised in a noisy environment. These are the main reasons why this technology has limited applications.

3.2.2 Transmission theory of X-10 signals

The X-10 communication is based on the "injection" of high-frequency signals (120 kHz) on the 220Vac network, representing binary signals (1 or 0). The signal is inserted immediately after the passage through the origin of the sine wave of 50Hz, with a maximum delay of 200 microseconds. This special feature is used by receivers to know when to listen to the line. The signal is sent through the electric energy network to the X-10 receivers connected to the network.

To allow the use in three-phase electrical networks, the 120 kHz signs are transmitted three times in each cycle, in moments that coincide with the passage of zero voltage of each of the phases. Thus, using its own couplers, it is possible to communicate with any device, regardless of the phase in which it is installed. In order to simplify the explanation, this fact will be omitted in the continuation of the text, referring only to the signals of a single phase.

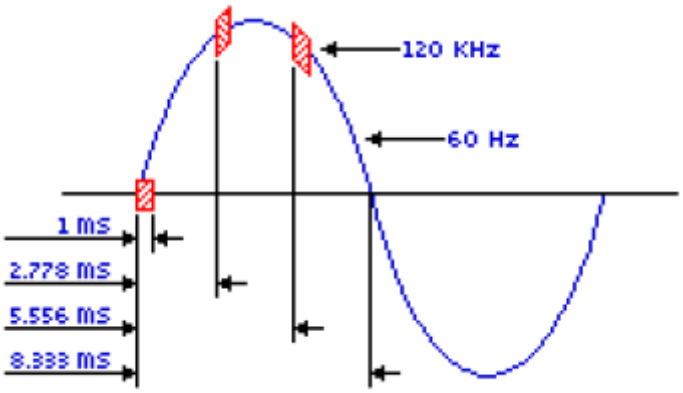


Figure 5 Sine wave with the injection of an X-10 signal

Since the means of distribution of energy is electrically very noisy, a policy in which a bit is never sent alone was adopted, and the bit is always sent together with its complement. In practice this means that whenever you want to send the bit 1, it corresponds to sending a 1 (120 kHz sign at the source) followed by a 0 (lack of signal). The sending of bit 0 corresponds to send a 0 (lack of signal) followed by a 1 (120 kHz frequency at the source). This is illustrated in Figure 3. This aims to minimize the probability of the electrical noise being confused with a valid signal. However, it has disadvantage of reducing the rate of transmission, which is thus restricted to a mere 50 bps (a bit is sent per cycle of the electricity network).

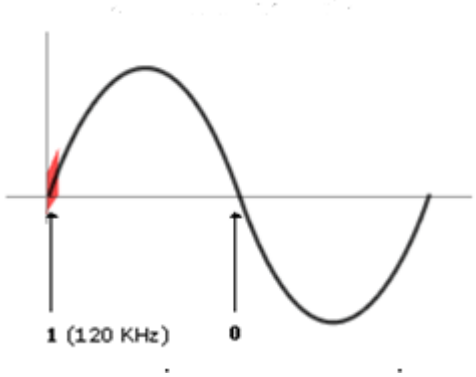


Figure 6 Sending of binary signals 1 and 0

A complete transmission of an X-10 command includes the transmission of four fields that "occupy" eleven cycles of the electric wave. The first field (2 cycles) represents the "Start Code" - sequence of bits (1 1 1 0). It should be pointed out that this is the exact sequence indicated and that the rule of each bit being followed by its complement is not confirmed. The following field, represented by 4 cycles, presents the home code and their respective supplements. Similarly 4 more bits are followed, which occupy 4 cycles that represent the device code or the code of the function. In order to distinguish this last field a bit is sent (and its respective supplement), which identifies whether the previous field refers to the number of a unit (bit = 0) or to the code of a function (bit = 1).

Each complete package must be sent in two groups (the first to indicate the device and the second the function to be executed) with a maximum of three cycles of the sine wave alternating between each group. The commands Dim and Bright are exceptions to this rule and should be continuously transmitted without a cycle interval between them.

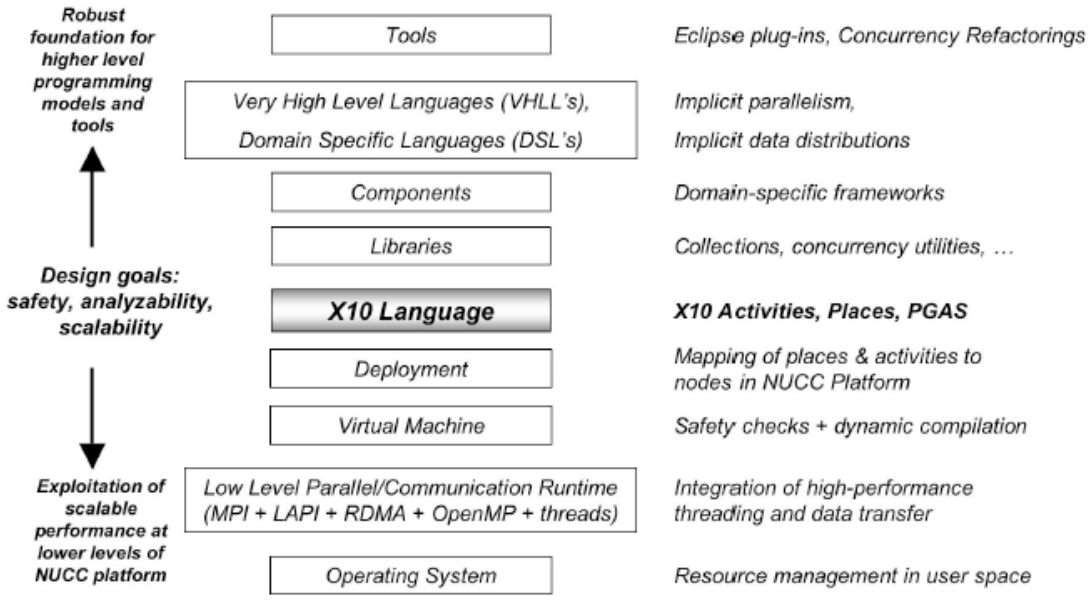


Figure 7 Standard X-10 Transmission Routine

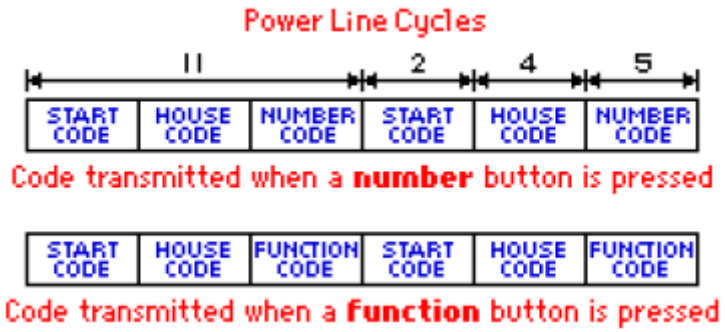


Figure 8 Standard Frame of X-10

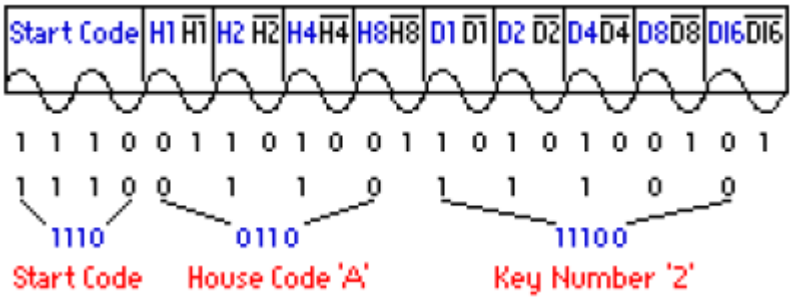


Figure 9 Example of X-10 Frame

An X-10 command usually includes two actions: activate a particular device (message code indicating device), and then send the function to be executed (message with the function code). Note that after a certain device is activated, it will remain active until another is located. While a device is active you can send it multiple commands. Sine wave with the injection of an X-10 signal sending of binary signals 1 and 0 Example of the transmission of an A2 ON command.

List of X-10 commands

House Codes					Unit/Function Codes						
	H8	H4	H2	H1		D8	D4	D2	D1	F	
A	0	1	1	0	1	0	1	1	0	0	
B	1	1	1	0	2	1	1	1	0	0	
C	0	0	1	0	3	0	0	1	0	0	
D	1	0	1	0	4	1	0	1	0	0	
E	0	0	0	1	5	0	0	0	1	0	
F	1	0	0	1	6	1	0	0	1	0	
G	0	1	0	1	7	0	1	0	1	0	
H	1	1	0	1	8	1	1	0	1	0	
I	0	1	1	1	9	0	1	1	1	0	
J	1	1	1	1	10	1	1	1	1	0	
K	0	0	1	1	11	0	0	1	1	0	
L	1	0	1	1	12	1	0	1	1	0	
M	0	0	0	0	13	0	0	0	0	0	
N	1	0	0	0	14	1	0	0	0	0	
O	0	1	0	0	15	0	1	0	0	0	
P	1	1	0	0	16	1	1	0	0	0	
					All Units Off	0	0	0	0	1	
					All Units On	0	0	0	1	1	
					On	0	0	1	0	1	
					Off	0	0	1	1	1	
					Dim	0	1	0	0	1	
					Bright	0	1	0	1	1	
					All Lights Off	0	1	1	0	1	
					Extended Code	0	1	1	1	1	
					Hail Request	1	0	0	0	1	Note 1
					Hail Acknowledge	1	0	0	1	1	
					Pre-Set Dim	1	0	1	X	1	Note 2
					Extended Data	1	1	0	0	1	Note 3
					Status is On	1	1	0	1	1	
					Status is Off	1	1	1	0	1	
					Status request	1	1	1	1	1	Note 4

Figure 10 X-10 Codes

Note 1: Hail Request is transmitted to see if there are any other X10 compatible transmitters within listening range.

Note 2: In a Pre-Set Dim function, the D1 bit represents the MSB of the level and the 4 House code bits represent the 4 least significant bits. No known X10 device responds to the Pre-Set Dim function.

Note 3: The Extended Data code is followed by eight-bit bytes which can be any data you might want to send (like temperature). There must be no delay between the Extended Data code and the actual data bytes, and no delay between data bytes.

Note 4: The X10 RF to AC Gateway model RR501 is a two-way module. If the RR501 is addressed by transmitting its House Code and Unit Code and then the STATUS

REQUEST is transmitted, the RR501 will respond by transmitting Status ON if it's turned on or Status OFF if it's off.

3.2.3 Why X10 Technology?

There are two main reasons why choosing the X10 protocol.

A. It is easy. Power line communication is patented. There is no need for 'control wires' or 'buses'. The modules simply plug in or replace existing switches, there is no complicated wiring. It is easily expandable to over 250 modules and it has the widest range of home control products available.
B. It is affordable. You can build a system for very little expense, and expand the system over time to suit your needs.

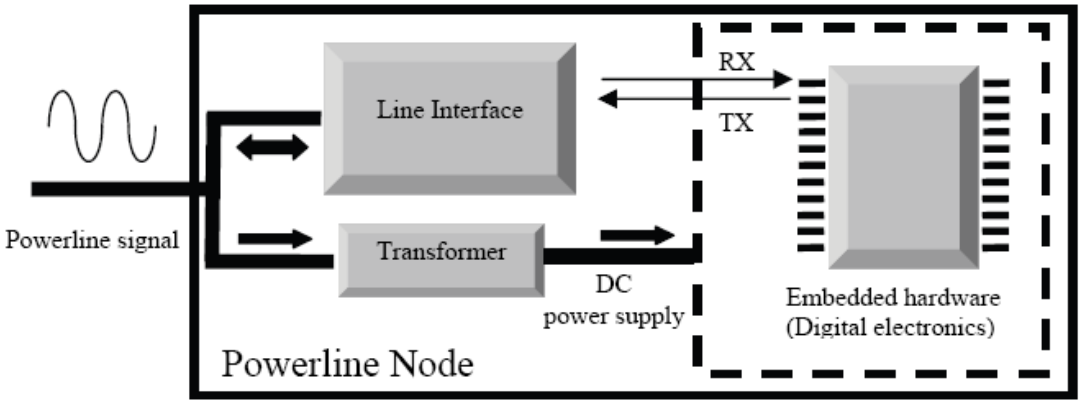


Figure 11 General structure of a power line node

3.2.4 What Are the Tradeoffs?

The following are factors in favor of X10 as a home automation system:

A. X10 equipment is inexpensive.
B. It requires no special wiring.
C. It is easy to set up and use.
D. Systems can be small or large - you can start with just a couple of pieces and grow if you like it.
E. X10 is easily placed under computer control.
F. A radio-controlled version is available and very compatible with the rest of the system.

The following are drawbacks of X10 as a home automation system:

A. X10 communication can be thwarted by other carrier-current devices, including wireless intercoms.
B. X10 signals can be degraded, damped, or stopped by power-conditioning equipment, including inexpensive "noise-suppressing" power strips, certain brands of computer power supplies, and my DAK bread maker. [There is a solution for this - a "choke".]

C. There is no guarantee that an X10 command will get to its destination. If you send a command to turn off the heater, the command might get zapped by line noise and never have a chance to turn off the heater.
D. The limitation of 16 house codes and 16 unit codes makes the address space a bit tight - there is no way to grow above 256 devices.
E. Most houses are wired with two separate 110 circuits. X10 signals sent from a control panel plugged into one outlet might not get to the lamp module plugged into the outlet across the room - if it is on the other "leg" of the 110. [There is a solution of this - a "signal bridge".]
F. It takes about a second to send an X10 command. While that command is being sent, you can't send another, or both commands will be lost.

3.2.5 X10 implementation

3.2.5.1 XM10 Module

Two-way PLC interface for OEM applications (xm10). The xm10 is a transmitter – receiver that plugs into a regular AC outlet and connects to the controller via a modular RJ 11 telephone jack. Alternatively, the xm10may be fitted inside the controller cabinet, connected to the 230 V AC supply before the power transformer. It provides an opto-coupled 50 HZ. Square wave, synchronized to the zero cross point of the AC line. The controller generates X-10 compatible codes synchronized to this zero crossing point. The two-way interface then couples the X-10 codes onto the AC line.

Figure 12 XM10 Module

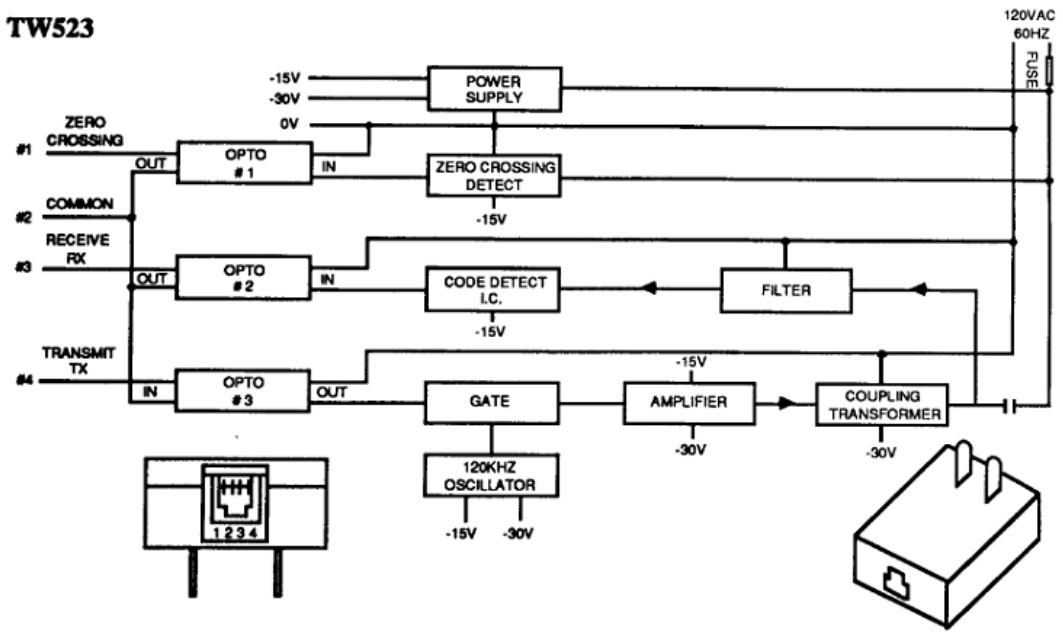


Figure 13 TW523 Connection Block diagram

3.2.5.2 X-10 Transmission Circuit

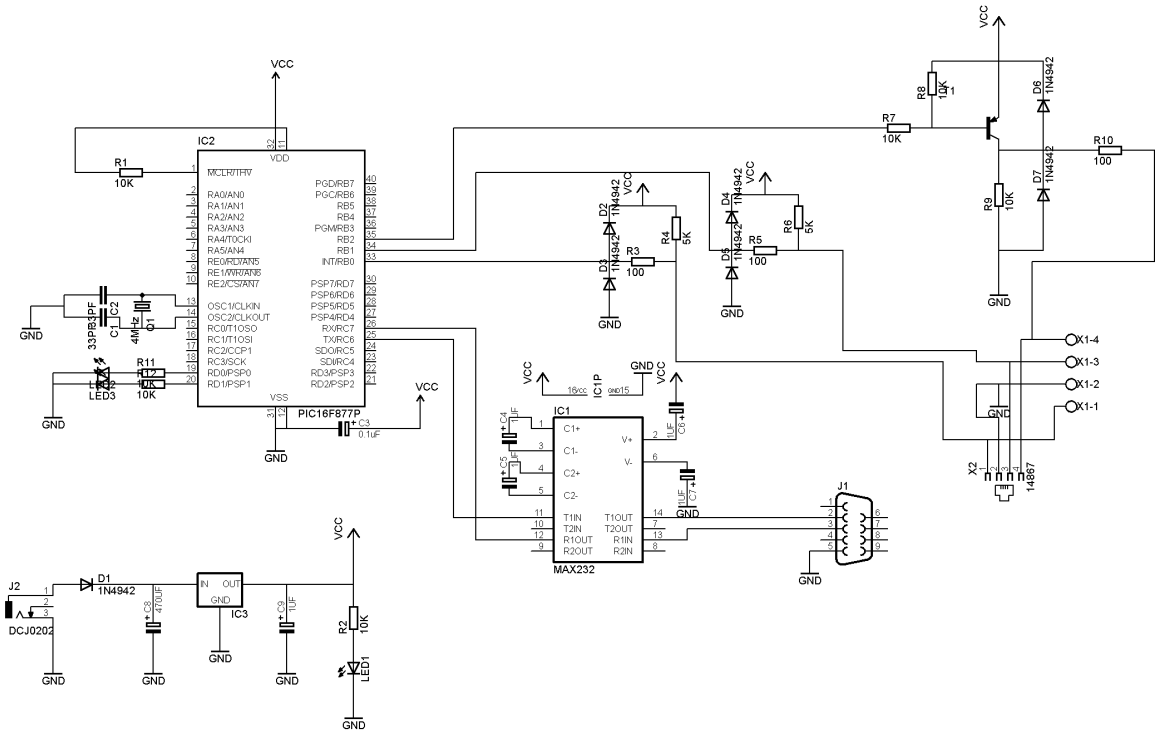


Figure 14 X-10 Transmitter Schematic

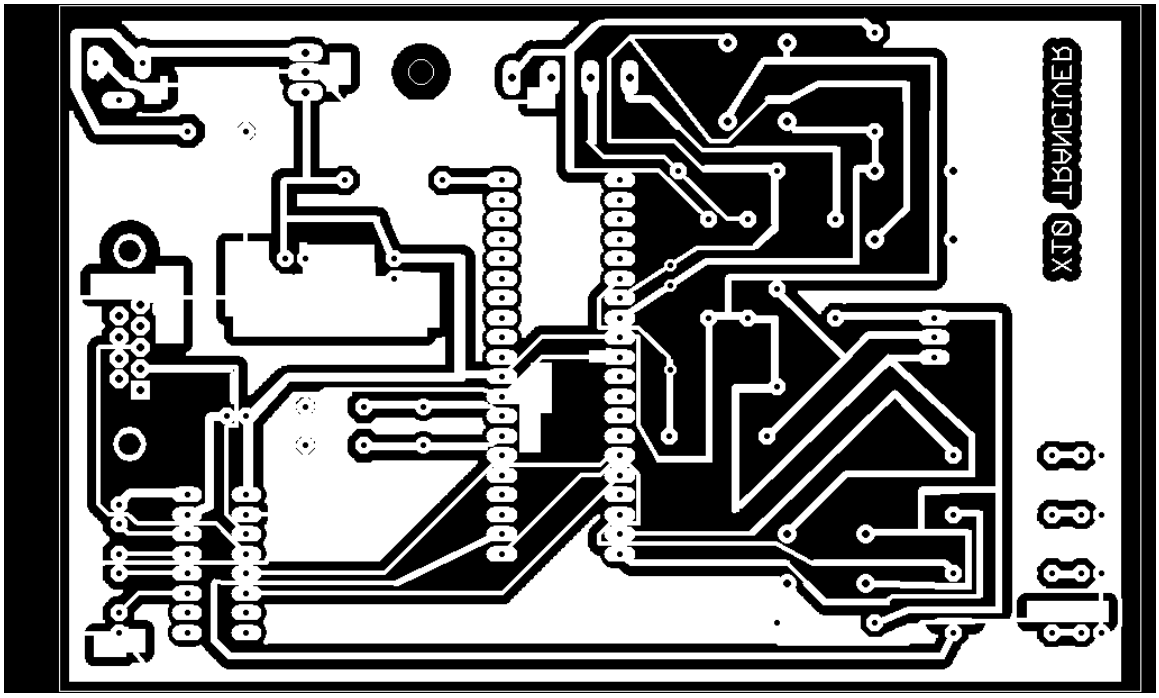


Figure 15 X10 Board Layout

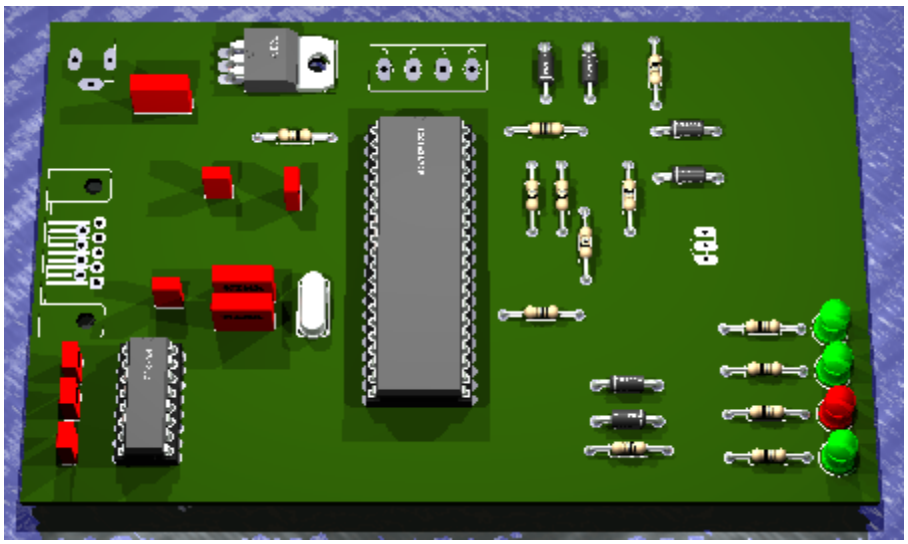


Figure 16 3D View for PCB

3.2.5.3 Zero Cross Detector

In X-10, information is timed with the zero-crossings of the AC power. A zero-crossing detector is easily created by using the external interrupt on the RB0 pin and just one external component, a resistor, to limit the current into the PICmicro MCU (see Figure 3). In India, the peak line voltage is 230V. If we select a resistor of 6 M Ω, I peak = 230V/6 M Ω=38 μA, which is well within the current capacity of a PICmicro MCU I/O pin. Input protection diodes (designed into the PICmicro MCU I/O pins) clamp any voltage higher than VDD or lower than VSS. Therefore, when the AC voltage is in the negative half of its cycle, the RB0 pin will be clamped to VSS - 0.6V. This will be interpreted as a logic zero. When the AC voltage rises above the input threshold, the logical value will become a '1'. In this application, RB0 is configured for external interrupts, and the input buffer is a Schmitt trigger. This makes the input threshold 0.8 VDD = 4V on a rising edge and 0.2 VDD = 1V on a falling edge.

Upon each interrupt, the Interrupt Edge Select bit within the OPTION_REG register is toggled, so that an interrupt occurs on every zero-crossing.

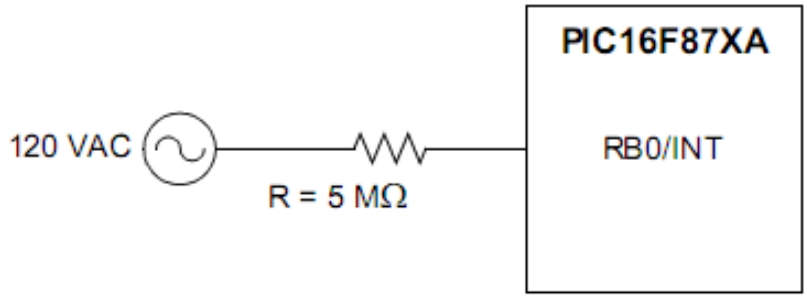


Figure 17 Zero Cross Detection Circuit

3.2.5.4 120 KHZ Carrier Generator

X-10 uses 120 kHz modulation to transmit information over 50 Hz power lines. It is possible to generate the 120 kHz carrier with an external oscillator circuit. A single I/O pin would be used to enable or disable the oscillator circuit output. However, an external oscillator circuit can be avoided by using one of the PICmicro MCU's CCP modules. The CCP1 module is used in PWM mode to produce a 120 kHz square-wave with a duty cycle of 50%. After initialization, CCP1 is continuously enabled, and the TRISC bit for the pin is used to gate the PWM output. When the TRISC bit is set, the pin is an input and the 120 kHz signal is not presented to the pin. When the TRISC bit is clear, the pin becomes an output and the 120 kHz signal is coupled to the AC power line through a transistor amplifier and capacitor, as depicted in Figure.

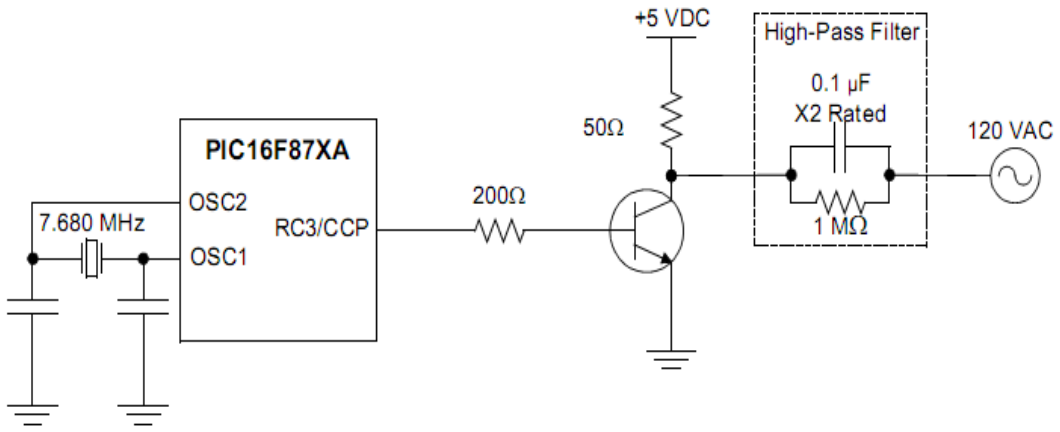


Figure 18 120 KHZ Carrier Generation Circuit

Since the impedance of a capacitor is Zc = 1/ (2*π *f*C), a 0.1 µF capacitor presents a low impedance to the 120 kHz carrier frequency, but a high impedance to the 50 Hz power line frequency. This high-pass filter allows the 120 kHz signal to be safely coupled to the 50 Hz power line, and it doubles as the first stage of the 120 kHz carrier detector. To be compatible with other X-10 receivers, the maximum delay from the zero crossing to the beginning of the X-10 envelope should be about 300 µs. Since the zero crossing detectors has a maximum delay of approximately 64 µs, the firmware must take less than 236 µs after detection of the zero crossing to begin transmission of the 120 kHz envelope.

3.2.5.5 X-10 Transmit Software

As explained in the previous sections, the X-10 Frame consist of Start code (1110), house code and number code , also for doing this in the software the message sent firstly to MAX232(use RS232 Protocol)that in rule send data to Microcontroller to Packed and prepare it to transmit over the power line.

First, choose the id of the serial port that device connected to it that is done using the following interface

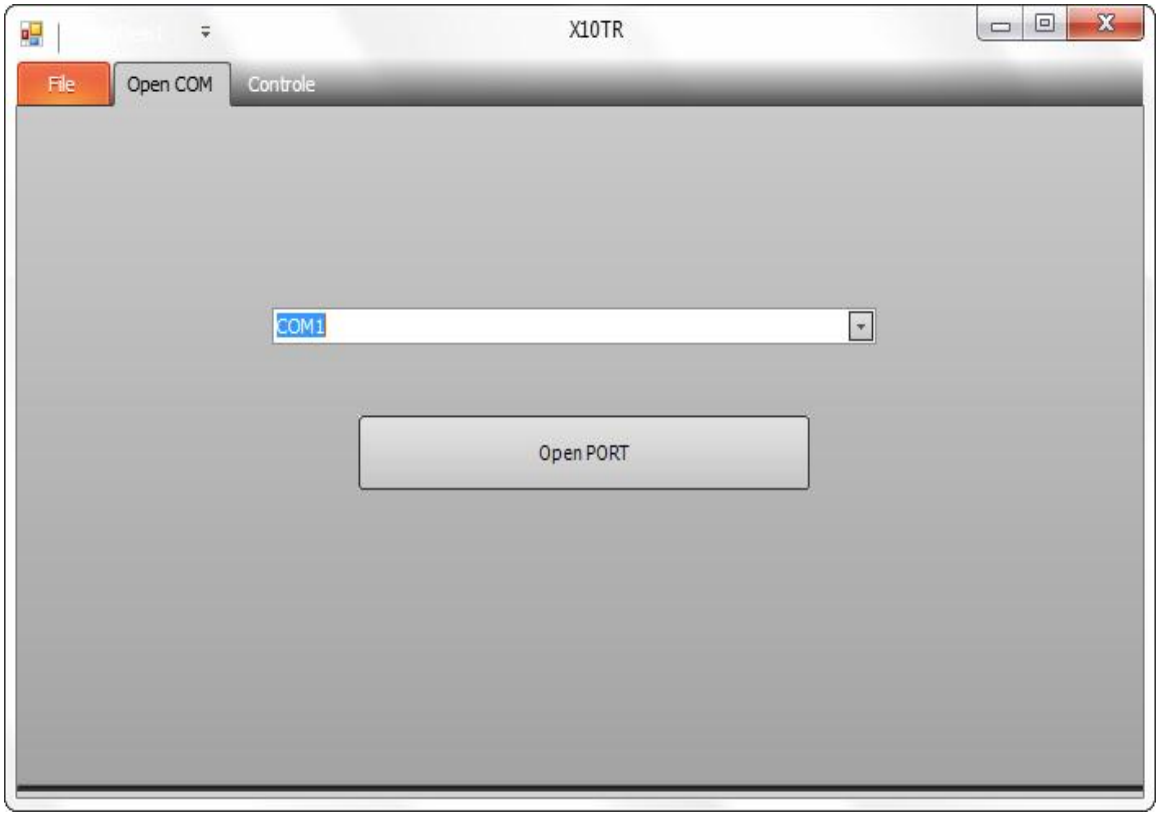


Figure 19 Open Com Interface

Second, choosing the house id and the unit id and then write the command code as in the following interface

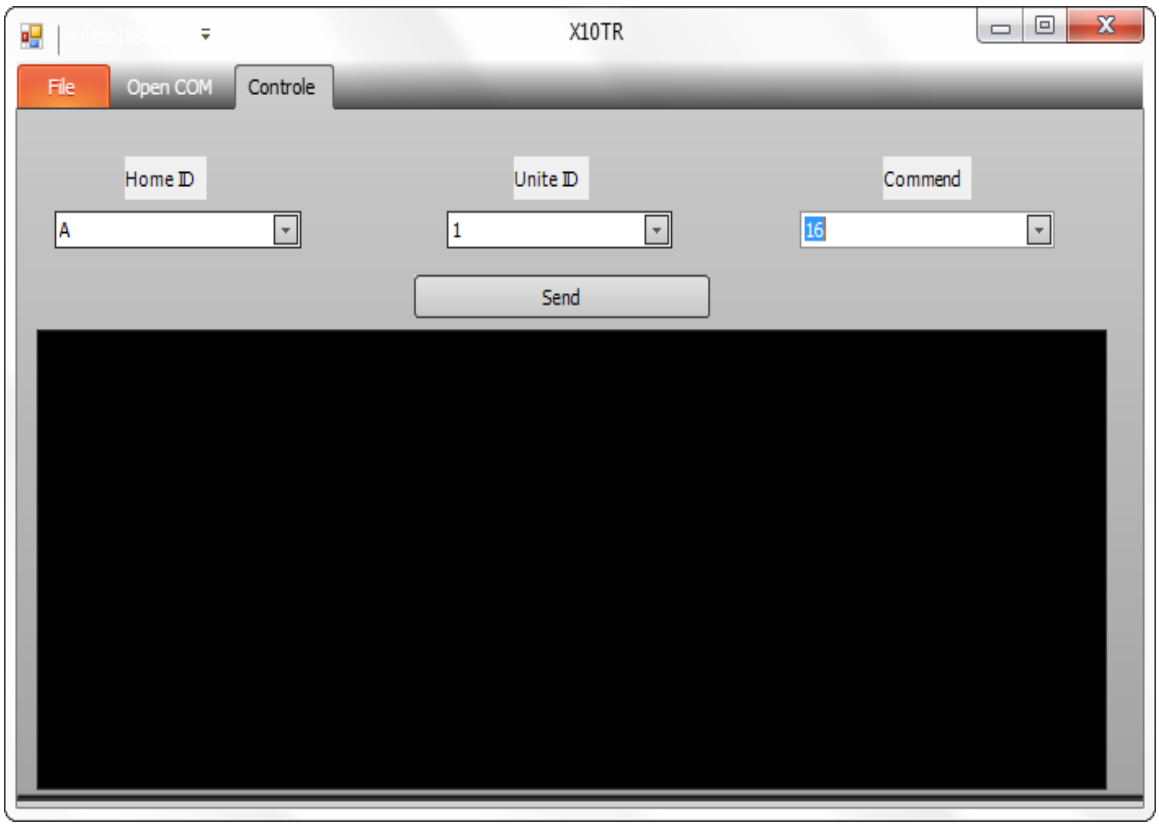


Figure 20 Send data Interface

A. Send data method

```
private void send_Click(object sender, EventArgs e)
        {
            string s = comboBoxEx1.Text + comboBoxEx2.Text;
            t.Text += "\n\r";
            t.Text += "Send :" + s + "\n\r";
            port.Write(s);
        }

        private void buttonX1_Click(object sender, EventArgs e)
        {
            t.Text = "";
        }
```

Code 1 Send data method

B. Receive data method

```
private void serialPort1_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {

            string m;
            m = port.ReadExisting();
            t.Text += "\n\r";
            t.Text +="Recive :"+m+"\n\r";

        }
```

Code 2 Receive data method

3.3 RS485 Protocol

3.3.1 What is RS485 Protocol?

RS-485 is a telecommunications standard for binary serial communications between devices. It is the protocol or specifications that need to be followed to allow devices that implement this standard to speak to each other. This protocol is an updated version of the original serial protocol known as RS-232. While the original RS-232 standard allowed for the connection of two devices through a serial link, RS-485 allows for serial connections between more than 2 devices on a networked system.

A RS-485 compliant network is a multi-point communications network. The RS-485 standard specifies up to 32 drivers and 32 receivers on a single (2-wire) bus. New technology has since introduced "automatic" repeaters and high-impedance drivers and receivers such that the number of drivers and receivers can be extended to hundreds of nodes on a network. RS-485 drivers are now even able to withstand bus contention problems and bus fault conditions.

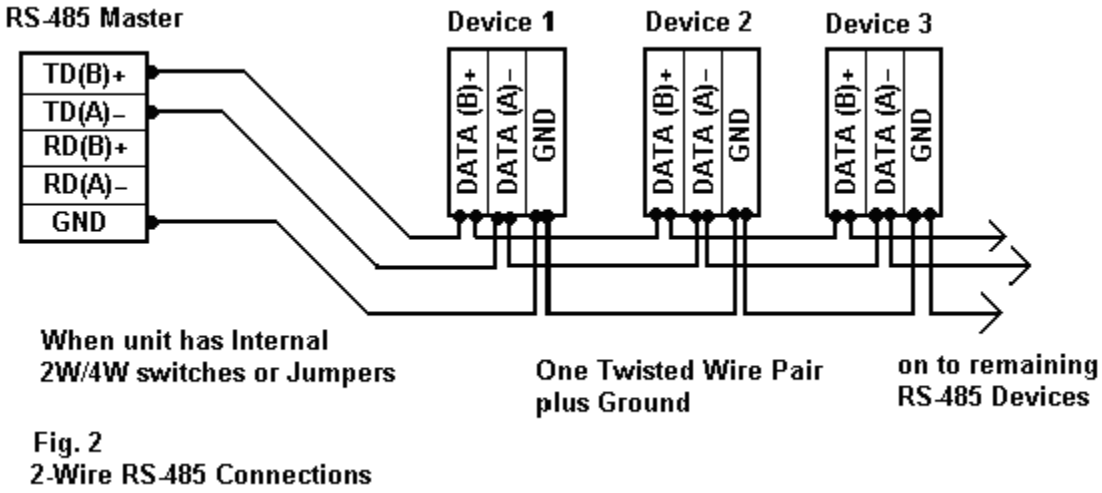


Figure 21 RS485 Network architecture

A RS-485 network can be constructed as either a balanced 2 wire system or a 4 wire system. If a RS-485 network is constructed as a 2 wire system, then all of the nodes will have equal ranking. A RS-485 network constructed as a 4 wire system has one node designated as the master and the remaining nodes are designated as slaves. Communication in such a system is only between master and slaves and never between slaves. This approach simplifies the software protocol that needs to be used at the cost of increasing the complexity of the wiring system slightly.

3.3.2 The RS485 Advantages:

A. RS485 allows multiple devices (up to 32) to communicate at half-duplex on a single pair of wires, plus a ground wire (more on that later),
B. At distances up to 1200 meters (4000 feet).
C. Both the length of the network and the number of nodes can easily be extended using a variety of repeater products on the market.
D. The properties of differential signals provide high noise immunity and long distance capabilities.

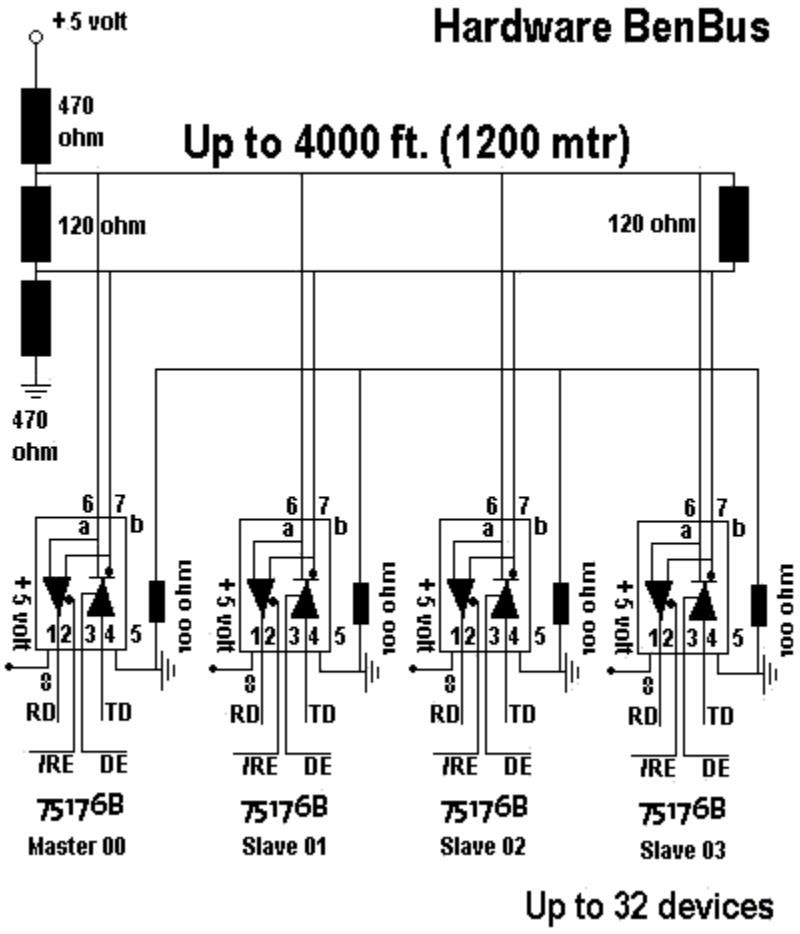


Figure 22 Devices connection in the Network

3.3.3 How does the hardware work?

Data is transmitted differentially on two wires twisted together, referred to as a "twisted pair." The properties of differential signals provide high noise immunity and long distance capabilities. It is like the following figure.

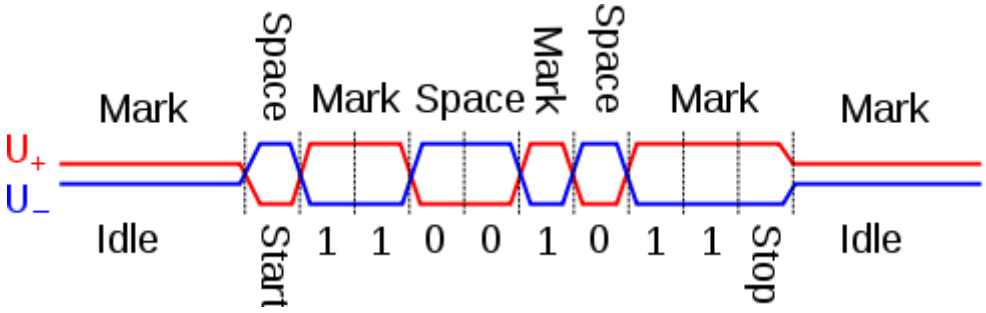


Figure 23 RS485 Signals

A 485 network can be configured two ways, "two-wire" or "four-wire." In a "two-wire" network the transmitter and receiver of each device are connected to a twisted pair. "Four-wire" networks have one master port with the transmitter connected to each of the "slave" receivers on one twisted pair. The "slave" transmitters are all connected to the "master" receiver on a second twisted pair.

In either configuration, devices are addressable, allowing each node to be communicated to independently. Only one device can drive the line at a time, so drivers must be put into a high-impedance mode (tri-state) when they are not in use. Some RS-485 hardware handles this automatically. In other cases, the 485 device software must use a control line to handle the driver. (If your 485 device is controlled through an RS-232 serial port, this is typically done with the RTS handshake line.)

A consequence of tri-stating the drivers are a delay between the ends of a transmission and when the driver is tri-stated. This turn-around delay is an important part of a two-wire network because during that time no other transmissions can occur (not the case in a four-wire configuration). An ideal delay is the length of one character at the current baud rate (i.e. 1 ms at 9600 baud).

