

BUILDING AUTOMATION THROUGH WEB INTERFACE

CHEAH JUN HONG

**A project report submitted in partial fulfilment of the
requirements for the award of the degree of
Bachelor (Hons.) of Electrical and Electronic Engineering**

**Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

April 2012

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : CHEAH JUN HONG

ID No. : 08UEB04686

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**BUILDING AUTOMATION THROUGH WEB INTERFACE**” was prepared by **CHEAH JUN HONG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor: Mr. See Yuen Chark

Date : _____

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of University Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2012, Cheah Jun Hong. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Mr. See Yuen Chark for his invaluable advice, guidance and his enormous patience throughout the development of the project.

In addition, I would also like to express my gratitude to my loving parents who have financially supportive and friends who had helped and given me ideas and encouragement throughout the project.

BUILDING AUTOMATION THROUGH WEB INTERFACE

ABSTRACT

Many times people are forgetful when it comes to switching off appliances that are no longer in used. By the time we remember that the appliance has not been switched off, the switch may be too far away, out of immediate reach. With building automation and remote control, appliances can be switched on or off without physically switching them on by their switches. In this project an embedded web server is developed to achieve remote control of switches and autonomous control with predefined rules. Web server is used instead of conventional approach where the controls of switches are performed through the web browser via local area network or the internet. The embedded server is developed from a Microchip PIC24FJ256GB106 16-bit microcontroller together with an Ethernet controller ENC28J60 using TCP/IP stack provided by Microchip and utilize USB flash drive as storage medium. This report discusses on the testing of components that make up the system, together with problems encountered and methods to solve the problems. Performance of the embedded webserver is obtained and documented. Optimizations were performed to increase the system's performance. The system is designed to use JavaScript to emulate HTTP digest authentication by using a one way hash function to encode the salted username and password for user authentication purpose. Power control modules were designed and developed to control AC power devices. AC light bulbs were used to demonstrate the capability of the system to operate in AC environment.

TABLE OF CONTENTS

DECLARATION	ii
APPROVAL FOR SUBMISSION	iii
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	x
LIST OF FIGURES	xi
LIST OF SYMBOLS / ABBREVIATIONS	xiii
LIST OF APPENDICES	xiv

CHAPTER

1	INTRODUCTION	1
	1.1 Background	1
	1.2 Motivation and Problem Statements	2
	1.3 Project Scope	3
	1.4 Project Objectives	3
2	LITERATURE REVIEW & UNDERLYING TECHNOLOGIES	5
	2.1 Building Automation	5
	2.2 Communication Medium	7
	2.2.1 X10	7
	2.2.2 INSTEON	8
	2.2.3 Universal Powerline Bus	8
	2.2.4 ZigBee	9

2.3	TCP/IP	9
2.4	Embedded Web Server	10
3	METHODOLOGY / PROPOSED SYSTEM	12
3.1	Hardware	12
3.1.1	Microcontroller	13
3.1.2	Ethernet Module	14
3.1.3	MagJack	15
3.1.4	Data Storage	16
3.2	Software	17
3.3	Flowcharts	18
4	RESULTS AND DISCUSSIONS	21
4.1	Testing	21
4.1.1	Microcontroller	21
4.1.2	Ethernet Module	22
4.1.3	TCP/IP Stack	23
4.1.4	Web Server	25
4.1.5	USB	26
4.1.6	Dimmer Circuit	27
4.2	Results	29
4.2.1	Performance	29
4.2.2	User Interface	31
4.2.3	Dimmer	37
4.2.4	Power Consumption	39
4.3	Discussions	40
4.3.1	HTTP	40
4.3.2	HTML	41
4.3.3	SPI	42
4.3.4	Authentication and Authorisation	43
4.3.5	Input Capture and Output Compare Module	45
4.3.6	Encryption	45
4.3.7	Event Logging	46

4.3.8	Dimmer Circuit	47
4.3.9	Relay Module	49
4.3.10	Optimizations	49
4.4	Design and Development Flow	50
4.5	Cost	52
5	CONCLUSION AND RECOMMENDATIONS	54
5.1	Conclusion	54
5.1.1	Personal Breakthrough	55
5.2	Recommendations	55
5.2.1	Real-time Clock	55
5.2.2	HTTP Redirection	56
5.2.3	TCP/IP Stack Modifications	56
5.2.4	Microchip dsPIC Microcontroller	57
5.2.5	Event logging	57
5.2.6	Social Network Integration	57
	REFERENCES	58
	APPENDICES	61

LIST OF TABLES

TABLE	TITLE	PAGE
1.1	Electricity Consumption of Malaysia From Year 1971 to Year 2008	3
4.1	Page Loading Benchmark	30
4.2	Page Loading Benchmark After Optimization	30
4.3	Power Consumption	40
4.4	Comparison of Different Types of Authentication Methods	43
4.5	Bill of Materials	52

LIST OF FIGURES

FIGURE	TITLE	PAGE
2.1	Example of X10 binary 1 and 0 signal	7
2.2	Four Layers of TCP/IP Model	10
3.1	PIC24F Architecture Block Diagram	13
3.2	ENC28J60 Ethernet Termination and External Connections	15
3.3	Schematic Diagram of RJ-45 Ethernet MagJack	16
3.4	General Operation Flowchart	19
3.5	User Authorisation Process	20
4.1	Schematic Diagram to Test Microcontroller	21
4.2	Schematic Diagram to Test Ethernet Module	23
4.3	Schematic Diagram to Test Ethernet Module	24
4.4	Screenshot of Ping Result	25
4.5	State Diagram of Microchip USB stack	27
4.6	Schematic Diagram of AC Dimmer circuit	27
4.7	Schematic Diagram of Zero-Crossing Detector	28
4.8	100Hz Zero Crossing Pulse	28
4.9	Firebug Showing Total Time Used to Load Timer Page	29
4.10	Maximum Download Speed	31
4.11	Screenshot of Login Page	33

4.12	Screenshot of Status Page	34
4.13	Screenshot of Setup Page	34
4.14	Screenshot of Control Page	35
4.15	Screenshot of Timer Page	35
4.16	Screenshot of Advanced Settings Page	36
4.17	Screenshot of Logout Page	36
4.18	Zero Crossing Detector Output with Mains Sinusoid	37
4.19	Phase Trigger Waveform for Minimum Light Intensity	37
4.20	Light Bulb at Minimum Intensity	38
4.21	Phase Trigger Waveform for 50% Intensity	38
4.22	Light Bulb at 50% Intensity	38
4.23	Phase Trigger for Maximum Intensity	39
4.24	Light Bulb at 100% Intensity	39
4.25	AC Dimmer Circuit with PC123 Opto-isolator	48
4.26	AC Dimmer Circuit with Transistor	48
4.27	Relay Circuit with Transistor	49
4.28	Design and Development Work Flow	51

LIST OF SYMBOLS / ABBREVIATIONS

API	Application programming interface
BAS	Building automation system
BPSK	Binary Phase Shift Keying
EEPROM	Electrically Erasable Programmable Read-Only Memory
DHCP	Dynamic Host Configuration Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
I/O	Input Output
I2C	Inter-Integrated Circuit
ICMP	Internet Control Message Protocol
LAN	Local Area Network
PHP	Hypertext Pre-processor
POSIX	Portable Operating System Interface
REST	Representational state transfer
SD	Secure Digital
SPI	Serial Peripheral Interface Bus
TCP/IP	Transmission Control Protocol and Internet Protocol
TQFP	Thin Quad Flat Pack
URL	Uniform resource locator
USB	Universal Serial Bus
WAN	Wide Area Network
XML	Extensible markup language

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Programme Listing	61
B	Schematic Diagram	83
C	Pictures of Product	84

CHAPTER 1

INTRODUCTION

1.1 Background

Building automation system (BAS) also referred to as intelligent building system. Intelligent building generally consists of Communications Networks and Office Automation, Building Management System, and Integrated Services Infrastructure (Afra Technical Solutions Est., 2009). BAS is an example of distributed control system where the control unit is not centralized at one central location. Each component sub-system is being controlled by one or more controllers. These controllers are then connected through networks for communication. These controllers have different capabilities to control devices in a building such as lightings, air handlers, alarms, public address system, or sensors.

Currently there are many manufacturers of building automation systems specialized in different types of equipment such as Alerton Technologies for heating, ventilation and air conditioning equipment(Honeywell International, 2008), Crestron Electronics, Inc. in lighting and audio/video (Essig, 2011), and Priva P.V. In Horticulture (Priva).

Communication between devices is done through wired connection or wireless connection. Examples of wired connections used by current technology are X10 and Universal Powerline Bus. Wireless communication such as Infrared, Z-Wave, Zigbee and INSTEON are also used in building automation. X10 currently dominates the majority of home automation market (Anthony, 2009a).

1.2 Motivation and Problem Statements

Building automation technologies are well established in different field of application such as automation of lighting control, alarm and security system. Building automation existed in exhibitions (Messe Frankfurt, 2011) and popular in science fiction, however due to its complexity, start-up cost, multiple incompatible standards, has resulted in limited adoption (Park Associates, 2006; Nunes, 2004) in modern buildings.

Electricity consumption is ever increasing with consumption in Malaysia grew by 27 times since year 1971 to year 2008 (International Energy Agency, 2010; Barrientos). About 50% of world's energy is estimated to be consumed by buildings ("Intelligent Building Automation conference and exhibition,"). Simple step to switch off unused appliances, to switch off lights in an empty room and reduce cooling requirement on cool weather can effectively reduce energy consumption and thus reducing utility bills. However, this might negatively impact productivity level and leads to comfort complaints.

Currently, a low cost web power switch produced by Digital Loggers, INC. is being sold at USD 119 (Digital Loggers) or approximately MYR 380. This project aims to produce a low cost building automation system and energy efficient system by utilizing existing infrastructure. Besides, Microchip provided a free licensed TCP/IP stack for its microcontroller, implementing an embedded web server can be low cost.

With web pages as control interface, it can be accessed by wide range of devices, such as mobile phones, tablet PCs, game consoles, smart televisions or desktop computers. It is essentially platform independent.

Table 1.1: Electricity Consumption of Malaysia from Year 1971 to Year 2008

Year	Value (GWh)	Year	Value (GWh)	Year	Value (GWh)	Year	Value (GWh)
1971	3,464	1981	9,901	1991	23,976	2001	69,198
1972	3,933	1982	10,789	1992	27,550	2002	71,187
1973	4,347	1983	11,642	1993	30,572	2003	74,827
1974	4,752	1984	12,588	1994	36,048	2004	78,804
1975	5,229	1985	13,655	1995	41,518	2005	81,460
1976	5,819	1986	14,676	1996	46,220	2006	86,311
1977	6,790	1987	15,788	1997	53,923	2007	93,552
1978	7,525	1988	17,552	1998	58,496	2008	94,278
1979	8,453	1989	19,215	1999	59,478		
1980	9,224	1990	21,323	2000	63,826		

1.3 Project Scope

This project aims to develop a smart building automation system. The system includes embedded web server with microcontroller and a standalone Ethernet controller which serves as a HTML web based user interface building automation control panel. The system will enable user to remotely control switches through web browser via computer network.

1.4 Project Objectives

The goal of this project is to develop an affordable web based control panel for building automation through embedded web server. Other objectives of this project are:

- To research on existing building automation technology.
- To develop an embedded web server with microcontroller for the control system.
- To develop an embedded web server with free licensed Microchip TCP/IP stack.
- To develop HTML user control interface for power devices controls.
- To develop an energy efficient building automation controller.

- To utilize existing local area network infrastructure for the control system with minimal modification to reduce total system implementation cost.
- To allow remote power switching from any devices with a web browser and Internet access.

CHAPTER 2

LITERATURE REVIEW & UNDERLYING TECHNOLOGIES

2.1 Building Automation

Automation has been utilized in different sectors, from domestic sectors to industrial sectors, ranging from a typical household living room to sophisticated factories (Han & Hwang, 2005). Applications of automation include lighting, security, HVAC, audio and video systems, energy managements, cleaning etc. (Ryan, 1989). It is estimated that by 2011 the number of home automation systems installed would reach 200 million (Alkar, Roach, & Baysal, 2010).

Building automation products not only being developed by specialized automation developers but also software developer such as Microsoft Corporation. A new feature of Windows Media Player 12 bundled in most version of Windows 7 allows remote streaming of multimedia files over the local network or internet to supported remote devices such as XBOX 360 game console, other computer running Windows 7 and other compatible devices (Microsoft, 2009). Multimedia files can be streamed and controlled all over the building by each node connected to the network.

Building automation was made for certain purposes. There is home automation made to assist elderly and disabled at home. Review shows that elderly or disabled have needs, problems or difficulties such as loneliness, preparing food, eating, cleaning, memory losses and etc. (Harmo, Taipalus, Knuuttila, Vallet, & Halme, 2005) These problems may be addressed by various form of home automation. Some problem faced by the elderly may exists in the younger generation

with busy life such as cleaning and memory problem. Home automation in the form of cleaning robot may solve cleaning problem in a building. There is gesture recognition type of home automation to assist elderly and disabled with their daily life, for example, switching on the television, closing the door, or switching a light on with just a hand gesture as describe by (Starner, Auxier, Ashbrook, & Gandy, 2000) using wearable infrared camera to track user's hand movement. To overcome memory problem such as forgetting to switch off some devices, remote monitoring and controlling system may be used to remotely switch on or off or simply to check on the status of a device remotely. However, there are very few systems available with price affordable by the public. Even if the system's price is affordable, its installation cost might be unbearable (Harmo et al., 2005).

Other usage of automation can be seen in industrial, such as pipe measurement by automated camera to measure pipe length, bending, and quality control as discussed by (Bösemann, 1996). "The optical tube measurement system is a powerful tool for fast and precise 3-dimensional reconstruction of tubes of different types and shapes." Described by Bösemann. "The system is integrated in the production line through a computer network for online process and quality control."

Automation system through internet is the most efficient way to control or monitor remote appliances like lighting, heating, cooling and etc. (Alkar et al., 2010).

Many building automation technology relies on a computer to coordinate the whole system. While some system runs on Portable Operating System Interface (POSIX) and utilize web technology (Alkar et al., 2010), some runs on Microsoft Windows, requiring proprietary software to be installed (Acson). Java Virtual Machine is also used as describe by (Corcoran & Desbonnet, 1997) , it requires java to be installed and will also works on Set-Top Box or Web-TV. These types of automation systems requires a computer running around the clock consuming modest amount of power, ranging from tens of Watts to hundreds of Watts

2.2 Communication Medium

Communications between controllers are made through wired network, wireless network or both. They use different types of protocols for communication. Some of the common standards are X10, Universal Powerline Bus, and INSTEON. There are also other types of communication method such as proprietary communication protocol on twisted pair cables (Acson). Communication between controller and devices is important to facilitate automation. Without communication, individual devices can only act as a stand-alone device.

2.2.1 X10

X10 Power Line Carrier technology was invented in late 1970s by PICO Electronics. X10 devices communicate without requiring additional control wiring, instead, the power lines were used. Signals are transmitted as a 120 kHz coded signal. A binary one is represented by a one millisecond burst of 120 kHz followed by absent of the signal at the zero crossing point and a binary zero is the absent of the signal followed by a one millisecond burst of 120kHz signal(X10).

X10 communication is slow due to its architectural design, where a bit is only transmitted during the zero crossing point of the power carrier waveform of 60Hz sine wave providing 60 bps of data transfer rate.

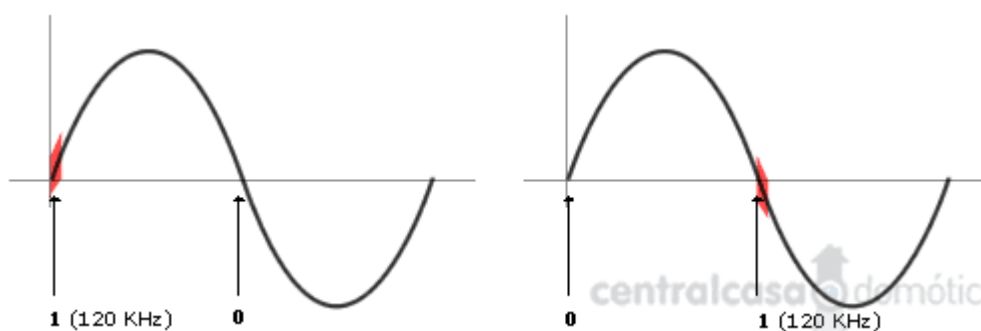


Figure 2.1: Example of X10 binary 1 and 0 signal (euro X10 CentralCasa)

Generally, X10 is a one way communication where the control unit transmit signal to receiver modules to perform remote commands as most of the X10 modules do not support two way communications(Jay, 2010). That being said, X10 communication is not very reliable compared with other type of communication method.

2.2.2 INSTEON

INSTEON was created by Smarthome. INSTEON devices can communicate via power lines and radio frequency. There are two kinds of INSTEON message. INSTEON standard message consists of 10 Bytes while INSTEON extended message consists of 24 Bytes of data. 3 Bytes of data was allocated for device address. Similarly to X10, INSTEON only transmit data during the zero crossing of the power waveform. INSTEON signal are modulated onto a carrier at 131.65 kHz through binary phase shift keying (BPSK) bit modulation. It uses 10 cycle of the carrier signal for each bit. With a binary one begins with a positive-going carrier cycle, and a binary zero begins with a negative-going carrier cycle. The raw data transfer rate of INSTEON is 2880 bps which is 48 times faster than X10 (INSTEON, 2005).

2.2.3 Universal Powerline Bus

Universal Powerline Bus (UPB) developed by Powerline Control Systems, Inc. also communicates through powerline. UPB transmits precisely timed electrical pulses onto the AC power waveform. UPB transmits 2 pulses in an AC cycle. Each pulse represents 2 bits of digital information (Powerline Control Systems, 2007). Thus, UPB is capable of achieving raw bitrate of 240 bps. 99.9% of reliability can be achieved by UPB communication compared to 70 to 80% of reliability achievable by X10 communication (Anthony, 2009b).

2.2.4 ZigBee

ZigBee serves as a low cost option for wireless communication in building automation. ZigBee is low cost and consume low power, it is capable of transmitting data at up to 250 kbps at 2.4 GHz band using quadrature phase shift keying (O-QPSK) (Ergen, 2004)that provides reliable communication at low signal to noise ratio (SNR) environment (Ploeg). High data rate is the key to low power consumption, the faster data can be transmitted, the more time can be spent idling saving energy. Its transmission range can be up to 100 meters, however, range of up to thousands of meters can be achieve by creating networks with many hops if latency is not of essential parameter of the system.

2.3 TCP/IP

Transmission Control Protocol / Internet Protocol (TCP/IP) is the communication protocol for the internet. It defines how computers communicate with each other over the internet (W3Schools). TCP/IP communication consists of layers. The protocol is defined in a series of documents known as Request for Comments (RFC). The TCP/IP model is made up of four layers as defined in RFC1122 entitled host requirement("RFC1122," 1989).

