



**ACCESS CONTROL USING WIRELESS FOR DATA COMMUNICATATION  
TO DATA TERMINAL**

**By**

**TAN JIA MIN**

**Dissertation**

**Submitted to the Electrical & Electronics Engineering Programme  
in Partial Fulfillment of the Requirements  
for the Degree  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)**

**Universiti Teknologi PETRONAS  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan**

# **CERTIFICATION OF APPROVAL**

## **ACCESS CONTROL USING WIRELESS FOR DATA COMMUNICATION TO DATA TERMINAL**

by

Tan Jia Min

A project dissertation submitted to the  
Electrical & Electronics Engineering Programme  
Universiti Teknologi PETRONAS  
in partial fulfilment of the requirement for the  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

Approved:



---

Mr. Patrick Sebastian  
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS  
TRONOH, PERAK

June 2010

## **CERTIFICATION OF ORIGINALITY**

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



---

Tan Jia Min



## ABSTRACT

Wireless access control system is using Wi-Fi technology for the systems to communicate with data terminal, a computer. It has been gaining popularity in access control industry for its flexibility, higher efficiency on systems installation and reduces cost of material. The main objective of this project is to develop wireless access control that can be accessed via HTML website using Microchip development board. TCP/IP protocol is used for its wireless technology, using ZeroG module that provides 802.11 standards. Application of wireless access control starts with initialization of hardware and computer through IP address assigned by access point. Transaction data can be monitored from computer through HTML website, and user can update database from computer to hardware device wirelessly. This project was divided into two phases. Phase I includes hardware design while Phase II is focusing on firmware algorithm design. Interrupt service routine was used to prompt development board to read input signal fed from Wiegand wiring signal. Wiegand data obtained is then compared to the database on hardware board and indication output is displayed. Hardware and firmware algorithm are designed, tested, debugged and verified for few user and show significant result. This operational project is complete and has achieved its objectives.

## **ACKNOWLEDGEMENT**

I would like to express the greatest gratitude to several people for their contribution. This project would have been incomplete without their advices and technical supports.

First and foremost, my final year project supervisor, Mr. Patrick Sebastian, for all his invaluable help, advices, suggestions, and encouragement given while doing the project, right from its conception to its completion.

I would like to gratefully thank Mr. Yap and Ms. Stacy Wong, manager and supervisor from Micro ID and all the staffs for their full supports on technical issues and resources.

I would like to acknowledge my colleague, Ms. Ong Wen Sher who provides her support on designing HTML data organization on computer. Her help has helped me to complete this project on time.

Last but not least, I would like to thank Electrical and Electronics Engineering Department for providing resources, facilities, and needs throughout this project.

# TABLE OF CONTENTS

LIST OF FIGURES.....	viii
LIST OF TABLES .....	x
ABBREVIATIONS AND NOMENCLATURES.....	XI
CHAPTER 1 INTRODUCTION .....	12
1.1 Background of Study .....	12
1.2 Problem Statement .....	13
1.3 Project Overview .....	14
1.4 Objectives .....	15
1.5 Scope of Studies .....	15
CHAPTER 2 LITERATURE REVIEW .....	16
2.1 ZeroG Wi-Fi Module.....	16
2.2 Wiegand 26-bit Format .....	18
2.3 TCP/IP .....	19
2.3.1 The Microchip TCP/IP Stack.....	20
2.4 Microchip PIC24F Microcontroller.....	21
2.5 Common Gateway Interface (CGI) .....	23
CHAPTER 3 METHODOLOGY .....	24
3.1 Proposed Method.....	24
3.2 Tools/Equipments Required .....	27
3.3 Project Progress.....	28
3.3.1 Lab Activities.....	28
3.3.1.1 Working with C30 and MPLAB IDE.....	28
3.3.1.2 Interrupt Handling .....	28
3.3.2 Project Development on Micro ID MX7 Board .....	29
3.3.2.1 Prototype hardware debugging.....	29
3.3.2.2 Firmware Development.....	29
3.3.3 Product development on Microchip Explorer 16 Development Board.....	30
3.3.3.1 Hardware development.....	30
3.3.3.2 Firmware Development.....	31
3.3.3.3 Source Code Flow Chart .....	31
3.4 Key Milestones/Project Activities.....	33



CHAPTER 4 RESULTS AND DISCUSSION .....	34
4.1 Results .....	34
4.1.1 Lab Activities.....	34
4.1.2 Project Development on Micro ID MX7 controller board ....	35
4.1.2.1 Hardware Debugging .....	35
4.1.2.2 Firmware Development.....	36
4.1.3 Project Development on Microchip Explorer 16 Development Board.....	37
4.1.3.1 Hardware Debugging .....	37
4.1.3.2 Firmware Development.....	37
4.2 Discussion .....	42
4.2.1 Lab Activities and development on Micro ID MX7 Controller Board.....	42
4.2.2 Project Development on Explorer 16 Development Board ...	43
4.2.2.1 Additional devices/changes on hardware .....	43
4.2.2.2 Access control application on firmware algorithm design.	43
4.2.2.3 HTML user interface customized for access control.....	44
CHAPTER 5 CONCLUSION AND RECOMMENDATION .....	45
5.1 Conclusion.....	45
5.2 Recommendation.....	45
REFERENCES .....	46
APPENDICES.....	48
Appendix A Gantt chart for final year project .....	49
Appendix B Gantt chart for final year project II .....	50
Appendix C ZeroG wi-fi module .....	51
Appendix D Microchip development board.....	52
Appendix E Firmware Source Code.....	53

## LIST OF FIGURES

Figure 1 Stand-alone access control system.....	12
Figure 2 PC-linked access control system .....	13
Figure 3 ZeroG Wi-Fi module .....	17
Figure 4 Sample Wiegand data stream.....	18
Figure 5 26-bit Wiegand format data .....	18
Figure 6 TCP/IP layers.....	19
Figure 7 Comparison of Microchip TCP/IP Stack and TCP/IP Reference Model .....	21
Figure 8 PIC24F memory organization.....	22
Figure 9 Working mechanism and data exchange mode of CGI .....	23
Figure 10 Final Year Project Process Flow.....	25
Figure 11 Source Code Development Chart .....	25
Figure 12 Microchip Development Board .....	26
Figure 13 Micro ID MX7 controller board .....	26
Figure 14 Interrupt Handling .....	28
Figure 15 Modification made on Explorer 16 board.....	30
Figure 16 Flow chart for board initialization .....	31
Figure 17 Flow chart for main program.....	32
Figure 18 Flow chart for interrupt service routine .....	33
Figure 19 LEDs blinking on development board.....	34
Figure 20 Lab session on PIC24FJ128.....	35
Figure 21 Modified MX7 controller board .....	36
Figure 22 Debugging controller board using PICkit2.....	36
Figure 23 Explorer 16 board with EM card reader and ZeroG wireless module .....	37
Figure 24 EM card being read by card reader.....	38
Figure 25 User ID showed on LCD display .....	38



Figure 26 Default HTML website during start-up ..... 39

Figure 27 HTML format displaying access control systems ..... 40

Figure 28 Add function for user ID to be stored in database ..... 40

Figure 29 Display user when transaction occurs..... 41

## **LIST OF TABLES**

Table 1 Tools required in project development .....	27
Table 2 Input/output devices' functionality .....	29
Table 3 Planned task for Final Year Project II.....	33

## **ABBREVIATIONS AND NOMENCLATURES**

ADC	Analog to Digital Conversion
ARP	Address Resolution Protocol
CGI	Common Gateway Interface
EEPROM	Electrically Erasable Programmable Read-Only Memory
HTML	HyperText Markup Language
IP	Internet Protocol
LAN	Local Area Network
LSB	Least Significant Bit
MSB	Most Significant Bit
MCU	Microcontroller Unit
MPFS	Microchip File System
SPI	Serial Peripheral Interface
TCP/IP	Transmission Control Protocol / Internet Protocol

# CHAPTER 1

## INTRODUCTION

### 1.1 Background of Study

Access control is process of individuals are identified and being permitted to certain accessibility to information, systems or resources [2]. Individuals experience access control in normal daily routine including to access entrance to individuals' office, to log in individuals' computer or ATM system in a bank [8]. Access control plays an important role as to protect private and confidential information, transaction or area from unauthorized individuals [1].

At the present time, access control has varied into different functionality and features, yet still underlying the main principle of preserving and protecting the confidentiality, integrity and availability of information, systems and resources. Physical access control allows accessibility of an individual to a certain area [7]. Individuals are given tokens such as access pin number, electromagnetic (EM) card or using their own fingerprint to seek the permission of accessing a building or an area.

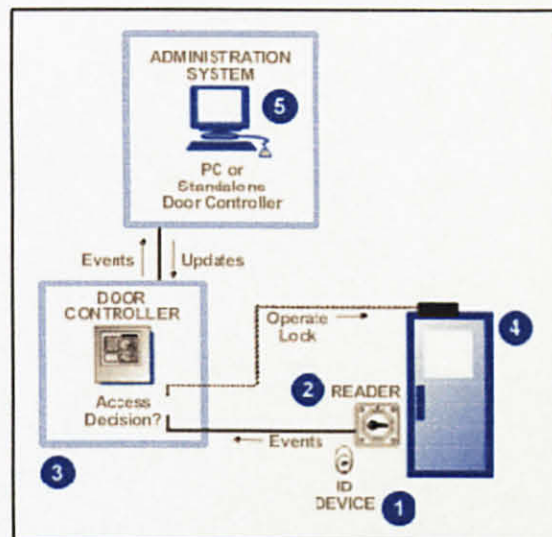
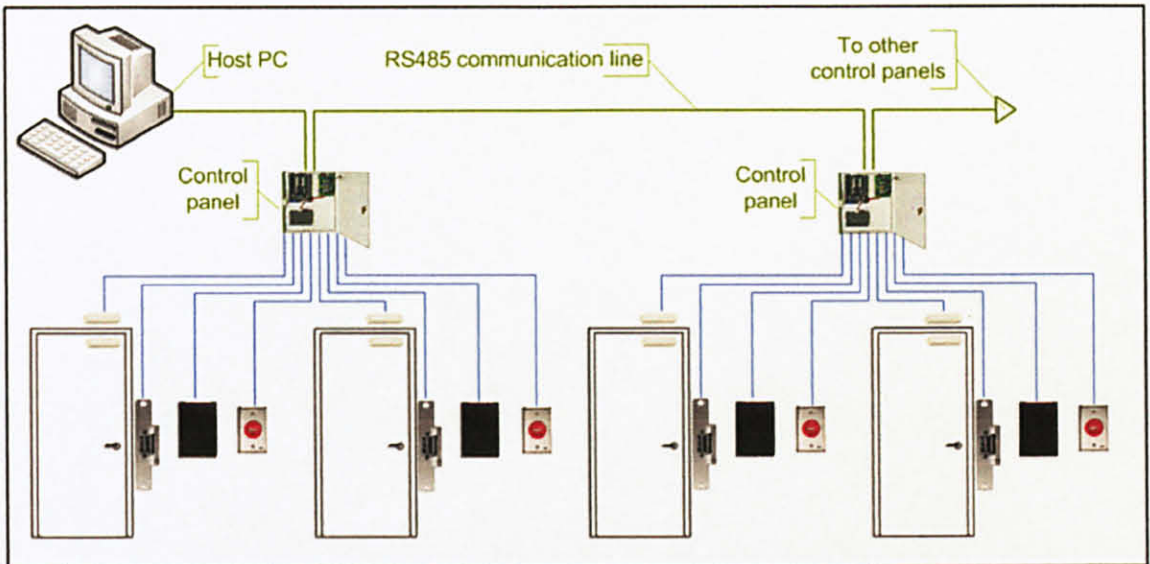


Figure 1 Stand-alone access control system

Physical access control can be divided into two main categories:

- Stand-alone access control
- PC-linked access control

Stand-alone access control is a basic system of access control which usually is installed at any physical barrier without a computer as central data base. It is simple and suitable to be used in small buildings. On the other hand, PC-linked access control involves a few access control and a computer as central processing unit and it retrieves, processes and analyses data in access control systems for detailed accessibility of individuals. It is more complicated and hence often used in bigger building or strict security area.



**Figure 2** PC-linked access control system

## 1.2 Problem Statement

Physical access control system is gaining more attention for private and security purposes. Conventional PC-linked access control system uses RS485 or RS232 cable to communicate between host computer and access control system [9]. With such transmission medium, additional software such as SSoftNet is required to interpret data retrieved from access control. Accessibility of data could be



inconvenient sometimes as user has to configure setting or update database on controller hardware itself, which requires rebooting; or, through external software on PC such as SSoftNet that sometimes failed to organize data in order. Apart from that, transmission medium using cable limits the distance between access controls and host computer. Due to its limitation, conventional access control systems have smaller coverage area and inconvenience during installation process.

Hence, these problems drive to the focus of this prototype-based project, a wireless PC-linked access control system that serves as an alternative solution to conventional systems. With Wi-Fi technology, it provides flexibility and wider coverage area for the system's installation as repeater (access point) will be added to enhance wireless signal. Also, user can retrieve data in access control systems at anywhere and anytime via LAN connection through internet browser.

### **1.3 Project Overview**

In collaboration with Micro ID Sdn Bhd, access control systems using Wi-Fi technology to communicate with PC was proposed. Conventional access control systems require user to monitor data transaction and add or delete authorized personnel by interfacing with access control controller. However, this project will provide flexibility to user, as data monitoring and authorizing personnel can be done via PC. Using Wi-Fi protocol, access control systems will communicate in TCP/IP stacks with PC and presented it in html format, a more graphic user interface as compare to hardware controller.

## **1.4 Objectives**

This project is expected to achieve the following objectives at the end of Final Year Project:

- To design and develop an algorithm to control microcontroller in access control to transmit data wirelessly to communicate with PC
- To develop User Interface on PC in HTML and Telnet files for user to retrieve and analyze data transaction in access control systems
- To develop wireless access control systems that is practical in the industry

## **1.5 Scope of Studies**

In general, the project encompasses the following scopes of study:

- C language on developing project's algorithm
- Microcontroller features and application
- Application of TCP/IP stack on embedded systems on wireless transmission
- HTML and Telnet application for developing user interface between PC and users

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 ZeroG Wi-Fi Module**

Wi-Fi is wireless technology that uses radio waves that enables connection for data sharing purposes between two or more devices wirelessly [12]. The organization which owns Wi-Fi (registered trademark) term, Wi-Fi Alliance, defines Wi-Fi as any “wireless local area network (WLAN) system that are based on Institute of electrical and Electronics Engineers’ (IEEE) 802.11 standards” [11]. Wi-Fi has gained its popularity over conventional wired network between devices such as computers for data transferring as it requires minimum usage of hardware [13].

