Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

ZigBee-Based System for Remote Monitoring and Control of Switches

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Engineering

at Massey University, Albany,

New Zealand.

© Matthew Lyon

October 2010

Abstract

Home automation technology has existed for nearly four decades, but is nonetheless mostly absent in the average home today. The systems that do exist are often highly customised and expensive, catering to a very niche market, or overly sophisticated and complicated. Many of these also require extensive, dedicated cabling as their communications backbone and as such are only practical to install during the construction of a new house.

The core aims of this project are to develop a cheap and simple home automation system that can be easily installed in new and existing houses. These aims are achieved by creating a centralised system where most of the intelligence is managed by a PC server and the end nodes are kept as simple as possible.

The server is responsible for basic security, maintaining awareness of the current system state and providing the user interface. At the outer edge of the system is a ZigBee network of wall switches and, in between, a home gateway provides a protocol translation service between the two. The new, “smart” switches are designed to be entirely compatible with existing wall switches in terms of their mounting and wiring requirements, and so ZigBee is chosen to provide a reliable wireless communication channel between the end nodes and the gateway.

Development of the system is undertaken in three stages; design of the server software (including the user interface and server processes), design of the home gateway embedded software, and design of the hardware and embedded software of the switches.

The end result is an effective, entry-level system that provides the benefits of remote management without the need for a costly or complex infrastructure.

Acknowledgements

A number of people have made a valuable contribution to this thesis through their support, guidance and help with various technical problems.

Many thanks are extended to Jamie McIntyre for his assistance with the use of Altium, and sound advice during the design and production of the switch hardware.

I also gratefully acknowledge Reece McCarthy, long-time friend and electrician, for lending his knowledge and experience in mains wiring and existing smart house products.

To my parents, many thanks for your endless motivation and support, especially in the final stages.

And finally, this project owes much of its success to the support and guidance of supervisor, Dr Tom Moir, whose direction throughout the process of developing and writing has been hugely appreciated.

Table of Contents

Abstract 2
Acknowledgements 3
List of Tables 8
List of Figures 9
1 Introduction 10
1.1 Project Overview 10
1.2 Design Specification 11
1.2.1 Requirements 11
1.2.2 Constraints 13
1.3 Thesis Outline 13
2 Literature Review 15
2.1 Problems Facing the Home Automation Industry 15
2.2 Existing Technologies 16
2.2.1 X10 and INSTEON 16
2.2.2 CEBus and C-Bus 16
2.2.3 Vantage 17
2.2.4 Control4 17
2.2.5 KNX 18
2.3 Research Approaches 18
2.4 Standards 19
2.4.1 OSGi 19
2.4.2 ZigBee 20
2.5 Summary 21
3 System Design 22
3.1 Project Framework 22

3.1.1 Remote Switching Process 23
3.1.2 Local Switching Process 23
3.2 Communication Protocol 24
3.2.1 Command Messages 25
3.2.2 Data Messages 25
3.2.3 Notification Messages 25
3.2.4 Acknowledgement Messages 26
3.2.5 Server Disconnect Signal 26
3.2.6 Command Mode Protocol 26
3.2.7 Communication Overview 27
4 Server Design 28
4.1 User Interface 28
4.1.1 Security 28
4.1.2 User Authentication 29
4.1.3 Control Centre 30
4.1.4 Switchboard 31
4.1.5 Alert Boxes 33
4.1.6 System Management 33
4.1.7 Event History 36
4.2 Database 37
4.2.1 Users Table 37
4.2.2 Devices Table 38
4.2.3 History Table 40
5 Home Gateway Design 42
5.1 Programming the Microcontroller 42
5.1.1 Microchip TCP/IP Stack 42
5.2 TCPHandler Module 42

5.2.1 Handler States 43
5.2.2 Managing Error Situations 43
5.3 UHandler Module 45
5.3.1 Handler States 45
5.3.2 Receiving Serial Data 45
5.3.3 Notifying the Server 46
5.4 LCD Display 46
5.5 Gateway Architecture 47
6 Switch Design 49
6.1 First Switch: Hardware Design 49
6.1.1 Power Supply 50
6.1.2 LED Indicators 51
6.1.3 Programming Header 51
6.1.4 Reset Button 52
6.1.5 ZigBee Module Pin Connections 52
6.1.6 PIC18F1220 Microcontroller Pin Connections 52
6.2 First Switch: Software Design 53
6.2.1 Initialisation 53
6.2.2 Interrupt Service Routines 54
6.2.3 Main Software Loop 55
6.2.4 Design Shortcomings 56
6.3 Switch Two: Hardware and Software Refinements 57
6.3.1 Power Supply 58
6.3.2 Switch Button as External Interrupt 60
6.3.3 Switch Announcement 61
7 Development and Testing 62
7.1 High Level Overview 62

7.2 System Development and Testing 64
7.3 ZigBee Performance Testing 65
8 Conclusions and Future Work 67
8.1 Conclusions 67
8.2 Future Work 68
References 71
Appendix 74
A. Circuit Schematics 74
B. Web Server Source Code 76
B.1 login.html 76
B.2 login.php 78
B.3 mainpage.php 79
B.4 switchboard.php 80
B.5 manage_system.php 83
B.6 add_device.php 87
B.7 add_user.php 89
B.8 show_history.php 91
C. Server Processes Source Code 94
C.1 server_actioner_cgi.c 94
C.2 server_listener.c 100
D. Gateway Source Code 106
D.1 TCPHandler (TCPH.c) 106
D.2 UHandler.c 112
E. Switch Firmware Source Code 115
E.1 Switch 1 (End_device/ex1.c) 115
E.2 Switch 2 (End_device2/ex1.c) 119

List of Tables

Table 4.1: *MySQL description of auth_users table* 37
Table 4.2: *MySQL description of devices table* 38
Table 4.3: *MySQL description of history table*...41
Table 5.1: *LED variable assignments*...44

List of Figures

Figure 3.1: *Project framework* 22
Figure 3.2: *server_listener output* 24
Figure 3.3: *Space-time diagram of system communications* 27
Figure 4.1: *Main MAMP window* 28
Figure 4.2: *Certificate Warning (Internet Explorer 8)* 29
Figure 4.3: *Address bar indications for certificate error and secure browsing (Internet Explorer 8)* 29
Figure 4.4: *Login window* 30
Figure 4.5: *Control centre options for admin user* 31
Figure 4.6: *Typical switchboard display* 32
Figure 4.7: *Offline switch indication* 32
Figure 4.8: *JavaScript alert box* 33
Figure 4.9: *System management page* 34
Figure 4.10: *Devices table with completed form data* 34
Figure 4.11: *JavaScript confirmation box* 35
Figure 4.12: *Users table* 35
Figure 4.13: *Event history page* 36
Figure 5.1: *LEDs located below the LCD display* 44
Figure 5.2: *LCD display at start-up* 47
Figure 5.3: *Gateway architecture* 48
Figure 6.1: *Mounting box* 49
Figure 6.2: *PCB layout of first switch* 50
Figure 6.3: *Completed hardware assembly of first switch* 53
Figure 6.4: *General architecture of second switch* 57
Figure 6.5: *PCB layout of second switch* 58
Figure 6.6: *Circuitry removed from PowerTech Plus Plugpack* 58
Figure 6.7: *Various assemblies of the second switch* 59
Figure 7.1: *Logical system architecture* 62
Figure 7.2: *Physical system architecture* 63
Figure 7.3: *Project setup* 64

1 Introduction

1.1 Project Overview

Home automation has been around for a while. Since the arrival of X-10 in 1975 [1], there have been numerous developments and sophisticated modern systems incorporate features to manage comfort, security and entertainment. Unfortunately, along the way, the technology has not been widely embraced by consumers. The average home today contains few, if any, 'smart' features for automated or remote management.

One of the major reasons for this is that most developments have been undertaken independently. The end result is the availability of a wide variety of smart systems and products that are unable to work together, and integrating them becomes the domain of specialists who in turn must design custom, integrated solutions. This increases the complexity of the systems and drives the cost up to a level most cannot afford. As such, they are largely to be found in only a handful of "upscale" new properties [2].

The focus of this master thesis is to develop and demonstrate a much simpler home automation system that provides the convenience of remote monitoring and switching at a price point that might attract more consumers to the technology. Cost and complexity are minimised by basing as much of the system as possible on infrastructure that already exists in most homes, such as a PC and Wi-Fi/LAN network, and developing the software on a free, open-source platform. ZigBee is chosen as the communication medium for switches since it provides a robust networking solution without the need for additional wiring.

The major components of the project are a web server, home gateway and a set of switches containing ZigBee modules. Development was undertaken in three key stages, reflecting the work on each of these components. The server was originally set up on a home desktop PC, running a particular flavour of the Ubuntu operating system with LAMP installed. LAMP stands for Linux-Apache-MySQL-PHP and provides the key elements necessary to create a functional web server. Towards the end of the project, the server was shifted to an Apple MacBook provided by the university, where the equivalent platform for Mac OSX is MAMP. Development kits for the home gateway and ZigBee network were purchased online from the United States; from Microchip and Digi respectively.

During the first stage, work focused on the PC server, which provides the intelligence for the overall system and is thus the most critical component. It was also necessary at this stage to consider the wider project framework and establish a protocol for communication between devices. This framework was then tested and validated in software, using a small sockets program to emulate end devices.

The second stage involved writing embedded software for the home gateway and interfacing this with the server and ZigBee coordinator. The PICDEM.net2 development kit was chosen as the hardware platform because it sports both Ethernet and serial (USART) interfaces and is largely designed around the PIC18F97J60, which has an integrated Ethernet controller. Networking tasks are facilitated by Microchip's TCP/IP stack.

The final stage focused on the hardware and software design of the wall switches. Aside from the ZigBee module, each switch contains a PIC18F1220 microcontroller to perform the limited amount of processing necessary to respond to inputs, effect switching and communicate with the gateway. It is intended that each new switch should look and operate similarly to an ordinary wall switch, but provide the additional convenience of remote connectivity.

1.2 Design Specification

The principal aim of this project is to produce an entry-level system that might attract consumers to home automation technology through low cost and simplicity. In keeping with this philosophy, the system should require the minimum of additional infrastructure or expertise to install and use. The project is specifically designed so that the two main interfaces to the system - namely, the web interface and the switches themselves - are intuitively familiar to an average user.

1.2.1 Requirements

The server should operate independently of any particular computer or operating system. The web server platform is available in a number of versions for common operating systems; WAMP for Windows, MAMP for Mac OSX and LAMP for Linux. Moreover, the platform is open source and can be freely downloaded and installed from the internet, reducing cost.

The web interface should be intuitive. This means that switches should look and function on-screen the same way as they would in the real world. However, the user experience can be enhanced by providing additional visual cues as to the current state of devices. Also, the web interface should offer only those features that are of functional importance to the user. For a typical user, this is limited to viewing and operating switches. An administrative user benefits from the ability to add and remove devices and users from the system as necessary.

The network infrastructure should be reusable. With many homes now equipped with broadband access to the internet, it makes sense to use the same home network infrastructure to connect the PC server and home gateway. This also helps to reduce the overall cost.

Similarly, there should be no additional wiring necessary to connect switches in the home. Rather, communication between switches and the gateway should be over a suitable wireless channel and the switch control circuitry should derive its power from the mains supply already available at the wall. Equivalently, the switch should provide wiring connections similar to a standard switch.

The system should be secure. Eavesdropping and 'man-in-the-middle' attacks can be prevented by providing a secure connection to the server from the internet. ZigBee communications should also be encrypted.

The specific tasks of the project are:

- Developing the web pages for the user interface
- Designing and implementing the database
- Developing the software interfaces between the server and the home gateway
- Developing the embedded software for the PIC18F97J60, which provides the intelligence for the home gateway
- Interfacing the PIC18F97J60 with the ZigBee coordinator
- Designing and assembling the switch hardware
- Designing the embedded software for the switch microcontroller
- Integrating the various components into a reliable, working system
- Debugging and improvement

1.2.2 Constraints

To fulfil the requirements previously set out, elements of the project are subject to certain important design constraints.

Firstly, using the TCP/IP stack within the limited resources of the microcontroller, and especially keeping it abstract from the main program, requires a carefully considered programming approach. That approach is termed *cooperative multitasking* and facilitates the coordinated and timely execution of asynchronous tasks. Not only does the stack use this approach, but the main program must as well in order for the system to work effectively [3].

Secondly, with regard to the switches, the control circuitry must be safely and suitably housed, but still be compatible with a standard switch plate. Most often the switch plates are screwed to a flush box inside the wall. However, they can be screwed on top of mounting boxes of various depths, with the mounting box screwed to the wall. The mounting box chosen for the project has usable internal dimensions of 75mm x 58mm x 30mm. Thus, the printed circuit board (PCB) is restricted in size to 75mm x 58mm and the maximum height of any component mounted on it is approximately 28mm, taking into account the thickness of the PCB.

Finally, various components – namely the microcontroller, the ZigBee module and the relay – have specific power supply requirements. The PIC18F1220 operates ideally with a supply voltage between +4.2V and +5.5V [4]. The ZigBee module, on the other hand, requires only +2.8V to +3.4V [5] while the relay demands +12V. With limited circuit area available, a trade-off is made between satisfying each component's ideal requirement and reducing circuit complexity to a minimum.

1.3 Thesis Outline

Chapter 1 has provided a brief background to the current state of home automation technology and set out the purpose of this research project. It has also briefly discussed the key components of the proposed system and the design steps taken in developing it.

Chapter 2 presents information on other research work in the field of home automation. In doing so, it provides a context for this project and how it contributes to the work already done.

Chapter 3 outlines the framework of the project. It discusses the relationship between components in the system and how they interact with one another by providing a walk-through of the switching process. The chapter concludes with a description of the communication protocol used.

Chapter 4 provides details about the PC server. All aspects of the user interface are covered, including a description of the background software processes and the database structure.

Chapter 5 describes the software modules written for the home gateway and discusses how various error situations are managed.

Chapter 6 describes two iterations of the design of a wall switch prototype. The key hardware and software elements of each switch are detailed, with emphasis on how the later design improves on the first.

Chapter 7 outlines the process followed in developing each major component, as well as the integration and testing of the system.

Chapter 8 summarises the key findings of the project and examines opportunities for improvement and future development.

2 Literature Review

2.1 Problems Facing the Home Automation Industry

Home automation is not a new concept. The emergence of microprocessors in the 1970s gave rise to a new realm of possibility in this area and the first steps were taken on the path to the intelligent home of the future [6]. Yet, four decades on, the average household seems little closer to realising this dream.

A variety of reasons have been put forward for the lack of widespread smart housing. Some are valid observations about the current solutions in the market, whereas others address elements of the underlying technology itself. [7] provides an excellent summary of the first type, identifying five problem areas; cost and complexity, intrusive installation, lack of network interoperability, interface inflexibility, and safety and security. Most of these are echoed in the other literature. [2, 8-10] all observe in one way or another that existing systems offer comprehensive control options, but that these come at considerable expense and are difficult to install, especially in legacy homes. Extending these ideas, it is suggested that the cost far outweighs the features available [11] and that there is limited appreciation of the benefits that can be offered [12, 13].

Since its inception, home automation technology has progressed along independent lines. Developments in different parts of the world have led to a number of regional standards and individual systems, yet none of these has emerged as an industry leader. Consequently, the various systems available are not compatible with one another and products from different vendors are difficult to integrate. The adoption of a single, unifying standard is widely recognised as crucial for home automation to evolve and gain wider acceptance [6, 11, 14].

It is easy to infer a causal relationship between these two sets of reasons; that a lack of cooperation between industry players from the outset has resulted in a diverse offering of products that have little or no ability to interact with one another, and uniting them under a single, customised interface has become the domain of professionals, driving up the cost and complexity. Because “technology evolves very fast and solutions not well established can be supplanted by newer ones rapidly and easily” [6], perhaps this is the reason that consumers have so far been reluctant to embrace home automation technology. For most people, buying or building a home represents a significant, long-term investment. Therefore, making it reliant on expensive technology with a potentially uncertain future is an unattractive proposition.

2.2 Existing Technologies

2.2.1 X10 and INSTEON

The earliest home automation technology appears to be X10, developed in 1975 by Pico Electronics in Scotland. X10 uses power-line signalling to link and control devices; a system whereby the home's existing mains wiring provides the communications medium and data is sent in short bursts at the zero crossing of the AC signal. This provides advantages of low cost and easy installation, plus the fact that the medium is already in place supplying power to the devices that one might wish to control. Unfortunately, the mains provides an inhospitable environment for information signals due to noise from the grid and local interference caused by devices being switched on and off. Also, in instances where not all of a premises' power is supplied in phase, X10 cannot communicate over this phase gap without additional equipment. In an effort to overcome the noise problem, the bandwidth of the signal is restricted, but this results in a very low effective data rate of ~20bps and limits the usefulness of the technology to basic control functions [1]. Furthermore, X10 is considered unreliable, labelled "plug and pray" technology on account of there being no acknowledgment of commands [15]. To that end, it has been largely succeeded by INSTEON, although X10 devices are still in use and available today.

INSTEON provides compatibility with existing X10 devices, but improves communications by providing an additional network medium (RF), increasing the data rate to 2880bps and adding reliability mechanisms like acknowledgments and retries. It also supports a much larger set of devices (65536 vs. 256) and commands (65536 vs. 16) [15].

2.2.2 CEBus and C-Bus

CEBus (Consumer Electronics Bus) was born in 1984 out of a standards movement in the US to get various devices communicating using a common language over various media, including power-line, twisted pair, infra-red and RF [1]. In the mid-90's, it began to show some promise [6], although support for the standard subsequently waned [1].

In Australia and New Zealand, an equivalent system is C-Bus, developed by Clipsal Integrated Systems. It is a distributed system where devices are individually intelligent and can communicate with every other device on the network to share status information. Although wireless (RF) options are available, a wired C-Bus network requires extensive Cat-5 cabling to link devices, which needs to be integrated into the home during construction [16]. This combination of features contributes to a high overall cost, although performance and capability is greatly improved over X10 and INSTEON.

2.2.3 Vantage

The Vantage home automation system is a product of Vantage Controls, Inc., based in Orem, Utah. Vantage offers both wired and wireless solutions, but, unlike C-Bus, the system is managed by a powerful central controller. Each controller supports a limited – albeit reasonable – number of devices and controllers can be linked to extend the system if necessary. They also provide Ethernet connectivity; however, a remote interface to the system from the internet is only provided with the addition of specific software, running on a dedicated PC connected to the controller. Ultimately, Vantage caters to the luxury market [17].

In Auckland, AudioVisual Solutions promotes and sells the wired Vantage system, as well as developing their own equipment to complement it. Examples include touch-screen wall plates and control panels, which are priced around $900 and $4000-$6000 respectively.

2.2.4 Control4

Control4 was established in 2003 and is also based in Utah. Like Vantage, it is a centralised system with various controllers available, but whereas Vantage caters to a niche, affluent market, Control4 have capitalized on existing IP technology and focused on wireless solutions to make home automation more accessible to the average homeowner. The result is a system that combines ZigBee mesh networking for home control and Ethernet/Wi-Fi to deliver multimedia content [13].

2.2.5 KNX

KNX was formed in 1999 out of the convergence of three previous systems; the European Installation Bus (EIB), European Home Systems (EHS) and BatiBUS. Devices are either sensors or actuators, but all use a common language to communicate. Since each of its predecessors was designed around different communication media, KNX is able to incorporate them all and thus supports twisted pair, power-line, RF and IP/Ethernet. It is also now an international standard (ISO/IEC 14543-3) [18].

This is by no means an exhaustive list of all of the systems available, but provides an overview of those relevant to the research and work done for this project. Other systems include LonWorks, BACNet, HomePlug, ModBus, Z-Wave and EnOcean [19]. What this intends to show is the diversity of the systems available, in terms of architecture, functionality and target market, and reinforces some of the reasons suggested for limited adoption of the technology.

2.3 Research Approaches

How best to overcome these barriers is still a matter for debate and research projects have attempted to address the various shortcomings of existing technology in a variety of ways. Since each of the established systems has its own advantages and disadvantages, it is desirable to be able to combine elements of each. As a result, much focus is placed on how best to integrate these systems and still provide a simple, generic interface to the user for controlling them all.

[20] argues that a centralised architecture is problematic when trying to provide high level management for multiple heterogeneous systems because of the number of protocols and interfaces that must be handled by the central server. Instead, it promotes the use of protocol gateways that are connected by a home network backbone. Mobile software agents are dispatched throughout the network to gather status information and perform various functions, under the control of a software platform known as the agent host. [9] presents a similar approach, whereby end devices interact with “room bridges” and the bridges are in turn connected via a home LAN. Clients can access the LAN and interact with the bridges to obtain information about the sub-networks and interact with end devices.

The advantage to these approaches is that a failure of one of the gateways has no effect on the rest of the system. Indeed, another criticism of centralised solutions is that they provide a “single point of failure” [9]. However, a major disadvantage of distributed systems lies in the amount of extra equipment required, which adds to expense, and their inherent complexity.

In the articles surveyed, centralised solutions were much more common and fairly consistent in terms of their general architecture. [11] provides a typical example of a centralised system, both in terms of the overall framework and the internal structure of the server. A home PC hosts the server functions, including a web interface, hardware interface and a database to maintain state and configuration information. All of these are then linked and supervised by a management application. Thus, despite the arguments against a centralised architecture, it would appear to be the much simpler option for implementing a home automation system in terms of infrastructure, management, maintenance and hardware.

Such systems are often differentiated by the back-end network infrastructure employed or, alternatively, whether they attempt to support and integrate a variety of network types. Another point of differentiation is the user interface provided. [21] suggests that the “lack of alternative control mechanisms” is a major weakness of many systems and so provides three methods of control; remotely, via the internet or a GSM mobile phone, and locally by way of speech recognition. Some research systems have very specific interfaces. [22] requires the installation of client side software, called SMARTPC, in order to interact with end devices, and [2] provides remote control via REMOTILE, an application designed for Java-enabled mobiles only. Such targeted applications are examples of interface inflexibility, which was highlighted as a problem earlier. With the internet now a mainstay of modern life, and the rapid convergence of internet and mobile platforms, it makes much more sense to provide a standard web interface instead.

