

**DIGITAL SECURITY LOCKING SYSTEM –  
USB BASED GSM SECURITY SYSTEM**

**CHAI CHONG YI**

**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 2011**

## 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 : CHAI CHONG YI

ID No. : 07UEB06518

Date : 15 APRIL 2011

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**DIGITAL SECURITY LOCKING SYSTEM– USB BASED GSM SECURITY SYSTEM**” was prepared by **CHAI CHONG YI** 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: Dr. Yong Thian Khok

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.

© 2011, Chai Chong Yi. All right reserved.

Specially dedicated to  
my beloved family,  
supervisor Dr. Yong Thian Khok  
and partner Miss Chan Sieu Chen

## 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, Dr. Yong Thian Khok for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would like to express my gratitude to my loving parent and friends who had helped and given me encouragement during the development of the project as well as providing guidance on the project.

In addition, I would like to express my sincere thanks to my partner Miss Chan Sieu Chen for providing me guidance and help me in completing this project as well as providing me the opportunity to work as a team to complete a project.

Lastly, I would like to thanks UTAR for providing me an opportunity to complete a project as a part of fulfilment of the requirement for the Bachelor of Engineering Programme.

## **DIGITAL SECURITY LOCKING SYSTEM**

### **ABSTRACT**

Security threat has become an important issue to most of the people especially in urban area. With the increase in criminal rate, having a security system is desirable to ensure home is a safe place to live in. However, security alarm system may not be cheap depending on the features available on the security system and most of the security system can only provide local alarm which property owner cannot know what is happening to their property at the time emergency case happens. This problem has become the motivation of developing a low cost, low power, simplicity of operation, small and standalone security system. The developed project has to make use of mobile phone that use USB cable as GSM terminal instead of GSM modem which is not cost effective. Mobile phones that use USB cable can be found everywhere and some of the old model may potentially become GSM terminal to lower down the cost of the whole project. In the project, PIC24FJ64GB002 is used as the core processor of the project to act as a USB host as well as processing some task required by the security system. Mobile phone Sony Ericsson C902 is the main experimental GSM terminal in the system. A LCD and a 4x4 keypad are used to change setting for the security system. This system can then integrate with RFID based locking system to become a complete digital security locking system which is more useful and can be used in home, office or automotive.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>ii</b>
<b>APPROVAL FOR SUBMISSION</b>	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>vi</b>
<b>ABSTRACT</b>	<b>vii</b>
<b>TABLE OF CONTENTS</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF FIGURES</b>	<b>xii</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>	<b>xiv</b>
<b>LIST OF APPENDICES</b>	<b>xv</b>

### CHAPTER

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 Background and Motivation	1
	1.2 Aims and Objectives	3
	1.3 Outline of Thesis	3
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>4</b>
	2.1 Introduction	4
	2.2 Alarm System	4
	2.2.1 History of Alarm System	4
	2.2.2 Types of Alarm System	5
	2.3 USB Based Embedded System	6
	2.4 Short Message Service (SMS)	7
	2.4.1 History of SMS	7



	2.4.2	Advantages of Short Service Message	7
2.5		AT Command	8
	2.5.1	History of AT Command	8
	2.5.2	Types of AT Command	9
	2.5.3	AT Command Mode	10
2.6		Comparison and Critic	12
<b>3</b>		<b>METHODOLOGY</b>	<b>13</b>
	3.1	Introduction	13
	3.2	Overall System Setup	13
	3.3	USB Based GSM Alarm System Setup	15
	3.4	GSM terminal	16
	3.4.1	GSM/GPRS Modems	16
	3.4.2	Cellular Phones	17
	3.4.3	Choosing GSM terminal	17
	3.5	Modem Commands	18
	3.6	Sensor	18
	3.7	Microcontroller	19
	3.8	Matrix Keypad	21
	3.9	LCD Display	22
	3.10	MPLAB IDE	24
	3.11	EAGLE Layout Editor	25
	3.12	Schedule of FYP 2	25
<b>4</b>		<b>RESULTS AND DISCUSSIONS</b>	<b>26</b>
	4.1	Introduction	26
	4.2	Software Implementation	26
	4.2.1	Software Language	26
	4.2.2	USB	27
	4.2.3	Emulating Data EEPROM	37
	4.2.4	Decoding keypad	39
	4.2.5	Program on LCD Display	42
	4.3	Program overview	44

4.3.1	Option menu	46
4.3.2	Detailed Operation on Changing Password	50
4.3.3	Detailed Operation on Changing Target Phone Number	51
4.3.4	Detailed Operation on Changing SMSC Number	52
4.4	Results of Tested Possible Mobile Phone as GSM Terminal	53
4.5	Hardware implementation	54
4.5.1	Hardware Placement	54
4.5.2	Circuit Diagram	54
4.6	Problem Encounter	56
<b>5</b>	<b>CONCLUSION AND RECOMMENDATIONS</b>	<b>58</b>
5.1	Conclusion	58
5.2	Limitation and Future Enhancement	59
	<b>REFERENCES</b>	<b>60</b>
	<b>APPENDICES</b>	<b>61</b>

**LIST OF TABLES**

<b>TABLE</b>	<b>TITLE</b>	<b>PAGE</b>
Table 2.1:	PDU String Decoding	11
Table 3.1:	PIC24FJ64GB002 Specification	20
Table 3.2:	LCD Pin Table	23
Table 4.1:	USB Device Classes	29
Table 4.2:	Enumeration Steps	30
Table 4.3:	USB Files in Programming Part of the Project	31
Table 4.4:	Effect of USBHostTasks Calling Interval on USB Enumeration	32
Table 4.5:	USB Result Code	37
Table 4.6:	Data Memory Structure	38
Table 4.7:	User Data Storage Location	38
Table 4.8:	List of Option Menu	47
Table 4.9:	Detail Operation on Changing Password	50
Table 4.10:	Detail Operation on Changing Target Phone Number	51
Table 4.11:	Detail Operation on Changing SMSC Number	52
Table 4.12:	Test Result of Mobile Phones	53

## LIST OF FIGURES

<b>FIGURE</b>	<b>TITLE</b>	<b>PAGE</b>
Figure 2.1:	PDU Encoding Method	11
Figure 3.1:	Digital Security Locking System Setup	14
Figure 3.2:	Project Division of Digital Security Locking System	15
Figure 3.3:	USB Based GSM Alarm System Setup	16
Figure 3.4:	GSM Modem from Wavecom	17
Figure 3.5:	Snap Action Switch	19
Figure 3.6:	Magnetic Proximity Sensor	19
Figure 3.7:	PIC Microcontroller (SPDIP Package)	20
Figure 3.8:	Pin Diagram of PIC24FJ64BG002	21
Figure 3.9:	A 4x4 Matrix Keypad	22
Figure 3.10:	Internal Connection of a 4x4 Matrix Keypad	22
Figure 3.11:	A 2x16 LCD Display	23
Figure 3.12	Gantt Chart of FYP Part 2	25
Figure 4.1:	Example of TPL List	31
Figure 4.2:	Transfer Types of Different Endpoint	34
Figure 4.3:	Code to Test Transfer Types and Initiating Data Transfer	34
Figure 4.4:	Flow Chart of Testing USB Transfer Type and Initializing Data Transfer	35
Figure 4.5:	Generic Host Driver Write Function	36

Figure 4.7: Flow Chart of Decoding Keypad Part 1	40
Figure 4.8: Flow Chart of Decoding Keypad Part 2	41
Figure 4.9: LCD Initiation Process	42
Figure 4.10: Program Code for Function Set and Entry Mode Set of LCD	43
Figure 4.11: Program Code of Writing Data to LCD	44
Figure 4.12: Program Flow Chart of USB Based GSM Security System	45
Figure 4.13: LCD Display on (a) Start up, (b) Supported Terminal Attached, (c) Unsupported Terminal Attached, (d) Terminal Detach	46
Figure 4.14: Menu Flow Chat Part One	48
Figure 4.15: Menu Flow Chat Part Two	49
Figure 4.16: Circuit Diagram of Digital Security Locking System	55
Figure 4.17: PCB Layout of Digital Security Locking System	56

**LIST OF SYMBOLS / ABBREVIATIONS**

ASCII	American Standard Code for Information Interchange
AT	ATtention
CDC	Communications Device Class
CPU	Central Processing Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
HID	Human Interface Device
I/O	Input / Output
LCD	Liquid Crystal Display
LED	Light-emitting Diode
OTG	On-The-Go
PCB	Printed Circuit Board
PDU	Protocol Data Unit
PIC	Peripheral Interface Controller
PID	Product Identification
RAM	Random-access Memory
RFID	Radio Frequency Identification
SMS	Short Message Service
SMSC	Short Message Service Center
SPDIP	Skinny Dual In-line Package
SPI	Serial Peripheral Interface
TPL	Target Peripheral List
USB	Universal Serial Bus
VID	Vendor Identification
WLAN	Wireless Local Area Network

**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
	APPENDIX A: Main Program Code	61

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 Background and Motivation**

In this modern age, security has become an important issue for most of the people especially in the urban area. With the living standard of people improved, people tend to have more valuable property inside their house. Some people will try to rob or steal the property which may endanger the safety of property owner or people inside the house. To overcome the security threat, most people will install bunch of locks or alarm system.

Alarm system is useful in reducing the security threat at home regardless it is a high technology or a simple alarm system. Robbers and thieves will tend to make house with no alarm system as a target rather than those with alarm system. Even a dummy alarm system can scare off some of the thieves and robbers due to human nature where most people tend to stay out of trouble.

With alarm system, property owner do not have to always look after their property or having people guard at the property. This can reduce the time and cost on guarding property. Moreover, security system can guard the property all the time without have to worried getting tired or boring.

There are many types of alarm system in the market which utilize different types of sensor. Each types of sensor can detect different types of changes occur in



the surrounding and given out alert when the value is exceed the pre-set value. When combining few types of sensor, a place can me monitor very well.

Alarm system typically consists of sensor, alarm siren and relay. For home alarm system, alarm siren will sound whenever the sensor detects the door or window is open. This kind of security system can only provide local alarm and property can never know when emergency case happen. Some of the new alarm system includes a modem which will send alert to property owner when any security threat occurs. Property owner can know what is going on and taking any further action as soon as possible to minimize any loses.

When comes to security product in the market, the price for security product is usually high. Security manufacturer have to come out with new security technique form time to time to ensure their product has high level of security level. There are some low cost security systems which offer limited security. Thus, it is desirable to build a low cost, low power, simple operation, small and standalone digital security alarm system.

For digital security alarm system, the cost of fixed line security alarm system is fair enough. However, the prices for wireless security alert system are generally high in the market due to the cost of GSM/GPRS modem are high. Even the module of the GSM/GPRS modem is costly. There are many old models of cellular phones in the market. This cellular phone can be use as a low cost terminal to replace GSM/GPRS modem. This cellular phone can send alert to particular person when emergency case happened. By integrating these phones in the digital security alert system, the cost would be much more lowered. When integrate to the digital locking system, we can create a low cost digital security locking system.

## **1.2 Aims and Objectives**

The aims and objective for this project is to develop a security system which is based on GSM network and USB host terminal. The security system is able to send SMS to user when emergency case occurs. The cost of the developed project should be low cost, low power, simple operation, small and standalone. This project is then able to integrate with RFID based locking system to form digital security locking system.

## **1.3 Outline of Thesis**

This thesis starts with introduction of the digital security locking system and the focused part USB based GSM security system. In this chapter, the motivation as well as objective of the project development is stated.

Second chapter state the review on the currently available security system and possible method to develop a security system using GSM terminal. If possible, USB connection is used to connect between GSM terminal and microcontroller. Next chapter will be discussing on which potential component will be used in the project and how the project development progress is identified.

Chapter four is about how the project is developed and the results of each experiment are stated. After the project is developed, it will then combine with RFID based locking system to form a complete digital security locking system. The results of integration as well as problems encounter are stated in this chapter. The final chapter will be on the conclusion as well as future enhancement can be made on the project to develop a more powerful digital security locking system.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Introduction**

To understand better how a security system should work, a review on others people work is important to develop a better solution. In this chapter, history of alarm system as well as types of alarm system will be reviewed. GSM history as well as AT command is reviewed to develop a GSM based security system. Furthermore, working principle of USB and embedded host can provide cheaper solution for GSM based security system.

#### **2.2 Alarm System**

##### **2.2.1 History of Alarm System**

Long before the invention of electronic alarm system, people use animal such as dogs to alert them when someone is trying to steal the belonging from them or enter certain area. In the middle age, people uses large gong to alert people when there is invasion. Things started to change when Tildesley invented the first home alarm system, which contain a series of bells connecting together and link to the door (Gregory, 2007). When someone is open the door, the bells will ring and alerting the people inside. Soon after that, Edwin Homes invented first electro-mechanical alarm system (Gregory, 2007). The alarm system contains a spring connected to the switch

which attach to door and whenever the door is open, the circuit will closed and the bell will ring. This alarm system only uses simple electrical circuit such as switch, relay and bells. With the invention of integrated circuit, the size of alarm system has becomes smaller and the features has increased. The alarm system nowadays is no longer work only on switch, but also can work on infrared, ultrasonic, microwave, photoelectric and more. Some alarm system even has integrated circuit which can send alert to property owner when there is emergency case occurs (Gregory, 2007).

### **2.2.2 Types of Alarm System**

Alarm system can be categorized into two main types. First type is the alarm system that uses wired sensor nodes and another type uses wireless sensor nodes. Wired alarm system has hard wire connected between the sensor network and the main processing unit while wireless alarm system utilize wireless local area network (WLAN), wireless USB and Bluetooth for communication between sensor and the main processing unit.

Research done by Ahmed, A., Ali, J., Raza, A. & Abbas, G. (2007) shows that wireless sensor network has advantages over wired sensor network in the sense that the sensor can be placed at any terrain without have to worry about the wiring problem. However, most of the wireless sensors run on battery supply. Battery lifetime becomes the most deadly disadvantages to the wireless network especially the distance is long. Longer distance require stronger signal which also means higher power require to drive the signal. In another words, power efficiency of wireless sensor determines the operating lifetime of sensor network. Once battery is drained up, it can no longer provide any protection to the perimeter and are vulnerable to any attack.

On the other hand, wired sensor network does not suffer power source problem but it does has disadvantages where the quantity of sensors is limited due to the wiring problems. The network becomes complicated with the number of sensor build up. However, the transmission speed of wired network is fast when compare

with wireless network. Additionally, the wired sensor network can provide more reliable security over wireless sensor network as wired sensor network does not suffer signal lost where wireless sensor network did. Besides, wired sensor network is easier to setup compare with wireless sensor network (Ahmed, 2007).

Based on the survey done by Zhoa & Ye (2008) indicate the cost for wireless sensor network vary by technology used. Both WLAN and Bluetooth are high cost and high power consumption technology while wireless USB offer low on both cost and power consumption. To inform the user when any event occur, it can be done via internet or GSM network. In order for user get the information as soon as possible, GSM network is preferable as user will carry phone almost all the time compare to staying in front of computer all the time. According to research done by Huang, H. P., Xiao, S. S., Meng, X. Y. & Xiong, Y. (2010), GSM is a mature technology and hence, high security, wide coverage and can transmit over long distance.

### **2.3 USB Based Embedded System**

With the growth of USB technology, USB has been gradually replaces traditional serial protocol and parallel protocol. Due the ease of use, fast transfer rate and high stability, USB becomes a favourite choice of most people. USB can support control transfer, bulk transfer, interrupt transfer and isochronous transfer. Control transfer enable USB host read the configuration of device connected to it. Bulk transfer can transfer non-fixed amount of data with error correction while interrupt transfer is use to transfer small amount of data. For isochronous transfer, the data is transfer at a constant rate and no retransmission if error occurs (Kim, Y. S., Kim, H. S. & Lee, 2005). Most electronic appliances in the market nowadays uses USB interface.

## **2.4 Short Message Service (SMS)**

SMS is a technology which enables a short message to send and receive between mobile phone. SMS can send up to 140 bytes or equivalent to 1120 bits of data. If these 1120 bits is fully utilize, SMS can include 160 characters of 7-bits character encoding which is the English character and 70 characters of 16-bit Unicode used for many language. SMS is compatible for all GSM phones. GSM 07.05 is a standard for short message rules for SMS (Bodic, 2005).

### **2.4.1 History of SMS**

SMS was first appeared in 1992 in Europe. SMS is included in Global System for Mobile Communication (GSM). Both SMS and GSM standard was developed by European Telecommunication Standard Institute (ETSI). The development and maintenance of GSM and SMS standard was then turn to Third Generation Partnership Project (3GPP) (Bodic).

### **2.4.2 Advantages of Short Service Message**

According to the survey done by CTIA Wireless Association in year 2009, there are about 4.1 billion messages being sent daily. According to Bodic, the core reason why SMS is so successful is mainly due to its conveniences.

- i. **SMS can be sent or read at anytime**

When a SMS is received, the message will store into the memory. User can either read it immediately or read it later. Nowadays, most people have their mobile phone with them most of the time. SMS can be read or sent anytime regardless the user is in office, home, street or even in the bus.

ii. SMS is less disturbing than phone call

If user is on a phone call, every word the user says will immediately pass to another side of the phone. Sometimes, user may require some thinking before replying the question. Furthermore, SMS will not create load noise as talking on phone. This enable user to send or receive SMS while they are in library or theater without have to going outside. This enable users to keep in touch all the time without causing much disturbing.

iii. SMS can be sent to offline mobile phone

The benefits of SMS is if the destination phone is offline, the user can still receive the message after he or she is turn on the phone. Most of the time, the network signal may be weak on some area especially rural area. The message can still be receive by user if the user moves to a place with better network coverage.

## **2.5 AT Command**

AT Command is the modem command code and it is define in GSM 07.07 standard. GSM 07.07 provides AT Command which enable mobile platform to communicate with Data Terminal Equipment in serial communication (Cao, 2009).

### **2.5.1 History of AT Command**

In early 1980s, a company named Hayes has developed a modem called Hayes Smart modem 1200. Shortly after that, the company releases another modem call Hayes smart modem 2400. The model number simply represents the baud rate of the modem. They call the modem as smart modem because of the ability of the modem to auto-dial a number. Furthermore, this modem can send morse mode, work with Radioteletype (RTTY) and amateur radio repeaters. These two modems accept same programming command with only the operating speed is different. This has a great

advantage where the old driver can be use by new model of modem. Few modems makers has follow what Hayes did, where they develop modem that accept same command set as previous modem and called it “Hayes Compatible”. Soon after that, Hayes sued these modems makers for using the word “Hayes” in their product. Subsequently, these modem makers has change to “AT Command Set compatible”. AT command is taken from the words Attention, where the first two letters is used. AT command is the programming command that modem receive from the input and execute it. However, mobile usually do not implement full AT command and some of the command used in mobile phone may varied from manufacturer to manufacturer and model to model. Overall, the GSM/GPRS modem will have better support on AT command (Bodic).