Basically, Wi-Fi operates without any physical wired connection between data terminal by using radio frequency, RF technology, which is frequency within electromagnetic spectrum associated with radio wave propagation [14]. When RF current is applied to an antenna, an electromagnetic field will be created and hence can be propagated through space. A wireless access point (WAP) connects a group of wireless devices to an adjacent wired LAN, which is similar to a network hub, also it is to broadcast wireless signal for other data terminals to detect and are connected to it.

Wi-Fi is widely used in many applications and consumers electronics especially computers, major operating systems, PDAs, mobile phones and others. Any devices that are tested and approved as “Wi-Fi Certified” by Wi-Fi Alliance is interoperable to each other, even they are produced from different manufacturers [15].

ZeroG Wi-Fi module (Refer Appendix B) will be used in this wireless access control systems. It is designed to be easily fit in embedded systems in applications such as consumer electronics, remote device management, medical, health and fitness



applications and etc., allowing users to access information from internet. ZeroG Wi-Fi provides functions such as [14]:

- Robust networking stack and Wi-Fi driver that is compatible with microcontrollers
- Secure network connectivity supporting WEP, WPA and WPA2
- A community of development partners with expertise in RF design, software drivers, and building backend servers and client devices.
- Certified modules to minimize certification efforts navigating through the process for government and industry compliance



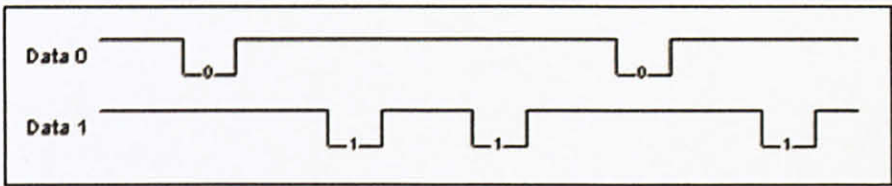
**Figure 3 ZeroG Wi-Fi module**

ZeroG Wi-Fi module consists of ZG2100 single-chip transceiver, a single chip 802.11 radio supporting data rates up to 2M bits per second, which with all associated RF components, crystal oscillator, and bypass and bias passives along with a printed antenna. With these features, the module provides a fully integrated Wi-Fi solution that can be controlled by an 8 or 16 bit processor or microcontroller. ZeroG Wi-Fi module is designed for low-power and low-duty applications and has four different power modes which are extremely low leakage and has fast “wake-up” architecture. ZeroG Wi-Fi module supports AES and RC4 based ciphers (WEP, WPA, WPA2 security) and is under FCC Certified and Wi-Fi Certified.

## 2.2 Wiegand 26-bit Format

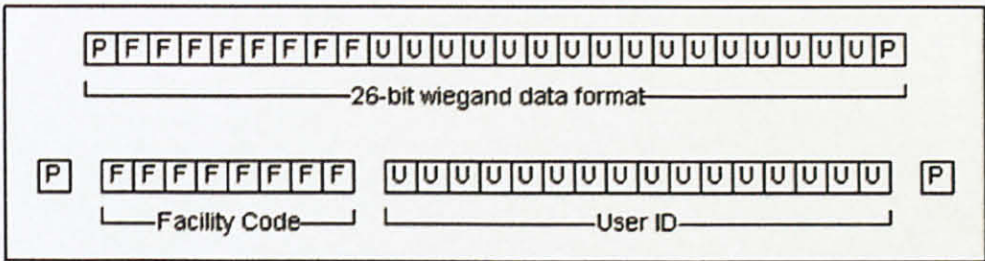
Due to the wide use of Wiegand technology in 1980's, when Wiegand card were factory coded with magnetic wires embedded in between plastic sheets most card readers implemented the "Wiegand interface". Hence, resulting Wiegand format is a wiring standard which is widely used in access control industry field. Wiegand interface are devices that receive input from users' identification token and output into data of 26-bits, 32-bits or 40-bits Wiegand data.

A Wiegand standard wiring consist of five wires: DATA0, DATA1, LED0, power supply and ground. DATA0 and DATA1 are the two signaling wires that transmit Wiegand data serially. Negative pulse on DATA0 line represents a 0 bit data, while negative pulse on DATA1 line represents a 1 bit data. Figure 4 shows the Wiegand data stream of for binary value '01101'. Each dip on the line represents a change from 5V to 0V.



**Figure 4** Sample Wiegand data stream

26-bits Wiegand data is the most commonly used format compare to 32-bits and 40-bits. This Wiegand format consists of a parity bit at Most Significant Bit (MSB), followed by 8-bit facility code, then a 16-bit user identification number, and a parity bit at Least Significant Bit (LSB). Two parity bits were added to ensure transmission of data is correct to avoid any confusion between access control systems and users.



**Figure 5** 26-bit Wiegand format data



### 2.3 TCP/IP

Commonly known as TCP/IP, TCP/IP Internet Protocol suite is used to communicate across any set of interconnected networks. It is a set of networks standards and convention on computers communication and interconnecting networks and routing traffic respectively [10]. TCP/IP comprises of three layers,

- TCP layers
- IP layers
- Sockets

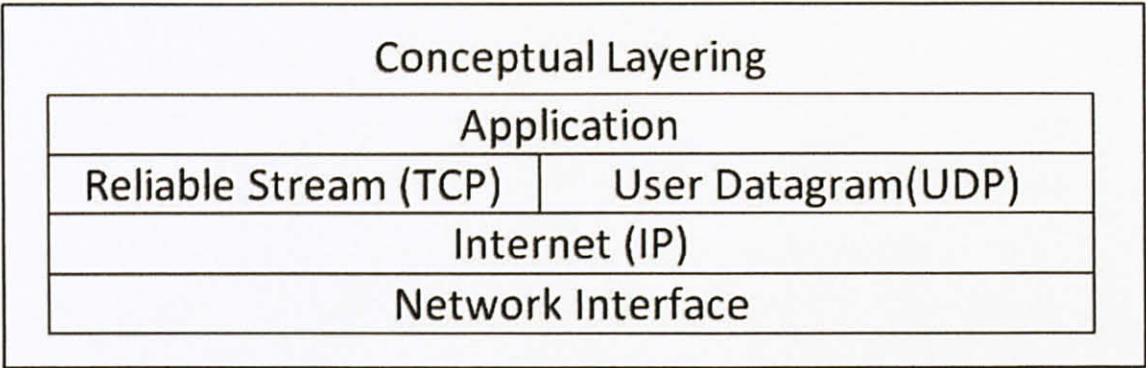


Figure 6 TCP/IP layers

The Transmission Control Protocol (TCP) is a communication protocol that provides format of the data and acknowledgements between two computers exchange for a reliable transfer. Also, it is the procedures computers involved to ensure data arrives correctly [10]. TCP is to have reinforcement on detecting error data during transmission until data is correctly received [11].

Internet Protocol (IP) has three main functions. It specifies the exact format of data transfer used throughout a TCP/IP internet. Also, it consists of the routing function that chooses a path to send data. Besides, it define a set of rules of that represent the idea of unpackets, how and when error messages should be generated and the conditions under which packets can be discarded [10]. IP can be concluded that it move packet of data from a place to another that based on a four byte destination address (the IP number) [11].

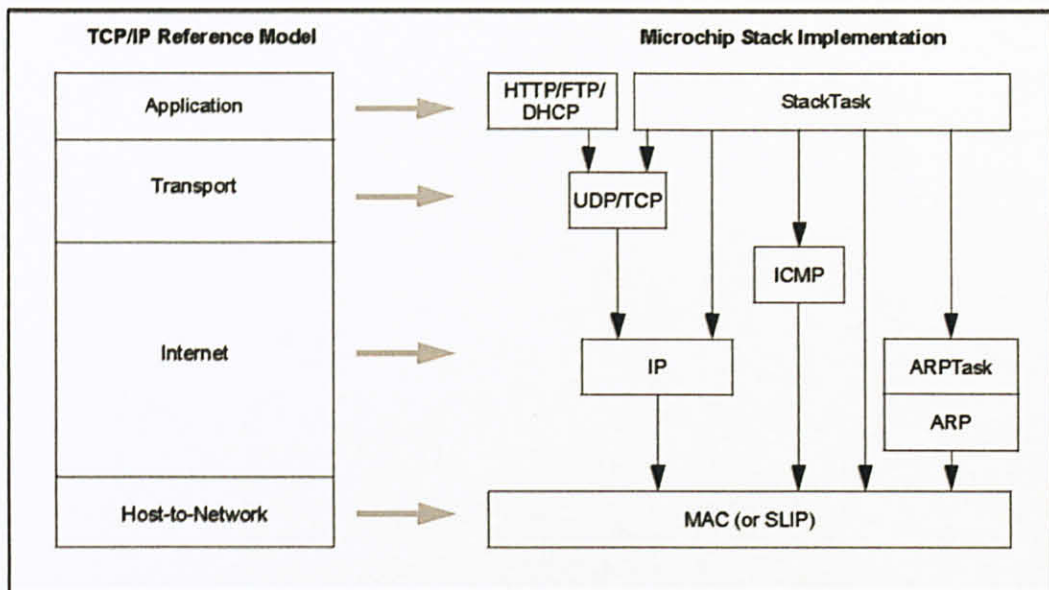
In application layer, socket is used to transmit data [11]. It can be known as packet in subroutine systems in TCP/IP network [10].

TCP/IP communication is primarily point-to-point that is a host computer from a point in the network communicates with another host computer, at another point. Higher layer application protocols usually are packaged together with TCP/IP in data transferring, including World Wide Web's Hypertext Transfer Protocol (HTTP), the File Transfer Protocol (FTP), Telnet that allows user to logon to remote computers and the Simple Mail Transfer Protocol (SMTP) [12].

### ***2.3.1 The Microchip TCP/IP Stack***

The Microchip TCP/IP Stack is a suite of programs developed by Microchip to provide services to standard or custom TCP/IP based application. It is implemented in a modular block, with all of its services creating highly abstracted layers. Microchip TCP/IP Stack divides itself into multiple layers, which is similar to TCP/IP reference model.

However, Microchip TCP/IP Stack differs from the reference model with its feature of directly accessing one or more layers which are not directly below it. Also, it has additional two modules, which are "StackTask" and "ARPTask" as compare to conventional TCP/IP stack. StackTask handles the operations of the stack and its entire module. On the other hand, ARPTask operates the services of the Address Resolution Protocol (ARP) layer. These two modules are cooperative tasks, which are implemented in the stack's cooperative multitasking systems as Microchip TCP/IP Stack is designed to be independent of any operating systems.



**Figure 7** Comparison of Microchip TCP/IP Stack and TCP/IP Reference Model

Figure 7 shows the comparison between Microchip TCP/IP Stack and a reference model. From the figure, it is observed that the Microchip TCP/IP Stack does not implement all of the modules, that reference model normally employ. However, they can always be implemented as a separate task or module if required. Microchip's stack will implement additional protocols based on this stack.

## 2.4 Microchip PIC24F Microcontroller

Microchip PIC24F microcontroller is chosen to be implemented in wireless access control project. In the programmer's model, it has 16-bit working register. Each of the working registers can act as a data, address or address offset register. The 16<sup>th</sup> working register (W15) operates as a Software Stack Pointer for interrupts and calls. The program to data space mapping feature lets any instruction access program space as if it were data space. The Instruction Set Architecture (ISA) has been significantly enhanced beyond that of the PIC18, but maintains an acceptable level of backward compatibility, is optimized by high level language such as C language. The core supports Inherent (no operand), Relative, Literal, Memory Direct and Three group of addressing modes. All modes support Register Direct and various Register Indirect modes. Each group offers up to seven addressing modes. Instructions are



associated with predefined addressing modes depending upon their functional requirements. For most instructions, the core is capable of executing a data (or program data) memory read, a working register (data) read, a data memory write and a program (instruction) memory per instruction cycle.