3.3.3.1 Two-wire or four-wire?

Two-wire 485 networks have the advantage of lower wiring costs and the ability for nodes to talk amongst themselves. On the downside, two-wire mode is limited to half-duplex and requires attention to turn-around delay. Four-wire networks allow full-duplex operation, but are limited to master-slave situations (i.e. “master" node requests information from individual "slave" nodes). "Slave" nodes cannot communicate with each other. Remember when ordering your cable, "two-wire" is really two wires + ground, and "four-wire" is really four wires + ground.

3.3.4 How does the software work?

RS485 software handles addressing, turn-around delay, and possibly the driver tri-state features of 485. Determine before any purchase whether your software handles these features. Remember, too much or too little turn-around delay can cause troubleshooting fits, and delay should be a function of baud rate. If you're writing your own software or using software written for an RS-232 application, be certain that provisions are made for driver tri-state control. Luckily, there are usually hardware alternatives for controlling driver tri-stating. Contact B&B Technical Support for further details.

3.3.4.1 The EIA RS485 Specification

The EIA RS485 Specification labels the data wires "A" and "B", but many manufacturers label their wires "+" and "-". In our experience, the "-" wire should be connected to the "A" line, and the "+" wire to the "B" line. Reversing the polarity will not damage a 485 device, but it will not communicate. This said, the rest is easy: always connect A to A and B to B.

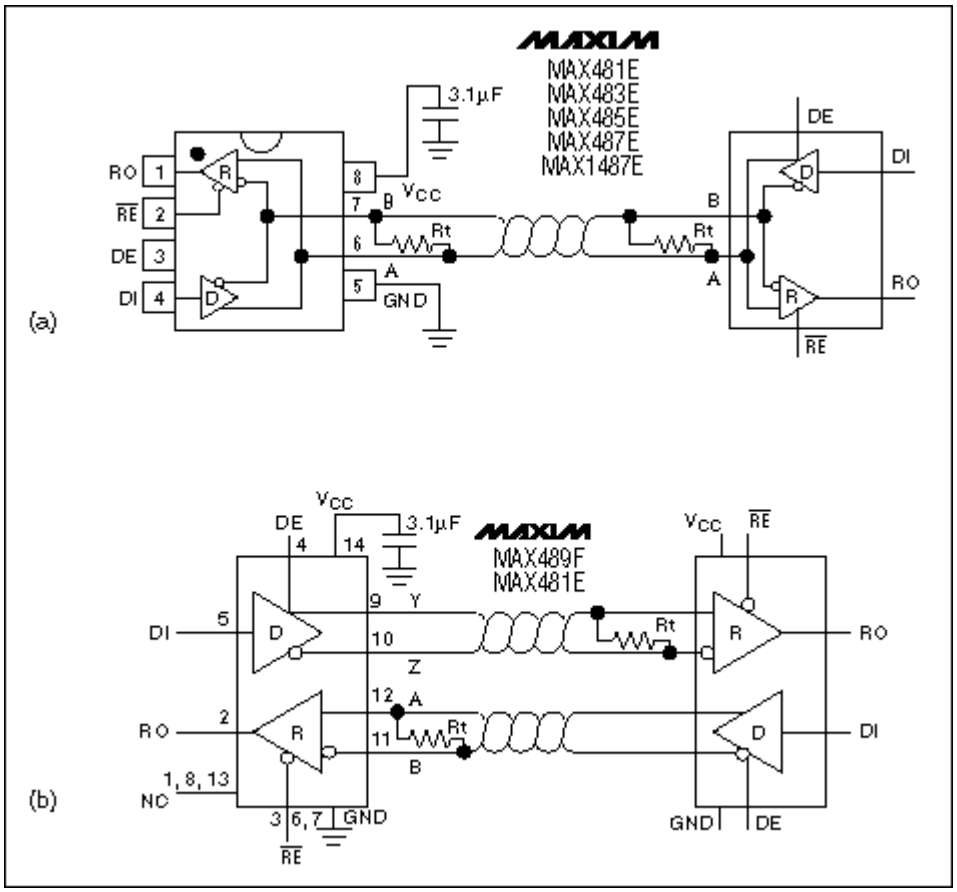


Figure 24 Max485 Connection

3.3.5 Important to communicate

Signal ground, don't forget it. While a differential signal does not require a signal ground to communicate, the ground wire serves an important purpose. Over a distance of hundreds or thousands of feet there can be very significant differences in the voltage level of "ground." RS-485 networks can typically maintain correct data with a difference of -7 to +12 Volts. If the grounds differ more than that amount, data will be lost and often the port itself will be damaged. The function of the signal ground wire is to tie the signal ground of each of the nodes to one common ground. However, if the differences in signal grounds are too great, further attention is necessary.

3.3.6 RS485 Hardware implementation

Using microcontroller is the easy way for our devices to support RS485 as most of the microcontroller support the serial communication (RS232) which is the bias of the RS485 .also all the microcontroller compilers support the RS232 protocol so it is easy to implement the RS485 within it.

3.3.6.1 Microcontroller:

We select PIC microcontroller to be used in our project as we previously give the causes for using it.

The circuit of the project as the following

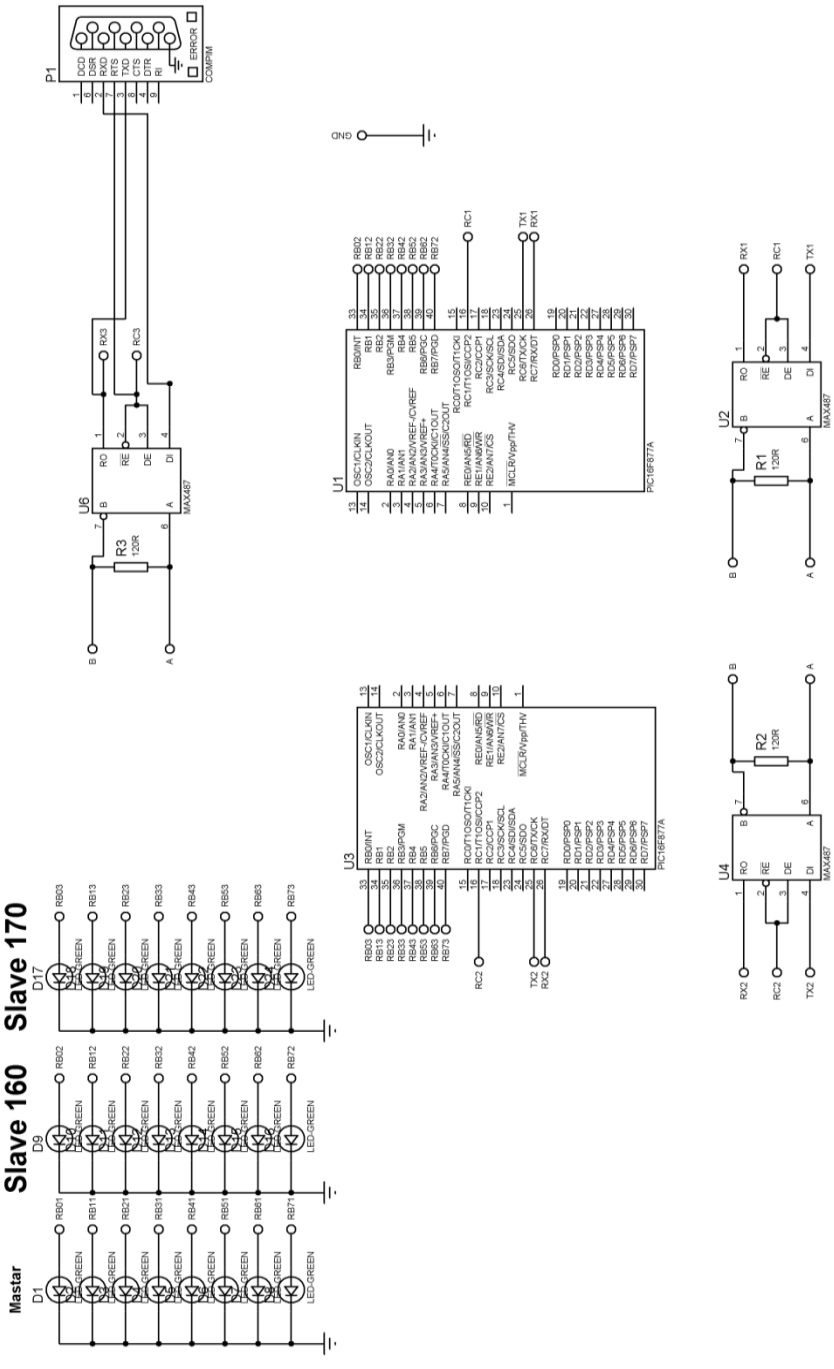


Figure 25 Total RS485 Network Circuit

Microcontroller circuit elements

A. RS232 to RS485 converter

Our Devices will be connected to PC using serial cable but it is not use RS232 protocol it use RS485 protocol so we need to convert RS232 interface to RS485 interface. It is like the following

a. Schematic

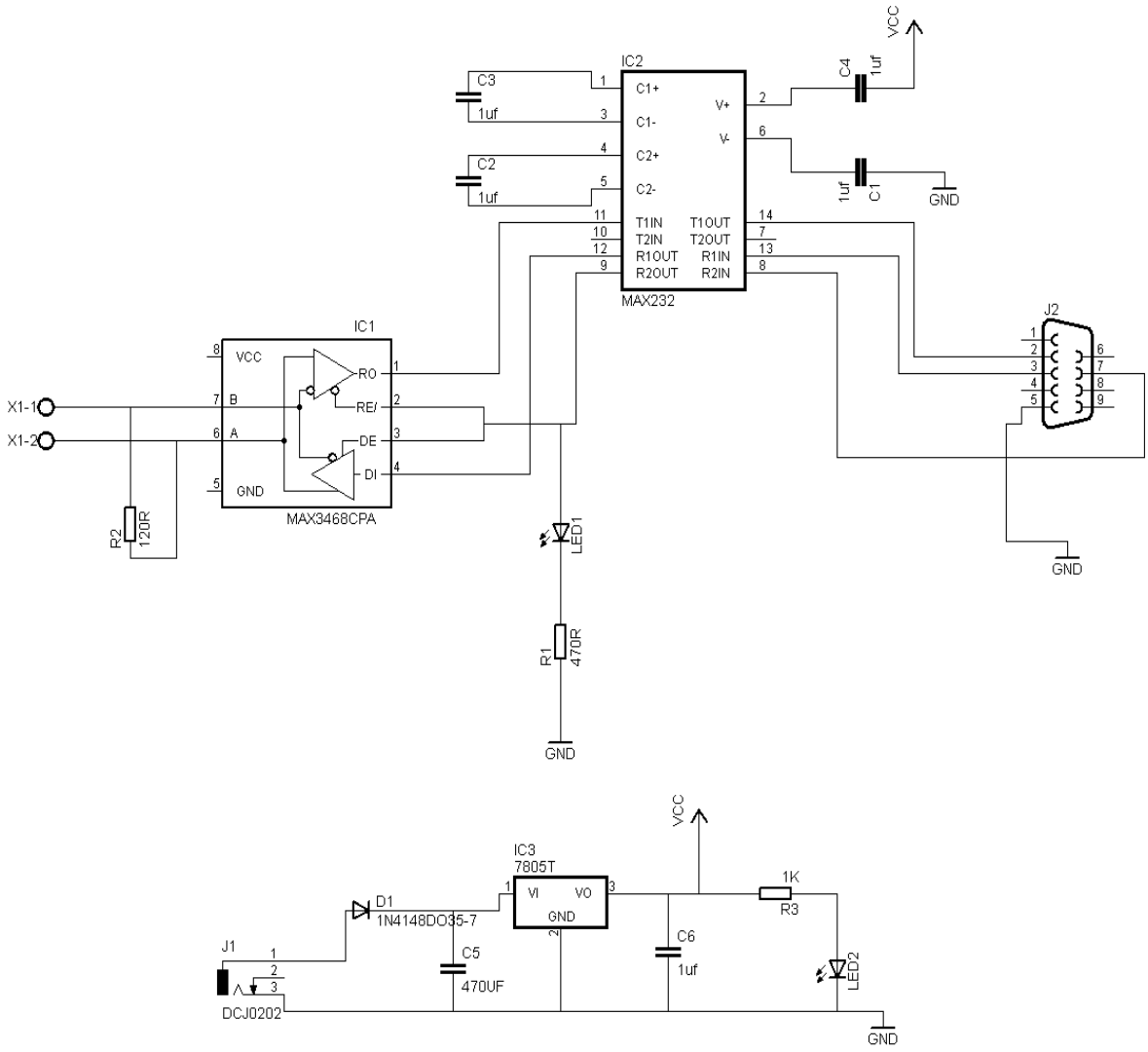


Figure 26 RS232 to RS485 Converter Schematic

b. PCB

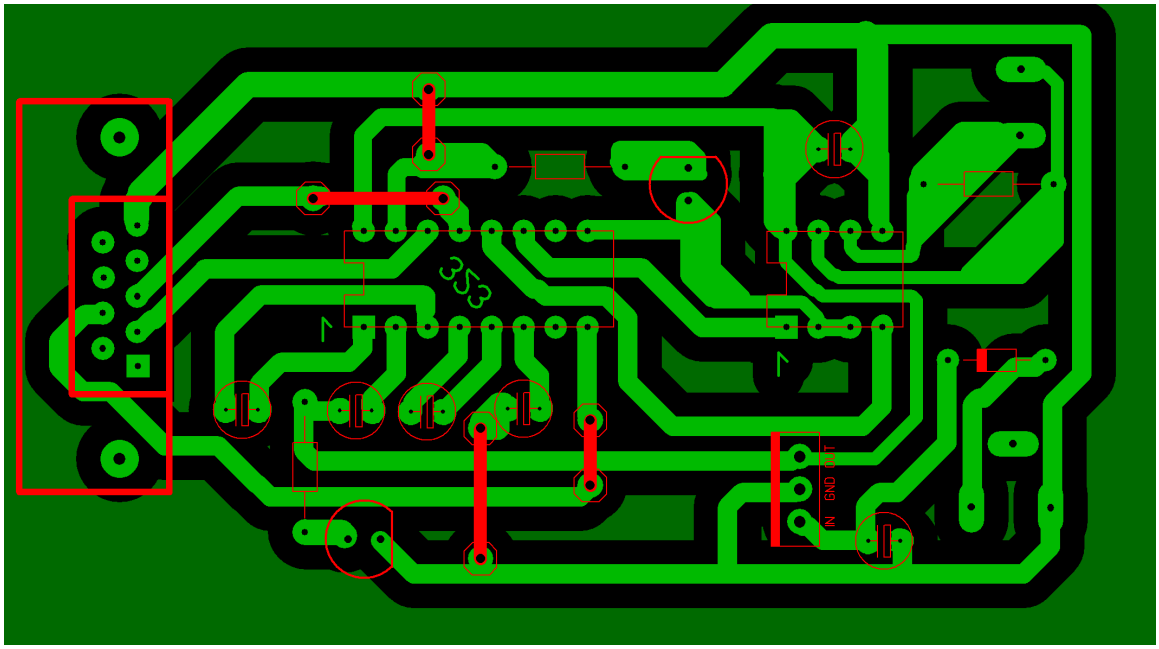


Figure 27 RS232 to 485 Converter PCB

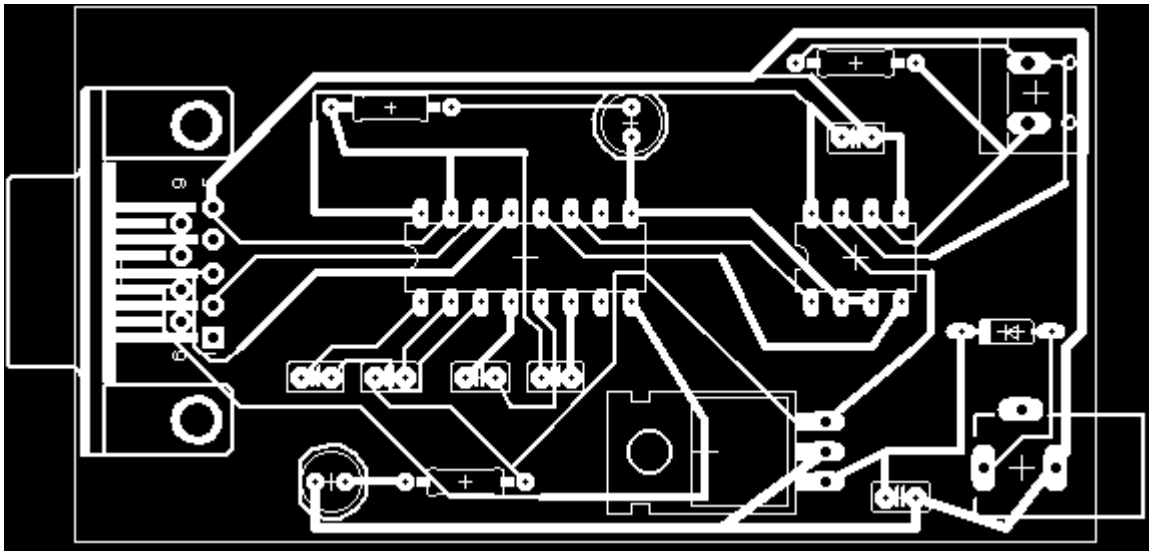


Figure 28 RS232 to 485 Converter PCB 2

c. final PCB

Figure 29 RS232 to 485 Converter bottom view

Figure 30 RS232 to 485 Converter Top view

3.3.6.2 RS485 Receiver

The following circuit will receive the data from the computer over a pair of wire as explained in the previous sections. Every RS485 receiver circuit has a microcontroller with a unique address in which they can control in up to 3 bytes of data which means that by only one microcontroller we can drive up to 24 devices. The circuit receives data from the computer check for their ID address if it is the check the entire message and calculates the CRC if it is equal to that transmitted by the computer it means that the message was correct so the micro will appears the message at its outputs

a. RS485 receiver schematic

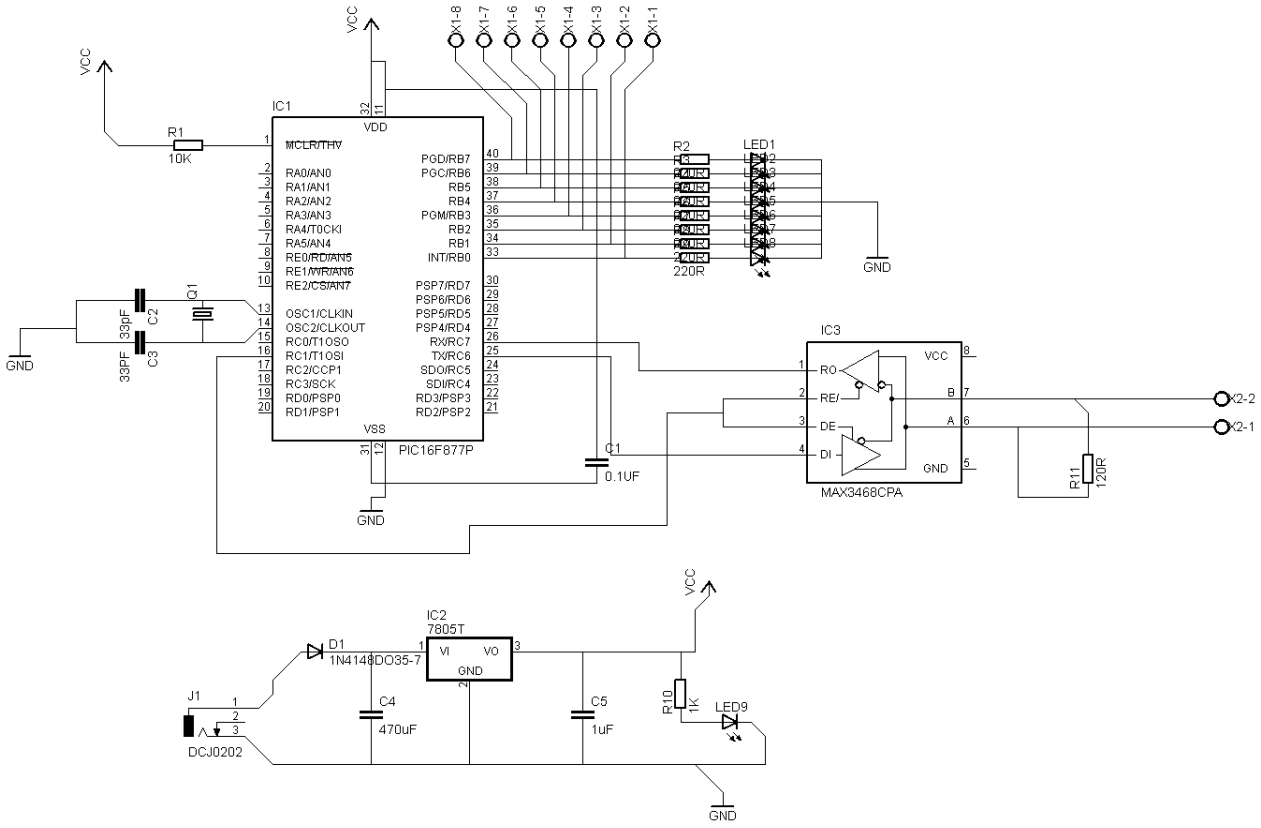


Figure 31 RS485 receiver schematic

b. RS485 receiver PCB

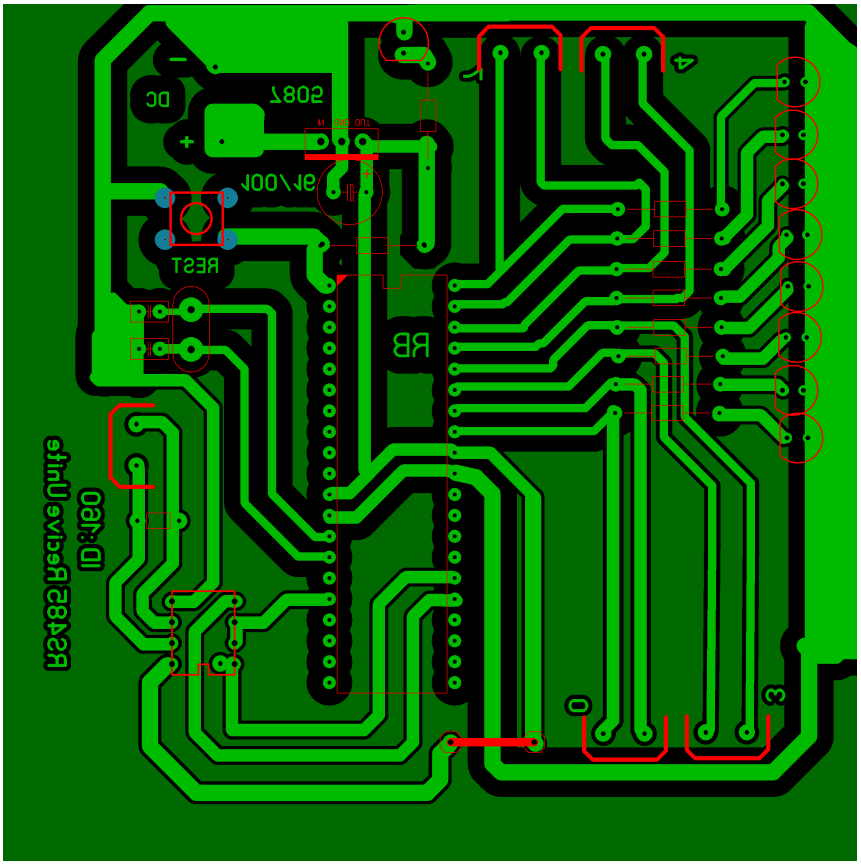


Figure 32 RS485 receiver PCB

c. Final PCB

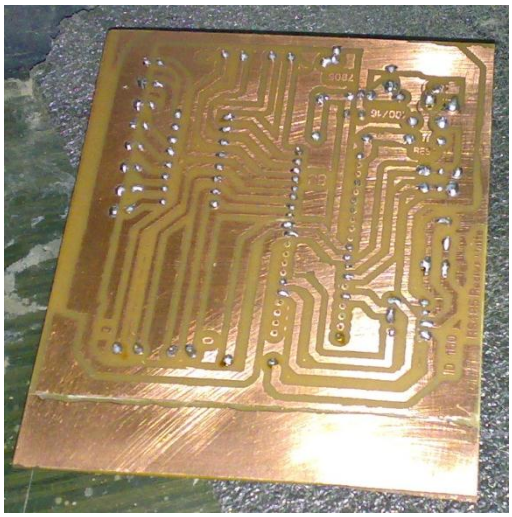


Figure 33 RS485 receiver PCB 2

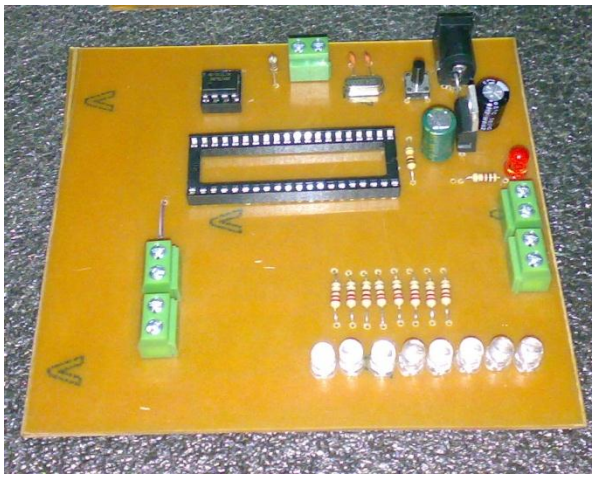


Figure 34 RS485 receiver PCB 3

3.3.6.3 RS485 Transmitter

The transmitter circuit is used to transmit data from the device to our PC to display it .also every transmitter circuit is addressable so we can receive data from devices up to 32 devices and the data size to be transmitted is up to 3 bytes which is very good to transmit what we want to transmit.

In the transmitter circuit it will be used to transmit data from:

1. temperature sensor
2. Light sensor (used in security)

a. RS485 transmitter schematic

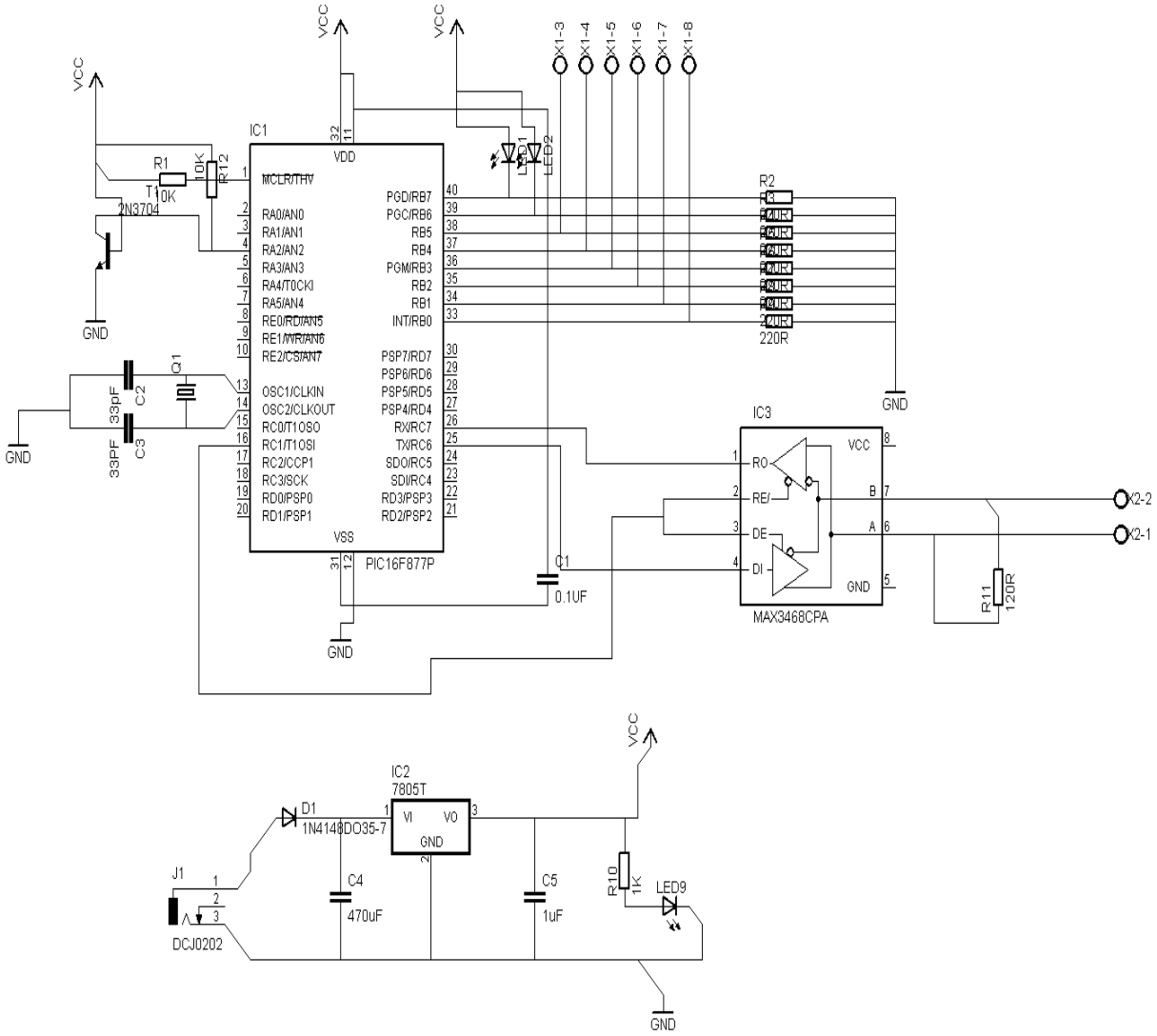


Figure 35 RS485 transmitter schematic

b. RS485 transmitter PCB

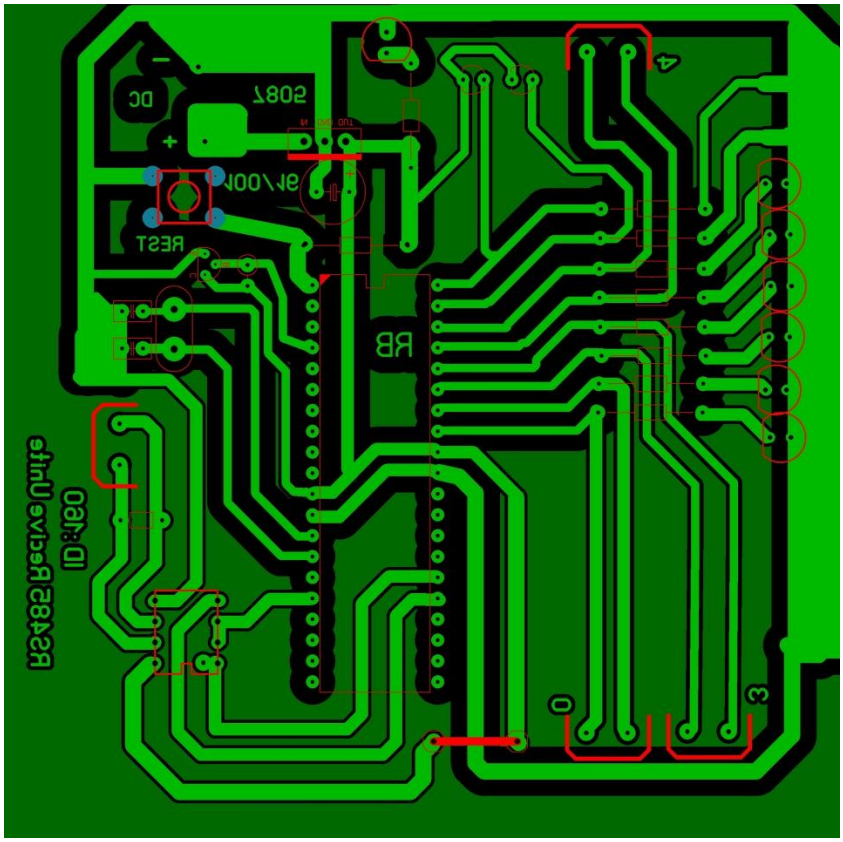


Figure 36 RS485 transmitter PCB

3.3.7 Microcontroller Programming

The used microcontroller is the PIC microcontroller it can be programmed using Assamply,C,Basic,and Pascal .also there are more than one compiler like MicroC,CCS and High-Tec. and each differ from other by its offered library to make programming easily and efficient.

In this project the used compiler for this part of it is the MicroC compiler it is very easy to be used also it offer a RS485 library which contain Master and slave classes in Appendix there is more information about this library.

3.3.7.1 Data Send and received format

start byte	address byte	number of data bytes	first data byte	second data byte	third data byte	redundancy check (CRC)	End Byte

a. Start byte
   Is the first byte in the packet which is always is equal 0X96

b. Address byte
   The address of the device this byte can take the value from 0 to 255 but it can't take the value 50 decimal which is used for broadcast

c. Number of data bytes
   This byte indicate the number of data bytes being transmitted from the slave and the number of data being transmitted from the M aster plus 128

d. First data byte
   This is the first byte of data being transmitted

e. Second data byte
   This is the second byte of data to be transmitted

f. Third data byte
   This is the third byte to be transmitted

g. Redundancy check (CRC) byte
   The algorithm for calculating the CRC is also given as;

   CRC = NOT ($aa XOR $bb XOR $dd [XOR $dd XOR $dd])

   Where;

   $aa = one byte address

$bb = one byte showing number of data bytes (slave) and 128+number of bytes (master)

$dd = one to three data bytes (depending on what was put in RS485 send command)

$cc = cyclic redundancy check (CRC) byte

I.e. XOR $aa, $bb and all $dd bytes then invert all the bits in the answer then if the answer is $96 or $A9 add one

3.3.7.2 Some important configuration for RS485

a. Buad Rate

The baud rate can be configured in both master and slave devices but it must be the same for the data to be received correctly in this project the slave is a PIC microcontroller and the Maser is the PC and the used buad rate is 9600 bps

b. RTS (Request to Send) bin

Control RS-485 Transmit/Receive operation mode this bin must be connected to the RTS bin of the DB9 connector via the MAX232 IC to control the transmit and receive operation from and to the PC this also will be configured in the software program

3.3.7.3 Complete Microcontroller codes

For the complete microcontroller code see appendix G

3.3.8 Software Program

In this project the programming language used is the C# because its facility in interfacing the hardware with the software .the start with the interfacing with hardware using serial cable with the protocol RS232 but in this project the used protocol is the RS485 so the software must be to written to send RS485 packet instead of RS232 packet .in the following is the work done by us to develop a new class that can send RS485 packets

3.3.8.1 RS485 Class

In the previous section (Microcontroller Programming) the RS485 packet details was introduced .and now the computer is the master so our software must send the packet with the same specification and also has to calculate the CRC to make our slave device which is the microcontroller to receive data correctly so we develop the following class to take data and format it with the same specification to be transmitted

3.3.8.1.1 RS485 Variables

```
byte add,data,bytes_no=129, crc,start_byte=150,end_byte=169;

byte[] sbuffer;
```

Code 3 RS485 Variables

Data: is the data to be send to the specific added.

bytes_no: is the number of data bytes to be transmitted +128 (for now it is only 1 byte).

Crc: redundancy check byte (check for error in the following we will explain how)

start_byte: the start byte which used to indicate to the receiver the start of the message it is constant for now and equal to 150 in decimal

End byte: the end byte which used to indicate to the receiver the end of the message it is constant for now and equal to 169 in decimal.

Sbuffer: the packet to be transmitted.

3.3.8.1.2 RS485 Methods

a. Set_add Method

This method is used to specify the address to the new created object of our class. It is like the following

```
//=====================================================================
        // set add Method
//=====================================================================
        public void set_add(byte d)
        {
            this.add = d;
        }
 //=====================================================================
```

Code 4 set_add Method

b. Set_data Method

This method is used to specify the data to the new created object with the specific add of our class. It is like the following

```
//=====================================================================
        // set data Method
//=====================================================================
        public void  set_data(byte da)
        {
            this.data = da;
        }
  //=====================================================================
```

Code 5 Set_data Method

c. Take Not Method

This method is apart from the method that used to calculate the CRC. This method take an integer convert it to its binary representation then invert each bit and return the equivalent integer

```
//============================================================
        //Not Method
//============================================================
        public double takeNOT(int b)
        {
            int t, j = 7;
            string g = string.Empty;
            double k = 0;
            for (t = 128; t > 0; t = t / 2)
            {
                if ((b & t) != 0) g += "0";
                if ((b & t) == 0) g += "1";
            }
            chars = g.ToCharArray();
            foreach (char c in chars)
            {
                if (c == '1')
                {
                    k += Math.Pow(2, j);
                }
                j--;

            }
            return k;
        }
    //============================================================
```

Code 6 Take Not Method

d. get_data Method

This method responsible for calculating CRC for the current ID address and data of the object that called for it. The CRC calculation algorithm has been explained in the previous section (2.1-Data Send and received format).

```
//=============================================================
        // get data Method
//=============================================================
        public byte[]  get_data()
        {
            t4 = (add ^ bytes_no ^ data);
            crc = (byte)takeNOT(t4);
            sbuffer = new byte[] { 150, add, bytes_no, data, crc, 169 };
            return sbuffer;
        }

//=============================================================
```

Code 7 get_data Method

3.3.8.1.3 The Complete Code

For the Complete Code see Appendix F

3.3.8.1.4 Screen Shot of Test Program

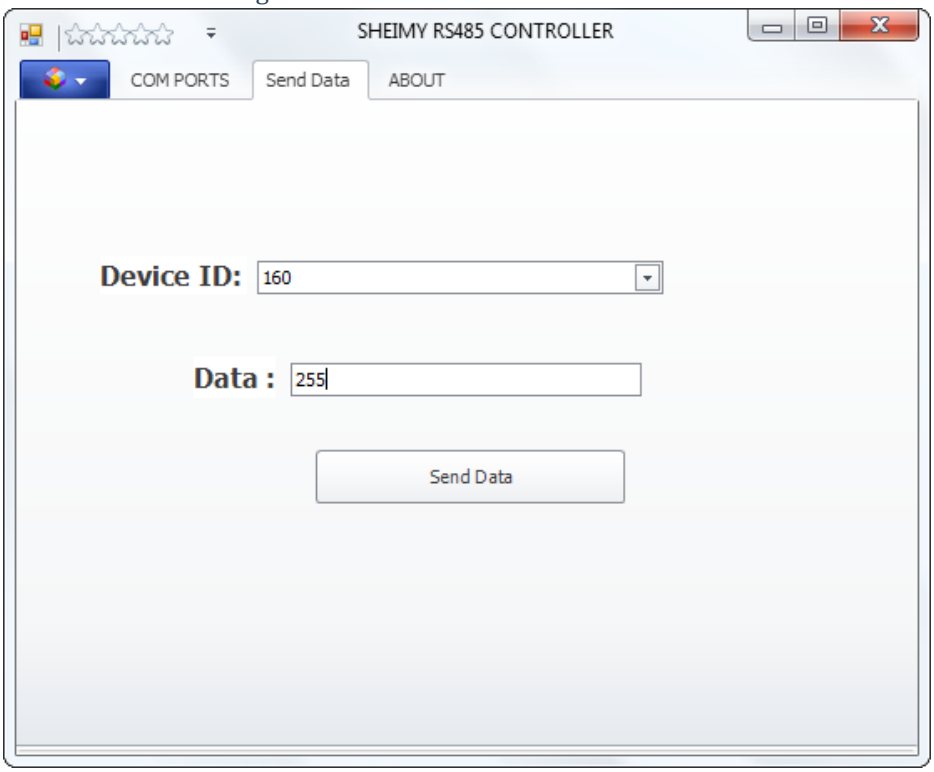


Figure 37 RS485 Test Program

3.4 FBUS Protocol

3.4.1 Introduction

Most Nokia phones have F-Bus and M-Bus connections that can be used to connect a phone to a PC or in our case a microcontroller. The connection can be used for controlling just about all functions of the phone, as well as uploading new firmware etc. This bus will allow us to send and receive SMS messages.

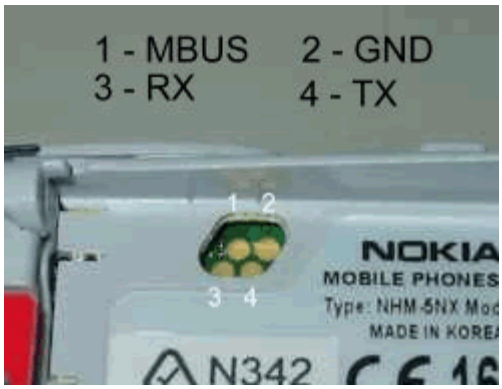


Figure 38 Nokia 3310/3315 F/M Bus connection

The very popular Nokia 3310/3315 has the F/M Bus connection under the battery holder. This is a bit of a pain to get to and requires a special cable to make the connection. The picture above shows the 4 gold pads used for the F and M Bus. The Table below shows the F/M- Bus connection signal direction.

Table 1 F/M-BUS Signal Direction

Pin Number	Pin Name	Direction
1	MBUS	<-->
2	GND	---
3	RX	<--
4	TX	-->

3.4.2 FBUS Protocol

The F-Bus is bi-directional serial type bus running at 115,200bps, 8 data bits. The serial cable contains electronics for level conversion and therefore requires power. The first thing to do is supply power to the cable electronics and this is done by setting the DTR

(Data Terminal Ready) pin and clearing the RTS (Request to Send) pin. Connect the DTR pin to a +3 to 12 Volt supply and RTS to a -3 to -12Volt supply. The easy way to achieve this is by using a Max232 or similar transceiver for the RS232 TX and RX pins and then connecting the DTR pin on the serial cable to the V+ pin on the Max232. Do the same for the RTS; however connect it to the V- pin on the Max232. The V+ and V- pins are derived from internal charge pumps that double the input voltage. I.e. for a 5V Max232, the V+ will +10V and the V- will be -10V.

The next step is to synchronize the UART in the phone with your PC or microcontroller. This is done by sending a string of 0x55 or 'U' 128 times. Simple! The bus is now ready to be used for sending frames.

The Nokia protocol has a series of commands that allow the user to make calls, send and get SMS messages and lots more (see appendix D AT Commands).

So here's the difficult part. The FBUS protocol is made up of numerous bytes and always start with '0x1E' for cable type of connection (we are really not worried for IR or Bluetooth here)

Let's see a frame of bytes which when sent to the Nokia 3310/5110 phone will reply back with the h/w and s/w version of the phone.

Table 2 frame of bytes sent to the Nokia 3310/5110

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	1E	00	0C	D1	00	07	00	01	00	03	00	01	60	00	72	D5
	cable	cell	PC	get ver.	framlen + 3								seq	Padd	even	odd

Byte0=Frame id 1E=cable
Byte1=Destination address=00=phone
Byte2=Source address=0C=terminal/micro/PC
Byte3=Type of command (D1=get version)
Byte4=MSB of frame length
Byte5=LSB of frame length=07 (7bytes ahead and more)
Byte6=Byte7=Byte8=Byte9=Byte10=Byte11=don't need to worry about them! J
Byte12=Sequence number or Seq.No. (Very important)
Byte13=is padding byte and is present if Frame length is odd.
Byte14=Even checksum
Byte15=Odd checksum (embedtronics.com was wrong here)

Byte5 has the info of how many more bytes there are about to come till Seq.No.. comes (which means that after Seq.No.., minimum 2 bytes checksum are always present and a padding byte=0x00 is inserted to make the whole frame even .In our case padding byte=0x00 is present since frame length is odd=0x07)

A little more about the sequence number here: The Seq.No. as the name suggests defines the sequence of frames and goes from 0 to 7 and back to 0.and so on.