### 2.5.2 Types of AT Command

There are two types of AT commands, which categorize as basic commands and extended commands. Basic commands are the commands that do not start with “+”. For example ATA stands for Answer, ATD stands for dial and ATH stands for hook control. On the other hand, extended command are the command that start with “+”. As for example, AT+CMGS will send the message and AT+CMSS will send message from storage. Following is some example of extended AT command (Developer's Home, 2004-2010).

+CSMS	Select message service
+CPMS	Preferred message storage
+CMGS	Message format
+CESP	Enter SIM block mode protocol
+CMS ERROR	Message failure code
+CSCA	Service center address
+CSMP	Set text mode parameter
+CSDS	Show text mode parameter
+CSCB	Select cell broadcast message type
+CSAS	Save setting



+CRES	Restore setting
+CNMI	New message indication to TE
+CMGL	List message
+CMRG	Read message
+CNMA	New message acknowledgement
+CMGS	Send message
+CMSS	Send message from storage
+CMGW	Write message to memory
+CMGD	Delete message
+CMGC	Send command
+CMMS	More message to send

### 2.5.3 AT Command Mode

Control of SMS function can be made via AT command in three mode which is Block mode, Text mode, and Protocol Data Unit (PDU) mode. AT stands for Attention. Block mode is a binary communication protocol including error protection. Text mode is a character based protocol suitable for high level software application while PDU mode is a character based protocol with hexadecimal-encoded binary transfer of command. PDU mode is usually for low level software driver that do not understand the content of command while text mode is suitable for high level programming language (Bodic).

PDU is harder for people to read it but it has advantages where it has lots more features compare to the simple text mode. Thus, PDU mode can support better on GSM modem and some of the mobile phone will only understand PDU mode. PDU mode has three encoding mode namely 7-bit code, 8-bit code, and 16-bit code. 7-bit and 8-bit code is use for normal ASCII code which is English message while 16-bit code is use for Unicode character which supports not only English but Chinese and other language as well. It is clear that to use other language in sending message, PDU mode in 16-bit code must be adopted since text mode does not support 16-bit mode (Cao, 2009).

When come to encoding in PDU mode, we need to know the meaning of each of the string. Since the syntax is fixed, the PDU string can easily created with the reference of the meaning for each words. The only tricky part of the PDU string is the semi-octets string. For example an international number +60123456789. To encode it into PDU string, we need to remove the “+” sign and add “F” to the end of the string. Then, each of the pair is twisted to convert to PDU string. +60123456789 will become 0621436587F9 (Wan, Zhang & Fang, 2008).

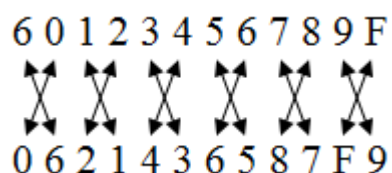


Figure 2.1: PDU Encoding Method

One example of PDU string contains the following information. 07910621000010F511000B910621436587F90000AA08C7F79B0C2287F3 will have the meaning as shown in the table below.

**Table 2.1: PDU String Decoding**

Octet(s)	Description	Example
07	Length of the SMSC information	7 octets
91	Type of address of SMSC	International format
0621000010F5	SMSC number	+60120000015
00	First Octets of SMS-DELIVER message	-
0B	Length of sender address	11
91	Type of address of the sender number	International format
0621436587F9	Sender number	+60123456789
00	Protocol identifier	-
00	Data encoding scheme	-
08	Length of User data	8
C7F79B0C2287F3	User data	Good Day

## **2.6 Comparison and Critic**

The security system purpose by Zhoa & Ye and Huang et al. has a disadvantage where it runs on battery supply. User will never know when the power of battery will drained up and causing the security system failed to work. Besides, they uses RS-232 serial interface which normally found on GSM modems and rarely on cellular phones. The system built by Kim et al. is based on USB which is widely used by cellular phones and most electronic appliances today.

As for the modem command, Cao and Wan, Zhang el al. recommend PDU mode as Chinese text is required and compatibility issue on GSM modem and most of the phones. Text mode is easier to apply but is hardware limited. PDU mode is better option since it can be supported by most of the phone and GSM modem.

## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Introduction**

To start the development of a project, the component needed must first be identified clearly. In this chapter, the list of the component required as well as technical specification of the component is discussed. The software needed for project development is also discussed in this chapter.

#### **3.2 Overall System Setup**

Figure 3.1 shows the core components of digital security locking system and how the components are related to each other. Direction of arrow represent flow of data and each block represent one component. Digital security locking system consist of seven core component which is PIC microcontroller, RFID reader, keypad, magnetic lock, alarm siren, door opening sensor, LCD display and GSM terminal.

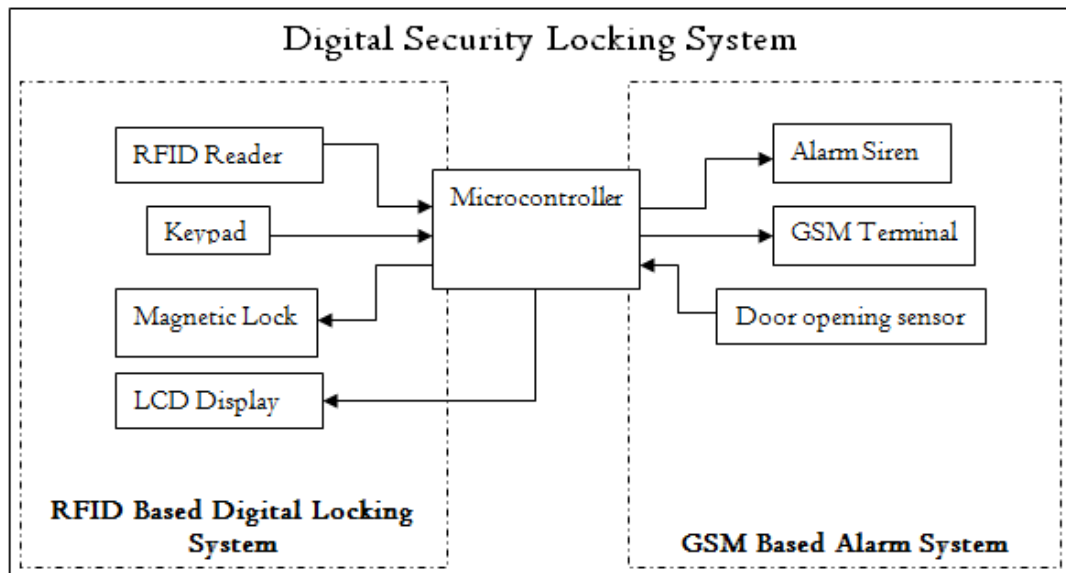


Figure 3.1: Digital Security Locking System Setup

The RFID reader will read a RFID tag and sent the data to the microcontroller. If the tag ID is in the access list, a signal will be sent to magnetic lock to release the lock. When there is no card present, microcontroller will keep on reading from keypad for password input. If the password matches the current password, lock will be release. User can choose to use either way to release lock or both of them depend on user setting. However, if the door is force to open without entering password or scanning tag, the microcontroller will send out a signal to the alarm and GSM terminal. The alarm will sound and GSM terminal will send a message to inform the property owner. Password is needed to turn off the alarm.

Digital security locking system consist two main parts which is the RFID based locking system and USB based GSM alarm system. These parts will be developed separately and combined at the final stage. After combination of both parts, optimization of project will be carry on. Figure 3.2 shows the project division of digital security locking system.

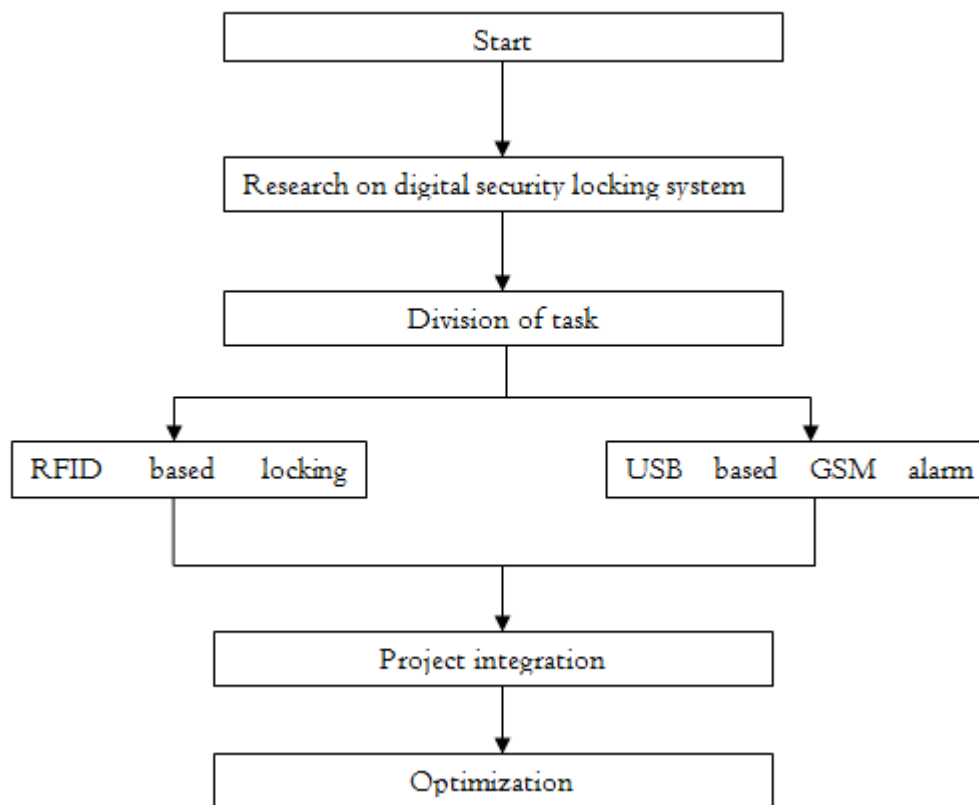


Figure 3.2: Project Division of Digital Security Locking System

### 3.3 USB Based GSM Alarm System Setup

USB based GSM alarm system consist of six core component which are door opening sensor, alarm siren, GSM terminal, LCD display, keypad and microcontroller. When the system is turn on, door opening sensor will keep on checking the door. If the door is open, microcontroller will send signal for the alarm siren to sound. At the same time, microcontroller will send a message to the property owner through GSM terminal. Figure 3.3 show the structure of USB based GSM alarm system.

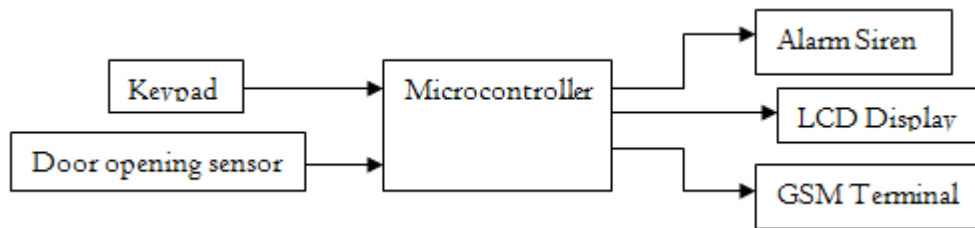


Figure 3.3: USB Based GSM Alarm System Setup

There are few major components that are needed to develop a digital security locking system. These components are PIC microcontroller PIC24FJ64GB002, 125 kHz RFID card reader, 4x4 matrix keypad, LCD display, magnetic lock and GSM terminal.

### 3.4 GSM terminal

#### 3.4.1 GSM/GPRS Modems

GSM/GPRS modem is the easiest way to build the security alert system. The best thing about the GSM/GPRS modem is that this modem can support AT commands very well and come in two types on connectivity interface, serial RS-232 and USB. However, the choice of these modems on the market is limited and the prices are expensive. USB has advantages on the ease of use but the way to program it is slightly difficult. The deep understanding in USB is a must to work with programming with USB.



Figure 3.4: GSM Modem from Wavecom

### 3.4.2 Cellular Phones

Cellular phones can serve as a cheaper terminal for sending SMS. However, not all cellular phones can work with AT Commands. Only certain models of phones are compatible with AT Command. Another issue will be the availability of data cable of old cellular phones. To find an old model of cell phone, it will be very easy. However, to get the data cable will be the hardest work as old model of cell phone mostly did not come with data cable. New model of cell phones usually come with cable which is in USB interface. Old model phones use RS-232 interface which is easier to work with compared to new model of phone which utilize USB cable.

### 3.4.3 Choosing GSM terminal

After doing some research, it is better to use a mobile phone as a GSM terminal for lower the cost of the system. Since the price of mobile phone is very much depends on the model numbers, old model of mobile phones will be used in the project as long as the phones support AT Command. Next, USB interface phone will be used as the availability is very much higher than those phones using RS-232 serial interface cable. Mobile phone Sony Ericsson C902 is the main experimental GSM terminal for this project and few other phones will be taken to test the performance of the project.



### **3.5 Modem Commands**

AT command will be the best choice among the modem commands as it has a standard command set. AT command is applicable in all AT Commands compatible modem while other modem command only work on certain model of phones only and usually the model of phones uses same modem commands are limited. Most of the phone in the market supports AT command. PDU mode will be adopted as it support on most of the GSM modem as well as mobile phone. Text mode is easier to implement but only very little phone does support text mode while almost all the phone that support AT command will support PDU mode.

### **3.6 Sensor**

Sensor can be either in wired or wireless. Since the number of sensor is very small and to lower the cost, wired sensor is recommended. Wired sensor is also easier to configure than wireless sensors. To get a stable signal from the sources, switch is the best choice. Switch can give a very accurate signal to feed to the microcontroller as an input to indicate the door is closed or opened. This is important as good signal can ensure the happening of false alarm is minimized. Since signal is required when the door touches the switch, snap action switch or proximity sensor will be the best available choice. Snap action switch may be good but due to it mechanical switch properties makes the device cannot last for long runs. There are many types of proximity sensor available and cheapest solution with high stability will be magnetic sensor.



Figure 3.5: Snap Action Switch



Figure 3.6: Magnetic Proximity Sensor

### 3.7 Microcontroller

There are several types of major microcontroller available in the market for our choice. They are 68HC08/68HC11 from Freescale or formally known as Motorola, 8051 from Intel, AVR from Atmel, Z8 from Zilog and PIC from Microchip Technology. Among the microcontroller, PIC microcontroller from Microchip is recommended. PIC stands for Peripheral Interface Controller. Microchip is the top supplier of 8 bits microcontroller in the world.

Since USB interface is chosen, a microcontroller with host capability is a must. USB require a host to control the speed of buses and manage the device connect to it which is called slave. Majority of the PIC microcontroller do not have USB capability and even they have USB capability, the only serve as a device or USB slave. USB On-The-Go (OTG) is a technology which enables a PIC

microcontroller to serve as a USB host. This technology was developed few years ago and until recently, the technology is included in some new PIC microcontroller.

There are two way to getting USB work on microcontroller. One is using USB host peripheral control together with another microcontroller and another is using microcontroller with USB OTG capability. The price of USB host peripheral is not cheap and deep knowledge on the device is required for it to work. Thus, microcontroller with USB OTG capability will be the choice.

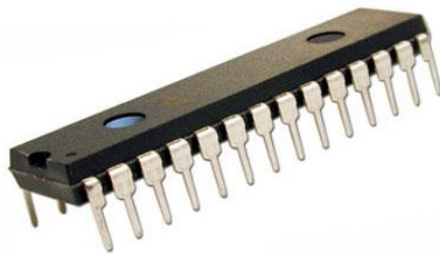


Figure 3.7: PIC Microcontroller (SPDIP Package)

PIC24FJ64GB002 is a 16 bits microcontroller with 64k flash memory. It has USB-OTG capability and 28 pins Shrink Plastic Dual In-Line Package (SPDIP) which is suitable to fit into our design. Besides, it is the cheapest and only PIC with USB-OTG capability in SPDIP. PIC24FJ64GB002 has the following features. PIC24 has advantage over earlier PIC family in the pin assignment. Most of the pin are remappable to easier the design of the whole circuit. Furthermore, more communication module can be use compared to earlier family of PIC if the pins are available. It comes with 5 timer and internal Real-Time Clock.

**Table 3.1: PIC24FJ64GB002 Specification**

Architecture	16-bit
Program Memory	64KB
CPU Speed	16 MIPS
Timer	5 x 16 bit

USB	1, Full Speed, USB OTG 2.0
Digital Communication Peripherals	2-UART, 2-SPI, 2-I2C
I/O Pins	21
RAM bytes	8192

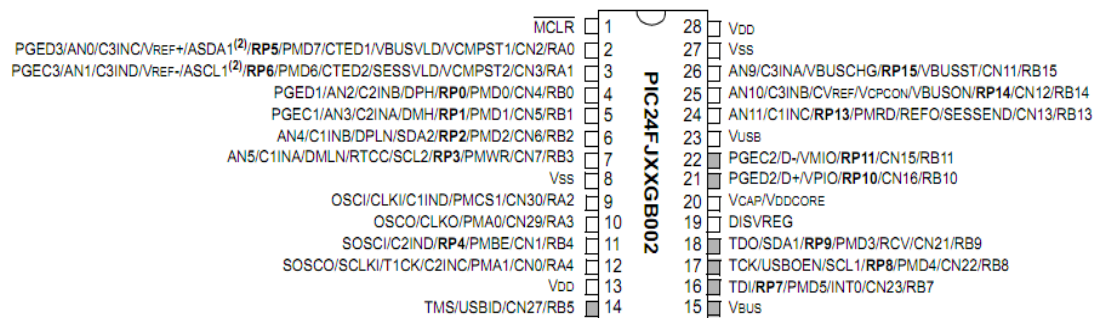


Figure 3.8: Pin Diagram of PIC24FJ64BG002

### 3.8 Matrix Keypad

To enter phone number and change setting, it is essential to have a keypad for this project. Matrix keypad is the most common keypad available due to its cheap and easy production. Compared with a SPI output keypad, the price of a SPI output keypad can be as high as five times of normal matrix keypad. Figure 3.8 and Figure 3.9 show the figure of 4x4 matrix keypad and internal connection respectively.