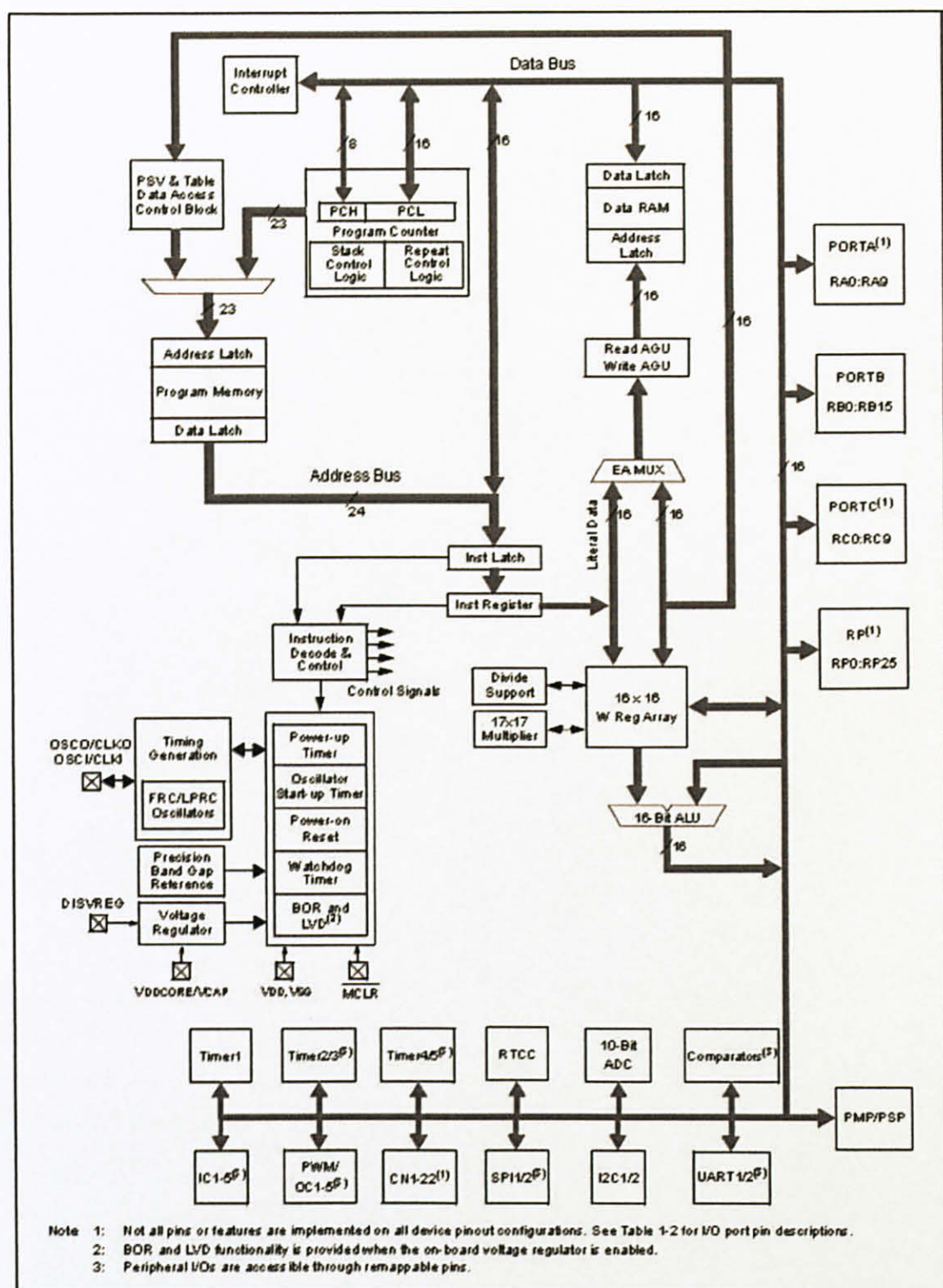
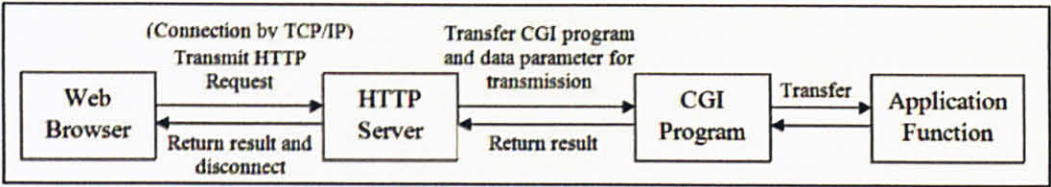


Figure 8 PIC24F memory organization

## 2.5 Common Gateway Interface (CGI)

Common Gateway Interface (CGI) is a standard interface of external extended application program interacting with WWW server [4]. It is an interface used to communicate with HTML form and server program, a static information provider. CGI request is defined as users on client side send a HTTP request to the server through browser and then the script program is executed according to file name of CGI script program of client request [5]. When the program is executing, client information is transferred to the running program through server. After program execution, operation results were sent to the server and the server take over the role to transfer result obtained back to client side. HTTP server may create HTML document dynamically via execution of CGI program [4].



**Figure 9** Working mechanism and data exchange mode of CGI

Many languages such as C, C++, Perl, Visual Basic can be used to design the algorithm of CGI application, but commonly in C.



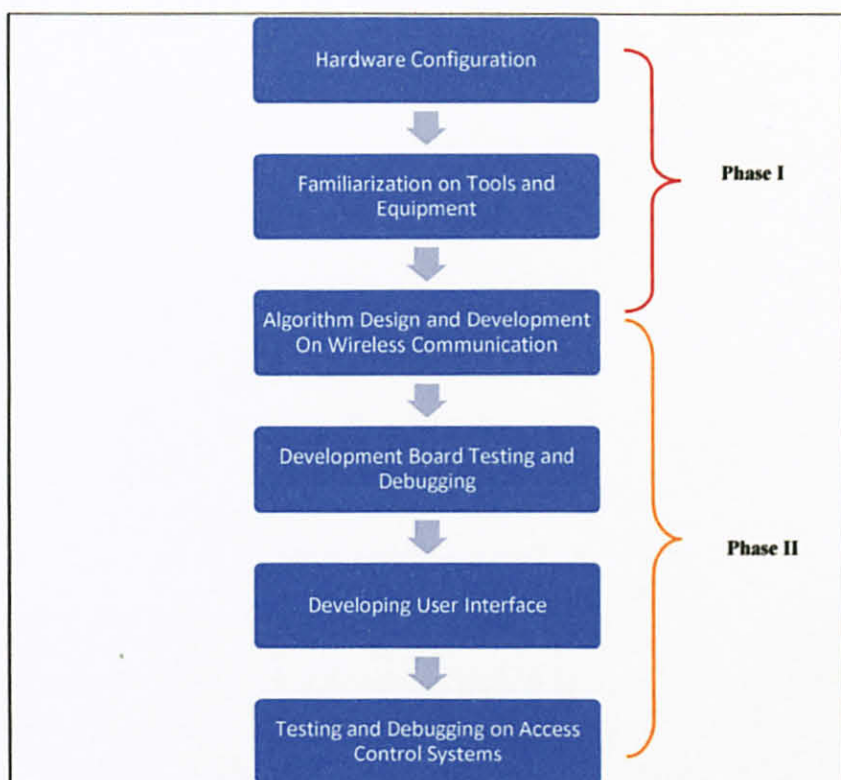
## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Proposed Method**

From the studies, the overall project will be divided into two main phase, Phase I and Phase II. Phase I of the project includes on hardware design and configuration. These include Microchip Development Board, Micro ID MX7 controller, ZeroG wireless module and other output indication devices. Also, laboratory activities were performed to study C language on C30 compiler and both Microchip development board and MX7 controller. Phase II was focusing on firmware algorithm design: on receiving input from Wiegand card reader, verify it with database and execute proper output; and, updating user ID database via computer internet browser. The prototype project was tested, debugged and verified for few users' transaction. Both phases will be conducted in semester 1 and semester 2 correspondingly.

Figure 10 shows the process flow throughout the one year period from hardware configuration and initialization until testing and debugging stage on an actual access control systems.

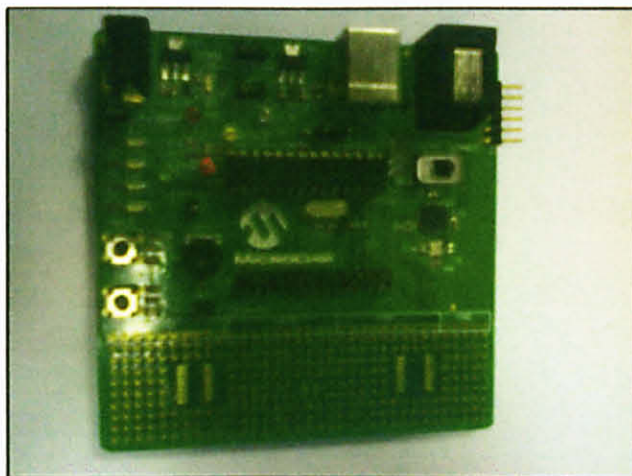


**Figure 10 Final Year Project Process Flow**

Figure 11 shows the source code development in wireless access control. In board initialization section, ports and setting of Microchip Development board are configured to provide suitable setting for programming code later. TCP/IP Stack software configuration is to allow data transmit data to computer in network form. Main program will process corresponding input and instruct hardware to assign suitable output. HTML application will organize and display transmitted data in HTML form for further analysis.



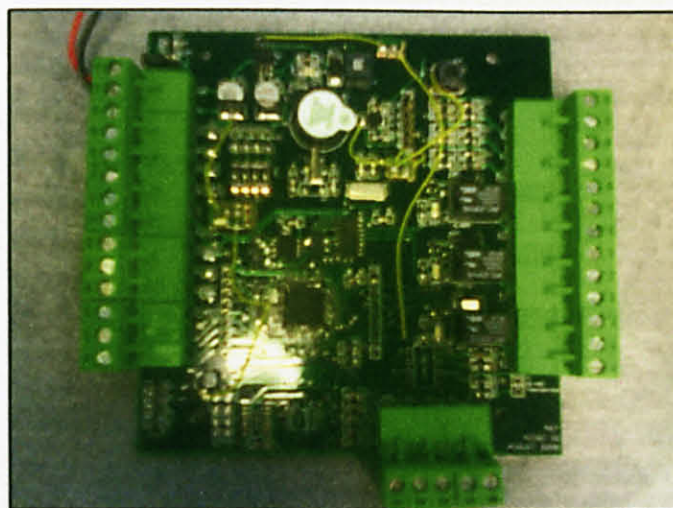
**Figure 11 Source Code Development Chart**



**Figure 12** Microchip Development Board

Figure 12 shows the Microchip Development Board which will be developed into wireless access control hardware during initial stage. Later, the firmware developed will be transfer into Micro ID's designed hardware.

Figure 13 shows Micro ID MX7 hardware board that is specially designed for wireless access control. MX7 hardware circuit board consists of PIC24FJ that serves as central processing unit of wireless access control systems, EEPROM to store users' data and transaction, and input ports for push button, Electromagnetic (EM) card readers, and output port for door lock.



**Figure 13** Micro ID MX7 controller board



### 3.2 Tools/Equipments Required

To develop this wireless access control systems, several tools are needed. The following tools are:

**Table 1** Tools required in project development

Category	Tools	Description
Software	MPLABIDE v8.33	An integrated toolset for the development of Microchip's PIC® and dsPIC® microcontroller applications. Version 8.33 is used as it allows PICKit2 debugging feature.
	Microchip C30 C Compiler	C Compiler for microcontroller PIC24FJ series.
Hardware	Microchip PICKit2 Programmer/Debugger	A development tool with user-friendly interface to program and debug Microchip's Flash families microcontrollers.
	Microchip Development Board Explorer 16	A general microcontroller board consists of LCD, I/O ports for user to understand and familiarize with the features and application of Microchip's microcontroller.
	Micro ID MX 7 Board	Controller board designed by Micro ID that serves as central processing unit in an access control systems.
	ZeroG Wireless Module	It is a single-chip Wi-Fi transceiver, which will be attached to access control systems to transmit data via wireless medium to communicate with PC.
	EM Card Reader	To serve as an input device to access control systems that read the data on an EM card and convert it into 26-bits Wiegand format, a standard protocol used in access control
	Wireless Access Point	Wireless router

### 3.3 Project Progress

#### 3.3.1 Lab Activities

Several lab activities on MPLAB IDE were performed in order to understand and familiarize with C30 C compiler and Microchip Development Board.

##### 3.3.1.1 Working with C30 and MPLAB IDE

This lab exercise introduces C language in C30 and compiled the source file and programmed Microchip Development Board using PICKIT2. This exercise introduced the structure of a main program that comprises of Initialization, Main, and Subroutines. The compiled program will turn on LED3 all the while.

##### 3.3.1.2 Interrupt Handling

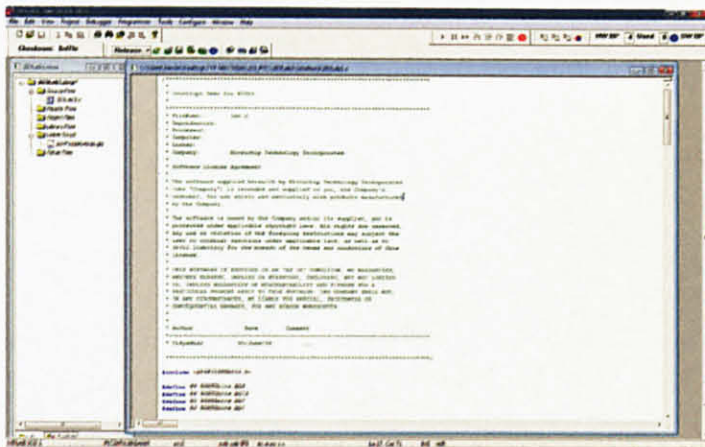


Figure 14 Interrupt Handling

Interrupt handling exercise shows how the user can set the level of priority on interrupt functions based on value of input switch. The program can later assign different output functions based on level of interrupt priority. This feature allows the main program to be multitasking.

### 3.3.2 Project Development on Micro ID MX7 Board

After Micro ID MX7 hardware circuit board which is specifically designed for wireless access control was ready, hardware debugging and firmware development were conducted.

#### 3.3.2.1 Prototype hardware debugging

MX7 board was designed on printed circuit board (PCB) and soldered with respective electronics components. Then, it was verified using connectivity test based on schematic design done by design engineers in Micro ID. Finally, the input output ports were tested to check their functionality after microcontroller was programmed.

#### 3.3.2.2 Firmware Development

Firmware development is generally referring to algorithm design that will instruct what microcontroller need to do. For this project, MX7 hardware consists of three input devices, two output indications, and an HTML format output on computer. Table 2 shown below describes the role of devices attached to MX7 controller board.

**Table 2** Input/output devices' functionality

Input/output Devices	Functionality
Electromagnetic (EM) Card Reader	To read the data stored in EM card and send out 26-bits Wiegand format to controller board
Push Button	An alternative for user to exit an enclosed area
Electromagnetic (EM) Lock	A lock that is attached to door and will be released if authorized personnel access the enclosed area
Buzzer	An indication for user
PC	Transaction data will be projected using HTML form

As described in Table 2, firmware needs to be capable of generating an interrupt service routine when there is an EM card entry. Then, it needs to verify the user and prompt the right decision. If it is a valid entry, EM lock will be released and vice versa. Also, the results will be shown to PC, displaying the transaction occurs at all time.



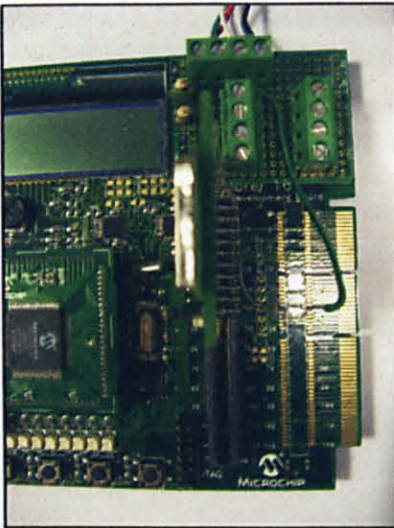
### 3.3.3 *Product development on Microchip Explorer 16 Development Board*

Project progress was decided to continue on developing Microchip Explorer 16 Development board as development on Micro ID MX7 controller board has encountered few problems on hardware design:

- Faulty circuit design on voltage regulator to step down 12V from power adaptor, causing voltage regulator failed to produce 3.3V. High voltage from power source has resulted in a few electronics components failure, including microcontroller
- Pin connections on Wiegand EM card reader to microcontroller were not connected to interrupt pins thus cannot generate hardware interrupts
- EEPROM on MX7 board could not be accessed as Serial Peripheral Interface (SPI) registers on microcontroller could not be declared resulting in data could not be stored in EEPROM
- Hardware interface for ZeroG wireless module was not included on MX7 board

#### 3.3.3.1 *Hardware development*

As Explorer 16 board is a general purpose development board, few modifications were made to suit the features of MX7 board as an access control controller board. Passive electronic components were added for as pull-up resistors and connector were connected to interface with Wiegand EM card reader, push buttons and electromagnetic (EM) lock.