E.g: In above example if we were to send the same Get version frame 9 times in a row, what should be done? You are absolutely correct! In the first frame, Seq.No.. is 0x60, next its 0x61,next 0x62……0x67 (and back to..) 0x60. That's all. The phone will be having a count of the source's (micro) Seq.No.'s, and if it's incorrect, then phone will not respond. And you will have to re-initiate the whole 'U' sending again.

Now about the checksums. These are nothing but the XOR of all bytes. In our case they are XOR of all bytes in even positions and odd positions

E.g.: in above example the even checksum will be calculated like this
Byte0 XOR Byte2 XOR Byte4 XOR Byte6 XOR Byte8 XOR Byte10 XOR Byte12
0x1E ^ 0x0C ^ 0x00 ^ 0x00 ^ 0x00 ^ 0x00 ^ 0x60

=0x72

Whereas odd check sum is obtained by XORing all odd placed bytes
Byte1^Byte3^Byte5^Byte7^Byte9^Byte11^Byte13=0xD5

When such a frame is sent, the nokia phone replies with 2 frames

A. An acknowledge frame to tell us that 'he' read the frame.
B. Actual data frame.

Let's see how the ACK frame is made up

Table 3 ACK frame

Byte	0	1	2	3	4	5	6	7	8	9
HEX	1E	0C	00	7F	00	02	D1	00	CF	71
	cable	PC	cell	Ack	framlen + 2		Type	seq	even	Odd

Keep in mind, the above frame is sent by Nokia. (Nokia is source now)

Byte0=Frame id 1E=cable
Byte1=Destination address=0C= terminal/micro/PC
Byte2=Source address=00= phone
Byte3=Type of command (7F=acknowledge frame)
Byte4=MSB of frame length
Byte5=LSB of frame length=02 (2 bytes)
Byte6= (replying to what was sent/asked by micro) D1=get version
Byte7=Seq.No. (No padding byte present after Seq.No. since Frame length=0x02=even)

Byte8=Even check sum
Byte9=Odd check sum
After this ACK frame, the phone will send the actual data frame, with the h/w and s/w version (check my excel sheet on this)

It isn't over yet! The phone also wants a confirmation that we/microcontroller have received the data, so it will be waiting for the ACK frame that we are supposed to send. If we don't send this ACK frame, the phone sends the data two more times (after the first frame) which means that we are entitled to the data 3 times before phone stops sending data.

This also means we need to send an ACK frame to the phone now. This too is quite simple .The ACK frame that we should send is like this

Table 4 ACK frame

Byte	0	1	2	3	4	5	6	7	8	9
HEX	1E	00	0C	7F	00	02	D2	01	C0	7C
	cable	Cell	micro	ack	framlen + 2		Type	seq	even	Odd

I think there's no need to explain the bytes now, but for the sequence number.
When we/micro sends an ACK frame, the sequence number is not the one we are generating (from x0 to x7) but it is the last three bits of the Seq.No. Of previous received data frame. No? Haven't got it? Let's take the example of the Get-version frame.

The phone had sent 0x41 as the sequence number in its h/w, s/w data frame
0x41= 0100 0001 b

Sample frame sent to my Nokia 3310 (showed as a Hex dump)

Byte:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Data:	1E	00	0C	D1	00	07	00	01	00	03	00	01	60	00	72	D5

this sample frame is used to get the hardware and software version from a Nokia phone. It is a good starting point to test if our implementation of the protocol is working.

Byte 0: All frames sent by cable will start with the character 0x1E first. This is the F-Bus Frame ID. Cable is 0x1E and IR is 0x1C. Byte 1: This is the destination address. When sending data, it's the phone's device ID byte. In our case it's always 00 for the phone.
Byte 2: This is the source address. When sending data, it's the PC's device ID byte. In our case it's always 0x0C (Terminal).
Byte 3: This is the message type or 'command'. 0xD1 is Get HW & SW version.

Byte 4 & 5: Byte 4 & 5 is the message length. In our case it is 7 bytes long. Byte 4 is the MSB and byte 5 is the LSB.
Byte 6: The data segment starts here and goes for 7 bytes in our case. As The Nokia is a 16 bit phone and therefore requires an even number of bytes. As ours is odd the last byte will be a padding byte and the message will end at location 13.
The last byte in the data segment (Byte 12 above) is the sequence number. The last 3 bits of this byte increment from 0 to 7 for each frame. This part needs to be sent back to the phone in the acknowledge frame. The other bits I am unsure about what they mean!
Bytes 14 & 15: The second to last byte is always the odd checksum byte and the last byte is the even checksum byte. The checksum is calculated by XORing all the odd bytes and placing the result in the odd Checksum location and then XORing the even bytes and then placing the result in the even byte.

Well that is our first frame for our Nokia Phone. If the phone received it is show reply with the following data

1E 0C 00 7F 00 02 D1 00 CF 71
1E 0C 00 D2 00 26 01 00 00 03 56 20 30 34 2E 34 35 0A 32 31 2D 30 36 2D 30 31 0A
4E 48 4D 2D 35 0A 28 63 29 20 4E 4D
50 2E 00 01 41 3F A4

The first line is an Acknowledge command frame. Notice how the destination and source addresses are now swapped. This is because the Nokia phone is now talking. This message is two bytes long with the two bytes representing the message type received (0xD1) and the sequence number (0x00). The last two bytes are the checksum and should be checked to make sure the data is correct. The 3310 will be waiting for an acknowledge frame after these two frames were sent. If the acknowledge frame is not sent the 3310 will retry sending the data. The 3310 will only send the data 3 times and then gives up.

The second frame from our Nokia 3310 is the data we requested. The message type is 0xD2. This is 'receive Get HW&SW version'. This 38-byte (0x26) message should show 0x0003 "V" "firmware\n" "firmware date\n" "model\n" "(c) NMP." The last byte in the data is the sequence number. As with standard F-bus frames, the last two bytes in the frame are checksum bytes.

The received data without f-bus frame

01 00 00 03 56 20 30 34 2E 34 35 0A 32 31 2D 30 36 2D 30 31 0A 4E 48 4D 2D 35 0A 28 63 29 20 4E 4D 50 2E 00 01 41

0003 V 0 4 . 4 5 \n 2 1 / 0 6 / 0 1 \n N H M - 5 \n (c) N M P. Sequence no. All that is required now is to send a acknowledge frame back to the phone to say 'I got it!'

1E 00 0C 7F 00 02 D2 01 C0 7C 0x7F is the acknowledge frame's command. We are only required to send a two-byte message so length is set to 0x02. The message contains the acknowledged message type (0xD2) and the sequence no. (0x01). the sequence number is made from the last 3 bits of the sequence number in the previous frame. The checksum needs to be calculated and sent

3.4.3 FBUS Communication

3.4.3.1 Signal levels

The FBUS data sent by the PC and the reply from the phone are shown in the following figure. Please note that the signal level of the computer (the data burst on the left) quite bad only because the measurement circuit attenuates the signal. It should a rail-to-rail signal from 0 to 3 volts like the phone response. These images are captured from Nokia Data Suite 2.0 (NDS) communication.

The FBUS data sent by the PC and the reply from the phone are shown in the figure 1.2. Please note that the signal level of the computer (the data burst on the left) quite bad only because the measurement circuit attenuates the signal. It should a rail-to-rail signal from 0 to 3 volts like the phone response. These images are captured from Nokia Data Suite 2.0 (NDS) communication.

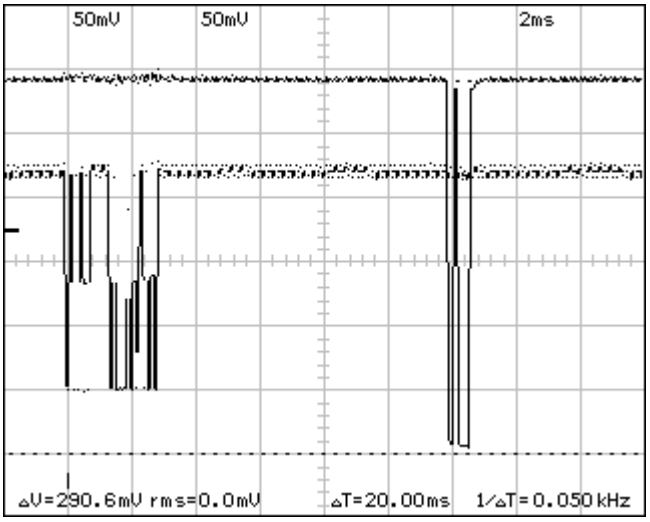


Figure 39 a part of the FBUS communication of the Nokia Data Suite

A good guess about the FBUS drivers is that there is a high-impedance input in the FBUS Rx pin of the phone, and a real push-pull buffer output in the FBUS TX pin.

But it is not so simple. By looking the startup situation of the Nokia Data Suite communication. The start is shown in figure below.

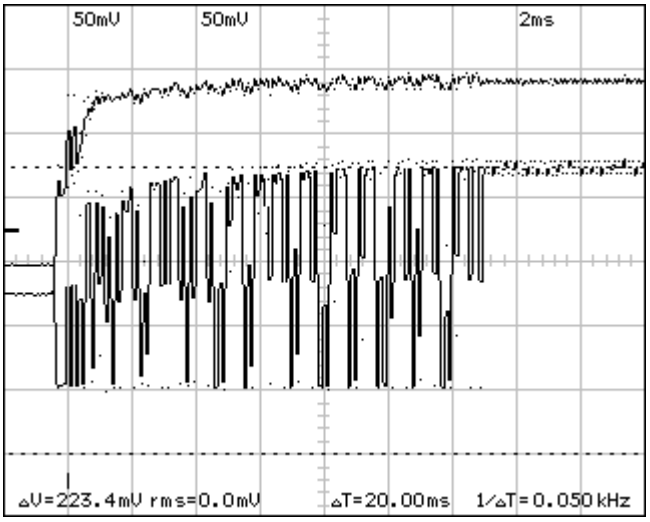


Figure 40 the start of FBUS communication measured with the test circuit

By looking at the above figure you must see that the FBUS TX line, the phone FBUS output (the signal without data pulses) will raise after the buffers of the measuring circuit are powered. (The power is applied to the measurement circuit just before the first time-division line on the picture.) The "middle" state of the phone before the power-up means that the FBUS Tx output driver does not drive the pin high all the time, but it is a three-state output. (Measurement on an idle phone with a current meter shows the same thing.)

By pulling the FBUS Tx down instead of the 400 kilo-ohm pull-up shows that the FBUS Tx *drives the signal up to 3 V* during the data pulses, so it is not an open-collector type output.

The above figure shows also that the Nokia Data Suite 2.0 starts the communication immediately after supplying power to the measurement circuit (the data pulses on the PC-to-phone line). The first pulses are possibly not transmitted correctly because the adapter circuit is not yet correctly powered. (The power-up ramp can be seen in the upper signal on the scope.) Fortunately, this does not cause problems with the NDS.

3.4.3.2 Data speed

A part of a data burst from the PC to the phone is shown in the figure below.

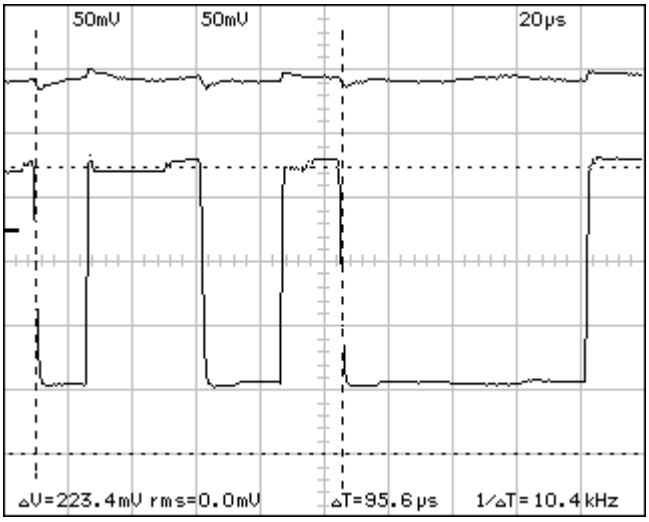


Figure 41 A close-up view to the FBUS data sent by the PC

By assuming that the data format contains the typical start and stop bits, one possible data byte is shown in the figure with the time cursors. If it contains 11 bits (2 low, 4 high, 3 low, and 2 high), then the baud rate near the standard value of 115200 baud.

A similar view to the data sent by the phone is shown in the figure below.

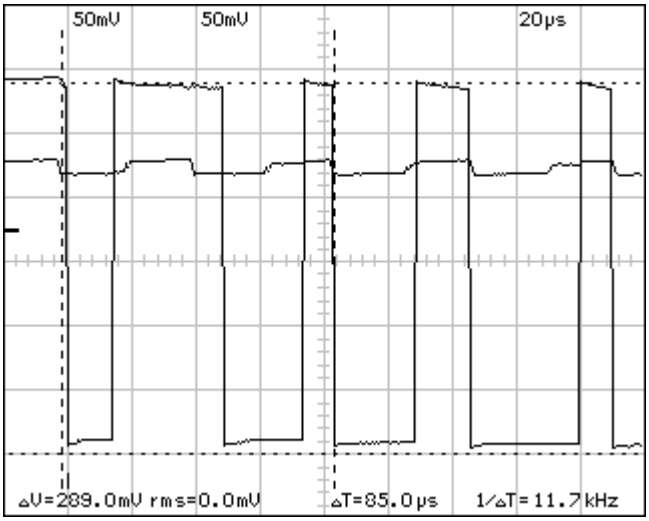


Figure 42 FBUS data sent by the phone.

3.4.4 FBUS Implementation

3.4.4.1 Hardware Implementation

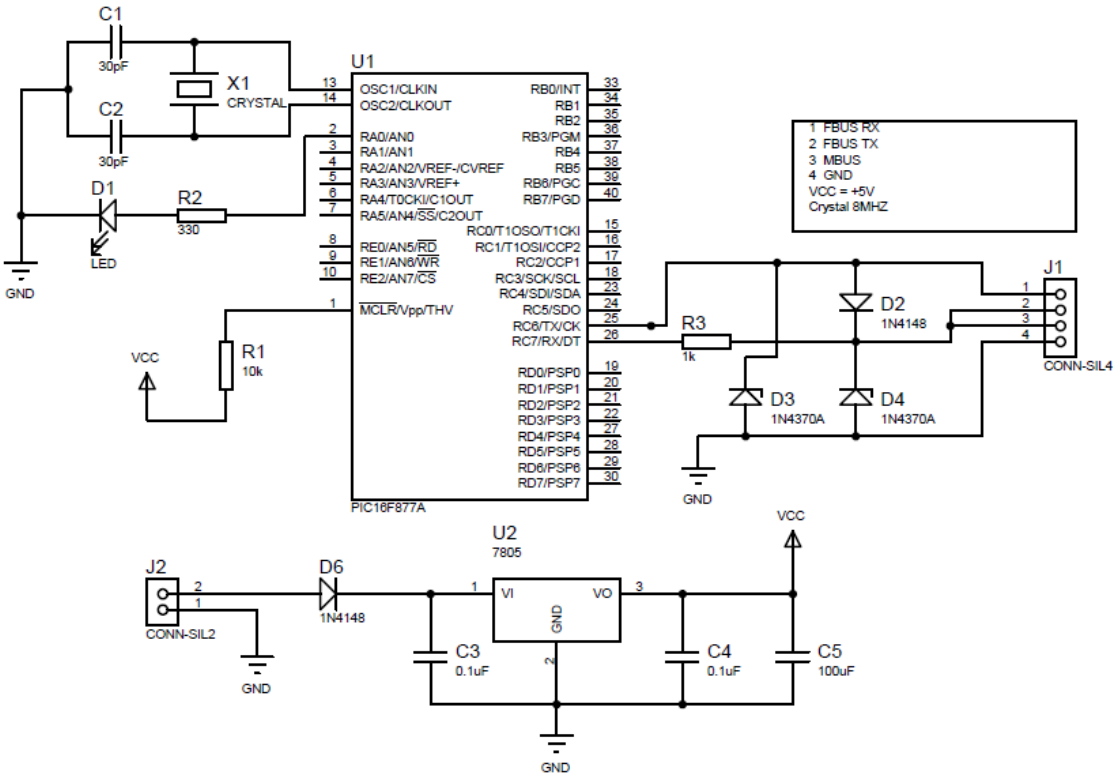


Figure 43 FBUS Interfacing Schematic

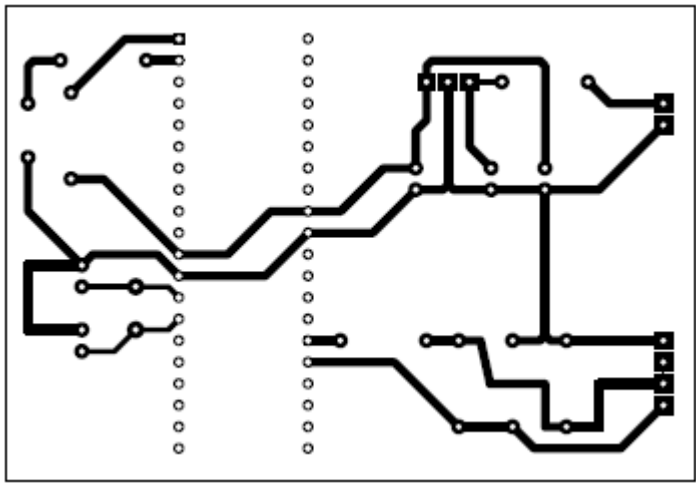


Figure 44 FBUS Interfacing Layout

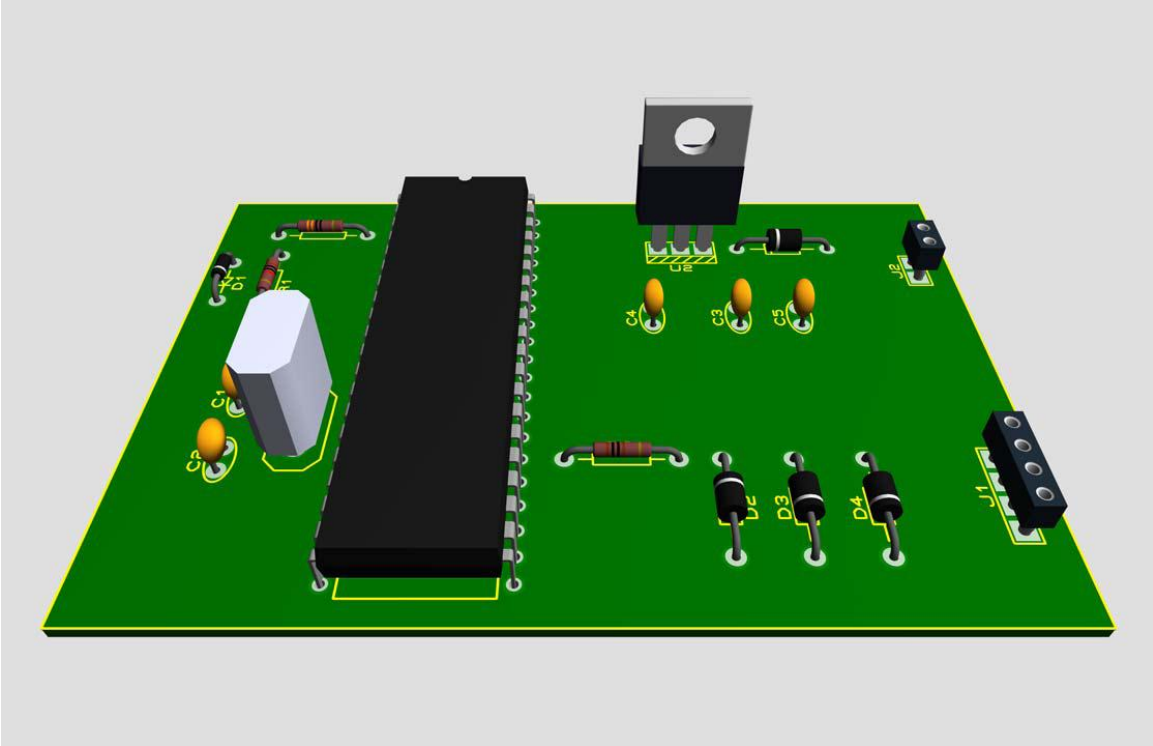


Figure 45 3D View of PCB

3.4.4.2 *Software Implementation*

a. Start FBUS Method

In the Start FBUS Method should send to the phone the Starting byte 128 times to begin the Synchronization with the Microcontroller, the Starting Byte is (0x55) , so the method will be,

```
void StartFbus ()

   {

UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55
);

UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55
);

UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55
);…….. }
```

Code 8 StartFbus Method

b. Message one Sending

```
/*Message1 here*/
void mess ()
{
UART1_Write (0x1E); UART1_Write (0x00);
UART1_Write (0x0C); UART1_Write(0x02 );
UART1_Write(0x00 );UART1_Write(0x35 );
UART1_Write(0x00 );UART1_Write(0x01 );
UART1_Write(0x00 );UART1_Write(0x01 );
UART1_Write(0x02 );UART1_Write(0x00 );
UART1_Write(0x07 );UART1_Write(0x91 );
}  // end method
```

Code 9 get the Phone software version Method

c. UART Initialization

```
// Initialize Uart at 115200 baud rate
  UART1_Init(115200);
```

Code 10 Initialize the Uart

d. Receive Reply Message

```
// If data is ready, read it:
  If (UART1_Data_Ready () == 1) {
        // received parameter
        char r;
        // Assign received data to a variable r
        r = UART1_Read ();
        // display data on portB
        portB = r;
  } // end if
```

Code 11 reading from phone

3.5 TCP/IP protocol

3.5.1 What is TCP/IP Protocol?

The TCP/IP model is a description framework for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It evolved from ARPANET, which were the world's first wide area network and a predecessor of the Internet. The TCP/IP Model is sometimes called the Internet Model

The TCP/IP model, or Internet Protocol Suite, describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers.

TCP/IP is generally described as having four abstraction layers. This layer architecture is often compared with the seven-layer OSI Reference Model; using terms such as Internet Reference Model in analogy is however incorrect as the Internet Model is descriptive while the OSI Reference Model was intended to be prescriptive, hence Reference Model.

The TCP/IP model and related protocols are maintained by the Internet Engineering Task Force (IETF).

The TCP/IP model is a description framework for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It evolved from ARPANET, which were the world's first wide area network and a predecessor of the Internet. The TCP/IP Model is sometimes called the Internet Model or the DoD Model.

The TCP/IP model, or Internet Protocol Suite, describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers.

We will use this protocol to control in devices connected to server PC from any client pc in our network.

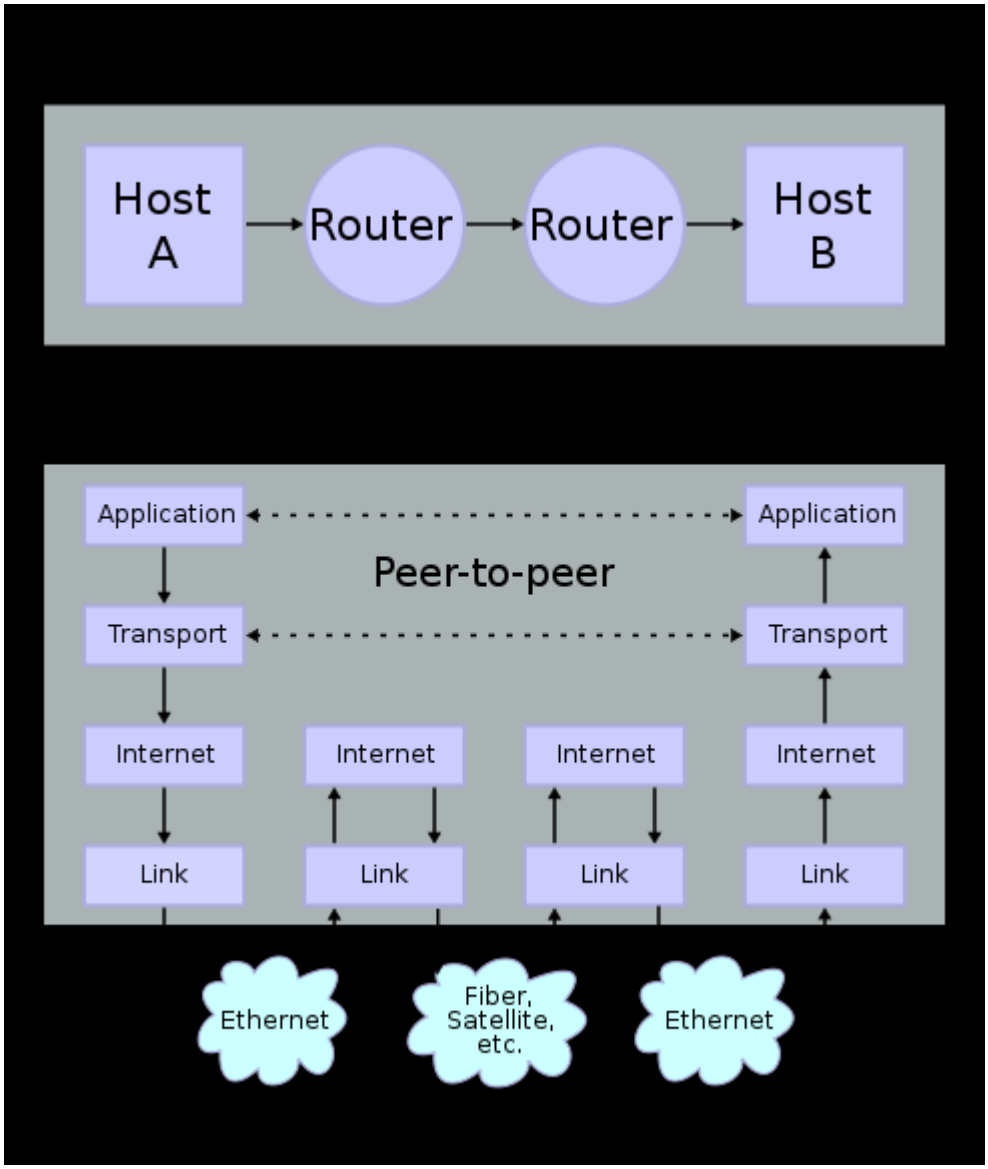


Figure 46 TCP/IP Connection

3.5.2 The Advantages of using TCP/IP in control:

a. Setup Connection Before Transmission (handshaking)

b. Reliable, in order data transmission.

c. Flow Control (no side can overwhelm the other side with packets).

d. Congestion Control (Slow down when network is congested).

e. Example of application: HTTP, SMTP and FTP.

3.5.3 How does the hardware work?

The hardware is that of the RS485 as explained in the previous sections. The added section is that the LAN network between the two PCs .now it is easy to transmit the control signal (data) between the PCs in the LAN to control in the Devices from any PC in the network that know the IP address and the password of the devices that is connected to the server PC. The structure is like the following

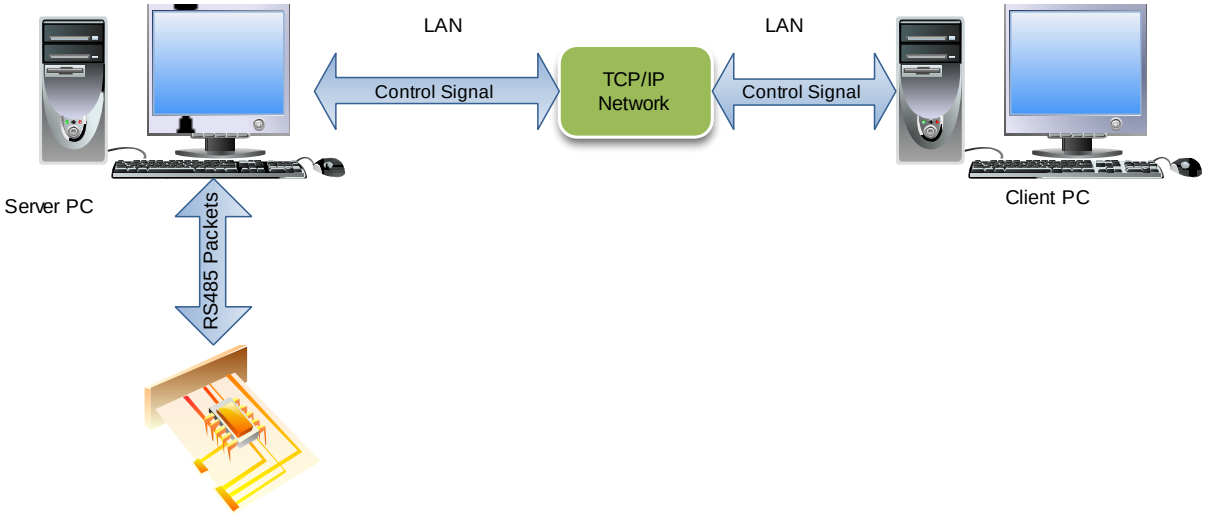


Figure 47 TCP/IP control Structure

As shown in the above Figure the control data is transmitted from the Clint PC to the Server PC over the LAN cables as a TCP/IP packets .the server is translate it to RS485 Packets to send it to the specified Device .this will be shown in the next section.

3.5.4 How does the software work?

The software is a program that can receive a TCP/IP's packets and can extract the data and the ID address from it, also it check for the password to check if the data received from the Clint program is Authorized or not if it is good the data and the ID address is repackage to a RS485 packet to be send to the device that can Understood it. the following figure is explain this structure

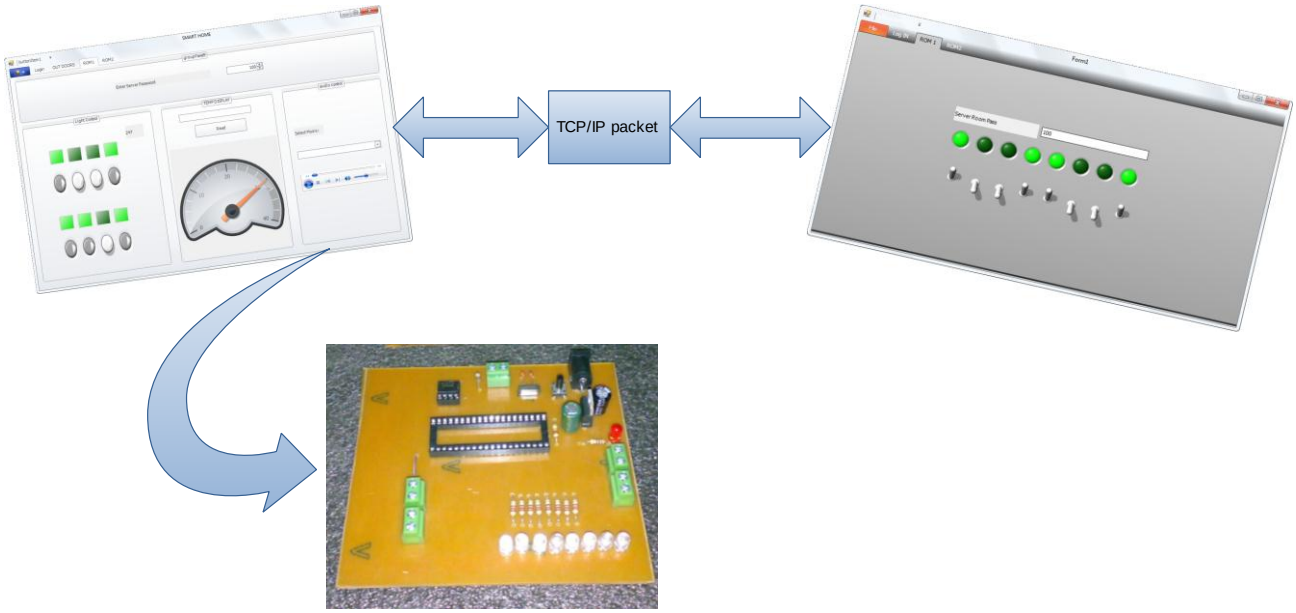


Figure 48 how does the TCP/IP software work

3.5.5 Steps to begin Communication

There is some important settings for connection to be successful they are below:

a. PC in the same LAN
b. client PC software has to know the server IP
c. client Pc software has to know the each room password to make successful connection with it
d. the server program must be on all time for the Clint to make successful connection
e. the hardware must be connected to the sever PC
f. the serial port com must be opened

3.5.6 TCP/IP Hardware implementation

The hardware is the same of the RS485 the only added hardware is the switch and the LAN structure between the server and the client PC.

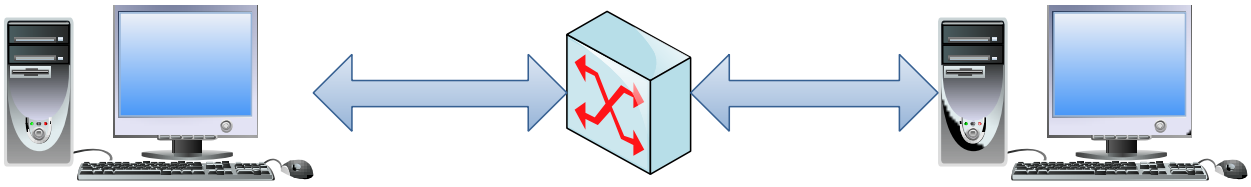


Figure 49 Client and Server connection

3.5.7 TCP/IP Software Program

As explained that the programming language used is the C# .in the following we will see how the connection between 2 PCs will be established using the C# language. The software is a two parts which is the client and the server. In the following the software of the client and server will be explained.

3.5.7.1 TCP/IP Server Program

The server program is the program that will be run in the PC witch all devices is connected to it. this software will accept the data from all client connected to it, and then will check for some filed in the received massage like ID address, password and the data if the ID address and the password is true it will pass the data to another part (this part is the RS485 software part) of the software program `that is responsible for passing that data to the specified ID address.

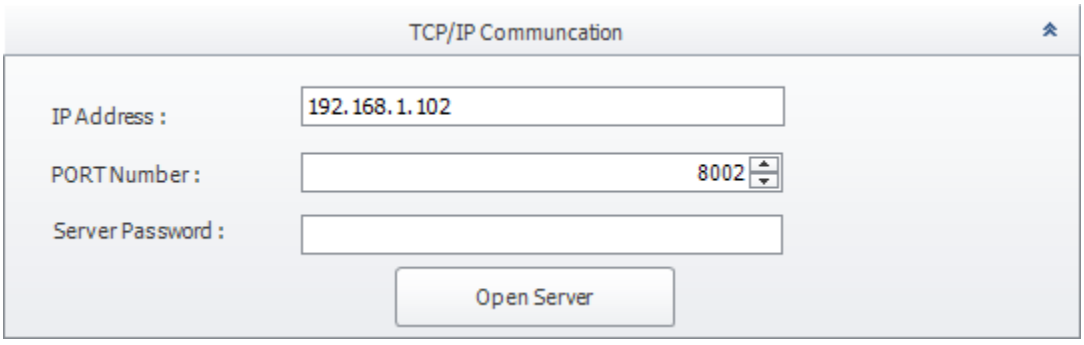


Figure 50 TCP/IP part in login screen

As shown in the above figure which is apart from the login screen it responsible for specifying the port number for the server to connect to the clients through it, also to specify the password of the server to make a secure connection between it and its client, the IP address is automatically gate from the PC that the server will run in it.

After pressing the open server button the server window will be opened .it is like the following

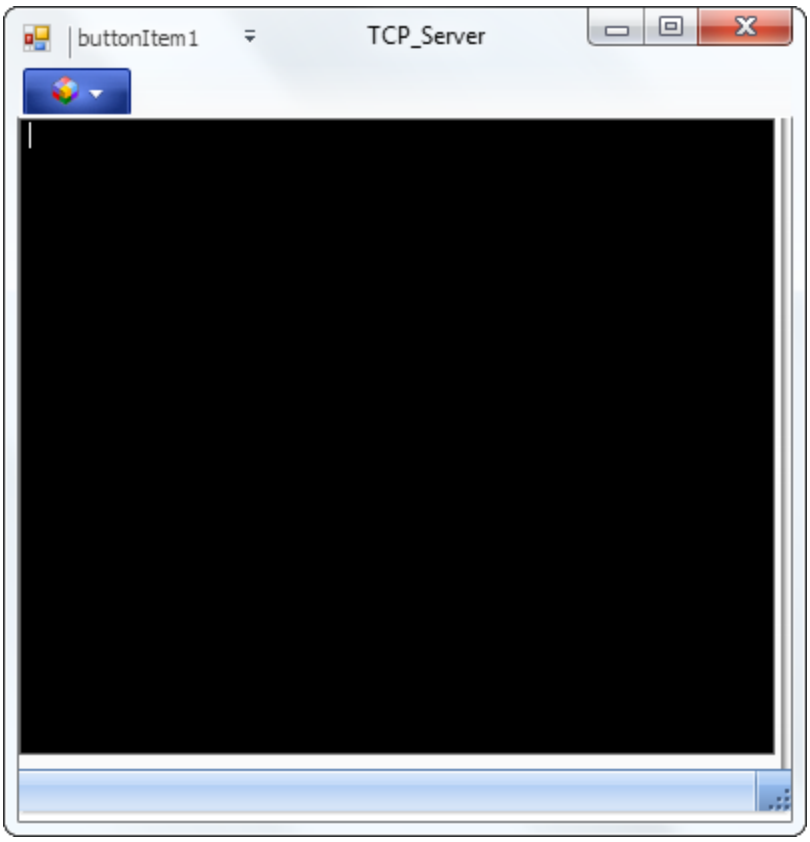


Figure 51 TCP Server window

The black screen will display the state of the connected and the disconnected client

3.5.7.1.1 The software implementation of the server

The following is the monitoring of the important parts of the sever TCP program .

a. Server used library

The used library for the connecting over the TCP/IP is the Net library it is imported to be used in the program using the following code.

```
using System.Net;

using System.Threading;
using System.Net.Sockets;
using System.IO;
```

Code 12 Used Library in Server TCP/IP

b. Struct Client Data

Next to store data for each client that will connect to the server .the struct Client Data will be defined .it is like the following:

```
public struct ClientData
{
    public Socket structSocket;
    public Thread structThread;
}
```

Code 13 struct ClientData

c. Server used variables

```
private TcpListener tcpLsn;
private Hashtable dataHolder = new Hashtable();
private static long connectId = 0;
private Thread tcpThd;
delegate void SetTextCallBack(string text);
Form1 form_1;
Sheimy_RS485 ROM;
private byte[] rom_data;
```

Code 14 Server used variables

tcpLsn: is a TcpListener that will listen the specified port in the specified IP assigned to the server PC.

dataHolder: it is a Hashtable used to store the connecte ID and its data

tcpThd :is a thread used for the TCP

SetTextCallBack(string text): is a delegate Method that is used to pass data between two threads.

form_1: is an object from Form1 that is the main form so it is possible to send data to the devices connected to it over r the serial port.

ROM : is an object from Sheimy_RS485 class that is used to calculate the RS485 pattern to be transmitted over the serial port.

rom_data: is a byte Array that is used to store the calculated RS485 pattern to be sends.

d. TCP_Server constructor

This is used to make initialization for the variables also it is called every time a create and object from it. This constructor is shown in the following block code.

```
//=====================================================================
        public TCP_Server(IPAddress ip, int portno, Form1 f)
        {
            InitializeComponent();
            tcpLsn = new TcpListener(ip, portno);
            tcpLsn.Start();
            statusBar1.Text = "Listen at: " +
tcpLsn.LocalEndpoint.ToString();
            tcpThd = new Thread(new ThreadStart(WaitingForClient));
            tcpThd.Start();
            form_1 = f;
           // pass = form_1.getpass();
            ROM= new Sheimy_RS485();
        }
 //=====================================================================
```

Code 15 TCP_Server constructor

e. WaitingForClient Method

This method is used to add each new connected client to the dataHolder hash table also to indicate the ID of connects. This method is called inside a thread inside the constructor

```
//=====================================================================
        public void WaitingForClient()
        {
            ClientData CData;
            while (true)
            {     /* Accept will block until someone connects */
                CData.structSocket = tcpLsn.AcceptSocket();
                Interlocked.Increment(ref connectId);
                CData.structThread = new Thread(new
ThreadStart(ReadSocket));

                lock (this)
                { // it is used to keep connected Sockets and active
thread
                    dataHolder.Add(connectId, CData);
                    upDateDataGrid("Connected > " + connectId + "    "
+ DateTime.Now.ToLongTimeString());
                }
                CData.structThread.Start();
            }
        }
 //=====================================================================
```

Code 16 Waiting for Client Method

f. Read Socket Method

This method read is called inside anew thread for every connected new client to handle the data from this client and to get the control data from it to send it to the devices after checking for the password and insure it is correct. It is in the following code block

```
  public void ReadSocket()
        {
            /* realId will be not changed for each thread, but
connectId is
             * changed. it can't be used to delete object from
Hashtable*/
            long realId = connectId;
            Byte[] receive;
            ClientData cd = (ClientData)dataHolder[realId];
            Socket s = cd.structSocket;
            int ret = 0;

            while (true)
            {
                byte room1_pass,room2_pass;
                room1_pass = form_1.getRoom1_pass();
                room2_pass = form_1.getRoom2_pass();
                if (s.Connected)
                {
                    receive = new Byte[100];
                    try
                    {   /* Receive will block until data coming ret is
0 or Exception
                                        *  happen when Socket connection
is broken*/
                        ret = s.Receive(receive, receive.Length, 0);

                        if (ret > 0)
                        {

                            if ((receive[0] == 160 && receive[1] ==
room1_pass) || (receive[0] == 170 && receive[1] == room2_pass))
                            {
                                ROM.set_add(receive[0]);
                                ROM.set_data(receive[2]);
                                rom_data = ROM.get_data();

                                form_1.send_data(rom_data);
                            }
                            else
                            {
                                form_1.speak_text("Dear User : someone
try Hacking your Device");
```

Code 17 ReadSocket Method

```
foreach (ClientData clntData in dataHolder.Values)
                                {
                                    if (clntData.structSocket.Connected)

clntData.structSocket.Send(receive, ret, SocketFlags.None);
                                }
                            }
                            else { break; }
                        }
                        catch (Exception e)
                        {
                            upDateDataGrid(e.ToString());
                            if (!s.Connected) break;
                        }
                    }
                }
                CloseTheThread(realId);
                                        }
```

Code 18 ReadSocket Method (continued)

g. CloseTheThread Method

This method is used to close the thread assigned to specific client when it is disconnected from the server. Also to display this in the server data grid.

```
//=====================================================================
        private void CloseTheThread(long realId)
        {
            try
            {
                ClientData clientData =
(ClientData)dataHolder[realId];
                clientData.structThread.Abort();
            }

            catch (Exception e)
            {
                lock (this)
                {
                    dataHolder.Remove(realId);
                    upDateDataGrid("Disconnected > " + realId + "    "
+ DateTime.Now.ToLongTimeString());
                }
            }
        }
 //=====================================================================
```

Code 19 CloseTheThread Method

h. The complete Code

For the complete code see appendix……..