2.4 Standards

2.4.1 OSGi

The Open Systems Gateway initiative is a middleware platform intended to facilitate interoperability between applications and devices on heterogeneous networks. It is a component-based, service-oriented architecture built upon a Java Virtual Machine. The idea is to allow manufacturers, vendors, service providers and the like to develop software bundles independently of one another, which are then able to interact by sharing details about the interfaces and services they provide. By checking the platform register, a bundle can dynamically discover other bundles that offer services it may wish to consume. Components can also be reused and combined to form larger applications. To differentiate it from other standards, the OSGi Alliance claims that this framework complements them, rather than competing with them [23].

Research projects incorporating OSGi provide insight into the benefits of OSGi from different angles. [14] focuses on providing remote access to e-Services offered by home automation systems. It uses OSGi to deliver these via a residential gateway between the home network and the internet. [24] proposes an integrated home server that combines the functions of a PC, home gateway and set-top box into a single unit. The unit contains interfaces to a variety of other systems as well as processing hardware to manage and deliver broadcast and entertainment media. OSGi is suggested as the server's service gateway to allow modular installation and replacement of software. [8] proposes a flexible system infrastructure to allow the creation of smart, assistive environments for people with special needs. Focus is placed on the software representation of devices so that smart spaces can be set up programmatically. By using OSGi for its middleware, the wider system can be implemented by developing OSGi bundles.

However, OSGi is not necessarily suitable in all situations. [25] notes that "platform specificity" can be a problem, since OSGi is based on Java, and that inclusion of the Java Virtual Machine imposes code space requirements that can limit its suitability for inclusion in embedded devices.

2.4.2 ZigBee

The ZigBee Alliance is a worldwide group of companies that develops and promotes the ZigBee standard. This standard aims to simplify the integration of wireless networking capability into embedded devices and is particularly suited to home automation applications. It is based on the 802.15.4 physical radio standard and ZigBee devices generally operate within the unlicensed 2.4GHz band. Since this band is shared by a number of other wireless systems, there is potential for interference with other devices. ZigBee inherently transfers a low volume of packets, which minimises the potential for clashes in the first place, but additionally is able to operate on 16 distinct channels and employs collision avoidance techniques (CSMA-CA) to further ensure reliability. Altogether, it provides a cost-effective way to wirelessly network everyday devices and, compared to other wireless standards such as GSM, Wi-Fi and Bluetooth, it is able to offer significantly longer battery life and supports much larger networks. However, it has a raw data throughput of 250Kbps, which is more than adequate for home control applications, but is slower than Bluetooth and significantly slower than Wi-Fi [26].

As an example of how ZigBee can complement and extend a wired home automation system, [27] proposes a KNX-ZigBee gateway that performs address and data translation between the two network types. This approach allows each network, as well as the gateway, to be individually maintained. Similarly, [7] demonstrates a completely wireless home automation system based on ZigBee and Wi-Fi, with a gateway to integrate those two networks. ZigBee satisfies the low data rate requirements for home control while Wi-Fi is able meet the demands for high-speed streaming of multimedia content. [7] also cites other key advantages such as non-intrusive installation, the widespread deployment of Wi-Fi in homes[1] already, and the coexistence of ZigBee and Wi-Fi networks with minimal interference.

2.5 Summary

The literature provides a number of possible reasons for the stifled acceptance of home automation technology. Two key factors identified are the cost and complexity of available systems. Researchers have tried to address these issues using a variety of approaches; however, a centralised architecture appears to be the most common and the most favourable. Generally, this consists of a home server and/or gateway, which comprises a web server and other elements to manage the system. Given the ubiquity and popularity of the internet, it is the logical platform for the user interface. Additionally, wireless technologies such as Wi-Fi and ZigBee provide a highly effective and cost-effective network backbone over which to deliver home automation services with minimal installation difficulty. Consequently, these features form the basis of this Masters project.

[1] The article refers specifically to UK homes

3 System Design

3.1 Project Framework

As just discussed, the project framework is built upon a central server platform. Each of the end devices communicates with a central hub, or "home gateway", which is comprised of a ZigBee coordinator and Ethernet-enabled microcontroller. The user interface to the system is provided by an Apache web server, closely coupled with a MySQL database to maintain current state and history information. Lastly, the interface between the web server and the system hub is provided by two programs that run on the server; *server_actioner.cgi* and *server_listener*. Figure 3.1 provides a pictorial overview of the entire system.

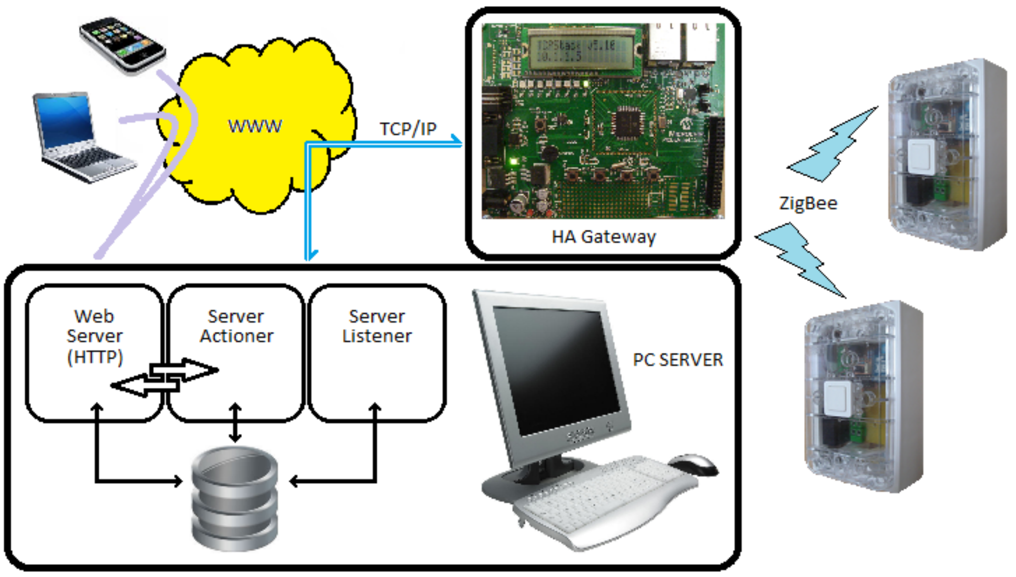


Figure 3.1: *Project framework*

For the project, the home gateway is made up of a Microchip PICDEM.net2 development board and a Maxstream development board connected via null-modem cable. The PICDEM.net2 board is designed around the PIC18F97J60 microcontroller, with an Ethernet connection, LCD display and various other inputs and outputs, and the Maxstream board holds the ZigBee coordinator module. Together they provide a data relay service, forwarding TCP/IP data from the web server to the ZigBee nodes and vice versa. The process for controlling an end device remotely is as follows.

3.1.1 Remote Switching Process

Each switch displayed on screen is actually a web form where the form data is hidden and the switch image is the submit button. The form's target is a common gateway interface (CGI) program, *server_actioner.cgi* – an executable file located in the *cgi-bin* directory on the web server. The program, written in C, uses UNIX sockets to communicate with the home gateway over TCP/IP and also uses a C API to give it access to the database. It extracts the device id from the form data and looks up the address of the ZigBee module for the corresponding device in the database.

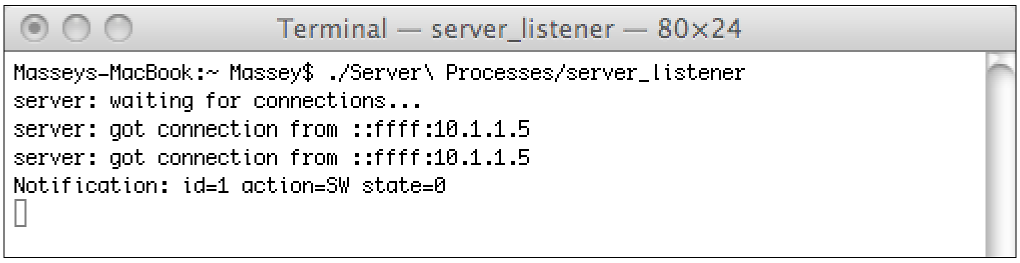
That address is sent to the home gateway as part of a control message, instructing the gateway microcontroller to place the ZigBee coordinator in *command mode* and set the coordinator's destination address. Once this has been completed successfully, an acknowledgment message is sent by the home gateway back to the server program.

Upon receipt of the acknowledgment, the server program replies with a data message containing the command to switch and the new setting, 1 for *ON* or 0 for *OFF*. No longer in command mode, the ZigBee module simply forwards the data over-the-air to the destination device where the command is carried out.

The end node responds in turn with an acknowledgment of the new state of the switch, which is picked up by the home gateway and forwarded to the server. *Server_actioner.cgi* continues by updating the database with the new switch state, recording an entry in the history table and finally redirecting the browser to the previous web page. Each time the web page loads or refreshes, it reads the current state of all devices from the database, so the new state of the switch is displayed to the user. At this point, the program has run to completion and exits.

3.1.2 Local Switching Process

So what happens if one of the switches themselves is operated by somebody at home? As mentioned at the beginning, there is a second program running on the server, *server_listener*. This program is very similar to the first, although it runs constantly in the background rather than being called by a web page. It communicates on a different port to *server_actioner.cgi* and its job is simply to listen for incoming connections from the home gateway.



```
Masseys-MacBook:~ Massey$ ./Server\ Processes/server_listener
server: waiting for connections...
server: got connection from ::ffff:10.1.1.5
server: got connection from ::ffff:10.1.1.5
Notification: id=1 action=SW state=0
```

Figure 3.2: *server_listener output*

When a switch button is pressed, that switch sends a notification message to the home gateway containing the switch id, the action performed and the new state of the switch, which can be seen in the console output in Figure 3.2. The gateway recognises this message as a notification and forwards the data on the correct port to *server_listener*. Like the CGI program, *server_listener* has access to the database and updates the devices table and the history table. However, unlike before, no acknowledgment is sent.

The reason for this is to keep the switch firmware as simple as possible. If the switch were to expect an acknowledgment, allowances would have to be made in the microcontroller code for situations where communication between the switch and the server fails. Getting data from the USART is a blocking operation, meaning that nothing else is executed in the code until the required data is received. If the server happens to be offline when the notification is sent, the switch will wait indefinitely for incoming serial data. In the meantime, it is likely to remain unresponsive.

Notifications serve the purpose of keeping the database up-to-date for remote users, but they are not critical to the functioning of the system. A person who presses the switch to turn on a light is only interested in having the light actually come on and being able to maintain manual control over that light, regardless of the state of the rest of the home network. It is possible to program in a timeout or other mechanism for dealing with a lost acknowledgment, but again the emphasis is on keeping the switch as simple as possible.

3.2 Communication Protocol

The protocol used for communications between the server, home gateway and end devices is not based on any standard. Instead, a simple protocol was established and implemented by the author to meet the basic demands of the system.

3.2.1 Command Messages

Command messages have the format “C-XXXXXXXXXXXXXXXX”, where the Xs represent a destination address to be written to the ZigBee coordinator. The address is a 16-character hex string, representing a 64-bit address that the coordinator stores in two separate fields; *Destination Address High* and *Destination Address Low*. The gateway microcontroller recognises the “C-” prefix as address data for the coordinator, rather than data for an end device. Consequently, it takes care of splitting the address string into its high and low components, and places the coordinator in *command mode*. Any serial data that the coordinator receives while in command mode is interpreted as such – a command – and not as data to be transmitted.

3.2.2 Data Messages

Data messages have the format “D-AA-V”, where AA represents an action instruction and V a value. Only two instructions have been implemented for the project; “SW”, the command to “switch”, and “ID”, the command to load a switch with its database id. Possible switch values during normal operation are 1 and 0, although 2 is used in other instances to denote a fault or “undefined” state. The valid range of ID values is 0 to 255. This requires special consideration because end devices are programmed to accept only one value character. In binary, values over this range can be represented by 8 bits, the same number of bits used to represent a single ASCII character, so the solution is to cast the id value to a character before sending it.

The gateway microcontroller strips away the “D-” prefix and hands off the data to the ZigBee coordinator. At this point, the coordinator is no longer in command mode and completely transparent to the PIC. Any serial data it receives is transmitted over-the-air to the destination ZigBee module.

3.2.3 Notification Messages

Notifications follow the format “N-I-AA-S”, where I is the device id, AA is the action code and S is the new state of the switch. There is one type of notification, however, that doesn’t follow this format. When a device comes online, it sends an advertisement message to the server. This feature was added as an afterthought, when the wider project framework was already well established, and thus it is somewhat clumsily implemented. The advertisement message contains the device address, which is really the address of the ZigBee module contained in that device, except that the first two digits of the address, which are generally “00”, are overwritten with the characters “AD” instead. The server program, *server_listener*, recognises this and replaces the zeroes when receiving the message.

3.2.4 Acknowledgement Messages

All communications are generally acknowledged by an “OK” message, indicating successful receipt of the previous message and any tasks associated with it. “ER” is used to signal a negative acknowledgement, or nACK, when an action fails. A nACK might be expected when a switch has gone offline. If a remote user commands a switch via the web page, the home gateway blindly forwards the request to the end node and waits a short amount of time for it to acknowledge. If the acknowledgment from the switch times out, the home gateway will respond to the server with “ER”. There are, of course, numerous other situations that can result in errors.

End nodes actually acknowledge with one extra piece of information – the new state of the switch. To confirm that a switch has been turned off, the acknowledgement message is “OK0”. Similarly, “OK1” confirms that a switch has been turned on. The home gateway passes on this extra information in its acknowledgement to the server.

3.2.5 Server Disconnect Signal

The final element of the protocol is the character combination “^Z”. It is used by the server to signal to the home gateway that it is about to close the TCP/IP connection. In practice, it serves no real purpose since the home gateway automatically detects when the socket is disconnected, but it is included as a kind of “goodbye” message to conclude the dialogue.

3.2.6 Command Mode Protocol

Where it is necessary to communicate directly with the ZigBee module (i.e. in command mode), those interactions are governed by a protocol outlined in the module’s datasheet. To place the module in *command mode*, the required character sequence is "+++" with no other characters transmitted before or after within a specified guard time. All commands are prefixed with AT, followed by a 2-character combination denoting a particular command. These are followed by an optional space and hex value, to set a parameter, and terminated with a carriage return, '\r' [5].

3.2.7 Communication Overview

Figure 3.3 illustrates the typical dialogue between system elements when turning on a switch. '\0' should be interpreted as the null character.

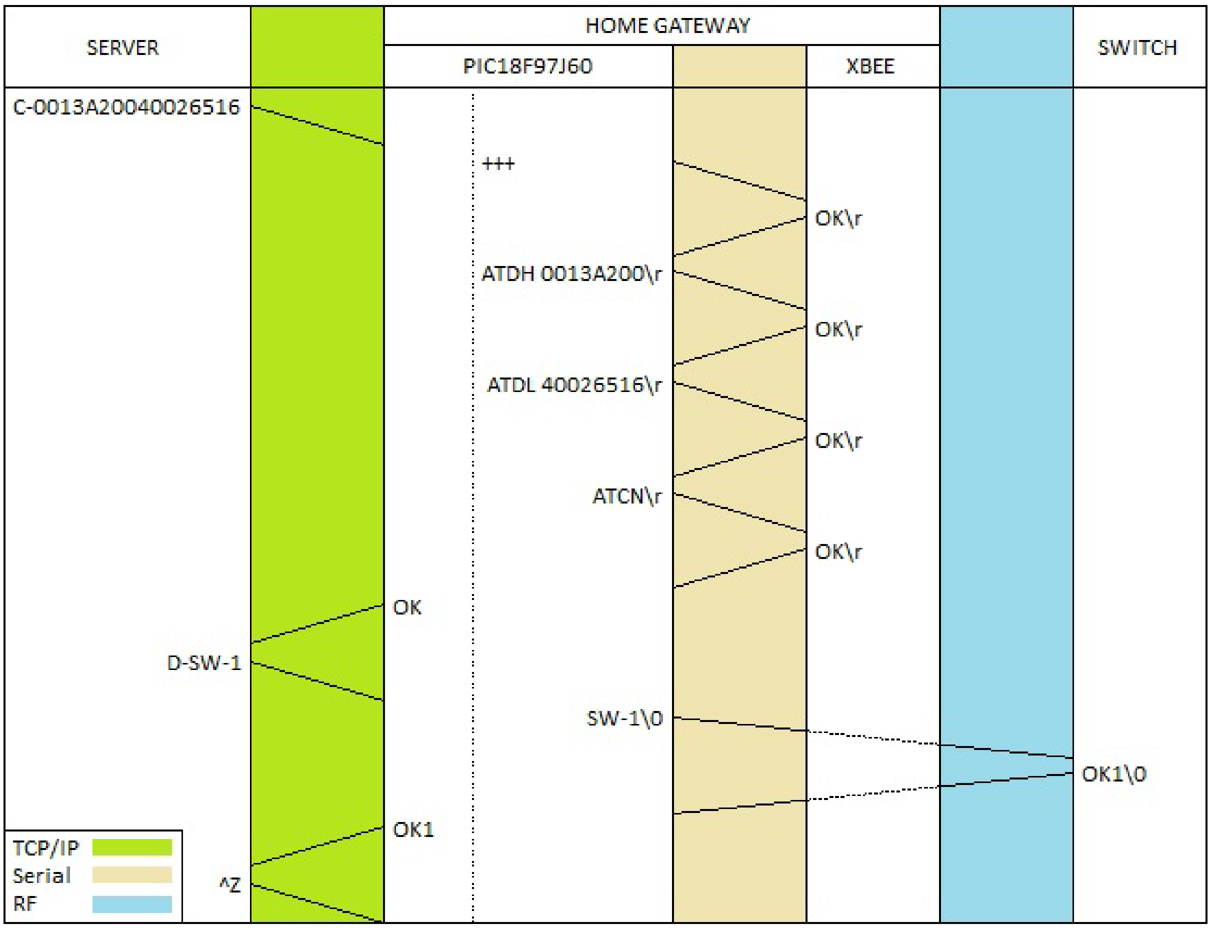


Figure 3.3: *Space-time diagram of system communications*

4 Server Design

4.1 User Interface

The user interface to the system is provided over the internet or a local area network (LAN) by a web server. Any user with a computer or web-enabled handheld device, like an iPhone, can access the system remotely. The server platform is provided by MAMP, shown in Figure 4.1, which stands for Mac-Apache-MySQL-PHP. Indeed, the project web site is hosted on an Apache server, running under the Mac OSX Leopard operating system on an Apple MacBook. All of the web pages are written in PHP and all data is stored in a MySQL database.

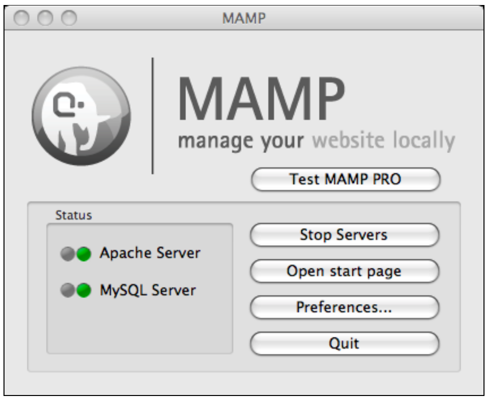
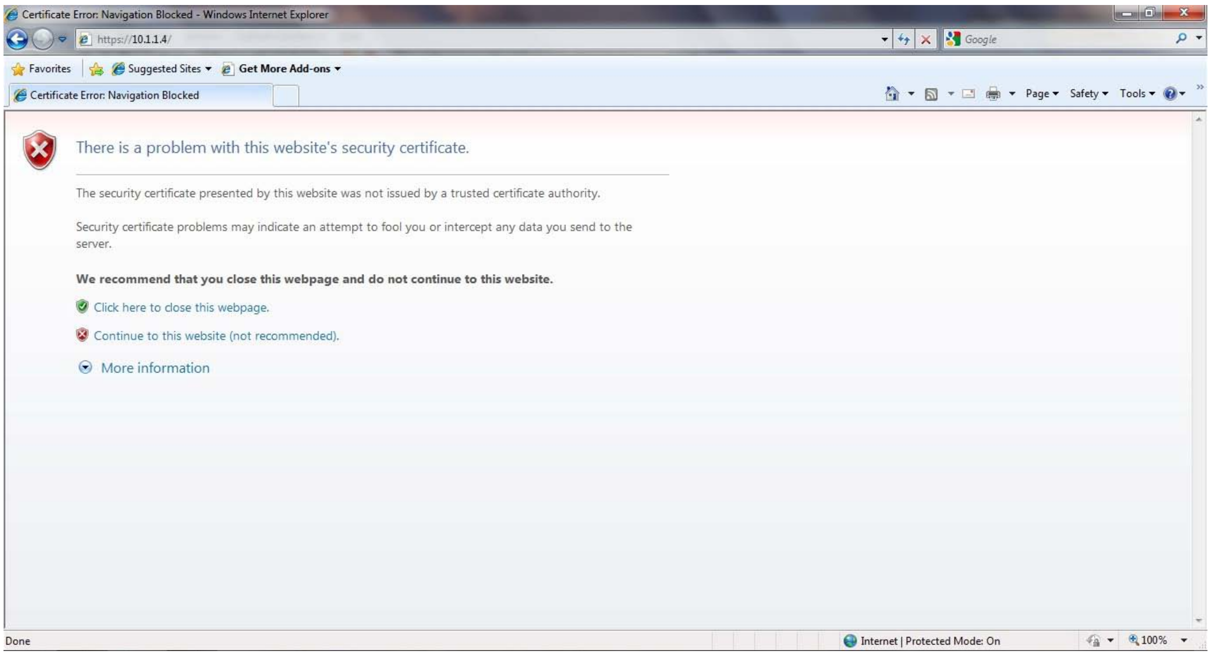


Figure 4.1: *Main MAMP window*

4.1.1 Security

The web server offers full management and control of the entire system, which requires a number of steps to safeguard its integrity. It is beyond the scope of the project to implement all of the measures necessary to guarantee system security, but the web interface does include some basic security features to protect against misuse by unauthorized or malicious users.