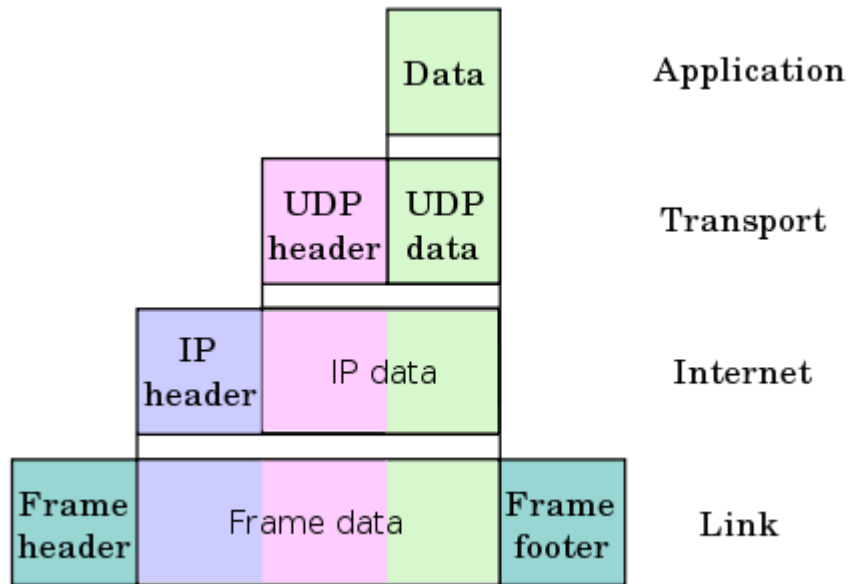


Figure 2.2: Four Layers of TCP/IP Model Note. Original artwork by Cburnett

IP is responsible for carrying packets of data from node to node. Each packet is attached with its destination address, IP will move packets to its destination based on the IP address specified ("RFC1122," 1989). TCP on the other hand, is responsible of ensuring correct data is delivered to its receiver. TCP splits data into IP packets for transmission and joins IP packets when received. At the top of this communication model is the application layer. Hypertext Transfer Protocol (HTTP) is one of the protocols in the application layer. HTTP is used for sending and receiving data between a web server and a client, often a web browser. HTTP is a cross-platform protocol and is freely available. HTTP requests together with Hypertext Markup Language (HTML) web pages allow communication between client and server for remote device control from any web enabled devices.

2.4 Embedded Web Server

Web server runs on various platforms ranging from high performance rack server to embedded devices to serve requests by clients. Web servers serve HTML documents that can be parsed and displayed by web browsers. Development of a HTML web

interface is fast and easy compared to developing specialized client software for each client's platform running incompatible operating systems and environment.

Typical requirement of an embedded web server is to have small memory footprint, and light CPU utilization. An embedded web server is specially designed for its purpose without considering additional features due to limited resources such as on board memory and processor packed into the device.

Embedded web server is sometimes packed into a device as an extension to allow remote management over the internet. For example, embedded web server embedded into a printer can serves user interface through the internet and can be access with a web browser without requiring any special proprietary software to be installed.

Embedded web server is small in size compared to a standard computer acting as a server. The simplicity of an embedded web server with little components means it is energy efficient, using only fractions of the energy required by a full size computer. Energy efficiency is important in building automation as reducing energy usage is one of the objectives of building automation (Ariane Controls).

Some of the advantages of running a building automation system on embedded web server are that they are reliable with little to no maintenance required (Ju, Choi, & Hong, 2000). There are no moving components in an embedded web server running on a microcontroller, mechanical failure can be avoided. Embedded web servers have low resource usage (Ju et al., 2000). Embedded web servers are small in size, with its small footprint, positioning the system can be flexible.

Embedded web servers are often used to transmit working status of a system. Embedded web server also relays user entered commands from web browsers to an embedded system.

CHAPTER 3

METHODOLOGY / PROPOSED SYSTEM

3.1 Hardware

This project will address some of the issues such as reducing power consumption of other building automation systems by building the system on microcontroller and an Ethernet module instead of a computer. On the other hand, web based interface increase compatibility for operation on multiple platforms.

The embedded web server will be able to serve Hypertext Markup Language (HTML) together with Extensible Markup Language (XML) documents as user interface to control and monitor device(s) operating status remotely via a web browser.

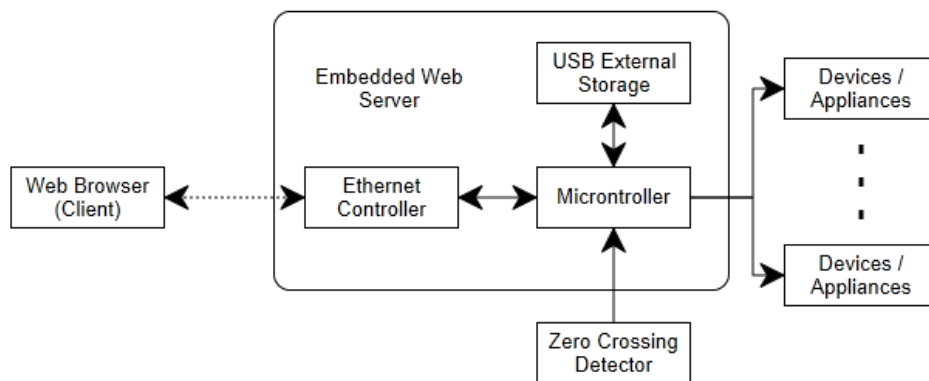


Figure 3.1: System Block Diagram

3.1.1 Microcontroller

This project will build an embedded web server from Microchip PIC24F 16bit microcontroller with low power consumption to serve as an alternative to computer based web server used in other web based building automation systems. Microcontroller acts as the central control unit, coordinating all operations in the system, such as switching on or off a connected device, serving control interface to user, and networking tasks.

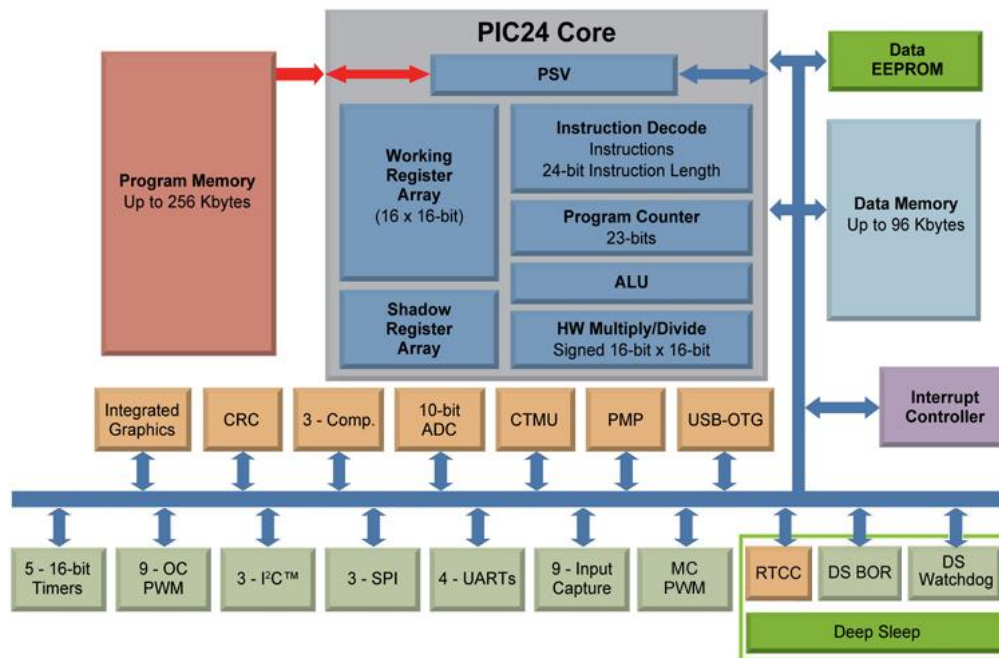


Figure 3.2: PIC24F Architecture Block Diagram (Microchip)

PIC24F family microcontroller has maximum execution rate of up to 16MIPS at 32MHz. Various peripherals are included in the microcontroller as shown in Figure 3.1. Digital I/O only pins are 5.5V tolerant which is useful for interfacing with 5V devices. PIC24F series microcontroller has built-in I²C support, I²C driver software does not have to be written. Multiple individual SPI modules are available, this allows interfacing with multiple SPI devices on each port individually. Some of the microcontrollers have deep sleep mode where the power consumption is in the Nano Watt range during deep sleep mode. The microcontroller has built in 10-bit analogue to digital converter, sensors which produce analogue signal output such as

LM34 temperature sensor may be connected directly to the microcontroller without external converter circuit.

The system is expected to be able to switch on and off appliances such as lamp, and fan through the use of digital I/O ports with external relays controlled by web based interface over the network. PIC24FJ64GA002 with 64k bytes will be used for preliminary design, it is available in dual in line through holes package where prototyping can be made easy. Microcontroller with larger program memory space such as PIC24FJ256GB106 available in Thin Quad Flat Pack (TQFP) packages will be used when extra features were to be included into the system. Extra features such as posting updates to social network, controlling lighting or cooling based on sensors input or environmental conditions may be included.

3.1.2 Ethernet Module

A pre-built Ethernet module based on a 10Mbps Microchip standalone Ethernet controller ENC28J60 will be used to interface the microcontroller to the network. Communication between the module and the microcontroller is done through SPI. The module contains all essential components for the Ethernet controller to function properly and allow direct connection to the microcontroller through SPI connection.

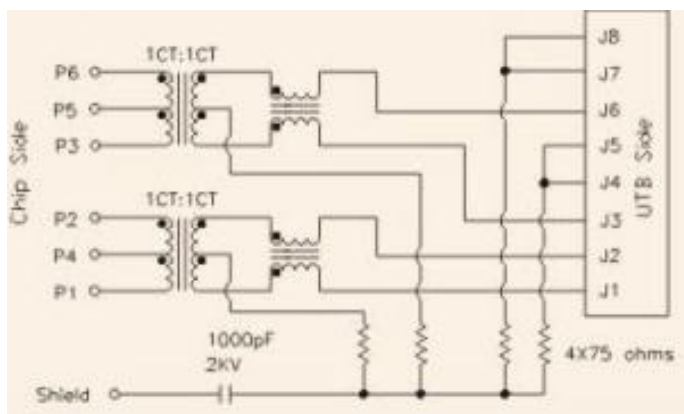


Figure 3.4: Schematic Diagram of RJ-45 Ethernet MagJack

3.1.4 Data Storage

Storage medium is required to store web pages and operating parameters of a web server. Several options such as EEPROM, internal flash memory, SD card, and USB storage are available as storage medium for this application. USB storage was chosen among the other available options due to some advantages of USB storage over the other options.

Cost of EEPROM is low. However, its data storage size is smaller compared to the others. 1Mbit is the largest available data storage space of SPI EEPROM manufactured by Microchip. The up side of using EEPROM is that its driver memory footprint is smaller compared to USB and SD card driver.

Limitation of the use of internal program flash memory to store web pages is storage size. For example, PIC24FJ256GB106 has 256KB of internal flash memory, however, this memory is intentioned for program storage, and there may not be enough memory available to store web pages with multimedia contents. In addition, if web page design were to be changed, the whole program will have to be recompiled and reprogramed to update the changes done to the web pages.

SD card comes in 3 different form factors: the original size, the mini size and the micro size. SD card are used in various types of portable devices as it is small in

size, relatively low cost and has huge data storage size. Editing of web pages can be performed easily, and edited web pages can be deployed quickly without having to reprogram the microcontroller if the editing does not involve dynamic pages. In this project, it was not chosen as a storage medium because the Microchip's Memory Disk Drive (MDD) File System Library only support SD card through SPI interface. Some older SD cards do not have built in SPI support.

USB storage support from the Microchip TCP/IP stack together with USB driver and Microchip's MDD File System allows the use of USB flash drives with various data storage size together with the embedded server. Similar to SD storage medium, web pages can be edited and stored back into the flash drive effortlessly. Besides the use of USB flash drive, the stack supports the use of USB memory card reader. The use of SD card is supported through the card reader. This does not limit the type of memory cards that can be used with the system as universal memory card reader supports different types of memory cards such as Memory Stick Micro M2, miniSD card, microSD card and others. Therefore, usage of USB interface for storage medium is a better choice comparing to the other options. The MDD file system only support legacy 8.3 filename also known as short file name (SFN). The file name can be at most 8 characters long together with a maximum of 3 characters extension.

3.2 Software

Development will be completed on MPLAB by Microchip with C programming language and compiles with C30 C Compiler. The free licensed TCP/IP stack provided by Microchip will be used to implement TCP/IP protocol required for a web server. The microcontroller may serve as a web server as well as a web client made possible by the TCP/IP stack.

Although an embedded server does not support most server side scripts such as Hypertext Pre-processor (PHP) scripts for server side data manipulations, however these functions may be off-loaded to the client side through extensive use of

JavaScript on modern web browsers to generate dynamic webpages. The TCP/IP stack also supports some extents of dynamic webpage generation. It is capable of parsing HTML documents produce dynamic version of the HTML documents.

Unlike any other system which supports server side scripting, embedded systems are designed for a designated purpose, any attempt to upgrade or change of functionality will require rewriting and recompiling source codes which will then be burnt into the microcontroller. Systems which support server side languages such as PHP, Perl, Python do not requires as much work when upgrading where its source code can be edited and deployed immediately.

3.3 Flowcharts

Figure 3.5 shows the general program flow of the system while Figure 3.6 shows the user authentication process. The system will listen and wait for a connection from the client and perform actions requested by authenticated users.

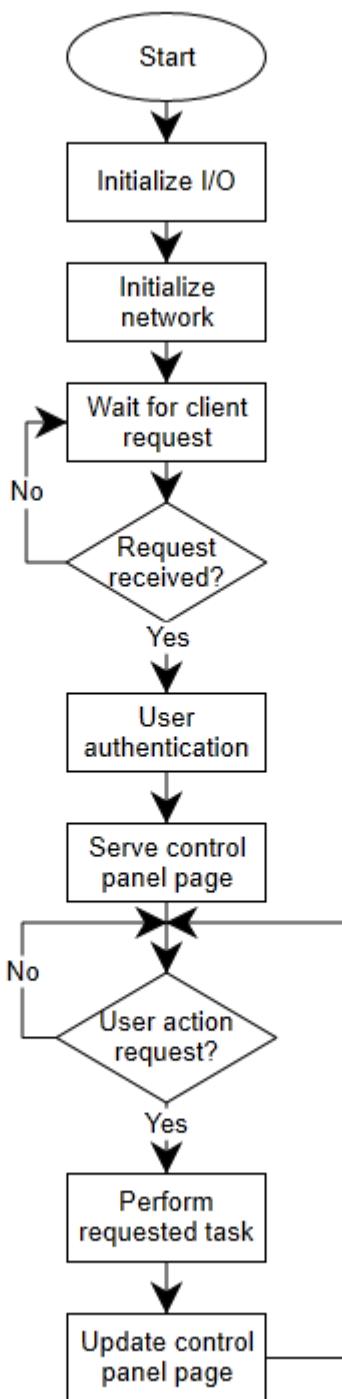


Figure 3.5: General Operation Flowchart

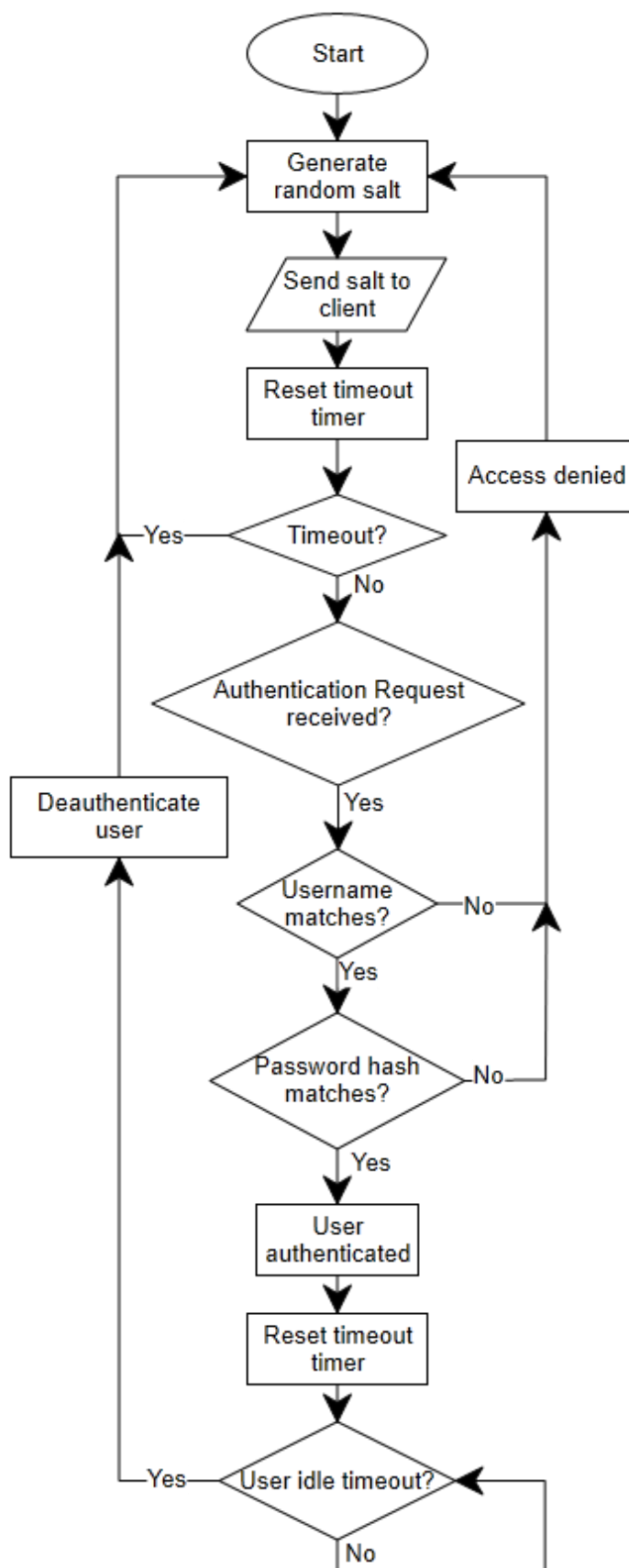


Figure 3.6: User Authentication Process

The microcontroller is configured to run with the primary external crystal oscillator. An I/O pin was configured as an output and programmed to toggle at a certain period. The LED connected to the output pin was observed. Blinking LED indicates that the microcontroller is operational.

4.1.2 Ethernet Module

The Ethernet module includes RJ-45 Magjack. There are 2 LEDs attached to the RJ-45 MagJack, they are labelled as LED A and LED B. The Ethernet module default setting will set LED A to light up when Ethernet cable is connected and LED B will blink if traffic is present. Based on the Ethernet Module datasheet, a 6.25MHz square wave can be observed at the CLKOUT pin of the Ethernet module. The schematic diagram of the Ethernet module is shown in Figure 4.2. Both LEDs were observed when RJ-45 cable was plugged into the Ethernet module and a network router. Traffic generated by standard TCP/IP protocol causes LED B to blink without user generated traffic while LED A remain stable indicating that the connection is established between the module and the network router.