3.5.7.2 TCP/IP client Program

The client program is that will be run in the client PC so that it can control in the devices connected to the server .this software is send the following pattern to the server

Figure 52 send pattern from the client

3.5.7.2.1 The software implementation of the client

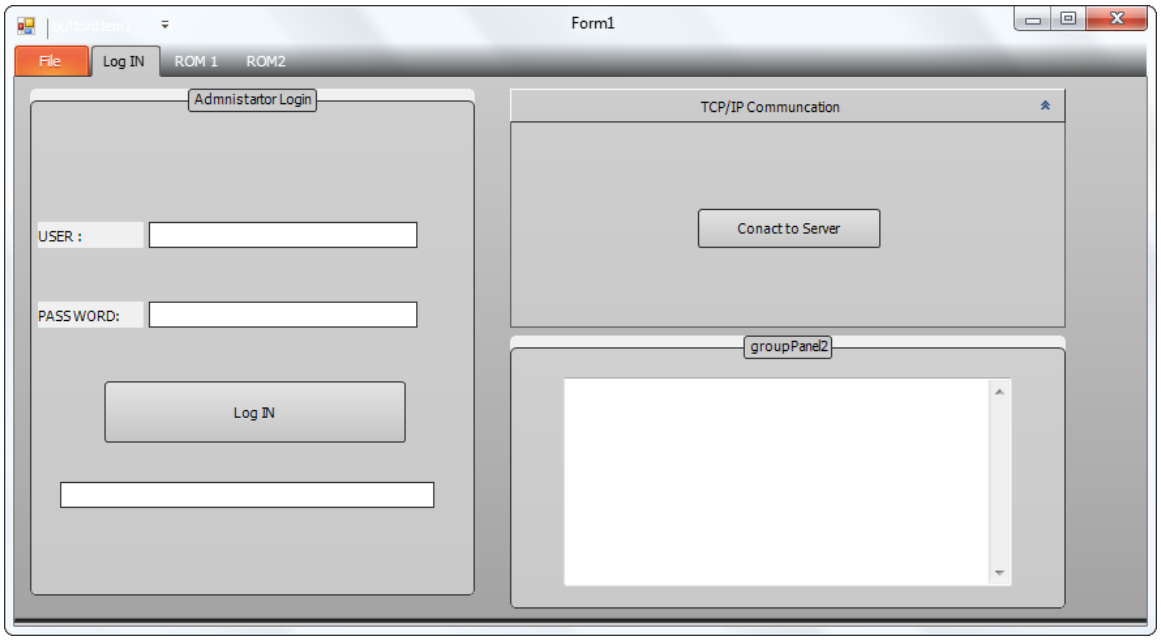


Figure 53 TCP client login form

When the connect to Server button is pressed the following window will displayed

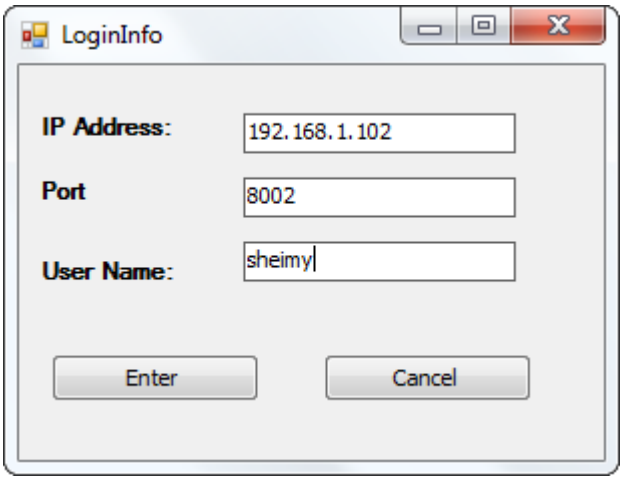


Figure 54 required data to connect to the server

The IP Address is the Address of the Server that it will connect to it. Also the port number is the same that is opened in the server PC and the User Name is the name of the client

.

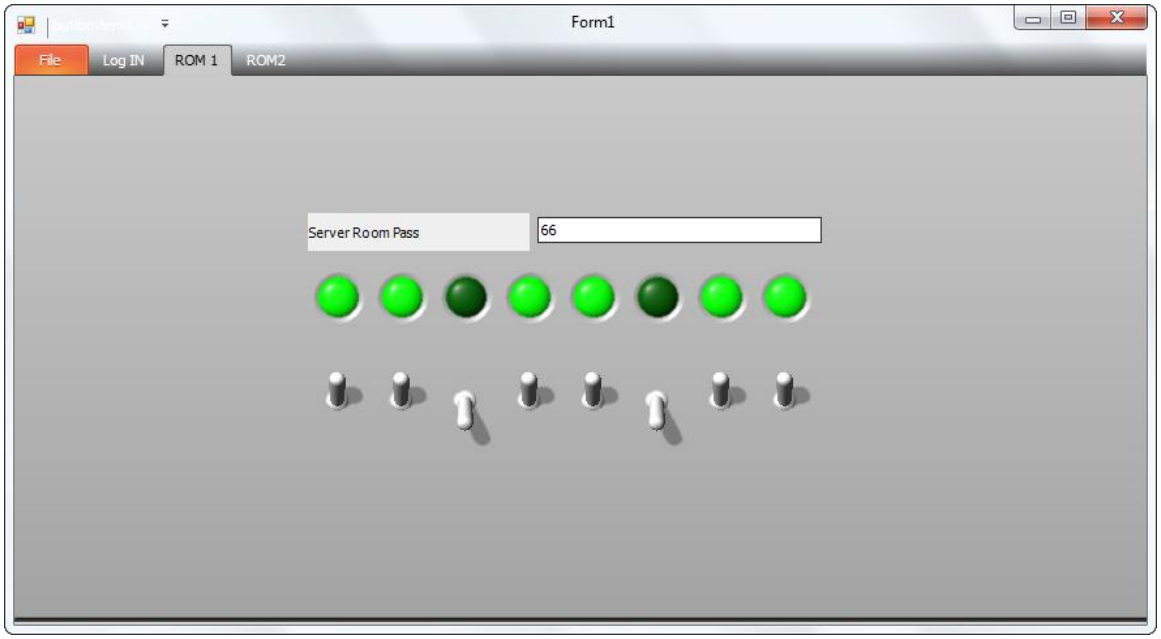


Figure 55 control of client PC software

In the above window the user have to specify the password of the Room that he wants to connected to .after this he can control in all devices in the room by sending the data to the server to send it to the room ID connected to it.

The following is the monitoring of the important parts of the client TCP program.

a. Client used library

```
using System.Net.Sockets;
using System.IO;
using System.Threading;
```

Code 20 client used library

b. Client used variables

```
public Thread tcpThd;
public byte[] readBuffer;
public byte[] writeBuffer;
public Stream stm;
public Socket socket;
public TcpClient tcpclnt;
public string loginName = "";
private LoginInfo loginForm;
```

Code 21 Client used variables

c. Convert bool to byte Method

This method is used to convert the bool value of the switches to byte value to be transmitted over TCP/IP protocol

```
//=========================convert bool to byte ========================
        private byte bool_to_byte(bool[] a)
        {
            int i = 0;
            byte result = 0;
            foreach (bool d in a)
            {
                if (d == true)
                    result += (byte)Math.Pow(2, i);
                i++;
            }
            return result;

        }
 //============================================================================
```

Code 22 convert bool to byte Method

d. Start Server Method

This method is used to connect to the server with specific IP address and port Number and the name of the connected client

```
//=====================================================================
        public void startServer(string ipAddress, int portNumber,
string loginName)
        {
            this.loginName = loginName;
            tcpclnt = new TcpClient();
            tcpclnt.Connect(ipAddress.Trim(), portNumber);
            textBoxWindow.AppendText("Connecting to server...");

            writeToServer("Hello " + loginName + " Now you are
connected to the server" + "\r\n");
            stm = tcpclnt.GetStream();
            tcpThd = new Thread(new ThreadStart(ReadSocket));
            tcpThd.Start();
        }
 //=====================================================================
```

Code 23 startServer Method

e. Send data to server Method

This Method is used to write a the chosen specific pattern value to the server

```
//=====================================================================
        private void switchArray1_ValuesChanged(object sender, EventArgs
e)
        {
            ledArray1.SetValues(switchArray1.GetValues());
            writeBuffer = new byte[3];
            writeBuffer[0] = 160;
            writeBuffer[1] = byte.Parse(rom1_pass.Text);
            writeBuffer[2] =bool_to_byte(switchArray1.GetValues());
            if (stm != null) stm.Write(writeBuffer, 0,
writeBuffer.Length);
        }
 //=====================================================================
```

Code 24 Send data to server Method

This Method is used to read data from Socket that is connected to the server

```
//=======================================================================
        public void ReadSocket()
        {
            while (true)
            {
                try
                {
                    readBuffer = new Byte[100];
                    stm.Read(readBuffer, 0, 100);

                    /* If the text box exceed the maximum lenght, then
get
                     * remove the top part of the text*/
                    if (textBoxWindow.Text.Length >
textBoxWindow.MaxLength)
                    {
                        textBoxWindow.Select(0, 300);
                        textBoxWindow.SelectedText = "";
                    }


textBoxWindow.AppendText(System.Text.Encoding.ASCII.GetString(readBuff
er) + "\r\n");
                }

                catch (Exception e)
                { break; }
            }
        }
 //=======================================================================
```

Code 25 client Read Socket Method

Chapter IV: Quick User Guide

4.1 Use cases

4.1.1 Server Use Case

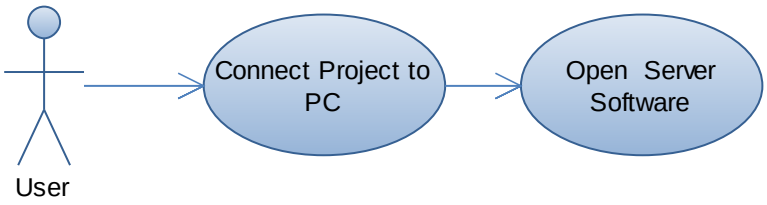


Figure 56 Server use Case

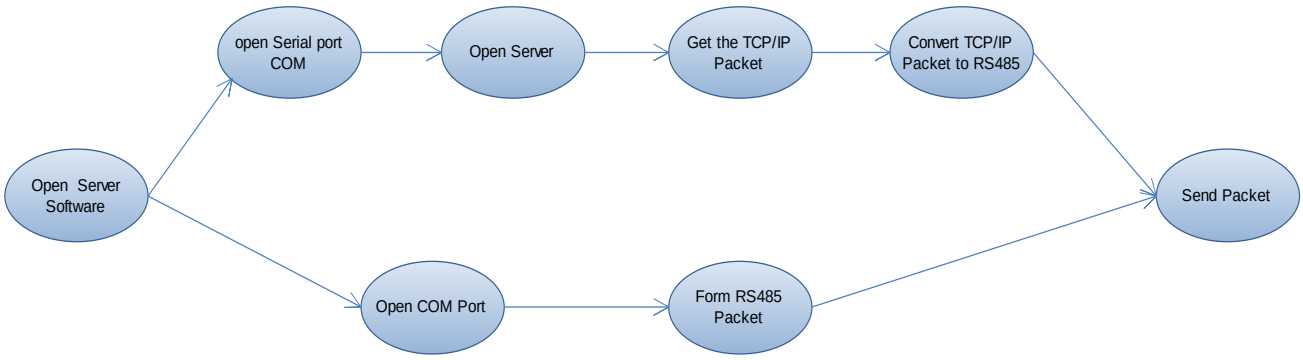


Figure 57 Server use Case (Continued)

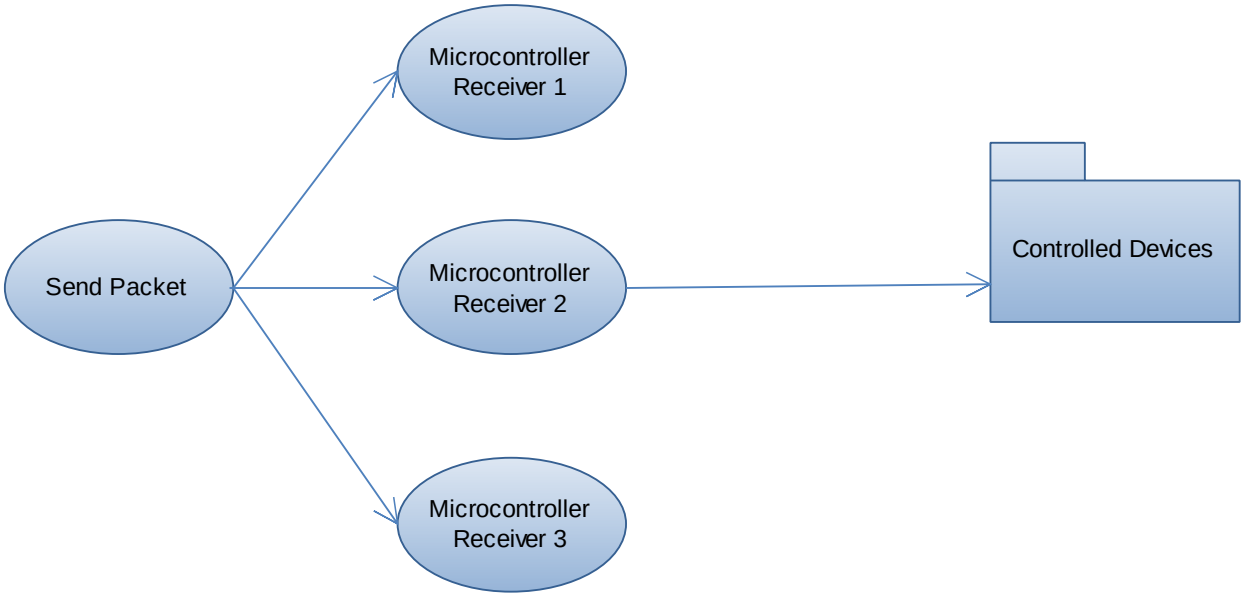


Figure 58 Server use Case (Continued)

4.1.2 Client Use Case

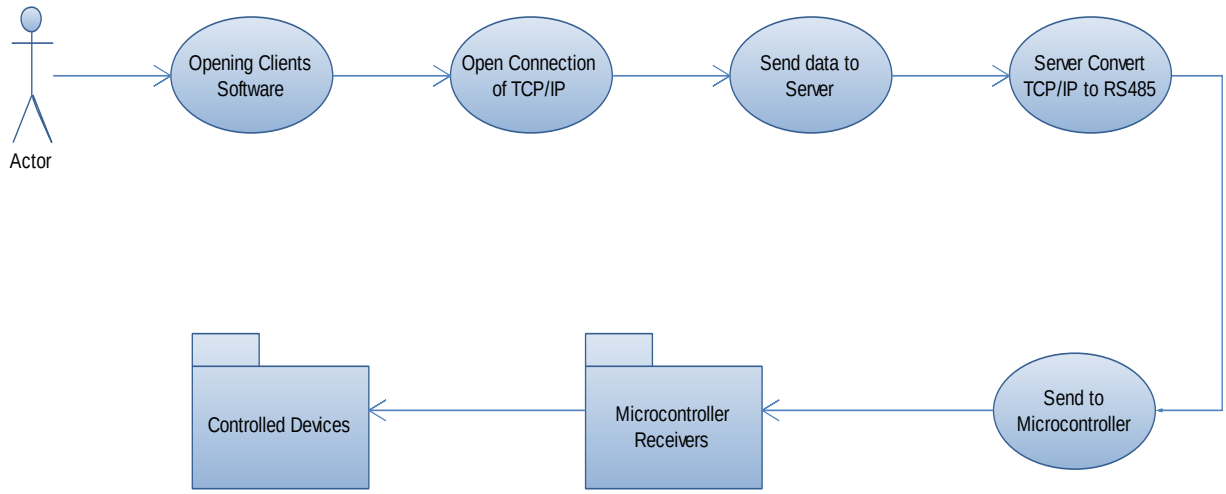


Figure 59 Client Use Case

4.1.3 X10 Transmitter/receiver use Case

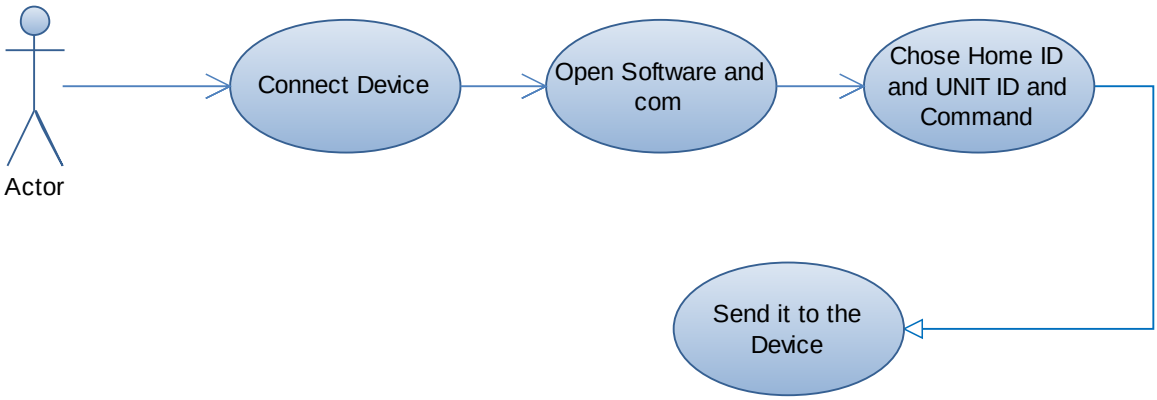


Figure 60 X10 Transmitter/receiver use Case

4.2 Basic Settings

4.2.1 Server Basic Settings

4.2.1.1 COM Port Settings:

a. Port Name (e.g. COM1, COM2).

b. Port Baud rate (e.g. 9600, 1400).

c. RTS(request to transmit)in this project it is true if we Want to Receive and false if we want to transmit

d. Parity (in this project it is none).

e. Data Bits(in this project it is 8 bits)

f. Stop Bits(in this project it is One)

4.2.2.2 TCP/IP Server Settings

a. IP address (it is the IP of the PC the server running in it).

b. Port number that will be used to send and receive data over it.

c. The password of each room

4.2.2.3 TCP/IP client Settings

a. the IP Address of the Server

b. the port Number

c. the user name

4.2.2.4 Microcontroller settings

a. the crystal oscillator must be like that used by the compiler program

b. the Serial port baud rate must be like that used by the software

c. the all RS485 boards must be connected to the A and B wires

d. for long distance the ground wire must be common for all boards

4.2.2.5 Power settings

a. all devices works by 5 V DC

b. for stable voltage the Voltage Regulator is used(12 V to 5 V)

c. for + and – of the source the diode is used to save the device if the polarity is not true

4.3 Training Mode

4.3.1 RS485 Transmitter

The training is done using firstly the simulation programs (Proteus 7 Professional) and with the help of the Virtual Serial Port Driver program to make a connection between the simulation program and the first developed test program. The following is the first RS485 transmitter test program

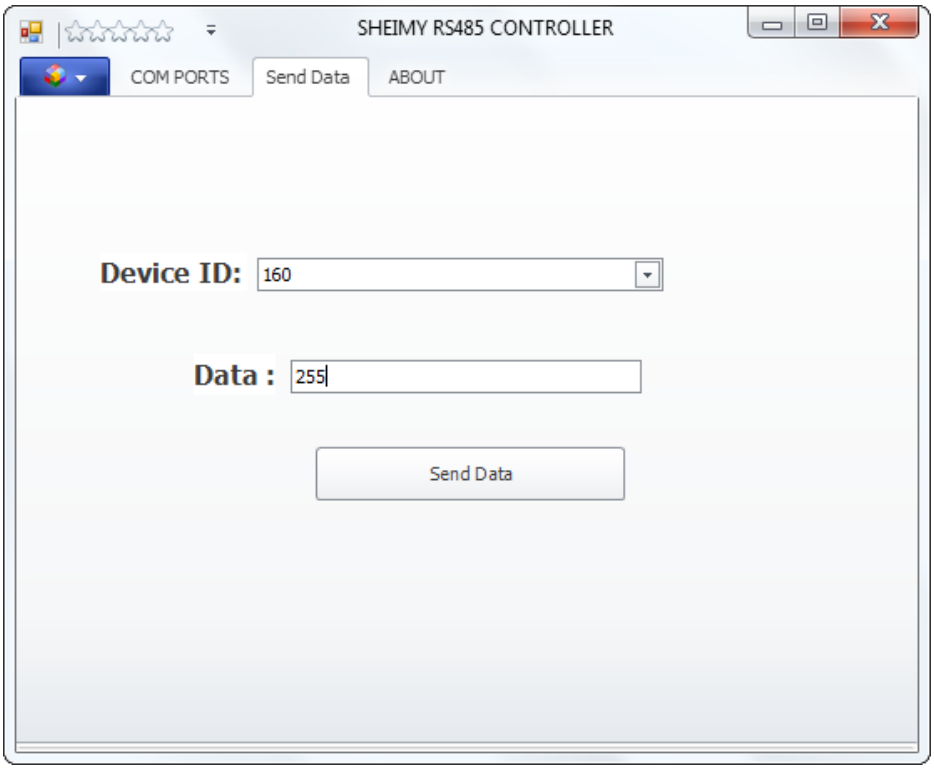


Figure 61 first RS485 transmitter test program

After the successful simulation the test board is used to make the second training operation. And after the successful of that the PCB was made.

4.3.2 RS485 Receiver

After the success of sending the RS485 packet the RS485 receiver was developed to receive data from the serial port and display it also the used simulation and virtual serial port driver used to simulate the transmitter is also used. The following is the transmitter circuit used in simulation to measure the temp and send it to the program

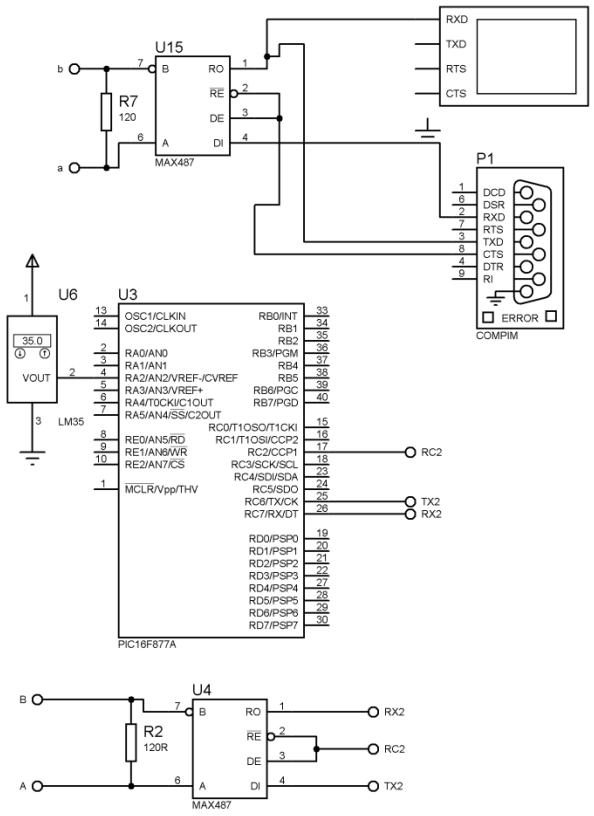


Figure 62 RS485 Receiver simulation circuit

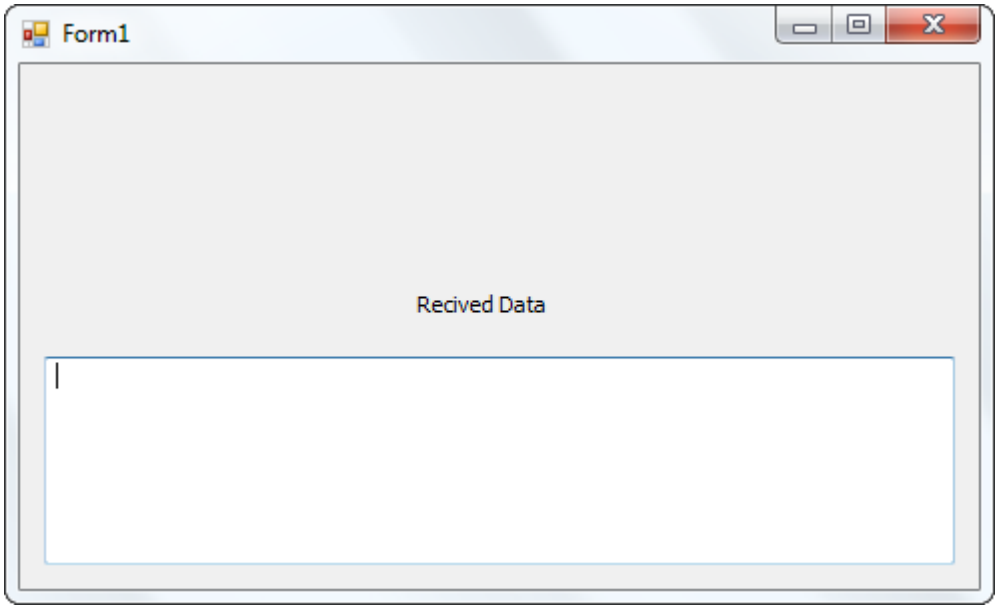


Figure 63 RS485 Receiver test program

4.3.3 RS485 Transmitter/ Receiver

After the success of sending the RS485 and Receiving it individually both of the transmitter and receiver program are combined in one program .also only one microcontroller can be programmed for transmitting and receiving operations. The following is the pictures of the transmitter/receiver program developed.

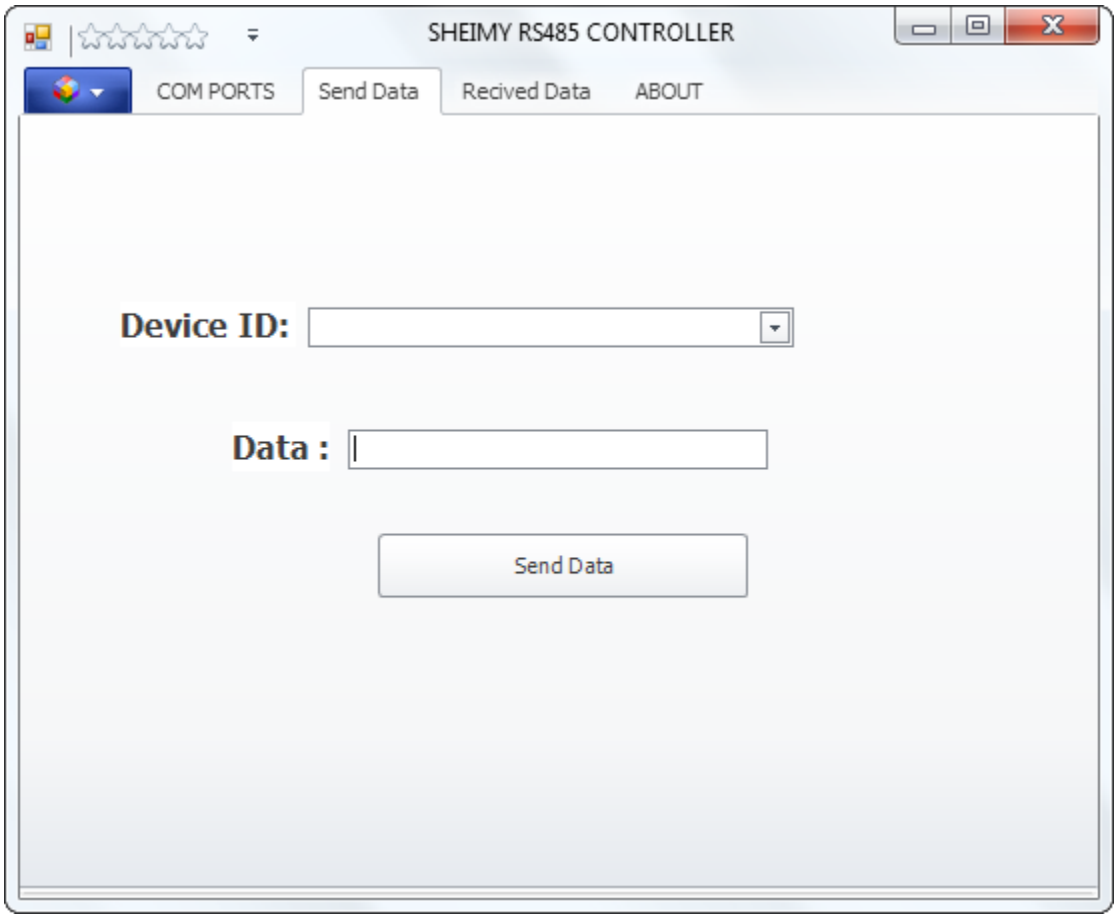


Figure 64 RS485 Transmitter/ Receiver(transmitter screen)

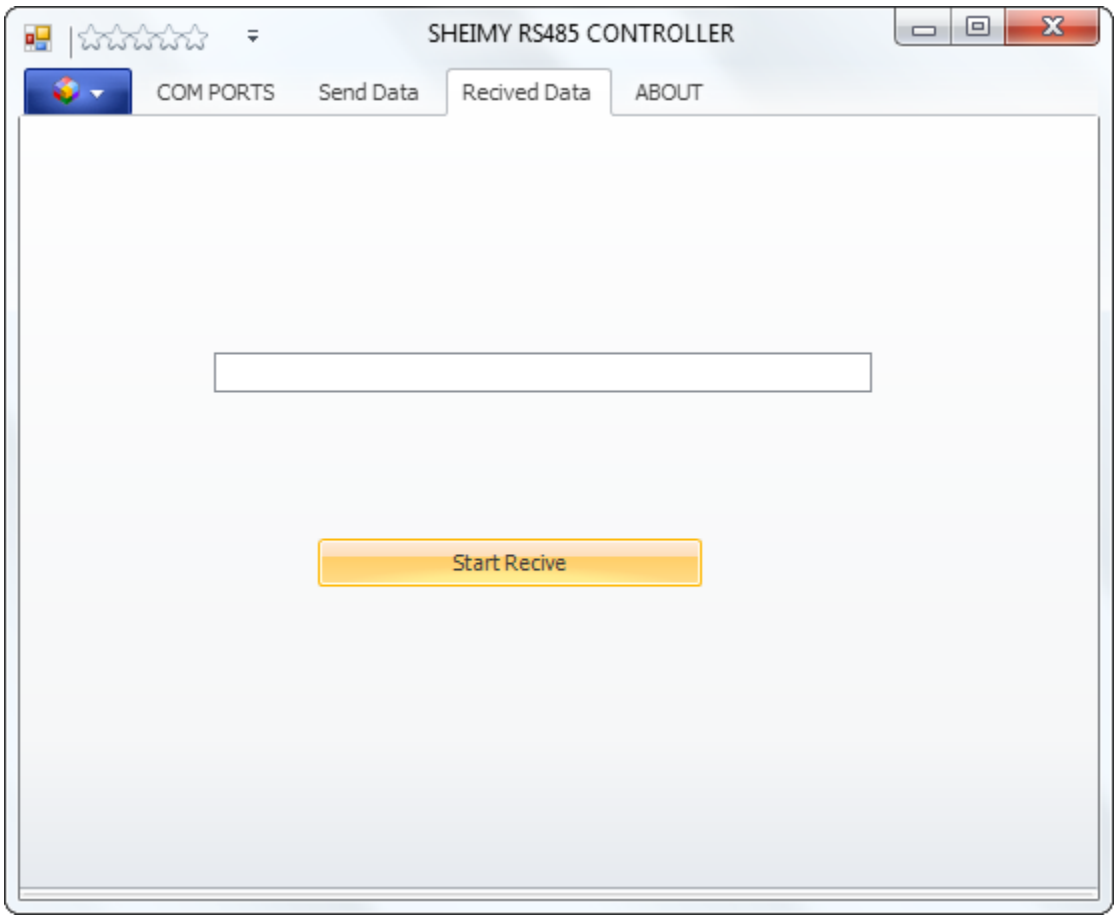


Figure 65 RS485 Transmitter/ Receiver (receiver screen)

4.3.4 X10 Transmitter/ Receiver

There was many tries to generate the X10 signal and to transmit it over the power lines the following is sample of this tries.

4.3.4.1 120 KHz carrier Generator

We start to generate a 120 KHz carrier generator using the PWM available in our PIC 16F877A

Desired PWM frequency is : 120 kHz,

Fosc = 20 MHz

TMR2 prescale = 1

1/120 kHz= [(PR2) + 1] • 4 • 1/20 MHz • 1

8.33 us = [(PR2) + 1] • 4 • 50 ns • 1

PR2 =40

The generated frequency will be 125 KHz

20/40 = 50% duty cycle

Initialize TIMR 2: setup_timer_2(T2_DIV_BY_1, 40, 1); // 125 KHz

Code:

```
#include <16F877A.h>
#fuses XT, NOWDT, NOPROTECT, BROWNOUT, PUT, NOLVP
#use delay(clock = 4000000)
byte const d[6]={1,0,1,0,1,0};
void main()
{
output_low(PIN_C2);    // Set CCP1 output low

setup_ccp2(CCP_PWM);  // Configure CCP2 as a PWM

setup_timer_2(T2_DIV_BY_1, 40, 1);  // 125 KHz

set_pwm2_duty(20);                    // 50% duty cycle on pin C1

while(1);
}
```

Code 26 120 KHz carrier Generator

Result:

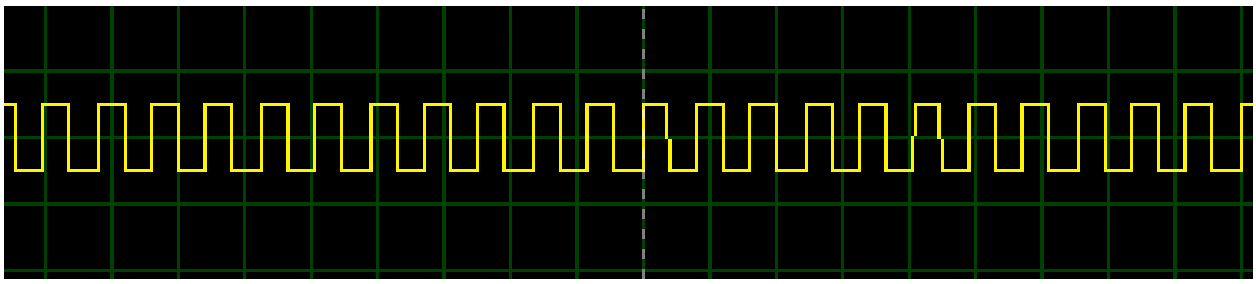


Figure 66 120 KHz carrier Generator

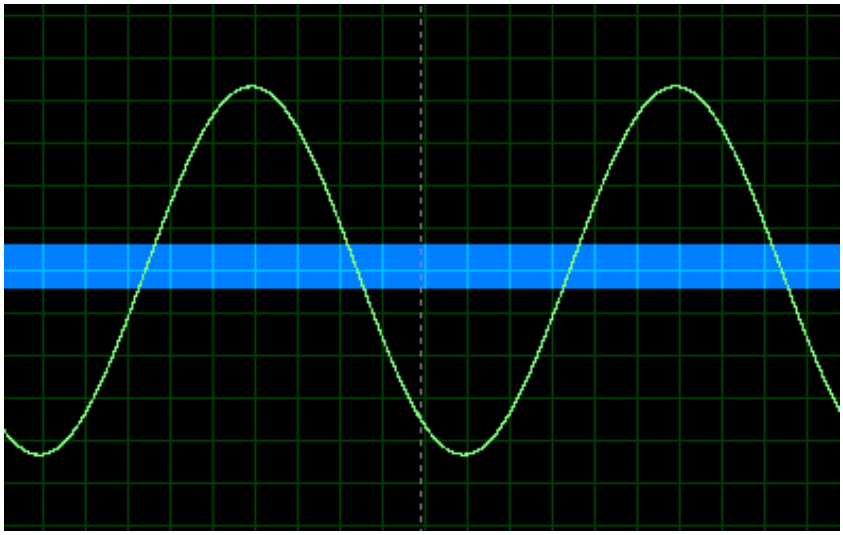


Figure 67 Together with 220 V 50 Hz AC

4.3.4.2 Tramitting X10 Signals

As a test we want to transmit the following array

Byte const d [6]={1,0,1,0,1,0};

The array will be transmitted as the following:

1-transmit '1': by the present of the 125 KHz generated signal

2-transmite '0': by the absence of 125 KHz signal

Code:

```
#include <16F877A.h>
#fuses XT, NOWDT, NOPROTECT, BROWNOUT, PUT, NOLVP
#use delay(clock = 4000000)
byte const d[6]={1,0,1,0,1,0};
void main()
{
output_low(PIN_C2);    // Set CCP1 output low

setup_ccp2(CCP_PWM);  // Configure CCP2 as a PWM

setup_timer_2(T2_DIV_BY_1, 40, 1);  // 125 kHz

set_pwm2_duty(20);                    // 50% duty cycle on pin C1

while(1)
{  // Prevent PIC from going to sleep  (Important !)
int i=0;
for(i=0;i<6;i++){
output_high(PIN_C2);    // Set CCP1 output low
if(d[i]==0){
SET_TRIS_C(0x02);
delay_ms(1);
}
else {
SET_TRIS_C(0x00);
delay_ms(1);
}
}
}
}
```

Code 27 X10 Data Transmitter

And this is the Result:

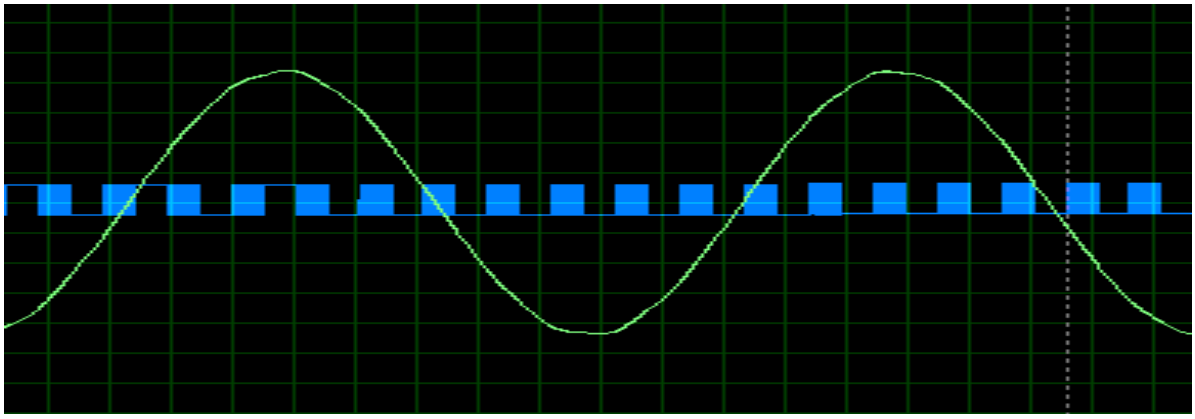


Figure 68 X10 Generated data

4.3.4.3 Merge data with the power signal

The following circuit was developed to merge data with the power signal every ZERO crossing

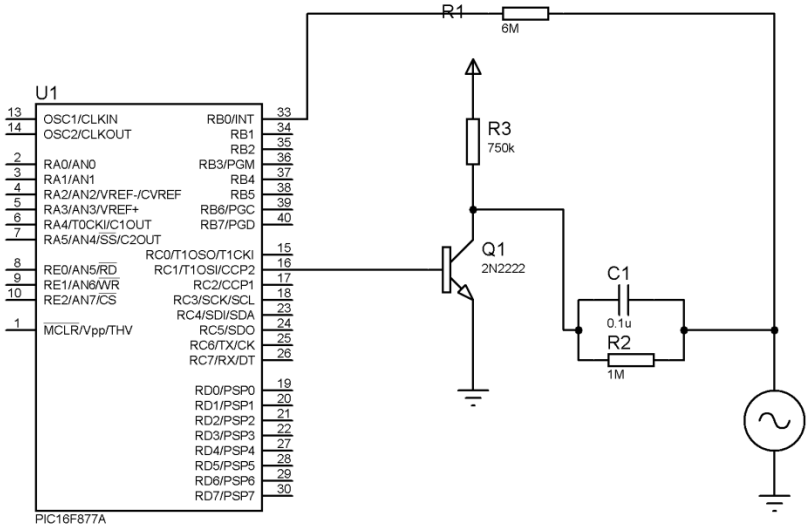


Figure 69 X10 Transmitter

The data transmitted over the power line was as the following:

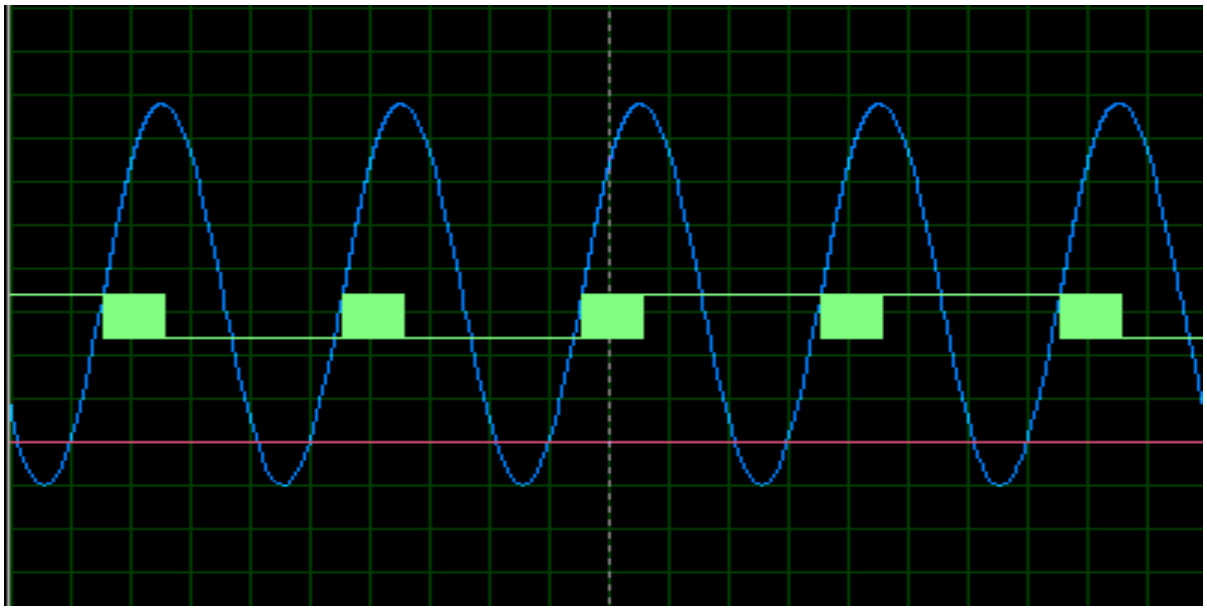


Figure 70 Data transmitted every Zero crossing

4.4 Step by step tutorial

4.4.1 How to connect the Hardware

4.4.1.1 Connect RS232 to RS485 converter

Connect the following board to the PC over the Serial port also it is tested by connecting it using USB to serial converter

Figure 71 Real R232 to RS485 converter PCB

4.4.1.2 connect RS485 receiver or Transmitter board

The First is to connect the power to the board and then connect the two wires of the max485 (A and B) that are responsible for the transmitting and receiving of the RS485 packet. This is like the following figure