Figure 3.9: A 4x4 Matrix Keypad

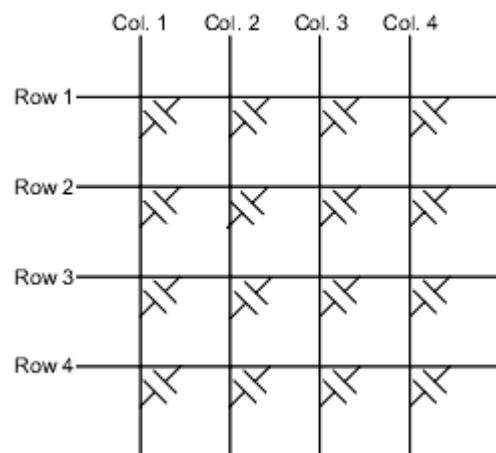


Figure 3.10: Internal Connection of a 4x4 Matrix Keypad

To read from the keypad, microcontroller has to do scanning through rows and column to read the key. Another option is using a keypad decoder which will be easier but a more costly solution. To stick with low cost solution, decode using microcontroller is preferable choice.

### 3.9 LCD Display

LCD is useful to display setting available so that setting can be changed on the project. LCD makes the project more user friendly and easier to understand on which operation is currently going on. A 2x16 LCD display is more than enough to display the information and changing setting. Figure 3.10 shows a 2x16 LCD display and table 3.1 shows the connection pin of the LCD.



Figure 3.11: A 2x16 LCD Display

**Table 3.2: LCD Pin Table**

Pin Number	Name	Description
1	V <sub>SS</sub>	Negative supply for LCD
2	V <sub>CC</sub>	Positive supply for LCD
3	V <sub>EE</sub>	Contrast Adjustment for LCD
4	RS	Register Select (0: Command, 1: Data)
5	R/W	Read or write selection (0: Write, 1: Read)
6	E	Enable Signal
7	DB0	Data Bus 0
8	DB1	Data Bus 1
9	DB2	Data Bus 2
10	DB3	Data Bus 3
11	DB4	Data Bus 4
12	DB5	Data Bus 5
13	DB6	Data Bus 6
14	DB7	Data Bus 7
15	LED+	Positive supply for LCD back light

16	LED-	Negative supply for LCD back light
----	------	------------------------------------

### 3.10 MPLAB IDE

MPLAB IDE is a free programming tool which is use together with PIC microcontroller developed by Microchip. This software has the ability to build the files require to program into microcontroller for project development as well as in circuit debugging.

MPLAB IDE will be use to write the program for digital security locking system. In this project, program of USB based GSM security system is developed using MPLAB IDE C30 compiler which the program is written using C language. Assemble language may have faster and more compact code but due to the complexity of the code in large program, it is hard to handle and C language is preferred over assembly language. There will be few major functions inside the program which is reading and decoding keypad, write to LCD display, read from sensor when interrupt occurs and send SMS using GSM terminal when anything happen.

MPLAB will be using together with PICKIT 2 to program the microcontroller. PIC24FJ64GB002 can only perform in circuit debugging when use with newer version of PICKIT which is PICKIT 3. However, the cost of PICKIT 3 cost around RM360 which is very expensive and it is beyond the budge. Thus, PICKIT 2 will be use for programming purpose. Debugging of the program will be done manually using test and trial method.

### 3.11 EAGLE Layout Editor

Eagle Layout Editor simply means Easily Applicable Graphical Layout Editor. It is software developed by Cadsoft. Eagle Layout Editor will be use to develop PCB layout for digital security locking system. Eagle Layout Editor has a lot of component to choose from and it has auto route function to express the PCB layout development. However, auto route function may not be suitable in this project but it may be a good guidance for new user.

### 3.12 Schedule of FYP 2

The schedule of FYP 2 starts with development of USB based GSM security system. After the focused part is done, the project will then integrate with RFID based locking system to create a digital security locking system. After that, whole project after integration will be built on PCB and a model is built to represent the house.

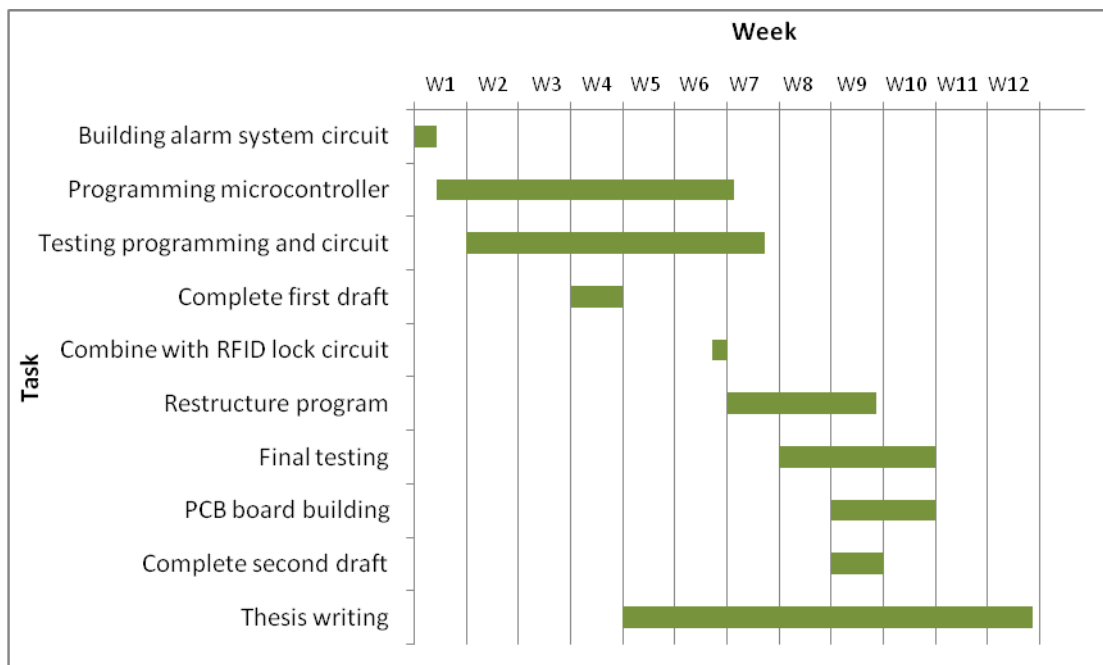


Figure 3.12 Gantt Chart of FYP Part 2



## **CHAPTER 4**

### **RESULTS AND DISCUSSIONS**

#### **4.1 Introduction**

In this chapter, there will be two major parts which is the software part and the hardware part. The software part includes the discussion of program codes as well as step by step system configuration while the hardware part includes the schematic and PCB drawing. Discussion of program codes include embedded generic USB host, LCD as well as matrix keypad.

#### **4.2 Software Implementation**

##### **4.2.1 Software Language**

Due to complexity of the program and the effectiveness of C language in MPLAB C30 compiler, C language is preferred over assembly as the program code is expected to be very long. There will be some long string appear in the program and with the advantages of C language in comparing the string, we can easily writing out the code and combine few of the part together using C language program.

## **4.2.2 USB**

### **4.2.2.1 USB System**

USB is a type of serial communication which is more intelligent than previous serial communication such as RS-232. In RS-232 interface, the content of data is treated as a piece of data regardless what types of data it is. In the USB, each data is categorized and the types of transfer depending of the data types. However, the data is transferred between host and device(s). Typically a host is a computer with host controller hardware and software and a device is a peripheral with device controller hardware and firmware.

Host is responsible to detect attachment of device and assign each device attached a driver. These drivers manage the communication between host and devices. When device is attached, host will perform enumeration in which host and device are exchanging information. Host will request device description which acts like an identity of the device. During the enumeration process, the device is assign with a device address 0.

### **4.2.2.2 USB Embedded Host System**

USB does not allow direct communication of host to host and device to device. When data transfer between two devices such as camera and printer is needed, it is inconvenience to have a computer to do the job. An USB embedded host is needed to solve the problem. Unlike USB host of computer, USB Embedded host only has limited capability and support certain device classes. Most of the embedded host do not support Hubs structure as computer USB host does which is part of the limitation of USB embedded host. USB embedded host is design for specific purpose and to meet the minimum requirement of data transfer.

#### **4.2.2.3 USB On-The-Go (USB OTG)**

To perform data transfer between two USB devices, one of them must be able to perform the role as the host and another as the device. On PC, the driver of USB host is complicated and with many function. However, the Embedded USB host has only limited function due to its memory size. USB has many classes and one of them is CDC (Communication Device Classes). Communication device classes are suitable for communication devices that utilized AP command.

There are few USB host driver provided by Microchip which is CDC, charger, mass storage, HID, printer and generic type driver. There are two types of driver we can choose from which is CDC and generic host. However, CDC embedded host is not suitable as mobile phone is not just a communication device but also a mass storage device. To deal with this problem, a generic embedded host driver is used instead of CDC host. Apart from combination of CDC device and mass storage device, mobile phone may fall in other category which is neither CDC nor mass storage.

When these devices do not falls to one of the class stated above, a generic host will be the last hope. A generic host has the basic ability of USB host which is managing the flow the data and at the same time, enabling both devices to communicate with each other. After some experiment, CDC classes host driver cannot support the mobile phone that is use in the experiment. On the other hand, generic driver can support most of the mobile phone in which the mobile phone can act like a GSM terminal.

#### **4.2.2.4 USB Device Classes**

Unlike older serial communication interface, USB categorized types of data transfer according to device classes. These device classes increase with the version of USB increase. There are some popular classes which are commonly used and Table 1.2 shows the popular device classes.

**Table 4.1: USB Device Classes**

Classes	Description	Example
0x01h	Audio	Speaker
0x02h	Communication Device Classes (CDC)	Modem
0x03h	Human Interface Device (HID)	Mouse
0x07h	Printer	-
0x08h	Mass Storage	Flash Drive
0x09h	Hub	-

When some of the composite device which has overlapped classes that does not belongs to any of the group above, typical a generic host driver is needed to solve the problem. Generic drivers enables the use of control, isochronous, bulk and interrupt transfer using driver specified application programming interface (API). A mobile phone may not be belongs to CDC classes since it includes modem and mass storage.

#### **4.2.2.5 USB Enumeration**

There are six steps in enumeration process which is Powered, Default, Address, Configuration, Attach and Suspend process. During a device attached to the host, the typical process in list in the table.

**Table 4.2: Enumeration Steps**

No	Step	Description
1	Power up device	The device attach is drawing power from the host up to 100mA.
2	Detect new device	The host detects the device by monitors the D+ and D- signal lines.
3	Host learn new device	Interrupt occurs and host check the new device attached.
4	Detect device speed	Host determine the speed by monitoring both D+ and D- line.
5	Reset device	Port is reset using the new configuration.
6	Detect if support high speed	If the device supports full speed, host will then try to use high speed if available.
7	Establish signal path	After port resume from reset, the device is now in default state. The device is ready to response from the host.
8	Learn Descriptor	Host request device descriptor from the device using endpoint 0.
9	Assign address	After getting device descriptor, the device is reset and gives a new address.
10	Learn device ability	The host now request for full device descriptor such as maximum packet size, number of configuration and some basic information.
11	Assign device driver	The host will search for any match in the device management and assign to it.
12	Select configuration	If device configuration is available, the device driver request configuration from the device.

#### 4.2.2.6 USB Files in Programming Part of the Project

Microchip provided USB host driver demo project and USB client driver project in their website for use with their development board. These files contain host driver file with can be use with other device if proper modification are made. Writing an own USB host driver may be time consuming and great knowledge in USB structure, as well as USB operation is needed. Thus, several files are taken from the demo which is then include in the project to make the USB host driver project easier. There are many categories of host driver files and the most suitable host driver will be included in the project. Apart from choosing host driver, defining types of transfer and targeted peripheral list (TPL) must be done to make the project working. Some

major files included in the project are as following. Table 4.1 shows some important file in the project and Figure 4.1 shows an example of target peripheral list.

**Table 4.3: USB Files in Programming Part of the Project**

File name	Comment
usb_config.c	This file defining types of transfer allowed, current needed by USB client and attach debounce interval. Types of transfer allowed must be select correctly or else the project cannot work properly or fail to work. This file configures the USB stack.
usb_config.h	This file contain target peripheral list (TPL) in which defining the client supported by the host. This file configures the USB stack.
usb_host_generic.c	This is generic USB embedded host driver file in which the operation of generic host is enabling by this file. Each type of host driver has different operation allowed which defining by this file.
usb_host.c	This file is the main USB embedded host driver file which use to control most of the USB operations. These operations include detection of device attached and detached, configuring speed of transfer, performing USB enumeration, and start of transferring data between host and client. All USB embedded host driver must have this file included in the project. This file does not provide class support.

```
// *****
// USB Embedded Host Targeted Peripheral List (TPL)
// *****

USB_TPL usbTPL[] =
{
    { INIT_VID_PID( 0x0fceul, 0xd016ul ), 0, 0, {0} } // k750
,
    { INIT_VID_PID( 0x0fceul, 0xd0d4ul ), 0, 1, {0} } // C902
,
    { INIT_VID_PID( 0x0fceul, 0xd039ul ), 0, 2, {0} } // K800
,
    { INIT_VID_PID( 0x0fceul, 0xd0b7ul ), 0, 3, {0} } // k770
,
    { INIT_VID_PID( 0x0421ul, 0x0025ul ), 0, 4, {0} } // 5300
};
```

Figure 4.1: Example of TPL List

#### 4.2.2.7 USB Operation

To use the USB function inside microcontroller, the function USBHostInit must be called before any USB operation can be performed. USBHostInit will initiate the USB host stack for USB operation. As soon as USBHostInit is successfully performed, USBHostTasks should be called to perform hardware initialization as well as perform USB host task. USBHostTasks include checking on device attach or detach and perform enumeration. USBHostTasks must be called regularly to make sure USB operation is working correctly. If USBHostTasks did not call regularly, device detachment or attachment will not be detected by microcontroller and USB operation might fail. Table 4.2 shows the experimental result of time for enumeration for different calling interval of USBHostTasks.

**Table 4.4: Effect of USBHostTasks Calling Interval on USB Enumeration**

Calling Interval (ms)	Enumeration Time (s)
1000.000	30.0
500.000	15.0
250.000	7.8
125.000	4.0
62.500	2.6
31.250	2.0
15.625	2.0

USB enumeration will take minimum 2 s to complete the whole process. For calling interval of 31.250 ms, the system will have periodic lag on displaying text on LCD and for 15.625 ms, the system can hardly display text on LCD screen. Calling

interval of 62.500 ms is the better choice for system performance and enumeration time.

#### **4.2.2.8 Device Endpoint**

Each of the data is travel from or to device endpoint. Device endpoint is served as a buffer for the data to be transferred to the host and buffer for data received from the host. When USB stack is initializes, each endpoint is assign with transferred types using and Endpoint 0 is especially for enumeration process. Every USB device must at lease have endpoint 0 for the host to obtain information from the device. Each endpoint can have data in both directions. For example endpoint 0 IN means data transfer from device to host in endpoint 0 and endpoint 0 OUT will means data transfer from host to device. Endpoint can have the number range from 0 to 15.

#### **4.2.2.9 USB Transfer Types**

As mention of earlier chapter, there are four types of transfer in USB. Control type transfer is supported by all the USB device of else the enumeration will fail. Three other transfer types is use for different purpose and both USB host and USB device must aware which types of transfer will going to take place or the transfer of data will fail.

Referring to embedded generic USB host driver provided by Microchip, generic host will utilize Endpoint 0 for enumeration and Endpoint 1 for data transfer. In the code, each endpoint has different types of data transfer. Figure 4.2 shows the transfer type on different endpoint.



```

// Section: USB Endpoint Transfer Types

#define USB_TRANSFER_TYPE_CONTROL      0x00 // Endpoint is a control endpoint.
#define USB_TRANSFER_TYPE_ISOCHRONOUS  0x01 // Endpoint is an isochronous endpoint.
#define USB_TRANSFER_TYPE_BULK         0x02 // Endpoint is a bulk endpoint.
#define USB_TRANSFER_TYPE_INTERRUPT    0x03 // Endpoint is an interrupt endpoint.