Firstly, the site is secured using SSL to deliver the pages over an encrypted link. Secure sites communicate on port 443, instead of port 80, and are identified by the URL prefix https:// and a padlock symbol. They use certificates to guarantee their authenticity, which would normally be issued by a trusted certificate authority for a fee, but the Mac OSX operating system has the facilities to create certificates as well. Since the issuer in this case is not trusted by default, on a user's initial visit to the site they are presented with the screen in Figure 4.2.



Certificate Error: Navigation Blocked - Windows Internet Explorer

There is a problem with this website's security certificate.

The security certificate presented by this website was not issued by a trusted certificate authority.

Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.

Continue to this website (not recommended).

More information

Figure 4.2: *Certificate warning (Internet Explorer 8)*

In the context of the project, it is safe to continue. The browser address bar continues to indicate a problem with the certificate, although this can be worked around by installing it in the browser so that the issuer is trusted. Figure 4.3, below, shows how the address bar is displayed in each of these situations.

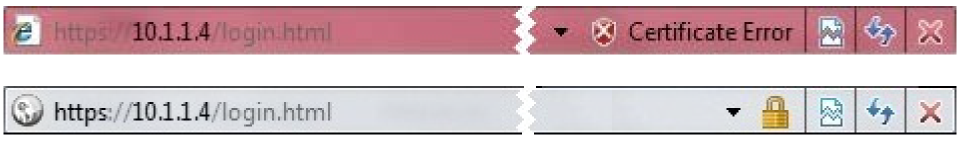


Figure 4.3: *Address bar indications for certificate error and secure browsing (Internet Explorer 8)*

4.1.2 User Authentication

The next level of security is a login screen. Authorized users can access the system with a username and password. The details for each user are stored in the database, although as a further security measure the hash value of the password is stored and not the plaintext version. Even an administrator who has direct access to the database would be unable to read it. Likewise, the password is masked on screen when the user types it in, as shown in Figure 4.4.

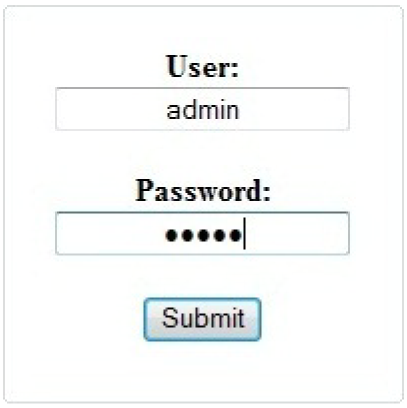


Figure 4.4: *Login window*

The system authorizes users and enforces user privileges through a combination of sessions and cookies. A session is created as part of the login process and a unique session id is generated that is stored in a session cookie on the remote machine. If the remote user provides valid login credentials, the same id is assigned as the value of an authorization cookie, also stored on the remote machine. All protected pages compare the current session id with that stored in the authorization cookie and deny access where there isn’t a match.

A failed login attempt causes the user to be returned to the login page, where they can try again. Any attempt to circumvent the login process by calling a protected page directly from the address bar also redirects the user to login, since the remote user won’t have a valid authorization cookie.

4.1.3 Control Centre

The *Control Centre* page offers switchboard access to all users, and additional management options to users with administrative privileges.

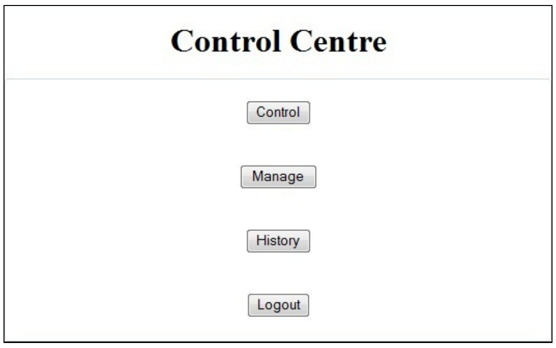


Figure 4.5: *Control centre options for admin user*

The *Control* button in Figure 4.5 links users to the switchboard page, which is discussed next. Standard users see only this and the logout button. The *Manage* button links to a page that allows administrative users to add and remove devices and users. Finally, the *History* button links to a page showing a table of past actions.

4.1.4 Switchboard

The *Switchboard* page presents users with a table of all devices in the system. Figure 4.6 shows the switchboard page for the project setup, with two available switches. At this time, the only device type is a switch and each switch is represented by a clickable image that reflects its current state. In normal use, the image changes to mimic the appearance of an actual switch as it is turned on and off, although the change is somewhat subtle. The current state is also shown below the image in plain, colour-coded text for extra clarity, and the number displayed above each switch is its database id.

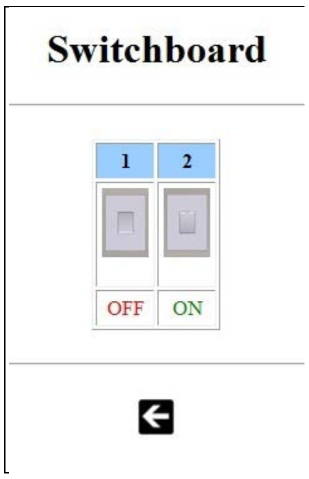


Figure 4.6: *Typical switchboard display*

4.1.4.1 Offline Switches

When a switch is first connected or added to the system, its status appears as OFFLINE and is depicted by a *stop hand* graphic, as in Figure 4.7. This indicates that the system is aware of the switch – there is an entry for it in the database – but it has not yet communicated with it to determine its current state. A switch will also appear offline if it has previously failed to acknowledge a remote request and serves as a warning to the user that there may be a fault.

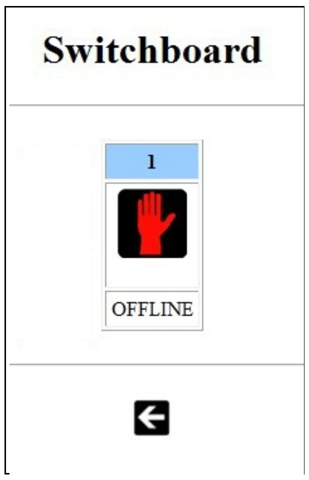
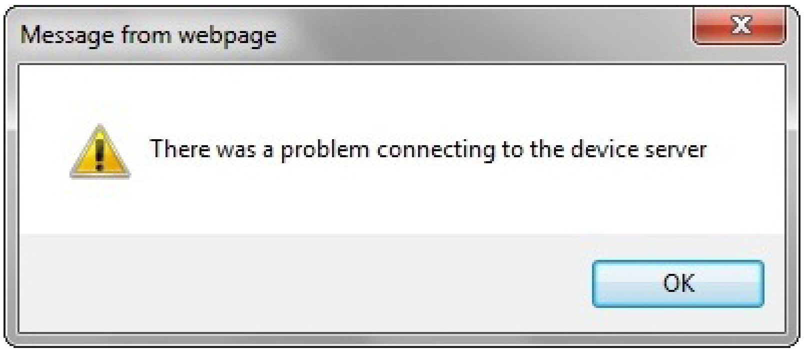


Figure 4.7: *Offline switch indication*

An attempt to establish communications with the switch is made by clicking the image. The server sends a message confirming the device id so that the switch can save it in memory, and the switch responds with its current state. If no response is received, the status remains as OFFLINE and the process can be repeated.

4.1.5 Alert Boxes

Another failure scenario could be that the home gateway goes offline, or there may be a problem accessing the database. The *server_actioner.cgi* program is able to detect many of these situations and present the user with a JavaScript alert box containing information about the nature of the problem, which can aid in troubleshooting.

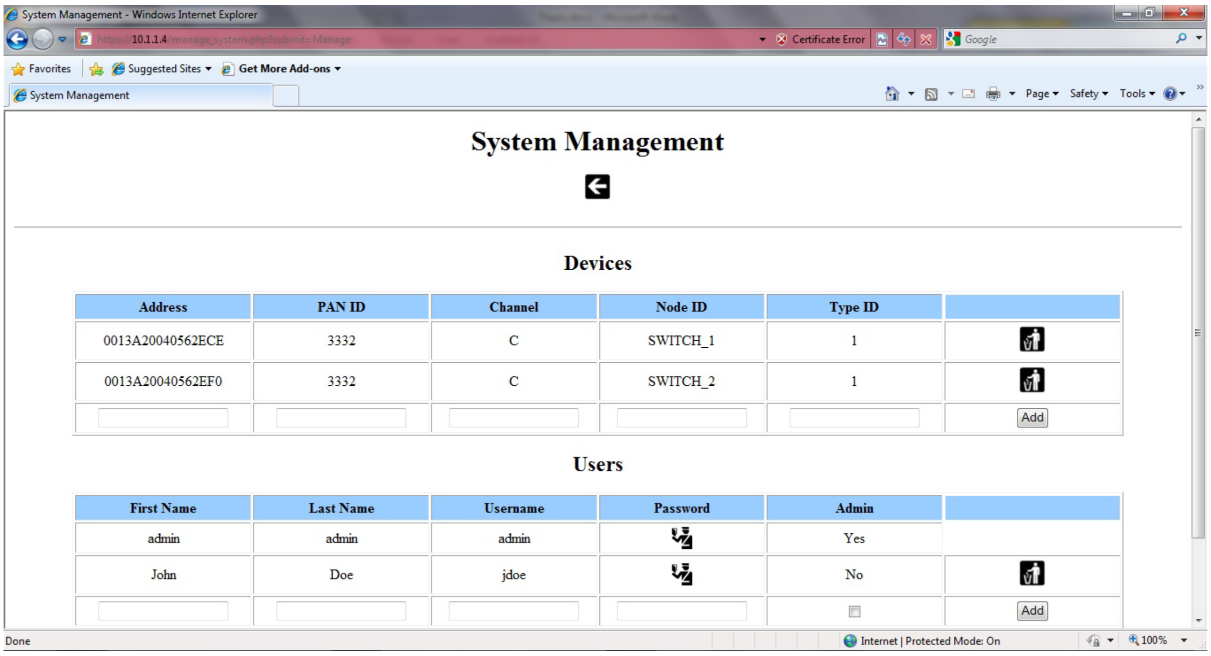


There was a problem connecting to the device server

Figure 4.8: *JavaScript alert box*

4.1.6 System Management

The *System Management* page, in Figure 4.9, displays a list of the current devices in the system and a list of authorized users. The lists are presented as tables and the last row of each table contains a form where data for a new entry can be added.



System Management

Devices

Address	PAN ID	Channel	Node ID	Type ID	
0013A20040562ECE	3332	C	SWITCH_1	1	🗑
0013A20040562EF0	3332	C	SWITCH_2	1	🗑
					Add

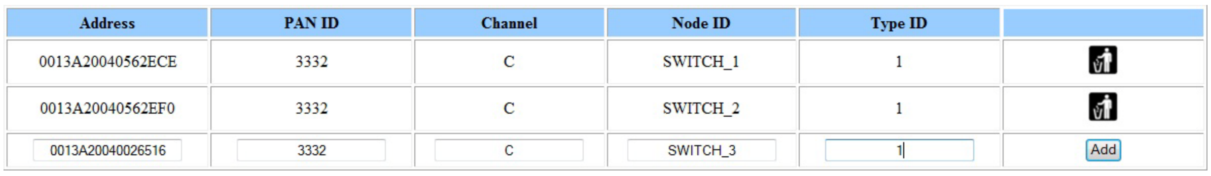
Users

First Name	Last Name	Username	Password	Admin	
admin	admin	admin	✎	Yes	
John	Doe	jdoe	✎	No	🗑
				☐	Add

Figure 4.9: *System management page*

4.1.6.1 Adding/Removing Devices and Users

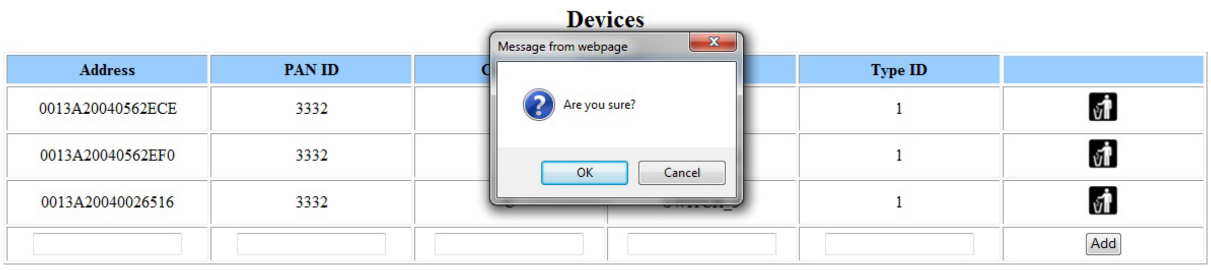
If a new device or new user is added, the actual addition to the table is performed by a separate PHP page – *add_device.php* or *add_user.php* as the case may be. Figure 4.10 shows the form fields in the third table row filled out, ready for a new device to be added. The form is checked to ensure that all of the required data has been entered and failure to complete a field results in a JavaScript alert, similar to that shown in Figure 4.8. However, at the time of writing, the fields are not yet checked to see if the data they contain is valid for that particular field, something that is equally important.



Address	PAN ID	Channel	Node ID	Type ID	
0013A20040562ECE	3332	C	SWITCH_1	1	🗑
0013A20040562EF0	3332	C	SWITCH_2	1	🗑
0013A20040026516	3332	C	SWITCH_3	1	Add

Figure 4.10: *Devices table with completed form data*

Each table entry has an icon in the last column, which is actually a button that can be clicked to remove that entry. Again, *add_device.php* and *add_user.php* are called to perform this action. As a guard against unintentional deletion, a JavaScript confirmation box appears that allows the user to confirm or cancel the action before it is committed, like that in Figure 4.11.


Devices

Message from webpage

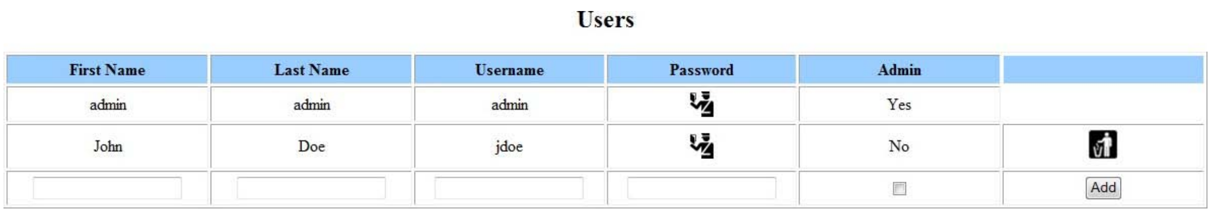
Are you sure?

Address	PAN ID	C		Type ID	
0013A20040562ECE	3332			1	🗑
0013A20040562EF0	3332			1	🗑
0013A20040026516	3332			1	🗑
					Add

Figure 4.11: *JavaScript confirmation box*

4.1.6.2 Managing Users

The *Users* table shows an icon in the password column. No feature is currently attached to this icon; rather, it is used as an indication that the password is hidden or protected. The *Admin* column displays whether or not a user can make management changes to the system and a new user is provided with administrative privileges by checking the box in the form field. Leaving it blank adds them as a standard user who can control switches via the switchboard, but not make any other changes.


Users

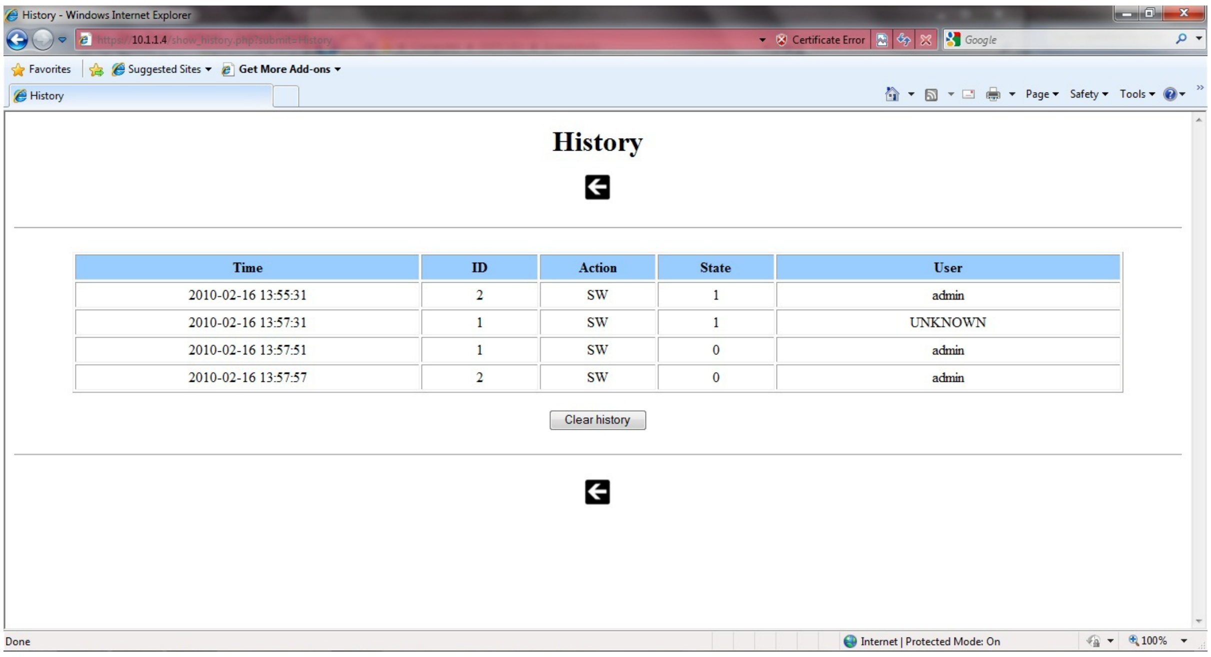
First Name	Last Name	Username	Password	Admin	
admin	admin	admin	🔑	Yes	
John	Doe	jdoe	🔑	No	🗑
				☐	Add

Figure 4.12: *Users table*

A final point of note about the *Users* table is that the "admin" entry lacks a button to remove it, as Figure 4.12 shows. This administrative user is included as a default, master user, with the intention of preventing any other administrative user from accidentally (or otherwise) deleting all users from the table and rendering the system inaccessible. Unfortunately, this approach creates a security issue. Since the "admin" user exists by default and with a default password, anybody can simply log in to the system and gain full control. One obvious solution to this problem is to allow the default password to be changed. This remains to be implemented.

4.1.7 Event History

The *History* page simply replicates the contents of the *history* table in the remote user's web browser.



History

Time	ID	Action	State	User
2010-02-16 13:55:31	2	SW	1	admin
2010-02-16 13:57:31	1	SW	1	UNKNOWN
2010-02-16 13:57:51	1	SW	0	admin
2010-02-16 13:57:57	2	SW	0	admin

History - Windows Internet Explorer
https://10.1.1.4/show_history.php?submit=History
Certificate Error
Google
Favorites
Suggested Sites
Get More Add-ons
History
Page
Safety
Tools
Clear history
Done
Internet | Protected Mode: On
100%

Figure 4.13: *Event history page*

The screenshot in Figure 4.13 shows only a few entries as an example, but there are currently no limits imposed on the amount of data that can be stored. As a result, the table can become quite large very quickly, requiring the user to scroll down to the bottom of the table to view the most recent entries. This issue will be touched upon again in a later section, but the problem is at least addressed to some degree on the web page in its present form. The back button, originally displayed only at the bottom of the page, is now included at the top as well, making it easier to navigate away from the page without having to scroll down. Also, the entire history can be deleted by pressing the *Clear history* button.

When a remote user makes a request, their username is passed to *server_actioner.cgi*. It and the other details of the request are recorded in the *History* table once the action is completed.

When a person physically operates the switch, the notification is handled by *server_listener* and recorded, though it can’t be known who has actually pressed the button. In such cases, the user is recorded as UNKNOWN.

4.2 Database

The database contains three tables; one for authorized users, one for devices and one for recording the event history.

4.2.1 Users Table

Relevant information about users that can access the system is stored in the *auth_users* table.

Table 4.1: *MySQL description of auth_users table*

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	tinyint(3) unsigned	NO	PRI	NULL	auto_increment
f_name	varchar(50)	NO		NULL	
l_name	varchar(50)	NO		NULL	
username	varchar(25)	NO	UNI	NULL	
password	varchar(75)	NO		NULL	
admin	tinyint(1)	NO		0	

The user *id* in Table 4.1 is the table’s primary key. It is of type *unsigned tinyint*, which stores 8-bit values over the range 0-255 and is the smallest practical data type available. Since the primary key must be unique, this should limit the number of users in the system to 256, but the effective limit is actually 255 because the first entry is always assigned a value of 1. Each time a new user is added, MySQL will automatically assign the next available value courtesy of the *auto_increment* parameter.

It should be pointed out, though, that *auto_increment* only ever increases the value monotonically, so that when users are deleted from the table their id numbers are not reassigned. This could prove problematic if the turnover of users is high; once the numerical limit is reached, attempting to add further users results in an error. It is also a highly wasteful use of a limited database resource.

Id numbers *can* be reused if they are manually assigned, but this adds an unnecessary additional complexity for the user as well as introducing other potential problems. Indeed, it can be argued that the *id* column is completely redundant in this context, since the *username* column must also be unique and therefore serves quite adequately as a primary key in its own right. The table structure and the way logins are processed by the server borrows heavily from [28], which includes an id as primary key, but does not impose the *unique* constraint on the username. In retrospect, the *id* column could (and probably should) be removed from here entirely.

The password field is noticeably longer than the other fields, able to take up to 75 characters. It is not anticipated that any user would ever create a password even nearly that long, but it must be remembered that the stored password is actually a cryptographic hash of the user's chosen plaintext password. Under current versions of MySQL, the *password* function generates 41-byte values [29]. The *varchar* data type is stored as one or two length bytes followed by the data [30] and so it doesn't make any appreciable difference in terms of optimization to specify *varchar(75)* as opposed to, say, *char(41).*

The admin column is the only one that specifies a default value. In the *add_user.php* code, this value is explicitly provided in the query to add a user, regardless of whether the value is 0 or 1. The default value is there only as a reminder that, in most cases, new users should be given only standard privileges.