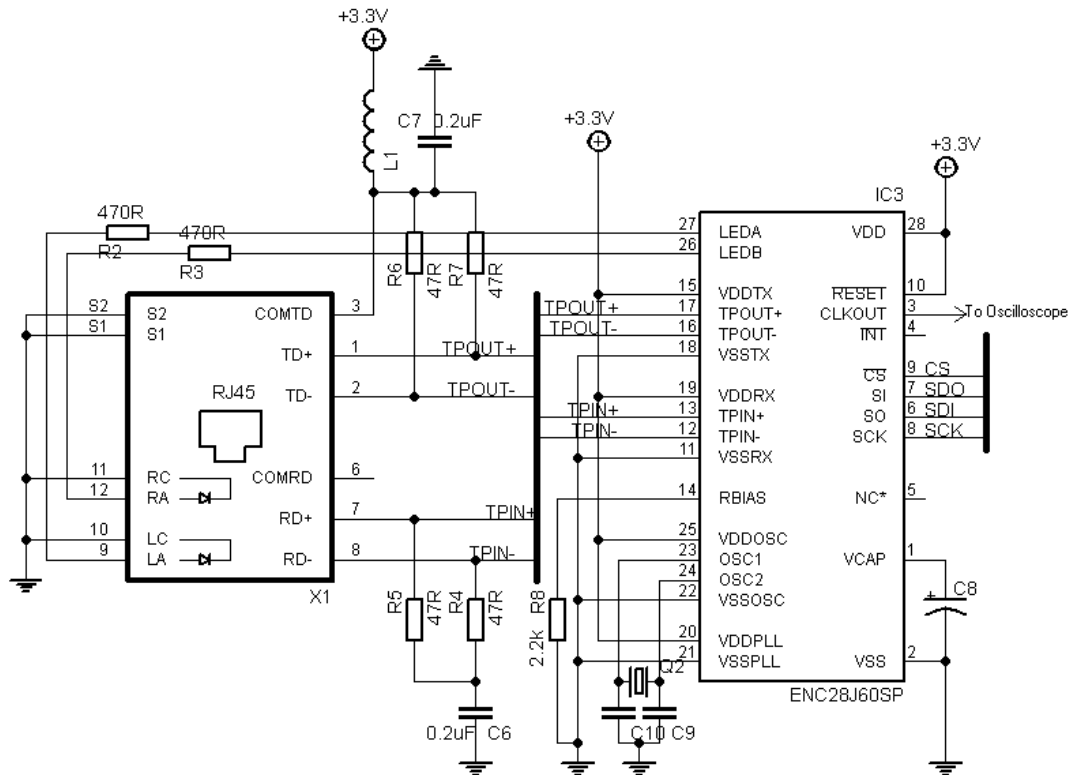


Figure 4.2: Schematic Diagram to Test Ethernet Module

4.1.3 TCP/IP Stack

In this project, the version of TCP/IP stack being used is the latest as of 1st January 2012 at version 5.36.4 released in October 2011. In order to test the stack, a working circuit is required. The schematic in Figure 4.3 shows the circuit used to test the stack and later further developed to the complete system.

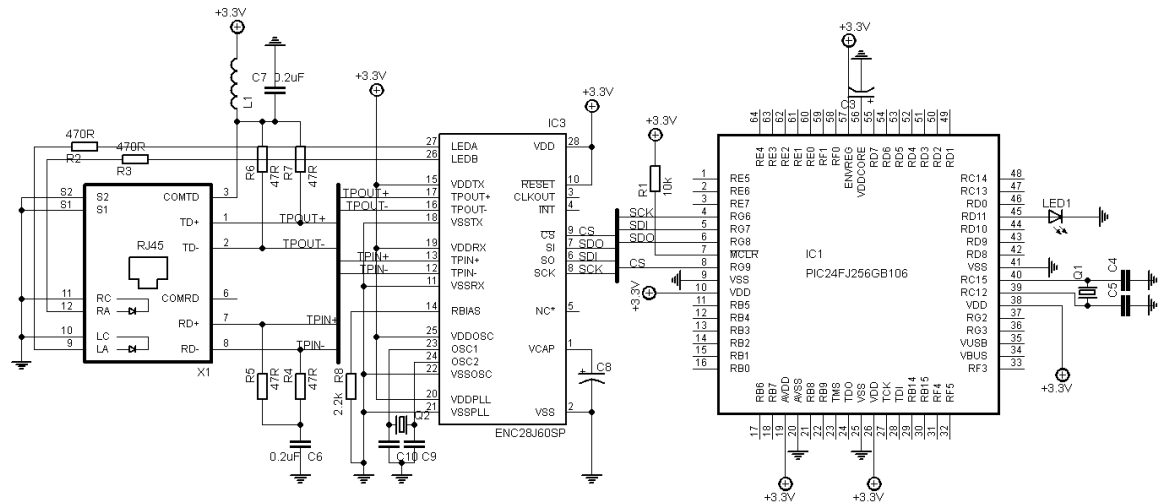


Figure 4.3: Schematic Diagram to Test Ethernet Module

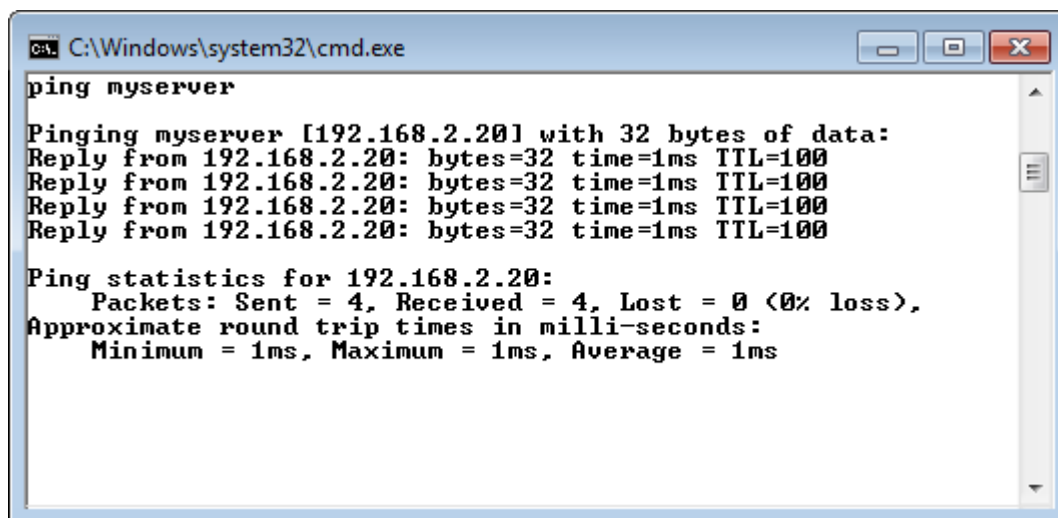
Some configurations were performed to the demo application included in the Microchip Application Libraries to suit the circuitry designed as in Figure 4.3.

I/O pins were remapped according to the circuit, unused functions in the TCP/IP stack such as the LCD, SPI EEPROM, and SPI Flash, were removed. The TCP/IP demo application included additional functions that are not required to test the circuit shown in Figure 4.3. The only functions used are the DHCP client, NetBIOS Name Service Server, and the ICMP client. DHCP client is required to get IP settings from the router, while NetBIOS Name Service Server is required to resolve the server host name in the local area network. The ICMP client is required for the server to respond to ping request.

TCP/IP configurations were performed to enable the server to work in a specific environment such as server host name, network address, subnet mask, default gateway, and DNS servers. The configuration codes are shown in Appendix A page 76.

By using ping function of a computer, pinging the host name of the server, the computer will receive reply from the server. For this project, the embedded server is configured to have “myserver” as its host name. The network router’s DHCP

server will assign the address 192.168.2.20 to the embedded server. Below is the screenshot of the ping result.

A screenshot of a Windows command prompt window. The title bar reads 'C:\Windows\system32\cmd.exe'. The command 'ping myserver' has been entered. The output shows four successful replies from 192.168.2.20, each with 32 bytes, 1ms time, and TTL=100. Below the replies, the ping statistics for 192.168.2.20 are displayed: 4 packets sent and received, 0% loss, and round trip times of 1ms minimum, maximum, and average.

```
C:\Windows\system32\cmd.exe
ping myserver

Pinging myserver [192.168.2.20] with 32 bytes of data:
Reply from 192.168.2.20: bytes=32 time=1ms TTL=100
Reply from 192.168.2.20: bytes=32 time=1ms TTL=100
Reply from 192.168.2.20: bytes=32 time=1ms TTL=100
Reply from 192.168.2.20: bytes=32 time=1ms TTL=100

Ping statistics for 192.168.2.20:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 1ms, Average = 1ms
```

Figure 4.4: Screenshot of Ping Result

4.1.4 Web Server

To enable the web server function, HTTP2 module of the TCP/IP stack is enabled. A html document is written and stored into a USB flash drive. Before the page can be served by the server, two binary index files have to be generated by a tool named “Microchip MPFS Generator” provided by microchip. These generated binary index files must be placed together with web pages in the USB flash drive due to the operating nature of the HTTP2 module in the TCP/IP stack.

Pointing the browser to the host name of the server or the IP address of the server, the browser will display the web page “index.htm”. At the same time, this test proved that the USB storage medium is operational together with the MDD file system.

4.1.5 USB

Based on USB standards, USB devices cannot communicate between each other directly without a USB host. A full featured Host is required to support any device connected to it with special driver software installed on the host. It is also a must that the host is able to supply a minimum of 100mA current to the attached device. However, for embedded host, the requirements are relaxed. Embedded host is required to support only specific class of devices.

In this project, the microcontroller also acts as an USB embedded host. It is configured to support USB mass storage medium or USB flash drives by specifying the class through Targeted Peripheral List (TPL) in the USB stack configuration shown in Appendix A page 76.

An USB device may act as a host or a device depending on its design. Here, the microcontroller only acts as a host. The USB module requires a 48 MHz clock. The microcontroller has a 96 MHz USB PLL to produce 96 MHz clock from 8 MHz internal FRC oscillator. From the 96 MHz PLL output, it is fed to a fixed divide-by-2 frequency divider to generate the 48 MHz clock required by the USB module. Meanwhile, the 96 MHz output is also fed into a divide-by-3 frequency divider to produce 32 MHz clock for system clock.

The USB stack operates as a state machine, its state machine diagram is shown in Figure 4.5.

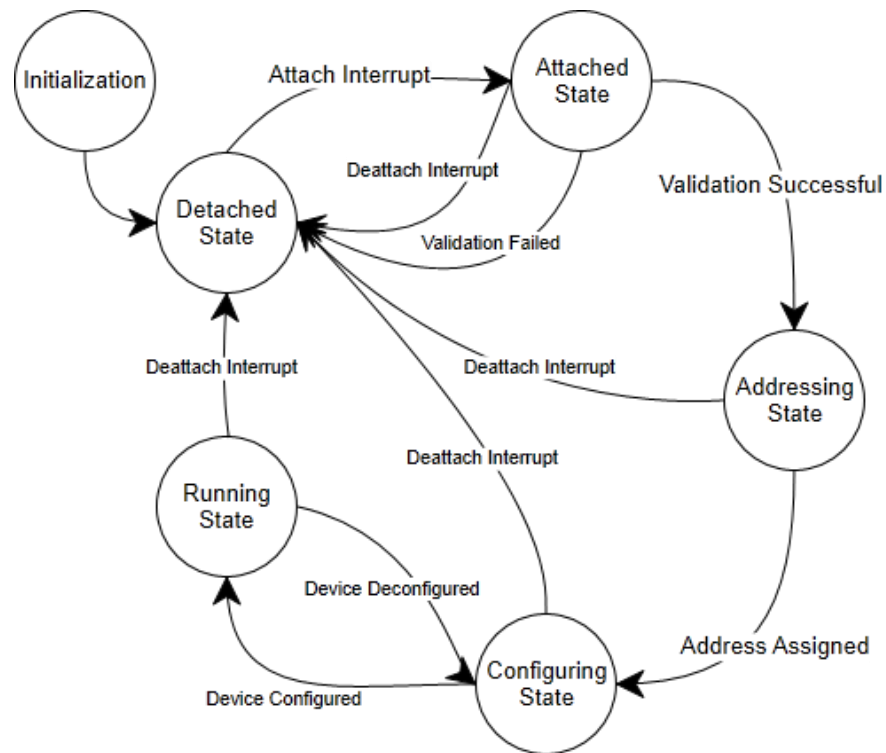


Figure 4.5: State Diagram of Microchip USB stack

4.1.6 Dimmer Circuit

The dimmer circuit consists of a TRIAC, opto-isolator, and other common electronic components.

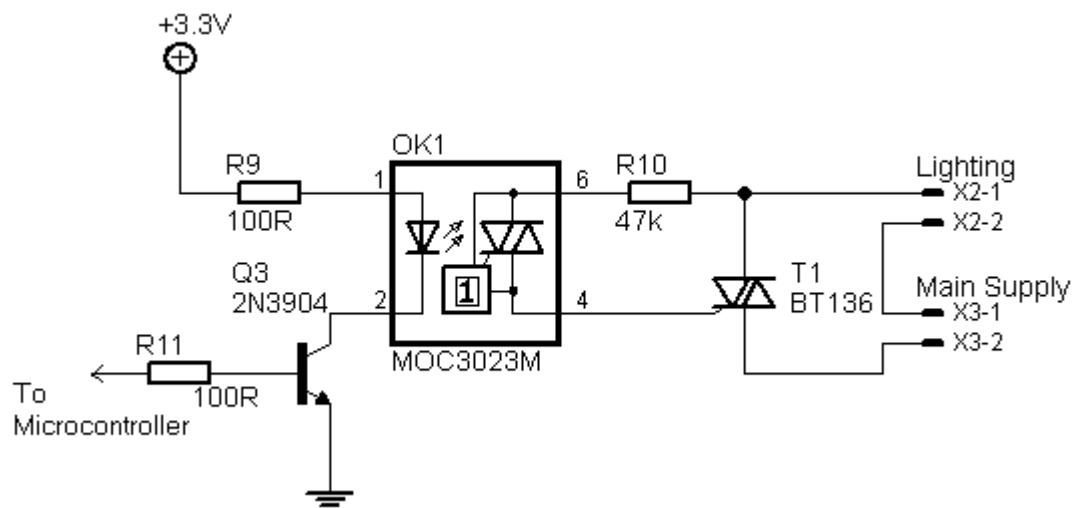


Figure 4.6: Schematic Diagram of AC Dimmer circuit

In order to test the circuit in Figure 4.6, an incandescent light bulb was connected to the circuit and the circuit is triggered by microcontroller I/O pin, when a high signal is passed, the light bulb will light up, and turn off when the signal passed is low.

To test the dimming functionality of the circuit, input capture and output compare module of the microcontroller is needed.

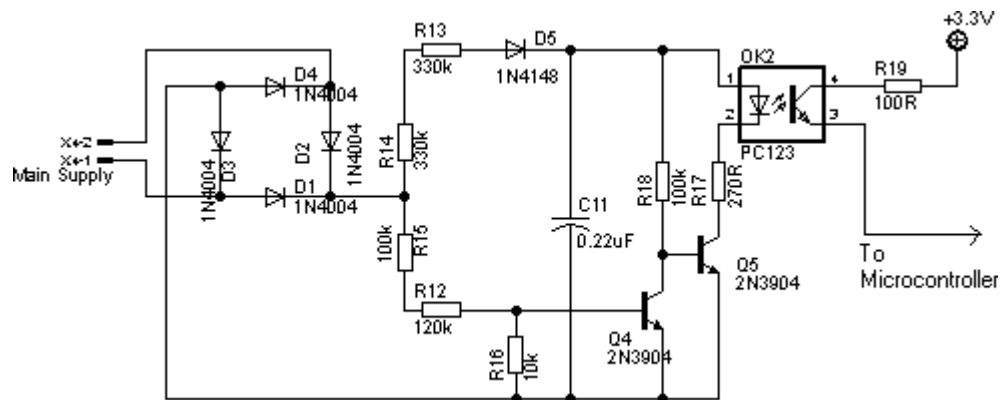


Figure 4.7: Schematic Diagram of Zero-Crossing Detector

The zero crossing detector circuit will produce a pulse whenever a zero crossing is detected at the main power supply. By connecting the output of the circuit to an oscilloscope, a 100Hz pulse can be observed. The pulse is shown in Figure 4.8.

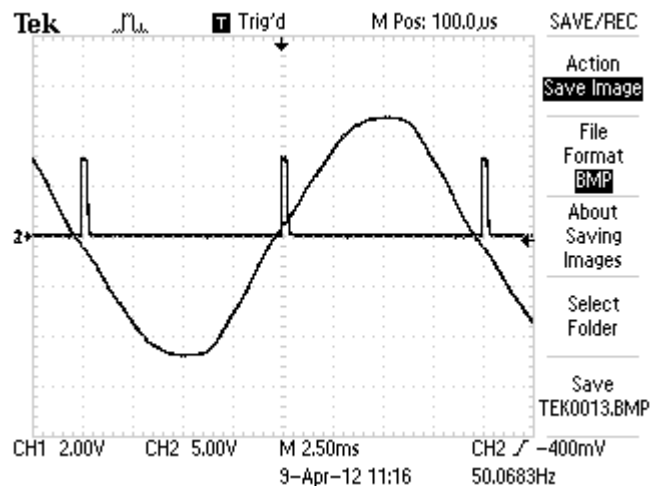


Figure 4.8: 100Hz Zero Crossing Pulse

4.2 Results

4.2.1 Performance

Page load time is determined by various factors. Times taken to serve and load each of the pages were tested using Firebug. Firebug is a web development tool integrates with Firefox browser to debug and develop web pages. It is capable of tracking page load time with in depth details of the page loading process.

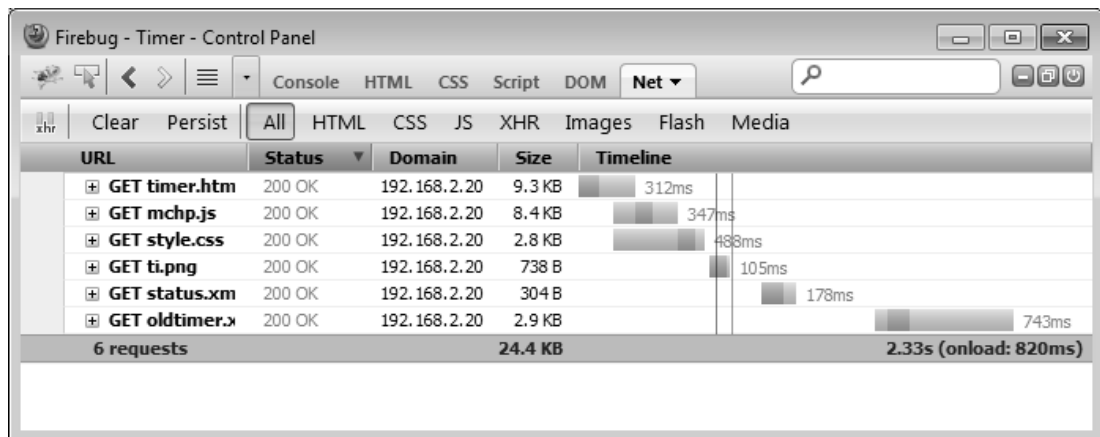


Figure 4.9: Firebug Showing Total Time Used to Load Timer Page

In Figure 4.9, Firebug shows that the timer page requires 6 HTTP requests to the server. The total page load time is 2.33 seconds, total document size 24.4 KB. It also shows the timeline of the page loading process.

Data obtained before optimization and after optimization were tabulated in Table 4.1 and Table 4.2 respectively.