```

Figure 4.2: Transfer Types of Different Endpoint

To use another endpoint other than the default Endpoint 1 for data transfers, another function is written to use Endpoint 2 for data transfer. Besides that, a function is written to test on which transfer types that the device supported. After the function has finish testing on types of transfer supported, the next transfer of data will utilize the type of transfer that is supported. Figure 4.3 shows the coding for testing transfer types available and begin to transfer immediately after the test. Figure 4.4 show the process of testing transfer types.

```

while (USBHostGenericTxIsBusy(deviceAddress));
RetVal = USBHostGenericEP2Write(deviceAddress, buff, 3);
if (RetVal == USB_SUCCESS){
    EP = 1;
}
else{
    EP = 0;
}

if (currstate == 0){
    while(USBHostGenericTxIsBusy(deviceAddress));
    if(EP == 1){
        RetVal=USBHostGenericEP2Write(deviceAddress, buff0, 11);
    }
    else{
        RetVal=USBHostGenericWrite(deviceAddress, buff0, 11);
    }
    currstate = 1;
    delay(100);
}
if (currstate == 1)
{ while(USBHostGenericRxIsBusy(deviceAddress));
  if (EP == 1){
    RetVal=USBHostGenericEP2Read(deviceAddress,Home, sizeof(Home));
  }
  else{
    RetVal=USBHostGenericRead(deviceAddress,Home, sizeof(Home));
  }
}

```

Figure 4.3: Code to Test Transfer Types and Initiating Data Transfer

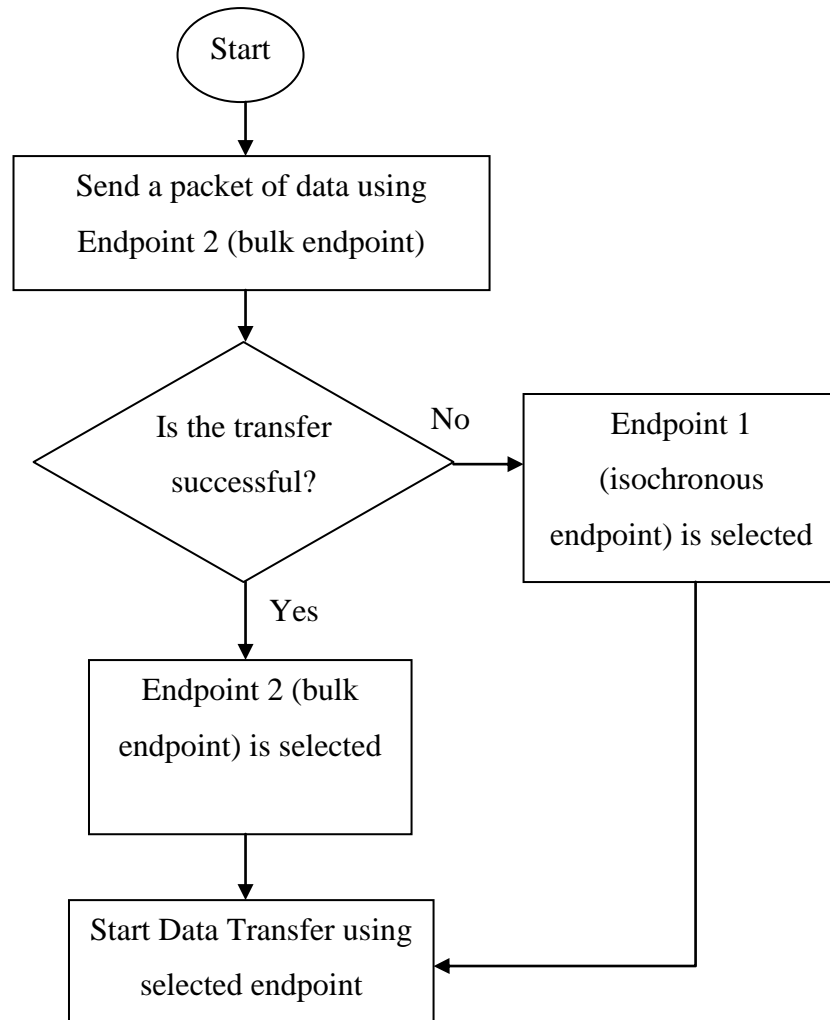


Figure 4.4: Flow Chart of Testing USB Transfer Type and Initializing Data Transfer

#### 4.2.2.10 Generic Host Driver Write Function

Figure 1 shows the default USB write function in the Microchip generic host driver.

```

BYTE USBHostGenericWrite( BYTE deviceAddress, void *buffer, DWORD length )
{
    BYTE RetVal;

    // Validate the call
    if (!API_VALID(deviceAddress)) return USB_INVALID_STATE;
    if (gc_DevData.flags.txBusy) return USB_BUSY;

    // Set the busy flag and start a new OUT transfer.
    gc_DevData.flags.txBusy = 1;
    RetVal = USBHostWrite( deviceAddress, USB_OUT_EP|USB_GENERIC_EP, (BYTE *)buffer, length );
    if (RetVal != USB_SUCCESS)
    {
        gc_DevData.flags.txBusy = 0; // Clear flag to allow re-try
    }

    return RetVal;
} // USBHostGenericWrite

```

Figure 4.5: Generic Host Driver Write Function

The function will first check for valid machine state as well as any USB host is still busy to transfer data to device. If machine state is not valid or USB transfer is still busy, the function is terminated and returns the error code. When the entire test passes, the write flag will be set high to indicate a write is performing at that time. Next, the write process is started by calling USBHostWrite function in host driver. USBHostWrite function takes four variables which are the device address, endpoint number, pointer to the data and the length of the data to be transferred. The default endpoint is endpoint 1 which is for isochronous transfer. By replacing USB\_GENERIC\_EP in the figure to other value, different types of transfer can be used as different endpoint has its own transfer types. Another function called USBHostGenericEP2Write is written same as USBHostGenericWrite function with underline part changed to USB\_GENERIC\_EP2. This function utilizes endpoint 2 which is bulk transfer instead of endpoint 1 isochronous transfer for USB data transfer. Note that USBHostWrite will return result of the USB write function. This result code is defined in USBHost.h and Table 1 shows the result code.

**Table 4.5: USB Result Code**

Result Code	Description
USB_SUCCESS	Write started successfully.
USB_UNKNOWN_DEVICE	Device with the specified address not found.
USB_INVALID_STATE	We are not in a normal running state
USB_ENDPOINT_ILLEGAL_TYPE	Must use USBHostControlWrite to write to a control endpoint.
USB_ENDPOINT_ILLEGAL_DIRECTION	Must write to an OUT endpoint
USB_ENDPOINT_STALLED	Endpoint is stalled. Must be cleared by the application.
USB_ENDPOINT_ERROR	Endpoint has too many errors. Must be cleared by the application.
USB_ENDPOINT_BUSY	A Write is already in progress.
USB_ENDPOINT_NOT_FOUND	Invalid endpoint.

With the USB result code, we can determine the transfer types which are supported by the device and use the transfer types to communicate between host and device.

### 4.2.3 Emulating Data EEPROM

This project need memory to store phone number, SMSC number as well as password so that the device is customizable and flexible to use. It is unacceptable for user to program the whole device again if they need to change password or target phone number.

In PIC24, there are no EEPROM which available in past family of PIC such as PIC18, PIC16 or PIC12. PIC uses flash memory as program memory. One of the methods is using table lookup to use the internal flash memory or using external memory storage such as Data EEPROM or memory card. In the design, the pin is

very limited as the PIC in the project has just twenty eight pins and nineteen pins are capable to become I/O pins. To integrate the microcontroller with LCD, keypad, USB connector and RS-232 converter, all the pins become unavailable and hence, external memory is not an option.

As a result, Emulating Data EEPROM which utilized internal flash memory will be a better option for its fast operation. PIC24 has twenty four bits width data size in which emulating data EEPROM will take eight bits as address and sixteen bits as data. This eight bits is not a real address but it act like a tag for the data. First Data EE Address is use store page status and it is not available for storing user data. Table 4.3 shows the data memory structure and Table 4.4 shows the user data storage location of this project.

**Table 4.6: Data Memory Structure**

Page Address	Data EE Address	Data EE Data
This is the real address in the program memory.	This is tag for data.	User data.

**Table 4.7: User Data Storage Location**

Data EE Address	User Data
1-110	Card unique ID. Each card uses 10 addresses to store data in word.
200	User password. Uses 6 addresses space.
210	Administrative password. Uses 6 addresses space.
220	SMSC number. Uses 12 addresses space. As for PDU modes, an 'F' is added after the number before it is encoded.
240	Target phone number. Uses 12 addresses space. As for PDU modes, an 'F' is added after the number before it is encoded.

#### 4.2.4 Decoding keypad

Matrix keypad is the most common keypad available due to most cost effective device. However, a hardware decoder is needed to decode the keypad or keypad can simply decode inside microcontroller by writing some function.

When a key is pressed, one wire from a column will touch with another wire from a row. To decode the wire, either row or column will be the input. If column is the input, the row will be the output and vice versa. Now, column is to be the output and row will be the input. Initially the entire row connected to the microcontroller is set to high.

Next, scan all the port connected to column to detect any port is high. Whenever port connected to column is high, entire row is set to low. Then, each row is turn on one by one to scan which row is pressed so that the location of the button can be determined. In between turning on and off, delay is needed to avoid detection of wrong key pressed. After the location of the key is determined, the program will return the value and wait until user leave their hand from the keypad. Again, delay is needed to avoid the function to go inside the loop again to test for key pressed when user leave their hand from keypad. Figure 4.5 and Figure 4.6 show the flow chart of decoding keypad.

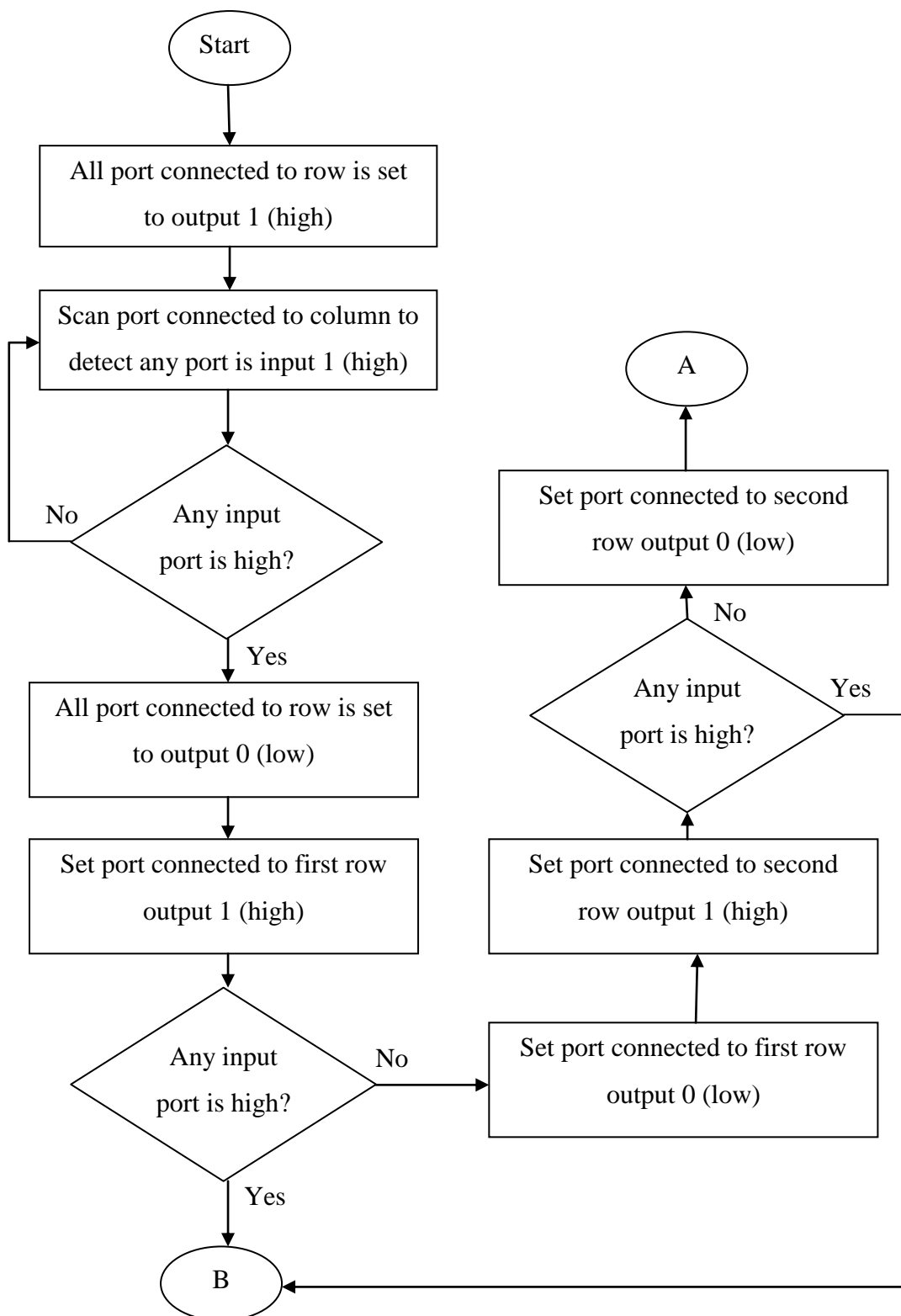


Figure 4.6: Flow Chart of Decoding Keypad Part 1

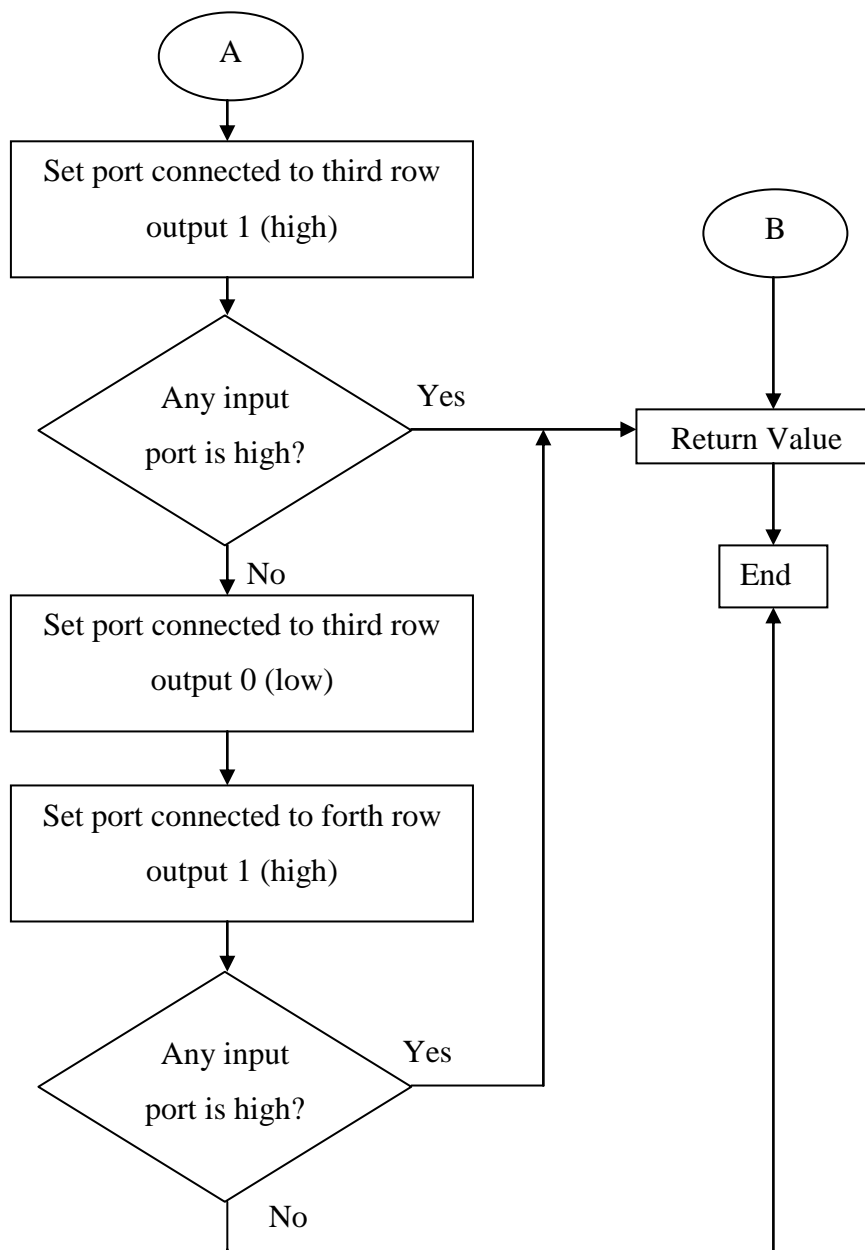


Figure 4.7: Flow Chart of Decoding Keypad Part 2



#### 4.2.5 Program on LCD Display

LCD is useful to show information and change setting. Before an LCD can display words or symbol, it has to be first initialize with proper configuration for it to display correctly. There are some sequences that need to be done for LCD initialization. Figure 4.7 shows initialization process for four bits mode of the LCD in this project.

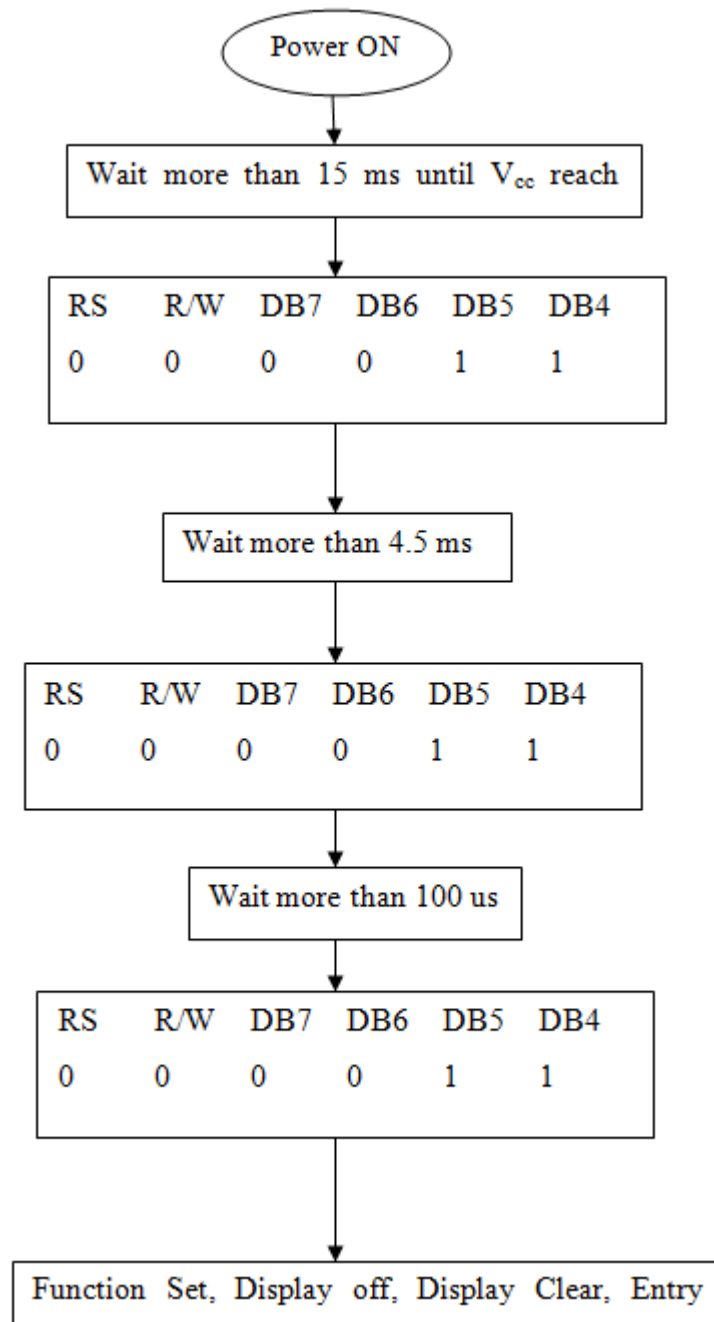


Figure 4.8: LCD Initiation Process

Suppose the LCD is to display two rows and showing the location of the cursor, function and entry mode had to be set to ensure proper operation. The LCD is use in four bits mode to reduce the I/O pins required by the LCD. To do that, LCD is configured to four bits mode operation and Figure 4.8 shows the coding for function set and entry mode set.

```

/* Function:  initlcd
   Overview:  This routine initialize port for lcd and configure
              lcd for system initialization */
void initlcd(void)
{
    TRISB = 0xFFFF0;          // Output RB0-RB3
    TRISBbits.TRISB13 = 0;   // Output RB13
    TRISBbits.TRISB14 = 0;   // Output RB14
    delay(1);
    writelcd4b(0,0x02);      // 4bit mode
    delay(8);
    writelcd8b(0,0x28);      // 4bit mode
    delay(8);
    writelcd8b(0,0x0E);      // turn on display, on cursor
    delay(8);
    writelcd8b(0,0x06);      // Increment mode
    delay(8);
    writelcd8b(0,0x01);      // Clear screen
    delay(8);
}

```

Figure 4.9: Program Code for Function Set and Entry Mode Set of LCD

To access LCD using four bits mode operation, each data bytes sent to LCD has to be done in two times of four bits each of the time. Higher nibble is sent before the lower nibble does. To achieve this operation, the data is first saved into temporary data space. After that, the data are performing with logical conjunction to obtain the lower nibble and higher nibble. When the data are ready, it will send to LCD with higher nibble follow by lower nibble. Register select is to be high when writing data to LCD and low when giving command to LCD. Figure 4.9 shows the program code for sending data to LCD.

```

/* Function: writelcd8b
   Overview: This routine writes a byte of data to lcd.
             Data is send to lcd in 4bit mode */
void writelcd8b(unsigned int fnc, char text)
{
    accesslcd();           // Switch to LCD
    delay(1);
    char temp = text;     // Save data inside temp
    temp = temp & 0xF0;   // Take upper 4 bit
    temp = temp >> 4;     // Right shift to output data
    text = text & 0x0F;   // Take lower 4 bit
    PORTB = temp;         // Output upper 4 bit first
    if(fnc)                // Check for intruction or data
        PORTBbits.RB13 = 1;
    else
        PORTBbits.RB13 = 0;
    PORTBbits.RB14 = 1;   // Enable
    delay(1);
    PORTBbits.RB14 = 0;   // Disable
    PORTB = text;
    if(fnc)
        PORTBbits.RB13 = 1;
    else
        PORTBbits.RB13 = 0;
    PORTBbits.RB14 = 1;
    delay(1);
    PORTBbits.RB14 = 0;
}

```

Figure 4.10: Program Code of Writing Data to LCD

### 4.3 Program overview

The program begins at initiating the USB peripheral. As soon as USB peripheral is started, it will check for any USB device plug into the USB type-A female connector. If there is no device detected, it will continue looping to search for device attached. On the other hand, if a USB device is attached, it will then check whether the device is supported. If the attached device is supported, the program will now ready to read the input from door sensor. Any unauthorized opening of door will cause the alarm to be activated and at the same time, a SMS is sent to warn the user. Figure 4.10 shows the program flow chart of the main program and Figure 4.11 shows the LCD screen of the main program.