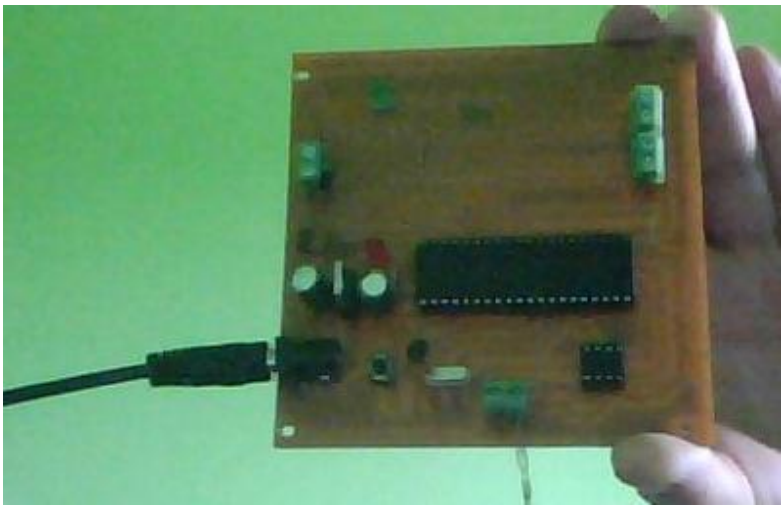


Figure 72 Real RS485 Transmitter PCB

4.4.1.3 connect light sensor

The laser led must be in front of the light sensor so we can indicate any thing that will cut the path between the laser led and the light sensor like that in the following figure

Figure 73 Light Sensor

4.4.2 How to Run the Software

The following is step by step tutorial of how to run the project software to send and receive data from the devices connected to the server PC

4.4.2.1 Server Software tutorial

1-

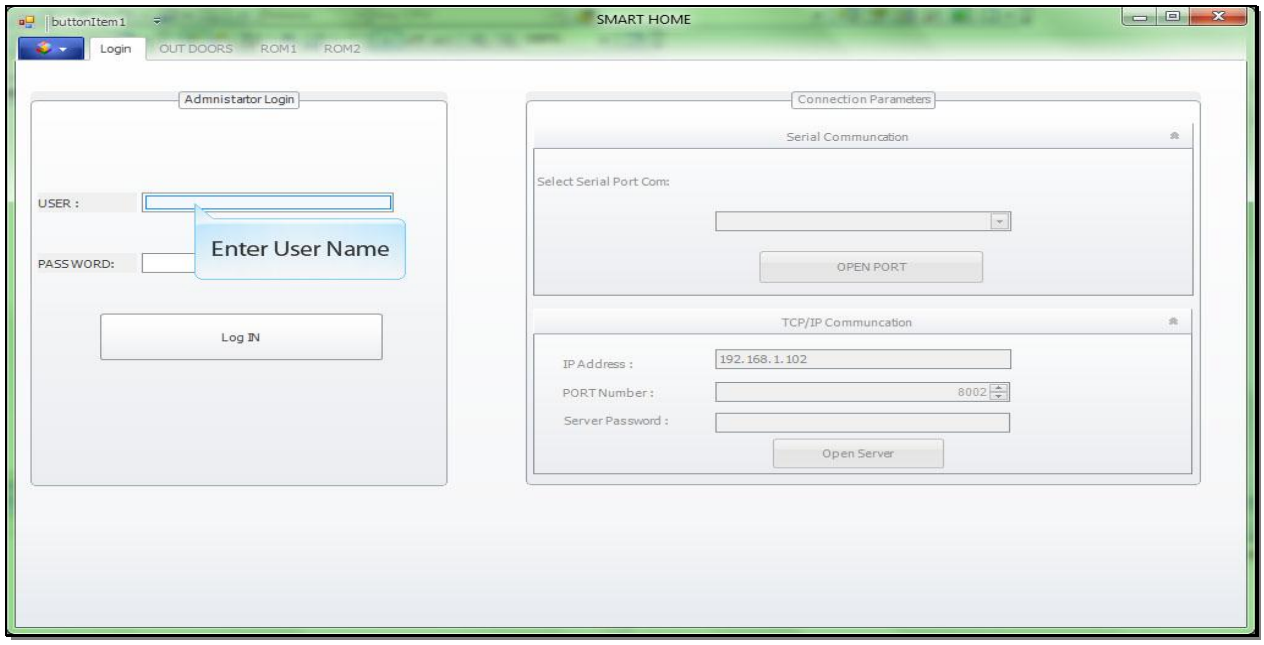


Figure 74 Enter User Name (login)