4.2.2 Devices Table

The *devices* table stores information about each overall end device as well as the communication parameters of the ZigBee module contained within it.

Table 4.2: *MySQL description of devices table*

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	tinyint(3) unsigned	NO	PRI	NULL	auto_increment
address	varchar(16)	NO	UNI	NULL	

pan_id	varchar(4)	NO		3332	
channel	varchar(2)	NO		C	
node_id	varchar(20)	NO		NULL	
type_id	tinyint(3)	NO		0	
state	tinyint(3)	NO		2	

Table 4.2 shares some properties with the *auth_users* table, such as the *id* field as primary key. It also prescribes a second unique field for the 64-bit address of the ZigBee module. Unlike previously, the existence of the *address* field does not render the *id* field unnecessary in this case. Each end device is made aware of its corresponding database id and stores this in memory. The id is also included in all notifications to the server. Since the device id requires only one byte, as opposed to eight for the address, it is much smaller and easier to store in memory, and reduces communication overhead.

4.2.2.1 ZigBee Communication Parameters

ZigBee modules will only communicate with one another if they are on the same channel and have the same PAN ID. ZigBee devices in New Zealand operate in the 2.4GHz band, which is shared with many other RF devices, such as Wi-Fi networks and some cordless phones. Interference from such devices is always a possibility and changing channels is one way of mitigating this problem. The PAN (Personal Area Network) ID is a way of further confining the communication to a select group of devices, akin to a SSID for wireless networks.

In a typical residential installation, a homeowner would select a PAN ID for the household. A neighbouring house, if installed with the same system, would have its own household network with a different PAN ID, so there is no conflict between devices in either location. For the purposes of demonstrating the project, default values have been set of channel C and 3332 for the PAN ID.

4.2.2.2 Node Identifier

The *node_id* field is intended to store a human-friendly name for each end node that would make it readily identifiable to the user. If a switch is one of two deployed in a living room, for example, then its *node_id* might be "LivingRm SW 1". A user is free to give any name they please to a device, provided it is 20 characters or less. This restriction is in line with the ZigBee module parameter, *Node Identifier*, which holds a 20-character ASCII string. The intended purpose of this parameter is to be able to look up a node using its identifier and obtain its 64-bit address [5], similar to obtaining an IP address from a domain name via DNS resolution. This feature isn't utilized in the current project framework, but may prove useful in future development of the system.

4.2.2.3 Device Type

The *type_id* is used to identify the type of device being controlled. The only device type developed and tested for the project is a simple switch, with value 1, but as more devices are created they can be assigned their own *type_id*. Ultimately, each end device should have its *type_id* in read-only memory and send this data as part of its advertisement to the server so that the system is made aware of its capabilities. At the time of writing, however, the only manner in which this id is used is to group devices by type in a single table row on the *Switchboard* page of the web interface. 256 device types are possible, given the data type, and the default value is 0, for an "undefined" device.

4.2.2.4 Device State

The default *state* when a switch is added to the system is 2. During normal operation, a switch can be either OFF (0) or ON (1), and any other value indicates a fault or error of some kind. The value 2 was arbitrarily chosen to represent an unknown or OFFLINE state, largely because it is the next number after 1. As the data type is not unsigned on this occasion, it is possible to have negative state values and -1 could just as easily be used.

Future devices may make use of the full numerical range of the data type, from -128 to 127, to control analogue settings like temperature, dimming, or the drawing of curtains or blinds.

4.2.3 History Table

When an event occurs, a record is saved in the *History* table of the database. The table records the *id* of the switch affected, the two-letter code of the action performed, the new state of the switch and, where the action is performed remotely, which user made the request.

Each data type in Table 4.3 obviously reflects that of the corresponding field in one of the previous tables.

Table 4.3: *MySQL description of history table*

Field	Type	Null	Key	Default	Extra
time	timestamp	NO		CURRENT_TIMESTAMP	
id	tinyint(3) unsigned	NO		NULL	
action	varchar(2)	NO		NULL	
state	tinyint(3)	NO		NULL	
user	varchar(25)	NO		NULL	

4.2.3.1 Timestamps

The *time* field records the date and time at which the action was performed. More precisely, it is the timestamp of the exact moment at which the database query is made that records the event in the *History* table. Since this coincides with the performance of the action, or with the receipt of a notification immediately following an action, it is essentially the same time. The system itself has no concept of time and no value is passed for this field in the query. Instead, MySQL takes care of the timestamp automatically, hence the default.

5 Home Gateway Design

5.1 Programming the Microcontroller

At the core of the home gateway is the PIC18F97J60 microcontroller, whose sole responsibility is to interpret the information it receives and forward it to the correct destination. In doing so, it must translate between TCP/IP data coming from the server and serial data going to the ZigBee coordinator, and vice versa.

5.1.1 Microchip TCP/IP Stack

Critical to achieving this functionality is the Microchip TCP/IP Stack, “a suite of programs that provides services to... or can be used in a custom TCP/IP-based application”. The various services are layered in accordance with the TCP/IP Reference Model, where functions available at the top layer rely on the services provided by the layers below. Tasks are called upon in turn to perform whatever processing they need to in as quick a time as possible and larger tasks are subdivided so that the system is not held up for too long [3].

The PICDEM.net2 development kit shipped with stack version 3.75, which has been upgraded for the project to version 5.10. An example application is included with the software suite that demonstrates many of the features offered by the TCP/IP stack and allows various stack modules to be easily enabled and disabled from within the code. This demo application forms the basis of the home gateway software.

Two additional software modules have been created for the project, making use of the TCP/IP stack and also some of the other output features of the development board to implement the required functionality of the home gateway.

5.2 TCPHandler Module

The first of these, *TCPHandler*, is responsible for communicating with the server. It listens for connections on port 3456 and handles remote requests that come from the server via the Ethernet connection. In keeping with the cooperative multitasking paradigm, the *TCPHandler* function is set up as a finite state machine. The three possible states – home, listening and processing – are contained within in an enumerated type, *_TCPHandlerState*, which is static so that the current state persists between calls to the function.

5.2.1 Handler States

The handler begins in the *home* state when called for the first time and creates a socket on which to listen for incoming connections. Once initialized, the state is changed to *listening* and the function returns.

On subsequent calls to the function, the handler remains in the *listening* state until a connection is made. In fact, even after a connection is established, the *listening* state persists until some TCP/IP data is received and buffered. Only then is it advanced to the *processing* state.

Much of what happens during processing is covered in the earlier section dealing with protocol. The handler examines the two-character prefix of the received message to determine what kind of data it contains and then proceeds as necessary.

After a particular communication from the server has been processed and acknowledged, *TCPHandler* returns to the listening state. A single request to switch involves three separate communications from the server and each communication requires two calls to the handler – one to receive the message and one to process it. Thus, the relatively large task of operating the switch is broken into a number of smaller tasks that take less time. The entire operation requires six calls to the handler and control is returned to the *main* function after each one.

5.2.2 Managing Error Situations

Along the way, there are a number of situations possible that can give rise to errors and the handler must be able to recover from these and respond accordingly.

One easy way of signalling an error is to raise an error flag. The PICDEM.net2 development board has eight LEDs connected to various microprocessor outputs and each of these is represented by a variable in the code. An LED is controlled by setting the corresponding variable to 1 or 0, for *on* and *off* respectively. Using these variables as error or condition flags makes the job of development and debugging much easier, since the LEDs provide a visual indication of the current situation.

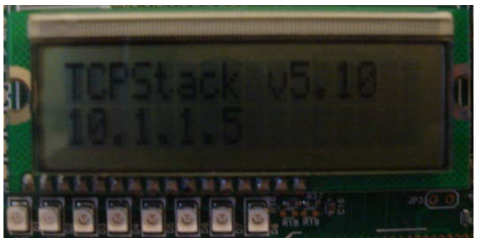


Figure 5.1: *LEDs located below the LCD display*

Figure 5.1 shows the eight LEDs arranged in a row below the LCD and, treated as a byte, the leftmost LED is bit 7 and the rightmost is bit 0. The variable names follow the format LEDx_IO, where x is the bit position. LED0 is already in use by the demo's *main* function, flashing on and off every half second when the system is operating normally. Should the program become stuck somewhere, this LED stops flashing and in doing so provides a critical error indication. Table 5.1 summarises the specific conditions indicated by LEDs 1, 2, 3 and 4.

Table 5.1: *LED variable assignments*

LED1	Connection flag	1 = TCP/IP connection active
LED2	Command mode flag	1 = Coordinator is in command mode
LED3	Coordinator error	1 = Error while communicating with coordinator
LED4	End device error	1 = Acknowledgement from end device timed out

The connection flag is set whenever the socket accepts a connection from the server and is cleared again when the server ends the connection, or if the connection is dropped.

The character sequence "+++" must be entered within certain time constraints to place the coordinator in command mode [5]. When successful, the coordinator replies with an OK message. At this point, the command mode flag is set. It is cleared when the coordinator responds OK to the *Exit Command Mode (ATCN)* command.

If at any point during the command mode dialogue the coordinator responds with an ERROR message, or if the coordinator fails to respond at all to the "+++" sequence, the coordinator error flag is set.

The error flag is used in *if-else* conditional statements to halt further processing where appropriate. For example, there is little point in sending address information to the coordinator if it has failed to enter command mode. Similarly, it is expected that neither the command mode nor coordinator error flags will be set at the point where the gateway acknowledges the server. If either is set, the gateway returns an "ER" message.

After the gateway relays data to a switch, it expects to receive an acknowledgment within two seconds. *TCPHandler* actually stops listening after this time and raises the end device error flag. Again, this results in an "ER" reply to the server.

It should be emphasized that some of these flags are set and cleared very quickly, both during normal operation and when there is a problem. Once the handler has acknowledged the server, there is no need to keep any flags set and they are indeed cleared in the process. Thus, although it is said that the choice of variables to coincide with the LED outputs is a deliberate one, the visual indications provided by the LEDs is mostly beneficial during development only. The flags are for the benefit of the handler and not the user.

5.3 UHandler Module

The second software module, *UHandler*, is also implemented as a finite state machine. Its purpose is to listen for notifications, which come via the coordinator through to the USART interface. The handler then opens a connection to *server_listener* on port 3490 and passes on the data using TCP/IP.

5.3.1 Handler States

There are four states, enumerated as *SM_LISTENING*, *SM_NOTIFYING1*, *SM_NOTIFYING2* and *SM_NOTIFYING3*. Most time is spent in the listening state, where the handler queries the USART to see if there is any data waiting. If there is, it allows one quarter second to read in the data one character at a time.

5.3.2 Receiving Serial Data

The reason for this approach, as opposed to simply reading *n* characters at a time, is that the length of the incoming string is not known in advance. A notification from a switch generally contains 8 characters, but an advertisement message has 17, including the null character. If the *getsUSART* function looks for 17 characters and receives only 8, *UHandler* will wait indefinitely for the remaining characters to arrive, effectively locking up the gateway. Conversely, if it requests only 8 characters and receives an advertisement message, that message will be treated as three separate notifications, confusing the server.

The *UHandler* routine must also adhere to the guidelines for cooperative multitasking, hence the need to impose a time limit on acquiring the message data. One quarter second is more than enough to receive either type of message and still short enough so as not to hold up the other gateway processes unnecessarily. When data *is* received, the state advances to *SM_NOTIFYING1*.

5.3.3 Notifying the Server

In this state, the handler sets up the socket to connect to the server and then sets state *SM_NOTIFYING2*. The connection is not completed yet because a number of other stack processes must also take place to help establish it. These are completely abstracted from the user and are another example of cooperative multitasking at work.

UHandler will persist in the *SM_NOTIFYING2* state over however many calls to the function it takes to complete the connection, up to a maximum of 5 seconds. Once established, the handler transfers the received serial data to the TCP/IP socket and sends it to the server. When this is finished, or if the pending connection times out, the handler changes state to *SM_NOTIFYING3* and the socket is disconnected. The state then returns to *SM_LISTENING*, ready to accept more notifications.

5.4 LCD Display

One final aspect of the two handler routines is that they both update the LCD display on the PICDEM.net2 board. The most recent event is displayed on the top line with the event before it displayed on the bottom line. As new events take place, the top line is copied to the bottom and then overwritten by the new message. Five different event types are displayed.

When an instruction comes from the server and is executed without any problem, the LCD shows “TCP Request”. If, on the other hand, the gateway does not recognise the prefix of the message data, it cannot act on any information contained in the rest of the message and so discards it. In this case, the LCD indicates that “Invalid data” was received. These messages are generated by the *TCPHandler* routine since they pertain to data coming from the server.

UHandler is responsible for echoing notifications to the display. Both general notifications and advertisement messages from an end device are displayed in the same format as they are received (see Protocol section). In the event that the handler cannot recognise the notification type from the prefix, it does not pass the information on to the server. Instead, it writes “Unhandled notif.” to the LCD.

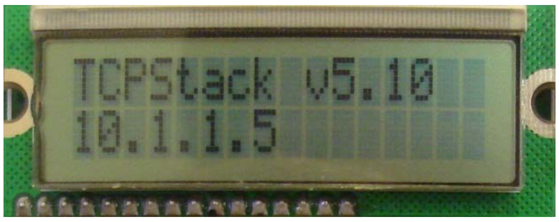


Figure 5.2: *LCD display at start-up*

Figure 5.2 shows the text displayed on the LCD when the gateway is powered on or reset. The top line shows the version of the TCP/IP stack in use and the second line displays the gateway’s current IP address. While the stack software supports Dynamic Host Configuration Protocol (DHCP), this address has been statically assigned and is hard-coded into the *server_actioner* program in order to keep the code and the setup as simple as possible.

5.5 Gateway Architecture

Finally, Figure 5.3 illustrates the overall architecture of the home gateway in terms of the functional components described in this chapter. In summary, the Microchip PIC18F97J60 and ZigBee provide the key hardware platforms. Residing on the next level are the various interfaces. The Microchip controller hosts the TCP/IP stack, which in turn provides a software interface for the two handler routines. The rest are hardware and communication interfaces, of which the serial (RS232) link is key. It bridges the communications between the server over Ethernet and the end devices over RF, and it provides a means of integrating the two hardware platforms into a single product – the home gateway.

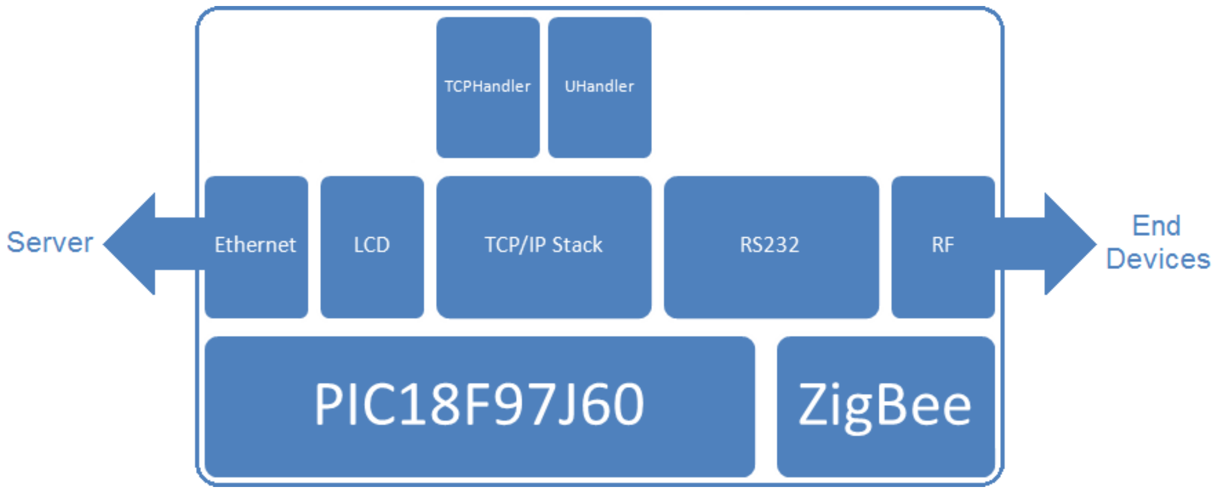


Figure 5.3: *Gateway architecture*

6 Switch Design

6.1 First Switch: Hardware Design

The overriding design constraint for the wall switch is that it must be compatible with existing mainstream switches. That way it would simply replace an existing switch without the need for any changes to the wall fittings or to the wiring. To accommodate the circuitry, a mounting box was purchased, which provides an extra 36mm space between the wall and the switch plate.

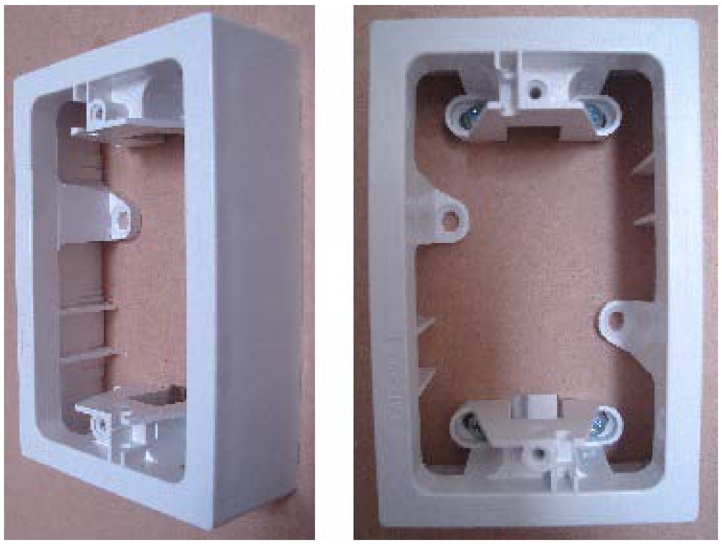


Figure 6.1: *Mounting box*

The internal dimensions of the mounting box, shown in Figure 6.1, are 73x58mm. These dimensions were used for the printed circuit board, the design of which was undertaken first on paper, then finalized using Altium. Figure 6.2 shows the completed Altium design. Additionally, the circuit schematic can be found in Appendix A.

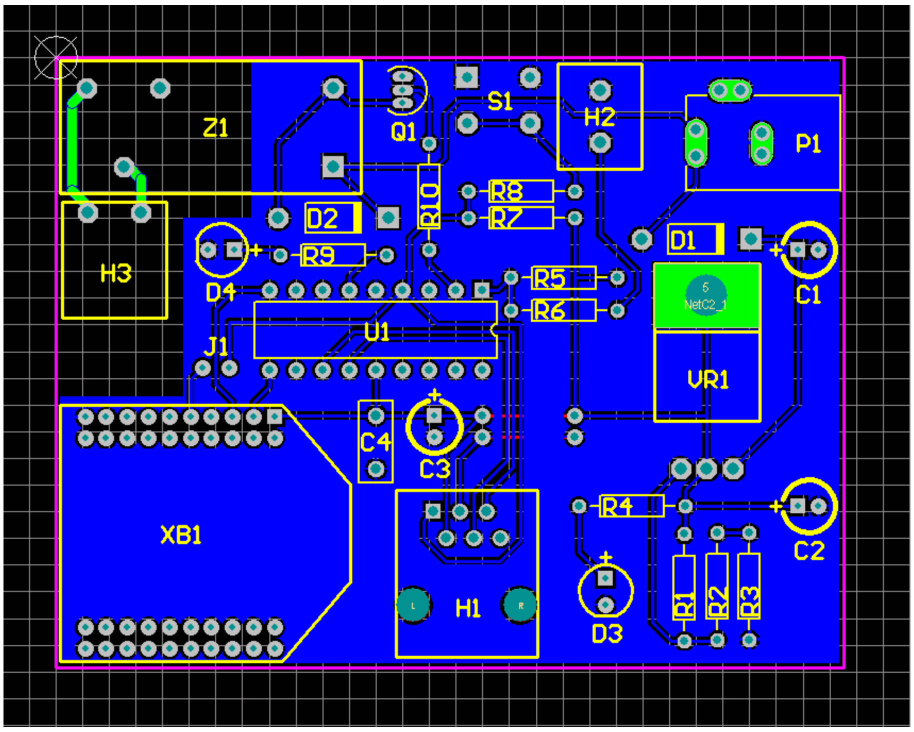


Figure 6.2: *PCB layout of first switch*

Some key features of the original design are:

- Red power indicator
- Green multi-purpose indicator
- IC programming header
- Reset button

6.1.1 Power Supply

The original design was intended as a prototype, to prove the concept as well as to provide a starting point from which to refine and further develop the design. This intention is evident by the inclusion of a 12VDC header. Power to the board is provided by a standard plug-in power supply, which would not be practical for a final product and strays from the objective of requiring only existing mains wiring to operate.

6.1.1.1 Supply Voltages

As mentioned in the introduction, the minimum supply voltage for the PIC18F1220 exceeds the upper limit specified for the ZigBee module. Fortunately, testing and practice have revealed that the PIC will still operate reliably at voltages below the recommended value and the decision was made to supply the circuit with ~3.4V in line with the ZigBee maximum. The PIC's supply voltage also determines the maximum output voltage on any of the IO pins, which is V_{DD}+0.3V, with the ability to source up to 25mA of current. Unfortunately, these parameters are not sufficient to drive the relay, which has a 12V coil and requires approximately 40mA of current.

This problem is solved by supplying the board with 12V and using a voltage regulator to reduce this to 3.35V to supply the PIC and the ZigBee. A single transistor – in this case, a 2N3904 NPN general purpose amplifier – is used to allow the smaller voltage from the output pin of the PIC to switch on the higher voltage to the relay. The output pin is connected to the gate of the transistor via a 10kΩ resistor and negligible current is drawn from the PIC.

6.1.2 LED Indicators

The red power indicator does what its name implies; it lights as soon as power is supplied to the control circuit. If this light is off, it could indicate a blown fuse or tripped circuit breaker, a power cut, a fault with the home's internal wiring, or a fault with the switch circuitry itself.