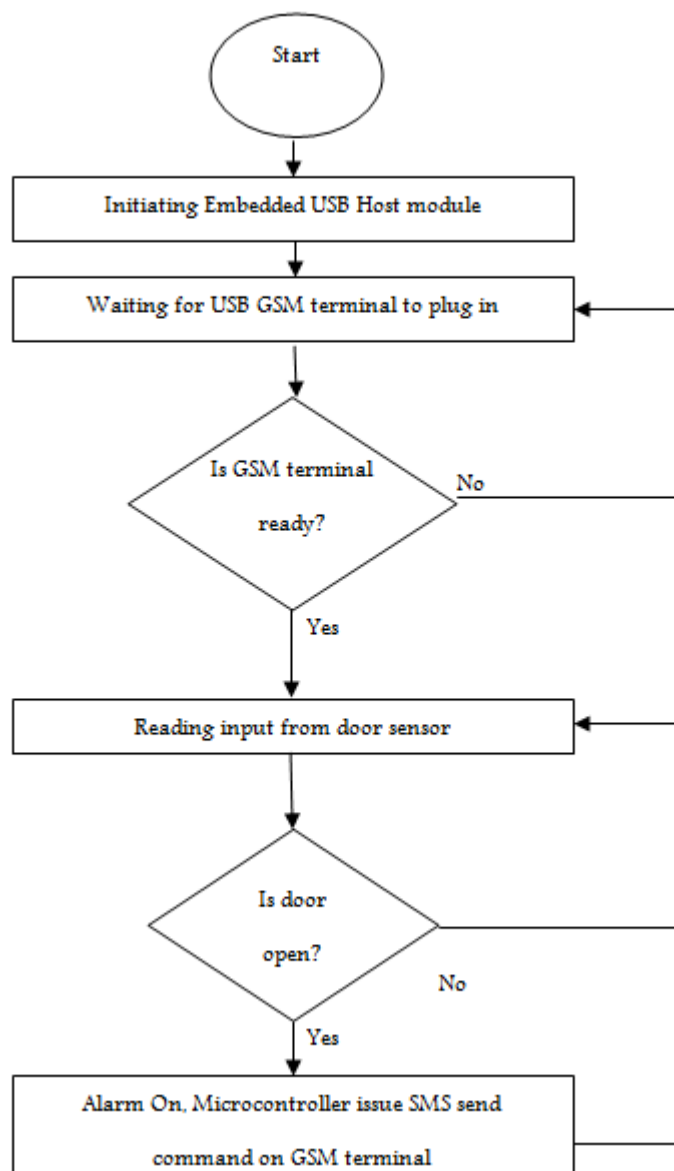


Figure 4.11: Program Flow Chart of USB Based GSM Security System

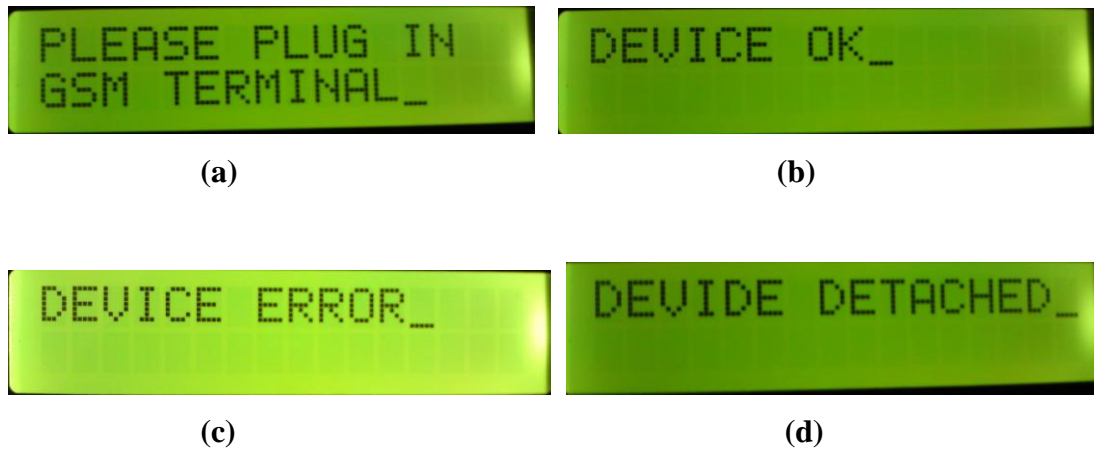


Figure 4.12: LCD Display on (a) Start up, (b) Supported Terminal Attached, (c) Unsupported Terminal Attached, (d) Terminal Detach

### 4.3.1 Option menu

Option menu can be accessible by pressing 'B' on the keypad. When pressing 'B' on keypad, user will be prompt for two set of password. First password will be user password where it uses to unlock door and second set of password is the administrative password which is use when user change setting. Table 4.5 shows the option menu for the program and Figure 4.12 and Figure 4.13 show the menu flow chart.

**Table 4.8: List of Option Menu**

Option key	Description
3	This option is use to change user password. User password is in 6 digits length and use for unlocks door. When this option is selected, user will be prompt to enter 6 digits of user password. If user accidentally press the wrong button, the key 'C' is use to re-enter the password again. A message "DONE" will be displayed on LCD for successful changing of user password.
4	This option is use to change administrative password. Administrative password is also in 6 digits length and use when changing setting. When this option is selected, user will be prompt to enter 6 digits of administrative password. If user accidentally press the wrong button, the key 'C' is use to re-enter the password again. A message "DONE" will be displayed on LCD for successful editing of administrative password.
5	This option enable user to change the target phone number for break in case occurs. The length of target phone number is 11 digits which including the country code. Example of phone number in Malaysia is "60123456789". A message "DONE" will be displayed on LCD for successful changing target phone number.
6	This option lets user change SMSC number. Different service provider will have different SMSC number. This is useful when user changes service provider of the GSM terminal. The length of SMSC number is same as phone number which is 11 digits. A message "DONE" will be displayed on LCD for successful editing of SMSC number.

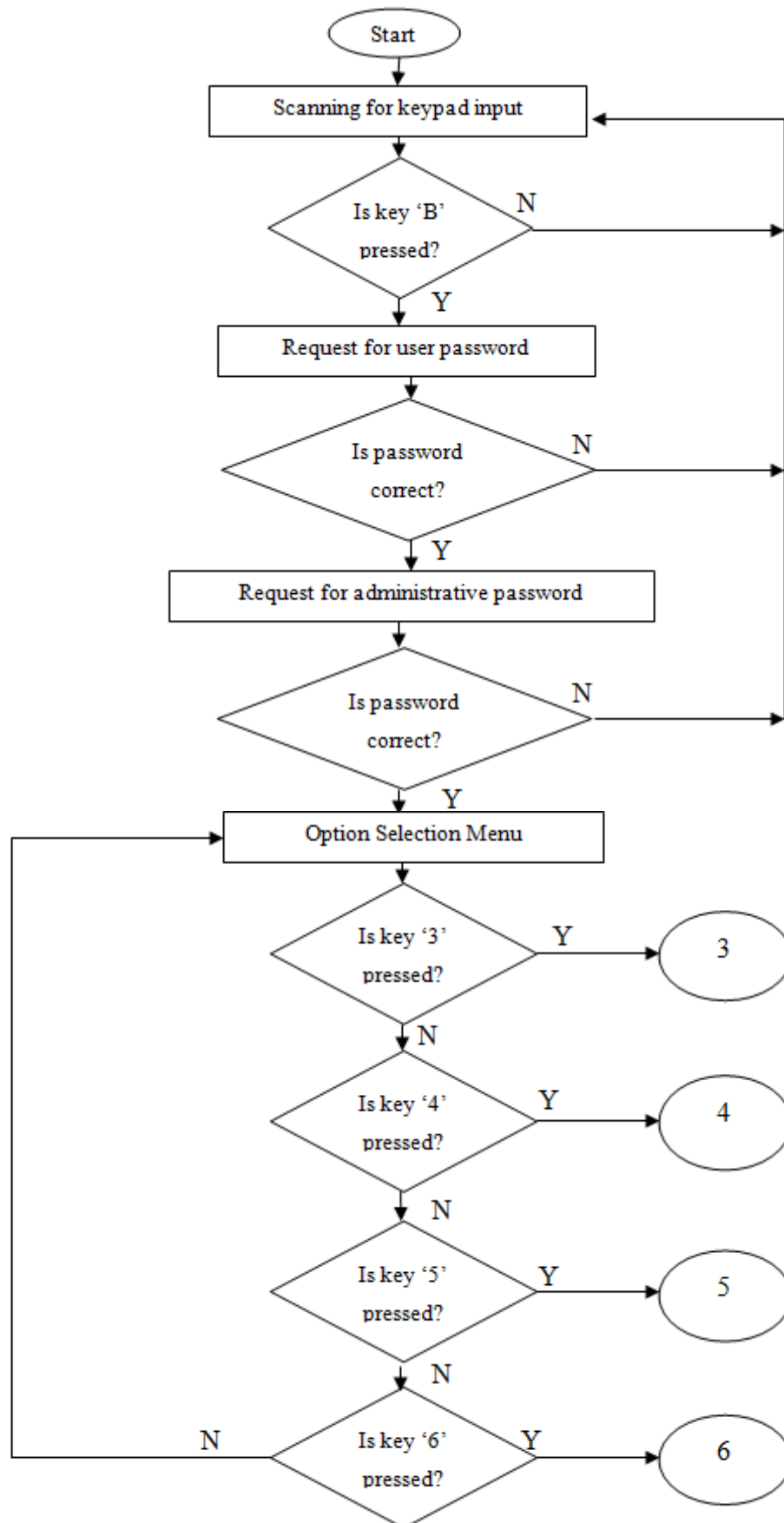


Figure 4.13: Menu Flow Chat Part One

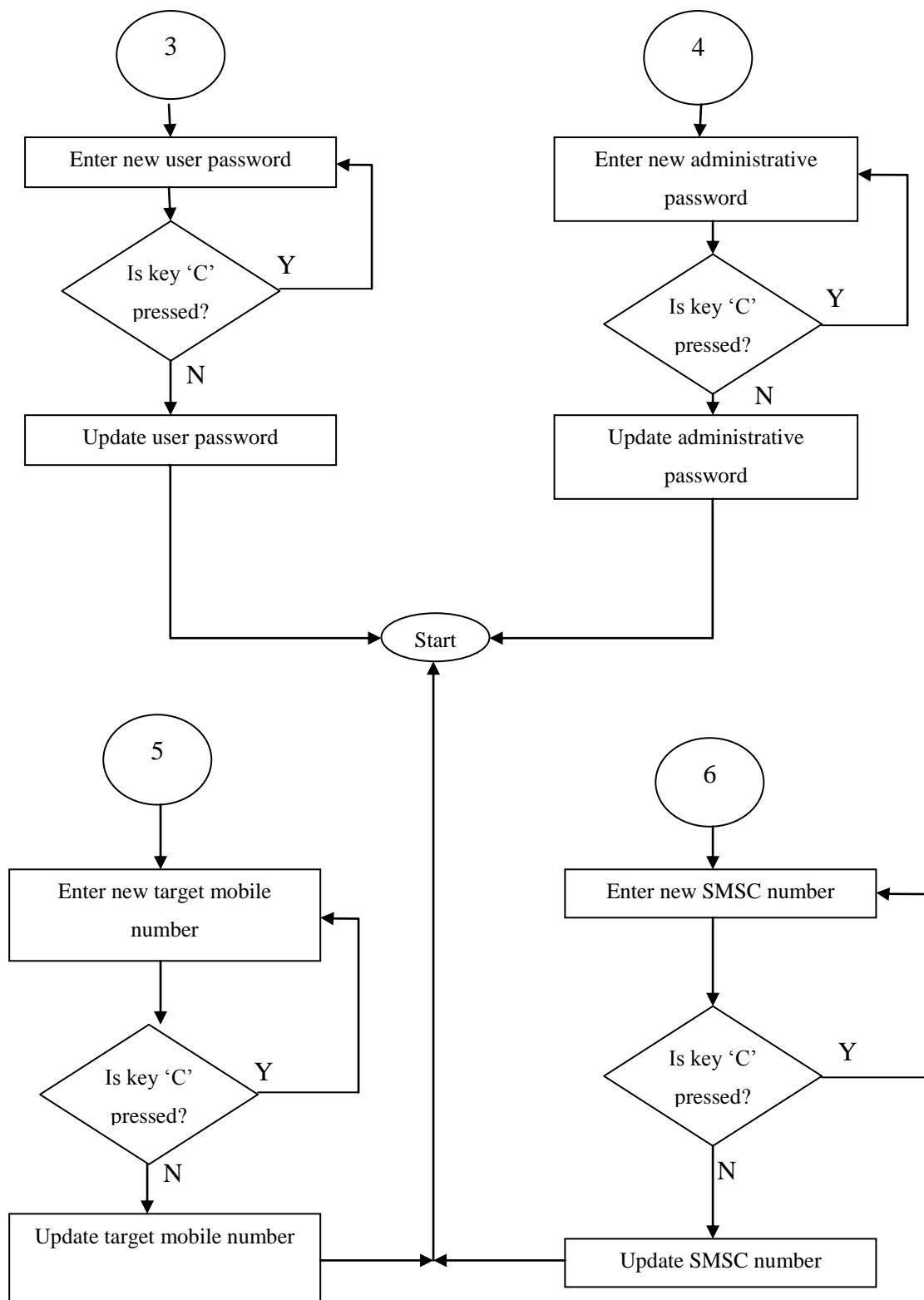


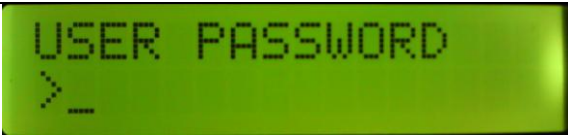
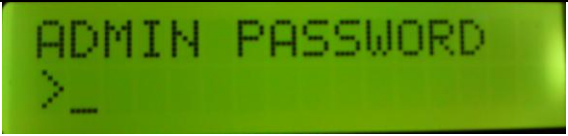




Figure 4.14: Menu Flow Chat Part Two



### 4.3.2 Detailed Operation on Changing Password

The operation of changing password will be show in detail steps by step together with display on LCD to explain how password changing process is carry on. Table 4.6 shows the detailed steps to change password.

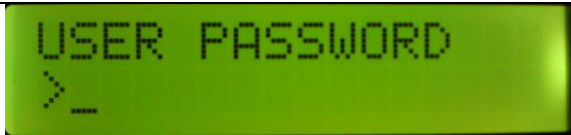
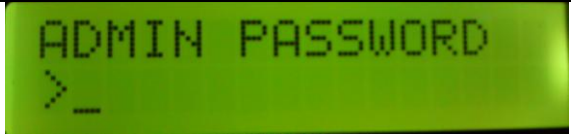




**Table 4.9: Detail Operation on Changing Password**

Detail Operation	Display shown on LCD
When user presses 'B' during normal operating condition, user will be prompt for user password. Password to be entered is in six digit lengths.	
After user password verification is successful, user will then prompt again for administrative password. The administrative password is in six digits length.	
If both passwords are entered correctly, user will then redirect to option selection menu. There are totally seven options available to select from.	
Press '3' on keypad for changing user password and '4' for changing administrative password. LCD will show enter password after the key is pressed.	
If user accidentally pressed the wrong key, 'C' on keypad is use to enter password from the beginning.	
When user successfully entered all the passwords, a message will show on LCD to indicate operation successful.	

### 4.3.3 Detailed Operation on Changing Target Phone Number

It is important for a GSM based security system able to change target phone number for emergency case. Detailed operation of changing target phone number is shown in Table 4.7.