Table 4.1: Page Loading Benchmark

Page	Number of requests	Total document size (KB)	Total page loading time (ms)
Login	5	19.5	1010
Status	4	15.9	969
Setup	4	18.9	962
Control	4	18.7	940
Timer	6	24.4	2330
Advanced settings	5	24.3	1080
Logout	4	17.1	738

Table 4.2: Page Loading Benchmark After Optimization

Page	Number of requests	Total document size (KB)	Total page loading time (ms)
Login	2	12.3	572
Status	2	14.8	798
Setup	2	13.9	661
Control	2	12.7	637
Timer	4	19.2	1780
Advanced settings	2	18.0	674
Logout	1	0.8	114

Optimizations include removal of unused code such as unused JavaScript, CSS selectors and unnecessary line breaks. Other than that, external JavaScript and CSS style sheet were embedded as inline codes in the HTML files itself.

Average page load time before optimization was 1147ms and after optimization was 748ms. Time required to load page was reduced by 34.79%. Average page size before optimization was 19.83 KB and after optimization was 13.1 KB. Average page size was reduced by 33.94%.

The maximum transmission throughput of the embedded server is about 65kBytes/sec. This was tested by downloading a 3.5MB image file from the server.

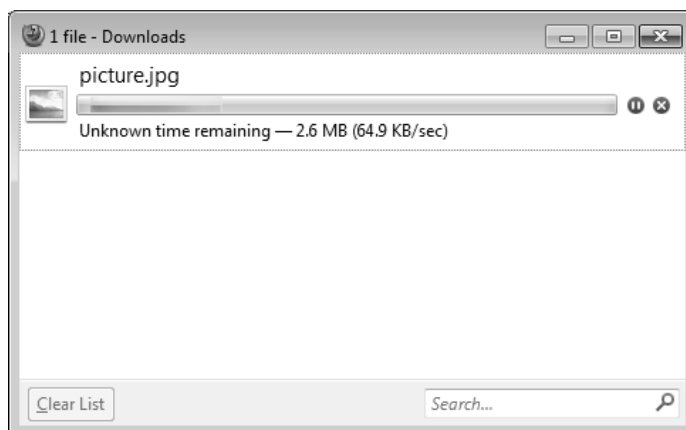


Figure 4.10: Maximum Download Speed

4.2.2 User Interface

This project utilize web based control, therefore the user interface is made up of web pages. Each page is an HTML file with file extension of “.htm”. Figure 4.11 to Figure 4.17 show the web based user interfaces.

Login page shown in Figure 4.11 will be presented to unauthenticated user. Users are required to login to use the system. User login is handled by JavaScript dynamically in the background.

Once the user have successfully authenticate through the login page, the user will be redirected to the status page shown in Figure 4.12. Here the name, type, and status of the devices attached to the system are displayed. The web page will continuously request the server for updated status every 2 seconds. This process is performed by JavaScript to request for device status updates and preventing page reloading every 2 seconds, conserving bandwidth.

Figure 4.13 shows the setup page, at this page, the user will be able to rename the devices, and to change the device type. Types available for the use to choose are

“On/ off”, “Fan”, “Light”, and “Pulse Triggered”. Figure 4.14 shows the control page, where the user is allowed to control the devices here. Different types of devices have different control method. For “On/off” type, the user is able to switch the device either on or off. This is more suitable to be used with the relay module. “Fan” type will allow user to choose from 5 speeds, to control speed of fan or motor, it is used with the dimmer module. “Light” will allow user to select from 10 step of intensity when used with the dimmer module. Lastly, for “pulse triggered”, the device will receive a 100ms pulse to either turn it on or off, some of the application of this is door bell, computer power supply, and auto gate.

Figure 4.15 shows the timer page. There are 5 sets of timer for each device. User will be able to set the device to perform a particular action such as to switch on, or switch off, or adjusting fan speed, adjusting light intensity, and to send a pulse at designated time.

Figure 4.16 shows the advanced setting page. In this page, the user will be able to configure some advanced settings. User can reset the system to its default configurations, select to turn off all devices at system startup, to change login username, to change login password and to change the server time zone. Password change requires the user to know the shared secret between the server and the user. The new password will be transmitted in encrypted form using Corrected Block TEA (XXTEA) algorithm, with the shared secret as encryption key. It is important to maintain the secrecy of the shared secret as the exposure of the secret key will allow hackers to gain access to the system and changing the password. The encrypted string is encoded into pairs of hexadecimal characters and transmitted to the server.

In order to ensure the user encrypted the new password with the correct secret key, the secret key entered by the user is encrypted with the entered secret key itself and transmitted to the server. The server will first validate the encrypted secret key. If the secret key is invalid, the new password will not be processed. This is absolutely essential to prevent user from being locked out from the system when the server decrypt the new password with an unmatched secret key.

Figure 4.17 shows the logout page, it is shown when user click on the logout link. In this page, user can click on the button to return to the login page.