2-

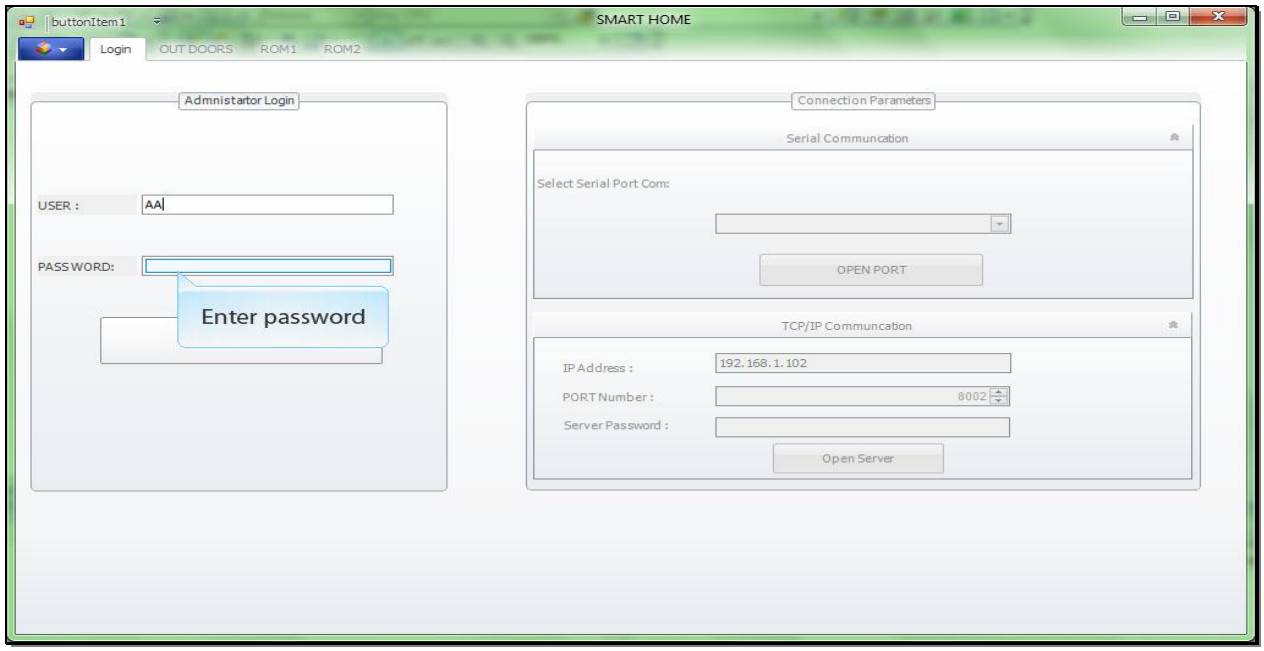


Figure 75 Enter password

3-

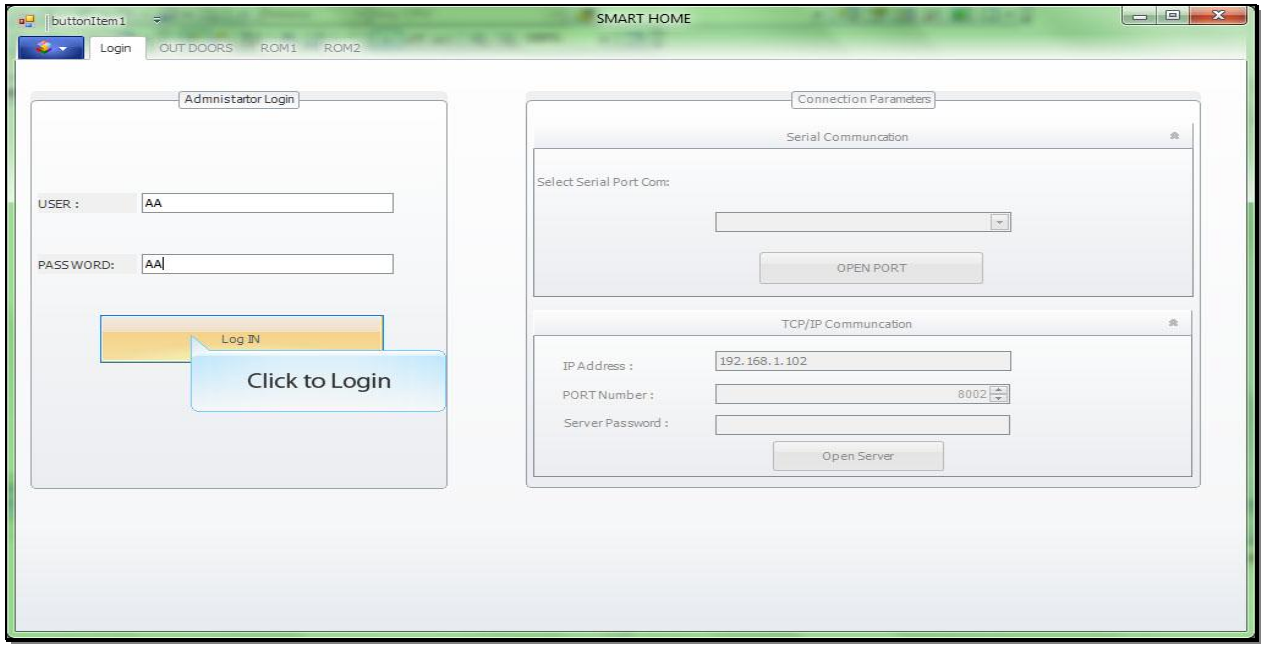


Figure 76 Click to Login

4- Select the Port ID

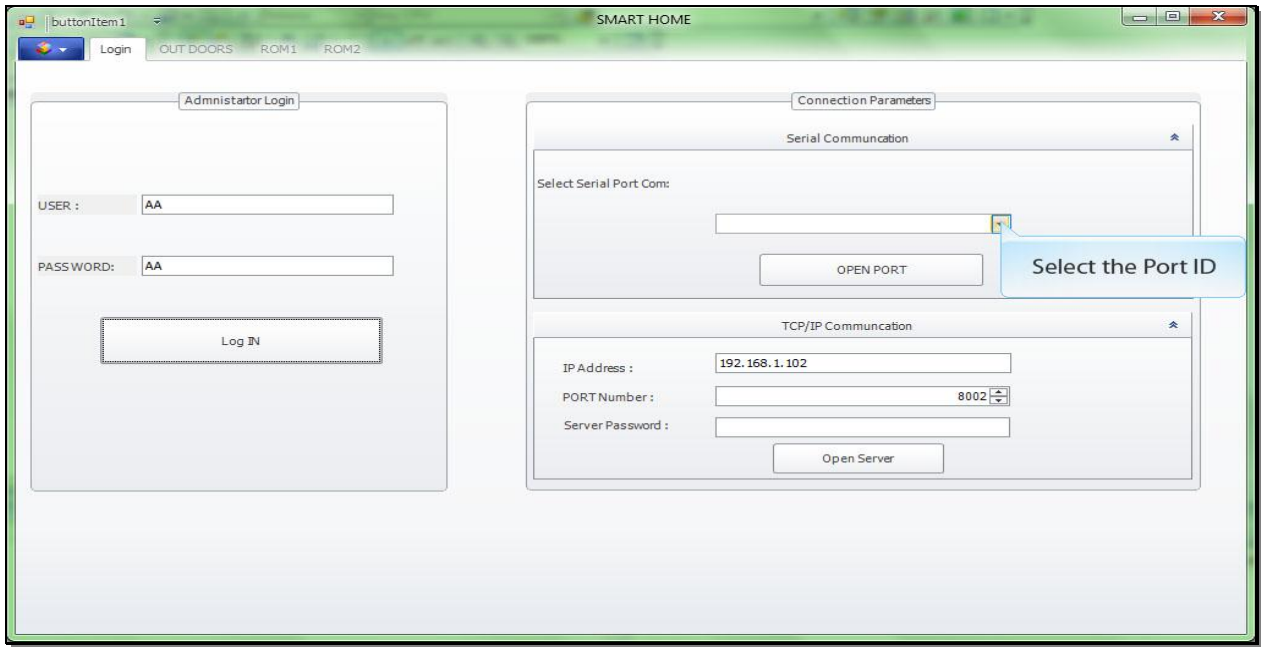


Figure 77 Select the Port ID

5- Click to OPEN PORT button

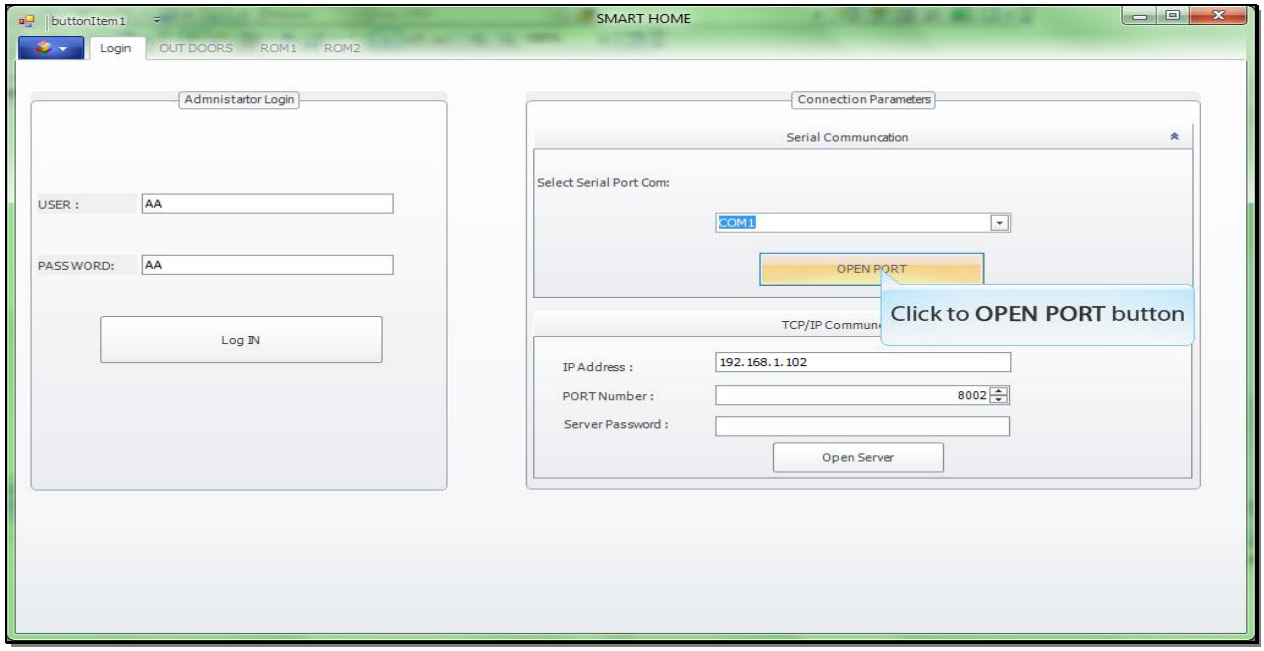


Figure 78 Click to OPEN PORT button

6- Click the OUT DOORS button

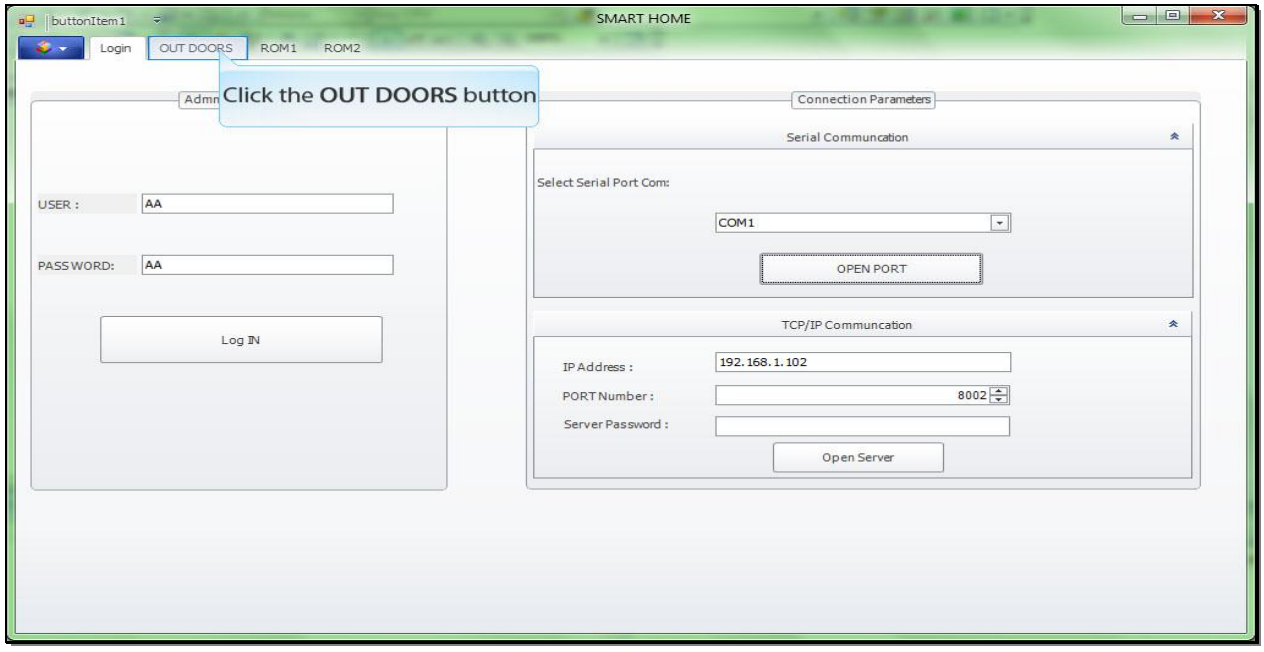


Figure 79 Click the OUT DOORS button

7- Click to search for camera1

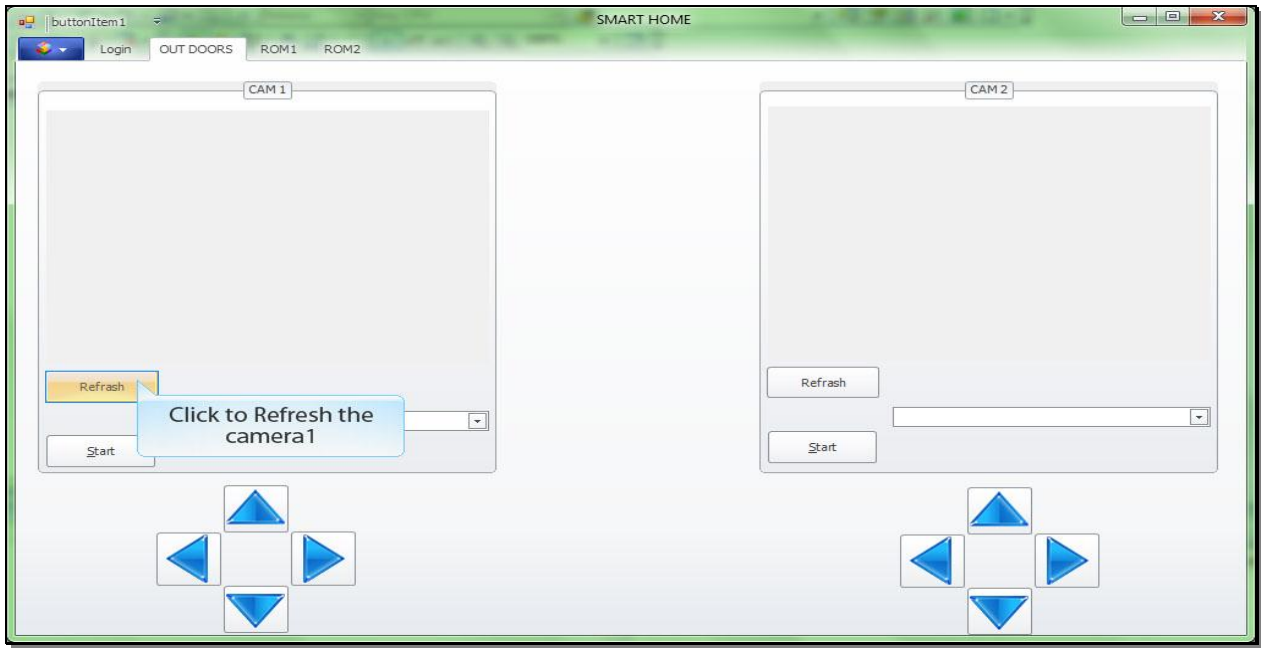


Figure 80 search for camera1

8- Select the Camera

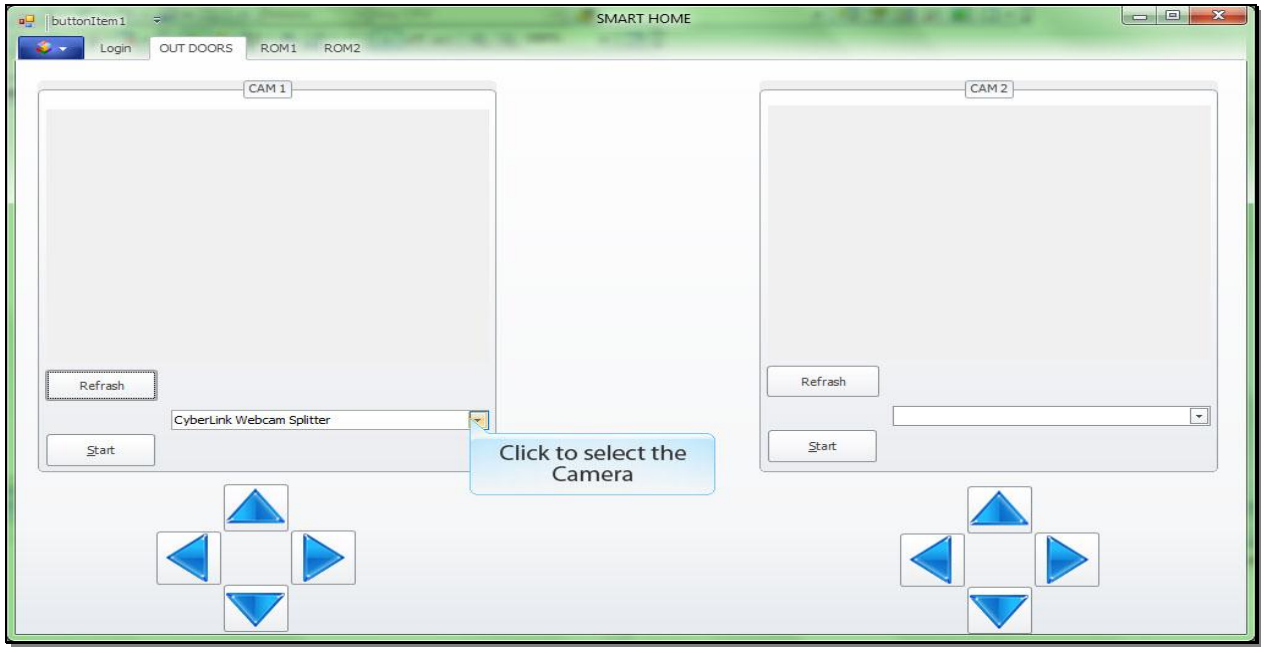


Figure 81 select the Camera

9- Start Camera

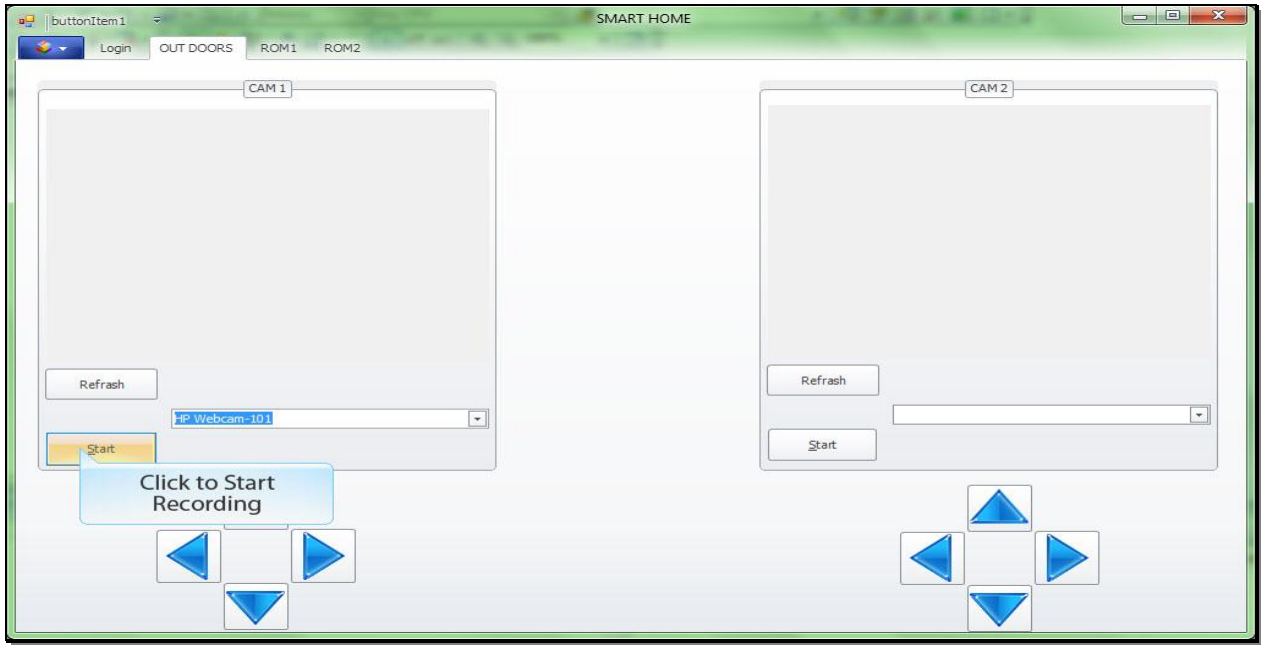


Figure 82 Start Camera

10-Stop Camera

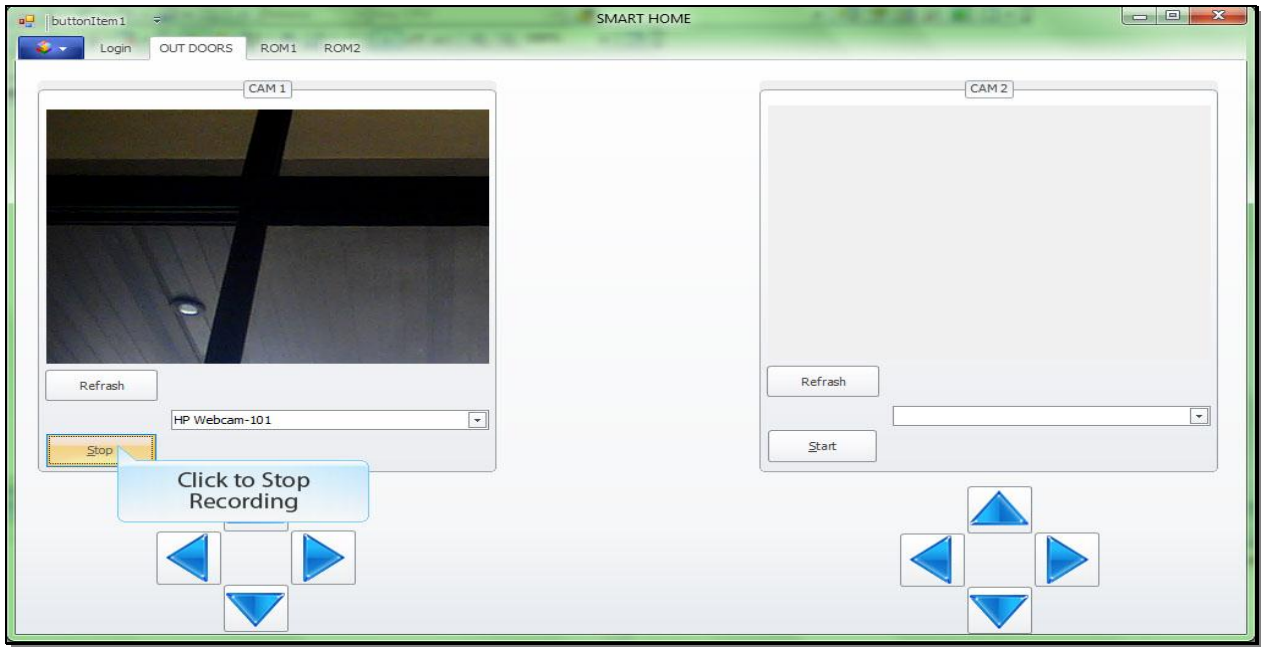


Figure 83 Stop Camera

11-control in camera angle

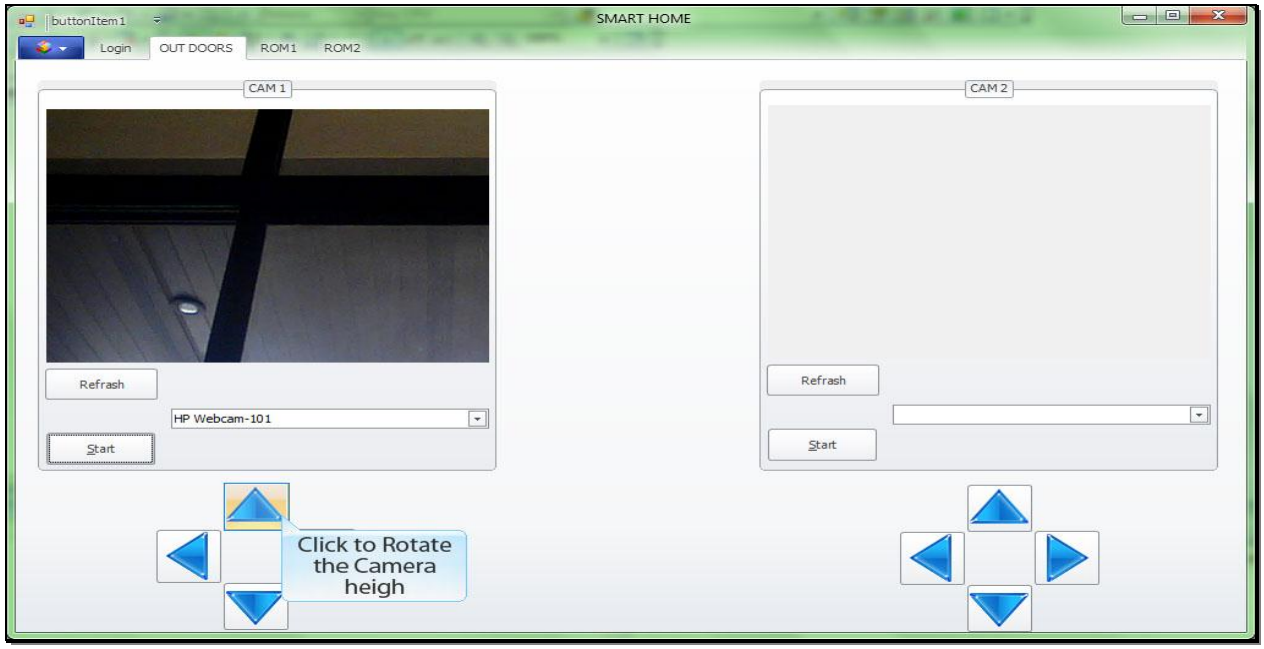


Figure 84 control in camera angle

12- Enter the Room 1 Control panel

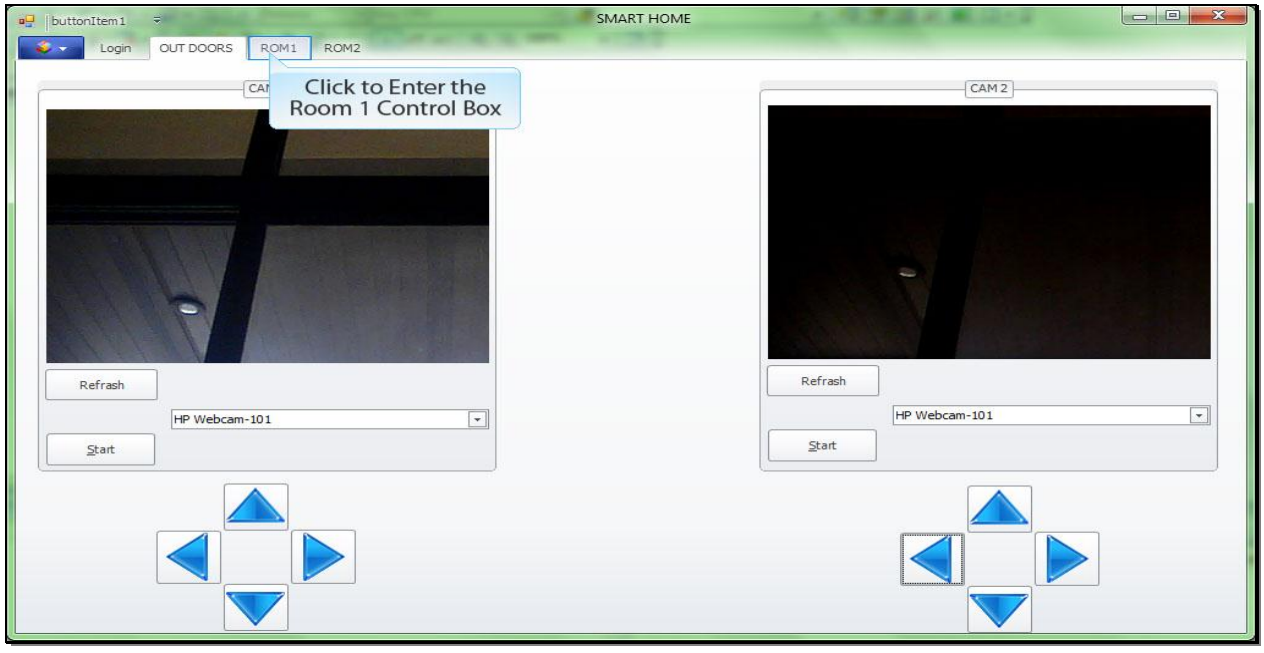


Figure 85 Enter the Room 1 Control panel

13- Turn on/off Specific Device

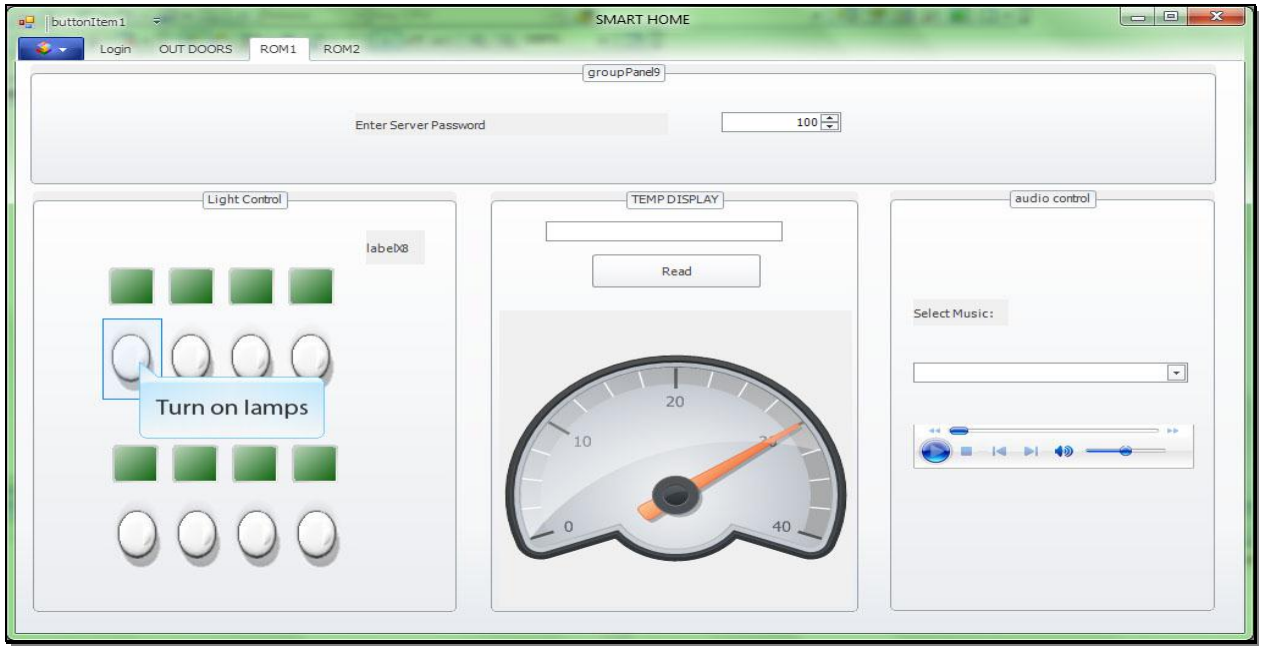


Figure 86 Turn on/off Specific Device

14- read the temperature degree

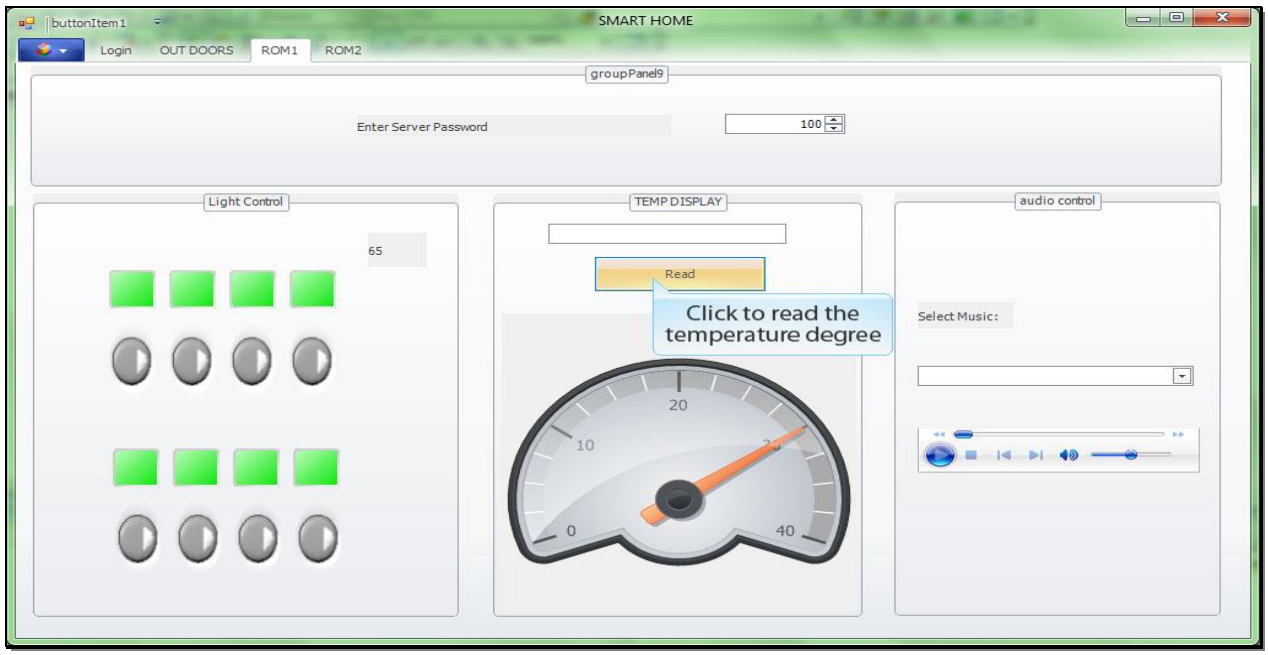


Figure 87 read the temperature degree

15- Select the track

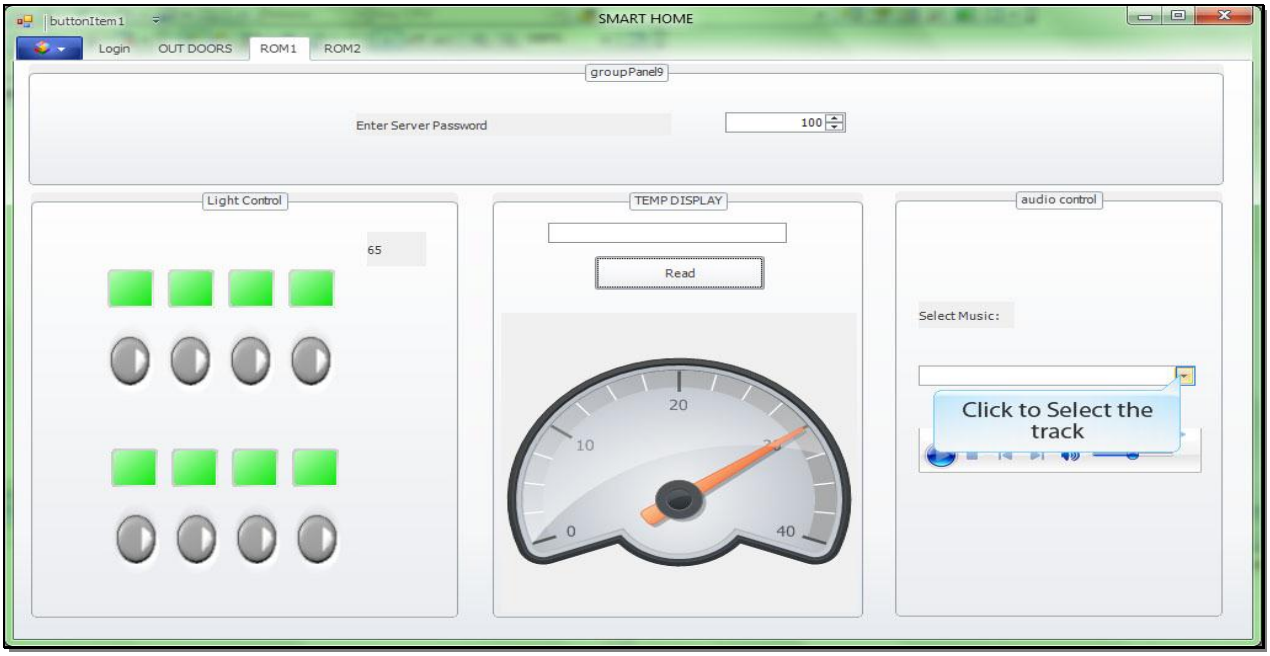


Figure 88 Select the track

16- Enter the Room 2 Control panel

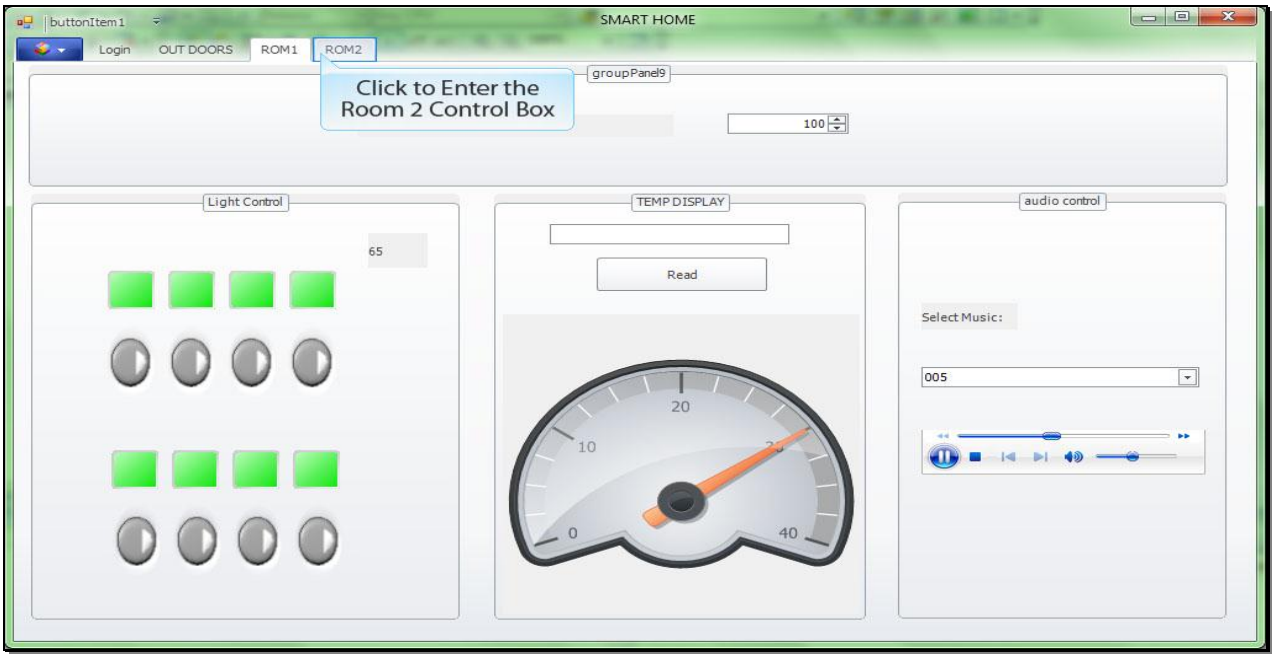


Figure 89 Room 2 Control panel

17- Turn on Device 3

Figure 90 Turn on Device 3

18- Change Room password

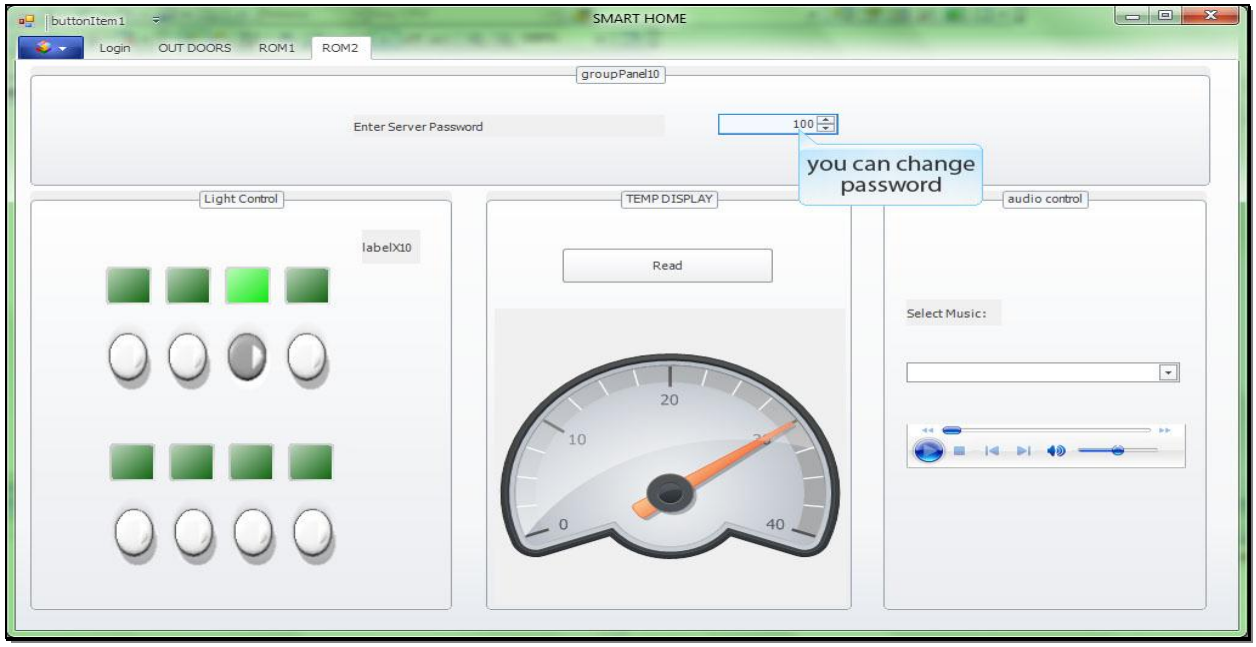


Figure 91 change Room password

19- Read Temp

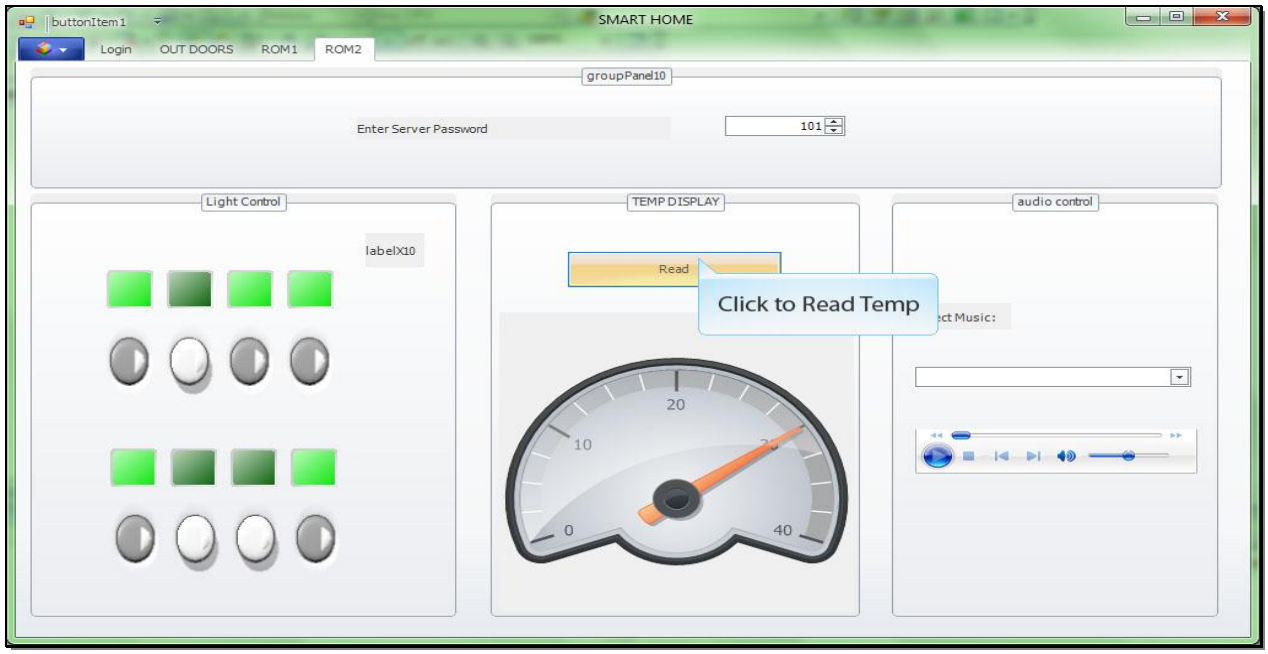


Figure 92 Read Temp

20- Back to Login

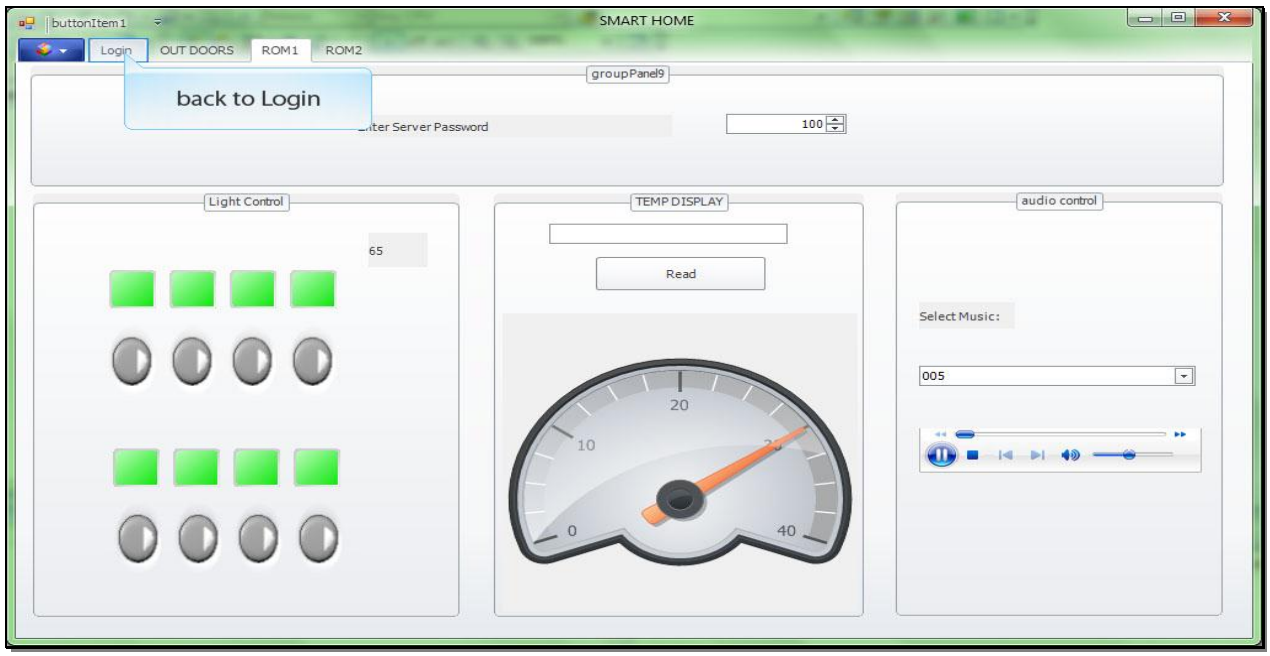


Figure 93 back to Login

21- Open TCP/IP Server

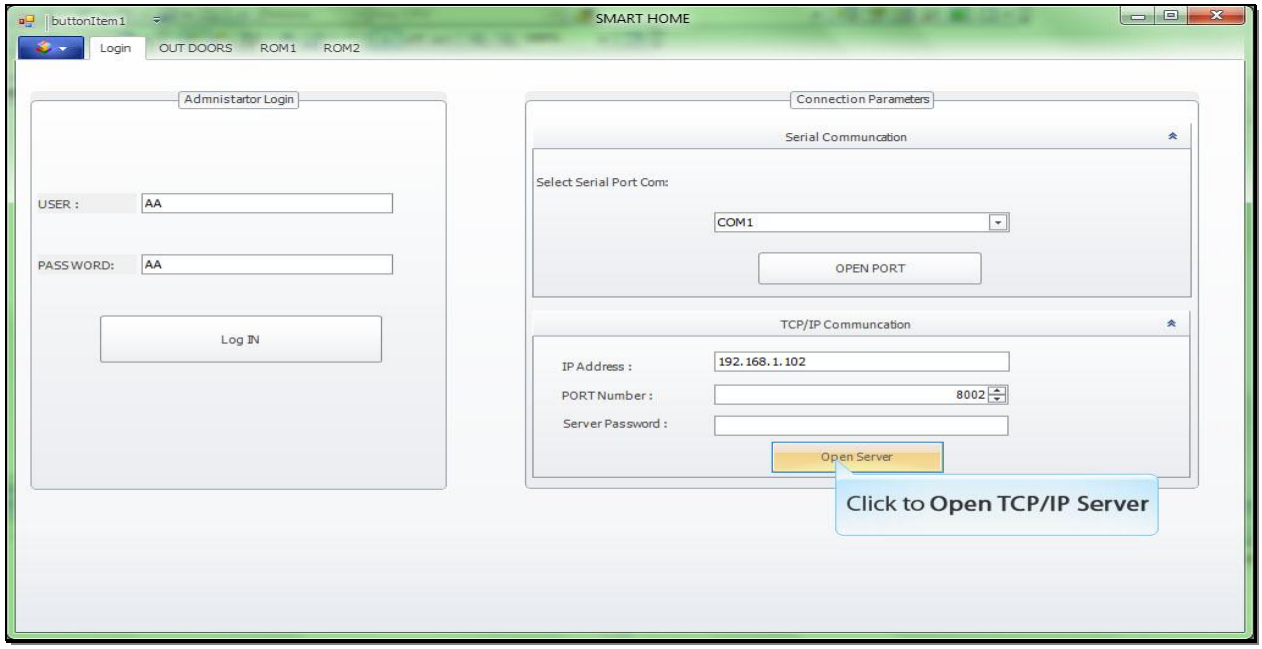


Figure 94 Open TCP/IP Server

22- The TCP Server window

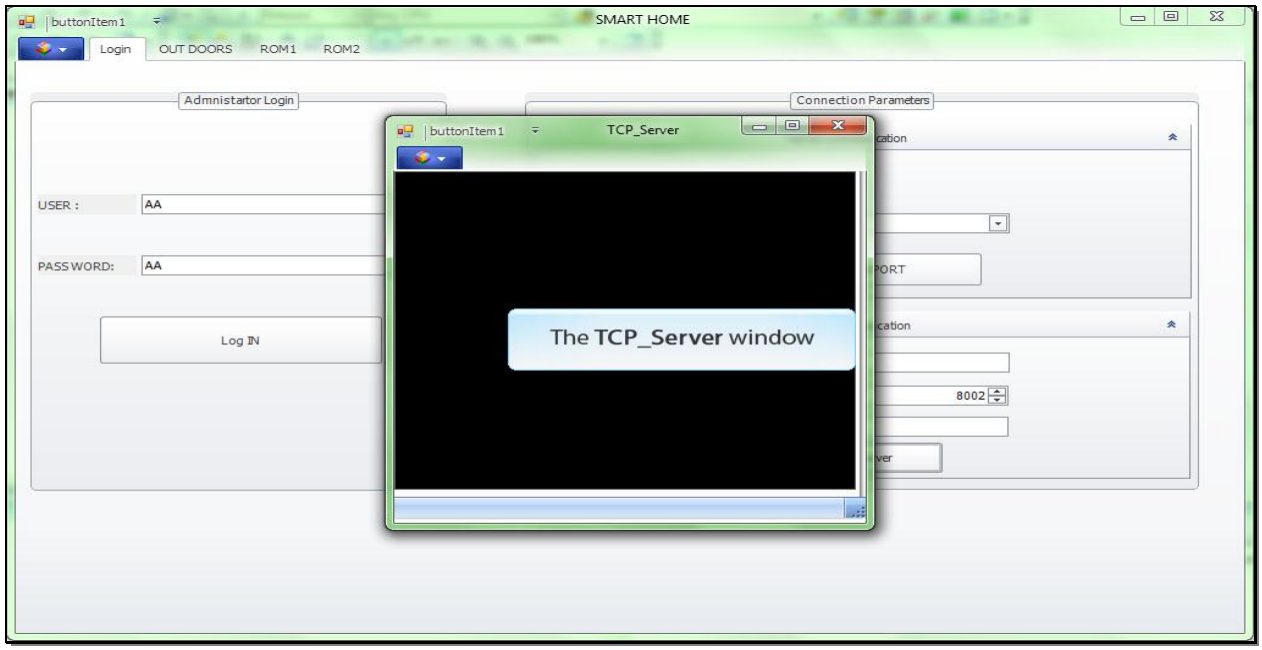


Figure 95 The TCP Server window

4.2.2 Client Software tutorial

1- Enter User Name

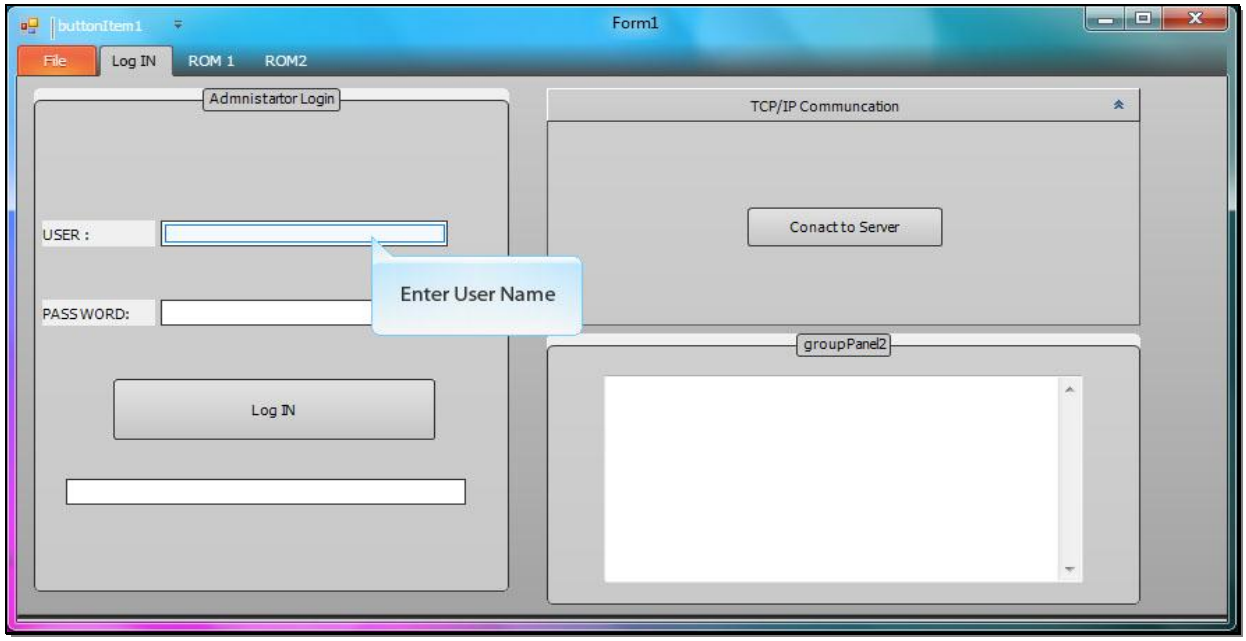


Figure 96 Enter User Name

2- Enter Password

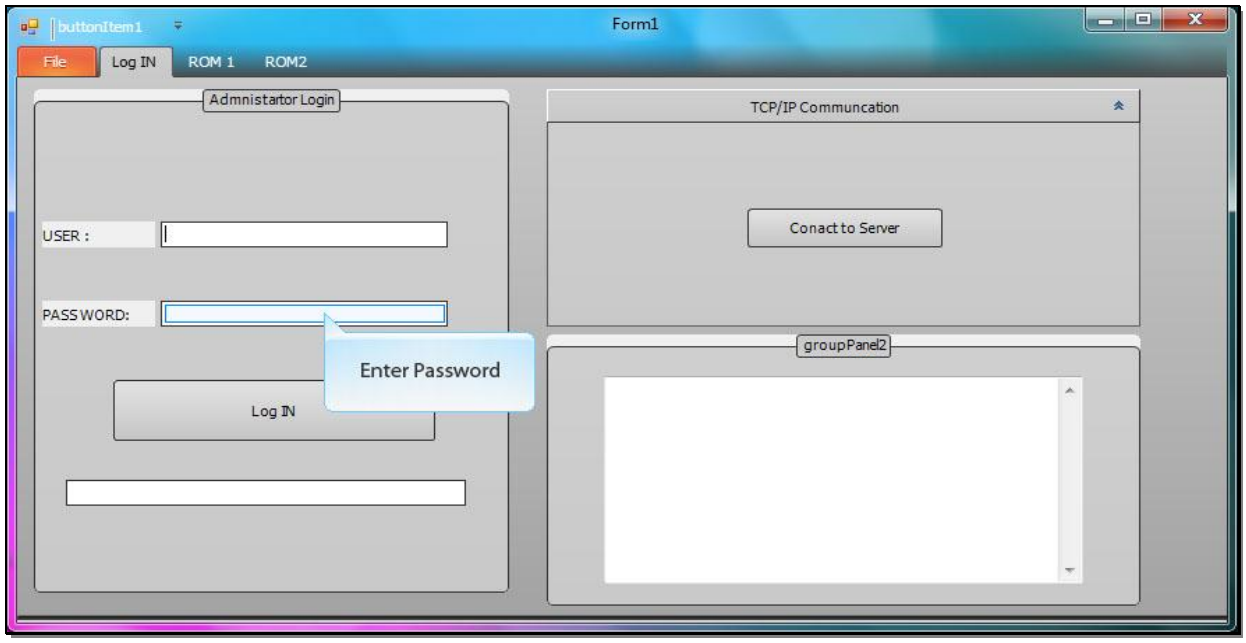


Figure 97 Enter Password

3- Connect to Server

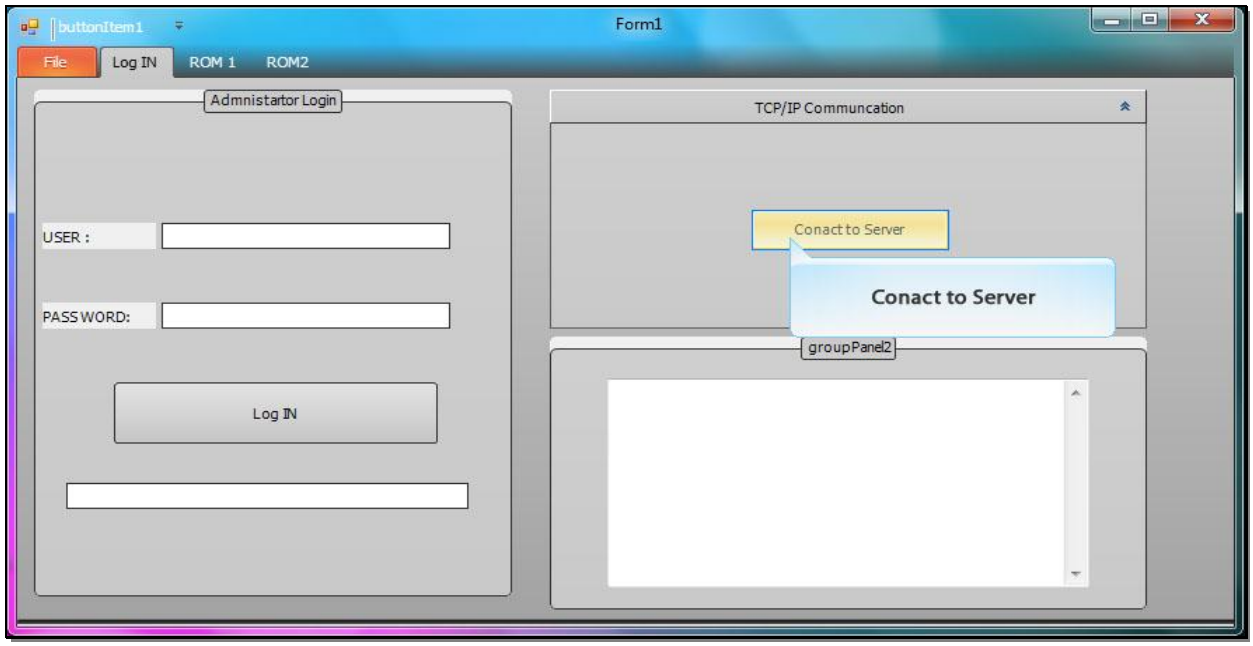


Figure 98 Connect to Server

4- The LoginInfo window opens

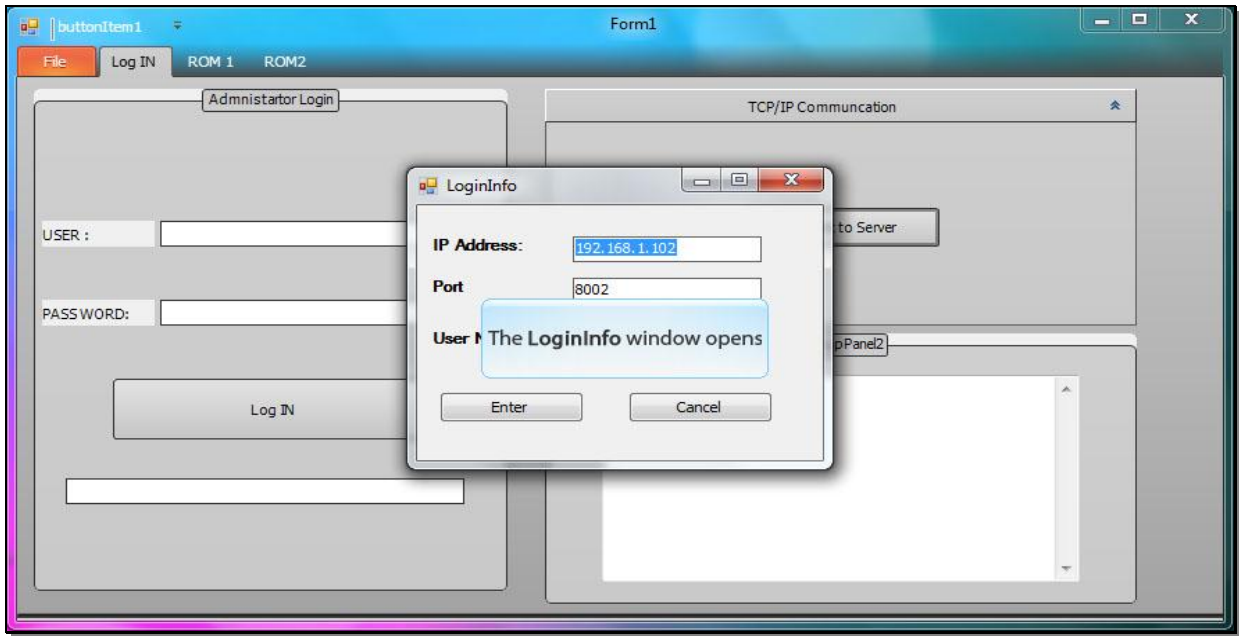


Figure 99 The LoginInfo window opens

5- Get the IP address of the Server

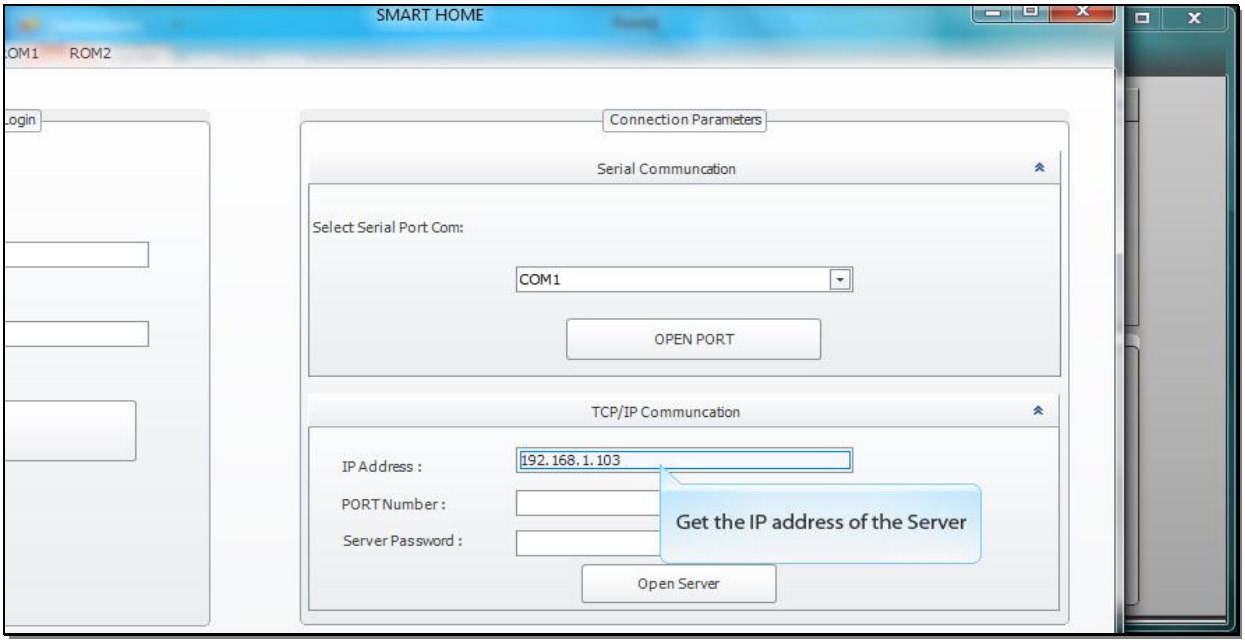


Figure 100 IP address of the Server

6- Enter the IP address of the Server

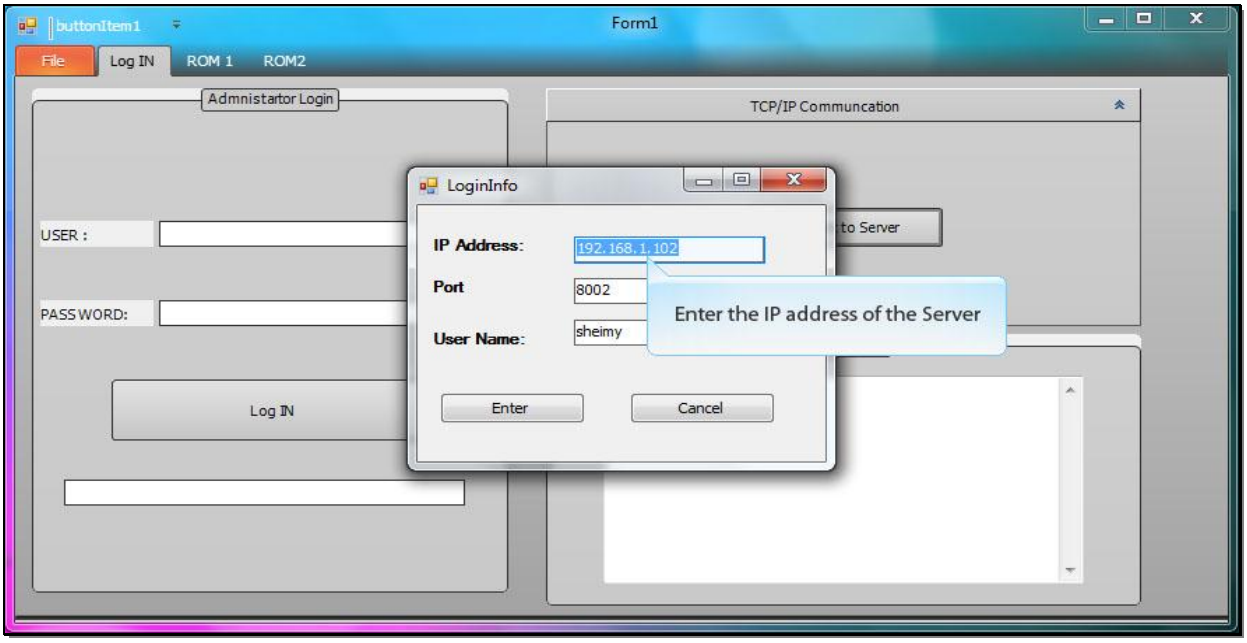


Figure 101 IP address of the Server

7- Enter to connect to server

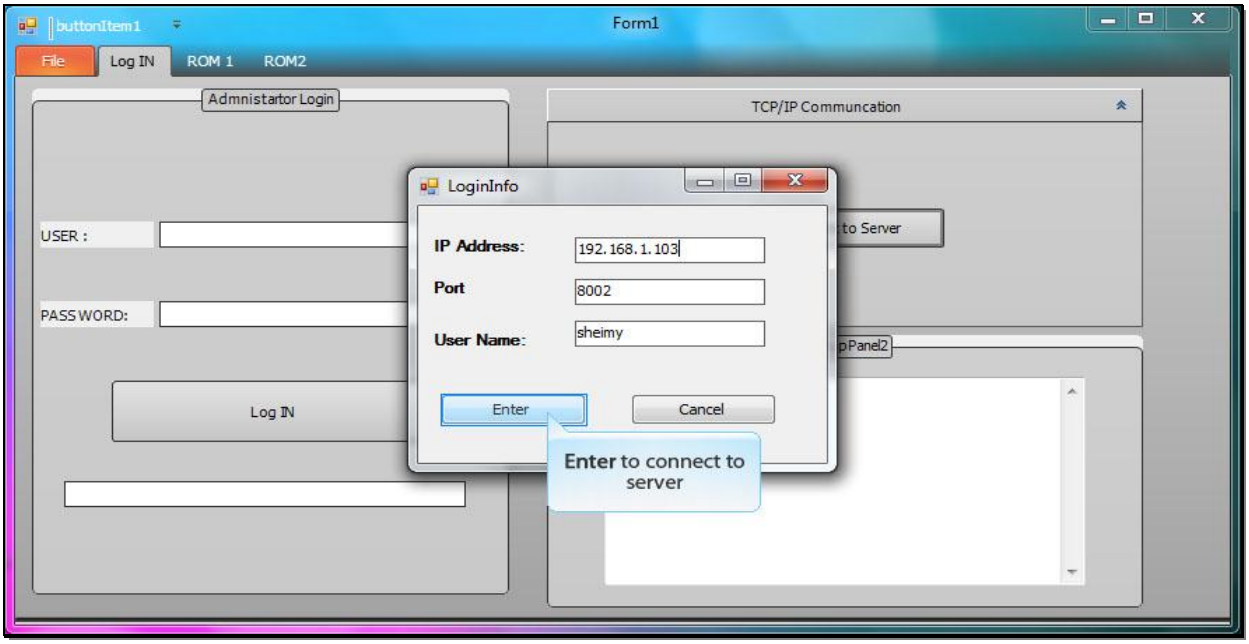


Figure 102 connect to server

8-Server window in Server PC indicate the connected client

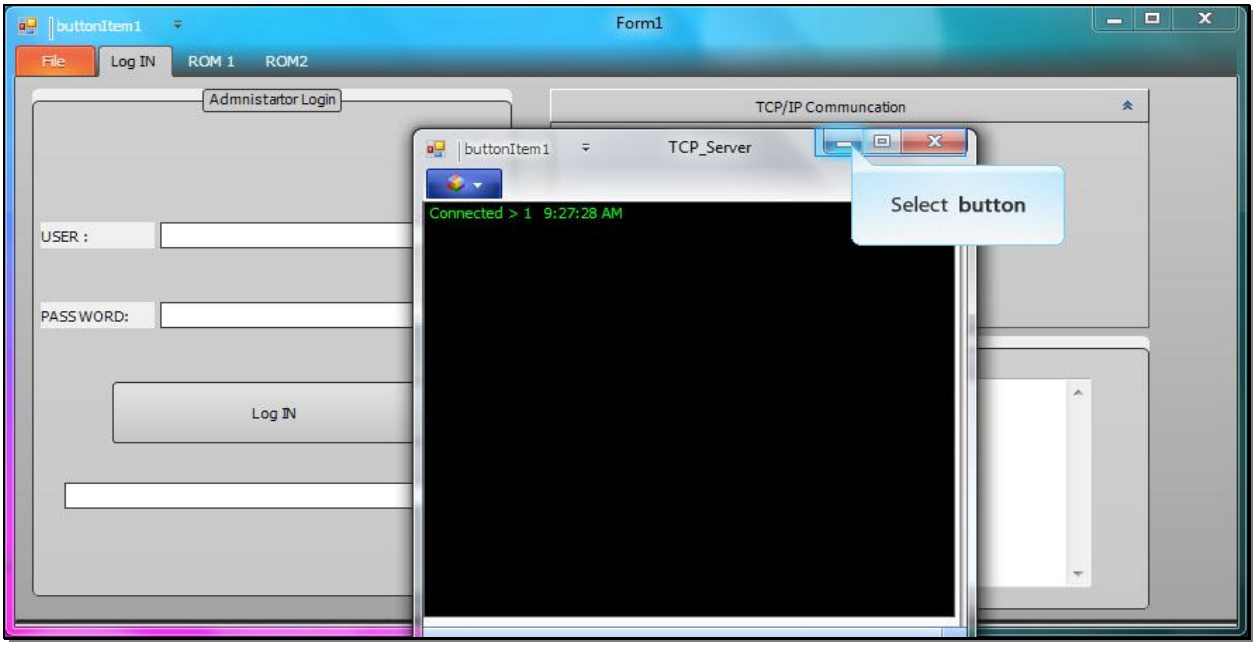


Figure 103 Server PC indicate the connected client

9- ENTER ROM 1 Control panel

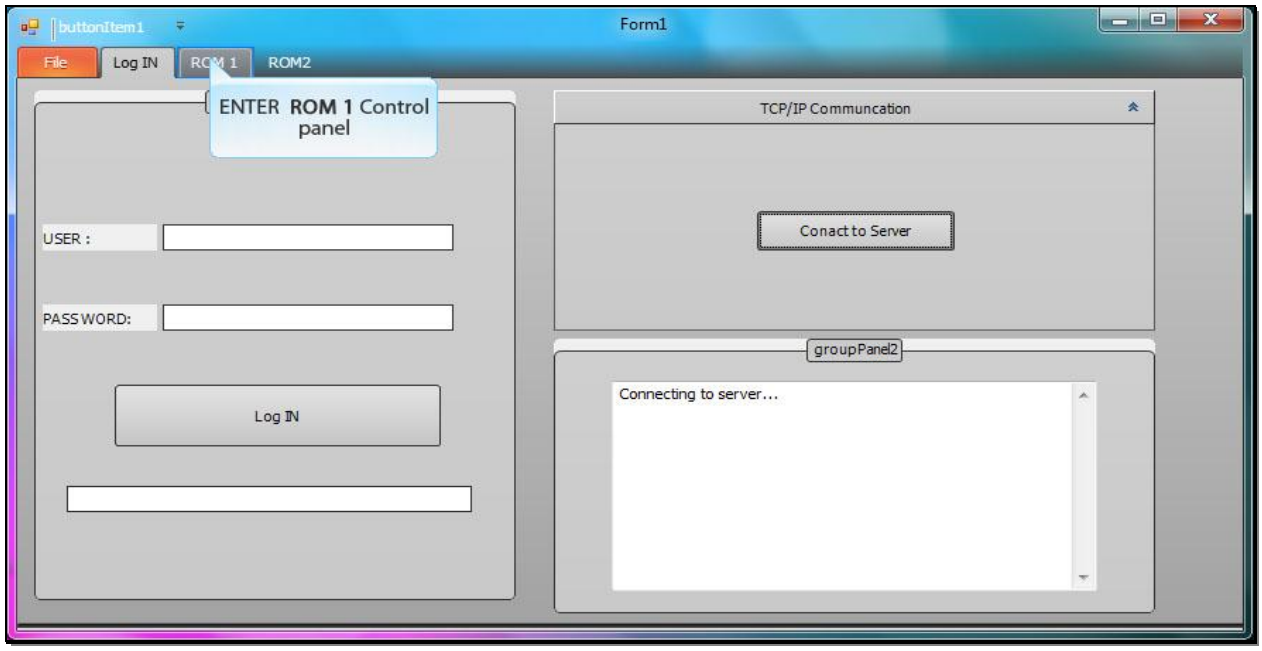


Figure 104 ROOM 1 Control panel

10- Enter the Room password

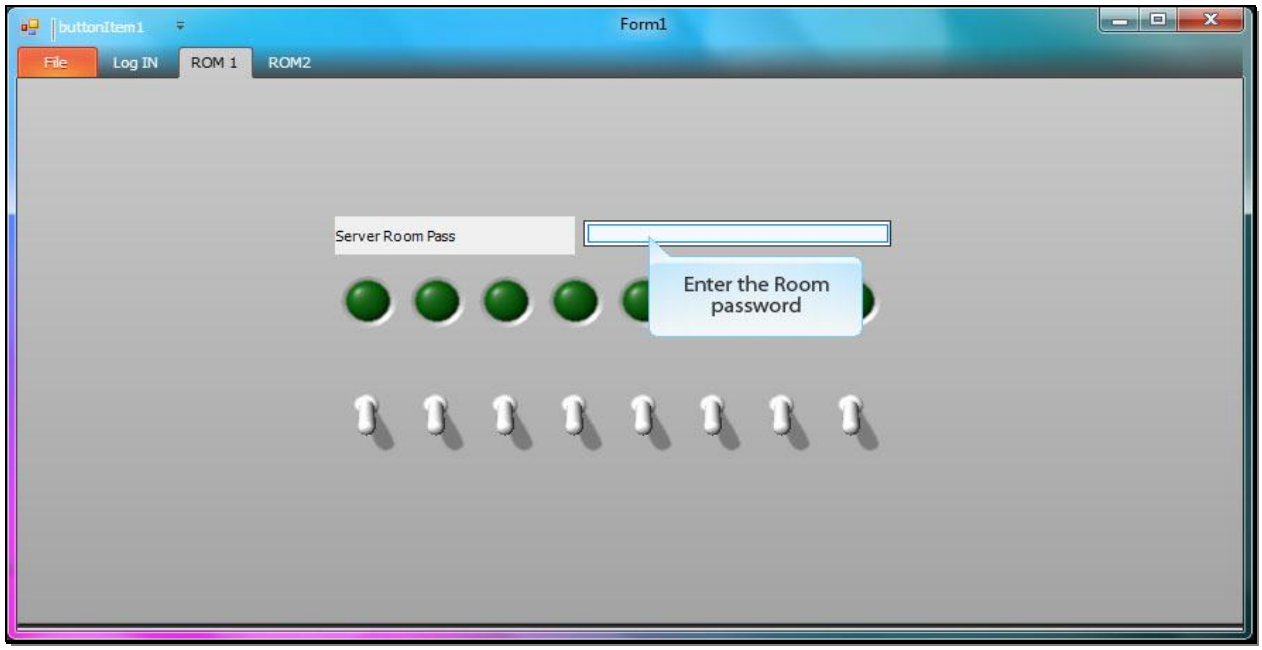


Figure 105 Room password

11- Now you can control in Devices

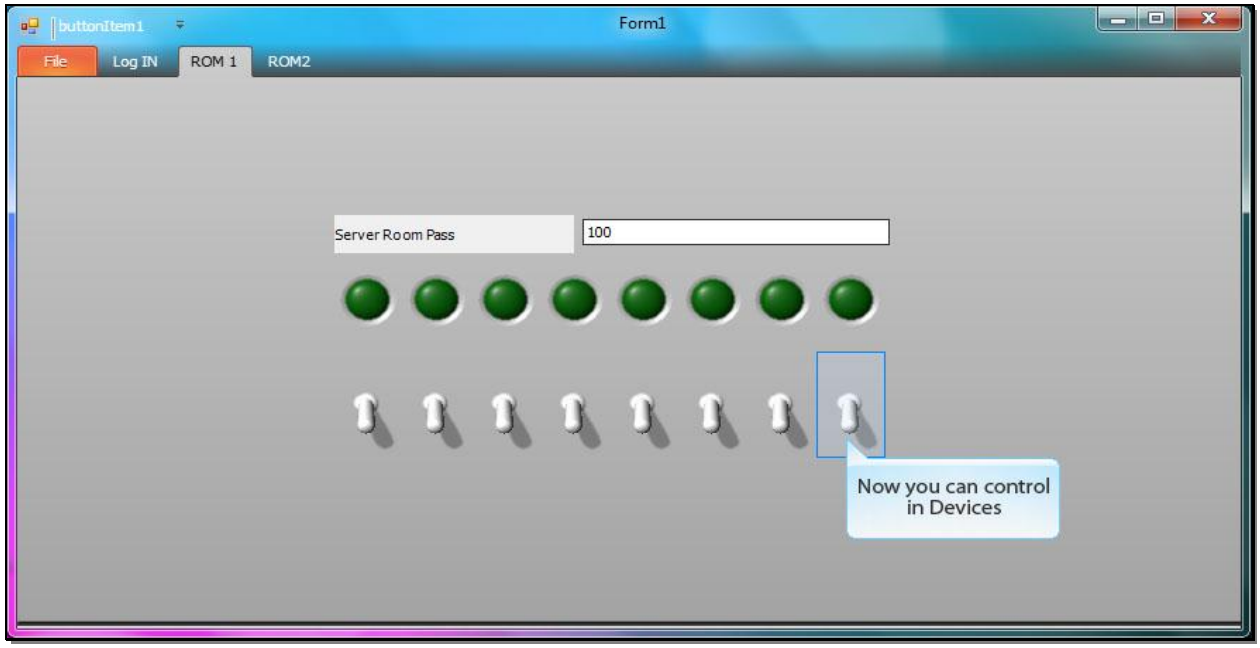


Figure 106 control in Devices

12- ENTER Room 2 Control Panel

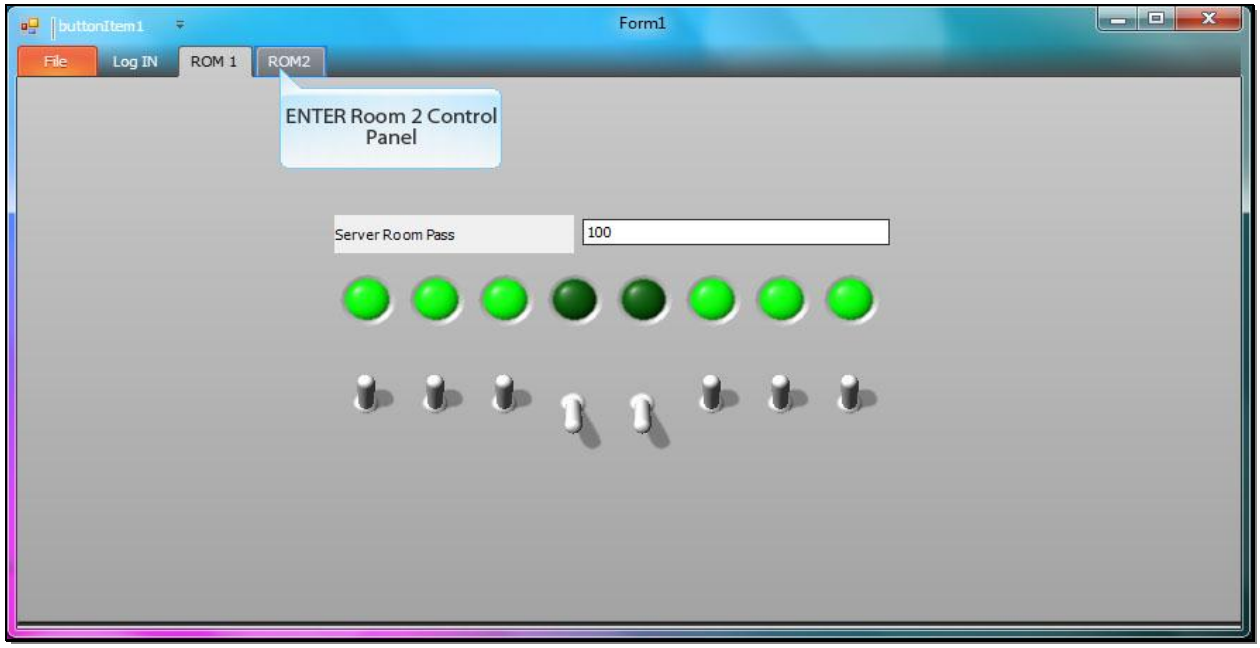


Figure 107 Room 2 Control Panel

Glossary

PLC	Power Line Communication
TX	Transmitter
RX	Receiver
X10	Zero crossing protocol
TW	Two-way
RS485	Recommended Standard 485
RS232	Recommended Standard 232
Ack	Acknowledgement
PCB	Printed Circuit Board
PWM	Pulse Width Modulation

Bibliography

Burroughs, J. (2002). X-10 Home Automation Using the PIC16F877A. Microchip Technology Inc.

embededtronic.com. (n.d.). Connecting Microcontrollers to Nokia 3310 . http://www.mytutorialcafe.com/.