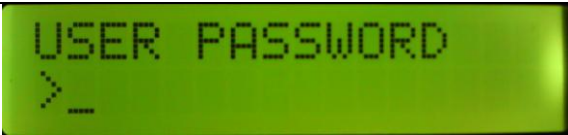
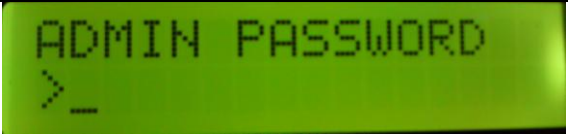




**Table 4.10: Detail Operation on Changing Target Phone Number**

Detail Operation	Display shown on LCD
When user presses 'B' during normal operating condition, user will be prompt for user password. Password to be entered is in six digit lengths.	
After user password verification is successful, user will then prompt again for administrative password. The administrative password is in six digits length.	
If both passwords are entered correctly, user will then redirect to option selection menu. There are totally seven options available to select from.	
Press '5' on keypad for changing target phone number. Phone number to be entered is in ten digits length and example is 60123456789.	
If user accidentally pressed the wrong key, 'C' on keypad is use to enter phone number from the beginning.	
When user successfully entered all the numbers, a message will show on LCD to indicate operation successful.	

#### 4.3.4 Detailed Operation on Changing SMSC Number

By using PDU mode, SMSC number has to be entering manually by user. Wrong SMSC number can lead to message unable to deliver out and leave to whole system unprotected. Table 4.8 show the step needed to change SMSC number.

**Table 4.11: Detail Operation on Changing SMSC Number**

Detail Operation	Display shown on LCD
When user presses 'B' during normal operating condition, user will be prompt for user password. Password to be entered is in six digit lengths.	
After user password verification is successful, user will then prompt again for administrative password. The administrative password is in six digits length.	
If both passwords are entered correctly, user will then redirect to option selection menu. There are totally seven options available to select from.	
Press '6' on keypad for changing target phone number. Phone number to be entered is in ten digits length and example of SMSC number for MAXIS is 60120000015.	
If user accidentally pressed the wrong key, 'C' on keypad is use to enter phone number from the beginning.	
When user successfully entered all the numbers, a message will show on LCD to indicate operation successful.	

#### 4.4 Results of Tested Possible Mobile Phone as GSM Terminal

To test the performance of generic host driver with GSM terminal, some phone has been taken into experiment. Before that, product number (PID) and vendor number (VID) must be obtain and insert into target peripheral list (TPL). Generic host will only support the device inside TPL table. Table 4.9 shows some result obtain from the experiment of few mobile devices.

**Table 4.12: Test Result of Mobile Phones**

Device Name	Enumerations	Send Message
Nokia 5300	Success	Supported, PDU and text mode
Nokia 5800 Express Music	Success	Supported, PDU and text mode
Nokia N95	Success	Supported, PDU and text mode
Sony Ericsson C501	Success	Supported, PDU mode only
Sony Ericsson C902	Success	Supported, PDU mode only
Sony Ericsson K750	Success	Supported, PDU mode only
Sony Ericsson K770	Success	Supported, PDU mode only
Sony Ericsson K800	Success	Supported, PDU mode only

The data in Table 4.9 show that most of mobile phone can communicate effectively with microcontroller using control transfer. Control transfer is use to perform enumerations and most of the device will support this types of data transfer. In the experiment that carried out, all the mobile phone listed in Table 4.9 successfully send out SMS by using either isochronous transfer or bulk transfer. Note that PDU mode is supported by all the phones use in the experiment and only Nokia phones in the experiment support text mode.

## **4.5 Hardware implementation**

### **4.5.1 Hardware Placement**

The main board which microcontroller is on it is to be placed inside the house while LCD and keypad is to be located on outside the house. With this placement, the main board can protected from attack to keep the security system working all the time while providing access to user outside the house. If the whole unit is place outside the house, people can easily damage the board and security system will offline and thus, leaving the house unprotected.

### **4.5.2 Circuit Diagram**

This project includes development of hardware as well as software. A circuit is needed achieve the goal. USB based GSM alarm system is part of the digital security locking system and it consist of a microcontroller, a keypad, a LCD display, a sensor and as well as USB GSM terminal. Microcontroller is the core of the project and all components is connected to it and main task is process here. With limited pin of a 28 pin microcontroller, the project is done by utilize all the available port. Port A has five I/O pins and two will be use for RFID based locking system. Two I/O pins is connected to external crystal for more accurate operation. With only one Port A pins left, the pin is use to connect to magnetic sensor.

Port B has total 16 I/O port and most of the pin will be use for LCD and keypad. Four common pin will be share by LCD and keypad to maximize the utilization of I/O pins and these pins are RB0 to RB3. RB4, RB7, RB8 and RB9 are use for keypad for input detection pin. These pins cannot be shared to avoid any error in decoding keypad. RB13 and RB14 are connected to LCD as the Register Select and Enable bit of LCD. These pins must no share with any other component to avoid error. RB5 is use to connect to buzzer which is use to indicate as an alarm siren and RB15 is connected to a transistor and a relay to drive the magnetic lock on and off.

Figure 4.14 shows the schematic diagram of digital security locking system and Figure 4.15 shows the PCB layout of digital security locking system.

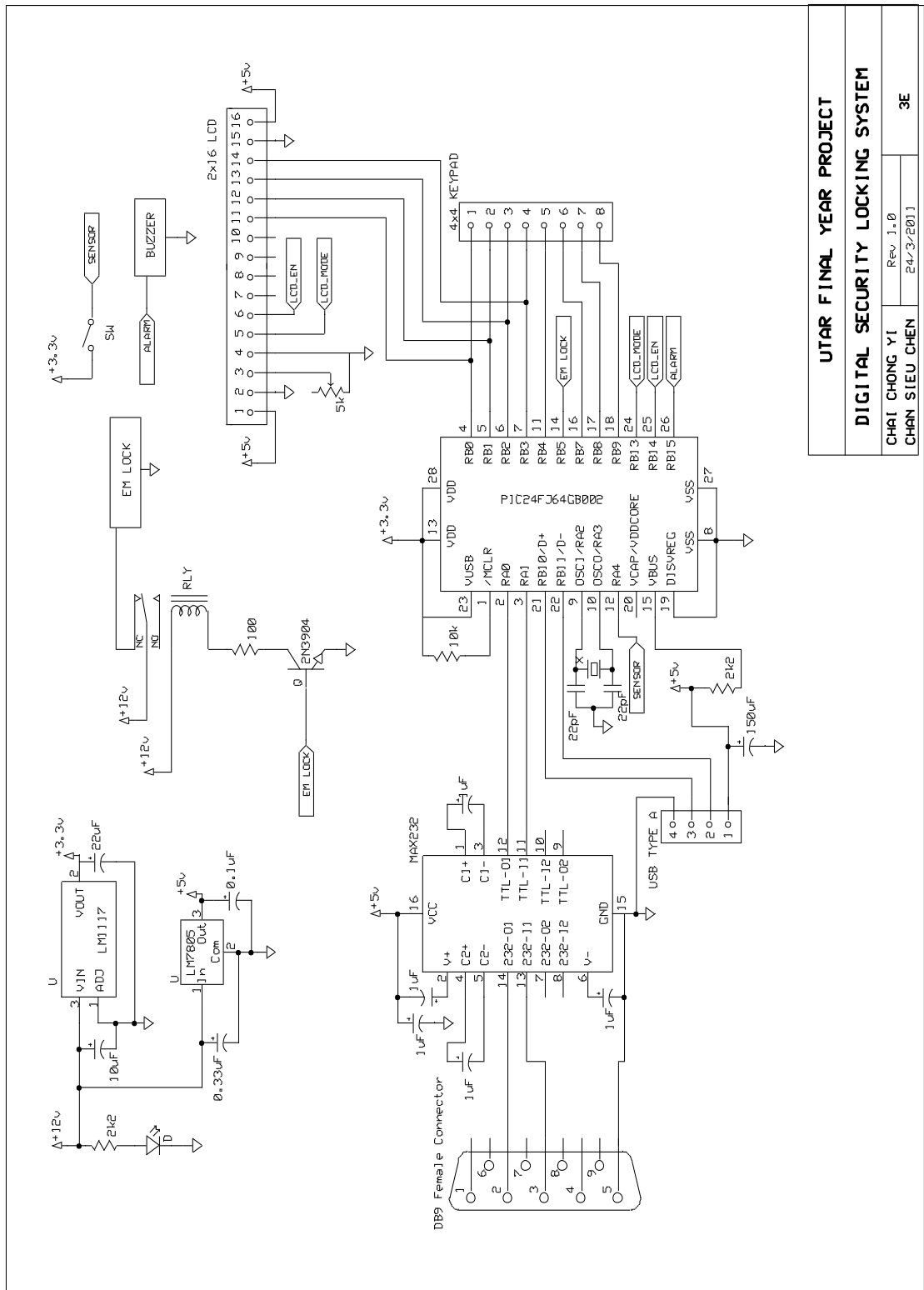


Figure 4.15: Circuit Diagram of Digital Security Locking System

<b>UTAR FINAL YEAR PROJECT</b>	
<b>DIGITAL SECURITY LOCKING SYSTEM</b>	
CHAI CHONG YI	Rev. 1.0
CHAN SIEU CHEN	24/3/2011
9E	

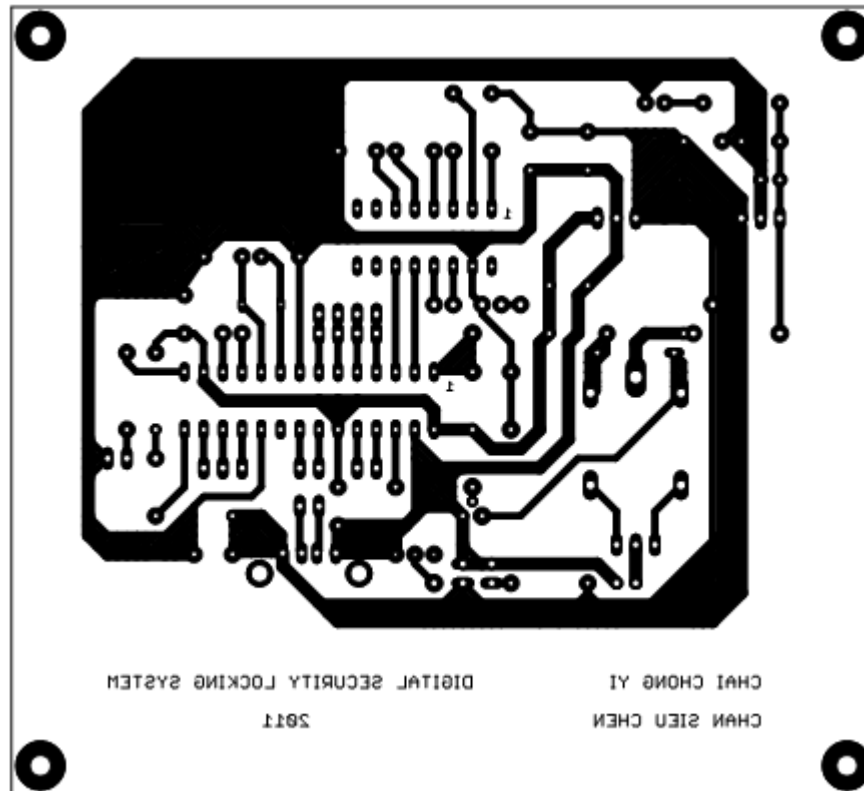


Figure 4.16: PCB Layout of Digital Security Locking System

#### 4.6 Problem Encounter

Most of the project will not go smoothly where some problem may be encountered during the process. The first problem encountered in this project is the USB enumerations. Enumerations process is quite complex in which requires deep knowledge of USB structure to make it right. As for the result, the USB host driver released by Microchip is used in the project to cope with this problem.

Second problem arises when other GSM terminals are used in the project instead of Sony Ericsson C902. The microcontroller fails to send data to the GSM terminal, which results in a USB error. This problem is due to the endpoint used by the embedded generic host driver provided by Microchip, which only utilizes Endpoint 1 for data transfer, which is an isochronous transfer. This problem is soon solved by adding another transfer type to the embedded host, which is bulk transfer that utilizes Endpoint 2.

Third problem is on the input pin on microcontroller that connected to keypad as well as LCD. Some input pin must be properly pulled to ground to avoid any floating state when microcontroller tries to read the state of the input pin. Unexpected input will generated when the input of microcontroller is in floating state. High value of resistor is used to connect between these pin and ground to provide proper grounding.

Forth problem is regarding the voltage regulator. Initially the design is cascading two voltage regulator to provide 5 V and 3.3 V from a 12 V source. The 12 V source is first regulated to 5 V and the further down to 3.3 V. The 12 V supply is use to power magnetic lock and 5 V supply is use for MAX232 IC for RS-232 communication between RFID reader and microcontroller and USB device. As for 3.3 V supply, it is use in USB as well as microcontroller itself. Cascading two voltage regulators may work fine if the current drawn is large enough to avoid too low of dropout voltage which will cause the voltage regulator fail to work. When the circuit is placed on PCB, the voltage regulator fails to operate properly. As such, two different voltage regulators is use in which both of them are connected directly to source.



## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATIONS**

#### **5.1 Conclusion**

As a conclusion, the aim and objective of this final year project had been achieved in which both RFID based digital locking system and USB based GSM security system have successfully integrated to form a digital security locking system.

This digital locking system is a low cost, low power, simple operation, small size, and standalone system. Digital locking system reads RFID tag or reads from keypad which will then unlock the door when the tag or password is matched. This system is able to change password, adding tag to access list, change the period the system is being blocked if password entered wrong for five times, change target phone number and change access types. Even the system is turned off, this information can still keep in memory and these setting will restored when system starts up next time.

Besides, this system can send SMS to user when security is breached or password is being entered wrong for five times. Alarm will be turn on whenever door is forced to open without first enter password or scan tag on the reader. Alarm can be turn off by enter correct password.

## 5.2 Limitation and Future Enhancement

The number of phone model that support AT command is considerable. However, most of them only support PDU mode instead of both PDU and Text mode. To decode the message in PDU mode, the program is complex. As such, if cannot decode the PDU string, microcontroller cannot response according to what message the GSM terminal has received. To enhance the project, it is desirable to write a program to decode PDU string so that this system can be controlled remotely.

To keep track on when and how door is unlocked, a Real-Time Clock is needed. There are real time clock inside the microcontroller but the precision is the most important thing. This can also be done through connecting the whole project to the computer and let the computer do the tracking.

Besides, this system can further enhance when combining with different types of sensor using wireless sensor such as Bluetooth sensor. All sensors using a single node will save all the pins required by wired sensor and a sensor network can be developed. Many different types of alert can be sent to user through SMS which makes the system even more powerful on monitoring surrounding.

At last, the system can also integrate with home automation system to create more complete home automation system. This system can provide home automotive as well as home security. User can keep track on most of the information at home when they are not around.

## REFERENCES

- Ahmed, A., Ali, J., Raza, A. & Abbas, G. (2007). Wired vs wireless deployment support for wireless sensor networks. Retrieved August 9, 2010 from <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4142257&tag=1>
- Bodic, G. L. (2005). *Mobile Messaging Technologies and Services SMS, EMS and MMS* (2nd ed.). Chichester: John Wiley & Sons, Ltd
- Cao, S. Q. (2009). GSM Modem-Based Mobile Auxiliary Learning System. 2009 International Conference on Computational Intelligence and Software Engineering, 1, 1-4.
- Developer's Home. (2004-2010). Short Message Service / SMS Tutorial. Retrieved July 1, 2010, from <http://www.developershome.com/sms/>
- Gregory, J. (2007). A short history of burglar alarms. Retrieved August 1, 2010 from <http://ezinearticles.com/?A-Short-History-of-Burglar-Alarms&id=475162>
- Huang, H. P., Xiao, S. S., Meng, X. Y. & Xiong, Y. (2010). A remote home security system based on wireless sensor network and GSM technology. 2010 second international conference on networks security, wireless communications and trusted computing, 1, 535-538.
- Jasio, L. D. (2007). *Programming 16-Bit Microcontroller in C - Learn to Fly the PIC24*. Oxford: Elsevier Inc.
- Kim, Y. S., Kim, H. S. & Lee, C. G. (2005). The development of USB home control network system. 2004 8<sup>th</sup> international conference on control, automation, robotics and vision, 1, 289-293.
- Wan, S. F., Zhang, Q. Q. & Fang, M. F. (2008). SMS Receiving and Dispatching System Based on Embedded Linux. 2008 ISECS International Colloquium on Computing, Communication, Control, and Management, 2, 475-479.
- Zhao, Y. & Ye Z. (2008). A low cost GSM/GPRS based wireless home security system. *IEEE Transactions on consumer electronics*, 54 (2), 567-572.