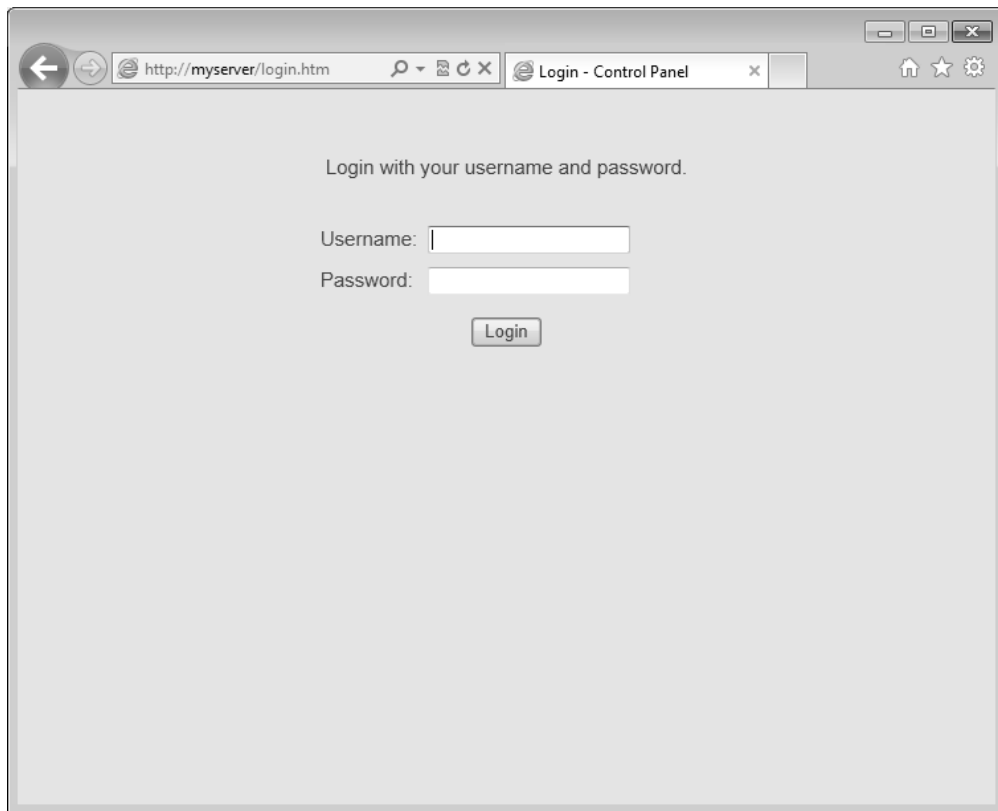


Figure 4.11: Screenshot of Login Page

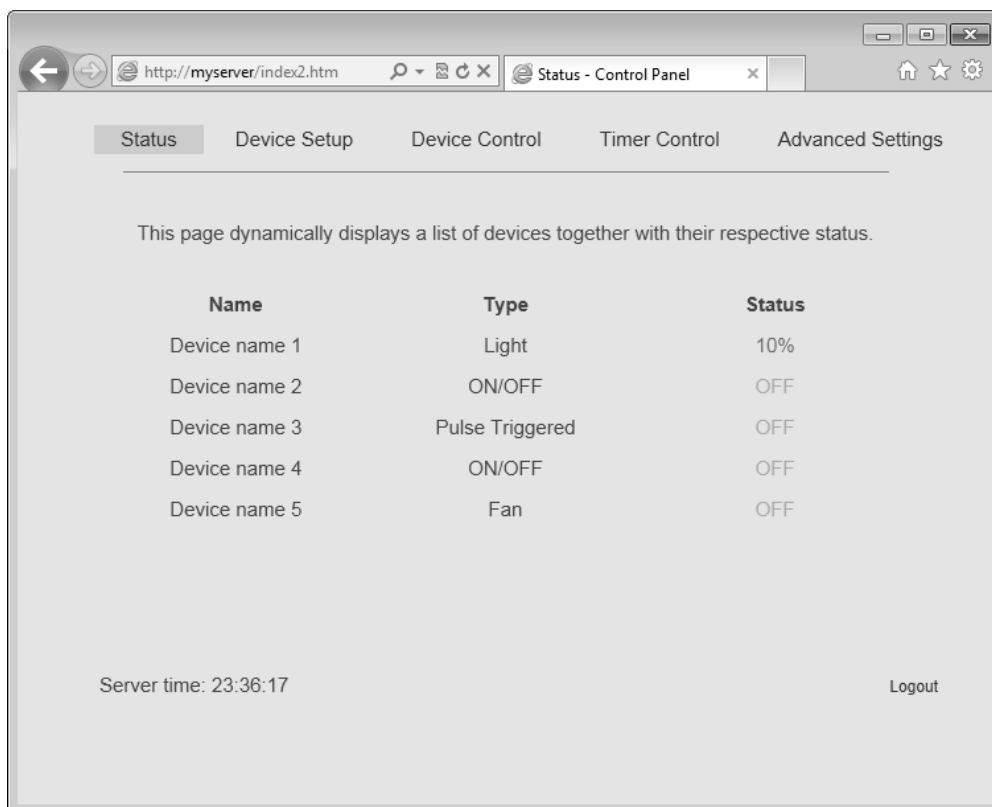


Figure 4.12: Screenshot of Status Page

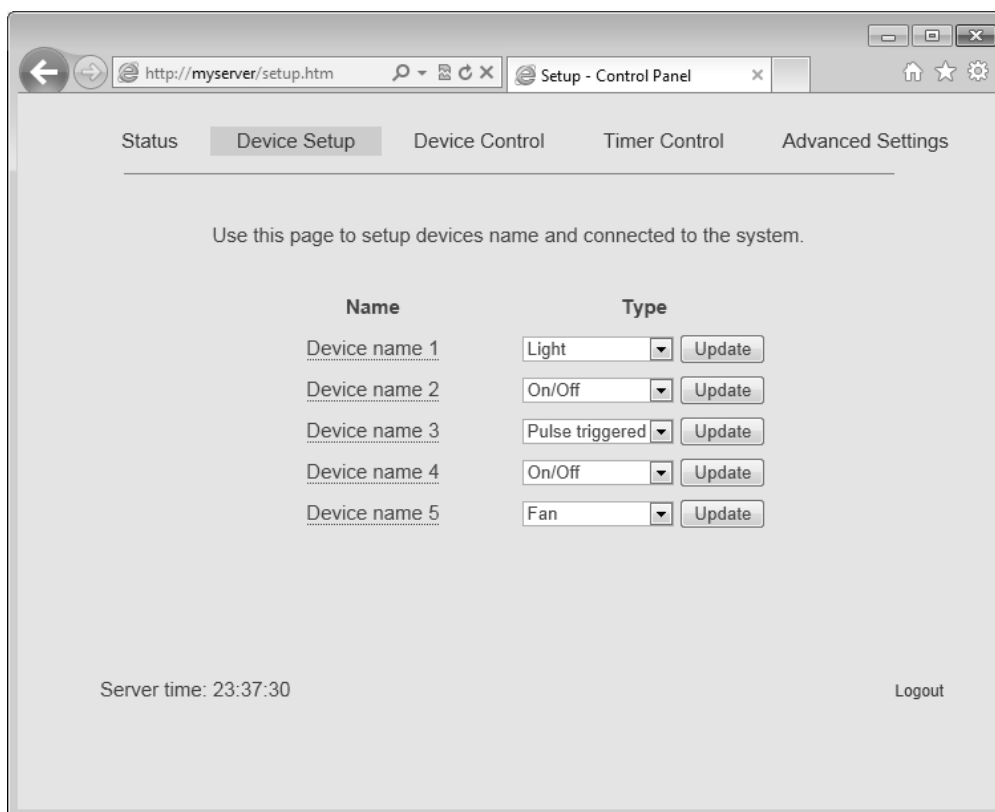


Figure 4.13: Screenshot of Setup Page

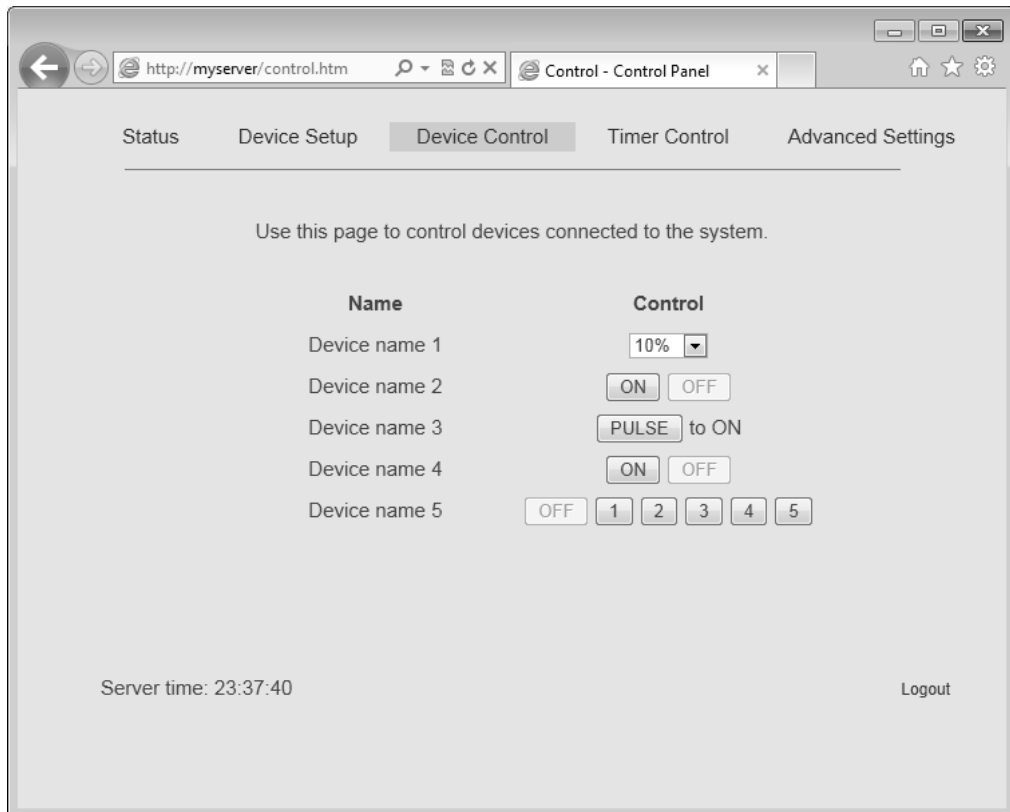


Figure 4.14: Screenshot of Control Page

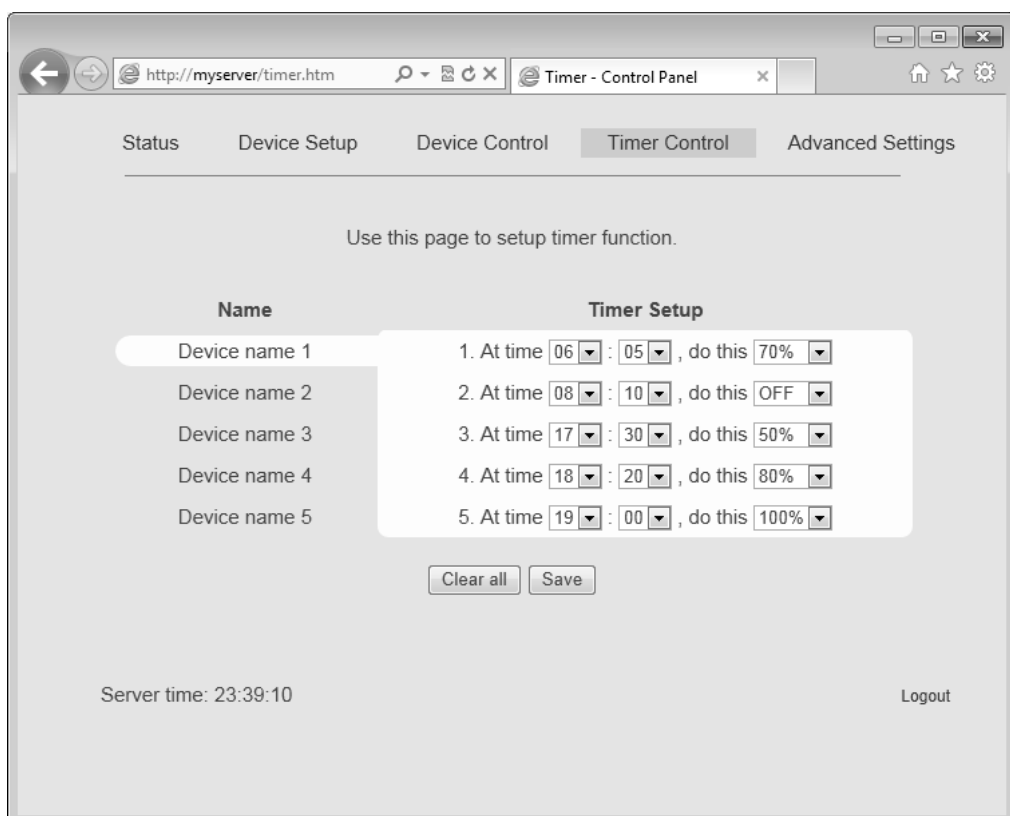


Figure 4.15: Screenshot of Timer Page

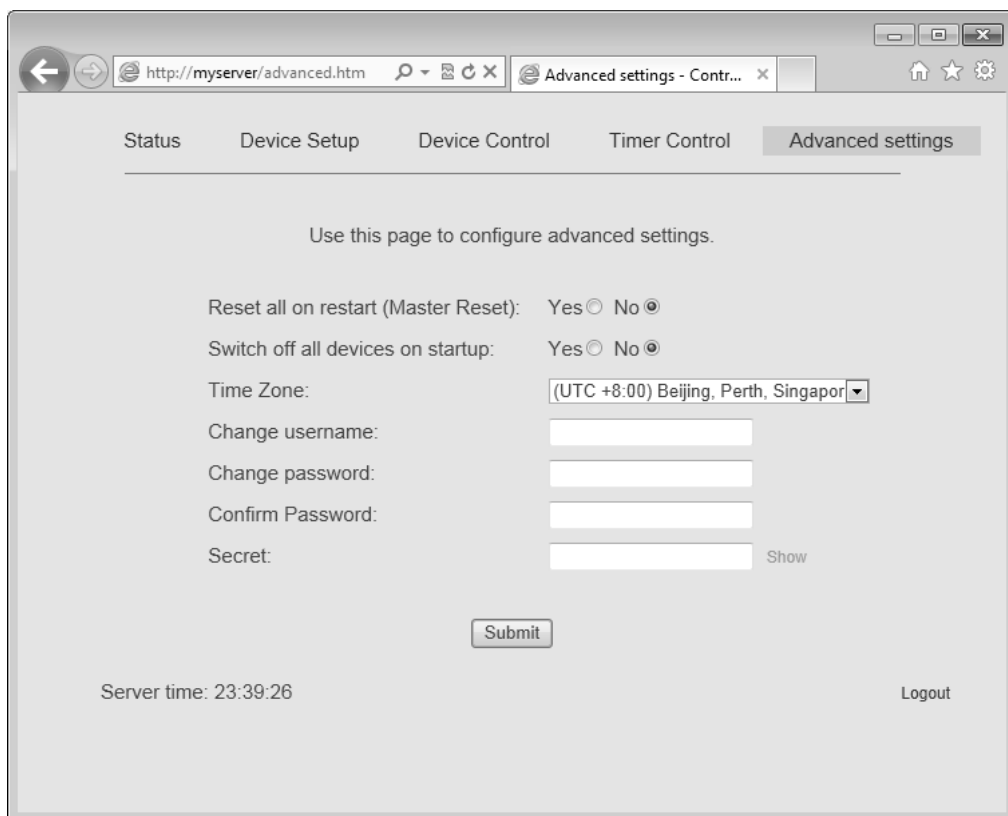


Figure 4.16: Screenshot of Advanced Settings Page

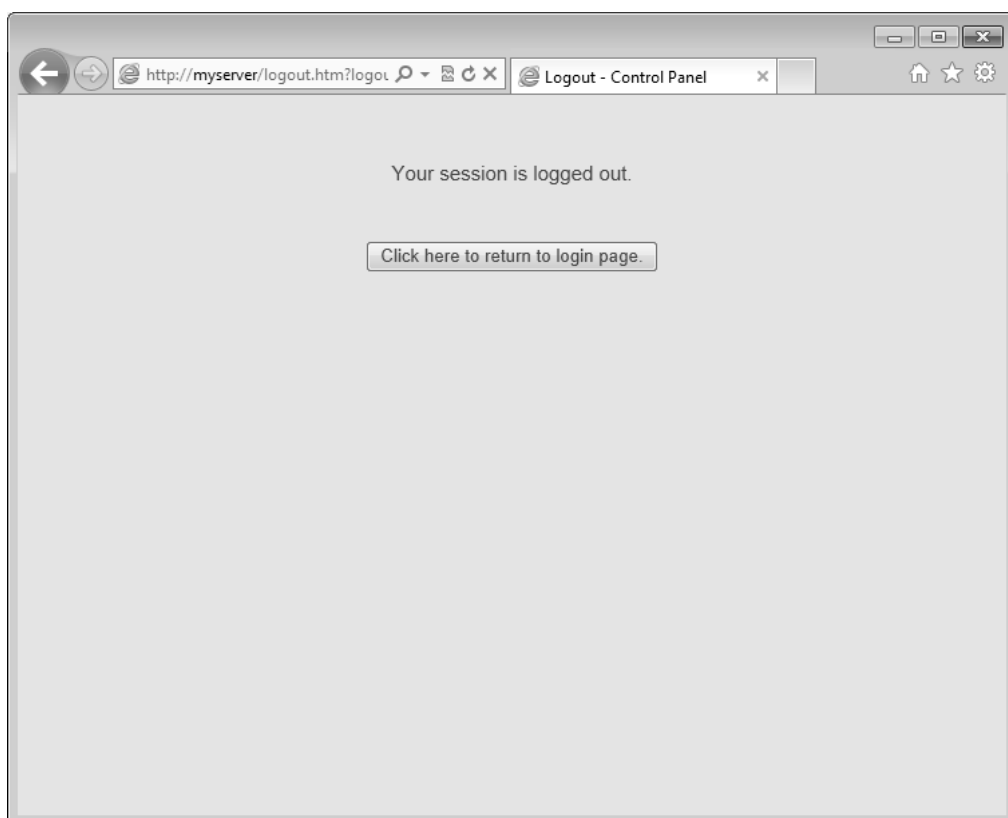


Figure 4.17: Screenshot of Logout Page

4.2.3 Dimmer

Light intensity is varied by changing the firing angle or increasing the delay before triggering the triac after a zero crossing is detected at the mains.

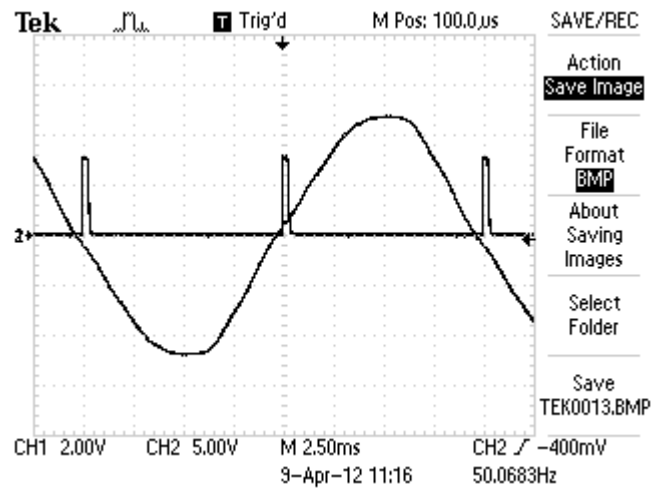


Figure 4.18: Zero Crossing Detector Output with Mains Sinusoid

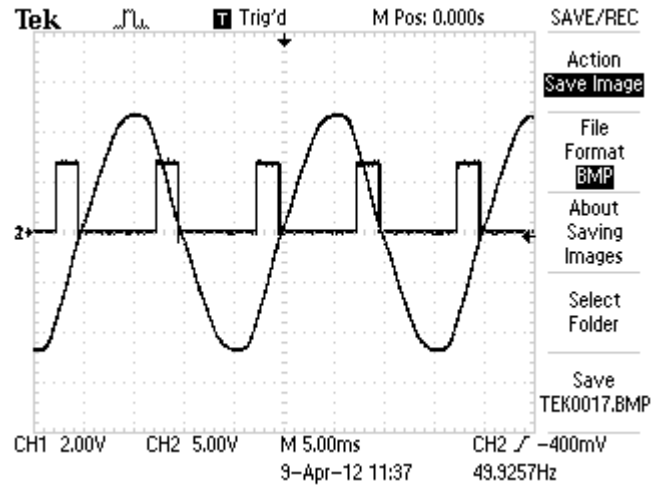


Figure 4.19: Phase Trigger Waveform for Minimum Light Intensity

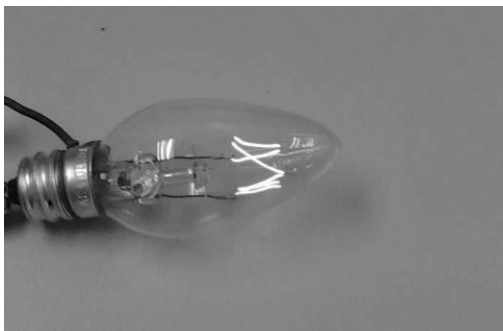


Figure 4.20: Light Bulb at Minimum Intensity

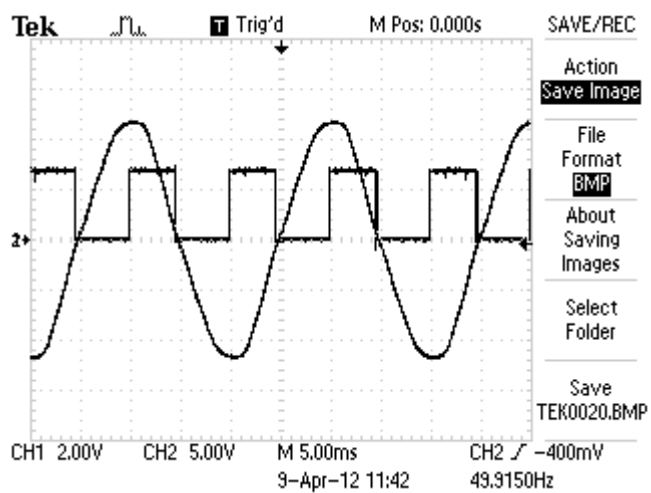


Figure 4.21: Phase Trigger Waveform for 50% Intensity



Figure 4.22: Light Bulb at 50% Intensity

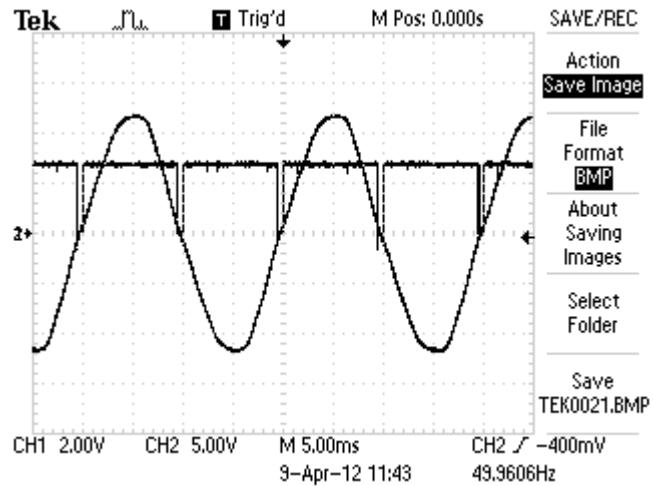


Figure 4.23: Phase Trigger for Maximum Intensity



Figure 4.24: Light Bulb at 100% Intensity

4.2.4 Power Consumption

Power measuring conditions of the system is based on measurement from output of a 5V regulated switch mode power supply unit, without considering power consumption by the power supply unit. Microcontroller clocked at 32MHz, instruction execution rate of 16MIPS. Zero crossing module is running at all time during the test.

Table 4.3: Power Consumption

Testing Condition	Current (A)	Power (W)
Without USB Flash Drive, Server Idle	0.16	0.80
With USB Flash Drive, Server Idle	0.21	1.05
With USB Flash Drive, Server Transmitting Data	0.22	1.10
With USB Flash Drive, One Dimmer Module Activated	0.23	1.15
With USB Flash Drive, One Relay Module Activated	0.24	1.20

4.3 Discussions

4.3.1 HTTP

Currently, Microchip's TCP/IP stack only supports two types of HTTP request methods. They are the GET method and POST method. There are limitations of GET request imposed by the TCP/IP stack and the microcontroller resources. HTTP GET method appends all data to the end of the request URL. The stack requires that the URL to be stored in the buffer in order to be further processed. There is limitation on available memory to be used as buffer for the URL. The default configuration of the server can accommodate GET request up to 98 characters in length. Therefore, another method shall be used if data to be transmitted to the web server is longer than the maximum buffer length.

HTTP POST method is another method that a web browser may transmit data to the web server. This method allows longer data length to be transferred to the server from the client. The maximum length is limited only by the architecture of the server implementation. The protocol does not specify any limit to the length of POST request.

However, in Microchip's TCP/IP stack, the *content-length* field of the HTTP header is parsed as 9 bytes characters with theoretical maximum length of 999,999,999 bytes, just under 1GB.

In the TCP/IP stack, GET requests and POST requests are handled by separate handlers, they are shown in Appendix A page 61 and page 66 respectively. For GET request, if the request length exceeds the buffer length, the TCP/IP stack will discard the request and response by transmitting an error message of "414 Request-URI Too Long" to the user.

For POST requests, the content is not put in the buffer immediately, but part of it is temporarily put onto the buffer for processing. Raw data is received from the TCP buffer. It is essential to keep track of how many bytes of data are received, how many bytes of data are processed, and how many bytes of data are yet to be processed manually. Therefore, the *content-length* field of each POST request is important for POST request handler to work properly. Losing track of the number of bytes remaining will cause data corruption of the next data received or server error of "500 Internal Server Error: Expected data not present" if desired variable is not received before all data have been received.

4.3.2 HTML

The web interface was developed and tested with modern browsers including Microsoft Windows Internet Explorer 9, Mozilla Firefox 11, Opera 12.00 Alpha, and Google Chrome 19. Interface was rendered slightly differently on different browsers, but their functionality remains intact. Users will most likely not notice any difference between the renderings from different browsers. The interface however is not optimized for mobile devices with smaller screen size, but the interface is fully functional as tested with Opera Mobile.

Webpage redirection performed using JavaScript works in all browsers without any complications except Windows Internet Explorer 9 (IE9). In IE9, the

rendering engine treats unclosed comment tag invalid, causing the page to display page content which was meant to be commented and failed to redirect. A fix was introduced to every web page that might be redirected using JavaScript by adding a pair of open and closing comment tag at the end of the HTML documents. This solution came from the idea that multi-line comment cannot be nested by another multi-line comment. Therefore the first open comment tag will match with the next closing tag ignoring nested open comment tag with a condition where there shall not be any other multi-line comment tags in the middle of the HTML document.

4.3.3 SPI

SPI communication is used between the microcontroller and the Ethernet controller. The Ethernet controller accepts SPI clock rate up to 20MHz. However the maximum SPI clock rate generated by PIC24F microcontroller is half of its maximum operation frequency of 16MHz, therefore the maximum SPI clock frequency for the system is 8MHz.

SPI is design for communication between integrated circuit at close distance. During prototype, the SPI connection between the microcontroller and the Ethernet controller is long (around 9 cm). The long wires captured noises from the environment causing the communication to be unreliable. Corrupted data were received at both ends. In an attempt to rectify the problem, a ground wires were twisted around the data lines to act as shielding, forming twisted pair's cable. The result is a more reliable communication, less data were corrupted. When corrupted data were received at the microcontroller, it will cause the microcontroller to reset. Therefore further reduction of the wire length was attempted. The cable was reduced to 4 cm and a reliable communication was achieved at 8MHz.

4.3.4 Authentication and Authorisation

There are at least three options available for authenticating user. The three options are basic access authentication, digest access authentication, and custom login form. Microchip's TCP/IP stack has built in support of basic access authentication.

Table 4.4: Comparison of Different Types of Authentication Methods

	Basic access authentication	Digest access authentication	Custom login form
Design	dependent on browser	dependent on browser	independent on browser
Security	Plain text with base64 encoding	Hashing of password with salt and dynamic variables	Custom
Method	Handled in HTTP request header	Handled in HTTP request header	GET or POST

The authentication method chosen for this project is custom login form. Unauthenticated users will be greeted by a login page containing login form. User will be required to type in their username and password in order to be authenticated by the server. The custom login method utilizes JavaScript to emulate digest access authentication. When a login page ("login.htm") is loaded at the client side, JavaScript will dynamically request for a "login.xml" XML file with "REALM" and "NONCE" values generated by the server. "NONCE" value will expire every 10 minute, when a new "login.xml" file is served or when a login attempt occurs, whichever comes first. The method used is by appending multiple parameters into one string with colon as separator, and hashing the string using MD5 algorithm. The hashing is performed at the client side with JavaScript. The resulting hash string will be sent back to the server, where the server will perform similar process internally to generate a hash string to compare with the received hash string. Upon matching of the hashes, an XML file is sent back to the client, where JavaScript will interpret the file content to decide on which page to redirect user to. If the hash string does not match, user will receive an alert popup and being redirected to the login page again,

else, the user will be redirected to the following page. Example of the login POST request string is:

“user=user&res=89A55A3387DDCF5BAEFDEF0652AA8BAA&cnon=pic24fj256gb106&nc=6543” where the “user” variable contains the username, “res” contains the MD5 hash of the salted password, while “cnon” and “nc” are variables required to calculate the password hash.

Due to the implementation of the TCP/IP stack, it is impossible to redirect user to another page using HTTP codes such as 302 or 301 redirection headers to redirect unauthenticated user to the login pages, or redirect authenticated users to the content page without major rewriting of the TCP/IP stack. Other method of redirection is the use of HTML Meta refresh or through the use of JavaScript redirection. The use of either HTML Meta refresh or JavaScript redirection is possible as this will only require manipulation of existing HTML files which is possible through a callback by the TCP/IP stack during transmission of the file.

However, in this project, JavaScript redirection is used instead of HTML Meta refresh. This is because the use of JavaScript is supposed to work as all the user interface pages designed will require JavaScript support from the browsers to operate correctly.

Although implementing redirection on client side is not secure as user is able to interrupt the redirection process, the server is further secured by performing checking on user authorization status before performing any requested actions that require a user to be authenticated.

User IP is logged temporarily when a user logged in successfully with a correct combination of username and password. Any user attempt to access the control interface from another device with a different IP will be presented the login screen. The system only allow one active session at any instance. For example, ‘User A’ logged in from machine with IP of 192.168.0.2 and later ‘User B’ login to the system from IP 192.168.0.3, ‘User A’ will be logged out upon successful login of ‘User B’. Other than that, a user will automatically be logged out if the session is inactive for 10 minutes.

4.3.5 Input Capture and Output Compare Module

Input capture module and Output Compare modules were used to design dimmer circuit. Preliminary design was to use an input capture module to capture an event generated by the zero crossing circuit. The input capture module will generate an interrupt on every occurrence of a rising edge at its mapped input pin. Output compare modules may operate at different modes for different purposes. It may operate as PWM signal generator, or a pulse generator. Output compare module is able to be synced or triggered by other modules such as timer, input capture, and ADC. The design will require the input capture to automatically trigger output compare module to start counting to its designated value to produce an output.

The delays before output compare produce an output is determined by the user input. For example, if the user selected the lowest light intensity from the control panel, the output compare module will be delayed longer before producing an output.

However, the input capture module failed to trigger the output compare's timer through hardware automatically, therefore, another design was implemented to trigger the output compare module manually by using interrupt service routine of the input capture module by manually setting the trigger flag of the output compare module.

4.3.6 Encryption

Hypertext Transfer Protocol Secure (HTTPS) may be used for secured transfer of information between the server and the client. However, HTTPS functionality requires the Data Encryption Libraries from Microchip, which is not bundled with the TCP/IP stack due to United States export regulations on encryption technologies.

Therefore an encryption scheme is required when transmitting password that user entered in the advanced setting page across the network through HTTP for

security reason. When choosing the correct encryption algorithm, the platform capability is one of the deciding factors. In this project, XXTEA block cipher is chosen, because the algorithm has a small memory footprint on both program memory and working memory, this is most suitable for embedded devices with small memory available. Other encryption algorithm requires higher program memory space. The XXTEA encryption uses a 128-bit key to encrypt the password. The key is pre-shared between the server and the user.

A chosen-plaintext attack as described by Elias Yarrkov in “Cryptanalysis of XXTEA” is not possible for this system. This is because the attack method requires that the system encrypt given plaintext with the secret key and returning the ciphertext to the attacker. In this system, the encryption is performed at the browser which the user is to enter the key manually, the system is unable to generate any ciphertext on its own without user entering the key.

It is also not possible to request the server to encrypt any plaintext given, this is due to the server is programmed to perform decryption only and the resulting output is not returned to user.

4.3.7 Event Logging

Data logging is the process of recording events to provide information that can be used for troubleshooting of errors and understanding the operations process.

In this project, the server is capable of logging the start-up event of the server. The server will write the event into a log file in the USB flash drive. The server will write the event title together with epoch timestamp into the file. File I/O operations is made possible by the MDD file system. In the case if the log file is not available on the flash drive, a new file will be created. If the log file is found, data will be appended into the existing log file. Code snippet of the event logging function is available in Appendix A page 77.

Besides that, the system will also send an update to Twitter. For Twitter update, the time stamp is sent as human readable time format unlike epoch timestamp logged in the server log file. This function is made possible by the HTTP client function of the TCP/IP stack. Code excerpt of the HTTP client function is shown in Appendix A page 78.

Due to the requirement of Twitter, OAuth user authentication is required, this authentication method is not supported by the TCP/IP stack. Therefore a proxy is required to authenticate the system, providing access to Twitter's REST API remotely without having to deal with OAuth at the microcontroller end.