The green indicator is connected to one of the IO pins on the microcontroller, so its function can change depending on how the microcontroller is programmed. During development of the switch firmware, this indicator was made to toggle on and off to show that the microcontroller was operating normally. Alternatively, the green indicator could be used to provide visual feedback to the user of the current state of the switch.

6.1.3 Programming Header

The programming header is interfaced with the programming pins, PGC and PGD, on the microcontroller. This allows reprogramming of the microcontroller *in-situ*, using an In-Circuit debugger, such as Microchip's ICD2. Since this circuit design is based around a dual inline package (DIP) microcontroller, which can be easily removed and reinserted, being able to program the chip directly on the board is not essential. Indeed, the programming header also takes up valuable space and complicates the circuit design. If a surface-mount IC were to be used, the programming header would become a much more valuable addition, especially in the absence of some mechanism to reprogram the IC remotely. At this stage in the development, the presence of the programming header further reinforces the circuit's position as a prototype rather than a final design.

6.1.4 Reset Button

The reset button is a small momentary push-button switch that pulls the $\overline{MCLR}$ (master clear) input on the microcontroller to ground potential, thus returning the microcontroller to its power-on configuration and restarting execution of the firmware from the beginning.

6.1.5 ZigBee Module Pin Connections

The ZigBee module is connected to power and ground, and the DIN and DOUT pins are connected to the PIC for serial communication between the two devices. Provision has been made on the board to jumper a connection to the ZigBee's $\overline{RESET}$ pin so that pressing the reset button causes both the microcontroller and the ZigBee to be reset at the same time. This feature remains unimplemented, though, since it was only necessary to reset the microcontroller while testing iterations of the firmware. The ZigBee module is configured using a program called X-CTU, which allows changes to various settings, but it was not (re)programmed in any way for the project.

6.1.6 PIC18F1220 Microcontroller Pin Connections

Similarly, the PIC microcontroller is also connected to power and ground, and to the ZigBee via pins 9 and 10, which are the USART transmit and receive pins respectively. Pin 1 (RA0) is configured as an input and is connected to the wall switch, which is a low-voltage momentary push-button type. The input is normally high and pressing the switch pulls the input low.

Pin 2 (RA1) is configured as an output and drives the relay via a transistor as described above. Pin 4 is the $\overline{MCLR}$ pin, an active low input, and connected to the reset push-button as described previously also. Pin 6 (RA2) is configured as an output and connected to the green LED.

Figure 6.3 shows the completed circuitry based on this design, with each of the main components clearly visible. The reset button is located next to the blue header, which provides a connection point for low-voltage wiring coming from the wall switch. The green header provides the connection point for the mains wiring.

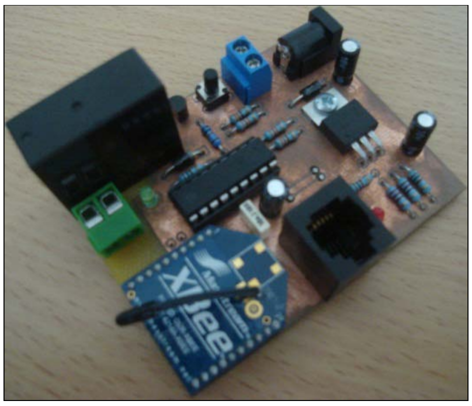


Figure 6.3: *Completed hardware assembly of first switch*

6.2 First Switch: Software Design

The PIC18F1220 has an 8MHz internal oscillator block [4]. The first few lines of firmware code define some of the operating parameters of the microcontroller, including selection of this internal oscillator as the system clock. Two interrupt service routines are defined next; one high-priority ISR for the UART receive interrupt and one low-priority ISR for the timer interrupt, though these will be covered in more detail later. A complete listing of the code can be found in the Appendix.

6.2.1 Initialisation

Since the internal oscillator can be configured for a number of different frequencies, the first two lines of the main function set it at 8MHz. Next, the PORTA register is zeroed and the various microcontroller pins that make up PORTA are configured for either input or output. All of the inputs and outputs discussed previously exist on pins that are part of PORTA and all of these pins are set up as outputs, except for RA0 and RA5. RA0 is the input from the wall switch and RA5 is on the same pin as $\overline{MCLR}$, which must *always* be an input [4]. An output pin is set (high) by writing a value of one to its corresponding PORTA bit, or cleared (set low) by writing a zero. An input value is obtained by reading the value of the corresponding bit. Since we are dealing with digital signals, either ON or OFF, the ADCON1 register is set to treat all PORTA inputs accordingly.

The *OpenUSART* function configures the PIC's serial interface. It is configured for 8-bit asynchronous communication at a baud rate of 19200 to match the communication parameters of the UART interface on the ZigBee module. Additionally, the receive interrupt is enabled. This allows the program loop to be interrupted whenever any serial data is received so that it can be acted upon immediately.

The next few lines of code are responsible for configuration of the interrupts. Since we are using more than one interrupt, one for the reception of serial data and one for the timer, the interrupt priority bit is enabled. The receive interrupt is given a high priority while the timer is given a low priority. This is important since the timer is only being used to flash the green LED to indicate that the microcontroller is working properly and is not critical to the actual switch function. In the event that the timer interrupt occurs and serial data is received, the latter will take precedence.

6.2.2 Interrupt Service Routines

At this point, the interrupt service routines will be looked at in more detail. The low priority interrupt deserves only a passing mention. When Timer0 overflows, control passes to the low priority ISR, which invokes the function, *tmr_handler*. The function causes the green indicator LED to be toggled either on or off via the output bit, RA2. It then clears the Timer0 interrupt flag before control is returned to the main program loop.

The high priority ISR invokes the function, *rx_handler*, whose first task is to capture the incoming serial data. The data is then parsed for an action command and a value. During normal operation, the expected command would be the switch command, *SW*, with a value of either 1 or 0 to switch the light on or off. Once the output has been switched accordingly, the program then responds with an acknowledgment reflecting the new switch state, or an error if the received value is not valid.

Another valid command is *ID* to set or update the switch's ID. The value corresponds to the *id* entry for that switch in the *devices* table in the database and is stored in FLASH memory at location 0x000FF0. Since FLASH memory is non-volatile, the microcontroller 'remembers' the ID even if the switch loses power. Again, the program responds with an acknowledgment of the current switch state, obtained by reading the current value of RA1. Finally, the interrupt flag is cleared and control returned to the main program loop.

It is important to note that when an interrupt occurs, further interrupts are disabled. An exception is that a high priority interrupt will interrupt a low priority interrupt, but that high priority interrupt then disables *all* further interrupts until it is finished. If a second high priority event occurs (i.e. more serial data is received) while the high priority ISR is being executed, the second incidence will be ignored.

6.2.3 Main Software Loop

The last element of the code is an infinite loop that polls for a change in the value of RA0. If RA0 is not set, this is an indication that the switch button has been pressed. The immediate response to this is to toggle the output to the relay. If the light is currently switched off, the RA1 output will be set and the relay coil energised so that the light comes on. Conversely, if the light is currently on, the output bit is cleared and both the relay and light will be turned off.

Following the change, a notification is sent via the USART to the server, announcing the ID of the switch and the new state of the switch so that the database can be updated. The ID is read from address 0x000FF in FLASH memory, where it has been previously stored.

6.2.3.1 Delay Function

The very last line of code invokes a delay of one million clock cycles following the notification. This serves two similar, although distinct, purposes. The primary purpose is to combat the effect of switch bounce. When the push-button is pressed, for a very brief period as the switch closes, the electrical connection is made and broken a number of times very quickly. The same thing happens when the switch is released. In normal situations, this is inconsequential since the transient connections occur far quicker than a human being can perceive them. However, the microcontroller is fast enough to be able to detect and react to these rapid changes. The result is that the PIC may toggle the output a number of times as a result of a single press of the button, with little actual control over the final state.

The second reason for the delay reinforces the first. Assuming that an average press of the push-button switch lasts for one tenth of a second, the PIC is able to poll and respond to the input a number of times in that period of time. Including a reasonable delay prevents the PIC from polling the switch input again while it is likely to still be depressed by the user.

6.2.4 Design Shortcomings

This switch design, ignoring the requirement for a separate power supply, has one major shortcoming. Polling the switch input within an infinite loop is not a good way of doing things and has one undesirable side-effect. Even with the delay, eventually the switch input will be polled again and if the switch is still depressed the output will be toggled once again. Keeping one's finger on the button will cause the output to alternate on and off constantly.

There is also the potential for a race condition. However unlikely it may be, it is still *possible* for somebody to send a remote command to the switch at the exact same time that a local user presses the button. Such a scenario may play out as follows.

While the light is off, the switch input is polled and a button-press detected. At that same instant, a remote user issues a command to turn on the light, which causes the main program to be interrupted. The interrupt service routine is executed, the light is switched on, and an acknowledgment is sent, confirming the on state of the switch. Finally, the interrupt flag is cleared and program flow returns to the point at which the interrupt occurred. As soon as this happens, the output is toggled once again as part of the main program loop, in response to the button having been pressed. The local user, who expects the light to be turned on, sees that it is still turned off. At the same time, the remote user is led to believe that the light has been switched on successfully, which is not the case.

Ideally, the infinite loop inside the main function of the firmware should do nothing and all actions should instead be interrupt-driven. Implementing this eliminates both problems.

6.3 Switch Two: Hardware and Software Refinements

The second switch prototype is a refinement of the first, attempting to address some of the shortcomings identified. Some design features remain unchanged, such as the voltage regulator sub-circuit to supply 3.35V to the microcontroller and the ZigBee module. The relay is also controlled the same way, using a transistor to switch on the supply voltage to the coil.

Figure 6.4 provides an abstracted view of the switch, showing the distribution of voltages and the interactions between major components. The PIC is central to the architecture, since this is where all of the firmware resides and where all inputs and outputs are handled. Manual inputs are provided by the two buttons (the wall and reset buttons) whereas data inputs are received via the ZigBee. The key outputs are to the ZigBee again, and to the relay.

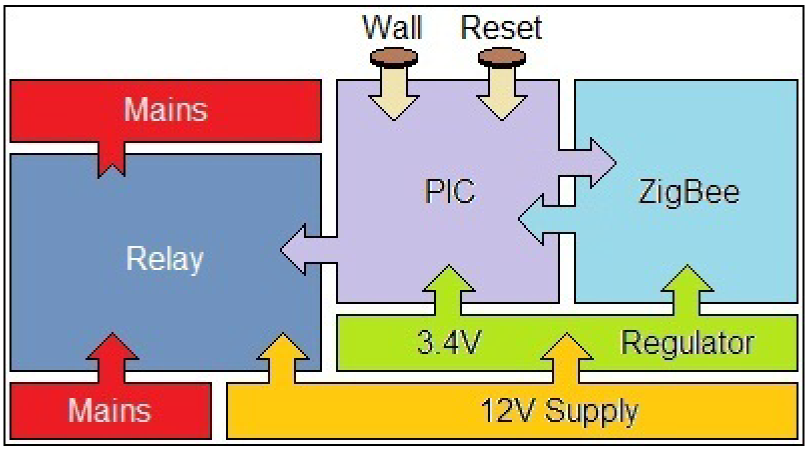


Figure 6.4: *General architecture of second switch*

Figure 6.5 shows the updated PCB layout produced using Altium. The circuit schematic is provided in Appendix A.

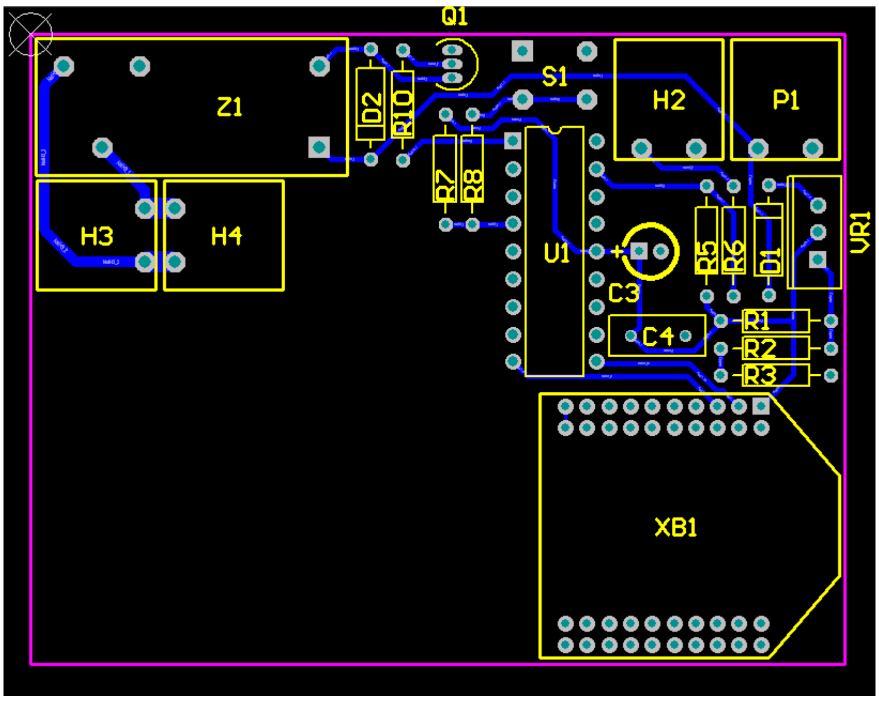


Figure 6.5: *PCB layout of second switch*

6.3.1 Power Supply

The design objective for this prototype was to make the switch circuitry fully self-contained and free from the need for an external power supply. To that end, it has a 2-terminal header in place of a DC socket (P1 in Figure 6.5) so that the 12V supply can be easily added in whatever form it takes. One idea is to use the innards of a switched-mode regulated plug-pack and one such plug-pack is the PowerTech Plus brand, which has a regulated output of 12VDC rated up to 400mA. The internal components of the plug-pack, shown in Figure 6.6, are small enough to fit neatly alongside the rest of the switch circuitry inside the wall mounting block, and the voltage and current ratings are more than adequate for the demands of the prototype.

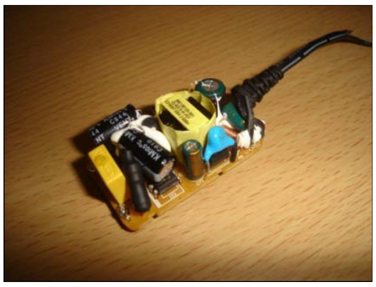


Figure 6.6: *Circuitry removed from PowerTech Plus Plugpack*

It is also possible to accommodate a 9V battery. 9V is sufficient to drive the relay and the voltage regulator still produces 3.35V at its output despite the lower input voltage. This was done for testing and demonstration purposes, since it was safer and easier to implement than a mains connection to the plug-pack circuitry. In the long term, however, using a battery is problematic since the relay places quite a large power demand on it. To be practical, there must be a suitable means of keeping the battery charged up, which is outside the scope of this design.

In the project's current form, the plug-pack provides the best solution by taking power from the existing mains wiring in the wall and converting that to a reliable, constant 12V supply for the switch circuit.

To make room for either a battery or the plug-pack innards, all features of the original design not critical to the switching function have been removed and the components packed more tightly together in the second design. Most notably, the two LEDs have been removed along with the programming header.

Figure 6.7 shows the final assembled circuit (left) and two possible configurations in terms of the type of power supply incorporated; plug-pack (centre) and 9V battery (right). These clearly show how each configuration fits neatly within the confines of the mounting box, although it should be noted that for practical purposes the plug-pack requires a more suitable housing so as not to leave the mains-level components exposed.

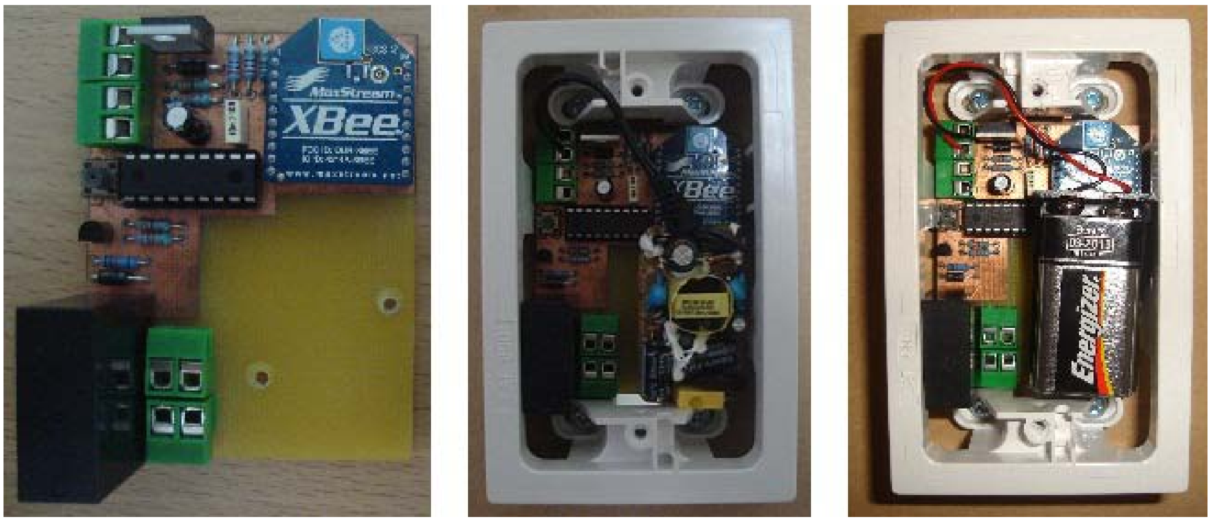


Figure 6.7: *Various assemblies of the second switch*

6.3.2 Switch Button as External Interrupt

One of the most important departures from the first design is in the way the wall switch is connected to the PIC. Instead of connecting to the input, RA0, the switch is connected to the input, RB2, on pin 17. This input doubles as an external interrupt, INT2, so that a signal change on that pin can trigger an interrupt in just the same way as a timer overflow or the receipt of a character by the USART. This approach addresses some of the problems with the original design.

6.3.2.1 Handling the Interrupt in Software

The interrupt is configured in the code as high priority, just the same as the receive interrupt. Additionally, it is configured to interrupt when a falling edge is detected at the input. Since the input is normally high and pressing the switch pulls the input low, this is the most appropriate setting. As an aside, setting the interrupt on a rising edge would also work; the interrupt would simply be triggered when the switch is released again, rather than when it is pressed. It is now no longer necessary to poll the input and the main program loop is empty as recommended.

6.3.2.2 Advantages

As mentioned previously, a high priority interrupt disables further interrupts from occurring while the interrupt service routine is running. This solves the problem of a potential race condition as discussed earlier because neither of the two situations can occur simultaneously. Either the switch will be pressed slightly before and block the remote request, or the remote request will occur first and the button-press will be ignored.

Another problem solved is that of the alternating output if the switch is held down. When the switch is first pressed, the falling edge at the input triggers the interrupt. Like before, a delay is included within the interrupt service routine to combat switch bounce. By the end of the delay, the ISR has run its course, switching the output, and the input has settled at the lower potential. Execution returns to the infinite loop, which does nothing. The input signal remains low as long as the switch is held down, preventing another falling edge from occurring and thereby preventing another external interrupt that would switch the output again. Also, since the ISR has completed, the receive interrupt is not blocked and an online user can still operate the switch remotely with no adverse effects.

As there are now *two* high priority interrupts, it is necessary to discover which has occurred so that the appropriate action can be taken. This is easily achieved by adding a conditional statement to the interrupt service routine that checks which interrupt flag has been set.

6.3.3 Switch Announcement

The code for the second switch is largely identical to the first, but contains an additional feature. When the switch first receives power or is reset, and once the USART has been configured, it announces its presence to the server. The microcontroller first puts the ZigBee module into command mode and then queries the module's address. This address is then passed on to the server in an advertisement message as discussed earlier.

When an advertisement message is received, the server queries the database for that address. If it exists, the state of that device is updated to 2, or "undefined". If a matching address is not found, a new entry is created in the devices table, again with state set to 2. This is effectively an auto-configuration mechanism, removing the need for the end user to manually add devices to the home network.

6.3.3.1 USART Anomaly

During testing, it was observed that the USART would frequently transmit a single character, 0xFF, when the switch was powered up, which proved problematic when attempting to implement the announce feature. The reason for the anomaly appears to be in the way the ports are configured when the microcontroller comes out of reset and particularly its impact on the TXD pin when the USART is initialized in the code. To prevent the pin level from manifesting itself as a transmitted character, a delay is introduced following the initialization of the IO ports and before the initialization of the USART, giving the signal levels on all pins time to stabilize [31].

7 Development and Testing

7.1 High Level Overview

The previous chapters have provided a detailed description of the design and working of each major component of the project. This chapter takes a step back, describing the setup of the project at the system level, and discussing the process of development and integration of the various components.

As previously described, the major components of the system are the server, the gateway and the end devices. The server is connected to the internet via a standard ADSL (broadband) modem router, providing always-on, high speed remote access from anywhere in the world. Commands are forwarded to the home gateway over a local TCP/IP connection and then transmitted wirelessly to the appropriate end device. Figure 7.1 shows the logical connections and the logical flow of information between system elements.

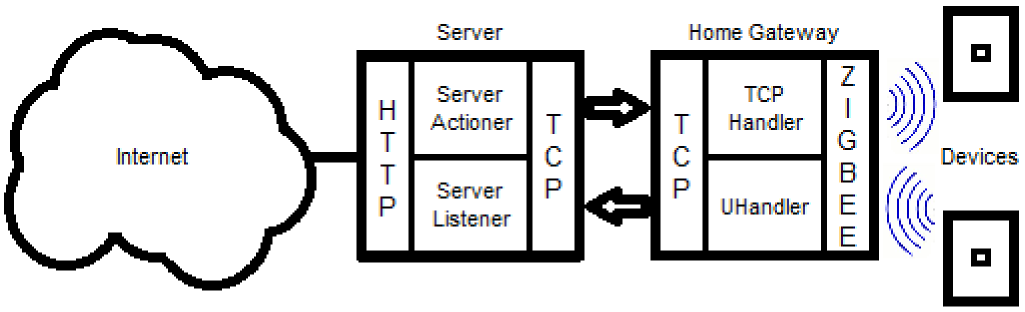


Figure 7.1: *Logical system architecture*

In reality, the server and gateway are connected over a home network, which can be either wired or wireless. Figure 7.2 provides a more accurate illustration of how the system is physically connected.