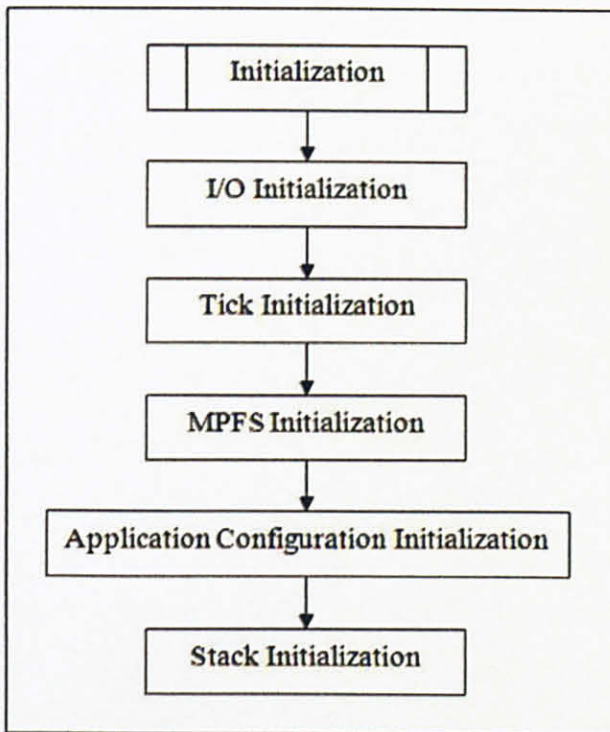
**Figure 15** Modification made on Explorer 16 board

### 3.3.3.2 Firmware Development

Firmware development on Explorer 16 development board will include interrupt functions on Wiegand EM reader when read in data, verifying input data with data stored in EEPROM, transmit required data wirelessly to PC in HTML format, and from PC transmit data back to controller board on users' identification and details. The inputs for this access control systems are Wiegand EM card reader and push button while the outputs are EM lock and html format on PC. On developing firmware, usage of external interrupt was used in Wiegand EM card reader to prompt microcontroller on getting ready to receive 26-bit Wiegand data. Also, paging method is used on EEPROM to store users' wiegand data on EEPROM.

### 3.3.3.3 Source Code Flow Chart

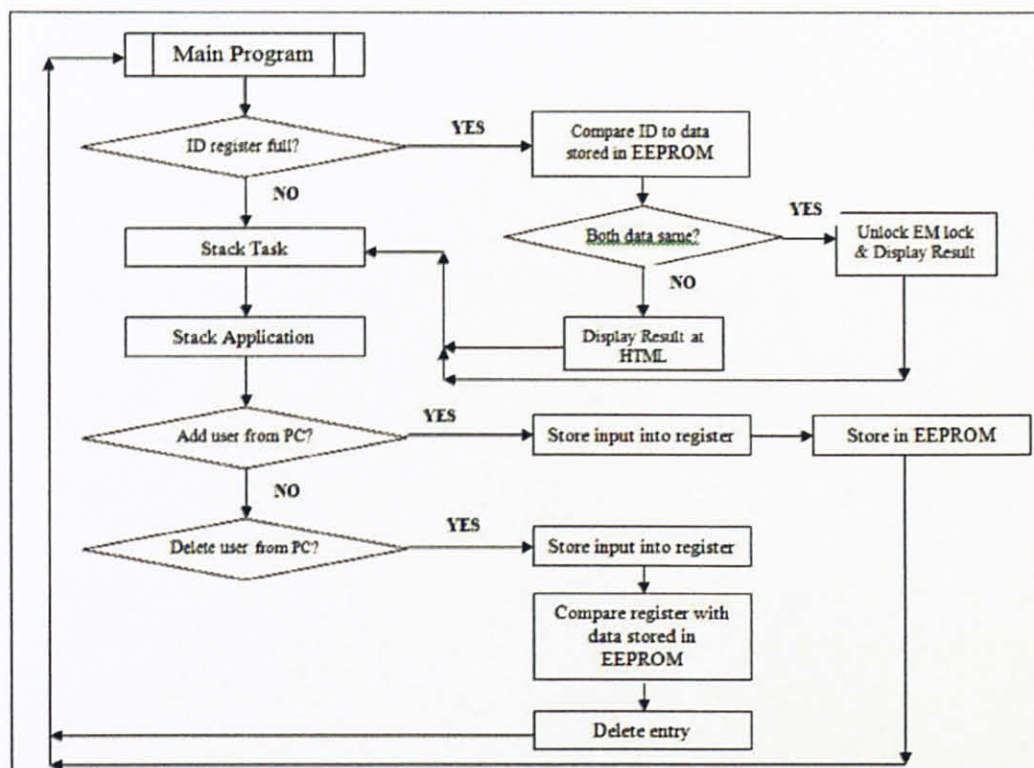
Flow chart of firmware development was prepared to display step-by-step logical approach to the given problem.



**Figure 16** Flow chart for board initialization

Figure 16 shows the algorithm flow for board initialization. I/O initialization will configure certain microcontroller pins into digital input output ports. Tick initialization is a tick manager that uses Timer0 interrupt to update the task part by

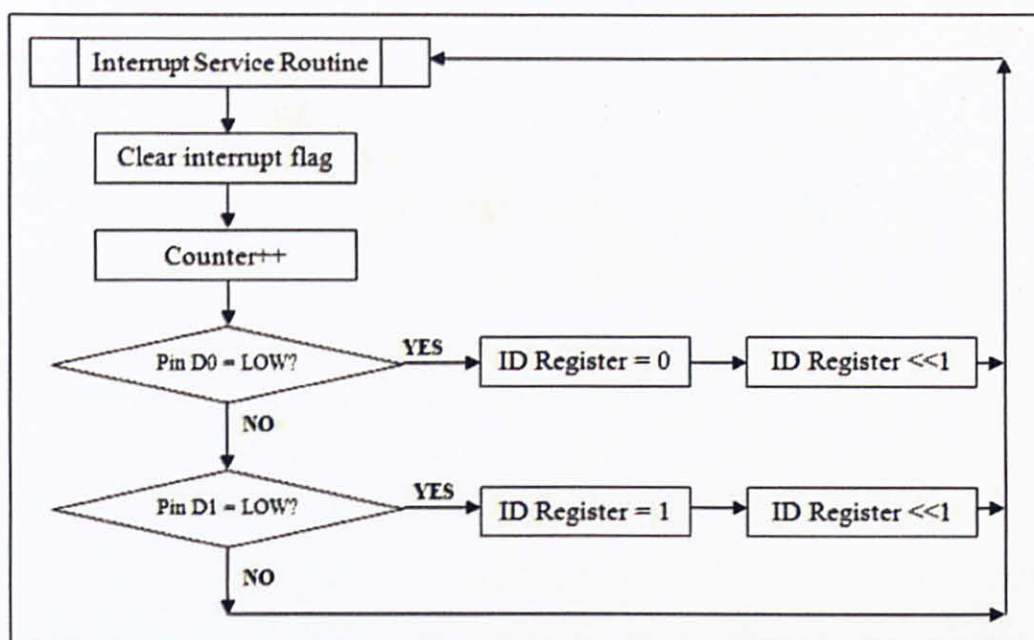
part. MPFS initialization is the configuration process of extracting data stored in EEPROM and convert into HTML format on PC. MAC address of the board was configured during application configuration initialization. Then, modules in TCP/IP stack will be configured in stack initialization.



**Figure 17** Flow chart for main program

From the main program flow chart, main program will keep looping to check on user ID register. If the register is filled with 26 bit of data, main program will compare ID to the data stored in EEPROM. Authorized user will be allowed to enter in and transaction data will be displayed on PC and vice versa. Stack Task will check for incoming packets, types of packets and calling for appropriate stack entity to process it. Stack Application is the core functions of each stack, defining its objective, flags and its features to process packets. Development board will process data when user ID is either being add or delete from PC, whether to store input into EEPROM or to delete the data entry from EEPROM. Via this method, EM card being fed from card reader can be verified as there is user database stored in development board's memory.





**Figure 18** Flow chart for interrupt service routine

Figure 18 shows the algorithm flow of interrupt service routine. When data signal at EM reader goes LOW, interrupt service routine will occur. Increment counter is added that when the counts reach 25, main program will verify the input from EM card reader. When interrupt occurs, value '1' or '0' will be assigned into register and shift to left side by a bit. This will result in an ID register which consist of 26 bits.

### 3.4 Key Milestones/Project Activities

**Table 3** Planned task for Final Year Project II

Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Task														
Verifying data in EEPROM														
Generate interrupt for input														
Display data to PC														
Add/Delete User from PC														
HTML formatting														

Table 3 displayed above describes the tasks that need to be completed during this fourteen academic weeks.

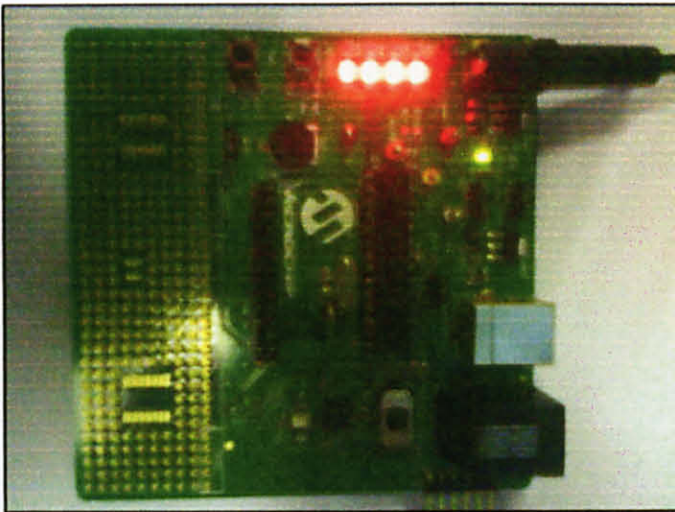
## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Results

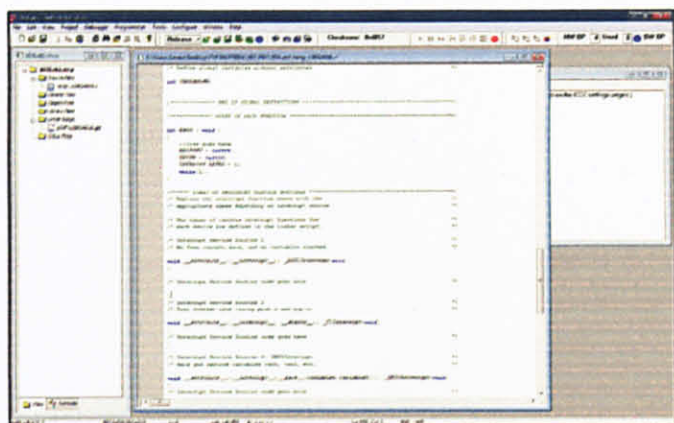
##### 4.1.1 Lab Activities

A working sample program code was tested on development board to test LED's blinking. This exercise is to provide better understanding on the usage of C language on Microchip C30 C compiler and the configuration of I/O (Input/ Output) ports on PIC24F microcontroller.



**Figure 19** LEDs blinking on development board

Figure 19 shows LED being turned on which is prompted by Switch 1 using polling-system approach. The program will check on input from Switch 1 all the time and if Switch 1 is pressed, LED will turn on. Another approach to blink LEDs is to use interrupt function. Hardware interrupt will process the high priority interrupt task which is to blink LED for a while then continue the uncompleted task in main program with the aid of program counter register in microcontroller.



**Figure 20** Lab session on PIC24FJ128

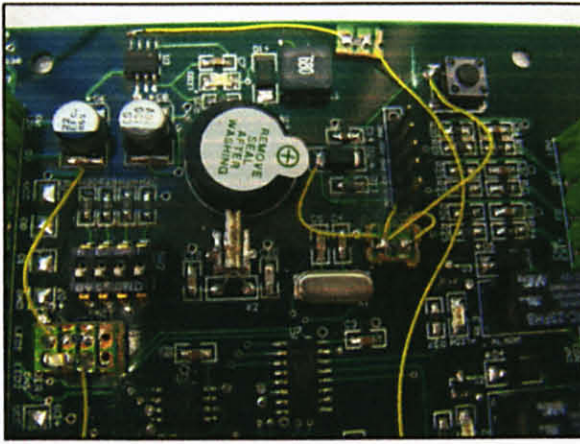
Figure 20 shows the lab sessions conducted on PIC24FJ128 controller that introduces the usage of C language on C30 C compiler using interrupt functions that will turn on LEDs. The following lab sessions were conducted on interrupt functions handling, attributes, PSV handling, ADC usage and 32-bits timer utilization. Timer0 interrupt was developed and tested on Development Board. Every occurrence of interrupt will count down a decrement counter value which its value was declared during initialization fraction. As counter value has not approach the value zero, LEDs will remain ON. This approach is known as system tick that allows a few tasks to be performed at the same time.

#### **4.1.2 Project Development on Micro ID MX7 controller board**

##### **4.1.2.1 Hardware Debugging**

MX7 controller board was soldered and verified using connectivity test and input/output test. The controller board was short-circuited after power-up as there was a few of errors being made including wrong configuration of voltage regulator resulting voltage supplied to the whole circuit exceeded the limiting level. Also, pull-up resistors were added to programming pin as microcontroller can differentiate between programming mode and normal-operating mode. Normal-operating mode runs at circuit's voltage level at programming pin while at programming mode PicKit2 will flush programming pin at higher voltage than circuit's voltage. An LED was added to MX7 board as indication purposes.