In this project, GTAP, GAE Twitter API Proxy (a Twitter API proxy based on Google App Engine) is used as the proxy software. The proxy is deployed onto Google App Engine environment. GTAP is an open-source software written in Python programming language. The main development purpose of GTAP is to allow user to access Twitter's API remotely without directly accessing Twitter and to allow legacy software that do not support OAuth authorization to communicate with Twitter.

The proxy requires some configuration including setup of OAuth parameters such as 'consumer key', and 'consumer secret'. Source code modification is required for the proxy to better accommodate the TCP/IP stack capability. The proxy is modified to convert a GET request sent from the automation system to the proxy server to a POST request required to access Twitter's API. The proxy code together with modifications is shown in Appendix A page 81.

4.3.8 Dimmer Circuit

The initial design of the dimmer circuit is not as shown in Figure 4.6. In figure 4.6, a photo triac output opto-isolator (MOC3023) is being used. In the initial design, a transistor output opto-isolator (PC123) was used. The performance of the circuit is poor, where the circuit is unable to trigger the triac due to insufficient current supply.

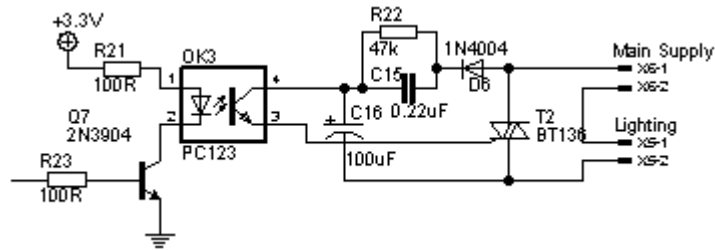


Figure 4.25: AC Dimmer Circuit with PC123 Opto-isolator

The circuit shown in Figure 4.25 does not work as predicted. When the phase angle delay is small, the time of keeping the triac on is longer and requires more energy. The energy stored in the 100 μ F capacitor may not be enough to supply to the triac gate and to charge up for the next cycle. Thus, the intensity of the light bulb connected to the circuit fluctuates. Attempt to rectify the problem by changing the capacitor value to a higher value does not improve the condition. Other attempt such as to reduce the resistor value from 47k Ω to lower values failed as the resistor failed to cope with the additional current and began to heat up.

Transistor can only switch current in one direction. The extra components were required because the output of PC123 is a transistor, they were designed to work as rectifier and current limiter to charge up the capacitor.

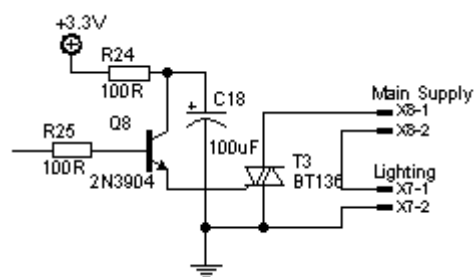


Figure 4.26: AC Dimmer Circuit with Transistor

Another design as shown in Figure 4.26 was considered before switching to use the circuit in Figure 4.5. In this circuit, instead of using opto-isolator to isolate the Mains and the DC, a transistor was used to switch on the triac with DC power. This

circuit is not used due to safety concern over the common ground between the mains and the DC supply. Touching the circuit may cause electric shock.

4.3.9 Relay Module

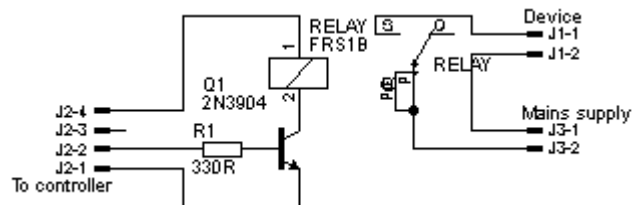


Figure 4.27: Relay Circuit with Transistor

Figure 4.27 shows the circuit designed to switch AC current by using a relay and a transistor. Relay is used to switch AC appliances through the microcontroller. A transistor is used as current amplifier to drive the relay coil. This is required as the microcontroller is unable to source enough current to drive the relay.

4.3.10 Optimizations

Optimization was performed to improve the overall system's performance. Amount of files required for the web server to access before able to serve a complete webpage is reduced to reduce page load time. This is vital step to reduce overall time to load a single page as the TCP/IP stack is capable of only serving one file at any moment. Multiple requests sent simultaneously will cause the requests to be queued and wait for the first request and response to be completed.

Other optimizations include reducing the HTML files size by removing unnecessary white spaces, and minify CSS and JavaScript. However, these

optimizations are not recommended as the resulting HTML files code will be difficult to be read and edited.

Other optimization of HTML documents includes server side compression such as gzip compression. This is however not possible as the compression is performed by the MPFS Generator for HTML documents to be stored in EEPROM only and is unavailable for MDD file system.

4.4 Design and Development Flow

The project is split into different parts and designed and tested in stages. The main part of the project is the implementation TCP/IP protocol on PIC24F microcontroller. Figure 4.28 shows the development flow.

After the development of the embedded web server, the project proceeds to the development of the automation system. Circuits for the web server and the automation system were designed and developed.

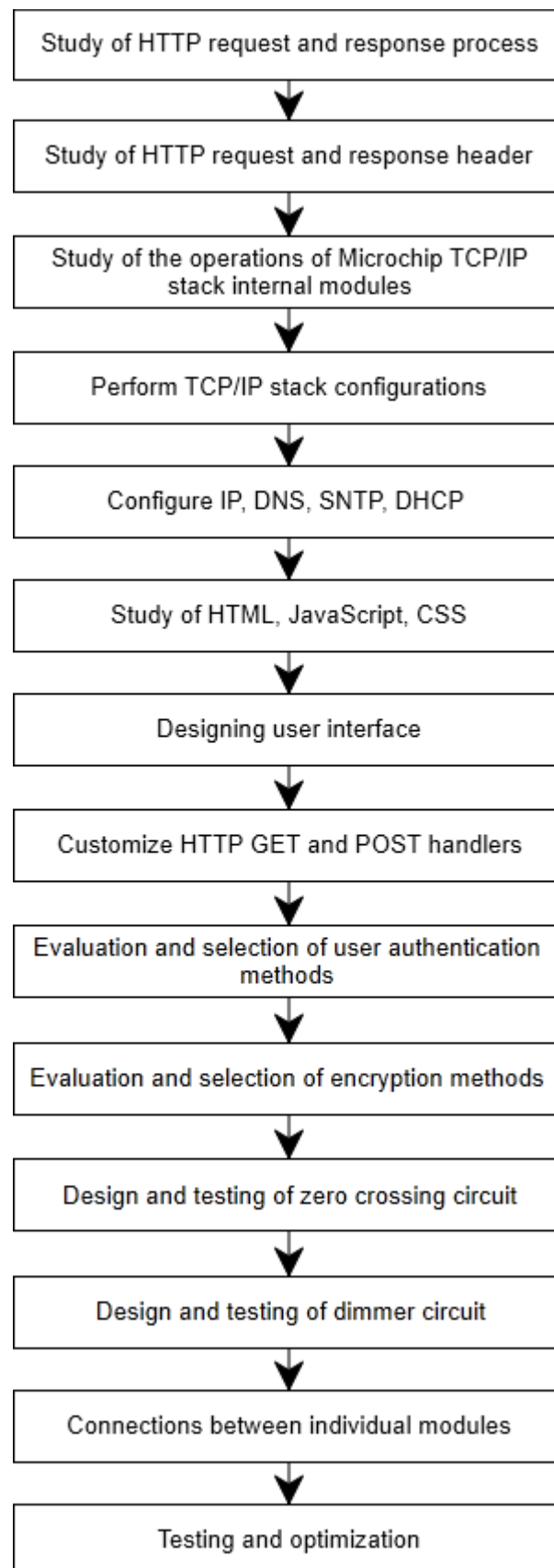


Figure 4.28: Design and Development Work Flow

4.5 Cost

One of the objectives of this project is to develop an affordable building automation system. By using low cost components together with open development tools for HTML development. As a comparison to proprietary software, many HTML development tools are open source or available as freeware, this has saved development cost, reducing total cost of end product. A simple text editing software such as Notepad++ is adequate for HTML development.

Table 4.5: Bill of Materials

No.	Parts Name	Quantity (pcs)	Unit Price (MYR)	Amount (MYR)
1	PIC24FJ256GB106 Microcontroller	1	22.80	22.80
2	ENC28J60 Ethernet Module	1	21.51	21.51
3	64/80 Pin TQFP to DIP Adapter	1	2.50	2.50
4	PCB 6 × 8 inch	1	5.10	5.10
5	Songle Relay 5VDC	2	2.00	4.00
6	BT136-600 TRIAC	1	1.15	1.15
7	1N4148 Diode	1	0.07	0.07
8	1N4004 Diode	4	0.20	0.80
9	PC123 Optocoupler	1	0.46	0.46
10	MOC3023 Optocoupler	1	2.02	2.02
11	P2N2222AG NPN Transistor	7	0.18	1.26
12	LED 3mm	1	0.20	0.20
13	Resistor 1/4W 270Ω	1	0.05	0.05
14	Resistor 1/4W 330Ω	7	0.05	0.35
15	Resistor 1/4W 1kΩ	2	0.10	0.20
16	Resistor 1/4W 10kΩ	1	0.05	0.05
17	Resistor 1/4W 22kΩ	1	0.05	0.05
18	Resistor 1/4W 100kΩ	2	0.05	0.10
19	Resistor 1/4W 120kΩ	1	0.05	0.05
20	Resistor 1/4W 330kΩ	2	0.05	0.10
21	Capacitor 220nF 275Vac	1	1.00	1.00

22	Capacitor 10 μ F 16V	2	0.30	0.60
23	Capacitor 0.22nF	1	0.30	0.30
24	Capacitor 22pF	2	0.30	0.60
25	Crystal 20.00MHz	1	2.00	2.00
26	Turn Pin Socket 40pins	2	2.00	4.00
27	Turn Pin Header 32pins	2	2.50	5.00
28	Molex Connector 2ways	5	0.45	2.25
29	Molex Connector 4ways	10	0.90	9.00
30	USB A Connector	1	1.50	1.50
31	Box Header 10pins	1	1.00	1.00
32	Header Socket 40 \times 2	1	2.50	2.50
33	USB Flash Drive	1	14.00	14.00
			Subtotal	106.57

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The goal and objectives of this project are achieved. An affordable web based control interface for building automation through embedded web server was developed. The total cost of this project is MYR106.57.

Existing building automation technology was studied on. Most existing automation system uses the X10 protocol to send and receive commands to connected nodes. In this project, the control is centralized and performed by the embedded web server. The embedded web server was built with a Microchip PIC24F microcontroller for the control system. It was built on top of the latest version of free licensed Microchip TCP/IP stack as of 1st January 2012.

HTML user control interface was developed for power devices controls. Web pages were designed with HTML, JavaScript and CSS style sheets. An energy efficient building automation controller was developed. Power consumed by the system when idling is 1.05W. Existing local area network infrastructure was utilized for the control. Existing network does not require major modification to incorporate the automation system. Remote power switching from any devices with a web browser and Internet access was made possible by the embedded web server.

5.1.1 Personal Breakthrough

Networking knowledge was gained in the process of completing this project. Computer networking was not introduced in curriculum activities. TCP/IP configurations, router port forwarding configuration were performed. Soldering skill was improved as there is little to no experience in soldering surface mount components throughout the course. In this project, the microcontroller used is a surface mount component. It comes in TQFP 64 package, with pitch of 0.3mm.

Besides that, I was given opportunity to design board layout using “EAGLE” software, creating components library and developing PCB by myself. Circuits were designed based on each component’s specification and their limitations.

5.2 Recommendations

There are some additional features not included in this project. Additional features may be added in the future to further enhance the system.

5.2.1 Real-time Clock

In this project, NTP server is used to obtain time for the server’s operations. If the internet connection is broken, the server will not be able to obtain time from the NTP server, server time will be incorrect. This may cause some functionality to be performed incorrectly.

Therefore, a real-time clock with secondary crystal oscillator is recommended to be added as an additional feature to enhance the system time tracking capability. A small battery can be used to keep the real-time clock running even after the system is shut down.

5.2.2 HTTP Redirection

JavaScript is used in web pages to redirect user to a correct page. For example, a user landed on a page which requires the user to be authenticated, the user will be redirected to the login page if the user is not already been authenticated.

HTTP redirection can be performed using HTTP Meta refresh. It is similar to JavaScript redirection as it is client side script and may be blocked by the user through browsers setting. However, it is better to have a backup solution in case JavaScript failed to redirect user to the correct page.

5.2.3 TCP/IP Stack Modifications

The implementation of TCP/IP protocol by Microchip is rigid. Modifications requires rewrite of certain modules. There are two main reasons to modify the TCP/IP stack. First is to perform HTTP redirection using HTTP header, either 302 or 301 redirections when user are required to be redirected to the login page. By using HTTP header redirection, it is more secure as user intervention can be prevented. No content is required to be sent to the client side, saving bandwidth and page redirection time. HTTP header redirection is only available if HTTP basic access authentication is used in the TCP/IP stack.

Next, the implementation of parsing HTML documents by the TCP/IP stack may be modified to reduce an additional step required when deploying edited HTML documents with dynamic variables contained in it. Whenever a HTML document is modified, the location of variables in the documents might have changed and will require the use of a tool provided by Microchip to re-index the document. If this step is not performed, the server will fail to replace the variable name with the variable value.

5.2.4 Microchip dsPIC Microcontroller

Currently used PIC24F has maximum system clock frequency of 16MHz and maximum instruction execution rate of 16MIPS. The performance of the web server can be improved by using dsPIC which has higher maximum instruction execution rate. With dsPIC, the SPI communication between the microcontroller and the Ethernet controller can be increased to 10MHz increasing maximum data throughput of the system.

5.2.5 Event logging

Currently the server will log server startup events together with the startup timestamp into the log file. Other events are not logged. Future improvement may include logging of other events such as user login attempt, user IP, user actions, and server errors. The log file is stored in the USB flash drive together with the web pages. Further improvement may store logs into cloud server over the internet.

5.2.6 Social Network Integration

The web server may act as a client as well, the server may post logs or status to social network such as Twitter. User with social network apps running on their portable devices all the time will receive real-time event update on their devices without having to login to the server interface with a browser. Currently, the server is only capable of logging the server startup events, future improvement may include update of other events to the social network.

REFERENCES

- Acson (Producer). (2011). Network Control NIM. *Intelligent Control Series*. Retrieved from http://acson.com.my/sites/default/files/Intelligent_Control_Series_2.pdf
- Agency, I. E. (Producer). (2010). Total primary energy supply. *IEA Energy Statistics*. Retrieved from http://www.iea.org/stats/pdf_graphs/MYTPES.pdf
- Alkar, A. Z., Roach, J., & Baysal, D. (2010). IP based home automation system. *Consumer Electronics, IEEE Transactions on*, 56(4), 2201-2207.
- America, P. N. Priva - Horticulture, from <http://www.priva.ca/en/solutions-products/horticulture/>
- Anthony. (2009a, 13/06). UPB Retrieved 01/08, 2011, from <http://smart-home-automation-guide.com/signal-relay/upb/>
- Anthony. (2009b, 13/06). UPB Pros and Cons Retrieved 23/07, 2011, from <http://smart-home-automation-guide.com/signal-relay/upb/upb-pros-and-cons/>
- Associates, P. (2006). Media Servers in the Digital Home. *White Paper*.
- Barrientos, M. Malaysia - electric power consumption. Available from index mundi Retrieved 05/08, from International Energy Agency <http://www.indexmundi.com/facts/malaysia/electric-power-consumption>
- Bösemann, W. (1996). The Optical Tube Measurement System OLM Photogrammetric Methods used for Industrial Automation and Process Control. *International Archives of Photogrammetry and Remote Sensing*, 31, 55-58.
- Controls, A. Case Study: Building Automation.
- Corcoran, P. M., & Desbonnet, J. (1997). Browser-style interfaces to a home automation network. *Consumer Electronics, IEEE Transactions on*, 43(4), 1063-1069.
- Digital Loggers, I. (09/06/2010). Web Power Switch Retrieved 16/07, 2011, from <http://www.digital-loggers.com/lpc.html>
- Elias Yarrkov. (2010, 04/05). Cryptanalysis of XXTEA Retrieved 25/04, 2012, from <http://eprint.iacr.org/2010/254.pdf>

- Ergen, S. C. (2004). ZigBee/IEEE 802.15.4 Summary.
- Essig, J. (2011, 03/08/2011). Crestron Controls AV in Entourage Star's New Home Theater Retrieved 01/08, 2011, from http://www.crestron.com/about/press_room/press_releases/show_release.asp?press_release_id=1657
- Est., A. T. S. (2009). Pre Qualification Document Retrieved 01-08, 2011, from <http://www.afragroup.com/FinalPQD.pdf>
- Frankfurt, M. (2011). Welcome to Light+Building 2012 Retrieved 15/07, 2011, from <http://light-building.messefrankfurt.com/frankfurt/en/besucher/willkommen.html>
- Han, D., & Hwang, D. H. (2005). A novel stereo matching method for wide disparity range detection. *Image Analysis and Recognition*, 643-650.
- Harmo, P., Taipalus, T., Knuuttila, J., Vallet, J., & Halme, A. (2005). *Needs and solutions-home automation and service robots for the elderly and disabled*.
- Honeywell International, I. (2008). Alerton: Products: HVAC: BACtalk Retrieved 01-08, 2011, from <http://www.alerton.com/s/Products/HVAC/BACtalk>
- INSTEON. (2005, 15/08). INSTEON: The Details, from <http://www.insteon.net/pdf/insteonthedetails.pdf>
- . Intelligent Building Automation conference and exhibition. Retrieved 16/07, 2011, from <http://www.eepublishers.co.za/article/intelligent-building-automation-conference-and-exhibition.html>
- Jay. (2010). X10 Device Support. Retrieved 09/08, from Perceptive Automation, LLC. http://www.perceptiveautomation.com/wiki/doku.php?id=x10_devices
- Ju, H. T., Choi, M. J., & Hong, J. W. (2000). An efficient and lightweight embedded Web server for Web-based network element management. *Int. J. Network Mgmt*, 10, 261-275.
- Microsoft. (2009). Play To - Windows 7 features - Microsoft Windows. *Windows 7 features* Retrieved 09/08, 2011, from <http://windows.microsoft.com/en-MY/windows7/products/features/play-to>
- Nunes, R. J. C. (2004). *A Web-based approach to the specification and programming of home automation systems*.
- Ploeg, J. ZIGBEE Specification Retrieved 08/08, 2011, from http://www.specifications.nl/zigbee/zigbee_UK.php
- . RFC1122. (1989) *Requirement for Internet Hosts -- Communication Layers*: Internet Engineering Task Force.

Ryan, J. (1989). Home automation. *Electronics & communication engineering journal*, 1(4), 185-192.

Starner, T., Auxier, J., Ashbrook, D., & Gandy, M. (2000). *The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring*.

Systems, P. C. (2007). UPB Technology Description version 1.4: Powerline Control Systems.