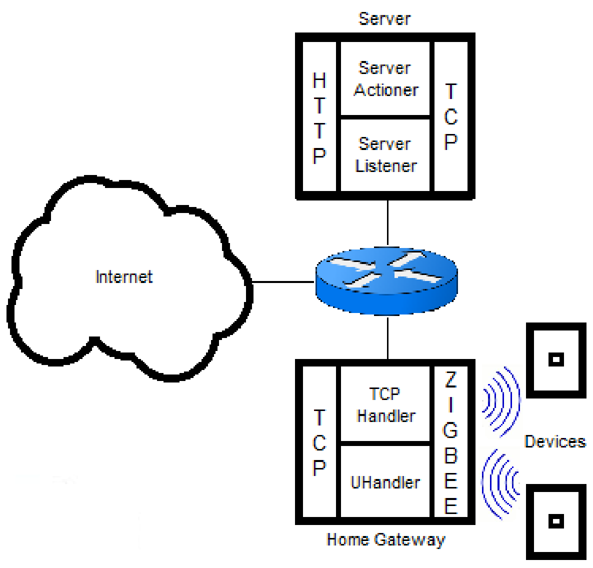


Figure 7.2: *Physical system architecture*

When viewed in this way, a number of possibilities become apparent. Firstly, it should be noted that the internet is not at all a necessary component of the system. A homeowner may wish to isolate their installation from the internet for security reasons, or maybe just limit accessibility and control to users within the home. Secondly, it would be equally valid from an abstract standpoint to simply shift the internet cloud over the router. In that case, the server and home gateway can be completely separated in space. This is an important consideration. It now becomes possible to outsource the server component to an external provider. It also becomes possible to control and manage multiple installations from a single server. Such considerations are, however, beyond the scope of this thesis.

This project makes use of a wireless router with no internet connection. An Apple MacBook hosts the server software and connects to the router wirelessly. In contrast, the home gateway is connected to the router via a Cat5 Ethernet cable. It is intended that the home gateway should be a single device, although for project demonstration it is comprised of the PICDEM.net2 development board and a separate ZigBee coordinator mounted on its own development board. These are then linked via serial cable. Figure 7.3 shows the final setup with a wall switch being the end device. For practicality and reliability while testing, a laboratory power supply is used to power the switch, rather than a battery.

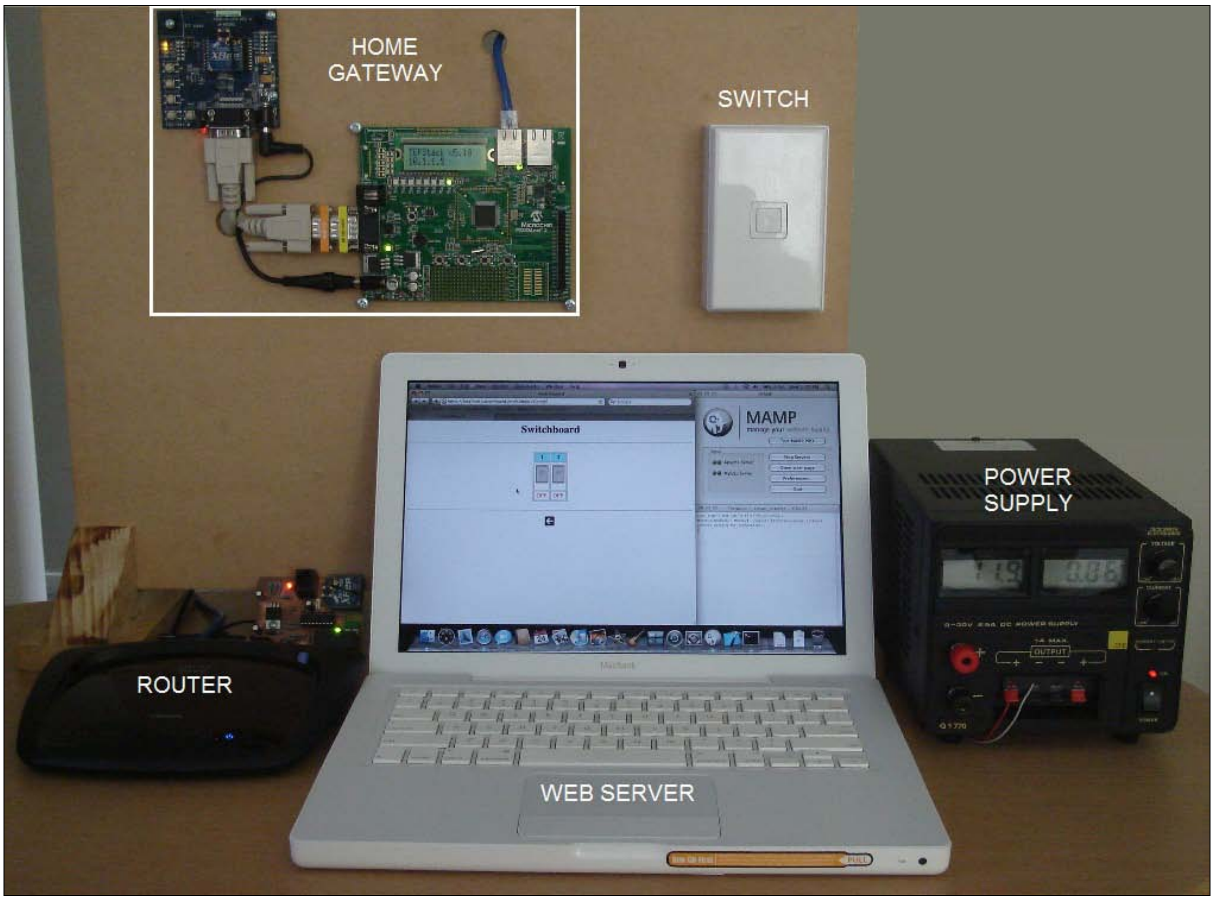


Figure 7.3: *Project setup*

7.2 System Development and Testing

One of the first tasks carried out for this project was to create a software emulation of an end device. Each device was represented by a simple TCP server running as a console application. Multiple instances of this application could be run concurrently, each provided with a different ID number from the command line in order to distinguish it, and each listening on a different port. Manual switching was simulated by pressing the spacebar, in which case the active window would respond to the key-press. For remote switching, each device could be individually addressed via its port number. Either way, it would output its current state and other important debugging information to the console window.

A great deal of time and effort was spent on this phase of development. It allowed many aspects of the web interface, database and communication protocol to be fine-tuned and finalised before embarking on the design of the home gateway or any of the switches. By that time, it was well understood what role each component would play and, more importantly, how it should react and communicate. It also allowed the intended working of the system to be demonstrated before committing further resources, namely money.

Once satisfied with the outcome of this phase, the PICDEM.net2 development kit was purchased and work begun on writing the software for the home gateway. As before, it was possible to simulate device responses and notifications from within the gateway code, and to properly test the communication between the gateway and the server before continuing.

The next step was to begin development on the interface between the PICDEM.net2 and the ZigBee coordinator. This task was assisted by the X-CTU utility, which provides a means of configuring and interacting with a ZigBee module on a development board. Furthermore, it is possible to test the interaction between a terminal and a ZigBee module in command mode as well as test communication between two modules. X-CTU was used extensively to view commands coming from the PICDEM.net2 and to provide manual responses.

Development of the switch software was undertaken using a PIC18F452 microcontroller on a QuikFlash development board, readily available in the university lab. Testing was performed in much the same manner as previously described. Commands to a switch could be provided manually via the X-CTU console and the responses from the switch could be viewed using the same. The QuikFlash board also has a variety of peripherals connected to the microcontroller with which to simulate the correct operation of the switch. An LED was turned on and off to observe proper switching, and a push-button provided a means of manually switching so as to generate a notification.

This point represented an important milestone in the project, since all of the individual elements of the system were now present and able to interact, and only the actual switch hardware was left to design. This final component was integrated into the rest of the system without any major difficulties. For the remainder of the project, ad hoc testing was carried out on the system as a whole. Various scenarios were investigated, such as how the system would respond to simultaneous manual and remote inputs, and how various components would cope with communication failures.

7.3 ZigBee Performance Testing

In its current configuration, where the ZigBee network has a star topology, the optimal physical arrangement is for the coordinator to be at the centre of the network. In real terms, this means that the home gateway should be located as near to the centre of a house as is practical. Since all nodes communicate directly with the coordinator, the placement of nodes is restricted by the effective range between two ZigBee devices.

The performance of this RF link was tested using the X-CTU utility once again, which includes a feature for qualitatively measuring the effective range in terms of received signal strength and the incidence of data errors. This is achieved by connecting a null modem terminator to the serial port on one of the ZigBee development boards. When RF data is received by this station, it is looped back via the adaptor and sent back over the air to the source. The source station then measures the signal strength of the returned data and detects errors between what was sent and what was received. In this case, the loopback adaptor was connected to the coordinator and a second development board was connected to a laptop with X-CTU running, providing a mobile test platform.

In clear, outdoor, line-of-sight conditions, few or no errors were observed up to approximately 50 metres. Indoors, the range drops significantly. As a guide to how the system might perform in an average home, the coordinator was situated in an upstairs room at one end of the house and readings taken at the location of switch plates throughout the rest of the house. Only at the point located furthest from the coordinator were a significant number of errors observed. Had the coordinator been more centrally located, it is likely that all points would have performed satisfactorily.

Another type of networking exists for ZigBee devices, called mesh networking, and it holds the key to solving the problem just discussed. In a mesh network, each of the ZigBee nodes is able to assist in routing data from one station to another where no direct path exists between them. This is just like routers in the internet. Mesh networks have many useful advantages, such as dynamic routing and the ability to “self-heal”, as well as the ability to broadcast a message to all nodes [5]. The improved range, reliability and flexibility ideally suits this type of application and so mesh networking is suggested as a focus of further research and development of this project.

8 Conclusions and Future Work

8.1 Conclusions

Home automation technology has been developing in different directions over the past few decades. Numerous smart products and systems exist, though most are unable to work together and creating complete, integrated solutions is still the domain of industry professionals. Many of these solutions have specific infrastructure requirements that can only be met when building a new house and are priced well out of reach of most people.

The focus of this project has been to produce a complete system for remotely monitoring and controlling switches within the home, with particular emphasis on minimising both the cost and complexity. These aims are best achieved using a centralised system framework.

The resulting system comprises a PC server, a home gateway and a network of ZigBee-enabled switches. Overall cost is reduced by confining as much of the system intelligence as possible to the server, so the end nodes require only basic processors to manage switching and communication. Also, many households nowadays have a broadband connection to the internet and, by extension, a computer on a home network. Thus, the fixed network infrastructure required to implement the system is already in place and can be reused to further reduce cost.

The home gateway is necessary to provide a TCP/IP interface to the ZigBee network. ZigBee is a natural choice for back-end connectivity because it provides reliable wireless networking for embedded devices. As a result, the system can be deployed in new and existing homes without the requirement for additional wiring. Also, for households that use Wi-Fi, there is little interference between the two network types [7].

In order to make the system as simple as possible, the interfaces are designed to be intuitive and familiar to a user. The system is accessed remotely via a standard web browser so that the experience of operating switches is no different to that of simply surfing online and, internet delays notwithstanding, the system responds quickly so that a remote user receives near-instant feedback about the outcome of a particular action. The switches themselves are also barely different to the common type, except that a low-voltage tactile push-button is fitted in place of the toggle variety, but it should be no less obvious to a user how these work.

It would have been possible, as has been done in earlier research projects, to write a custom client application to control and manage the system remotely, however this approach has disadvantages. Firstly, the software would have to be installed on every machine that a user would want remote access from and in many cases this would be impractical, as in the case of a work or public computer. Different versions of the software would be needed for different operating systems and it would also be more difficult to port to a different platform, such as a PDA. Changes to the system might require an update to the client software and this update would then have to be pushed out to each client device. The web-based approach solves all of these problems in one and fits better with the centralised framework model.

[9] argues that PC level gateways have not been widely accepted and that centralised systems suffer from the single-point-of-failure problem. This system is flexible in that a customer can opt to manage the server themselves on their home PC, or this element can be outsourced to a service provider. Remote access to the home is then provided by the gateway's TCP/IP interface. In fact, this approach is likely to offer advantages in terms of improved reliability, better security and technical support. The opportunity exists, then, for different companies to provide professional system management services, the same way that a different security companies can provide alarm monitoring services, and much like Honeywell's Global Home Server [4]. Since the server and database setup is the most complex element of the system, it makes sense to remove it from the hands of the user.

8.2 Future Work

Although the system has been reliably demonstrated, the implementation is by no means complete. One of the most complex and comprehensive tasks in developing the system is foreseeing all the possible ways in which the system can break down. In its current form, the system contains a number of bugs and some redundancies. Simply stated, there is much room for improvement.

The following is a brief list of current known issues.

- The *id* column in the *auth_users* table is redundant
- The *type_id* in the *devices* table should be of type *unsigned tinyint*
- The LCD currently does not display the correct device id when a notification is received
- Two devices switched together cause a notification clash

- The switch controller crashes when a switch is pressed and a remote request received at the same time - requires reset at the switch
- The server doesn't react to nACKs from the gateway
- When the home gateway nACKs an ID command, the switch state is set to -48 because the server reads the state of the switch from the 3rd character of the acknowledgement message, which in the case of a nACK is the null character
- The gateway should only send the ATCN command to the ZigBee coordinator if there is no error flag

The current hardware design of the switch is not compliant with any electrical standard. In actuality, it fails to comply with guidelines regarding isolation of mains and low-voltage circuits [32]. This is a critical safety concern as well as a legal constraint and issues relating to compliance should be further investigated.

This prototype system performs only switching, but additional devices could be introduced to perform other useful functions, such as dimming. The system could also incorporate pure sensing devices, such as motion detectors and temperature sensors, to provide information about the home environment. Based on this information, a user might choose to switch on a fan or a heater, or even notify police if a break-in is suspected. Even more likely, the system could apply its own intelligence, in line with user preferences, to automatically make control decisions. However, at all times the focus should be on adding value rather than complexity.

An obvious area for further development is the home gateway. Aside from moving away from development kits to a more dedicated hardware solution, the software could be upgraded so that the gateway plays a greater role in the operational stability of the system. For example, it could maintain a buffer of recent notifications in the event that communication with the server breaks down. Also on the subject of notifications, these could perhaps be better handled by the Uhandler module if each message contained information about its length. That way the handler would know how many bytes to expect rather than listening for individual characters over a fixed period, which can lead to errors when messages arrive simultaneously.

Such a situation is likely following a power outage. The feature of sending advertisement messages when a switch comes online is intended to demonstrate the potential for auto-configuration of devices, but in its current implementation is problematic. When power is restored, the gateway and all switches would come online at the same time and there would be a flood of advertisement messages to the gateway. A more organised approach to registering devices is recommended to address this.

In terms of the server, the web interface would benefit from improvements to make it more compatible with mobile devices and the history page could be improved by including functions to group or filter event data so that it is presented in a more manageable format. The server could also be made to periodically poll the state of devices to help maintain overall awareness of the system. Finally, in line with the earlier point about out-sourcing management of the server, work would need to be done, particularly in terms of the database design, to extend the system for multiple houses.

Ultimately, one cannot ignore the literature and repeated calls for cooperation towards a common, open standard. The possibilities presented by home automation technology are endless and can provide enormous benefits to society. It is hoped that this project will provide a springboard for further development in this area.

References

[1] Quinnell, R.A. *Networking moves to home automation*. 2007, EDN: Electronics Design, Strategy, News, 41-50.

[2] Rosendahl, A. and G. Botterweck. *Mobile Home Automation - Merging Mobile Value Added Services and Home Automation Technologies*. in *Management of Mobile Business, 2007. ICMB 2007. International Conference on the*. 2007.

[3] N. Rajbharti. Microchip TCP/IP Stack Application Note. `http://ww1.microchip.com/downloads/en/AppNotes/00833c.pdf`, 2008. Last accessed September 2009.

[4] Microchip Technology Inc. PIC18F1220/1320 Data Sheet. `http://ww1.microchip.com/downloads/en/DeviceDoc/39605F.pdf`, 2004. Last accessed November 2009.

[5] Digi International Inc. XBee/XBee-PRO DigiMesh 2.4 OEM RF Modules. `http://ftp1.digi.com/support/documentation/90000991_B.pdf`, 2008. Last accessed July 2009

[6] Nunes, R. and J. Delgado. *An architecture for a home automation system*. in *Electronics, Circuits and Systems, 1998 IEEE International Conference on*. 1998.

[7] Gill, K., et al., *A zigbee-based home automation system.* Consumer Electronics, IEEE Transactions on, 2009. **55**(2): p. 422-430.

[8] Abdulrazak, B. and A. Helal. *Enabling a Plug-and-play integration of smart environments*. in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*. 2006.

[9] Torbensen, R. *OHAS: Open home automation system*. in *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*. 2008.

[10] Lili, Y., Y. Shuang-Hua, and Y. Fang. *Safety and Security of Remote Monitoring and Control of intelligent Home Environments*. in *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*. 2006.

[11] Nunes, R.J.C. and J.C.M. Delgado. *An Internet application for home automation*. in *Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean*. 2000.

[12] Corcoran, P.M. and J. Desbonnet, *Browser-style interfaces to a home automation network.* Consumer Electronics, IEEE Transactions on, 1997. **43**(4): p. 1063-1069.

[13] Sangani, K., *Home automation - It's no place like home.* Engineering & Technology, 2006. **1**(9): p. 46-48.

[14] Topalis, E., et al., *A novel architecture for remote home automation e-services on an OSGi platform via high-speed Internet connection ensuring QoS support by using RSVP technology.* Consumer Electronics, IEEE Transactions on, 2002. **48**(4): p. 825-833.

[15] Smarthome Technology. INSTEON The Details. `http://www.insteon.net/df/insteondetails.pdf`, 2005. Last accessed March 2010.

[16] Clipsal Australia Pty Ltd. C-Bus Product Overview. `http://updates.clipsal.com/ClipsalOnline/Files/Brochures/C0000210.pdf`, 2009. Last accessed March 2010.

[17] Vantage Controls. Vantage. `http://www.vantagecontrols.com`. Last accessed March 2010.

[18] KNX. What is KNX?. `http://www.knx.org/knx/what-is-knx/`, 2009. Last accessed March 2010.

[19] Patricio, G. and L. Gomes. *Smart house monitoring and actuating system development using automatic code generation*. in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009.

[20] Chao-Lin, W., W. Wei-Chen, and F. Li-Chen. *Mobile agent based integrated control architecture for home automation system*. in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. 2004.

[21] Yuksekkaya, B., et al., *A GSM, internet and speech controlled wireless interactive home automation system.* Consumer Electronics, IEEE Transactions on, 2006. **52**(3): p. 837-843.

[22] Juing-Huei, S., L. Chyi-Shyong, and W. Wei-Chen, *The design and implementation of a low-cost and programmable home automation module.* Consumer Electronics, IEEE Transactions on, 2006. **52**(4): p. 1239-1244.

[23] OSGi Alliance. About the OSGi Alliance. `http://www.osgi.org/About/HomePage`. Last accessed March 2010.

[24] Intark, H., et al., *An integrated home server for communication, broadcast reception, and home automation.* Consumer Electronics, IEEE Transactions on, 2006. **52**(1): p. 104-109.

[25] Bergstrom, P., K. Driscoll, and J. Kimball, *Making home automation communications secure.* Computer, 2001. **34**(10): p. 50-56.

[26] ZigBee Alliance. ZigBee Alliance. http://www.zigbee.org/. Last accessed March 2010.

[27] Woo Suk, L. and H. Seung Ho. *KNX &#x2014; ZigBee gateway for home automation*. in *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*. 2008.

[28] Meloni, J.C., *Sams Teach Yourself PHP, MySQL and Apache All in One*. 2004, Indianapolis: Sams Publishing.

[29] Oracle Corporation. MySQL :: MySQL 5.1 Reference Manual :: 5.3.2.3 Password Hashing in MySQL. `http://dev.mysql.com/doc/refman/5.1/en/password-hashing.html`. Last accessed February 2010.

[30] Oracle Corporation. MySQL :: MySQL 5.5 Reference Manual :: 10.4.1 The CHAR and VARCHAR Types. `http://dev.mysql.com/doc/refman/5.5/en/char.html`. Last accessed February 2010.

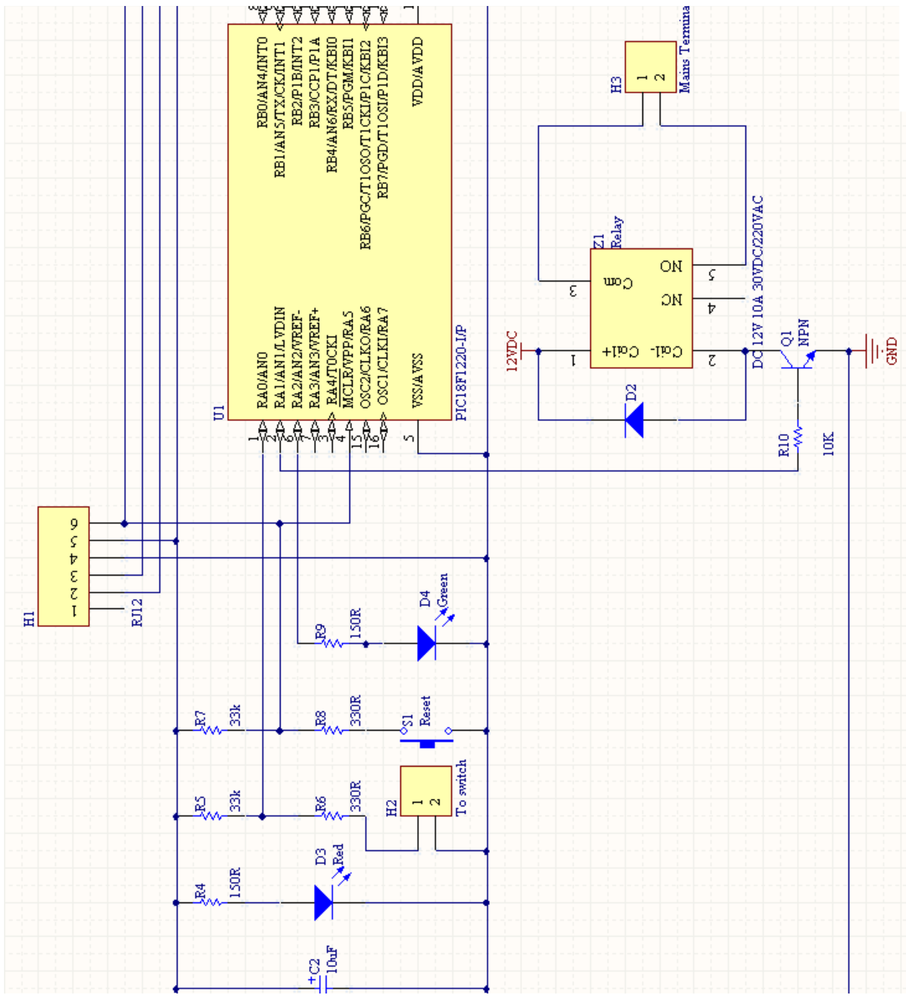
[31] The PNphpBB Group. How to initialize a USART? AVR sends 1 character immediately :: AVR Freaks. `http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&p=657588`. Last accessed January 2010.