**Figure 21** Modified MX7 controller board

#### 4.1.2.2 *Firmware Development*

Firmware development is started with read and writes SPI EEPROM function. SPI protocol was used because it provides faster speed on data transmission. As the EEPROM was not connected to physical SPI pins on microcontroller, reassignment on SPI pins is required using software declaration. Read and write tasks using dummy value were tested on EEPROM. To verify source code, debugger mode was chosen to monitor desired registers stored. During debugger mode, registers that stored correspondence dummy value showed no response and hence confirming that source code was not functioning.

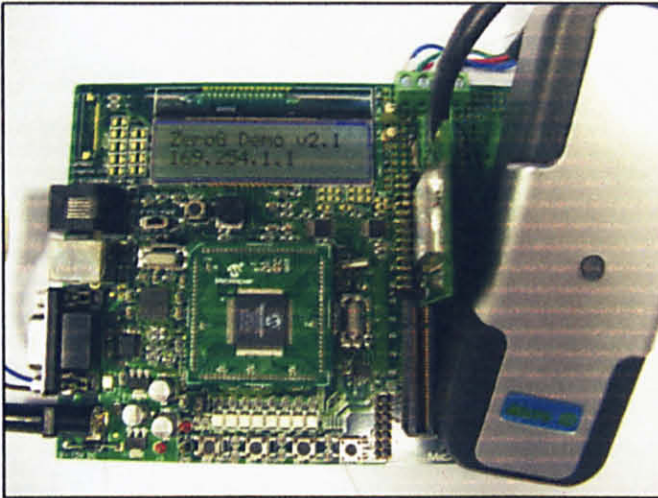


**Figure 22** Debugging controller board using PICkit2

### 4.1.3 Project Development on Microchip Explorer 16 Development Board

#### 4.1.3.1 Hardware Debugging

Since a few modifications made on Explorer 16 board, connectivity test were conducted to ensure added components were not short-circuited and damage the development board. Connectors were added to interface development board with EM card reader, EM lock and push button.



**Figure 23** Explorer 16 board with EM card reader and ZeroG wireless module

As shown in Figure 23, ZeroG wireless module and EM card reader were attached to Explorer 16 development board. LCD display on board showed an IP address, which indicates that PC could communicate with development board by using that address shown.

#### 4.1.3.2 Firmware Development

##### 4.1.3.2.1 Wiegand EM Card Reader

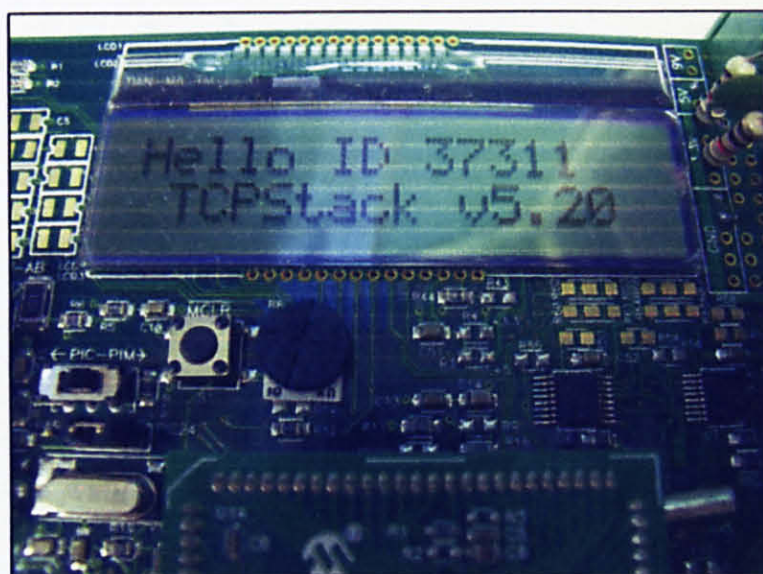
In order to prompt development board to feed in data from EM card reader, the reader device's connections are connected to external interrupt pins of microcontroller. When there is a change of logic level from high to low, interrupt service routine will be generated to receive 26-bit Wiegand data from two signal lines, Data0 and Data1. An increment counter was used to ensure 26-bit signals generated and to differentiate MSB and LSB parity bit.





**Figure 24** EM card being read by card reader

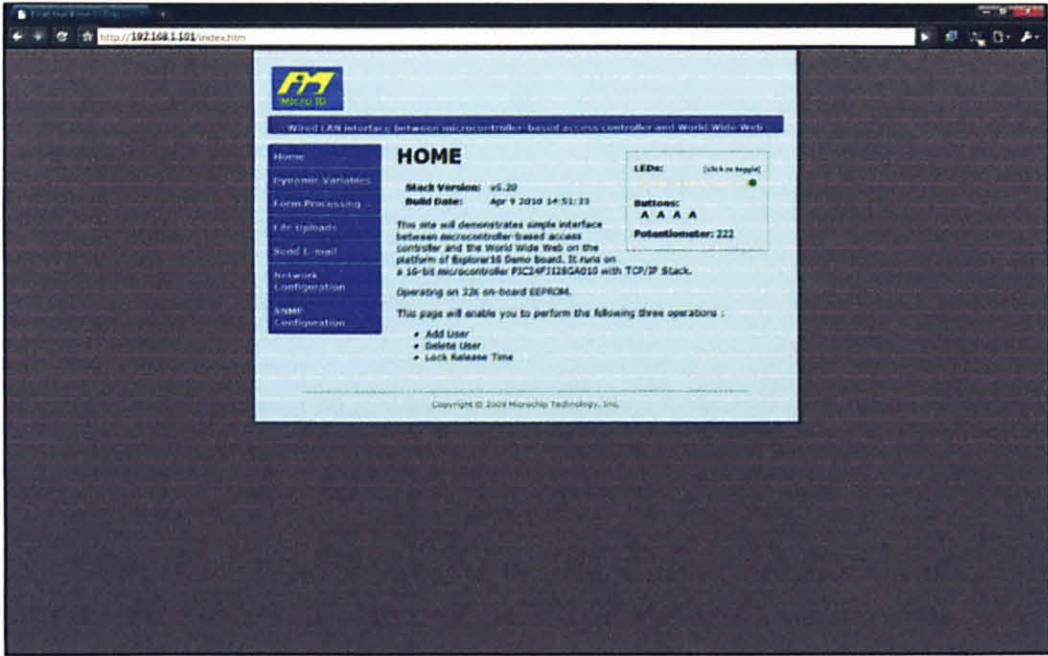
Figure 23 displayed the process of EM card being read by card reader. EM card is a LC circuit fabricated between PVC sheets. Red LED indication shows Wiegand 26-bit data was being fed into input port of development board.



**Figure 25** User ID showed on LCD display

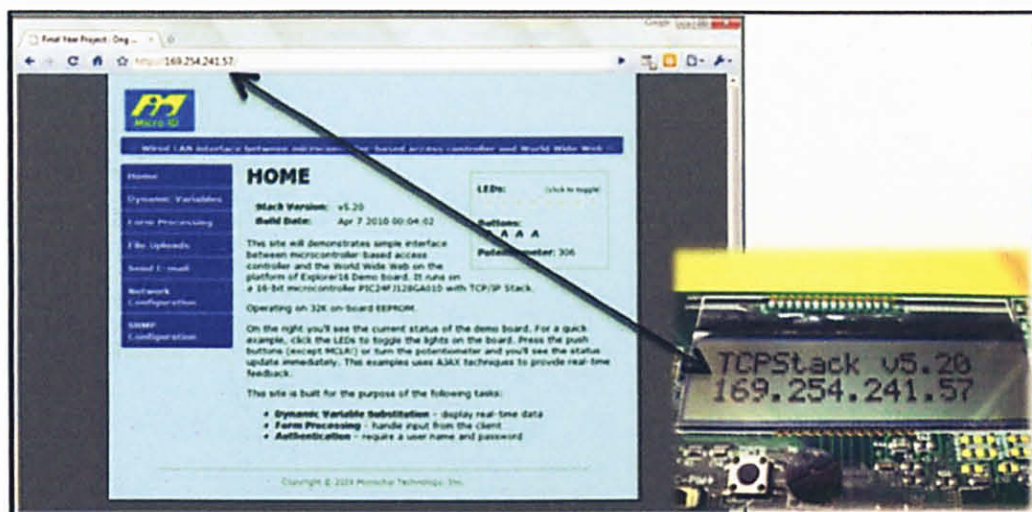
After EM card was being read, main program will process user ID from input port and compare it with the data stored in EEPROM. Once the data matches, it is known that ID is from authorized user and LCD display will show the user ID.





**Figure 26** Default HTML website during start-up

Access control system is displayed on computer using internet browser in HTML format provides convenience on data accessibility. Data shown on internet browser includes of build version of firmware, build date of firmware, eight LEDs indication, four buttons indication, potentiometer indicator, add and user ID features and transaction display. Potential meter indicator obtained from ADC functions, while LEDs blinking indication is obtained from Timer0 interrupt, showing tick features. When ZeroG wireless module is attached on development board, and connected to a LAN network through a wireless access point, a PC can access its data through the same network. IP address will be displayed on LCD display on development board. By accessing the IP address using web browser, the result will be shown in following figure.



**Figure 27** HTML format displaying access control systems

The html coding that is designed to display graphical interface for user was studied and modified to suit the data projection and data entry, which accommodate the features of wireless access control. Figure 27 portrayed that LED0 blinking, at one second period, showing that data from the board could be sent to PC instantaneously.

The **POST** method submits the data after the request headers are sent.

As an example, this POST form sets the text shown on the LCD display:

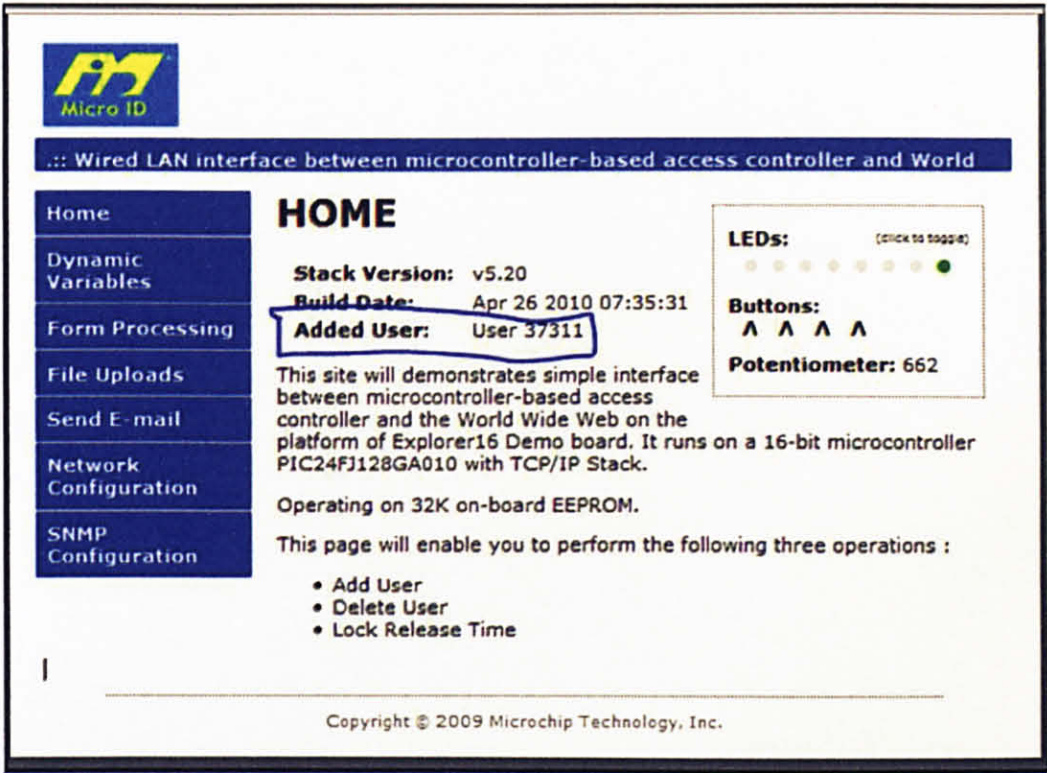
**ADD USER:**

Copyright © 2009 Microchip Technology, Inc.

**Figure 28** Add function for user ID to be stored in database

Figure 28 displays add function for user to store identification number in the access control system, specifically EEPROM on development board. When user key in 6-digit identification number on electromagnetic card, for example: 14568, and click save, the string of data will be sent through TCP/IP protocol and is received by development board. The string of data will be converted into integers and stored in two register, IDHigh and IDLow, then stored into EEPROM using paging method.

Vice versa, to delete a user, instead of storing IDHigh and IDLow into memory, these two registers will compare the data stored in memory and delete the entry.



**Figure 29** Display user when transaction occurs

Transaction occurs when user came across flashing their user ID to card reader. Wiegand input received will be processed and microcontroller will determine whether the user is authorized user or not. Valid transaction will display user ID at 'Added User' column showed in Figure 29.



## 4.2 Discussion

### 4.2.1 Lab Activities and development on Micro ID MX7 Controller Board

Through the sample code demonstration on Microchip Development Board, the C language portrayed using C30 compiler is more complicated than CCS C compiler as it has more built in functions. However, with these attributes and built in functions, the programming code can be utilized in a more efficient way.

Based on the lab activities performed, Timer0 interrupt and Interrupt functions play an important role in wireless access control systems. System tick approach was implemented using Timer0 interrupt as this method will replace the conventional delay function. For example, push button, an input in access control systems, is pressed to unlock a door for 10 seconds. Using system tick approach, a counter is declared with a certain value, then every occurrence of Timer0 interrupt will countdown the counter till it reaches zero. Door is remained unlock when value of counter has not reach zero and vice versa. At the same time, the main program in microcontroller will be able to process other data allowing the systems to be multitasking as compared to delay function that spend 10 seconds waiting for the door to remain unlock. Interrupt function is important in wireless access control systems as occurrence of emergency access requires the systems to service the highest priority of interrupt and halt the other tasks handled in access control systems. Lab sessions conducted on PIC24FJ128 microcontroller has given a deeper understanding on utilizing microcontroller using C30 C compiler.

The lab activities performed previously has provided a foundation in developing wireless access control systems on MX7 controller board. As MX7 controller board is the first version of circuit hardware, prototype debugging process is required to ensure the board will run smoothly. Few corrections and modifications were made for friendlier user interface and to fully utilize the board. LED indication was provided to monitor results and to verify MX7 controller board's functionality. Throughout this debugging process, it is known that user has to ensure the stability of voltage supply in circuit hardware design as it may cause damage to electronics circuit hardware.