W3Schools. TCP/IP Tutorial, from <http://www.w3schools.com/tcpip/default.asp>

X10, E. Transmission theory of X-10 signals, from <http://www.eurox10.com/Content/X10SignalTheory.htm>

APPENDICES

APPENDIX A: Programme Listing

HTTP GET Request Handler:

```
// HTTP GET method request handler function
HTTP_IO_RESULT HTTPExecuteGet(void)
{
    char *ptr,*ptr2;
    BYTE filename[15];
    FSFILE *pointer;

    temp=TCPGetRemoteInfo(sktHTTP);    //Get client IP

    t3 = 0;

    // Load the file name
    BYTE I,j,cntr1=0,cntr2=0,filext=0;
    do
    {
        if(curHTTP.file->name[cntr2] != 0x20) //spacebar
        {
            filename[cntr1]=curHTTP.file->name[cntr2];
            cntr1++;
        }
        else if(filext!=TRUE)
        {
            filename[cntr1]=0x2E; //dot
            cntr1++;
            filext=TRUE;
        }
    }
}
```

```

        }
        cntr2++;
    }while(cntr2 !=FILE_NAME_SIZE);

    for(i=0;i<20;i++) //Convert filename to lower case
        if((filename[i] >= (BYTE)'A') && (filename[i] <=
(BYTE)'Z'))
            filename[i] += 'a' - 'A';

if(!memcmppgm2ram(filename, "logout.htm", 10))
{
    isauth=0;    //de-authenticate user
    return HTTP_IO_DONE;
}
goodRequest=0;    //reset flag to 0

if(isauth)        //check if user is authenticated, if IP is same
if(clientIP.Val == (*temp).remote.IPAddr.Val)
{
    // If it is the update.xml file
    if(!memcmppgm2ram(filename, "update.xml", 10))
    {
        if(ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"id"))
        {
            switch(*ptr)
            {
                case '1': i=1;
                break;
                case '2': i=2;
                break;
                case '3': i=3;
                break;
                case '4': i=4;
                break;
                case '5': i=5;
                break;
                default:        //if no id, do nothing
                return HTTP_IO_DONE;
            }
        }
        //if field "name" exists

```

```

if(ptr2=HTTPGetROMArg(curHTTP.data, (ROM BYTE
*)"name"))
{
    //copy name to internal variable
    for(j=0;j<21;j++)
        devices[i-1].dname[j]=*(ptr2++);
    goodRequest=1;    //set flag
    //write new name to devices config file
    pointer=Fsfopen("DEVSTAT.BIN","r+");
    Fsfseek(pointer, (i-1)*22,SEEK_SET);
    Fsfwrite(&devices[i-1],1,22,pointer);
    Fsfclose(pointer);
}
else if(ptr2=HTTPGetROMArg(curHTTP.data, (ROM BYTE
*)"cat"))
//if "cat" exists
{
    //check if "cat" value is changed
    if((*ptr2)-0x30)!=devices[i-1].dtype)
    {
        if(devices[i-1].dstat!=0)
        {
            //if current device is on, turn off
            switch(devices[i-1].dtype)
            {
                case 1: toggle(I,0);
                    break;
                case 2: fan(I,0);
                    break;
                case 3: light(I,0);
                    break;
                case 4: pulse(I,0);
                    break;
                default: unsetpin(i);
                    break;
            }
        }
        //set new category to the device
        devices[i-1].dtype=(*ptr2)-0x30;
        if(devices[i-1].dtype == 2 ||
            devices[i-1].dtype == 3)
            setpin(i); //map output compare
        else
            unsetpin(i); //unmap pin
    }
}

```

```

        //clear timer of that device
        for(j=0;j<5;j++)
            devtimer[i-1].timer[j].isset=0;
        //write to file
        pointer=Fsfopen("DEVSTAT.BIN","r+");
        Fsfseek(pointer, (i-1)*22,SEEK_SET);
        Fsfwrite(&devices[i-1],1,22,pointer);
        Fsfclose(pointer);
        pointer=Fsfopen("TIMER.BIN","r+");
        Fsfseek(pointer, (i-1)*15,SEEK_SET);
        Fsfwrite(&devtimer[i-1],1,15,pointer);
        Fsfclose(pointer);
    }
    goodRequest=1;    //set flag

}
else if(ptr2=HTTPGetROMArg(curHTTP.data, (ROM BYTE
*)"toggle"))
    // on/off device?
    {
        toggle(I,atoi(ptr2));
        goodRequest=1;
    }
else if(ptr2=HTTPGetROMArg(curHTTP.data, (ROM BYTE
*)"speed"))
    //change fan speed?
    {
        if(devices[i-1].dtype==2)
            fan(I,atoi(ptr2));
        goodRequest=1;
    }
else if(ptr2=HTTPGetROMArg(curHTTP.data, (ROM BYTE
*)"dim"))
    //change light intensity?
    {
        if(devices[i-1].dtype==3)
            light(I,atoi(ptr2));
        goodRequest=1;
    }
else if(ptr2=HTTPGetROMArg(curHTTP.data, (ROM BYTE
*)"pulse"))
    //send a pulse?
    {
        //calibrated to 100ms pulse
        if(count==0 || count == 320)
            {

```

```

        if(devices[i-1].dtype==4)
        {
            if(i==1)
                DEVICE1_IO^=1;
            else if(i==2)
                DEVICE2_IO^=1;
            else if(i==3)
                DEVICE3_IO^=1;
            else if(i==4)
                DEVICE4_IO^=1;
            else if(i==5)
                DEVICE5_IO^=1;
        }
    }
    if(count<320) //calibrated to 100ms
    {
        count++;
        return HTTP_IO_WAITING;
    }
    else
    {
        devices[i-1].dstat^=1;
        count=0;
        goodRequest=1;
    }
}
}
}
//if GET request for "timer.xml", clear all timer settings
else if(!memcmpm2ram(filename, "timer.xml", 9))
{
    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            devtimer[i].timer[j].isset=0;
    goodRequest=1;
}
}
return HTTP_IO_DONE;
}

```

HTTP POST Request Handler:

```

// HTTP POST method request handler function
HTTP_IO_RESULT HTTPExecutePost(void)
{
    BYTE filename[15];
    temp=TCPGetRemoteInfo(sktHTTP);
    t3 = 0;    //reset timeout counter

    // Load the file name
    BYTE i,cntr1=0,cntr2=0,filext=0;
    do{
        if(curHTTP.file->name[cntr2] != 0x20)
        {
            filename[cntr1]=curHTTP.file->name[cntr2];
            cntr1++;
        }
        else if(filext!=TRUE)
        {
            filename[cntr1]=0x2E; //File extension starts
after this
            cntr1++;
            filext=TRUE;
        }
        cntr2++;
    }while(cntr2 !=FILE_NAME_SIZE);

    for(i=0;i<20;i++) //Convert file name to lower case
        if((filename[i] >= (BYTE)'A') && (filename[i] <=
(BYTE)'Z'))
            filename[i] += 'a' - 'A';

    if(isauth) //check user authenticated, IP remain same?
    if(clientIP.Val == (*temp).remote.IPAddr.Val)
    {
        //if "timer.xml" process with timer processor function
        if(!memcmpm2ram(filename, "timer.xml",9))
            return processTimerForm();
        //if "setting.xml" process with advanced setting function
        if(!memcmpm2ram(filename, "setting.xml",11))
            return processAdvancedForm();
    }
}

```



```
}  
  
//if user try to login, process login form  
if(!memcmpm2ram(filename, "login2.xml", 10))  
    return processLoginForm();  
  
return HTTP_IO_DONE;  
}
```

Device Controls Code:

```

void toggle(BYTE i, BYTE onoff)      //switch device on or off
{
    if(devices[i-1].dtype==1)
    {
        if(i==1)
        {
            OC1CON1bits.OCM = 0;      //turn off OC1 module
            devices[i-1].dstat=DEVICE1_IO=onoff;
        }
        else if(i==2)
        {
            OC2CON1bits.OCM = 0;      //turn off OC2 module
            devices[i-1].dstat=DEVICE2_IO=onoff;
        }
        else if(i==3)
        {
            OC3CON1bits.OCM = 0;      //turn off OC3 module
            devices[i-1].dstat=DEVICE3_IO=onoff;
        }
        else if(i==4)
        {
            OC4CON1bits.OCM = 0;      //turn off OC4 module
            devices[i-1].dstat=DEVICE4_IO=onoff;
        }
        else if(i==5)
        {
            OC5CON1bits.OCM = 0;      //turn off OC5 module
            devices[i-1].dstat=DEVICE5_IO=onoff;
        }
    }
}

void fan(BYTE i, BYTE speed)         //control fan speed
{
    if(devices[i-1].dtype==2)
    {
        devices[i-1].dstat = speed;   //update status register

        if(i==1)

```

```

{
    if(speed)
    {
        if(speed ==5)
            OC1R = 10;//OC1 first comparison point
        else
            OC1R = 18000-(speed*3000);
        OC1RS = 19000; //OC1 second comparison point
        OC1CON1 |= 0x4;
        //connect to OCx pin (1 cycle)
        asm volatile("BCLR OC1CON2,#5");
    }
    else
    {
        OC1CON1 &= 0xFFFF8;        //disable OC module
        // tristate OC pin (1 cycle)
        asm volatile("BSET OC1CON2,#5");
    }
    IFS0bits.OC1IF = 0;
    IEC0bits.OC1IE = 1;
}
else if(i==2)
{
    if(speed)
    {
        if(speed ==5)
            OC2R = 10;
        else
            OC2R = 18000-(speed*3000);
        OC2RS = 19000;
        OC2CON1 |= 0x4;
        //connect to OCx pin
        asm volatile("BCLR OC2CON2,#5");
    }
    else
    {
        OC2CON1 &= 0xFFFF8;        //disable OC module
        //tristate OC pin
        asm volatile("BSET OC2CON2,#5");
    }
    IFS0bits.OC2IF = 0;
}

```

```

        IEC0bits.OC2IE = 1;
    }
else if(i==3)
{
    if(speed)
    {
        if(speed ==5)
            OC3R = 10;
        else
            OC3R = 18000-(speed*3000);
        OC3RS = 19000;
        OC3CON1 |= 0x4;
        //connect to OCx pin
        asm volatile("BCLR OC3CON2,#5");
    }
else
{
    OC3CON1 &= 0xFFFF8;        //disable OC module
    //tristate OC pin
    asm volatile("BSET OC3CON2,#5");
}
IFS1bits.OC3IF = 0;
IEC1bits.OC3IE = 1;
}
else if(i==4)
{
    if(speed)
    {
        if(speed ==5)
            OC4R = 10;
        else
            OC4R = 18000-(speed*3000);
        OC4RS = 19000;
        OC4CON1 |= 0x4;
        //connect to OCx pin
        asm volatile("BCLR OC4CON2,#5");
    }
else
{
    OC4CON1 &= 0xFFFF8;        //disable OC module

```

```

        //tristate OC pin
        asm volatile("BSET OC4CON2,#5");
    }
    IFS1bits.OC4IF = 0;    //clear interrupt flag
    IEC1bits.OC4IE = 1;    //enable interrupt
}
else if(i==5)
{
    if(speed)
    {
        if(speed ==5)
            OC5R = 10;
        else
            OC5R = 18000-(speed*3000);
        OC5RS = 19000;
        OC5CON1 |= 0x4;
        //connect to OCx pin
        asm volatile("BCLR OC5CON2,#5");
    }
    else
    {
        OC5CON1 &= 0xFFFF8;    //disable OC module
        //tristate OC pin
        asm volatile("BSET OC5CON2,#5");
    }
    IFS2bits.OC5IF = 0;
    IEC2bits.OC5IE = 1;
}
}
}
}
}
}
}

void light(BYTE i, BYTE dim)
{
    if(devices[i-1].dtype==3)
    {
        devices[i-1].dstat=dim;

        if(i==1)
        {
            if(dim)

```

```

{
    if(dim ==10)
        OC1R = 10;
    else
        OC1R = 18000-(dim*1700);

    OC1RS = 19000;
    OC1CON1 |= 0x4;
    //connect to OCx pin
    asm volatile("BCLR OC1CON2,#5");
}
else
{
    OC1CON1 &= 0xFFFF8;        //disable OC module
    //tristate OC pin
    asm volatile("BSET OC1CON2,#5");
}
IFS0bits.OC1IF = 0;
IEC0bits.OC1IE = 1;
}
else if(i==2)
{
    if(dim)
    {
        if(dim ==10)
            OC2R = 10;
        else
            OC2R = 18000-(dim*1700);

        OC2RS = 19000;
        OC2CON1 |= 0x4;
        //connect to OCx pin
        asm volatile("BCLR OC2CON2,#5");
    }
    else
    {
        OC2CON1 &= 0xFFFF8;        //disable OC module
        //tristate OC pin
        asm volatile("BSET OC2CON2,#5");
    }
    IFS0bits.OC2IF = 0;
}

```

```
        IEC0bits.OC2IE = 1;
    }
    else if(i==3)
    {
        if(dim)
        {
            if(dim ==10)
                OC3R = 10;
            else
                OC3R = 18000-(dim*1700);

            OC3RS = 19000;
            OC3CON1 |= 0x4;
            //connect to OCx pin
            asm volatile("BCLR OC3CON2,#5");
        }
        else
        {
            OC3CON1 &= 0xFFFF8;        //disable OC module
            //tristate OC pin
            asm volatile("BSET OC3CON2,#5");
        }
        IFS1bits.OC3IF = 0;
        IEC1bits.OC3IE = 1;
    }
    else if(i==4)
    {
        if(dim)
        {
            if(dim ==10)
                OC4R = 10;
            else
                OC4R = 18000-(dim*1700);

            OC4RS = 19000;
            OC4CON1 |= 0x4;
            //connect to OCx pin
            asm volatile("BCLR OC4CON2,#5");
        }
        else
        {
```

```

        OC4CON1 &= 0xFFFF8;        //disable OC module
        //tristate OC pin
        asm volatile("BSET OC4CON2,#5");
    }
    IFS1bits.OC4IF = 0;
    IEC1bits.OC4IE = 1;
}
else if(i==5)
{
    if(dim)
    {
        if(dim ==10)
            OC5R = 10;
        else
            OC5R = 18000-(dim*1700);

        OC5RS = 19000;
        OC5CON1 |= 0x4;
        //connect to OCx pin
        asm volatile("BCLR OC5CON2,#5");
    }
    else
    {
        OC5CON1 &= 0xFFFF8;        //disable OC module
        //tristate OC pin
        asm volatile("BSET OC5CON2,#5");
    }
    IFS2bits.OC5IF = 0;
    IEC2bits.OC5IE = 1;
}
}

void pulse(BYTE i, BYTE p) //send pulse for pulse triggered device
{
    if(devices[i-1].dtype==4)
    {
        if(i==1){
            OC1CON1bits.OCM = 0;
            devices[i-1].dstat^=1;
            DEVICE1_IO=1;
        }
    }
}

```



```
        DelayMs(82);          //100ms from logic analyzer
        DEVICE1_IO=0;
    }
else if(i==2){
    OC2CON1bits.OCM = 0;
    devices[i-1].dstat^=1;
    DEVICE2_IO=1;
    DelayMs(82);
    DEVICE2_IO=0;
}
else if(i==3){
    OC3CON1bits.OCM = 0;
    devices[i-1].dstat^=1;
    DEVICE3_IO=1;
    DelayMs(82);
    DEVICE3_IO=0;
}
else if(i==4){
    OC4CON1bits.OCM = 0;
    devices[i-1].dstat^=1;
    DEVICE4_IO=1;
    DelayMs(82);
    DEVICE4_IO=0;
}
else if(i==5){
    OC5CON1bits.OCM = 0;
    devices[i-1].dstat^=1;
    DEVICE5_IO=1;
    DelayMs(82);
    DEVICE5_IO=0;
}
}
}
```

USB Targeted Peripheral List Configuration:

```

USB_TPL usbTPL[] =
{
    { INIT_CL_SC_P( 8ul, 6ul, 0x50ul ), 0, 0, {TPL_CLASS_DRV} }
};

```

TCP/IP Server Configurations:

```

#define MY_DEFAULT_HOST_NAME          "MYSERVER" //NetBIOS host name

#define MY_DEFAULT_MAC_BYTE1          (0x00) //Server MAC address
#define MY_DEFAULT_MAC_BYTE2          (0x04)
#define MY_DEFAULT_MAC_BYTE3          (0xA3)
#define MY_DEFAULT_MAC_BYTE4          (0x00)
#define MY_DEFAULT_MAC_BYTE5          (0x10)
#define MY_DEFAULT_MAC_BYTE6          (0x05)

#define MY_DEFAULT_IP_ADDR_BYTE1      (192ul) //Preconfigure
#define MY_DEFAULT_IP_ADDR_BYTE2      (168ul) //Server
#define MY_DEFAULT_IP_ADDR_BYTE3      (2ul) //IP address
#define MY_DEFAULT_IP_ADDR_BYTE4      (20ul)

#define MY_DEFAULT_MASK_BYTE1          (255ul) //Preconfigure
#define MY_DEFAULT_MASK_BYTE2          (255ul) //Subnet
#define MY_DEFAULT_MASK_BYTE3          (255ul) //Mask
#define MY_DEFAULT_MASK_BYTE4          (0ul)

#define MY_DEFAULT_GATE_BYTE1          (192ul) //Preconfigure
#define MY_DEFAULT_GATE_BYTE2          (168ul) //default gateway
#define MY_DEFAULT_GATE_BYTE3          (2ul)
#define MY_DEFAULT_GATE_BYTE4          (1ul)

```

```

#define MY_DEFAULT_PRIMARY_DNS_BYTE1      (192ul) //Preconfigure
#define MY_DEFAULT_PRIMARY_DNS_BYTE2      (168ul) //Primary DNS
#define MY_DEFAULT_PRIMARY_DNS_BYTE3      (2ul)   //server
#define MY_DEFAULT_PRIMARY_DNS_BYTE4      (1ul)

#define MY_DEFAULT_SECONDARY_DNS_BYTE1     (8ul)   //Preconfigure
#define MY_DEFAULT_SECONDARY_DNS_BYTE2     (8ul)   //Secondary
#define MY_DEFAULT_SECONDARY_DNS_BYTE3     (8ul)   //DNS server
#define MY_DEFAULT_SECONDARY_DNS_BYTE4     (8ul)

```

Event Logging Function

```

if(firststrun == TRUE)
{
    time=SNTPGetUTCSeconds();
    if(time>600)
    {
        pointer = FSfopen ("SERVLOG.LOG","r");
        if(pointer==NULL)
        {
            pointer = FSfopen ("SERVLOG.LOG","w");
            FSfwrite("Startup ",1,8,pointer);
            FSfclose(pointer);
        }
        else
        {
            FSfclose(pointer);
            pointer = FSfopen ("SERVLOG.LOG","a");
            FSfwrite("Startup ",1,8,pointer);
            FSfclose(pointer);
        }

        ultoa(time,timebuf);
        pointer = FSfopen ("SERVLOG.LOG","a");
        FSfwrite(timebuf,1,strlen(timebuf),pointer);
        FSfwrite("\r\n",1,2,pointer);
        FSfclose(pointer);
        firststrun = FALSE;
    }
}

```

```

        if(tweetit==TRUE)
            GenericTCPClient();           //send tweet
    }
}

if(tweetit==TRUE)
    GenericTCPClient();           //ensure tweet is sent