Embedtronics. (n.d.). Nokia F-Bus Protocol. http://www.embedtronics.com/.

Falquez, J. (Spring 2009). X-10 Protocol & Power Line Communication. The George Washington University.

FAZELA M. VOHRA, M. C. (n.d.). Power Line Carrier Communications. K.J.Somaiya College of Engg.

LABABIDI, S. (1998). Mobile Home Automation .

POWERLINE COMMUNICATION-Using X10 Protocol. (n.d.).

PRO, X.-1. (n.d.). X-10 Communications Protocol. X-10 PRO.

smarthome.com. (n.d.). Smart Home. www.smarthome.com.

mikroe.com.(n.d).Mikroelektronika.www.mikroe.com

c-sharpcorner.com.(n.d). c-sharpcorner forum.www. c-sharpcorner.com

Tony Northrup, c.-a. o. (2005). Introduction to X10 Home Automation Technology. http://oreilly.com/.

Appendix A: X10 tools

X10 is just one of many home automation technologies used in my latest book, Home Hacking Projects for Geeks. The book will guide you step by step through the process of adding common smart home features to your home.

I. What Can You Do with X10?

X10 is a powerful, flexible, and (mostly) inexpensive technology. With X10 technology and a little creativity, you can accomplish the following things:

A. Add a light switch to any wall without running any wires.
B. Control a lamp or built-in light with your computer.
C. Use a television at your home or a computer on the Internet to monitor multiple, inexpensive video cameras around your home.
D. Turn off the power outlets in your kid's bedroom between 4 p.m. and 6 p.m. to make sure he or she is studying and not playing a game or watching TV.
E. Build a custom security system that sounds an alarm if an intruder opens a window, or sends an email to your mobile phone if there is a water leak in your basement.

Figure 108 An X10 lamp module turns a lamp on and off when an X10 signal is sent from an X10 transmitter

II. X10 Addressing

To identify individual devices and groups of devices, X10 uses an addressing scheme that provides up to 256 unique addresses. House codes are written as a single letter in the range A-P. Unit codes are a decimal number between one and 16. Examples of valid house codes are A1, J13, and P16.

Note: If you're a network geek, think of the house code as the network portion of an IP address, and the unit code as the host portion.

Unlike the IP addresses used on the Internet, X10 addresses do not have to be unique. You should give a single address for each group of X10 devices that you would like to respond to the same command. For example, if you want to turn on two lamps with a single switch, connect an X10 lamp module to each lamp and configure both modules with a single address. If you want all of the lamps in a room to be controlled by a single command, they should all be assigned a single address.

While most X10 devices are one-way (because they are only capable of either sending or receiving), some devices are two-way. For example, one-way X10 light switches can receive X10 commands to enable them to be turned on and off remotely. You can also use the one-way light switch to control the light locally, just as you control a conventional light switch. However, when you flip the switch, a one-way X10 light switch does not transmit a signal. Therefore, while flipping the switch can turn the light on and off, it cannot turn on other X10 switches.

Two-way X10 light switches can receive X10 commands, and can also transmit an X10 command when you flip the switch. This allows you to use the switch to control both the light and another X10 device simultaneously. For example, if I replace the switch that controls my kitchen's under-cabinet lighting with a one-way X10 switch, and then replace the switch that controls my kitchen's overhead lighting with a two-way X10 switch, I could turn on both the overhead light and the under-cabinet light by using the overhead light switch.

Appendix B: RS485 Protocol

start byte	address byte	number of data bytes	first data byte	second data byte	third data byte	redundancy check (CRC)	End Byte

Figure 109 RS485 frame

I. start byte

Is the first byte in the packet which is always is equal 0X96

II. address byte

The address of the device this byte can take the value from 0 to 255 but it can't take the value 50 decimal which is used for broadcast

III. number of data bytes

This byte indicate the number of data bytes being transmitted from the slave and the number of data being transmitted from the M aster plus 128

IV. first data byte

This is the first byte of data being transmitted

V. second data byte

This is the second byte of data to be transmitted

VI. third data byte

This is the third byte to be transmitted

VII. redundancy check (CRC) byte

The algorithm for calculating the CRC is also given as;

CRC = NOT ($aa XOR $bb XOR $dd [XOR $dd XOR $dd])

Where;

$aa = one byte address

$bb = one byte showing number of data bytes (slave) and 128+number of bytes(master)

$dd = one to three data bytes (depending on what was put in RS485 send command)

$cc = cyclic redundancy check (CRC) byte

ie XOR $aa, $bb and all $dd bytes then invert all the bits in the answer then if the answer is $96 or $A9 add one

Appendix C: TCP/IP Protocol

I. Overview

Transmission Control Protocol (TCP) is an upper-layer protocol from the IP point of view. The first question that always occurs to a beginner is "Why do we need two protocols, IP and TCP?"

While IP transmits data between individual computers on the Internet, TCP transfers data between two actual applications running on these two computers. IP is used for data transfers between computers. An IP address is the address only of a computer's network interface, while TCP uses a port number as its address. If we were to compare this to a standard postal system, the IP address would be the building address and the port number (the address in TCP) would be the name of an actual resident in the building.

TCP is connection oriented. In other words, this is a service that establishes a connection between two applications, i.e., creates a virtual circuit for the time of connection. This is a full duplex circuit; data is simultaneously transferred in both directions independently as shown in Figure. The transferred bytes are numbered. Lost or damaged data is requested again. The integrity of the transferred data is ensured by a checksum.

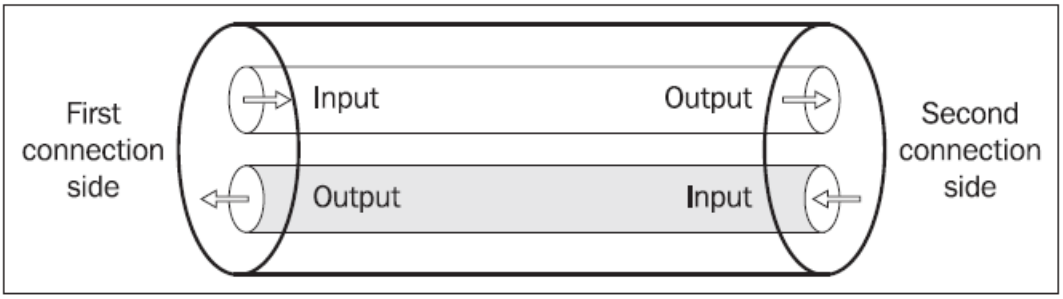


Figure 110 TCP creates a fully two-way link between the ends of the connection

In other words, an application that uses TCP does not have to worry about data getting lost during transfer or being modified by a transfer error. This safeguard is only effective against technical errors. It does not attempt to protect data from intelligent attackers, who could modify the data and also recalculate the checksum.

A TCP segment is inserted into an IP datagram. IP datagrams are inserted into a link frame. If the size of the TCP segment is too big to be entered into an IP datagram without exceeding the maximum capacity of the link frame (MTU), the IP has to perform fragmentation on the TCP datagram. See Figure 55

Fragmentation increases overhead, which is why we try to create segments that are not long enough to require fragmentation. Note that the TCP header is transported in the first IP fragment only.

II. TCP/IP Layering

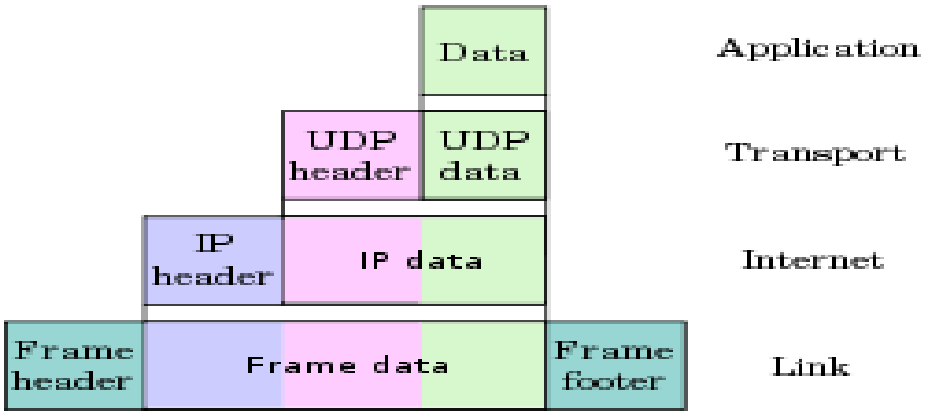


Figure 111 Encapsulation of data

1. The link layer: sometimes called the data-link layer or network interface layer normally includes the device driver in the operating system and the corresponding network interface card in the computer. **IEEE 802.3** is a collection of IEEE standards defining the Data Link Layer's media access control (MAC) sub layer of wired Ethernet
2. The network layer: sometimes called the internet layer handles the movement of packets around the network. Routing of packets.
3. The transport layer provides a flow of data between two hosts, for the application layer above. In the TCP/IP protocol suite there are two vastly different transport protocols: TCP provides a reliable flow of data between two hosts. It is concerned with things such as dividing the data passed to it from the application into appropriately sized chunks for the network layer below, acknowledging received packets, setting timeouts to make certain the other end acknowledges packets that are sent, and so on. Because this reliable flow of data is provided by the transport layer.
4. The application layer handles the details of the particular application. There are many common TCP/IP applications that almost every implementation provides.

III. IEEE 802.3 Ethernet Frames

Bits flowing across the Ethernet are grouped into structures called frames. A frame must be between 46 and 1500 octets in size.

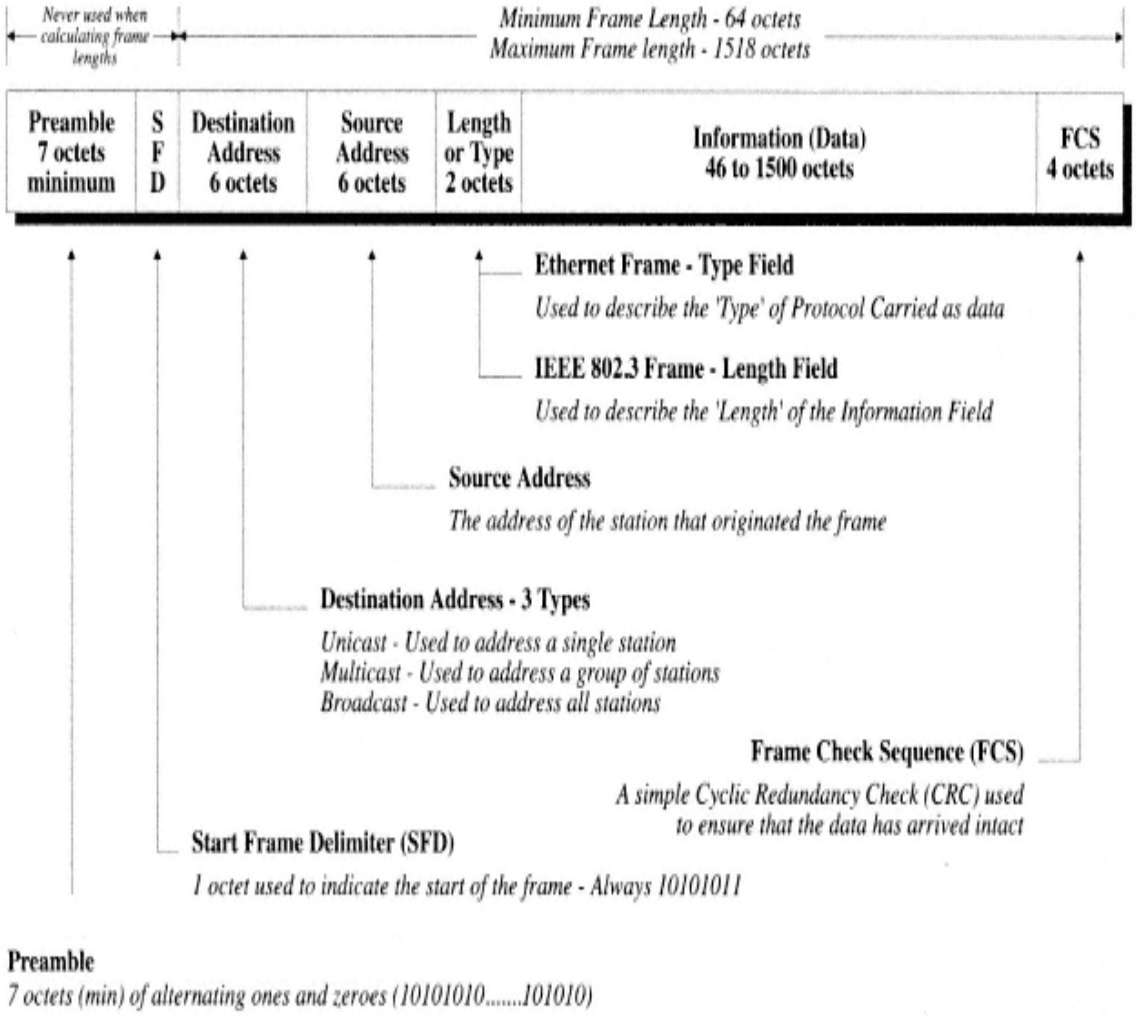


Figure 112 Ethernet/802.3 Frame Structure

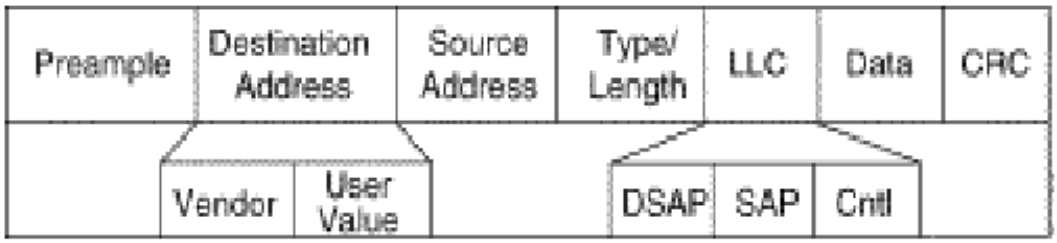


Figure 113 IEEE 802.3 Ethernet standard frame

IV. TCP Segments

The **source port** is the port of the TCP segment source while the **destination port** is the port of the TCP segment destination. The **sequence number** is the sequence number of the first byte of a TCP segment in the data flow from the source to the destination (TCP transfers bytes from the sequence number of the transferred byte to the length of the segment). **Header length** specifies the length of the TCP segment header in multiples of 32 bits (4 bytes), similar to the format of IP headers See figure 58.**Window size** specifies the maximum increment of the sequence number that will be still accepted by the destination.

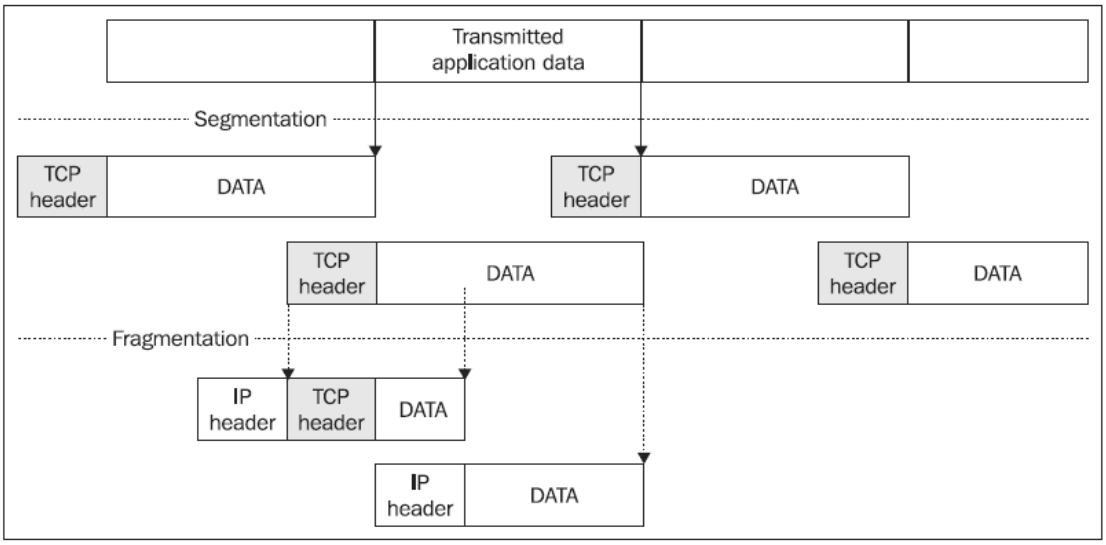


Figure 114 Segmentation and fragmentation

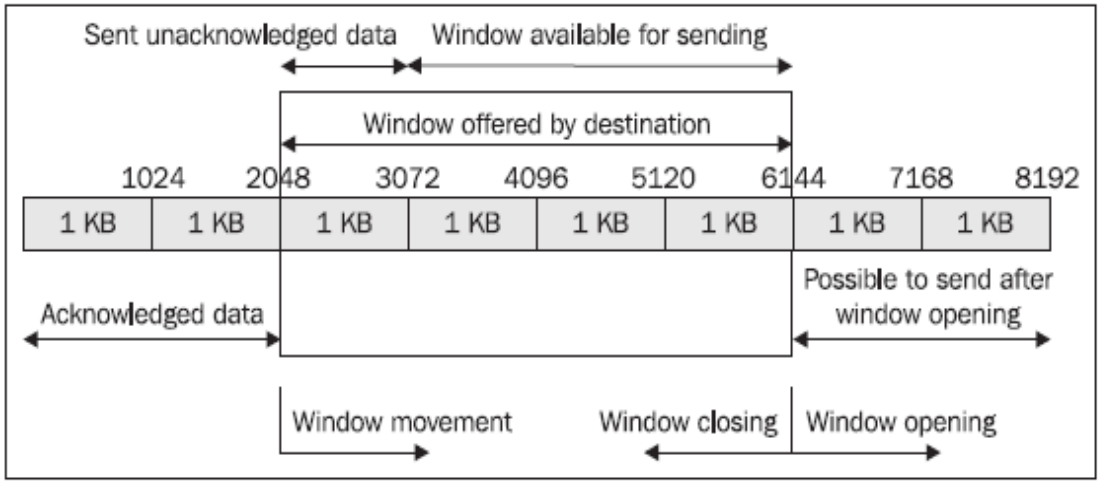


Figure 115 Window

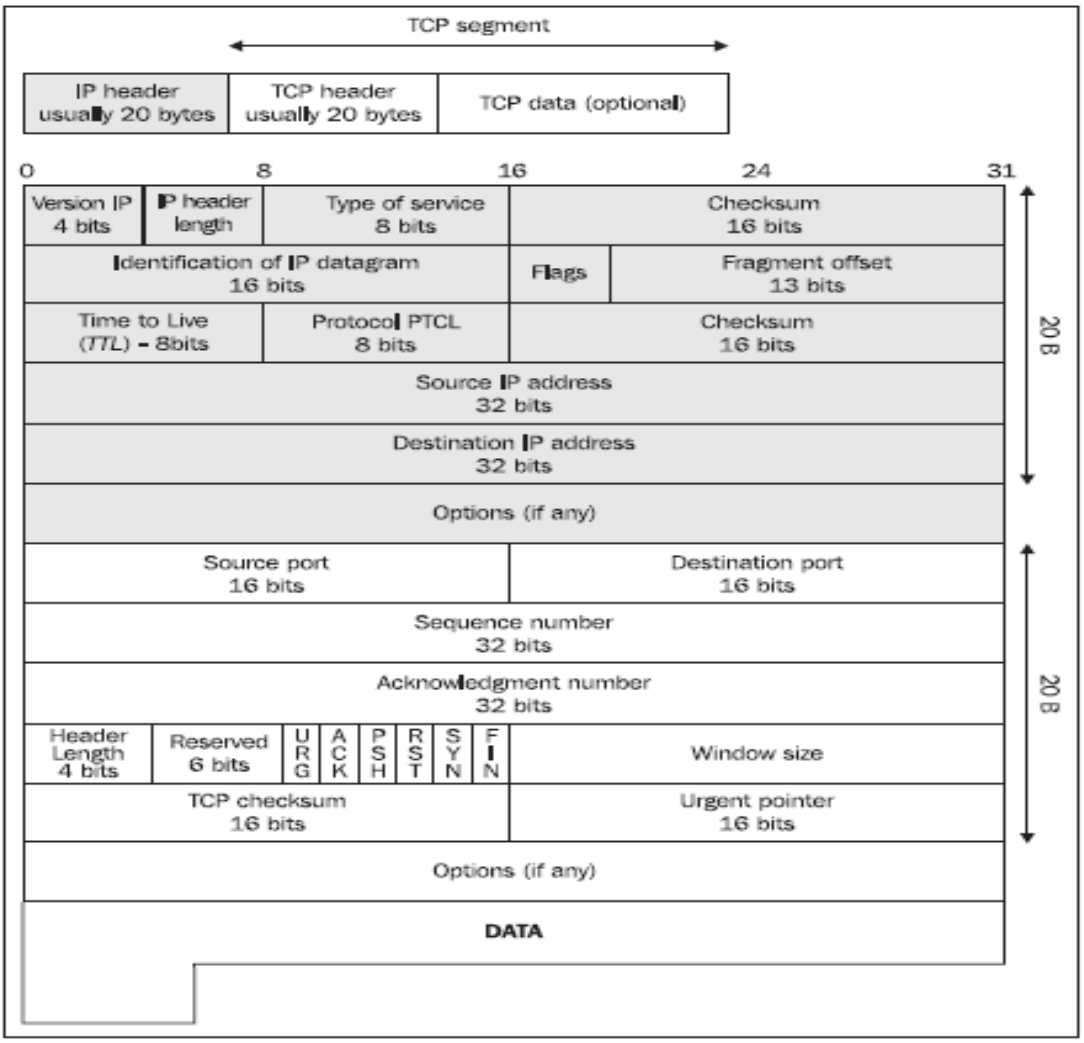


Figure 116 IP and TCP Header

V. Establishing and closuring a Connection with TCP

The core of IP was the IP datagram description. Since IP is a datagram-oriented (connectionless) service, there was not much of a need to prepare for cases in which the IP datagram was not delivered.

TCP uses IP for transferring data over the Internet, even though it establishes a reliable stream-oriented service over this protocol. It must solve the problems of establishing and closing a connection, confirming received data, and re-requesting lost data, and also solve problems with keeping the communication paths passable. The TCP segment description is obviously only one small part of TCP. A larger part of the protocol is the description of TCP segment exchange (handshaking) between both ends of the TCP connection. See figure 58.

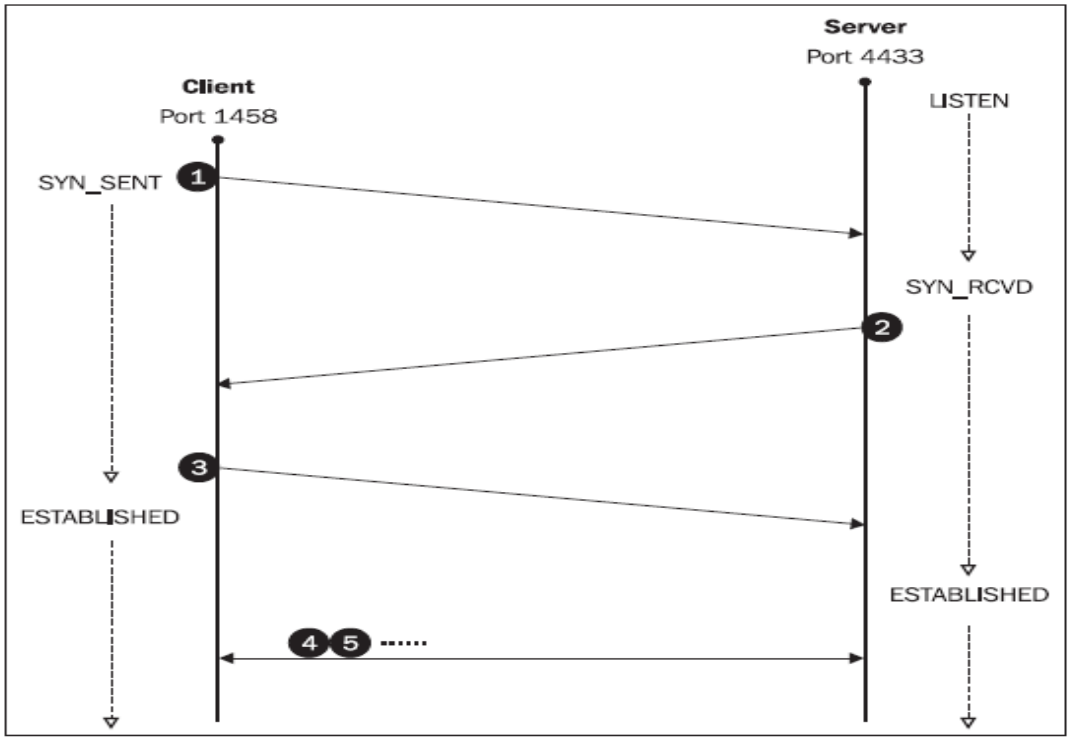


Figure 117 establishing a connection

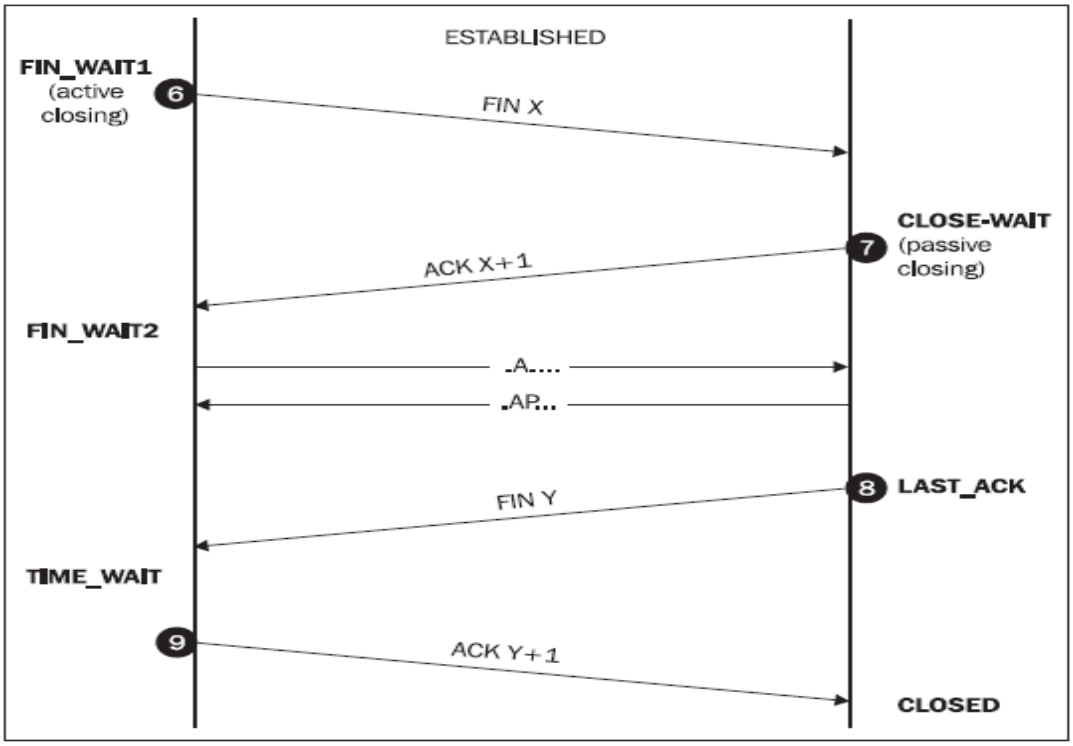


Figure 118 Connection closure

VI. Network Architecture

Network architecture is a design for the physical network and a collection of specifications defining communications on that physical network. The communication details are dependent on the physical details, so the specifications usually come together as a complete package. These specifications include considerations such as the following:

a. **Access method**: an access method is a set of rules defining how the computers will share the transmission medium. To avoid data collisions, computers must follow these rules when they transmit data.
b. **Data frame format:** The IP-level datagram from the Internet layer is encapsulated in a data frame with a predefined format. The data enclosed in the header must supply the information necessary to deliver data on the physical network.
c. **Cabling type**: The type of cable used for a network has an effect on certain other design parameters, such as the electrical properties of the bit stream transmitted by the adapter.
d. **Cabling rules:** The protocols, cable type, and electrical properties of the transmission have an effect on the maximum and minimum lengths for the cable and for the cable connector specifications.

Appendix D: FBUS Protocol

The general idea is like this

A. Microcontroller asks for some data
B. Phone say, "I understood what you want"
C. Phones send what Microcontroller wanted.
D. Phone waits to see if Microcontroller got what he wants.
E. If Microcontroller does not say that he received the data, Phone sends the data one more time.
F. Phone waits to see if Microcontroller got what he wants
G. If Microcontroller does not say that he received the data, Phone sends the data one more time.
H. Now Phone stop sending data.(Since Microcontroller is really deaf or he is rude)
I. On the contrary if Microcontroller says he got the data the very first time, Phone stop sending more data.

Just like 2 people are talking!

I. How to connect microcontrollers to your Nokia 3310

Most Nokia phones have F-Bus and M-Bus connections that can be used to connect a phone to a PC or in our case a microcontroller. The connection can be used for controlling just about all functions of the phone, as well as uploading new firmware etc. This bus will allow us to send and receive SMS messages. Want to turn your air-conditioner on remotely?

Figure 119 3310 Phone and FBUS connection

The very popular Nokia 3310/3315 has the F/M Bus connection under the battery holder. This is a bit of a pain to get to and requires a special cable to make the connection. The left picture above shows the 4 gold pads used for the F and M Bus. The right picture shows the F-Bus cable connected to my Nokia 3310.

Figure 120 Nokia 3310 and it's download cable

Nokia download cable is available from most mobile phone shops and some electronics stores.

The cable contains electronics to level convert 3V signals to RS232 type signals. There are also M and F bus switching in most cables.

You can use PC software like Logomanager from (here) and Oxygen Phone Manager from (here) to upload ringtones, graphics, phone numbers etc. No more paying for those cools ringtones, just download them off the internet or record your own!

Appendix E: Used Software and Hardware

I. Used Software

Table 5 Used Software

NO	Software Name	Company	Website
1	Visual Studio 2008	Microsoft	http://www.microsoft.com
2	MikroC Pro	Mikroelektronika	http://www.mikroe.com
3	Measurment Studio 8.6	National Instruments	http://www.ni.com/
4	ComponentOne	ComponentOne	http://www.componentone.com/
5	DotNetBar	DevComponents	http://www.devcomponents.com/dotnetbar/
6	Proteus 7 Professional	Labcenter Electronics	http://www.labcenter.com/
7	CCS PIC C Compiler	CCS	http://www.ccsinfo.com/
8	Eagle	cadsoft.	http://www.cadsoft.de/
9	Sprint-Layout 5.0	abacom	http://www.abacom-online.de/

II. Used Hardware

Table 6 Used Hardware

NO	Name	Company	Website
1	PIC16F877A	Microchip	http://www.microchip.com/
2	XM10 Modula	marmitek	http://www.marmitek.com/
3	LM35DZ	National Semiconductor	www.national.com
4	MAX232	maxim	http://www.maxim-ic.com/
5	MAX485	maxim	http://www.maxim-ic.com/
6	ULN2003	----------	----------
7	Relay	----------	----------

Appendix F: All Project software codes

F.1-Main Program(server program)

```
using NationalInstruments;
using NationalInstruments.UI;
using NationalInstruments.UI.WindowsForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using DevComponents;
using System.IO.Ports;
using AForge.Video;
using AForge.Video.DirectShow;
using System.Net;
using System.Speech.Synthesis;
namespace WindowsApplication1
{
    public partial class Form1 :
DevComponents.DotNetBar.Office2007RibbonForm
    {
        string Audio_Path;
        string mp3paz;
        SerialPort port;
        Sheimy_RS485 ROM_160,ROM_170;
        SpeechSynthesizer syn;
        private bool DeviceExist = false;
        private FilterInfoCollection videoDevices;
        private VideoCaptureDevice videoSource = null;
        private VideoCaptureDevice videoSource2 = null;
        public Form1()
        {
            InitializeComponent();
           ROM_160 = new Sheimy_RS485();
           ROM_170 = new Sheimy_RS485();
           syn = new SpeechSynthesizer();
            groupPanel2.Enabled = false;
            outdoor.Enabled = false;
            ROM1.Enabled = false;
            ROM2.Enabled = false;
            string[] ports = SerialPort.GetPortNames();
            foreach (string port in ports)
            {
                coms.Items.Add(port);
            }
        }
//=================================================================
======
                    // Get VEDIO Devices
```

```
//======================================================================
======

        private void getCamList()
        {
            try
            {
                videoDevices = new
FilterInfoCollection(FilterCategory.VideoInputDevice);
                cam1device.Items.Clear();

                if (videoDevices.Count == 0)
                    throw new ApplicationException();

                DeviceExist = true;
                foreach (FilterInfo device in videoDevices)
                {
                    cam1device.Items.Add(device.Name);

                }
                cam1device.SelectedIndex = 0; //make dafault to first
cam\

            }
            catch (ApplicationException)
            {
                DeviceExist = false;
                cam1device.Items.Add("No capture device on your
system");

            }
        }

//======================================================================
======
//======================================================================
======
 // Get VEDIO Devices
//======================================================================
======

        private void getCamList2()
        {
            try
            {
                videoDevices = new
FilterInfoCollection(FilterCategory.VideoInputDevice);

                cam2device.Items.Clear();
                if (videoDevices.Count == 0)
                    throw new ApplicationException();

                DeviceExist = true;
                foreach (FilterInfo device in videoDevices)
                {

                    cam2device.Items.Add(device.Name);
```

```
                }

                cam2device.SelectedIndex = 0; //make dafault to first
cam
            }
            catch (ApplicationException)
            {
                DeviceExist = false;

                cam2device.Items.Add("No capture device on your
system");
            }
        }


//=====================================================================
======
        // Close Vedio Devices1
//=====================================================================
======
 private void CloseVideoSource()
        {
            if (!(videoSource == null))
                if (videoSource.IsRunning)
                {
                    videoSource.SignalToStop();
                    videoSource = null;
                }
        }
//=====================================================================
======

//=====================================================================
======
 // Close Vedio Devices2

//=====================================================================
======
 private void CloseVideoSource2()
 {
     if (!(videoSource2 == null))
         if (videoSource2.IsRunning)
         {
             videoSource2.SignalToStop();
             videoSource2 = null;
         }
 }

//=====================================================================
======
        //Get New fram for cam 1
//=====================================================================
======
 private void video_NewFrame(object sender, NewFrameEventArgs
eventArgs)
 {
     Bitmap img = (Bitmap)eventArgs.Frame.Clone();
```

```
     pictureBox1.Image = img;
 }
//=====================================================================
======

//=====================================================================
======
 //Get New fram for cam 2

//=====================================================================
======
 private void video_NewFrame2(object sender, NewFrameEventArgs
eventArgs)
 {
     Bitmap img = (Bitmap)eventArgs.Frame.Clone();
     pictureBox2.Image = img;
 }

//=====================================================================
======
         //convert bool to byte
//=====================================================================
======
         private byte bool_to_byte(bool []a,bool []b)
    {
             int i=0;
             byte result=0;
         foreach (bool d in a)
         {
             if (d == true)
             result +=(byte) Math.Pow(2, i);
             i++;
         }
         foreach (bool d in b)
         {
             if (d == true)
                 result += (byte)Math.Pow(2, i);
             i++;
         }
         return result;

    }

//=====================================================================
======
         private void switchlight_ValuesChanged(object sender, EventArgs
e)
         {
             ledlight.SetValues(switchlight.GetValues());

ROM_160.set_data(bool_to_byte(switchlight.GetValues(),switchlight1.GetV
alues()));
             send_data(ROM_160.get_data());
             byte []asd=ROM_160.get_data().ToArray<byte>();
             labelX8.Text=""+asd[4];
         }
```

```
//=======================choseAudio=================================
======
        private void comboBoxEx1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            Audio_Path = Application.ExecutablePath;
            mp3paz = Audio_Path.Remove(Audio_Path.Length - 23) +
@"Audio\" + comboBoxEx1.Text+ ".mp3";
            player.URL = mp3paz;
        }
//=============================================================================
======
//=================================openport=============================
======
        private void buttonX2_Click(object sender, EventArgs e)
        {
            port = new SerialPort(coms.Text, 9600, Parity.None, 8,
StopBits.One);
            if (port.IsOpen) port.Close();
            try
            {
                //open serial port
                port.Open();
                port.DataReceived += new
SerialDataReceivedEventHandler(serialPort1_DataReceived);
                Control.CheckForIllegalCrossThreadCalls = false;
                port.RtsEnable = false;
            }
            catch (System.Exception ex)
            {
                if (port.IsOpen) port.Close();
            }
            outdoor.Enabled = true;
            ROM1.Enabled = true;
            ROM2.Enabled = true;
        }
//=============================================================================
======
//=============================================================================
======       //Recive Data From Serial
//=============================================================================
======
        private void serialPort1_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {

            int bytes = port.BytesToRead;
            if (bytes >= 8)
            {
               // bytes = 7;
                byte[] buffer = new byte[8];
                port.Read(buffer, 0, 8);
                string m = buffer[1] + "  " + buffer[3] + "  " +
                    buffer[4] + "  " + buffer[5];
                textBoxX2.Text = m;
                if (buffer[1] == 160)
                {
```

```
                    ROM1_Temp.Value = buffer[3];
                    syn.Speak("Eng sheimy : temperature change and now
it is : " + buffer[3]);
                }
                if (buffer[1] == 170)
                {
                    ROM2_Temp.Value = buffer[3];
                    syn.Speak("Eng sheimy : temperature change and now
it is : " + buffer[3]);
                }
            }

        }

//=====================================================================
======
//=====================================login=============================
======
        private void buttonX1_Click(object sender, EventArgs e)
        {
            if ((user.Text == "AA") && (pass.Text == "AA"))
            {
                groupPanel2.Enabled = true;

            }
        }
//=====================================================================
======
//==============================refreshcam1============================
======
        private void refresh_Click(object sender, EventArgs e)
        {
            getCamList();
        }
//==================================startcam1==========================
======
        private void Start_Click(object sender, EventArgs e)
        {
            if (Start.Text == "&Start")
            {
                if (DeviceExist)
                {
                    videoSource = new
VideoCaptureDevice(videoDevices[cam1device.SelectedIndex].MonikerString
);
                    videoSource.NewFrame += new
NewFrameEventHandler(video_NewFrame);
                    CloseVideoSource();
                    videoSource.DesiredFrameSize = new Size(362, 253);
                    //videoSource.DesiredFrameRate = 10;
                    videoSource.Start();
                  //  label2.Text = "Device running...";
                    Start.Text = "&Stop";
                   // timer1.Enabled = true;
                }
                else
                {
```

```
                    // label2.Text = "Error: No Device selected.";
                }
            }
            else
            {
              if (videoSource.IsRunning)
                {
                    // timer1.Enabled = false;
                     CloseVideoSource();
                    // label2.Text = "Device stopped.";
                     Start.Text = "&Start";
                }
            }
        }
//================================closeform
================================
        private void Form1_FormClosed(object sender,
FormClosedEventArgs e)
        {
            CloseVideoSource();
            CloseVideoSource2();
        }
//==========================refresh cam list
2============================
        private void refresh2_Click(object sender, EventArgs e)
        {
            getCamList2();
        }
//==========================================================================
======
//========================start cam 2
=======================================
        private void start2_Click(object sender, EventArgs e)
        {
            if (Start2.Text == "&Start")
            {
                if (DeviceExist)
                {
                    videoSource2 = new
VideoCaptureDevice(videoDevices[cam2device.SelectedIndex].MonikerString
);
                    videoSource2.NewFrame += new
NewFrameEventHandler(video_NewFrame2);
                    CloseVideoSource2();
                    videoSource2.DesiredFrameSize = new Size(362, 253);
                    //videoSource.DesiredFrameRate = 10;
                    videoSource2.Start();
                    //  label2.Text = "Device running...";
                    Start2.Text = "&Stop";
                    // timer1.Enabled = true;
                }
                else
                {
                    // label2.Text = "Error: No Device selected.";
                }
            }
            else
```

```
            {
                if (videoSource2.IsRunning)
                {
                    // timer1.Enabled = false;
                    CloseVideoSource2();
                    // label2.Text = "Device stopped.";
                    Start2.Text = "&Start";
                }
            }
        }
//==========================ROM
1=========================================
        private void ROM1_Click(object sender, EventArgs e)
        {
            ROM_160.set_add(160);
        }
//================================last 4
switchs===========================
        private void switchlight1_ValuesChanged(object sender,
EventArgs e)
        {
            ledlight1.SetValues(switchlight1.GetValues());
            ROM_160.set_data(bool_to_byte(switchlight.GetValues(),
switchlight1.GetValues()));
            send_data(ROM_160.get_data());
        }
//================================================================
======
//========================== Send
Data=====================================
        public void send_data(byte []sbuffer)
        {
         if (port.RtsEnable)
            {
                port.RtsEnable = false;  //Enable request to send
            }
            port.Write(sbuffer, 0, sbuffer.Length);
        }
//========================== ROM 2
=========================================
        private void ROM2_Click(object sender, EventArgs e)
        {
            ROM_170.set_add(170);
        }

        private void switchlight2_ValuesChanged(object sender,
EventArgs e)
        {
            ledlight2.SetValues(switchlight2.GetValues());
            ROM_170.set_data(bool_to_byte(switchlight2.GetValues(),
switchlight3.GetValues()));
            send_data(ROM_170.get_data());
        }

        private void switchlight3_ValuesChanged(object sender,
EventArgs e)
        {
```

```
            ledlight3.SetValues(switchlight3.GetValues());
            ROM_170.set_data(bool_to_byte(switchlight2.GetValues(),
switchlight3.GetValues()));
            send_data(ROM_170.get_data());
        }

        private void comboBoxEx2_SelectedIndexChanged(object sender,
EventArgs e)
        {
            Audio_Path = Application.ExecutablePath;
            mp3paz = Audio_Path.Remove(Audio_Path.Length - 23) +
@"Audio\" + comboBoxEx2.Text + ".mp3";
            player.URL = mp3paz;
        }

        private void readtemp_Click(object sender, EventArgs e)
        {
            port.RtsEnable = true;  //Disable request to send
        }

        private void Read_temp2_Click(object sender, EventArgs e)
        {
            port.RtsEnable = true;  //Disable request to send
        }
//======================== TCP connect
=====================================
        private void buttonX3_Click(object sender, EventArgs e)
        {
            IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());
            IPAddress ipAddress = ipHostInfo.AddressList[0];
            ip.Text = "" + ipAddress;
            TCP_Server tcp = new
TCP_Server(ipAddress,portnumber.Value,this );
            tcp.Show();
        }
//========================================================================
======
        public SerialPort getport()
        {
            return port;
        }
//========================================================================
======
        //====================Get Romm1 pass====================\\
        public byte getRoom1_pass()
        {
            byte b = byte.Parse(room1_pass.Text);
            return b;
        }
//========================================================================
======
        //
        public byte getRoom2_pass()
        {
            byte b = byte.Parse(room2_pass.Text);
            return b;
        }
```

```
//=====================================================================
======
//============================== speak_text
================================
        public void speak_text(string s)
        {
            syn.Speak(s);
        }
//=====================================================================
======
    }
    }
```