#### *4.2.2 Project Development on Explorer 16 Development Board*

In the course of doing this project, the main challenge is to overcome algorithm design on user interface application along with wireless communication protocol. This project design was referring as much as possible to Microchip Wi-Fi solutions using ZeroG technology. However, few changes were made to accommodate the features of an access control system. The changes are:

- Additional devices/changes on hardware
- Access control application on firmware algorithm design
- HTML user interface customized for access control

##### *4.2.2.1 Additional devices/changes on hardware*

This project which is designed on Microchip development board requires additional and modification on hardware. Additional input output components such as push buttons, alarm, electromagnetic lock, card reader were added to function as a basic access control unit that is installed at a physical barrier. Also, a few additional and changes on pins assignment were made to accommodate the practice of EEPROM and external interrupt functions for card reader.

##### *4.2.2.2 Access control application on firmware algorithm design*

To prompt development board to read Wiegand input from card reader, interrupt service routine was developed. Negative pulses at Wiegand signaling wire will generate interrupt. When the user ID data from EM reader is received through wireless transmission, data received will be checked on LSB and MSB parity bit, to ensure error free input data. Then, checked data will be compared to EEPROM using SPI protocol. SPI protocol is chosen over I<sup>2</sup>C on addressing EEPROM because it provides faster data rate up to 100M/bit which I<sup>2</sup>C is limited to 10M/bit.

Firmware development on EEPROM was important as access control system accommodates thousands of users and transactions, which requires large permanent memory system. Verification process on user ID occurs after interrupt service routine filled up user ID register and compare it with the data stored in EEPROM using paging method. Paging method is a virtual tables labeled by value from 0 to 9. MSB



of user ID will be stored according to the label of virtual tables in EEPROM. Similar process occurs on add or delete user function. Authorized staff can perform such operation via computer, entering string of integer on the form prepared in HTML display. String of data will be converted into integer value and stored in register. Through stack application, the data will be transmitted to development board. Depending on the requested operation:

- Add function – user ID register will be stored in the assigned location
- Delete function – user ID register will be compare to data stored in memory location, then clear stored data once it has the same data as user ID register.

#### *4.2.2.3 HTML user interface customized for access control*

HTML on computer browser is generated from MPFS binary image files stored in EEPROM on board. When hardware board is start up, the binary image file will be fetched from memory location and then posted up to PC. Then input and output parameters such as LED indications, user transaction and user ID add or delete functions will be updated every millisecond. HTML on PC will be responsible on user data entry, which will store details of user on EEPROM and project the transaction time on PC. A blank row was made for user to key in user ID on EM card. Then the string of data is converted into integer value and stored in two register, user ID Low and user ID High, as a register can only store 16 bits of data. Through stack application, the data will be transmitted to development board and store in EEPROM using paging method. Vice versa, to delete a user entry, string of data from computer's HTML input will be converted into integer value, stored into a register and then locates the same data register in EEPROM and clear the value.



## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATION**

#### **5.1 Conclusion**

In conclusion, wireless access control system provides flexibility to users for its data retrieval with Wi-Fi services at any corner around the world and for its larger coverage area for access control system installation. Microchip TCP/IP Stack module and wireless module are the main components in this system as the stack module involves in transmission protocol and wireless module is to transmit data wirelessly to data terminal. During the progress of this project, a few problems had been encountered such as electronics components shortage and failure, firmware algorithm design had met bottleneck, designed hardware did not fit the features and applications require. However, these problems had been overcome and this project has completed and the objectives were fully archived. An operational access control system based on Microchip Explore 16 development board has been designed, modified, tested, debugged and verified, with the maximum distance of 25m of it from access point. It is hoped that this project will bring forth notable resource to the future.

#### **5.2 Recommendation**

This project can be enhanced in the future based on the following suggestion:

- i. Additional features on access control such as:
  - a. Real-time clock operation
  - b. Authorized user ID at specific time slot
- ii. Increase the distance limit from controller hardware to computer by adding a few more access points in between

## REFERENCES

- [1] Il-Kyu Hwang, Jin-Wook Baek. (2007, November). Wireless Access Monitoring and Control based on Digital Door Lock. *IEEE Transactions on Consumer Electronics*, Vol. 53, No. 4.
- [2] Douglas E.Comer. (2000). *Internetworking with TCP/IP Principles, Protocols, and Architectures*. Prentice Hall International, 4<sup>th</sup> Edition.
- [3] Liu, K.T.; Yang, C. H. (2008). Design and Implementation of Campus Gate Control System Based on RFID. *IEEE Asia Pacific Conference*. National Kaoshiung Normal University.
- [4] Chang, F. C.; Huang, H. C; Hang, H. M. (2007). Layered Access Control Scheme on Watermarked Scalable Media. Dept. of Electronics Engineering, National Chiao Tung University.
- [5] Kou, C. Y.; Springsteel, F.; (1999). The Security Mechanism in the World Wide Web (WWW) and the Common Gateway Interface (CGI) Example of Central Police University Entrance Examination System. Central Police University.
- [6] Lester LaPierre. (2009, June 2). *Wireless Access Control and Security Demystified*. Retrived August 10, 2009, from Security Info Watch: <http://www.securityinfowatch.com/root+level/1279325?pageNum=3>
- [7] Lionel Silverman. (2009, April 2). *Where is Wireless Access Control?* Retrieved August 10, 2009, from Security Info Watch: <http://www.securityinfowatch.com/root+level/1295745>
- [8] Laura Taylor. (2003, October 11). *Access Control 101*. Retrieved August 10, 2009, from Intranet Journal: [http://www.intranetjournal.com/articles/200311/ij\\_11\\_10\\_03a.html](http://www.intranetjournal.com/articles/200311/ij_11_10_03a.html)
- [9] Andy Geremia. (2008, June 1). *Wireless Access Control Design 101*. Retrieved August 10, 2009, from Article Archive: <http://www.articlearchives.com/media-telecommunications/telecommunications/959972-1.html>
- [10] Mark Lesswing. (2006, July 13). *Access Control - You Practice It Everyday*. Retrieved August 10, 2009 from Realtor Secure:

[http://www.realtor.org/wps/wcm/connect/5d81b80048a28cafadcfe0c8bc1f2ed/SOS\\_Wk4\\_Access.pdf?MOD=AJPERES&CACHEID=5d81b80048a28cafadcfe0c8bc1f2ed](http://www.realtor.org/wps/wcm/connect/5d81b80048a28cafadcfe0c8bc1f2ed/SOS_Wk4_Access.pdf?MOD=AJPERES&CACHEID=5d81b80048a28cafadcfe0c8bc1f2ed)

[11] *Wi-Fi and Wireless Network*. Retrieved August 11, 2009 from Wifinotes: <http://www.wifinotes.com/>

[12] *Wi-Fi*. Retrieved August 11, 2009 from Wikipedia: <http://en.wikipedia.org/wiki/Wi-Fi>

[10] Douglas E. Comer. (2000). *Internetworking with TCP/IP Principles, Protocols, and Architectures*. Prentice Hall International, 4<sup>th</sup> Edition.

[13] H. Gilbert. (1995). *Introduction to TCP/IP*. Retrieved August 12, 2009 from PC and Lube: <http://www.yale.edu/pclt/COMM/TCPIP.HTM>

[14] *TCP/IP*. Retrieved August 12, 2009 from SearchNetworking: [http://searchnetworking.techtarget.com/sDefinition/0,,sid7\\_gci214173,00.html](http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci214173,00.html)

[14] *ZeroG Wireless Technology*. Retrieved August 12, 2009 from ZeroG wireless technology: <http://www.zerogwireless.com/technology/techchips.html>

[15] *Wi-Fi*. Retrieved September 10, 2009 from webopedia: [http://www.webopedia.com/TERM/W/Wi\\_Fi.html](http://www.webopedia.com/TERM/W/Wi_Fi.html)





## APPENDICES

# APPENDIX A

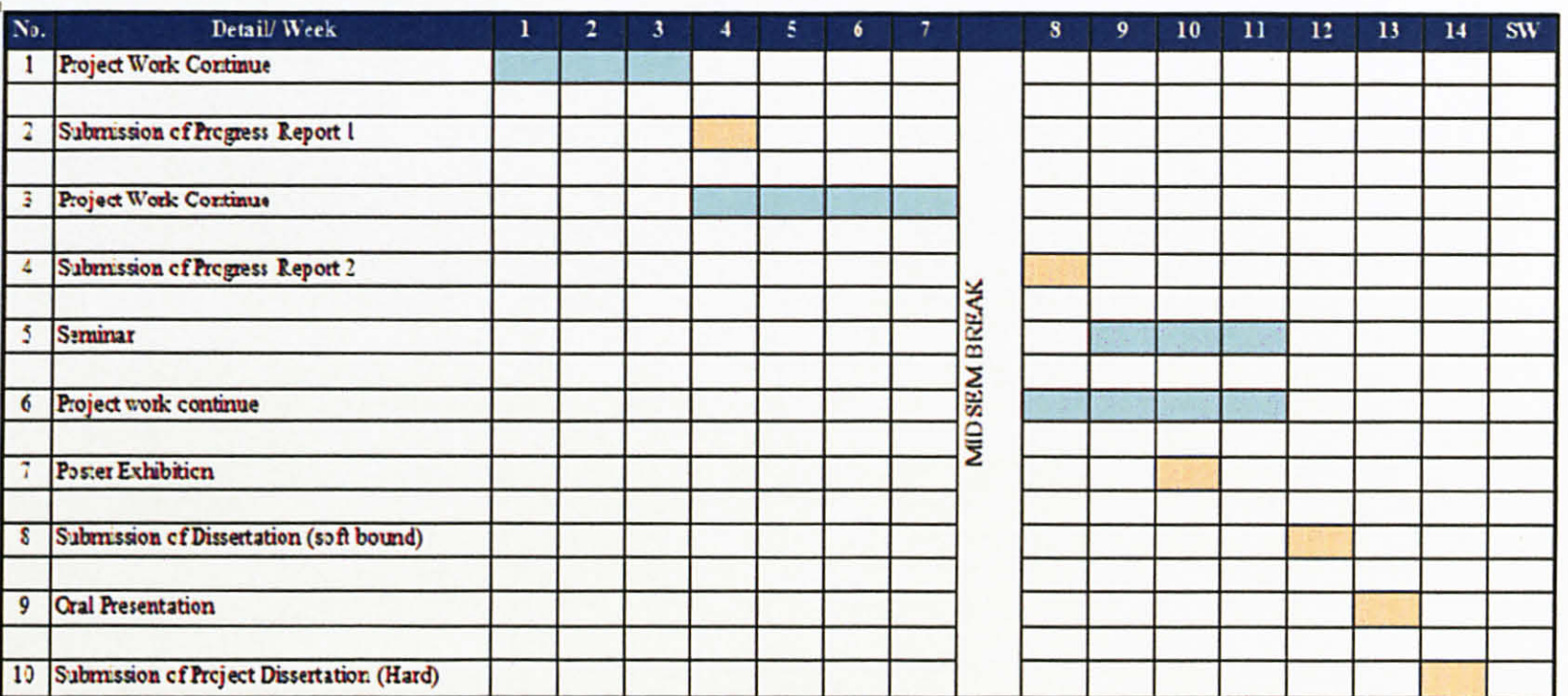
## GANTT CHART FOR FINAL YEAR PROJECT



No.	Detail/ Week	1	2	3	4	5	6	7	8	9	10		11	12	13	14	SW
1	Selection of Project Topic											MID SEM BREAK					
2	Preliminary Research Work																
3	Submission of Preliminary Report																
4	Project Work																
5	Submission of Progress Report																
6	Seminar																
7	Project work continues																
8	Submission of Interim Report Final Draft																
9	Submission of Interim Report																
10	Oral Presentation																

 Suggested milestone  
 Process

# APPENDIX B

## GANNT CHART FOR FINAL YEAR PROJECT II



 Suggested Milestone  
 Process



# APPENDIX C

## ZEROG WI-FI MODULE

### zeroG

CONNECTING THE INTERNET OF THINGS

#### Product Overview

ZeroG Wi-Fi Module

##### MODULE SPECIFICATIONS

Wireless Standard	IEEE 802.11b
Data Rate	1 & 2 Mbps
Wireless Security	WEP, WPA Personal, WPA2 Personal
Host Interface	SPI
Supply Voltage	2.7V - 3.6V
Operating Temperature	Commercial 0 - 70 °C
Module Dimensions	27mm x 31mm
Certification and Compliance	FCC, CE, ETSI, Japan, WEEE, RoHS, REACH

##### Ordering

**Part Numbers**

ZG2100: Module with integrated PCB antenna

ZG2101: Module with external PCB antenna

ZG2102: Module with external PCB antenna

ZG2103: Module with external PCB antenna

**Module Weight**

0.1g (0.0035 oz)

**Footprint**

27mm x 31mm

### zeroG

CONNECTING THE INTERNET OF THINGS

#### Product Overview

ZeroG Wi-Fi Module

#### The Fourth Age of Wireless

The ZG2100 Wi-Fi module allows almost any application to leverage the massive and rapidly growing Wi-Fi infrastructure. ZeroG understands the needs of embedded system designers and has optimized the ZG2100 for easy adoption in embedded systems.

The ZG2100 is available as a ZG2100M or a ZG2101M module. The ZG2100M contains an on-board PCB antenna and the ZG2101M module is designed to use an external antenna for extended range.

This highly integrated, low-power ZG2100 is designed from the ground up to ensure simple and low-cost connectivity regardless of host processor. The result is faster time to market and a significant reduction in bill of materials.

**LOW**  
Design Effort

**LOW**  
Power Consumption

**LOW**  
System Requirements

**LOW**  
System Cost

**ZeroG ZG2100 Wi-Fi Module**

8-bit host MCU

16-bit host MCU

32-bit host MCU

**Launch Wi-Fi with minimum investment**

ZeroG applications can be designed across three platforms and with lower engineering resources than other Wi-Fi solutions in the market.

**Leverage a massive Wi-Fi infrastructure**

By choosing Wi-Fi, customers can leverage a massive and growing infrastructure. Wi-Fi enables simple connectivity to the Internet, ensuring our team and resources remain to add functionality, service, and flexibility to new and existing products.