```

HTTP Client Function

```

// Defines the Twitter API proxy server name
static BYTE ServerName[] = "<ApplicationName>.appspot.com";

// Defines the port to be accessed for this application
static WORD ServerPort = 80;

// Defines the URL to be requested
static ROM BYTE RemoteURL[] =
"/api/1/statuses/update.xml?status=Server%20startup%20at%20";
static ROM BYTE RemoteMethod[] = "&met=1"; //required by API proxy

switch(GenericTCPClientState)
{
    case SM_HOME:
        // Connect a socket to the remote TCP server
        MySocket = TCPOpen((DWORD)&ServerName[0],
TCP_OPEN_RAM_HOST, ServerPort, TCP_PURPOSE_GENERIC_TCP_CLIENT);

        // Abort operation if no TCP socket of type
TCP_PURPOSE_GENERIC_TCP_CLIENT is available
        if(MySocket == INVALID_SOCKET)
            break;

        GenericTCPClientState++;
        Timer = TickGet();
        break;

```

```

    case SM_SOCKET_OBTAINED:
        // Wait for the remote server to accept our connection
request
        if(!TCPIsConnected(MySocket))
        {
            // Time out if too much time is spent in this
state
            if(TickGet()-Timer > 5*TICK_SECOND)
            {
                // Close the socket so it can be used by
other modules
                TCPDisconnect(MySocket);
                MySocket = INVALID_SOCKET;
                GenericTCPClientState--;
            }
            break;
        }
        Timer = TickGet();

        // Make certain the socket can be written to
        if(TCPIsPutReady(MySocket) < 400u)
            break;

// Place the application protocol data into the transmit buffer.
        TCPPutROMString(MySocket, (ROM BYTE*)"GET ");
        TCPPutROMString(MySocket, RemoteURL);
        TCPPutString(MySocket, hr);
        TCPPutROMString(MySocket, (ROM BYTE*)"");
        TCPPutString(MySocket, min);
        TCPPutROMString(MySocket, RemoteMethod);
        TCPPutROMString(MySocket, (ROM BYTE*)" HTTP/1.1\r\nHost:
");
        TCPPutString(MySocket, ServerName);
        TCPPutROMString(MySocket, (ROM BYTE*)"\r\nAuthorization:
Basic dXNlcjpwYXNzd29yZA==");
        TCPPutROMString(MySocket, (ROM BYTE*)"\r\nConnection:
close\r\n\r\n");
        // Send the packet
        TCPFlush(MySocket);
        GenericTCPClientState++;
        tweetit=FALSE;

```

```

        break;

    case SM_PROCESS_RESPONSE:
        // Check to see if the remote node has disconnected from
        us or sent us any application data
        // If application data is available, write it to the
        UART

        if(!TCPIsConnected(MySocket))
        {
            GenericTCPClientState = SM_DISCONNECT;
            // Do not break; We might still have data in the
            TCP RX FIFO waiting for us
        }

        // Get count of RX bytes waiting
        w = TCPIsGetReady(MySocket);

        // Obtain and print the server reply
        i = sizeof(vBuffer)-1;
        vBuffer[i] = '\0';
        while(w)
        {
            if(w < i)
            {
                i = w;
                vBuffer[i] = '\0';
            }
            w -= TCPGetArray(MySocket, vBuffer, i);

            if(GenericTCPClientState == SM_PROCESS_RESPONSE)
                break;
        }

        break;

    case SM_DISCONNECT:
        // Close the socket so it can be used by other modules
        TCPDisconnect(MySocket);
        MySocket = INVALID_SOCKET;
        GenericTCPClientState = SM_DONE;
        break;

```

```

    case SM_DONE:
        GenericTCPClientState = SM_HOME;
        break;
}

```

Twitter API Proxy

```

#key requested from Twitter
CONSUMER_KEY = '6YqCEcBoqlA3LSiz2aIkQ'
CONSUMER_SECRET = 'Eu9rObgsKDL0lgzhQyajioERZ2n5zOoLsyZP2nJ'
ACCESS_TOKEN = '62385041-0QDj50lhicEkMBZjlqQXCL596kVCcVmcZ2Jq5Mucg'
ACCESS_TOKEN_SECRET = 'sZii9CUdsMZTfCgpJs8ClxCzA8y1WefcFQ6KehKRP'

USER_PASSWORD = 'password'

def do_proxy(self, method):
    orig_url = self.request.url
    orig_body = self.request.body

    new_url, new_path = self.conver_url(orig_url)

    if new_path == '/' or new_path == '':
        global gtap_message
        gtap_message = gtap_message.replace('#gtap_version#',
gtap_version)
        return success_output(self, gtap_message )

    username, password = self.parse_auth_header(self.request.headers)
    if password!=USER_PASSWORD :
        if username is not None :
            return error_output(self, 'Wrong password.')

    user_access_token = None

    callback_url = "%s/oauth/verify" % self.request.host_url
    client = oauth.TwitterClient(CONSUMER_KEY, CONSUMER_SECRET,
callback_url)

```

```

if username is None :
    protected=False
    user_access_token, user_access_secret = '', ''
else:
    protected=True
    user_access_token, user_access_secret = ACCESS_TOKEN,
                                           ACCESS_TOKEN_SECRET

additional_params = dict([(k,v) for k,v in parse_qs(orig_body)])
parme= dict([(k,v) for k,v in parse_qs(orig_url)])

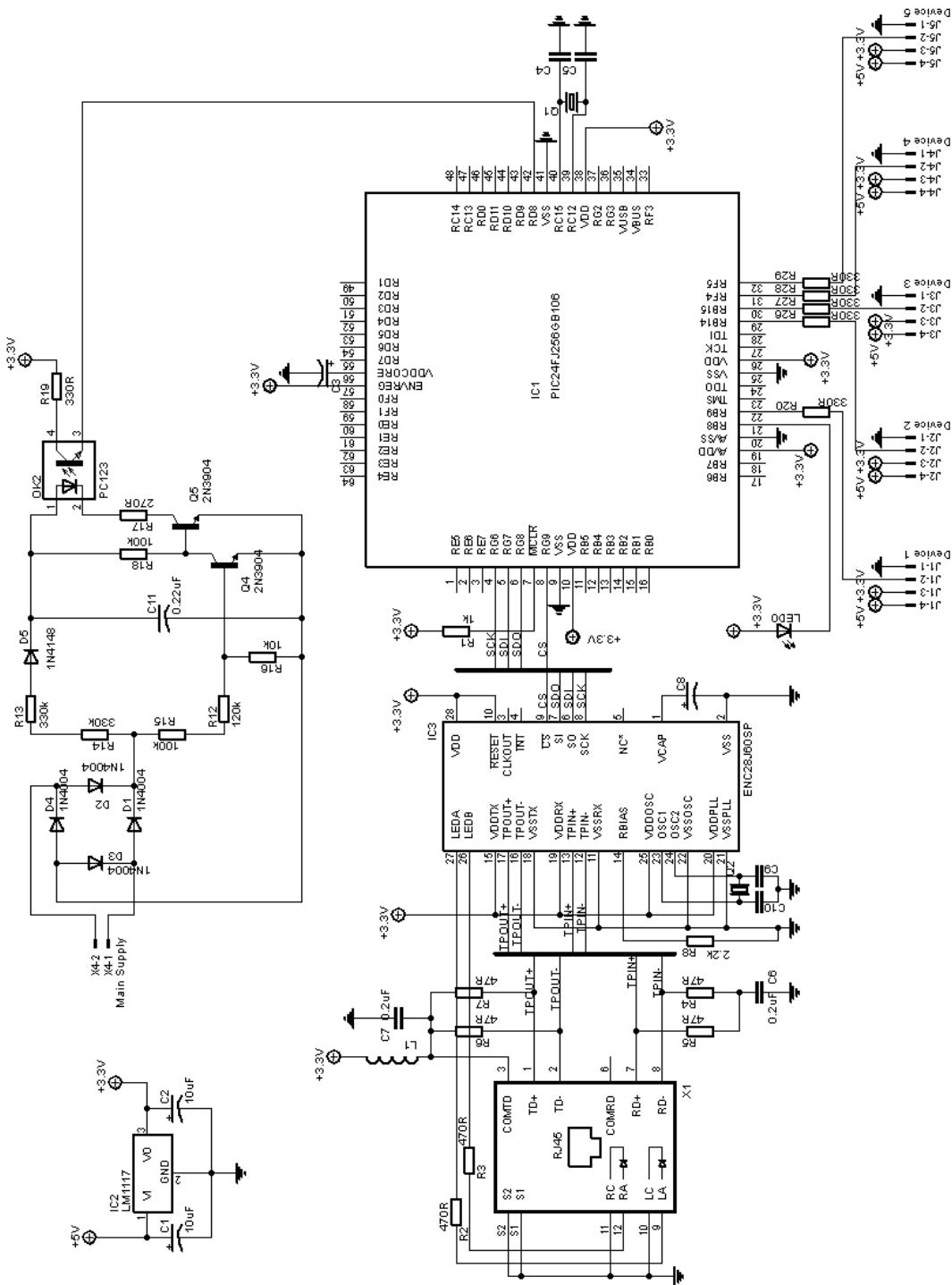
#set HTTP request method based on 'met' value
use_method = urlfetch.GET if (parme["met"]=='0') else
                urlfetch.POST

try :
#remove 'met' variable from the url before sending to Twitter
    new_url = new_url.replace("&met=1", "", 1)
    new_url = new_url.replace("met=1&", "", 1)

    data = client.make_request(url=new_url,
                              token=user_access_token,
                              secret=user_access_secret,
                              method=use_method,
                              protected=protected,
                              additional_params = additional_params)
except Exception,error_message:
    logging.debug( error_message )
    error_output(self, content=error_message)
else :
    #logging.debug(data.headers)
    self.response.headers.add_header('GTAP-Version',
gtap_version)
    for res_name, res_value in data.headers.items():
        if is_hop_by_hop(res_name) is False and
res_name!='status':
            self.response.headers.add_header(res_name, res_value)
            self.response.out.write(data.content)

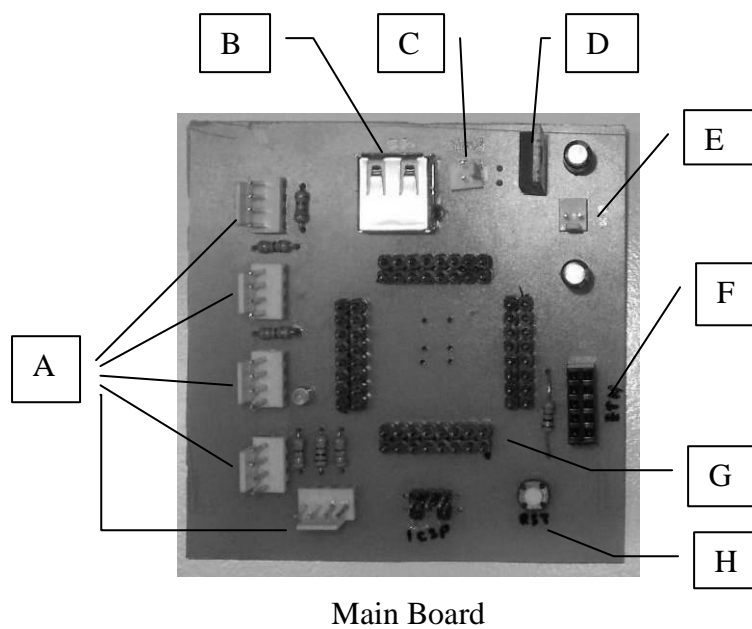
```


APPENDIX B: Schematic Diagram

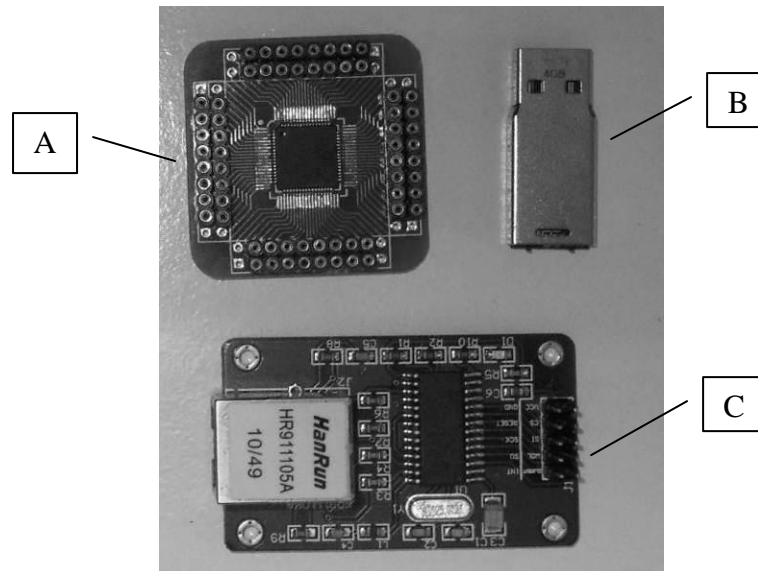


Schematic Diagram of Complete System

APPENDIX C: Pictures of Product

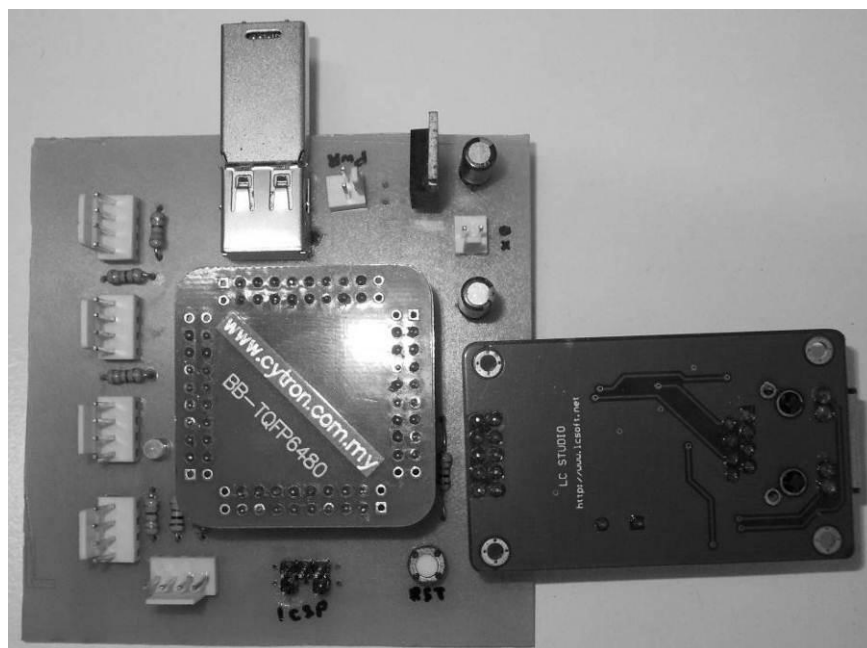


A	Connectors to Appliances
B	USB Connector
C	5V Power Supply Connector
D	3.3V Voltage Regulator
E	Zero Crossing Detector Input Connector
F	Ethernet Module SPI Connector
G	Pin Headers to Microcontroller
H	Reset Button

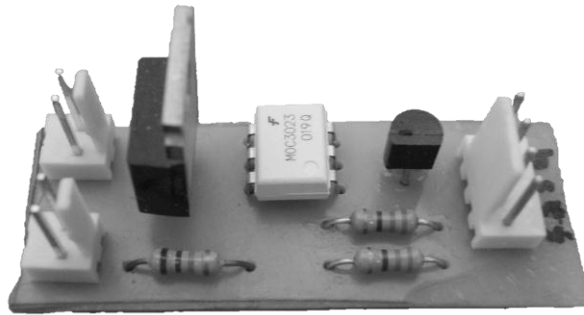


Components

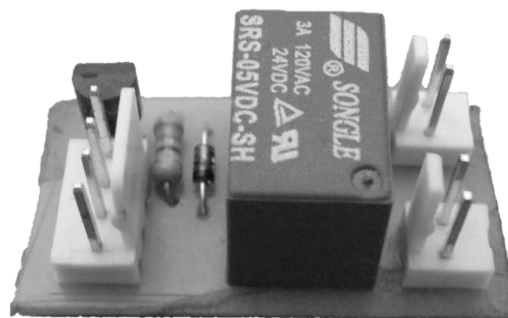
A	Microcontroller on TQFP to DIP Board
B	USB Flash Drive
C	Ethernet Module



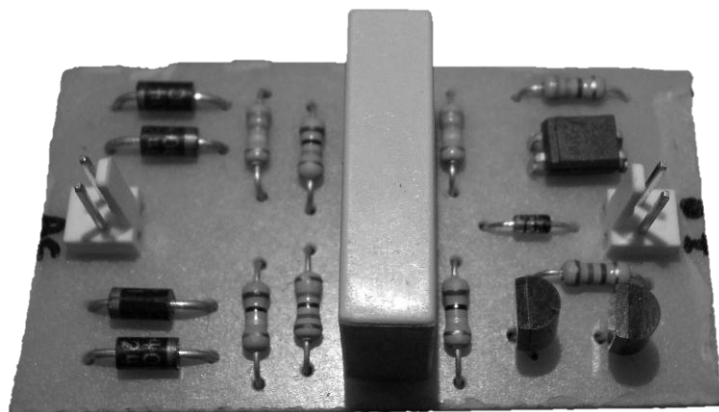
Main Board with Components Attached



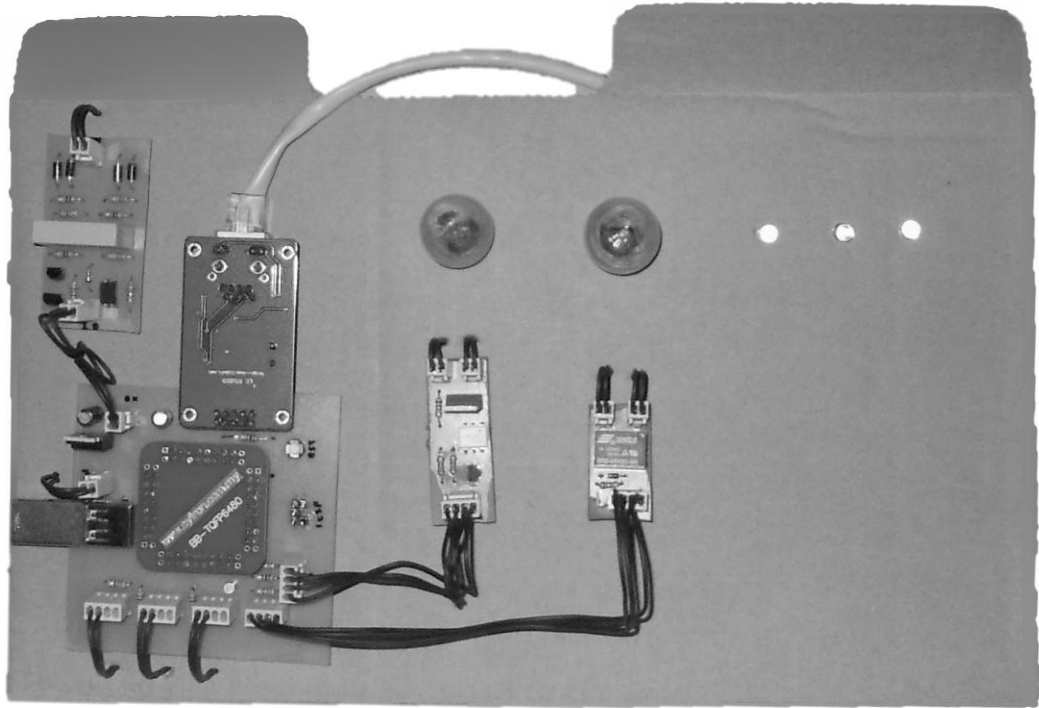
Dimmer Module



Relay Module



Zero Crossing Detector Module



Complete System - Test Configuration