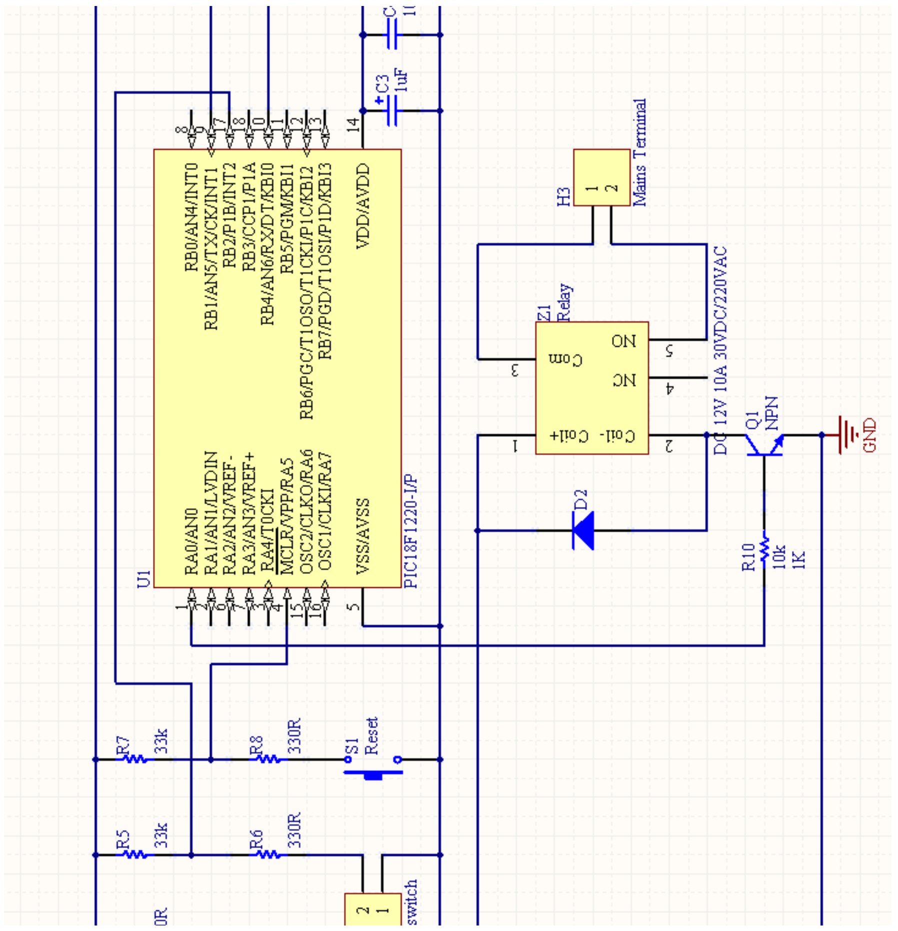
[32] Dick Smith Electronics. Mains Wiring & Handling Safety Guide. `http://www.dse.co.nz/dse.shop/en/catalog/LRN0002208`, 1996. Last accessed March 2010.

Appendix

A. Circuit Schematics

Circuit schematics for the Switch 1 and Switch 2 are presented on the following two pages.


U1
PIC18F1220-I/P
RA0/AN0
RA1/AN1/LVDIN
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
MCLR/VPP/RA5
OSC2/CLKO/RA6
OSC1/CLKI/RA7
VSS/AVSS
RB0/AN4/INT0
RB1/AN5/TX/CK/INT1
RB2/P1B/INT2
RB3/CCP1/P1A
RB4/AN6/RX/DT/KBI0
RB5/PGM/KBI1
RB6/PGC/T1OSO/T1CKI/P1C/KBI2
RB7/PGD/T1OSI/P1D/KBI3
VDD/AVDD
H1
RJ12
H2
To switch
H3
Mains Termina
Z1
Relay
Com
NC
NO
Coil+
Coil-
DC 12V 10A 30VDC/220VAC
12VDC
GND
Q1
NPN
R10
10K
D2
R4
150R
R5
33k
R6
330R
R7
33k
R8
330R
R9
150R
S1
Reset
D3
Red
D4
Green
C2
10uF



B. Web Server Source Code

B.1 login.html

```
<html>
<head>
      <title>User Login</title>
</head>

<body>
<div style="position:absolute;top:50%;left:0px;width:100%;height:1px">
      <div style="position:absolute;top:-100px;left:50%;width:200px;
      height:200px;margin-left:-100px">
            <fieldset style="width:200px;height:200px;text-align:center">
                  <form method="post" action="./login.php">
                        <br />
                        <p>
                              <strong>User:</strong><br />
                              <input type="text" name="username"
                              style="text-align:center" /><br />
                        </p>
                        <p>
                              <strong>Password:</strong><br />
                              <input type="password" name="password"
                              style="text-align:center" /><br />
                        <p>
                              <input type="submit" name="submit"
                              value="Submit" />
                        </p>
                  </form>
            </fieldset>
      </div>
</div>

</body>
</html>
```

B.2 login.php

```
<?php
session_start();
$sid = session_id();

function cleanup()
{
      // Clear all session variables
      $_SESSION = array();
      // Destroy session
      session_destroy();
      // Delete session cookie
      setcookie('PHPSESSID', '', time()-42000, '/');
      // Delete authorization cookie
      setcookie('auth', '', time()-42000, '/');

      // Redirect to login page
      header("Location: login.html");
      exit;
}

if (isset($_POST[logout]) || (!$_POST[username]) || (!$_POST[password]))
{
      cleanup();
}

// Connect to MySQL server and select database
$conn = mysql_connect("localhost", "root", "root") or die(mysql_error());
mysql_select_db("db1", $conn) or die(mysql_error());

// Create and issue the query
$sql = "SELECT f_name, l_name, admin FROM auth_users WHERE username =
       '$_POST[username]' AND password = password('$_POST[password]')";
$result = mysql_query($sql, $conn) or die(mysql_error());

// Get the number of rows in the result set; should be 1 if a match
if (mysql_num_rows($result) == 1)
{
      $result_array = mysql_fetch_array($result);

      // Set authorization cookie
      setcookie('auth', $sid, 0, '/', '', 0);
      $_SESSION['f_name'] = $result_array['f_name'];
      $_SESSION['l_name'] = $result_array['l_name'];
      $_SESSION['username'] = $_POST[username];
      $_SESSION['admin'] = $result_array['admin'];

      // Continue to main page
      header("Location: mainpage.php");
      exit;
}
else
{
      // Unauthorized user
      cleanup();
}
?>
```

B.3 mainpage.php

```
<?php
session_start();
$sid = session_id();

// Redirect to login form if not authorized
if ($_COOKIE[auth] != $sid)
{
      header("Location: login.php");
      exit;
}
?>

<html>
<head>
<title>Control Centre</title>
</head>

<body>
<h1 align="center">Control Centre</h1>

<div align="center">

<!-- Display control option to all users -->
<fieldset style="width:40%;text-align:center">
<br />
<form method="link" action="switchboard.php">
<input type="submit" name="submit" value="Control" />
</form>

<?php
// Display management options to admin users only
if ($_SESSION['admin'] == '1')
{
      echo '<br />';
      echo '<form method="link" action="manage_system.php">';
      echo '<input type="submit" name="submit" value="Manage" />';
      echo '</form>';

      echo '<br />';
      echo '<form method="link" action="show_history.php">';
      echo '<input type="submit" name="submit" value="History" />';
      echo '</form>';
}
?>

<!-- Display logout option to all users -->
<br />
<form method="post" action="login.php">
<input type="hidden" name="logout" value="logout" />
<input type="submit" name="submit" value="Logout" />
</form>
</fieldset>

</div>

</body>
</html>
```

B.4 switchboard.php

```
<?php
session_start();
$sid = session_id();

// Redirect to login form if not authorized
if ($_COOKIE[auth] != $sid)
{
      header("Location: login.php");
      exit;
}
?>

<html>
<head>
<title>Switchboard</title>
<meta http-equiv="pragma" content="no-cache" />
<meta http-equiv="expires" content="-1" />
<meta http-equiv="refresh" content="60" />
</head>
<body>

<?php
$conn = mysql_connect("localhost", "root", "root") or die(mysql_config());
mysql_select_db("db1", $conn) or die(mysql_error());
?>

<h1 align="center">Switchboard</h1>
<hr />
<br />

<table border="1" cellpadding="4" align="center">
<thead>
<tr style="background-color:#99CCFF">

<?php
$sql = "SELECT * FROM devices WHERE type_id = '1'";
$result = mysql_query($sql, $conn) or die(mysql_error());

while ($result_array = mysql_fetch_array($result))
{
      print '<th style="text-align:center">'.$result_array['id'].'</th>';
}
?>

</tr>
</thead>
<tbody align="center">
<tr>

<?php
$result = mysql_query($sql, $conn) or die(mysql_error());

while ($result_array = mysql_fetch_array($result))
{
      if ($result_array['state'] == '0')
      {
            print '<td align="center">';
            print '<form method="POST"
                   action="./cgi-bin/server_actioner.cgi">';
            print '<input type="hidden" name="id"
                   value="'.$result_array['id'].'" />';
            print '<input type="hidden" name="action" value="SW" />';
            print '<input type="hidden" name="nss" value="1" />';
            print '<input type="hidden" name="user"
                   value="'.$_SESSION['username'].'" />';
            print '<input type="image" name="submit"
                   src="./images/wall_switch_OFF.jpg"
                   style="width:37;height:55" alt="Switch" />';
            print '</form>';
            print '</td>';
      }
      else if ($result_array['state'] == '1')
      {
            print '<td align="center">';
            print '<form method="POST"
                   action="./cgi-bin/server_actioner.cgi">';
            print '<input type="hidden" name="id"
                   value="'.$result_array['id'].'" />';
            print '<input type="hidden" name="action" value="SW" />';
            print '<input type="hidden" name="nss" value="0" />';
            print '<input type="hidden" name="user"
                   value="'.$_SESSION['username'].'" />';
            print '<input type="image" name="submit"
                   src="./images/wall_switch_ON.jpg"
                   style="width:37;height:55" alt="Switch" />';
            print '</form>';
            print '</td>';
      }
      else
      {
            print '<td align="center">';
            print '<form method="POST"
                   action="./cgi-bin/server_actioner.cgi">';
            print '<input type="hidden" name="id"
                   value="'.$result_array['id'].'" />';
            print '<input type="hidden" name="action" value="ID" />';
            print '<input type="hidden" name="nss"
                   value="'.$result_array['id'].'" />';
            print '<input type="hidden" name="user"
                   value="'.$_SESSION['username'].'" />';
            print '<input type="image" name="submit"
                   src="./images/dont_walk_light_red.png"
                   style="width:55;height:55" alt="Activate" />';
            print '</form>';
            print '</td>';
      }
}
?>

</tr>
<tr>

<?php
$result = mysql_query($sql, $conn) or die(mysql_error());

while ($result_array = mysql_fetch_array($result))
{
      if ($result_array['state'] == '0')
      {
            print '<td style="text-align:center;color:red">OFF</td>';
      }
      else if ($result_array['state'] == '1')
      {
            print '<td style="text-align:center;color:green">ON</td>';
      }
      else
      {
            print '<td style="text-align:center;color:black">OFFLINE</td>';
      }
}
?>

</tr>
</table>
<br />

<!-- Provide link to main page -->
<hr />
<p align="center">
<a href="./mainpage.php"><img src="./images/left_arrow.gif" alt="Back"
   border="0" style="width:30;height:30" /></a>
</p>

</body>
</html>
```

B.5 manage_system.php

```
<?php
session_start();
$sid = session_id();

// Redirect to login form if not authorized
if ($_COOKIE[auth] != $sid)
{
      header("Location: login.php");
      exit;
}

function handle_error($msg)
{
      echo '<script type="text/javascript">alert("'.$msg.'")</script>';
      echo '<meta http-equiv="REFRESH" content="0;url=./mainpage.php">';
      exit;
}

// Show alert and redirect to main page if non-admin user
if ($_SESSION[admin] != '1')
{
      handle_error("Only a user with admin privileges can view this page");
}

// Connect to MySQL server and select database
if (!($conn = mysql_connect("localhost", "root", "root")))
{
      handle_error("Cannot connect to database: mysql_connect failed");
}

if (!mysql_select_db("db1", $conn))
{
      handle_error("Cannot connect to database: mysql_select_db failed");
}
?>

<html>

<head>
<title>System Management</title>
<meta http-equiv="pragma" content="no-cache" />
<meta http-equiv="expires" content="-1" />

<script type="text/javascript">
function show_confirm()
{
      return confirm("Are you sure?");
}
</script>
</head>

<body>
<h1 align="center">System Management</h1>

<!-- Provide link to main page -->
<p align="center">
```

```
<a href="./mainpage.php"><img src="./images/left_arrow.gif" alt="Back"
   border="0" style="width:30;height:30"></a>
</p>
<hr />

<?php
// Create and issue database query for devices
$sql = "SELECT * FROM devices";
if (!($result = mysql_query($sql, $conn)))
{
      handle_error("Cannot query database: mysql_query failed");
}
?>

<h2 align="center">Devices</h2>

<!-- Display device table -->
<table border="1" cellpadding="4" style="width:90%" align="center">
<thead>
<tr style="background-color:#99CCFF">
<th width="17%">Address</th>
<th width="17%">PAN ID</th>
<th width="16%">Channel</th>
<th width="16%">Node ID</th>
<th width=17%">Type ID</th>
<th />
</tr>
</thead>
<tbody align="center">

<?php
while ($result_array = mysql_fetch_array($result))
{
      $address = $result_array['address'];
      $pan_id = $result_array['pan_id'];
      $channel = $result_array['channel'];
      $node_id = $result_array['node_id'];
      $type_id = $result_array['type_id'];

      echo '<tr><td>'.$address.'</td><td>'.$pan_id.'</td><td>'.$channel.
           '</td><td>'.$node_id.'</td><td>'.$type_id.'</td>';

      // Display option to remove device
      echo '<form method="post" action="./add_device.php"
                  onsubmit="return show_confirm();">
            <td>
            <input type="hidden" name="remove" value="remove" />
            <input type="hidden" name="address" value="'.$address.'" />
            <input type="image" name="submit" src="./images/trash.gif"
                  alt="Remove" style="width:30;height:30" />
            </td>
            </form>
            </tr>';
}
?>

<!-- Last row of table contains form for adding a new device -->
<tr>
<form method="post" action="./add_device.php">
<td><input type="text" name="address" style="text-align:center" /></td>
<td><input type="text" name="pan_id" style="text-align:center" /></td>
```

```
<td><input type="text" name="channel" style="text-align:center" /></td>
<td><input type="text" name="node_id" style="text-align:center" /></td>
<td><input type="text" name="type_id" style="text-align:center" /></td>
<td><input type="submit" name="submit" value="Add" /></td>
</form>
</tr>
</tbody>
</table>

<?php
// Create and issue database query for users
$sql = "SELECT * FROM auth_users";
if (!($result = mysql_query($sql, $conn)))
{
      handle_error("Cannot query database: mysql_query failed");
}
?>

<h2 align="center">Users</h2>

<!-- Display user table -->
<table border="1" cellpadding="4" style="width:90%" align="center">
<thead>
<tr style="background-color:#99CCFF">
<th width="17%">First Name</th>
<th width="17%">Last Name</th>
<th width="16%">Username</th>
<th width="16%">Password</th>
<th width="17%">Admin</th>
<th />
</tr>
</thead>
<tbody align="center">

<?php
while ($result_array = mysql_fetch_array($result))
{
      $f_name = $result_array['f_name'];
      $l_name = $result_array['l_name'];
      $username = $result_array['username'];

      echo '<td>'.$f_name.'</td><td>'.$l_name.'</td><td>'.$username.
           '</td>';
      echo '<td><img src="./images/passport_control.gif" /></td><td>';

      if ($result_array['admin'] == '1')
      {
            echo 'Yes</td>';
      }
      else
      {
            echo 'No</td>';
      }

      // Display option to remove user - the admin user cannot be removed
      if ($username != "admin")
      {
            echo '<form method="post" action="./add_user.php"
                  onsubmit="return show_confirm();">
                  <td>
                  <input type="hidden" name="remove" value="remove" />
                  <input type="hidden" name="username"
                        value="'.$username.'" />
                  <input type="image" name="submit"
                        src="./images/trash.gif" alt="Remove"
                        style="width:30;height:30" />
                  </td>
                  </form>';
      }

      echo '</tr>';
}
?>

<!-- Last row of table contains form for adding a new user -->
<tr>
<form method="post" action="./add_user.php">
<td><input type="text" name="f_name" style="text-align:center" /></td>
<td><input type="text" name="l_name" style="text-align:center" /></td>
<td><input type="text" name="username" style="text-align:center" /></td>
<td><input type="password" name="password" style="text-align:center" />
</td>
<td><input type="checkbox" name="admin" /></td>
<td><input type="submit" name="submit" value="Add" /></td>
</form>
</tr>
</tbody>    <title>System Management</title>
</table>

<!-- Provide link to main page -->
<br />
<hr />
<p align="center">
<a href="./mainpage.php"><img src="./images/left_arrow.gif" alt="Back"
   border="0" style="width:30;height:30"></a>
</p>

</body>
</html>
```

B.6 add_device.php

```
<?php
session_start();
$sid = session_id();

if ($_COOKIE[auth] != $sid)
{
      // Redirect to login form if not authorized
      header("Location: login.php");
      exit;
}

// Show alert and redirect to main page if non-admin user
if ($_SESSION[admin] != '1')
{
      echo '<script type="text/javascript">
            alert("Only a user with admin privileges can do this")
            </script>';
      echo '<meta http-equiv="REFRESH" content="0;url=./mainpage.php">';
      exit;
}

function handle_error($msg)
{
      echo '<script type="text/javascript">alert("'.$msg.'")</script>';
      echo '<meta http-equiv="REFRESH" content="0;
            url=./manage_system.php">';
      exit;
}

if (!isset($_POST[remove]))
{
      // If adding a new device, check that all the required information
         has been provided
      if ((!$_POST[address]) || (!$_POST[pan_id]) || (!$_POST[channel]) ||
            (!$_POST[node_id]) || (!$_POST[type_id]))
      {
            // If information is missing, show javascript alert and
               redirect to system management page
            handle_error("One or more fields was left blank");
      }

      // Load POST data into variables (not necessary, but makes code more
         readable)
      $address = $_POST[address];
      $pan_id = $_POST[pan_id];
      $channel = $_POST[channel];
      $node_id = $_POST[node_id];
      $type_id = $_POST[type_id];
}

// Connect to MySQL server and select database
if (!($conn = mysql_connect("localhost", "root", "root")))
{
      handle_error("Cannot connect to database: mysql_connect failed");
}
```

```
if (!mysql_select_db("db1", $conn))
{
      handle_error("Cannot connect to database: mysql_select_db failed");
}

// Create and issue the query
if (isset($_POST[remove]))
{
      $sql = "DELETE FROM devices WHERE address='$_POST[address]'";
}
else
{
      $sql = "INSERT INTO devices VALUES ('', '$address', '$pan_id',
             '$channel', '$node_id', '$type_id', '2')";
}

if (!($result = mysql_query($sql, $conn)))
{
      handle_error("Cannot update database: mysql_query failed");
}

// Return to system management page
header("Location: manage_system.php");
exit;
?>
```

B.7 add_user.php

```
<?php
session_start();
$sid = session_id();

$domain = parse_url($_SERVER[HTTP_REFERER]);

if ($_COOKIE[auth] != $sid)
{
      // Redirect to login form if not authorized
      header("Location: login.php");
      exit;
}

// Show alert and redirect to main page if non-admin user
if ($_SESSION[admin] != '1')
{
      echo '<script type="text/javascript">alert("Only a user with admin
            privileges can do this")</script>';
      echo '<meta http-equiv="REFRESH" content="0;url=./mainpage.php">';
      exit;
}

function handle_error($msg)
{
      echo '<script type="text/javascript">alert("'.$msg.'")</script>';
      echo '<meta http-equiv="REFRESH" content="0;
            url=./manage_system.php">';
      exit;
}

if (!isset($_POST[remove]))
{
      // If adding a new user, check that all the required information has
         been provided
      if ((!$_POST[f_name]) || (!$_POST[l_name]) || (!$_POST[username]) ||
            (!$_POST[password]))
      {
            // If information is missing, show javascript alert and
               redirect to system management page
            handle_error("One or more text fields was left blank");
      }

      // Load POST data into variables (not necessary, but makes code more
         readable)
      $f_name = $_POST[f_name];
      $l_name = $_POST[l_name];
      $username = $_POST[username];
      $password = $_POST[password];

      // When admin is checked in the form, it is posted as 'on', otherwise
         it is not sent at all
      // Change to format used in the database (0/1)
      if ($_POST[admin] == "on")
      {
            $admin = 1;
      }
      else
      {
            $admin = 0;
      }
}

// Connect to MySQL server and select database
if (!($conn = mysql_connect("localhost", "root", "root")))
{
      handle_error("Cannot connect to database: mysql_connect failed");
}

if (!mysql_select_db("db1", $conn))
{
      handle_error("Cannot connect to database: mysql_select_db failed");
}

// Create and issue the query
if (isset($_POST[remove]))
{
      $sql = "DELETE FROM auth_users WHERE username='$_POST[username]'";
}
else
{
      $sql = "INSERT INTO auth_users VALUES ('', '$f_name', '$l_name',
             '$username', password('$password'), '$admin')";
}

if (!($result = mysql_query($sql, $conn)))
{
      handle_error("Cannot update database: mysql_query failed");
}

// Return to system management page
header("Location: manage_system.php");
exit;
?>
```