## APPENDICES

### APPENDIX A: Main Program Code

```

#include <stdlib.h>
#include "GenericTypeDefs.h"
#include "usb_config.h"
#include "USB/usb.h"
#include "USB/usb_host_generic.h"
#include "DEE Emulation 16-bit.h"

// Configuration Bits
// *****
_CONFIG1(WDTPS_PS1 & FWPSA_PR32 & WINDIS_OFF & FWDTEN_OFF & ICS_PGx1 &
GWRP_OFF & GCP_OFF & JTAGEN_OFF)
_CONFIG2(POSCMOD_HS & I2C1SEL_PRI & IOL1WAY_OFF & OSCIOFNC_ON &
FCKSM_CSDCMD & FNOSC_PRIPLL & PLL96MHZ_ON & PLLDIV_DIV2 & IESO_ON)
_CONFIG3(WPFP_WPFP0 & SOSSEL_IO & WUTSEL_LEG & WPDIS_WPDIS &
WPCFG_WPCFGDIS & WPEND_WPENDMEM)
_CONFIG4(DSWDTPS_DSWDTPS3 & DSWDTOSC_LPRC & RTCOSC_LPRC &
DSBOREN_OFF & DSWDTEN_OFF)

// Global Variables
// *****
#define SLEN 10
BYTE deviceAddress; // Address of the device on the USB
unsigned int USBOn = 0, currstate = 0, retry = 0, alarm = 0, offalarm = 0, halt = 0, wrong = 0,
halttime = 0, cardkey = 0;
char Home[150];
char buff[4] = {'A','T',13,'\0'};
char buff0[12] = "AT+CMGS=52";
char buff1[123] =
"07910621000010F511000B910621520291F1000AA2BD3F2B82E4FD3F3A020BB2CA7835AA0
AA3B5CA7A3DFF2B4BE4C0691DF6F39E80D2FBBD3EE3308";
char buff2[12] = "AT+CMGS=64";
char buff3[147] =
"07910621000010F511000B910621520291F1000AA39D3F2B82E4FD3F3A020BB2CA7835AA0
ABFCED3E83A0E1F9FCFE9693414537BD2C2F9341CDB7BC0CA2A2C36E500D444DB7CB73";

// Function Prototype
void delay(unsigned int msec);
void initt2(unsigned int time);
void initt1(void);
void initU1(void);
void writeU1(char c);
void writesU1(char *str);

```

```

char readU1(void);
char *readsU1(char *str, unsigned int slen);
void accesslcd(void);
void writelcd8b(unsigned int fnc, char text);
void writelcd4b(unsigned int fnc, char text);
void writeslcd (unsigned int fnc, char *string);
void accesskey(void);
char readkey(void);
void initlcd(void);
void DataInit(void);
BOOL InitializeSystem (void);
void readcard(void);
BOOL CheckForNewAttach ( void );
BOOL USB_ApplicationEventHandler ( BYTE address, USB_EVENT event, void *data, DWORD
size );
void SendAlert(void);
void SendAlert1(void);
void RWFlash(unsigned int task);
void GetPass(unsigned int Add);
void GetNum(unsigned int Add);
void GetTime(void);
void setup(void);
void grandaccess (void);
void blockaccess (void);
unsigned int verifykey(unsigned int Add);
unsigned int disarm(unsigned int Add);

```

// Interrupt Routines

```

void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void);
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void);
void __attribute__((interrupt, no_auto_psv)) _T3Interrupt(void);
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void);

```

```

void _ISR_U1RXInterrupt(void)

```

```

{
    IEC0bits.U1RXIE = 0;    // Disable U1RX interrupt
    readcard();
    IFS0bits.U1RXIF = 0;    // Reset U1RX interrupt register
    IEC0bits.U1RXIE = 1;    // Enable U1RX interrupt
}

```

```

void _ISR_CNInterrupt(void)

```

```

{
    if (PORTAbits.RA4 == 0 && offalarm == 0){
        TRISBbits.TRISB5 = 0;
        PORTBbits.RB5 = 1;
        alarm = 1;
        SendAlert();
    }

    if (PORTAbits.RA4 == 1 && offalarm == 1){
        offalarm = 0;
    }
    while (alarm){
        if (disarm(200)){
            if (alarm == 1 && PORTAbits.RA4 == 1){
                offalarm = 0;
            }
        }
    }
}

```

```

        else{
            offalarm = 1;
        }
        alarm = 0;
    }
}
IFS1bits.CNIF = 0;
}

void _ISR_T3Interrupt(void)
{
    halt = 0;
    IFS0bits.T3IF = 0;
}

void _ISR_T1Interrupt(void)
{
    USBHostTasks();
    IFS0bits.T1IF = 0;
}
// Local Routines
//*****

/*      Function:      delay
      Overview:      This routine take time in millisecond and cause system to do nothing. */
void delay(unsigned int msec)
{
    unsigned int fac;
    for (;msec > 0;msec--)
    {
        for (fac = 16000; fac > 0; fac --);
    }
}

/*      Function:      initt2
      Overview:      This routine initializes timer 2 in 32bit operation for blockaccess function.
*/
void initt2(unsigned int time)
{
    T2CONbits.T32 = 1;          //32bit operation
    T2CONbits.TCKPS = 3;      // 1:256 Prescale
    T2CONbits.TCS = 0;        // Intrnal fosc/2
    T2CONbits.TGATE = 0;      // Disable gated timer operation
    PR3 = time;                //
    PR2 = 0;                  //
    IFS0bits.T3IF = 0;        // Clear interrupt
    IEC0bits.T3IE = 1;        // Enable interrupt
    T2CONbits.TON = 1;
}

/*      Function:      initt1
      Overview:      This routine initializes timer 1 in 16bit operation for usbhosttask. */
void initt1(void)
{
    T1CONbits.TCKPS = 3;      // 1:256 Prescale
    T1CONbits.TCS = 0;        // Intrnal fosc/2
    T1CONbits.TGATE = 0;      // Disable gated timer operation
    PR1 = 3906;                //
}

```

```

        IFS0bits.T1IF = 0;        // Clear interrupt
        IEC0bits.T1IE = 1;       // Enable interrupt
        T1CONbits.TON = 1;
    }

    /*
    Function:      initU1
    Overview:      This routine initializes the Tx/Rx pin, setting the baud rate and enabling
    interrupt on Rx pin. */
    void initU1(void)
    {
        U1BRG = 103;
        U1MODE = 0x8000;
        U1STA = 0x8400;
        IFS0bits.U1RXIF = 0;
        RPINR18bits.U1RXR = 5; // Assign U1RX To Pin RR5/RA0
        RPOR3bits.RP6R = 3;    // Assign U1TX To Pin RP6/RA1
        IFS0bits.U1RXIF = 0;    // Reset U1RX interrupt register
        IEC0bits.U1RXIE = 1;    // Enable U1RX interrupt
        IPC2bits.U1RXIP = 7;
    }

    /*
    Function:      writeU1
    Overview:      This routine takes a character as input and writes to Tx port. */
    void writeU1(char c)
    {
        while (U1STAbits.UTXBF);
        U1TXREG = c;
    }

    /*
    Function:      writesU1
    Overview:      This routine takes a string from input and writes to Tx port. */
    void writesU1(char *str)
    {
        while (*str)
            writeU1(*str++);
    }

    /*
    Function:      readU1
    Overview:      This routine reads a single character from Rx port and return the character.
    */
    char readU1(void)
    {
        while (!U1STAbits.URXDA);
        return U1RXREG;
    }

    /*
    Function:      readsU1
    Overview:      This routine reads a string from Rx port and return the string. */
    char *readsU1(char *str, unsigned int slen)
    {
        char waste;
        char *ptr = str;
        waste = readU1(); // discard 1st unused character
        do{
            *str = readU1();
            str++;
        }
    }

```

```

        slen--;
}while (slen > 0);
    waste = readU1();           // discard last unused character
    *str = '\0';               // add null terminal to string
    return ptr;
}

/*    Function:      accesslcd
    Overview:      This routine enable write to lcd and disable input from keypad */
void accesslcd(void)
{
    TRISBbits.TRISB0 = 0;
    TRISBbits.TRISB1 = 0;
    TRISBbits.TRISB2 = 0;
    TRISBbits.TRISB3 = 0;
    TRISBbits.TRISB13 = 0; //Output RB13
    TRISBbits.TRISB14 = 0; //Output RB14
    PORTB = 0x0000;
}

/*    Function:      writelcd8b
    Overview:      This routine writes a byte of data to lcd. Data is send to lcd in 4bit mode */
void writelcd8b(unsigned int fnc, char text)
{
    accesslcd();               // Switch to LCD
    delay(1);
    char temp = text;         // Save data inside temp
    temp = temp & 0xF0;      // Take upper 4 bit
    temp = temp >> 4;        // Right shift to output data
    text = text & 0x0F;      // Take lower 4 bit
    PORTB = temp;            // Output upper 4 bit first
    if(fnc)                  // Check for intruction or data
        PORTBbits.RB13 = 1;
    else
        PORTBbits.RB13 = 0;
    PORTBbits.RB14 = 1;      // Enable
    delay(1);
    PORTBbits.RB14 = 0;      // Disable
    PORTB = text;
    if(fnc)
        PORTBbits.RB13 = 1;
    else
        PORTBbits.RB13 = 0;
    PORTBbits.RB14 = 1;
    delay(1);
    PORTBbits.RB14 = 0;
}

/*    Function:      writelcd4b
    Overview:      This routine writes a 4bit of data to lcd. */
void writelcd4b(unsigned int fnc, char text)
{
    accesslcd();
    delay(1);
    PORTB = text;
    if(fnc)
        PORTBbits.RB13 = 1;
}

```



```

else
PORTBbits.RB13 = 0;
PORTBbits.RB14 = 1;
delay(10);
PORTBbits.RB14 = 0;
delay(1);
}

/*      Function:      writeslcd
Overview:      This routine writes string to lcd in 4bit mode. */
void writeslcd (unsigned int fnc, char *string)
{
    while(*string)
    {
        writelcd8b(fnc,*string);
        string++;
    }
}

/*      Function:      accesskey
Overview:      This routine enable input from keypad and disable write to lcd */
void accesskey(void)
{
    TRISBbits.TRISB0 = 0;
    TRISBbits.TRISB1 = 0;
    TRISBbits.TRISB2 = 0;
    TRISBbits.TRISB3 = 0;
    TRISBbits.TRISB4 = 1;
    TRISBbits.TRISB7 = 1;
    TRISBbits.TRISB8 = 1;
    TRISBbits.TRISB9 = 1;
    TRISBbits.TRISB13 = 1;
    TRISBbits.TRISB14 = 1;

    if (alarm)
        PORTB = 0x002F;
    else
        PORTB = 0x000F;
}

/*      Function:      readkey
Overview:      This routine scan matrix keypad for key pressed and return the key pressed
*/
char readkey(void)
{
    accesskey();
    delay(1);
    unsigned int col = 0;
    char key = '\0';
    if(PORTBbits.RB4||PORTBbits.RB7||PORTBbits.RB8||PORTBbits.RB9)
    {
        if(PORTBbits.RB4){
            col = 1;
        }
        else if (PORTBbits.RB7){
            col = 2;
        }
        else if (PORTBbits.RB8){

```

```

        col = 3;
    }
    else if (PORTBbits.RB9){
        col = 4;
    }

    if (alarm)
        PORTB = 0x0020;
    else
        PORTB = 0x0000;
    delay(1);

    switch (col)
    {
    case 1 :
    if (alarm)
        PORTB = 0x0021;
    else
        PORTB = 0x0001;
    delay(1);
    if(PORTBbits.RB4)
    {
    key = '1';
    break;
    }

    if (alarm)
        PORTB = 0x0022;
    else
        PORTB = 0x0002;
    delay(1);
    if(PORTBbits.RB4)
    {
    key = '4';
    break;
    }

    if (alarm)
        PORTB = 0x0024;
    else
        PORTB = 0x0004;
    delay(1);
    if(PORTBbits.RB4)
    {
    key = '7';
    break;
    }

    if (alarm)
        PORTB = 0x0028;
    else
        PORTB = 0x0008;
    delay(1);
    if(PORTBbits.RB4)
    {
    key = '*';
    break;
    }

    case 2 :

```

```

if (alarm)
    PORTB = 0x0021;
else
    PORTB = 0x0001;
delay(1);
if(PORTBbits.RB7)
{
key = '2';
break;
}

```

```

if (alarm)
    PORTB = 0x0022;
else
    PORTB = 0x0002;
delay(1);
if(PORTBbits.RB7)
{
key = '5';
break;
}

```

```

if (alarm)
    PORTB = 0x0024;
else
    PORTB = 0x0004;
delay(1);
if(PORTBbits.RB7)
{
key = '8';
break;
}

```

```

if (alarm)
    PORTB = 0x0028;
else
    PORTB = 0x0008;
delay(1);
if(PORTBbits.RB7)
{
key = '0';
break;
}

```

```

case 3 :
if (alarm)
    PORTB = 0x0021;
else
    PORTB = 0x0001;
delay(1);
if(PORTBbits.RB8)
{
key = '3';
break;
}

```

```

if (alarm)
    PORTB = 0x0022;
else
    PORTB = 0x0002;

```

```
delay(1);
if(PORTBbits.RB8)
{
key = '6';
break;
}

if (alarm)
    PORTB = 0x0024;
else
    PORTB = 0x0004;
delay(1);
if(PORTBbits.RB8)
{
key = '9';
break;
}

if (alarm)
    PORTB = 0x0028;
else
    PORTB = 0x0008;
delay(1);
if(PORTBbits.RB8)
{
key = '#';
break;
}

case 4 :
if (alarm)
    PORTB = 0x0021;
else
    PORTB = 0x0001;
delay(1);
if(PORTBbits.RB9)
{
key = 'A';
break;
}

if (alarm)
    PORTB = 0x0022;
else
    PORTB = 0x0002;
delay(1);
if(PORTBbits.RB9)
{
key = 'B';
break;
}

if (alarm)
    PORTB = 0x0024;
else
    PORTB = 0x0004;
delay(1);
if(PORTBbits.RB9)
{
key = 'C';
```

```

        break;
    }

    if (alarm)
        PORTB = 0x0028;
    else
        PORTB = 0x0008;
    delay(1);
    if(PORTBbits.RB9)
    {
        key = 'D';
        break;
    }
    default: break;
    }
    while(PORTBbits.RB4||PORTBbits.RB7||PORTBbits.RB8||PORTBbits.RB9);
    delay(10);
}
return key;
}

```

/\* Function:        initlcd  
 Overview:         This routine initialize port for lcd and configure lcd for system  
 initialization \*/

```

void initlcd(void)
{
    TRISB = 0xFFFF;                                 // Output RB0-RB3
    TRISBbits.TRISB13 = 0; // Output RB13
    TRISBbits.TRISB14 = 0; // Output RB14
    delay(1);
    writelcd8b(0,0x08);                            // turn off display
    delay(8);
    writelcd4b(0,0x02);                            // 4bit mode
    delay(8);
    writelcd8b(0,0x28);                            // 4bit mode
    delay(8);
    writelcd8b(0,0x0E);                            // turn on display, on cursor
    delay(8);
    writelcd8b(0,0x06);                            // Increment mode
    delay(8);
    writelcd8b(0,0x01);                            // Clear screen
    delay(8);
}

```

/\* Function:        DataInit  
 Overview:         This routine copy data from flash memory to user variable and setting password for  
 first time running \*/

```

void DataInit(void){
    unsigned int i;
    char temp;
    buff0[10] = 13;
    buff0[11] = '\0';
    buff2[10] = 13;
    buff2[11] = '\0';
    buff1[120] = 26;
    buff1[121] = 13;
    buff1[122] = '\0';
}

```

```

    buff3[144] = 26;
    buff3[145] = 13;
    buff3[146] = '\0';
    DataEEInit();
dataEEFlags.val = 0;
    temp = DataEERead(254);
    if (temp == 'Y'){
        for( i = 0; i < 12; i++){
            buff1[24+i] = DataEERead(240+i);
            buff1[4+i] = DataEERead(220+i);
            buff3[24+i] = DataEERead(240+i);
            buff3[4+i] = DataEERead(220+i);
        }
        halttime = DataEERead(234);
        cardkey = halttime = DataEERead(235);
    }
    else{
        for( i = 0; i < 6; i++){
            DataEEWrite('0',200+i);
            DataEEWrite('0',210+i);
        }
        halttime = 1;
    }
}

```

/\* Function: InitializeSystem  
 Overview: This routine initializes the processor and peripheral, setting clock speeds  
 and enabling any required features.

Return true if successful and vice versa. \*/

```

BOOL InitializeSystem (void)
{
    delay(200);
    CLKDIVbits.CPDIV = 0;
    CLKDIVbits.PLEN = 1;
    AD1PCFGL = 0xFFFF; //Set to all digital I/O
    TRISB = 0xFFFF; //Configure all PortB as input
    TRISA = 0xFFFF;
    TRISBbits.TRISB5 = 0;
    CNEN1bits.CN0IE = 1;
    IFS1bits.CNIF = 0;
    IEC1bits.CNIE = 1;
    initlcd();
    initU1();
    USBOn = 0; // Set Default demo state
    DataInit();
    return TRUE;
}

```

/\* Function: readcard  
 Overview: This routine verify card read by reader \*/

```

void readcard(void)
{
    unsigned int i, flag, num;
    char str[11],str1[11];
    readsU1(str,SLEN);
    flag = 0;
    DataEEInit();
    dataEEFlags.val = 0;
}

```

```

for( num = 0; num < 10; num++)
{
    for ( i = 1; i < 11 ; i++ )
    {
        str1[i-1]=DataEERead(num*10+i);
    }
    str1[10] = '\0';
    if(!strcmp(str, str1))
        flag = 1;
}

if (flag){
    if (cardkey == 1){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"USER PASSWORD");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        if (verifykey(200)){
            TRISBbits.TRISB15 = 0; //Output RB15
            PORTBbits.RB15 = 1;
            delay(3000);
        }
        else{
            TRISBbits.TRISB15 = 0; //Output RB15
            PORTBbits.RB15 = 0;
        }
    }
    else{
        writelcd8b(0,0x01);
        writeslcd(1,"ACCESS GRANTED");
        TRISBbits.TRISB15 = 0; //Output RB15
        PORTBbits.RB15 = 1;
        delay(3000);
    }
}
else{
    writelcd8b(0,0x01);
    writeslcd(1,"ACCESS DENIED");
    TRISBbits.TRISB15 = 0; //Output RB15
    PORTBbits.RB15 = 0;
}
}

```

/\*      Function:        CheckForNewAttach  
       Overview:       This routine checks to see if a new device has been attached. If it has, it records the address. \*/

BOOL CheckForNewAttach ( void )

```

{
    if (deviceAddress == 0)
    {
        GENERIC_DEVICE_ID DevID;

        #ifdef USB_GENERIC_SUPPORT_SERIAL_NUMBERS
            DevID.serialNumberLength = 0;
            DevID.serialNumber = NULL;
        #endif
    }
}

```

```

if (USBHostGenericGetDeviceAddress(&DevID))
{
    deviceAddress = DevID.deviceAddress;
    #ifdef DEBUG_MODE
    #endif
    return TRUE;
}
return FALSE;
}

// USB Support Functions
//*****
/* Function:    USB_ApplicationEventHandler
   Overview:    This routine is called by the Host layer or client driver to notify the
                application of events that occur. If the event is recognized, it is
                handled and the routine returns TRUE.  Otherwise, it is ignored
(or
                just "sniffed" and the routine returns FALSE. */
BOOL USB_ApplicationEventHandler ( BYTE address, USB_EVENT event, void *data, DWORD
size )
{
    #ifdef USB_GENERIC_SUPPORT_SERIAL_NUMBERS
    BYTE i;
    #endif

    // Handle specific events.
    switch (event)
    {
        case EVENT_GENERIC_ATTACH:
            writesU1( "Generic demo device attached\r\n" );
            if (size == sizeof(GENERIC_DEVICE_ID))
            {
                deviceAddress = ((GENERIC_DEVICE_ID *)data)->deviceAddress;
                USBOn = 1;

                writelcd8b(0,0x01);
                writelcd8b(0,0x80);
                writeslcd(1,"DEVICE OK");
                writelcd8b(0,0xC0);
                writeslcd(1,"ADDRESS = ");
                writelcd8b(1,deviceAddress+0x30);

                #ifdef USB_GENERIC_SUPPORT_SERIAL_NUMBERS
                for (i=1; i<(((GENERIC_DEVICE_ID *)data)->serialNumberLength; i++)
                {
                    writesU1( ((GENERIC_DEVICE_ID *)data)->serialNumber[i] );
                }
                #endif
                writesU1( "\r\n" );
                return TRUE;
            }
            break;

        case EVENT_GENERIC_DETACH:
            deviceAddress = 0;
            USBOn = 0;

            writelcd8b(0,0x01);
            writeslcd(1,"DEVIDE DETACHED");
            delay(300);
    }
}