F.2- TCP_Server:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using DevComponents;
using System.Threading;
using System.Net.Sockets;
using System.IO;
using System.IO.Ports;
using System.Net;
namespace WindowsApplication1
{

    public partial class TCP_Server
:DevComponents.DotNetBar.Office2007RibbonForm
    {
        /* This stores data about each client */
        public struct ClientData
        {
            public Socket structSocket;
            public Thread structThread;
        }
        // Required designer variable.
        private TcpListener tcpLsn;
        private Hashtable dataHolder = new Hashtable();
        private static long connectId = 0;
        private Thread tcpThd;
        private string ipAddress = "192.168.1.100";
        delegate void SetTextCallBack(string text);
        Form1 form_1;
        Sheimy_RS485 ROM;
        private byte[] rom_data;
        SerialPort p;
        byte pass;
//=====================================================================
======
```

```
        public TCP_Server(IPAddress ip, int portno, Form1 f)
        {
            InitializeComponent();
            tcpLsn = new TcpListener(ip, portno);
            tcpLsn.Start();
            statusBar1.Text = "Listen at: " +
tcpLsn.LocalEndpoint.ToString();
            tcpThd = new Thread(new ThreadStart(WaitingForClient));
            tcpThd.Start();
            form_1 = f;
           // pass = form_1.getpass();
            ROM= new Sheimy_RS485();
        }
//=====================================================================
======
        public void WaitingForClient()
        {
            ClientData CData;
            while (true)
            {     /* Accept will block until someone connects */
                CData.structSocket = tcpLsn.AcceptSocket();
                Interlocked.Increment(ref connectId);
                CData.structThread = new Thread(new
ThreadStart(ReadSocket));

                lock (this)
                { // it is used to keep connected Sockets and active
thread
                    dataHolder.Add(connectId, CData);
                    upDateDataGrid("Connected > " + connectId + "    " +
DateTime.Now.ToLongTimeString());
                }
                CData.structThread.Start();
            }
        }

        public void ReadSocket()
        {
            /* realId will be not changed for each thread, but
connectId is
             * changed. it can't be used to delete object from
Hashtable*/
            long realId = connectId;
            Byte[] receive;
            ClientData cd = (ClientData)dataHolder[realId];
            Socket s = cd.structSocket;
            int ret = 0;

            while (true)
            {
                byte room1_pass,room2_pass;
                room1_pass = form_1.getRoom1_pass();
                room2_pass = form_1.getRoom2_pass();
                if (s.Connected)
                {
                    receive = new Byte[100];
                    try
```

```
                    {   /* Receive will block until data coming ret is
0 or Exception
                                    *  happen when Socket connection
is broken*/
                        ret = s.Receive(receive, receive.Length, 0);

                        if (ret > 0)
                        {

                            if ((receive[0] == 160 && receive[1] ==
room1_pass) || (receive[0] == 170 && receive[1] == room2_pass))
                            {
                                ROM.set_add(receive[0]);
                                ROM.set_data(receive[2]);
                                rom_data = ROM.get_data();

                                form_1.send_data(rom_data);
                            }
                            else
                            {
                                form_1.speak_text("Dear User : someone
try Hacking your Device");
                            }
                            foreach (ClientData clntData in
dataHolder.Values)
                            {
                                if (clntData.structSocket.Connected)
                                    clntData.structSocket.Send(receive,
ret, SocketFlags.None);
                            }
                        }
                        else { break; }
                    }
                    catch (Exception e)
                    {
                        upDateDataGrid(e.ToString());
                        if (!s.Connected) break;
                    }
                }
            }
            CloseTheThread(realId);
        }

        private void CloseTheThread(long realId)
        {
            try
            {
                ClientData clientData = (ClientData)dataHolder[realId];
                clientData.structThread.Abort();
            }

            catch (Exception e)
            {
                lock (this)
                {
                    dataHolder.Remove(realId);
```

```
                    upDateDataGrid("Disconnected > " + realId + "    " +
DateTime.Now.ToLongTimeString());
                }
            }
        }

        public void upDateDataGrid(string displayString)
        {
            if (this.textBox1.InvokeRequired)
            {
                SetTextCallBack t = new
SetTextCallBack(upDateDataGrid);
                this.Invoke(t, new object[] { displayString });
            }
            else
            {
                textBox1.AppendText(displayString + "\r\n");
            }

        }

        private void TCP_Server_FormClosed(object sender,
FormClosedEventArgs e)
        {
            tcpLsn.Stop();
            foreach (ClientData cd in dataHolder.Values)
            {
                if (cd.structSocket.Connected) cd.structSocket.Close();
                if (cd.structThread.IsAlive) cd.structThread.Abort();
            }
            tcpThd.Abort();
        }


    }
}
```

F.3- class Sheimy_RS485

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsApplication1
{

    class Sheimy_RS485
    {
        string InputData = String.Empty;
        byte[] sbuffer;
        byte[] rbuffer;
        char[] chars;
        byte add,data,bytes_no=129, crc;
        int t4;
```

```
//============================================================
        //Not Method
//============================================================
        public double takeNOT(int b)
        {
            int t, j = 7;
            string g = string.Empty;
            double k = 0;
            for (t = 128; t > 0; t = t / 2)
            {
                if ((b & t) != 0) g += "0";
                if ((b & t) == 0) g += "1";
            }
            chars = g.ToCharArray();
            foreach (char c in chars)
            {
                if (c == '1')
                {
                    k += Math.Pow(2, j);
                }
                j--;

            }
            return k;
        }
//============================================================
        // get data Method
//============================================================
        public byte[]  get_data()
        {
            t4 = (add ^ bytes_no ^ data);
            crc = (byte)takeNOT(t4);
            sbuffer = new byte[] { 150, add, bytes_no, data, crc, 169
};
            return sbuffer;
        }

//============================================================================
======
        // set add ,data and get them
//============================================================================
======        public void set_add(byte d)
        {
            this.add = d;
        }
//============================================================================
======
        public void  set_data(byte da)
        {
            this.data = da;
        }
//============================================================================
======
    }
}
```

F.4-Software Controller Client

```
using NationalInstruments;
using NationalInstruments.UI;
using NationalInstruments.UI.WindowsForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using DevComponents;
using System.Net.Sockets;
using System.IO;
using System.Threading;
namespace Smart_Home_Clinte
{
    public partial class Form1
:DevComponents.DotNetBar.Office2007RibbonForm
    {
        public Thread tcpThd;
        public byte[] readBuffer;
        public byte[] writeBuffer;
        public Stream stm;
        public Socket socket;
        public TcpClient tcpclnt;
        public string loginName = "";
        private LoginInfo loginForm;
        public Form1()
        {
            InitializeComponent();
        }
//=========================convert bool to byte
============================
        private byte bool_to_byte(bool[] a)
        {
            int i = 0;
            byte result = 0;
            foreach (bool d in a)
            {
                if (d == true)
                    result += (byte)Math.Pow(2, i);
                i++;
            }
```

```
            return result;

        }
//=====================================================================
======
        public void startServer(string ipAddress, int portNumber,
string loginName)
        {
            this.loginName = loginName;
            tcpclnt = new TcpClient();
            tcpclnt.Connect(ipAddress.Trim(), portNumber);
            textBoxWindow.AppendText("Connecting to server...");

            writeToServer("Hello " + loginName + " Now you are
connected to the server" + "\r\n");
            stm = tcpclnt.GetStream();
            tcpThd = new Thread(new ThreadStart(ReadSocket));
            tcpThd.Start();
        }
//=====================================================================
======
        public void ReadSocket()
        {
            while (true)
            {
                try
                {
                    readBuffer = new Byte[100];
                    stm.Read(readBuffer, 0, 100);

                    /* If the text box exceed the maximum lenght, then
get
                     * remove the top part of the text*/
                    if (textBoxWindow.Text.Length >
textBoxWindow.MaxLength)
                    {
                        textBoxWindow.Select(0, 300);
                        textBoxWindow.SelectedText = "";
                    }


textBoxWindow.AppendText(System.Text.Encoding.ASCII.GetString(readBuffe
r) + "\r\n");
                }

                catch (Exception e)
                { break; }
            }
        }
//=====================================================================
======
        public void writeToServer(string strn)
        {
            System.Text.ASCIIEncoding encord = new
System.Text.ASCIIEncoding();
            writeBuffer = encord.GetBytes(strn);
```

```
            if (stm != null) stm.Write(writeBuffer, 0,
writeBuffer.Length);
        }

        private void buttonX3_Click(object sender, EventArgs e)
        {
            loginForm = new LoginInfo();
            loginForm.infoChecker(this);
        }

        private void buttonX1_Click(object sender, EventArgs e)
        {
            writeToServer(loginName + " > " + textBoxX1.Text.Trim() +
"\r\n");
            textBoxX1.Text = "";
        }
//=====================================================================
======
        private void switchArray1_ValuesChanged(object sender,
EventArgs e)
        {
            ledArray1.SetValues(switchArray1.GetValues());
            writeBuffer = new byte[3];
            writeBuffer[0] = 160;
            writeBuffer[1] = byte.Parse(rom1_pass.Text);
            writeBuffer[2] =bool_to_byte(switchArray1.GetValues());
            if (stm != null) stm.Write(writeBuffer, 0,
writeBuffer.Length);
        }
//=====================================================================
======
        private void switchArray2_ValuesChanged(object sender,
EventArgs e)
        {
            ledArray2.SetValues(switchArray2.GetValues());
            writeBuffer = new byte[3];
            writeBuffer[0] = 170;
            writeBuffer[1] = byte.Parse(rom2_pass.Text);
            writeBuffer[2] = bool_to_byte(switchArray2.GetValues());
            if (stm != null) stm.Write(writeBuffer, 0,
writeBuffer.Length);

        }
//=====================================================================
======
    }
}
```

F.5-X10 TR Program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using DevComponents;
using System.IO.Ports;
namespace X10TR
{
    public partial class Form1 :
DevComponents.DotNetBar.Office2007RibbonForm
    {
        SerialPort port;
        public Form1()
        {
            InitializeComponent();
            string[] ports = SerialPort.GetPortNames();
            foreach (string p in ports)
            {
                coms.Items.Add(p);
            }
        }
        //================================================
        private void serialPort1_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {

            string m;
            m = port.ReadExisting();
            t.Text += "\n\r";
            t.Text +="Recive :"+m+"\n\r";

        }

        private void openbutton_Click(object sender, EventArgs e)
        {
            port = new SerialPort(coms.Text, 9600);
            if (port.IsOpen) port.Close();
            port.Open();
//=========================Defin
this=======================================
            port.DataReceived += new
SerialDataReceivedEventHandler(serialPort1_DataReceived);
            Control.CheckForIllegalCrossThreadCalls = false;

//=====================================================================
======
        }

        private void send_Click(object sender, EventArgs e)
        {
            string s = comboBoxEx1.Text + comboBoxEx2.Text;
            t.Text += "\n\r";
            t.Text += "Send :" + s + "\n\r";
            port.Write(s);
        }

        private void buttonX1_Click(object sender, EventArgs e)
        {
            t.Text = "";
```

```
            }
            //=============================================
        }
}
```

Appendix G: All Project Microcontroller codes

All of the following codes are written using mikroC PRO for PIC V3.8 from mikroelektronika company

G.1-RS485 Receiver

```
char dat[9];                                    // buffer for receving/sending messages
char i,j;

sbit  rs485_rxtx_pin at RC1_bit;                // set transcieve pin
sbit  rs485_rxtx_pin_direction at TRISC1_bit;   // set transcieve pin direction

// Interrupt routine
void interrupt() {
 RS485Slave_Receive(dat);
}

void main() {
 ADCON1.PCFG3=0;                          // Configure AN pins as digital I/O
 ADCON1.PCFG2=1;
 ADCON1.PCFG1=1;
 ADCON1.PCFG0=0;
 CMCON.CM0=1;
 CMCON.CM1=1;
 CMCON.CM2=1;

 PORTB = 0;
 PORTD = 0;
 TRISB = 0;
 TRISD = 0;


 UART1_Init(9600);              // initialize UART1 module
 Delay_ms(100);
 RS485Slave_Init(160);          // Intialize MCU as slave, address 160

 dat[4] = 0;                 // ensure that message received flag is 0
 dat[5] = 0;                 // ensure that message received flag is 0
 dat[6] = 0;                 // ensure that error flag is 0

 RCIE_bit = 1;                // enable interrupt on UART1 receive
 TXIE_bit = 0;                // disable interrupt on UART1 transmit
 PEIE_bit = 1;                // enable peripheral interrupts
```

```
 GIE_bit = 1;                  // enable all interrupts
 PORTB=0xFF;
 Delay_ms(1000);
 while (1) {
  if (dat[5])  {               // if an error detected, signal it by
    PORTD = 0xAA;              //   setting portd to 0xAA
    dat[5] = 0;
  }
 if (dat[4]) {                 // upon completed valid message receive
  dat[4] = 0;
  PORTB = dat[0];
  }              //   data[4] is set to 0xFF
  }
}
```

G.2-RS485 Transmitter

```
char dat[9];          // buffer for receving/sending messages
char i,j;

sbit  rs485_rxtx_pin at RC1_bit;          // set transcieve pin
sbit  rs485_rxtx_pin_direction at TRISC1_bit;   // set transcieve pin direction

// Interrupt routine
void interrupt() {
 RS485Slave_Receive(dat);
}

void main() {


 PORTB = 0;
 PORTD = 0;
 TRISB = 0;
 TRISD = 0;


 UART1_Init(9600);              // initialize UART1 module
 Delay_ms(100);
 RS485Slave_Init(160);          // Intialize MCU as slave, address 160

 dat[4] = 0;                   // ensure that message received flag is 0
```

```
 dat[5] = 0;                 // ensure that message received flag is 0
 dat[6] = 0;                 // ensure that error flag is 0

 RCIE_bit = 1;               // enable interrupt on UART1 receive
 TXIE_bit = 0;               // disable interrupt on UART1 transmit
 PEIE_bit = 1;               // enable peripheral interrupts
 GIE_bit = 1;                // enable all interrupts

 while (1) {
    PORTB=dat[0];            // increment received dat[0]
    Delay_ms(1000);
    RS485Slave_Send(dat,1);      //   and send it back to master
    PORTB=~PORTB;
 }
}
```

G.3-RS485 Temperature sender

```
char dat[9];          // buffer for receiving/sending messages
char i,j;
long y,x;
sbit  rs485_rxtx_pin at RC1_bit;          // set transceiver pin
sbit  rs485_rxtx_pin_direction at TRISC1_bit;   // set transceiver pin direction

// Interrupt routine
void interrupt() {
 RS485Slave_Receive(dat);
}
void Read_AD(){
  y=ADC_Read(2);
  y=y*5000;  // Convert to
  y=y/1023;  // MV  OR Convert from level to voltage
  y=y/10;
}
void main() {
 ADCON1.PCFG0=0;   //ALL PORT A AS Analog AND reference are VDD
 ADCON1.PCFG1=1;
 ADCON1.PCFG2=0;
 ADCON1.PCFG3=0;
 CMCON.CM0=1;
 CMCON.CM1=1;
 CMCON.CM2=1;
```

```
 PORTB = 0;
 PORTD = 0;
 TRISB = 0;
 TRISD = 0;


 UART1_Init(9600);               // initialize UART1 module
 Delay_ms(100);
 RS485Slave_Init(160);           // Initialize MCU as slave, address 160
 dat[4] = 0;                    // ensure that message received flag is 0
 dat[5] = 0;                    // ensure that message received flag is 0
 dat[6] = 0;                    // ensure that error flag is 0

 RCIE_bit = 1;                   // enable interrupt on UART1 receive
 TXIE_bit = 0;                   // disable interrupt on UART1 transmit
 PEIE_bit = 1;                   // enable peripheral interrupts
 GIE_bit = 1;                    // enable all interrupts
  y=0;
  PORTB=0xFF;
  Delay_ms(1000);
  PORTB=0x00;
 while (1) {
 if (dat[5])  {                // if an error detected, signal it by
    PORTB = 0xFF;                //   setting portd to 0xAA
    dat[5] = 0;
 }
     x=y;
     Read_AD();
     if(x!=y){
     dat[0] =y;                 // increment received dat[0]
     dat[1]=4;
     dat[2]=5;
     Delay_ms(2000);
     RS485Slave_Send(dat,3);        //   and send it back to master
     }
  }
}
```

G.4-RS485 Transceiver

```
char dat[9];           // buffer for receving/sending messages
char i,j;
```

```
long y,x;
sbit  rs485_rxtx_pin at RC1_bit;           // set transcieve pin
sbit  rs485_rxtx_pin_direction at TRISC1_bit;   // set transcieve pin direction

// Interrupt routine
void interrupt() {
 RS485Slave_Receive(dat);
}
void Read_AD(){
  y=ADC_Read(2);
  y=y*5000;  // Convert to
  y=y/1023;  // MV  OR Convert from level to voltage
  y=y/10;
}
void main() {
 ADCON1.PCFG0=0;   //ALL PORT A AS Analog AND refrance are VDD
 ADCON1.PCFG1=1;
 ADCON1.PCFG2=0;
 ADCON1.PCFG3=0;
 CMCON.CM0=1;
 CMCON.CM1=1;
 CMCON.CM2=1;

 PORTB = 0;
 PORTD = 0;
 TRISB = 0;
 TRISD = 0;


 UART1_Init(9600);              // initialize UART1 module
 Delay_ms(100);
 RS485Slave_Init(160);           // Intialize MCU as slave, address 160

 dat[4] = 0;                  // ensure that message received flag is 0
 dat[5] = 0;                  // ensure that message received flag is 0
 dat[6] = 0;                  // ensure that error flag is 0

 RCIE_bit = 1;                 // enable interrupt on UART1 receive
 TXIE_bit = 0;                 // disable interrupt on UART1 transmit
 PEIE_bit = 1;                 // enable peripheral interrupts
 GIE_bit = 1;                  // enable all interrupts
  y=0;
```

```
  PORTB=0xFF;
  Delay_ms(1000);
  PORTB=0x00;
 while (1) {
 if (dat[5])  {              // if an error detected, signal it by
    PORTB = 0xFF;               //   setting portd to 0xAA
    dat[5] = 0;
 }
 if (dat[4]) {               // upon completed valid message receive
    dat[4] = 0;                //   data[4] is set to 0xFF
   PORTB = dat[0];
     }
     x=y;
     Read_AD();
if(x!=y){
     dat[0] =y;               // increment received dat[0]
     dat[1]=4;
     dat[2]=5;
     Delay_ms(2000);
     RS485Slave_Send(dat,3);        //   and send it back to master
  }
 }
}
```

G.5-X10 Transceiver

The following code is written an compiled by CCS C compiler

```
#if defined(__PCB__)
#include <16C56.h>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_A3, rcv=PIN_A2)

#elif defined(__PCM__)
#include <16F877A.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

#elif defined(__PCH__)
```

```
#include <18F452.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#endif

#include <x10.c>
#include <input.c>


void main() {
  char house_code;
  BYTE key_code;

  printf("Online\n\r");

  while (TRUE) {

    if(kbhit()) {
      house_code = getc();
      if((house_code>='A') && (house_code<='P')) {
        putc(house_code);
        key_code=gethex();
        x10_write(house_code,key_code);
        x10_write(house_code,key_code);
        printf("Eng");
      }

    }
      if(x10_data_ready()) {
      putc('>');
      x10_read(&house_code, &key_code);
      printf("%c%2X", house_code, key_code);
    }
  }
}
```

G.6- FBUS

```
/*Routine for starting the fbus Protocol*/
void startfbus()
   {
```

```
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );UART1_Write(0x55 );
}  // end method
/*Refresh the 3310 fbus link*/
void refresh()
{
UART1_Write(0x1E );UART1_Write(0x00 );UART1_Write(0x0C );UART1_Write(0xD1 );
UART1_Write(0x00 );UART1_Write(0x07 );UART1_Write(0x00 );UART1_Write(0x01 );
UART1_Write(0x00 );UART1_Write(0x03 );UART1_Write(0x00 );UART1_Write(0x01 );
UART1_Write(0x40 );UART1_Write(0x00 );UART1_Write(0x52 );UART1_Write(0xD5 );
}   // end method
```

```
/*Message1(getting the version of phone software) here*/
void mess()
{
UART1_Write(0x1E );UART1_Write(0x00 );UART1_Write(0x0C );UART1_Write(0x02 );
UART1_Write(0x00 );UART1_Write(0x35 );UART1_Write(0x00 );UART1_Write(0x01 );
UART1_Write(0x00 );UART1_Write(0x01 );UART1_Write(0x02 );UART1_Write(0x00 );
UART1_Write(0x07 );UART1_Write(0x91 );UART1_Write(0x36 );UART1_Write(0x19 );
UART1_Write(0x08 );UART1_Write(0x00 );UART1_Write(0x10 );UART1_Write(0x10 );
UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
UART1_Write(0x15 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
UART1_Write(0x0A );UART1_Write(0x0C );UART1_Write(0x91 );UART1_Write(0x36 );
UART1_Write(0x39 );UART1_Write(0x19 );UART1_Write(0x13 );UART1_Write(0x21 );
UART1_Write(0x70 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
UART1_Write(0x00 );UART1_Write(0xA7 );UART1_Write(0x00 );UART1_Write(0x00 );
UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
UART1_Write(0x65 );UART1_Write(0x39 );UART1_Write(0x3D );UART1_Write(0x2F );
UART1_Write(0xA7 );UART1_Write(0xE7 );UART1_Write(0xCB );UART1_Write(0xF2 );
UART1_Write(0x3C );UART1_Write(0x01 );UART1_Write(0x47 );UART1_Write(0x00 );
UART1_Write(0xA2 );UART1_Write(0x08 );
}  // end method

/*Message2 here*/
void burglar_alert()
{
 UART1_Write(0x1E );UART1_Write(0x00 );UART1_Write(0x0C );UART1_Write(0x02 );
 UART1_Write(0x00 );UART1_Write(0x56 );UART1_Write(0x00 );UART1_Write(0x01 );
 UART1_Write(0x00 );UART1_Write(0x01 );UART1_Write(0x02 );UART1_Write(0x00 );
 UART1_Write(0x07 );UART1_Write(0x91 );UART1_Write(0x36 );UART1_Write(0x19 );
 UART1_Write(0x08 );UART1_Write(0x00 );UART1_Write(0x10 );UART1_Write(0x10 );
 UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
 UART1_Write(0x15 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0xF0 );
 UART1_Write(0x2F );UART1_Write(0x0C );UART1_Write(0x91 );UART1_Write(0x36 );
 UART1_Write(0x39 );UART1_Write(0x19 );UART1_Write(0x13 );UART1_Write(0x21 );
 UART1_Write(0x70 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
 UART1_Write(0x00 );UART1_Write(0xA7 );UART1_Write(0x00 );UART1_Write(0x00 );
 UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );UART1_Write(0x00 );
 UART1_Write(0xC2 );UART1_Write(0xBA );UART1_Write(0xFC );UART1_Write(0xCC );
 UART1_Write(0x0E );UART1_Write(0xCB );UART1_Write(0x41 );UART1_Write(0x41 );
 UART1_Write(0x76 );UART1_Write(0x59 );UART1_Write(0x4E );UART1_Write(0x0F );
 UART1_Write(0x81 );UART1_Write(0x84 );UART1_Write(0xEF );UART1_Write(0xF9 );
 UART1_Write(0x9C );UART1_Write(0x05 );UART1_Write(0xCA );UART1_Write(0xBE );
 UART1_Write(0xEB );UART1_Write(0x72 );UART1_Write(0xD0 );UART1_Write(0x38 );
```

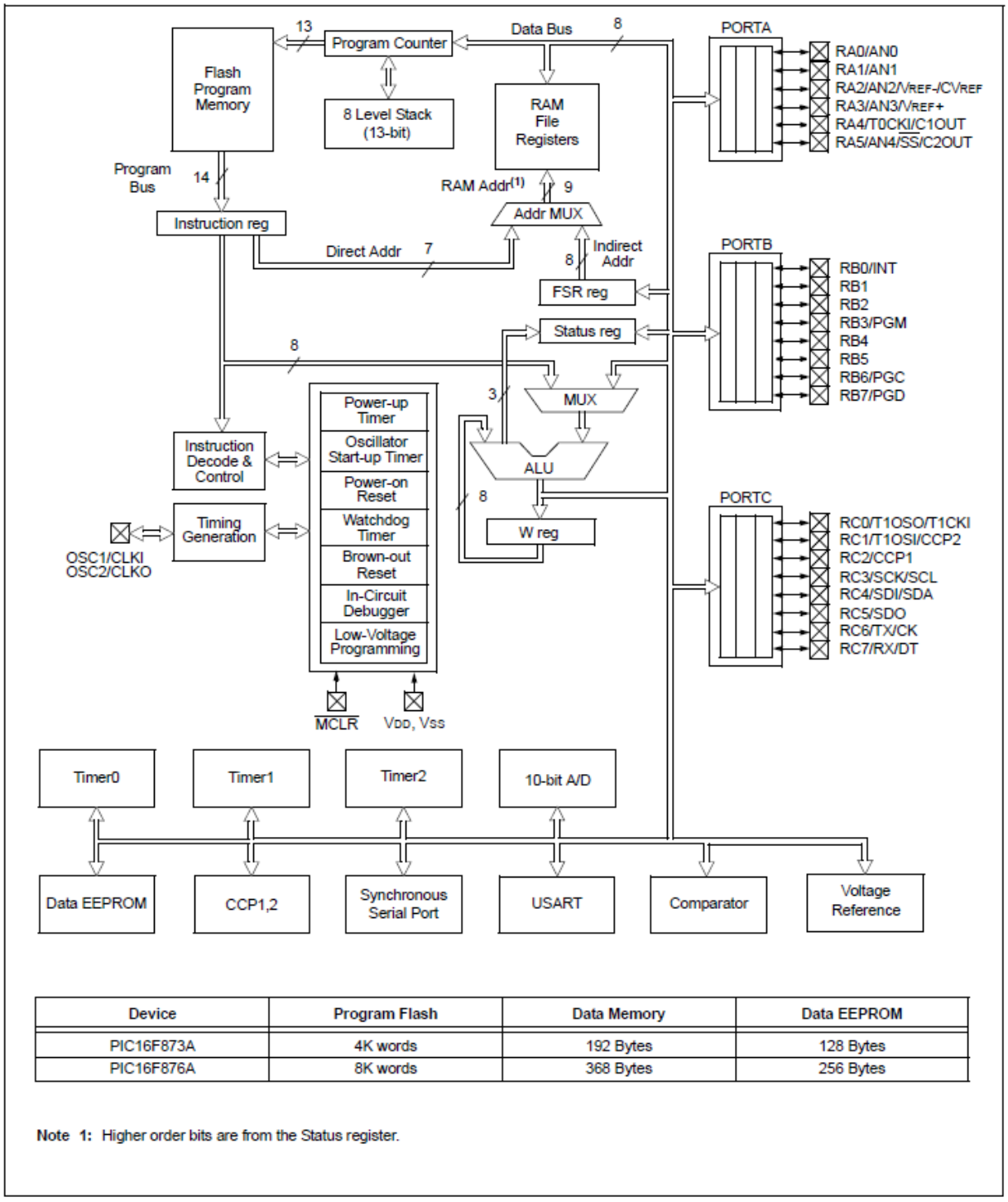
```
 UART1_Write(0x2C );UART1_Write(0x07 );UART1_Write(0xA1 );UART1_Write(0xC3 );
 UART1_Write(0x73 );UART1_Write(0x90 );UART1_Write(0xB8 );UART1_Write(0x5C );
 UART1_Write(0x76 );UART1_Write(0x83 );UART1_Write(0xE6 );UART1_Write(0xF4 );
 UART1_Write(0x37 );UART1_Write(0xBB );UART1_Write(0xEC );UART1_Write(0x0E );
 UART1_Write(0x81 );UART1_Write(0x00 );UART1_Write(0x01 );UART1_Write(0x45 );
 UART1_Write(0x0F );UART1_Write(0x30 );
}  // end method

void main()
{
   // Analogs are off
   ADCON1=0x0F;
   CMCON=0x07;
   // portb output
   TRISB=0x00;
   // TX output, RX input
   TRISC.F7=1;
   TRISC.F6=0;

   // Initialize Uart at 115200 baud rate
   UART1_Init(115200);

   // Wait for Uart to stablize
   delay_ms(100);

  do
  {
          // Start protocol
            startfbus();
          // Wait for uart to stablize
            delay_ms(100);

          // Message 1 sending
          // Refresh link
            refresh();
          // Send message 1
          mess();
          // Stabilize
            delay_ms(100);

          //Message 2 sending
          // Refresh Link
```

```
            refresh();
           //Send message 2
            burglar_alert();
           // Stabilize
            delay_ms(100);

  } while(0);//Execute Once

  // If data is ready, read it:
  if (UART1_Data_Ready() == 1) {
          // Recieved parameter
          char r;
          // Assign received data to a variable r
          r = UART1_Read();
          // display data on portB
          portB = r;

  }  // end if

}  // end main
```

Appendix H: Data Sheets

I. PIC16F877A

FIGURE 1-1: PIC16F873A/876A BLOCK DIAGRAM

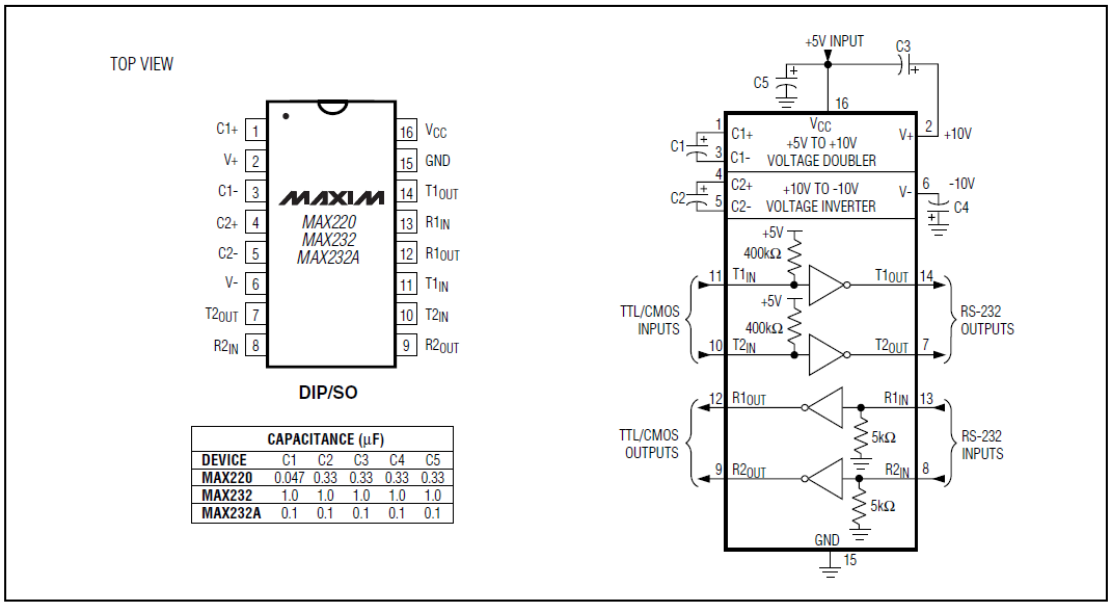


Device	Program Flash	Data Memory	Data EEPROM
PIC16F873A	4K words	192 Bytes	128 Bytes
PIC16F876A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

II. MAX232

+5V-Powered, Multichannel RS-232 Drivers/Receivers



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

III. MAX485

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

Pin Description

PIN					NAME	FUNCTION
MAX481/MAX483/ MAX485/MAX487/ MAX1487		MAX488/ MAX490		MAX489/ MAX491		
DIP/SO	µMAX	DIP/SO	µMAX	DIP/SO		
1	3	2	4	2	RO	Receiver Output: If A > B by 200mV, RO will be high; If A < B by 200mV, RO will be low.
2	4	—	—	3	$\overline{RE}$	Receiver Output Enable. RO is enabled when $\overline{RE}$ is low; RO is high impedance when $\overline{RE}$ is high.
3	5	—	—	4	DE	Driver Output Enable. The driver outputs, Y and Z, are enabled by bringing DE high. They are high impedance when DE is low. If the driver outputs are enabled, the parts function as line drivers. While they are high impedance, they function as line receivers if $\overline{RE}$ is low.
4	6	3	5	5	DI	Driver Input. A low on DI forces output Y low and output Z high. Similarly, a high on DI forces output Y high and output Z low.
5	7	4	6	6, 7	GND	Ground
—	—	5	7	9	Y	Noninverting Driver Output
—	—	6	8	10	Z	Inverting Driver Output
6	8	—	—	—	A	Noninverting Receiver Input and Noninverting Driver Output
—	—	8	2	12	A	Noninverting Receiver Input
7	1	—	—	—	B	Inverting Receiver Input and Inverting Driver Output
—	—	7	1	11	B	Inverting Receiver Input
8	2	1	3	14	V_{CC}	Positive Supply: 4.75V ≤ V_{CC} ≤ 5.25V
—	—	—	—	1, 8, 13	N.C.	No Connect—not internally connected

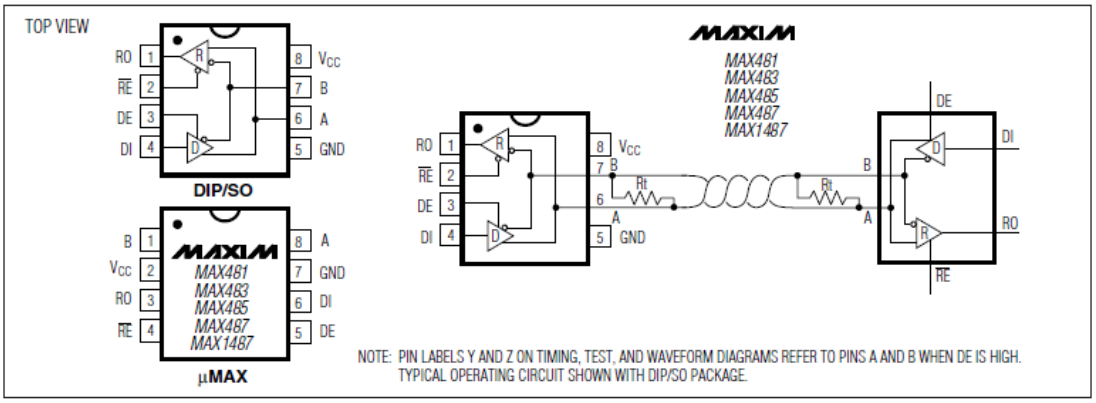


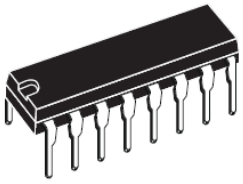
Figure 1. MAX481/MAX483/MAX485/MAX487/MAX1487 Pin Configuration and Typical Operating Circuit

IV. ULN2003

ULN2001A-ULN2002A
ULN2003A-ULN2004A

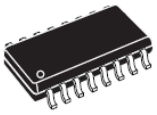
SEVEN DARLINGTON ARRAYS

- SEVEN DARLINGTONS PER PACKAGE
- OUTPUT CURRENT 500mA PER DRIVER (600mA PEAK)
- OUTPUT VOLTAGE 50V
- INTEGRATED SUPPRESSION DIODES FOR INDUCTIVE LOADS
- OUTPUTS CAN BE PARALLELED FOR HIGHER CURRENT
- TTL/CMOS/PMOS/DTL COMPATIBLE INPUTS
- INPUTS PINNED OPPOSITE OUTPUTS TO SIMPLIFY LAYOUT



DIP16

ORDERING NUMBERS: ULN2001A/2A/3A/4A



SO16

ORDERING NUMBERS: ULN2001D/2D/3D/4D

DESCRIPTION

The ULN2001A, ULN2002A, ULN2003 and ULN2004A are high voltage, high current darlington arrays each containing seven open collector darlington pairs with common emitters. Each channel rated at 500mA and can withstand peak currents of 600mA. Suppression diodes are included for inductive load driving and the inputs are pinned opposite the outputs to simplify board layout.

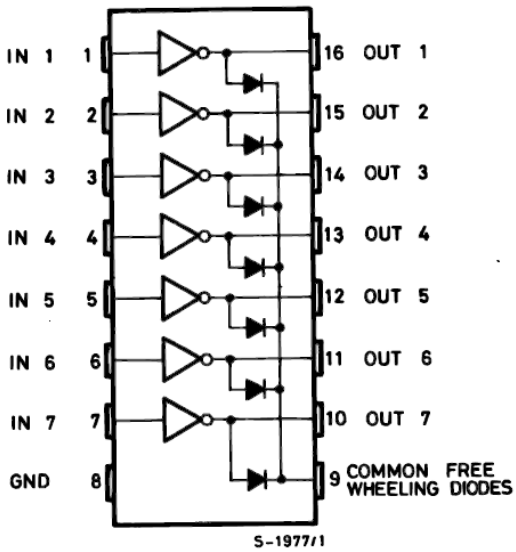
The four versions interface to all common logic families :

ULN2001A	General Purpose, DTL, TTL, PMOS, CMOS
ULN2002A	14-25V PMOS
ULN2003A	5V TTL, CMOS
ULN2004A	6–15V CMOS, PMOS

These versatile devices are useful for driving a wide range of loads including solenoids, relays DC motors, LED displays filament lamps, thermal printheads and high power buffers.

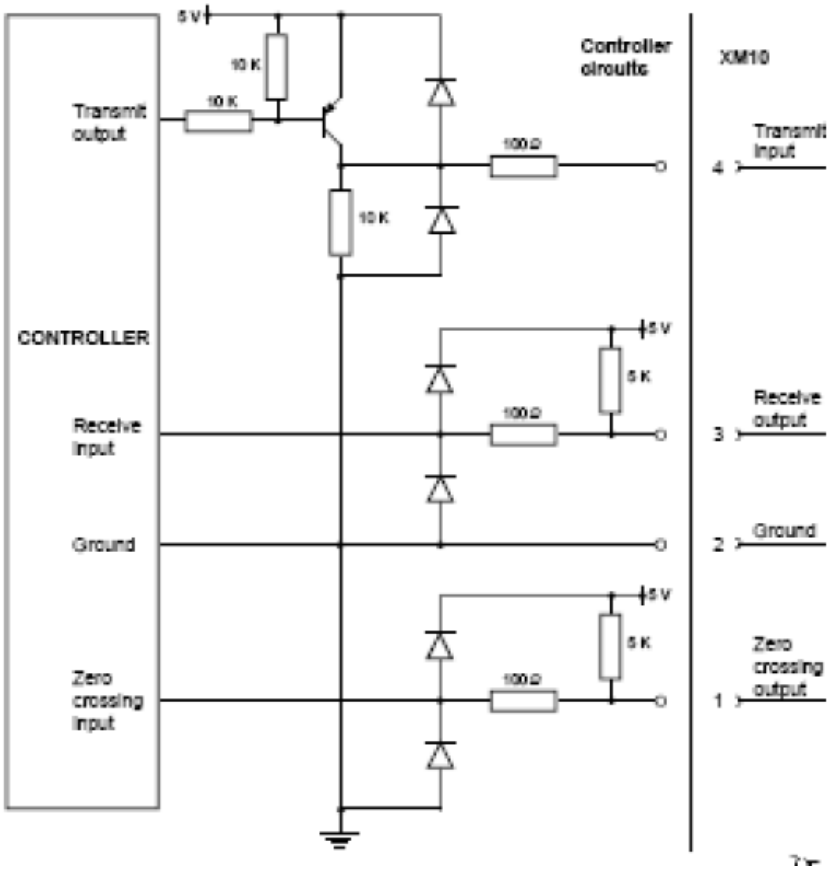
The ULN2001A/2002A/2003A and 2004A are supplied in 16 pin plastic DIP packages with a copper leadframe to reduce thermal resistance. They are available also in small outline package (SO-16) as ULN2001D/2002D/2003D/2004D.

PIN CONNECTION



V. XM10

Typical Controller Connection Diagram



Index