<http://www.zerogwireless.com/>

© Copyright 2008 ZeroG Wireless. All Rights Reserved.

<http://www.zerogwireless.com/>

© Copyright 2008 ZeroG Wireless. All Rights Reserved.

### zeroG

CONNECTING THE INTERNET OF THINGS

#### Product Overview

ZeroG Wi-Fi Module

##### LOW Design Effort

- Bring up a Wi-Fi prototype in 48 hours or a few hours
- Tightly integrated with major microcontroller providers and ideal for embedded applications
- Easy-to-use software suite contains APIs to support infrastructure and ad-hoc modes
- On-chip includes a comprehensive suite of commands that allow to easily turn on/off or go to sleep

##### LOW Power Consumption

- Battery life lasts as long as 10 years for "on-the-fly" wake-up
- Chip automatically goes on standby in between receiving and transmitting packets
- No host polling allows the microcontroller to shut down under ZG2100 operation as well
- Battery operating range 2.7-3.6V
- 250uA sleep mode with fast wake up, hibernate at 0.1uA

##### LOW System Requirements

- Optimized for low-endurance host processors and systems
- On-chip 802.11 MAC and WEP/WPA/WPA2 hardware accelerators reduce memory footprint and minimize host processor cycles
- Host RAM requirement is as low as 2.0KB
- On-chip memory footprint is as low as 30 bytes of RAM from a host microcontroller, and as low as 19KB of ROM

##### LOW System Cost

- Fully integrated with a wide range of microcontrollers, low-cost SoC or 32-bit microcontrollers
- No additional drivers required to integrate Wi-Fi
- Certified for regulatory and industry compliance with on-board and 13 external antennas (FCC, CE, ETSI, Japan, WEEE, RoHS, REACH)

### zeroG

CONNECTING THE INTERNET OF THINGS

#### Product Overview

ZeroG Wi-Fi Module

#### ZG2100M/ZG2101M Block Diagram

##### Utility and Smart Energy

- Configure and control a thermostat
- Monitor and update storage conditions
- Remotely reconnect during power outage
- Debug and analyze utility meters

##### Retail

- Manage assets remotely
- Notify inventory shortage
- Bill and deliver inventory automatically

##### Industrial controls

- Monitor traffic conditions with wireless cameras
- Update digital messaging in real time
- Detect and alert of intrusions using biometrics

##### Consumer Electronics

- Store and access media away from home
- Capture and store images at remote locations
- Control toys wirelessly

##### Healthcare and Fitness

- Maintain and access medical devices
- Collect and update health records
- Notify and record patient's test results

##### Remote Device Management

- Update advertisements in real time
- Configure and update data to multiple locations
- Track and manage assets

##### A World of Mini Webservers

The ZG2100Wi-Fi module makes it possible for low-cost webservers to exist in nearly any embedded device

<http://www.zerogwireless.com/>

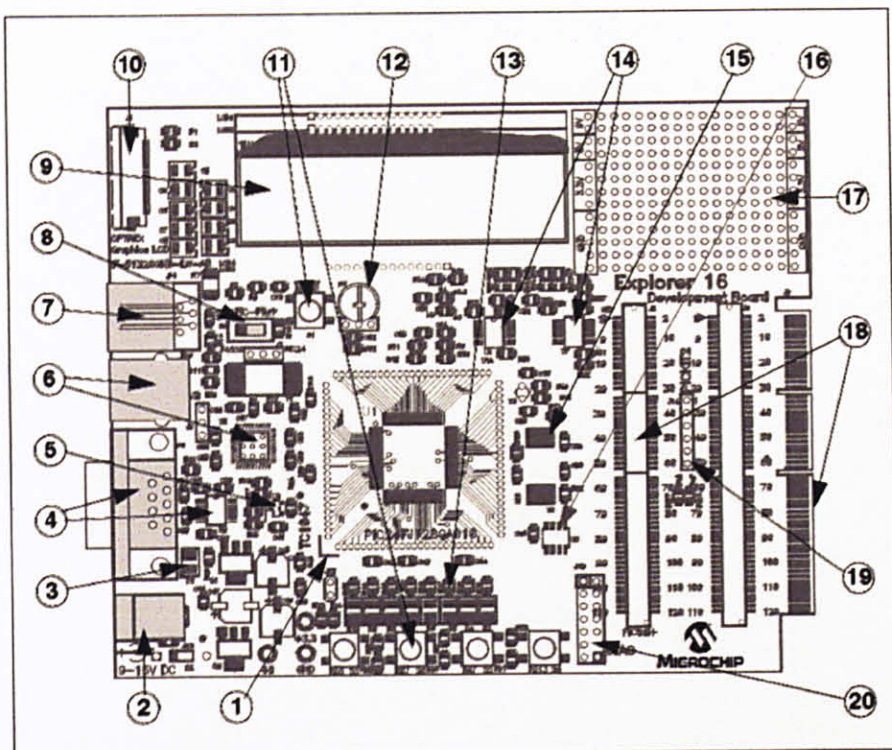
© Copyright 2008 ZeroG Wireless. All Rights Reserved.

<http://www.zerogwireless.com/>

© Copyright 2008 ZeroG Wireless. All Rights Reserved.

## APPENDIX D

### MICROCHIP DEVELOPMENT BOARD



#### EXPLORER 16 DEVELOPMENT BOARD FUNCTIONALITY AND FEATURES

A layout of the Explorer 16 Development Board is shown in Figure 1-1. The board includes these key features, as indicated in the diagram:

1. 100-pin PIM riser, compatible with the PIM versions of all Microchip PIC24F/24H/dsPIC33F devices
2. Direct 9 VDC power input that provides +3.3V and +5V (regulated) to the entire board
3. Power indicator LED
4. RS-232 serial port and associated hardware
5. On-board analog thermal sensor
6. USB connectivity for communications and device programming/debugging
7. Standard 6-wire In-Circuit Debugger (ICD) connector for connections to an MPLAB ICD 2 programmer/debugger module
8. Hardware selection of PIM or soldered on-board microcontroller (in future versions)
9. 2-line by 16-character LCD
10. Provisioning on PCB for add on graphic LCD
11. Push button switches for device Reset and user-defined inputs
12. Potentiometer for analog input
13. Eight indicator LEDs
14. 74HCT4053 multiplexers for selectable crossover configuration on serial communication lines
15. Serial EEPROM
16. Independent crystals for precision microcontroller clocking (8 MHz) and RTCC operation (32.768 kHz)
17. Prototype area for developing custom applications
18. Socket and edge connector for PICtail™ Plus card compatibility
19. Six-pin interface for PICkit 2 Programmer
20. JTAG connector pad for optional boundary scan functionality

*Both picture and descriptions are taken from Microchip Explorer 16 Development Board User's Guide page 12 and page 13.*



## APPENDIX E

### FIRMWARE SOURCE CODE

```
#define THIS_IS_STACK_APPLICATION

// Include all headers for any enabled TCPIP Stack functions
#include "TCPIP Stack/TCPIP.h"

// Include functions specific to this stack application
#include "MainDemo.h"

// Declare AppConfig structure and some other supporting stack variables
APP_CONFIG AppConfig;
BYTE AN0String[8];

// Use UART2 instead of UART1 for stdout (printf functions). Explorer 16
// serial port hardware is on PIC UART2 module.
#if defined(EXPLORER_16)
    int __C30_UART = 2;
#endif

static void InitAppConfig(void);
static void InitializeBoard(void);
static void ProcessIO(void);
void Change(void);

defined(__C30__)

#define UART2PrintString    putsUART

void __attribute__((interrupt, auto_psv)) _DefaultInterrupt(void)
{
    UART2PrintString( "!!! Default interrupt handler !!!\r\n" );
    while (1)
    {
        Nop();
        Nop();
        Nop();
    }
}

void __attribute__((interrupt, auto_psv)) _OscillatorFail(void)
{
    UART2PrintString( "!!! Oscillator Fail interrupt handler !!!\r\n" );
    while (1)
    {
        Nop();
        Nop();
        Nop();
    }
}

void __attribute__((interrupt, auto_psv)) _AddressError(void)
```



```

{
    UART2PrintString( "!!! Address Error interrupt handler !!!\r\n" );
    while (1)
    {
        Nop();
        Nop();
        Nop();
    }
}

void __attribute__((interrupt, auto_psv)) _StackError(void)
{
    UART2PrintString( "!!! Stack Error interrupt handler !!!\r\n" );
    while (1)
    {
        Nop();
        Nop();
        Nop();
    }
}

void __attribute__((interrupt, auto_psv)) _MathError(void)
{
    UART2PrintString( "!!! Math Error interrupt handler !!!\r\n" );
    while (1)
    {
        Nop();
        Nop();
        Nop();
    }
}

}

#elif defined(__C32__)
    void _general_exception_handler(unsigned cause, unsigned status)
    {
        Nop();
        Nop();
    }
#endif

// Main application entry point.
#if defined(__18CXX)
void main(void)
#else
int main(void)
#endif
{
    static DWORD t = 0;
    static DWORD dwLastIP = 0;

    // Initialize application specific hardware
    InitializeBoard();

    #if defined(USE_LCD)
    // Initialize and display the stack version on the LCD
    LCDInit();
    DelayMs(100);
    strcpypgm2ram((char*)LCDText, "ZeroG Demo v2.1 "
                                                           "TCPStack " VERSION " ");
    LCDUpdate();
    #if defined( STACK_USE_UART )

```

```

    putsUART((ROM char*)"r\n\r\n");
putsUART( (char *)LCDText );
    putsUART((ROM char*)"r\n");
    #endif
    #endif

// Initialize stack-related hardware components that may be
// required by the UART configuration routines
TickInit();
    #if defined(STACK_USE_MPFS) || defined(STACK_USE_MPFS2)
    MPFSInit();
    #endif

// Initialize Stack and application related NV variables into AppConfig.
InitAppConfig();

// Initiates board setup process if button is depressed
// on startup
if(BUTTON0_IO == 0u)
{
    #if defined(EEPROM_CS_TRIS) || defined(SPIFLASH_CS_TRIS)
    // Invalidate the EEPROM contents if BUTTON0 is held down for more than 4 seconds
    DWORD StartTime = TickGet();
    LED_PUT(0x00);

    while(BUTTON0_IO == 0u)
    {
        if(TickGet() - StartTime > 4*TICK_SECOND)
        {
            #if defined(EEPROM_CS_TRIS)
            XEEBeginWrite(0x0000);
            XEEWrite(0xFF);
            XEEEndWrite();
            #elif defined(SPIFLASH_CS_TRIS)
            SPIFlashBeginWrite(0x0000);
            SPIFlashWrite(0xFF);
            #endif

            #if defined(STACK_USE_UART)
            putsUART("r\n\r\nBUTTON0 held for more than 4 seconds.
Default settings restored.r\n\r\n");
            #endif

            LED_PUT(0x0F);
            while((LONG)(TickGet() - StartTime) <= (LONG)(9*TICK_SECOND/2));
            LED_PUT(0x00);
            while(BUTTON0_IO == 0u);
            Reset();
            break;
        }
    }
    #endif

    #if defined(STACK_USE_UART)
    DoUARTConfig();
    #endif
}

// Initialize core stack layers (MAC, ARP, TCP, UDP) and
// application modules (HTTP, SNMP, etc.)

```

```

StackInit();

// Blink LED0 (right most one) every second.
if(TickGet() - t >= TICK_SECOND/2u1)
{
    t = TickGet();
    LED0_IO ^= 1;
}

// This task performs normal stack task including checking
// for incoming packet, type of packet and calling
// appropriate stack entity to process it.
StackTask();

// This tasks invokes each of the core stack application tasks
StackApplications();

    // Process application specific tasks here.

    #if defined(STACK_USE_GENERIC_TCP_CLIENT_EXAMPLE)
    GenericTCPClient();
    #endif

    #if defined(STACK_USE_GENERIC_TCP_SERVER_EXAMPLE)
    GenericTCPServer();
    #endif

    #if defined(STACK_USE_SMTP_CLIENT)
    SMTPDemo();
    #endif

    #if defined(STACK_USE_ICMP_CLIENT)
    PingDemo();
    #endif

    #if defined(STACK_USE_SNMP_SERVER) && !defined(SNMP_TRAP_DISABLED)
    SNMPTrapDemo();
    if(gSendTrapFlag)
        SNMPSendTrap();
    #endif

    #if defined(STACK_USE_BERKELEY_API)
    BerkeleyTCPClientDemo();
    BerkeleyTCPServerDemo();
    BerkeleyUDPClientDemo();
    #endif

    ProcessIO();

}

}

// Writes an IP address to the LCD display and the UART as available
void DisplayIPValue(IP_ADDR IPVal)
{
    //      printf("%u.%u.%u.%u", IPVal.v[0], IPVal.v[1], IPVal.v[2], IPVal.v[3]);
    BYTE IPDigit[4];
    BYTE i;
#ifdef USE_LCD
    BYTE j;

```



```

    BYTE LCDPos=16;

#endif

    for(i = 0; i < sizeof(IP_ADDR); i++)
    {
        uitoa((WORD)IPVal.v[i], IPDigit);

        #if defined(STACK_USE_UART)
            putsUART(IPDigit);
        #endif

        #ifdef USE_LCD
            for(j = 0; j < strlen((char*)IPDigit); j++)
            {
                LCDText[LCDPos++] = IPDigit[j];
            }
            if(i == sizeof(IP_ADDR)-1)
                break;
            LCDText[LCDPos++] = '.';
        #else
            if(i == sizeof(IP_ADDR)-1)
                break;
        #endif

        #if defined(STACK_USE_UART)
            while(BusyUART());
            WriteUART('.');
        #endif
    }

    #ifdef USE_LCD
        if(LCDPos < 32u)
            LCDText[LCDPos] = 0;
        LCDUpdate();
    #endif
}

// Processes A/D data from the potentiometer
static void ProcessIO(void)
{
    #if defined(__C30__) || defined(__C32__)
        // Convert potentiometer result into ASCII string
        uitoa((WORD)ADC1BUF0, AN0String);
    #else
        // AN0 should already be set up as an analog input
        ADCON0bits.GO = 1;

        // Wait until A/D conversion is done
        while(ADCON0bits.GO);

        // AD converter errata work around (ex: PIC18F87J10 A2)
        #if !defined(__18F87J50)    &&    !defined(__18F87J50)    &&    !defined(__18F87J11)
        && !defined(__18F87J11)
            PRODL = ADCON2;
            ADCON2 |= 0x7;    // Select Frc mode by setting ADCS0/ADCS1/ADCS2
            ADCON2 = PRODL;
        #endif

        // Convert 10-bit value into ASCII string
        uitoa(*((WORD*)(%ADRESL)), AN0String);
    }
}