```



```
    return TRUE;

case EVENT_GENERIC_TX_DONE:
case EVENT_GENERIC_RX_DONE:
    return TRUE;

case EVENT_VBUS_REQUEST_POWER:
    return TRUE;

case EVENT_VBUS_RELEASE_POWER:
    return TRUE;

case EVENT_HUB_ATTACH:
    writelcd8b(0,0x01);
    writeslcd(1,"HUB DETECTED");
    return TRUE;
    break;

case EVENT_UNSUPPORTED_DEVICE:
    writelcd8b(0,0x01);
    writeslcd(1,"DEVICE ERROR");
    return TRUE;
    break;

case EVENT_CANNOT_ENUMERATE:
    writelcd8b(0,0x01);
    writeslcd(1,"ENUMERATE FAIL");
    return TRUE;
    break;

case EVENT_CLIENT_INIT_ERROR:
    writelcd8b(0,0x01);
    writeslcd(1,"CLIENT ERROR");
    return TRUE;
    break;

case EVENT_OUT_OF_MEMORY:
    writelcd8b(0,0x01);
    writeslcd(1,"OUT OF MEMORY");
    return TRUE;
    break;

case EVENT_UNSPECIFIED_ERROR:
    writelcd8b(0,0x01);
    writeslcd(1,"UNKNOW ERROR");
    return TRUE;
    break;

case EVENT_SUSPEND:
case EVENT_DETACH:
case EVENT_RESUME:
case EVENT_BUS_ERROR:
    return TRUE;
    break;

default:
    break;
}
return FALSE;
}
```

```

/* Function:      SendAlert
  Overview:      This routine send sms to user when door is forced to opened */
void SendAlert(void)
{
    BYTE RetVal;
    unsigned int EP = 0;
    USBHostTasks();
    if (USBOn == 1)
    {
        while (USBHostGenericTxIsBusy(deviceAddress));
        RetVal = USBHostGenericEP2Write(deviceAddress, buff, 3);
        if (RetVal == USB_SUCCESS){
            EP = 1;
        }
        else{
            EP = 0;
        }

        if (currstate == 0){
            while(USBHostGenericTxIsBusy(deviceAddress));
            if(EP == 1){
                RetVal=USBHostGenericEP2Write(deviceAddress, buff0, 11);
            }
            else{
                RetVal=USBHostGenericWrite(deviceAddress, buff0, 11);
            }
            currstate = 1;
            delay(100);
        }
        if (currstate == 1)
        {
            while(USBHostGenericRxIsBusy(deviceAddress));
            if (EP == 1){
                RetVal=USBHostGenericEP2Read(deviceAddress,Home,
sizeof(Home));
            }
            else{
                RetVal=USBHostGenericRead(deviceAddress,Home,
sizeof(Home));
            }
            currstate = 2;
        }

        if (currstate == 2){
            while(USBHostGenericTxIsBusy(deviceAddress));
            if(EP == 1){
                RetVal=USBHostGenericEP2Write(deviceAddress, buff1, 122);
            }
            else{
                RetVal=USBHostGenericWrite(deviceAddress, buff1, 122);
            }
            currstate = 3;
            delay(1000);
        }

        if (currstate == 3)
        {
            while(USBHostGenericRxIsBusy(deviceAddress));
            if (EP == 1){

```

```

RetVal=USBHostGenericEP2Read(deviceAddress,Home,
sizeof(Home));
    }
    else{
RetVal=USBHostGenericRead(deviceAddress,Home,
sizeof(Home));
    }
    currstate = 0;
}
}
}

/* Function:    SendAlert1
Overview:      This routine send sms to user when password entered was wrong 5 times */
void SendAlert1(void)
{
    BYTE RetVal;
    unsigned int EP = 0;
    USBHostTasks();
    if (USBOn == 1)
    {
        while (USBHostGenericTxIsBusy(deviceAddress));
        RetVal = USBHostGenericEP2Write(deviceAddress, buff, 3);
        if (RetVal == USB_SUCCESS){
            EP = 1;
        }
        else{
            EP = 0;
        }

        if (currstate == 0){
            while(USBHostGenericTxIsBusy(deviceAddress));
            if(EP == 1){
                RetVal=USBHostGenericEP2Write(deviceAddress, buff2, 11);
            }
            else{
                RetVal=USBHostGenericWrite(deviceAddress, buff2, 11);
            }
            currstate = 1;
            delay(100);
        }
        if (currstate == 1)
        {
            while(USBHostGenericRxIsBusy(deviceAddress));
            if (EP == 1){
                RetVal=USBHostGenericEP2Read(deviceAddress,Home,
sizeof(Home));
            }
            else{
                RetVal=USBHostGenericRead(deviceAddress,Home,
sizeof(Home));
            }
            currstate = 2;
        }

        if (currstate == 2){
            while(USBHostGenericTxIsBusy(deviceAddress));
            if(EP == 1){
                RetVal=USBHostGenericEP2Write(deviceAddress, buff3, 146);
            }
            else{

```

```

        RetVal=USBHostGenericWrite(deviceAddress, buff3, 146);
    }
    currstate = 3;
    delay(2000);
}

if (currstate == 3)
{
    while(USBHostGenericRxIsBusy(deviceAddress));
    if (EP == 1){
        RetVal=USBHostGenericEP2Read(deviceAddress,Home,
sizeof(Home));
    }
    else{
        RetVal=USBHostGenericRead(deviceAddress,Home,
sizeof(Home));
    }
    currstate = 0;
}
}
}
}

```

/\* Function: verifykey

Overview: This routine add/remove card unique ID from database \*/

```

void RWFlash(unsigned int task){
    unsigned int i = 1;
    unsigned int num;
    char str[11] = "*****";
    IEC0bits.U1RXIE = 0; // Disable U1RX interrupt
    do{
        num = readkey();
        USBHostTasks();
    }
    while(num == '\0' || num == 'B');
    writeU1(num);
    writelcd8b(1,num);
    if (task == 1){
        writelcd8b(0,0x01);
        writeslcd(1,"PLEASE SCAN CARD");
        readsU1(str,SLEN);
    }
    DataEEInit();
    dataEEFlags.val = 0;
    num -= 0x30;
    while(i < 11 && USBOn == 1){
        DataEEWrite(str[i-1],num*10+i);
        i++;
    }
    writelcd8b(0,0x01);
    writeslcd(1,"DONE");
    IFS0bits.U1RXIF = 0; // Reset U1RX interrupt register
    IEC0bits.U1RXIE = 1; // Enable U1RX interrupt
}

```

/\* Function: GetPass

Overview: This routine get password from user \*/

```

void GetPass(unsigned int Add){
    unsigned int i = 0;
    char Pass[7];
    char Temp;

```

```

IEC0bits.U1RXIE = 0; // Disable U1RX interrupt
while( i < 6 ){
    Temp = readkey();
    if(Temp != '\0' && Temp != 'A' && Temp != 'B' && Temp != 'D' ){
        Pass[i] = Temp;
        if (Pass[i]!='C'){
            writeU1(Pass[i]);
            writelcd8b(1,Pass[i]);
            i++;
        }
        else{
            i = 0;
            writelcd8b(0,0x01);
            writelcd8b(0,0x80);
            writeslcd(1,"ENTER PASSWORD");
            writelcd8b(0,0xC0);
            writelcd8b(1,>');
        }
    }
}
i = 0;
DataEEInit();
dataEEFlags.val = 0;
while(i < 6){
    DataEEWrite(Pass[i],Add+i);
    i++;
}
writelcd8b(0,0x01);
writeslcd(1,"DONE");
DataEEWrite('Y',254);
IFS0bits.U1RXIF = 0; // Reset U1RX interrupt register
IEC0bits.U1RXIE = 1; // Enable U1RX interrupt
}

```

/\* Function:     GetNum

Overview:     This routine get target phone number & SMSC number from user \*/

```

void GetNum(unsigned int Add){
    unsigned int i = 0;
    char Num[13];
    char Temp;
    Num[11] = 'F';
    Num[12] = '\0';
    char str1[12];

    IEC0bits.U1RXIE = 0; // Disable U1RX interrupt
    while( i < 11 ){
        Temp = readkey();
        if(Temp != '\0' && Temp != 'A' && Temp != 'B' && Temp != 'D' ){
            Num[i] = Temp;
            if (Num[i]!='C'){
                writeU1(Num[i]);
                writelcd8b(1,Num[i]);
                i++;
            }
            else{
                i = 0;
                writelcd8b(0,0x01);
                if (Add == 220){

```

```

        writelcd8b(0,0x80);
        writeslcd(1,"ENTER SMSC NUM.");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
    }
    else{
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER PHONE NUM.");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
    }
}
}
}

for(i = 0; i < 12; i+=2){
    Temp = Num[i];
    Num[i] = Num[i+1];
    Num[i+1] = Temp;
}
i = 0;
DataEEInit();
dataEEFlags.val = 0;
while(i < 13){
    DataEEWrite(Num[i],Add+i);
    i++;
}
writelcd8b(0,0x01);
writeslcd(1,"DONE");
i = 0;
while(i < 12){
    str1[i]=DataEERead(Add+i);
    buff1[24+i] = DataEERead(240+i);
    buff1[4+i] = DataEERead(220+i);
    i++;
}
writesU1(str1);
IFS0bits.U1RXIF = 0;    // Reset U1RX interrupt register
IEC0bits.U1RXIE = 1;   // Enable U1RX interrupt
}

```

```

/* Function:    GetTime
  Overview:    This routine get blocked time from user */
void GetTime(void){
    unsigned int i = 0, t = 0;
    char time[3];
    char Temp;

    IEC0bits.U1RXIE = 0;    // Disable U1RX interrupt
    while( i < 2 ){
        Temp = readkey();
        if(Temp != '\0' && Temp != 'A' && Temp != 'B' && Temp != 'D'){
            time[i] = Temp;
            if (time[i]!='C'){
                writeU1(time[i]);
                writelcd8b(1,time[i]);
                i++;
            }
        }
        else{

```

```

        i = 0;
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"MINUTES TO BLOCK");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
    }
}

t = ((time[0]-0x30) + ((time[1]-0x30)*10))*60;
DataEEInit();
dataEEFlags.val = 0;
DataEEWrite(t,234);
writelcd8b(0,0x01);
writeslcd(1,"DONE");
IFS0bits.U1RXIF = 0; // Reset U1RX interrupt register
IEC0bits.U1RXIE = 1; // Enable U1RX interrupt
}

/* Function:    GetType
   Overview:    This routine get access type from user */
void GetType(void){
    unsigned int i = 0;
    char Temp;

    IEC0bits.U1RXIE = 0; // Disable U1RX interrupt
    while( i < 1 ){
        Temp = readkey();
        if(Temp == '1' || Temp == '0'){
            if (Temp == '1'){
                cardkey = 1;
            }
            else{
                cardkey = 0;
            }
            i++;
        }
    }
    DataEEInit();
    dataEEFlags.val = 0;
    DataEEWrite(cardkey,235);
    writelcd8b(0,0x01);
    writeslcd(1,"DONE");
    IFS0bits.U1RXIF = 0; // Reset U1RX interrupt register
    IEC0bits.U1RXIE = 1; // Enable U1RX interrupt
}

/* Function:    setup
   Overview:    This routine configure user password, admin password, target phone number ,
               add/remove card, setting blocked time, setting access type and
               user SMSC number*/
void setup(void){

    unsigned int i = 1;
    char keypad;
    writelcd8b(0,0x01);
    writelcd8b(0,0x80);
    writeslcd(1,"SETUP MENU:");
    writelcd8b(0,0xC0);

```

```

writelcd8b(1,'>');

while (i){
    keypad = readkey();

    if (keypad == '1'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER CARD ID");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        RWFlash(1);
        i = 0;
    }
    else if (keypad == '2'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER CARD ID");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        RWFlash(2);
        i = 0;
    }
    else if (keypad == '3'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER PASSWORD");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        GetPass(200);
        i = 0;
    }
    else if (keypad == '4'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER PASSWORD");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        GetPass(210);
        i = 0;
    }
    else if (keypad == '5'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER PHONE NUM.");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        GetNum(240);
        i = 0;
    }
    else if (keypad == '6'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"ENTER SMSC NUM.");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        GetNum(220);
        i = 0;
    }
    else if (keypad == '7'){

```



```

        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"MINUTES TO BLOCK");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        GetTime();
        i = 0;
    }
    else if (keypad == '8'){
        writelcd8b(0,0x01);
        writelcd8b(0,0x80);
        writeslcd(1,"TAG & PASSWORD?");
        writelcd8b(0,0xC0);
        writelcd8b(1,'>');
        GetType();
        i = 0;
    }
    else if (keypad == '*'){
        i = 0;
        writelcd8b(0,0x01);
        writeslcd(1,"CANCELLED");
    }
}

}

/* Function:    grandaccess
   Overview:    This routine release magnetic door */
void grandaccess (void){
    TRISBbits.TRISB15 = 0; //Output RB15
    PORTBbits.RB15 = 1;
    IEC1bits.CNIE = 0;
    offalarm = 1;
    delay(3000);
    IEC1bits.CNIE = 1;
    if (PORTAbits.RA4 == 1 && offalarm == 1){
        offalarm = 0;
    }
}

/* Function:    blockaccess
   Overview:    This routine block access after 5 times trying */
void blockaccess (void){
    initt2(halttime);
    halt = 1;
    writelcd8b(0,0x01);
    writeslcd(1,"ACCESS BLOCKED");
    SendAlert1();
    while (halt);
    writelcd8b(0,0x01);
    writeslcd(1,"ACCESS UNBLOCKED");
    T2CONbits.TON = 0;
    wrong = 0;
}

/* Function:    verifykey
   Overview:    This routine verify the password entered by user */
unsigned int verifykey(unsigned int Add){
    unsigned int i = 0;
    char temp;

```

```

char pass[7];
char input[7];

while(i < 6){
    temp = readkey();
    if(temp != '\0' && temp != 'A' && temp != 'B' && temp != 'D'){
        input[i] = temp;
        if (input[i]!='C'){
            writeU1(input[i]);
            writelcd8b(1,input[i]);
            i++;
        }
        else{
            i = 0;
            writelcd8b(0,0x01);
            if (Add == 200){
                writelcd8b(0,0x80);
                writeslcd(1,"USER PASSWORD");
                writelcd8b(0,0xC0);
                writelcd8b(1,'>');
            }
            else{
                writelcd8b(0,0x80);
                writeslcd(1,"ADMIN PASSWORD");
                writelcd8b(0,0xC0);
                writelcd8b(1,'>');
            }
        }
    }
}
input[6] = '\0';
DataEEInit();
dataEEFlags.val = 0;
for( i = 0; i < 6; i++){
    pass[i] = DataEERead(Add+i);
    pass[6] = '\0';
}
if(!strcmp(input, pass)){
    writelcd8b(0,0x01);
    writeslcd(1,"ACCESS GRANTED");
    i = 1;
    wrong = 0;
}
else{
    writelcd8b(0,0x01);
    writeslcd(1,"ACCESS DENIED");
    i = 0;
    wrong += 1;
}
return i;
}

```

/\* Function: disarm

Overview: This routine is use to turn off the alarm \*/

```

unsigned int disarm(unsigned int Add){
    unsigned int i = 0;
    char temp;
    char pass[7];
    char input[7];

```

```

while(i < 6){
    USBHostTasks();
    temp = readkey();
    if(temp != '\0' && temp != 'A' && temp != 'B' && temp != 'D'){
        input[i] = temp;
        if (input[i]!='C'){
            i++;
        }
        else{
            i = 0;
        }
    }
    input[6] = '\0';
    DataEEInit();
dataEEFlags.val = 0;
    for( i = 0; i < 6; i++){
        pass[i] = DataEERead(Add+i);
        pass[6] = '\0';
    }

    if(!strcmp(input, pass)){
        i = 1;
        writelcd8b(0,0x01);
        writeslcd(1,"ALARM DISARM");
    }
    else{
        i = 0;
    }
    return i;
}

```

/\* Function: main

Overview: This is the Application's main entry point \*/

int main ( void )

```

{
    char keypad;

    if ( InitializeSystem() != TRUE )
    {
        writelcd8b(0,0x01);
        writeslcd(1, "SYSTEM FAILURE");
        while (1);
    }
    if ( USBHostInit(0) == TRUE )
    {
        writeslcd(1, "BOOTING UP.....");
    }
    else
    {
        writelcd8b(0,0x01);
        writeslcd(1, "USB FAILURE");
        while (1);
    }
    delay(500);
    initt1();
    while (1)

```

```

{
    while (USBOn == 1 || USBOn == 2){
        if (wrong > 4){
            blockaccess();
        }
        keypad = readkey();

        if (keypad == 'B')
        {
            writelcd8b(0,0x01);
            writelcd8b(0,0x80);
            writeslcd(1,"USER PASSWORD");
            writelcd8b(0,0xC0);
            writelcd8b(1,'>');
            if (verifykey(200)){
                writelcd8b(0,0x01);
                writelcd8b(0,0x80);
                writeslcd(1,"ADMIN PASSWORD");
                writelcd8b(0,0xC0);
                writelcd8b(1,'>');
                if (verifykey(210)){
                    setup();
                }
            }
        }
        else if (keypad == 'D'){
            writelcd8b(0,0x01);
            writelcd8b(0,0x80);
            writeslcd(1,"USER PASSWORD");
            writelcd8b(0,0xC0);
            writelcd8b(1,'>');
            if (verifykey(200)){
                grandaccess();
            }
        }
    }
    writelcd8b(0,0x01);
    writelcd8b(0,0x80);
    writeslcd(1,"PLEASE PLUG IN");
    writelcd8b(0,0xC0);
    writeslcd(1,"GSM TERMINAL");
    while(USBOn == 0){
        if (wrong > 4){
            blockaccess();
        }
        keypad = readkey();

        if (keypad == 'A'){
            writelcd8b(0,0x01);
            writelcd8b(0,0x80);
            writeslcd(1,"USER PASSWORD");
            writelcd8b(0,0xC0);
            writelcd8b(1,'>');
            if (verifykey(200)){
                writelcd8b(0,0x01);
                writelcd8b(0,0x80);
                writeslcd(1,"ADMIN PASSWORD");
                writelcd8b(0,0xC0);
                writelcd8b(1,'>');
                if (verifykey(210)){

```

```
        USBOn = 2;
    }
}
}
}
}
return 0;
}
```