```

```

#endif
}

/*****
Function:
    static void InitializeBoard(void)
*****/
static void InitializeBoard(void)
{
    // LEDs
    LED0_TRIS = 0;
    LED1_TRIS = 0;
    LED2_TRIS = 0;
    LED3_TRIS = 0;
    LED4_TRIS = 0;
    LED5_TRIS = 0;
    LED6_TRIS = 0;
    #if !defined(EXPLORER_16) // Pin multiplexed with a button on EXPLORER_16
        LED7_TRIS = 0;
    #endif
    LED_PUT(0x00);

    #if defined(__18CXX)
        // Enable 4x/5x/96MHz PLL on PIC18F87J10, PIC18F97J60, PIC18F87J50, etc.
        OSCTUNE = 0x40;

        // Set up analog features of PORTA

        // PICDEM.net 2 board has POT on AN2, Temp Sensor on AN3
        #if defined(PICDEMNET2)
            ADCON0 = 0x09;           // ADON, Channel 2
            ADCON1 = 0x0B;           // Vdd/Vss is +/-REF, AN0, AN1, AN2, AN3 are analog
        #elif defined(PICDEM2)
            ADCON0 = 0x81;           // ADON, Channel 0, Fosc/32
            ADCON1 = 0x0F;           // Vdd/Vss is +/-REF, AN0, AN1, AN2, AN3 are all
digital
            #elif defined(__18F87J11) || defined(__18F87J11) || defined(__18F87J50) ||
defined(__18F87J50)
                ADCON0 = 0x01;           // ADON, Channel 0, Vdd/Vss is +/-REF
                WDTCNbits.ADSHR = 1;
                ANCON0 = 0xFC;           // AN0 (POT) and AN1 (temp sensor) are analog
                ANCON1 = 0xFF;
                WDTCNbits.ADSHR = 0;
            #else
                ADCON0 = 0x01;           // ADON, Channel 0
                ADCON1 = 0x0E;           // Vdd/Vss is +/-REF, AN0 is analog
            #endif
            ADCON2 = 0xBE;           // Right justify, 20TAD ACQ time, Fosc/64 (~21.0kHz)

        // Enable internal PORTB pull-ups
        INTCON2bits.RBPU = 0;

        // Configure USART
        TXSTA = 0x20;
        RCSTA = 0x90;

        // See if we can use the high baud rate setting
        #if ((GetPeripheralClock()+2*BAUD_RATE)/BAUD_RATE/4 - 1) <= 255
            SPBRG = (GetPeripheralClock()+2*BAUD_RATE)/BAUD_RATE/4 - 1;

```

```

        TXSTAbits.BRGH = 1;
#else    // Use the low baud rate setting
        SPBRG = (GetPeripheralClock()+8*BAUD_RATE)/BAUD_RATE/16 - 1;
#endif

// Enable Interrupts
RCONbits.IPEN = 1;                // Enable interrupt priorities
INTCONbits.GIEH = 1;
INTCONbits.GIEL = 1;

#else    // 16-bit C30 and and 32-bit C32

    #if defined(__dsPIC33F__) || defined(__PIC24H__)
        // Crank up the core frequency
        PLLFBD = 38;                // Multiply by 40 for 160MHz VCO
output (8MHz XT oscillator)
        CLKDIV = 0x0000;            // FRC: divide by 2, PLLPOST: divide by 2,
        PLLPRE: divide by 2

        // Port I/O
        AD1PCFGHbits.PCFG23 = 1; // Make RA7 (BUTTON1) a digital input
        AD1PCFGHbits.PCFG20 = 1; // Make RA12 (INT1) a digital input for ZeroG ZG2100M
        PICTail Plus interrupt

        // ADC
        AD1CHS0 = 0;                // Input to AN0 (potentiometer)
        AD1PCFGLbits.PCFG5 = 0;     // Disable digital input on AN5 (potentiometer)
        AD1PCFGLbits.PCFG4 = 0;     // Disable digital input on AN4 (TC1047A temp
sensor)
    #else //defined(__PIC24F__) || defined(__PIC32MX__)
        #if defined(__PIC24F__)
            CLKDIVbits.RCDIV = 0;    // Set 1:1 8MHz FRC postscalar
        #endif

        // ADC
        AD1CHS = 0;                // Input to AN0 (potentiometer)
        AD1PCFGbits.PCFG4 = 0;     // Disable digital input on AN4 (TC1047A temp
sensor)

        #if defined(__32MX460F512L__) || defined(__32MX795F512L__) // PIC32MX460F512L
and PIC32MX795F512L PIMs has different pinout to accomodate USB module
            AD1PCFGbits.PCFG2 = 0;    // Disable digital input on AN2
(potentiometer)
        #else
            AD1PCFGbits.PCFG5 = 0;    // Disable digital input on AN5
(potentiometer)
        #endif
    #endif

    // ADC
    AD1CON1 = 0x84E4;              // Turn on, auto sample start, auto-convert, 12 bit
mode (on parts with a 12bit A/D)
    AD1CON2 = 0x0404;              // AVdd, AVss, int every 2 conversions, MUXA only, scan
    AD1CON3 = 0x1003;              // 16 Tad auto-sample, Tad = 3*Tcy
    #if defined(__32MX460F512L__) || defined(__32MX795F512L__) // PIC32MX460F512L and
PIC32MX795F512L PIMs has different pinout to accomodate USB module
        AD1CSSL = 1<<2;            // Scan pot
    #else

```



```

        AD1CSSL = 1<<5;                                // Scan pot
    #endif

    // UART
    #if defined(STACK_USE_UART)
        UARTTX_TRIS = 0;
        UARTRX_TRIS = 1;
        UMODE = 0x8000;                                // Set UARTEN. Note: this must be done before
setting UTXEN

        #if defined(__C30__)
            USTA = 0x0400;                                // UTXEN set
            #define CLOSEST_UBRG_VALUE
            ((GetPeripheralClock()+8ul*BAUD_RATE)/16/BAUD_RATE-1)
            #define BAUD_ACTUAL (GetPeripheralClock()/16/(CLOSEST_UBRG_VALUE+1))
        #else //defined(__C32__)
            USTA = 0x00001400;                            // RXEN set, TXEN set
            #define CLOSEST_UBRG_VALUE
            ((GetPeripheralClock()+8ul*BAUD_RATE)/16/BAUD_RATE-1)
            #define BAUD_ACTUAL (GetPeripheralClock()/16/(CLOSEST_UBRG_VALUE+1))
        #endif

        #define BAUD_ERROR ((BAUD_ACTUAL > BAUD_RATE) ? BAUD_ACTUAL-BAUD_RATE :
BAUD_RATE-BAUD_ACTUAL)
        #define BAUD_ERROR_PRECENT ((BAUD_ERROR*100+BAUD_RATE/2)/BAUD_RATE)
        #if (BAUD_ERROR_PRECENT > 3)
            #warning UART frequency error is worse than 3%
        #elif (BAUD_ERROR_PRECENT > 2)
            #warning UART frequency error is worse than 2%
        #endif

        UBRG = CLOSEST_UBRG_VALUE;
    #endif

#endif

#if defined(SPIRAM_CS_TRIS)
    SPIRAMInit();
#endif

#if defined(EEPROM_CS_TRIS)
    XEEInit();
#endif

#if defined(SPIFLASH_CS_TRIS)
    SPIFlashInit();
#endif
}

/*****
 * Function:      void InitAppConfig(void)
*****/
static ROM BYTE SerializedMACAddress[6] = {MY_DEFAULT_MAC_BYTE1, MY_DEFAULT_MAC_BYTE2,
MY_DEFAULT_MAC_BYTE3, MY_DEFAULT_MAC_BYTE4, MY_DEFAULT_MAC_BYTE5, MY_DEFAULT_MAC_BYTE6};
//#pragma romdata

static void InitAppConfig(void)
{
    AppConfig.Flags.bIsDHCPEnabled = TRUE;
    AppConfig.Flags.bInConfigMode = TRUE;
    memcpypgm2ram((void*)&AppConfig.MyMACAddr, (ROM void*)SerializedMACAddress,
sizeof(AppConfig.MyMACAddr));
}

```

```

    AppConfig.MyIPAddr.Val = MY_DEFAULT_IP_ADDR_BYTE1 | MY_DEFAULT_IP_ADDR_BYTE2<<8ul |
MY_DEFAULT_IP_ADDR_BYTE3<<16ul | MY_DEFAULT_IP_ADDR_BYTE4<<24ul;
    AppConfig.DefaultIPAddr.Val = AppConfig.MyIPAddr.Val;
    AppConfig.MyMask.Val = MY_DEFAULT_MASK_BYTE1 | MY_DEFAULT_MASK_BYTE2<<8ul |
MY_DEFAULT_MASK_BYTE3<<16ul | MY_DEFAULT_MASK_BYTE4<<24ul;
    AppConfig.DefaultMask.Val = AppConfig.MyMask.Val;
    AppConfig.MyGateway.Val = MY_DEFAULT_GATE_BYTE1 | MY_DEFAULT_GATE_BYTE2<<8ul |
MY_DEFAULT_GATE_BYTE3<<16ul | MY_DEFAULT_GATE_BYTE4<<24ul;

    AppConfig.PrimaryDNSServer.Val = MY_DEFAULT_PRIMARY_DNS_BYTE1 |
MY_DEFAULT_PRIMARY_DNS_BYTE2<<8ul | MY_DEFAULT_PRIMARY_DNS_BYTE3<<16ul |
MY_DEFAULT_PRIMARY_DNS_BYTE4<<24ul;
    AppConfig.SecondaryDNSServer.Val = MY_DEFAULT_SECONDARY_DNS_BYTE1 |
MY_DEFAULT_SECONDARY_DNS_BYTE2<<8ul | MY_DEFAULT_SECONDARY_DNS_BYTE3<<16ul |
MY_DEFAULT_SECONDARY_DNS_BYTE4<<24ul;

    // SNMP Community String configuration
    #if defined(STACK_USE_SNMP_SERVER)
    {
        BYTE i;
        static ROM char * ROM cReadCommunities[] = SNMP_READ_COMMUNITIES;
        static ROM char * ROM cWriteCommunities[] = SNMP_WRITE_COMMUNITIES;
        ROM char * strCommunity;

        for(i = 0; i < SNMP_MAX_COMMUNITY_SUPPORT; i++)
        {
            // Get a pointer to the next community string
            strCommunity = cReadCommunities[i];
            if(i >= sizeof(cReadCommunities)/sizeof(cReadCommunities[0]))
                strCommunity = "";

            // Ensure we don't buffer overflow. If your code gets stuck here,
            // it means your SNMP_COMMUNITY_MAX_LEN definition in TCPIPConfig.h
            // is either too small or one of your community string lengths
            // (SNMP_READ_COMMUNITIES) are too large. Fix either.
            if(strlenpgm(strCommunity) >= sizeof(AppConfig.readCommunity[0]))
                while(1);

            // Copy string into AppConfig
            strcpypgm2ram((char*)AppConfig.readCommunity[i], strCommunity);

            // Get a pointer to the next community string
            strCommunity = cWriteCommunities[i];
            if(i >= sizeof(cWriteCommunities)/sizeof(cWriteCommunities[0]))
                strCommunity = "";

            // Ensure we don't buffer overflow. If your code gets stuck here,
            // it means your SNMP_COMMUNITY_MAX_LEN definition in TCPIPConfig.h
            // is either too small or one of your community string lengths
            // (SNMP_WRITE_COMMUNITIES) are too large. Fix either.
            if(strlenpgm(strCommunity) >= sizeof(AppConfig.writeCommunity[0]))
                while(1);

            // Copy string into AppConfig
            strcpypgm2ram((char*)AppConfig.writeCommunity[i], strCommunity);
        }
    }
#endif

    // Load the default NetBIOS Host Name
    memcpypgm2ram(AppConfig.NetBIOSName, (ROM void*)MY_DEFAULT_HOST_NAME, 16);

```

```

FormatNetBIOSName(AppConfig.NetBIOSName);

#if defined(ZG_CS_TRIS)
    // Load the default SSID Name
    if (sizeof(MY_DEFAULT_SSID_NAME) > sizeof(AppConfig.MySSID))
    {
        ZGErrorHandler((ROM char *)"AppConfig.MySSID[] too small");
    }
    memcpypgm2ram(AppConfig.MySSID, (ROM void*)MY_DEFAULT_SSID_NAME,
sizeof(MY_DEFAULT_SSID_NAME));
#endif

#if defined(EEPROM_CS_TRIS)
{
    BYTE c;

    // When a record is saved, first byte is written as 0x61 to indicate
    // that a valid record was saved.

    XEEReadArray(0x0000, &c, 1);
    if(c == 0x61u)
        XEEReadArray(0x0001, (BYTE*)&AppConfig, sizeof(AppConfig));
    else
        SaveAppConfig();
}
#elif defined(SPIFLASH_CS_TRIS)
{
    BYTE c;

    SPIFlashReadArray(0x0000, &c, 1);
    if(c == 0x61u)
        SPIFlashReadArray(0x0001, (BYTE*)&AppConfig, sizeof(AppConfig));
    else
        SaveAppConfig();
}
#endif
}

#if defined(EEPROM_CS_TRIS) || defined(SPIFLASH_CS_TRIS)
void SaveAppConfig(void)
{
    // Ensure adequate space has been reserved in non-volatile storage to
    // store the entire AppConfig structure.
    #if defined(STACK_USE_MPFS) || defined(STACK_USE_MPFS2)
        if(sizeof(AppConfig) > MPFS_RESERVE_BLOCK)
            while(1);
    #endif

    #if defined(EEPROM_CS_TRIS)
        XEEBeginWrite(0x0000);
        XEEWrite(0x61);
        XEEWriteArray((BYTE*)&AppConfig, sizeof(AppConfig));
    #else
        SPIFlashBeginWrite(0x0000);
        SPIFlashWrite(0x61);
        SPIFlashWriteArray((BYTE*)&AppConfig, sizeof(AppConfig));
    #endif
}

```