B.8 show_history.php

```
<?php
session_start();
$sid = session_id();

if ($_COOKIE[auth] != $sid)
{
      // Redirect to login form if not authorized
      header("Location: login.php");
      exit;
}

// Show alert and redirect to main page if non-admin user
if ($_SESSION[admin] != '1')
{
      echo '<script type="text/javascript">alert("Only a user with admin
            privileges can do this")</script>';
      echo '<meta http-equiv="REFRESH" content="0;url=./mainpage.php">';
      exit;
}

function handle_error($msg)
{
      echo '<script type="text/javascript">alert("'.$msg.'")</script>';
      echo 'meta http-equiv="REFRESH" content="0;
            url=./manage_system.php">';
      exit;
}

// Connect to MySQL server and select database
if (!($conn = mysql_connect("localhost", "root", "root")))
{
      handle_error("Cannot connect to database: mysql_connect failed");
}

if (!mysql_select_db("db1", $conn))
{
      handle_error("Cannot connect to database: mysql_select_db failed");
}

if (isset($_POST[clr]))
{
      $sql = "DELETE FROM history";

      if (!($result = mysql_query($sql, $conn)))
      {
            handle_error("Cannot query database: mysql_query failed");
      }
}

// Create and issue the query
$sql = "SELECT * FROM history";

if (!($result = mysql_query($sql, $conn)))
{
      handle_error("Cannot query database: mysql_query failed");
}
?>
```

```
<html>
<head>
<title>History</title>
<meta http-equiv="pragma" content="no-cache" />
<meta http-equiv="expires" content="-1" />
<meta http-equiv="refresh" content="60" />
</head>
<body>

<h1 align="center">History</h1>

<!-- Provide link to main page -->
<p align="center">
<a href="./mainpage.php"><img src="./images/left_arrow.gif" alt="Back"
   border="0" style="width:30;height:30"></a>
</p>
<hr />
<br />

<table border="1" cellpadding="4" style="width:90%" align="center">
<thead>
<tr style="background-color:#99CCFF">
<th width="30%">Time</th>
<th width="10%">ID</th>
<th width="10%">Action</th>
<th width="10%">State</th>
<th width="30%">User</th>
</tr>
</thead>
<tbody>

<?php
while ($result_array = mysql_fetch_array($result))
{
      print '<tr>';
      print '<td align="center">'.$result_array['time'].'</td>';
      print '<td align="center">'.$result_array['id'].'</td>';
      print '<td align="center">'.$result_array['action'].'</td>';
      print '<td align="center">'.$result_array['state'].'</td>';
      print '<td align="center">'.$result_array['user'].'</td>';
      print '</tr>';
}
?>

</tbody>
</table>

<br />
<div align="center">
<form method="POST" action="./show_history.php">
<input type="hidden" name="clr" value="clr" />
<input type="submit" name="submit" value="Clear history" />
</form>
</div>

<!-- Provide link to main page -->
<hr />
<p align="center">
<a href="./mainpage.php"><img src="./images/left_arrow.gif" alt="Back"
   border="0" style="width:30;height:30"></a>
```

```
</p>

</body>
</html>
```

C. Server Processes Source Code

C.1 server_actioner_cgi.c

```
/*
** server_actioner_cgi.c
*/

#include <arpa/inet.h>
#include <errno.h>
#include <mysql.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define HUB_ADDRESS "10.1.1.5"
#define HUB_PORT "3456"
#define MAXDATASIZE 21
#define MAXLEN 80

void *get_in_addr(struct sockaddr *sa);
int connect_to_hub();

int main(void)
{
      // User variables
      char username[26] = "";       // varchar(25) + nul character

      // Device variables
      int id = 0;                   // tinyint(3)
      char address[17] = "";        // varchar(16) + nul character
      int state = 0;                // tinyint(1)
      int new_state = 0;

      char action[3] = "\0\0\0";

      // Get environment variables from the URL
      char formdata[MAXLEN];
      char *lenstr;
      long len;

      printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1",
             13,10);
      printf("<TITLE>Server actioner</TITLE>\n");

      lenstr = getenv("CONTENT_LENGTH");
      if (lenstr == NULL || sscanf(lenstr, "%ld", &len) != 1 ||
            len > MAXLEN)
      {
            free(lenstr);
            printf("<script type=\"text/javascript\">alert(\"There was a
                    problem with the form data\");</script>");
```

```
        printf("<meta http-equiv=\"REFRESH\" content=\"0;
                url=../switchboard.php\">\n");
        exit(1);
}
else
{
        fgets(formdata, len+1, stdin);
        if (sscanf(formdata, "id=%d&action=%c%c&nss=%d&user=%s",
              &id, &action[0], &action[1], &new_state, username) != 5)
        {
              // Invalid variable data - clean up and redirect to
                 calling webpage
              free(lenstr);
              printf("<script type=\"text/javascript\">alert(\"There
                      was invalid GET data\");</script>");
              printf("<meta http-equiv=\"REFRESH\" content=\"0;
                      url=../switchboard.php\">\n");
              exit(1);
        }
}

// Parse username string to remove submit data
int i;
for (i=0; i<26; i++)
{
        if (username[i] == '&')
        {
              username[i] = '\0';
              break;
        }
        else if (username[i] == '\0')
        {
              break;
        }
}

// Database variables
MYSQL *conn;
MYSQL_RES *res;
MYSQL_ROW row;
char query_string[255];

conn = mysql_init(NULL);

if (!mysql_real_connect(conn, "localhost", "root", "root", "db1",
        8889, "/Applications/MAMP/tmp/mysql/mysql.sock", 0))
{
        printf("<script type=\"text/javascript\">alert(\"%s\");
                </script>", mysql_error(conn));
        printf("<script type=\"text/javascript\">alert(\"There was a
                problem accessing MySQL\");</script>");
        printf("<meta http-equiv=\"REFRESH\" content=\"0;
                url=../switchboard.php\">\n");
}

sprintf(query_string, "SELECT address FROM devices WHERE id = '%d'",
        id);

if (mysql_query(conn, query_string))
{
        fprintf(stderr, "%s\n", mysql_error(conn));
```

```
      printf("<script type=\"text/javascript\">alert(\"There was a
              problem executing a MySQL query\");</script>");
      printf("<meta http-equiv=\"REFRESH\" content=\"0;
              url=../switchboard.php\">\n");
      exit(1);
}

res = mysql_store_result(conn);
row = mysql_fetch_row(res);

strcpy(address, row[0]);

// Communication variables
int data_socket;
char send_buffer[MAXDATASIZE];
char recv_buffer[MAXDATASIZE];
int numbytes = 0;

// Establish connection to hub
data_socket = connect_to_hub();

// Send control message to device
sprintf(send_buffer, "C-%s", address);
if (send(data_socket, send_buffer, strlen(send_buffer), 0) == -1)
{
      perror("actioner: send");
      printf("<script type=\"text/javascript\">alert(\"There was a
              problem sending data to the device server\");
              </script>");
      printf("<meta http-equiv=\"REFRESH\" content=\"0;
              url=../switchboard.php\">\n");
      exit(1);
}

// Listen for acknowledgement from device
numbytes = recv(data_socket, recv_buffer, MAXDATASIZE-1, 0);
switch (numbytes)
{
      case -1:
      case 0:
            perror("actioner: recv");
            printf("<script type=\"text/javascript\">alert(\"There
                    was a problem receiving data from the device
                    server\");</script>");
            printf("<meta http-equiv=\"REFRESH\" content=\"0;
                    url=../switchboard.php\">\n");
            exit(1);
      default:
            recv_buffer[numbytes] = '\0';
            break;
}

// Send data message to device
if (action[0] == 'I' && action[1] == 'D')
{
      sprintf(send_buffer, "D-%s-%c", action, (char)id);
}
else
{
      sprintf(send_buffer, "D-%s-%d", action, new_state);
}
```

```
if (send(data_socket, send_buffer, strlen(send_buffer), 0) == -1)
{
      perror("actioner: send");
      printf("<script type=\"text/javascript\">alert(\"There was a
              problem sending data to the device server\");
              </script>");
      printf("<meta http-equiv=\"REFRESH\" content=\"0;
              url=../switchboard.php\">\n");
      exit(1);
}

// Listen for acknowledgement from device
numbytes = recv(data_socket, recv_buffer, MAXDATASIZE-1, 0);
switch (numbytes)
{
      case -1:
      case 0:
            perror("actioner: recv(0)");
            printf("<script type=\"text/javascript\">alert(\"There
                    was a problem receiving data from the device
                    server\");</script>");
            printf("<meta http-equiv=\"REFRESH\" content=\"0;
                    url=../switchboard.php\">\n");
            exit(1);
      default:
            recv_buffer[numbytes] = '\0';
            break;
}

//Update the devices table with the new state and place an entry in
  the history table
sprintf(query_string, "UPDATE devices SET state='%d' WHERE id =
        '%d'", recv_buffer[2]-48, id);

if (mysql_query(conn, query_string))
{
      fprintf(stderr, "%s\n", mysql_error(conn));
      printf("<script type=\"text/javascript\">alert(\"There was a
              problem executing a MySQL query\");</script>");
      printf("<meta http-equiv=\"REFRESH\" content=\"0;
              url=../switchboard.php\">\n");
      exit(1);
}

sprintf(query_string, "INSERT INTO history VALUES (NULL, '%d', '%s',
        '%d', '%s')", id, action, new_state, username);

if (mysql_query(conn, query_string))
{
      fprintf(stderr, "%s\n", mysql_error(conn));
      printf("<script type=\"text/javascript\">alert(\"There was a
              problem executing a MySQL query\");</script>");
      printf("<meta http-equiv=\"REFRESH\" content=\"0;
              url=../switchboard.php\">\n");
      exit(1);
}

strcpy(send_buffer, "^Z");
if (send(data_socket, send_buffer, strlen(send_buffer), 0) == -1)
{
```

```
            perror("actioner: send");
            printf("<script type=\"text/javascript\">alert(\"There was a
                    problem sending data to the device server\");
                    </script>");
            printf("<meta http-equiv=\"REFRESH\" content=\"0;
                    url=../switchboard.php\">\n");
            exit(1);
      }

      // Finished with this connection - close it
      close(data_socket);

      // Finished with database - clean up
      mysql_free_result(res);
      mysql_close(conn);

      // Return to webpage
      printf("<meta http-equiv=\"REFRESH\" content=\"0;
              url=../switchboard.php\">\n");

      free(lenstr);
      return 0;
}


void *get_in_addr(struct sockaddr *sa)
{
      if (sa->sa_family == AF_INET)
      {
            return &(((struct sockaddr_in*)sa)->sin_addr);
      }

      return &(((struct sockaddr_in6*)sa)->sin6_addr);
}


int connect_to_hub()
{
      int data_socket, rv;
      struct addrinfo hints, *servinfo, *p;
      char s[INET6_ADDRSTRLEN];

      memset(&hints, 0, sizeof hints);
      hints.ai_family = AF_UNSPEC;
      hints.ai_socktype = SOCK_STREAM;

      if ((rv = getaddrinfo(HUB_ADDRESS, HUB_PORT, &hints, &servinfo)) !=
          0)
      {
            fprintf(stderr, "actioner getaddrinfo: %s\n",
                  gai_strerror(rv));
            printf("<script type=\"text/javascript\">alert(\"There was a
                    problem with getaddrinfo\");</script>");
            printf("<meta http-equiv=\"REFRESH\" content=\"0;
                    url=../switchboard.php\">\n");
            exit(1);
      }

      for (p = servinfo; p != NULL; p = p->ai_next)
      {
            if ((data_socket = socket(p->ai_family, p->ai_socktype,
                p->ai_protocol)) == -1)
            {
                  perror("actioner: socket");
                  continue;
            }
            if (connect(data_socket, p->ai_addr, p->ai_addrlen) == -1)
            {
                  close(data_socket);
                  perror("actioner: connect");
                  continue;
            }
            break;
      }

      if (p == NULL)
      {
            fprintf(stderr, "actioner: failed to connect\n");
            printf("<script type=\"text/javascript\">alert(\"There was a
                    problem connecting to the device server\");</script>");
            printf("<meta http-equiv=\"REFRESH\" content=\"0;
                    url=../switchboard.php\">\n");
            exit(1);
      }

      inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
                s, sizeof s);
      freeaddrinfo(servinfo);

      return data_socket;
}
```

C.2 server_listener.c

```
/*
** server_listener.c
*/

#include <arpa/inet.h>
#include <errno.h>
#include <mysql.h>
#include <netdb.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define BACKLOG 10      // How many pending connections queue will hold
#define MAXDATASIZE 21  // How many bytes we can send or receive at one
                           time
#define PORT "3490"     // The port users will be connecting to

void sigchld_handler(int s);
void *get_in_addr(struct sockaddr *sa);

int main(void)
{
      int i;

      // Communication variables
      int listening_socket, data_socket;        // Listen on
            listening_socket, new connection on data_socket
      socklen_t sin_size;
      struct addrinfo hints, *servinfo, *p;
      struct sockaddr_storage their_addr;       // Connector's address info
      struct sigaction sa;
      char s[INET6_ADDRSTRLEN];
      char buffer[MAXDATASIZE];                 // Communication buffer
      int rv, yes=1, numbytes;

      // Device variables
      int id = 0;
      unsigned char idh = 0;
      char address[17] = "";
      char action[3] = "";
      int state = 0;

      action[2] = '\0';

      // Set up server listener
      memset(&hints, 0, sizeof hints);
      hints.ai_family = AF_UNSPEC;
      hints.ai_socktype = SOCK_STREAM;
      hints.ai_flags = AI_PASSIVE;              // Use own IP
```

```
if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0)
{
      fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
      return 1;
}

for (p = servinfo; p != NULL; p = p->ai_next)
{
      if ((listening_socket = socket(p->ai_family, p->ai_socktype,
          p->ai_protocol)) == -1)
      {
            perror("server: socket");
            continue;
      }
      if (setsockopt(listening_socket, SOL_SOCKET, SO_REUSEADDR,
          &yes, sizeof(int)) == -1)
      {
            perror("setsockopt");
            exit(1);
      }
      if (bind(listening_socket, p->ai_addr, p->ai_addrlen) == -1)
      {
            close(listening_socket);
            perror("server: bind");
            continue;
      }
      break;
}

if (p == NULL)
{
      fprintf(stderr, "server: failed to bind\n");
      return 2;
}

freeaddrinfo(servinfo);

if (listen(listening_socket, BACKLOG) == -1)
{
      perror("listen");
      exit(1);
}

// Reap all dead processes
sa.sa_handler = sigchld_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1)
{
      perror("sigaction");
      exit(1);
}

// Feed back status information to the user
printf("server: waiting for connections...\n");

// Main accept() loop
while(1)
{
      // Accept incoming connection
      sin_size = sizeof their_addr;
```

```
data_socket = accept(listening_socket, (struct sockaddr *)
                     &their_addr, &sin_size);
if (data_socket == -1)
{
      perror("accept");
      continue;
}
inet_ntop(their_addr.ss_family, get_in_addr((struct sockaddr *)
          &their_addr), s, sizeof s);
printf("server: got connection from %s\n", s);

if (!fork())
{
      // This is the child process
      // The child doesn't need the listener
      close(listening_socket);

      // Database variables
      MYSQL *conn;
      MYSQL_RES *result;
      char temp[255];

      // Connect to database
      conn = mysql_init(NULL);
      if (!mysql_real_connect(conn, "localhost", "root",
          "root", "db1", 8889, "/Applications/MAMP/tmp/mysql/
          mysql.sock", 0))
      {
            fprintf(stderr, "%s\n", mysql_error(conn));
            exit(1);
      }

      // Receive notification data
      if ((numbytes = recv(data_socket, buffer, MAXDATASIZE-1,
          0)) == -1)
      {
            perror("recv");
            exit(1);
      }
      buffer[numbytes] = '\0';

      // Check the received data
      if (numbytes == 0)
      {
            // No data was received
            // The remote terminal may have closed the
               connection unexpectedly
            printf("server: received nothing\n");
      }
      else
      {
            // Check for advertisement message
            if (buffer[0] == 'A' && buffer[1] == 'D')
            {
                  address[0] = '0';
                  address[1] = '0';
                  for (i=2; i < 16; i++)
                  {
                        address[i] = buffer[i];
                  }
                  address[16] = '\0';
```

```
        sprintf(temp, "SELECT id FROM devices WHERE
                address = '%s'", address);
        if (mysql_query(conn, temp))
        {
              fprintf(stderr, "%s\n",
                      mysql_error(conn));
              return 3;
        }

        result = mysql_store_result(conn);
        if (mysql_num_rows(result))
        {
              sprintf(temp, "UPDATE devices SET
                      state='2' WHERE address =
                      '%s'", address);
        }
        else
        {
              sprintf(temp, "INSERT INTO devices
                      VALUES ('', '%s', '3332', 'C',
                      'SW', '1', '2')", address);
        }

        if (mysql_query(conn, temp))
        {
              fprintf(stderr, "%s\n",
                      mysql_error(conn));
              return 3;
        }
  }

  // Parse notification data
  else if (sscanf(buffer, "N-%c-%c%c-%d", &idh,
           &action[0], &action[1], &state) == 4)
  {
        id = (int)idh;

        // Data ok - update database
        // Update devices table with new device
           device_state
        sprintf(temp, "UPDATE devices SET state='%d'
                WHERE id = %d", state, id);

        // DEBUG
        printf("Notification: id=%d action=%s
                state=%d\n", id, action, state);

        if (mysql_query(conn, temp))
        {
              fprintf(stderr, "%s\n",
                      mysql_error(conn));
              return 3;
        }

        // Update history table with event
        sprintf(temp, "INSERT INTO history VALUES
                (NULL, '%d', '%s', '%d', 'UNKNOWN')",
                id, action, state);
```

```
                        if (mysql_query(conn, temp))
                        {
                            fprintf(stderr, "%s\n",
                                    mysql_error(conn));
                            return 3;
                        }
                    }
                    else
                    {
                        // Received invalid data
                        printf("Invalid data received: ");
                        i = 0;
                        while (buffer[i] != '\0')
                        {
                            if (buffer[i] == '\r')
                            {
                                printf("\\r");
                            }
                            else if (buffer[i] == '\n')
                            {
                                printf("\\n");
                            }
                            else
                            {
                                printf("%c", buffer[i]);
                            }
                            i++;
                        }
                        printf("\n");
                    }
                }

                // Finished with this connection - close it
                close(data_socket);

                // Close connection to database
                mysql_close(conn);

                // Terminate thread
                _exit(0);
            }

            // This is the parent process
            // The parent doesn't need the data socket
            close(data_socket);
        }

        return 0;
}

void sigchld_handler(int s)
{
        while(waitpid(-1, NULL, WNOHANG) > 0);
}

void *get_in_addr(struct sockaddr *sa)
{
        if (sa->sa_family == AF_INET)
        {
            return &(((struct sockaddr_in*)sa)->sin_addr);
        }
        return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
```

D. Gateway Source Code

D.1 TCPHandler (TCPH.c)

```
#define __TCPH_C

#include "TCPIPConfig.h"
#include "TCPIP Stack/TCPIP.h"

#define SERVER_PORT 3456
#define MAX_DATA_LENGTH 21

void TCPHandler(void)
{
      static TICK Timer;
      static TCP_SOCKET server_socket = INVALID_SOCKET;
      static BYTE state;
      BYTE TCPBuffer[MAX_DATA_LENGTH];
      WORD data_length;
      BYTE DAH[9];
      BYTE DAL[9];
      BYTE UBuffer[15];
      BYTE UReply[7];
      BYTE temp[16];
      WORD i;
      BYTE j;

      static enum _TCPHandlerState
      {
            SM_HOME = 0,
            SM_LISTENING,
            SM_PROCESSING
      } TCPHandlerState = SM_HOME;

      switch (TCPHandlerState)
      {
            case SM_HOME:
                  // Allocate a socket for this server to listen and accept
                     connections on
                  server_socket = TCPOpen(0, TCP_OPEN_SERVER, SERVER_PORT,
                                  TCP_PURPOSE_GENERIC_TCP_SERVER);
                  if (server_socket == INVALID_SOCKET)
                        return;

                  TCPHandlerState = SM_LISTENING;
                  break;

            case SM_LISTENING:
                  // See if anyone is connected to us
                  if (!TCPIsConnected(server_socket))
                  {
                        // No-one is connected; clear connection flag
                        LED1_IO = 0;
                        return;
                  }

                  // Set connection flag
                  LED1_IO = 1;
```

```
        // Check for incoming data
        if (data_length = TCPIsGetReady(server_socket))
        {
              TCPGetArray(server_socket, TCPBuffer, data_length);
              TCPBuffer[data_length] = '\0';
              TCPDiscard(server_socket);

              TCPHandlerState = SM_PROCESSING;
        }
        break;

  case SM_PROCESSING:
        if (TCPBuffer[0] == 'C' && TCPBuffer[1] == '-')
        {
              // C- signals an XBee AT command

              // Extract DESTINATION ADDRESS HIGH
              j = 0;
              for (i = 2; i < 10; i++)
                    DAH[j++] = TCPBuffer[i];
              DAH[j] = '\0';

              // Extract DESTINATION ADDRESS LOW
              j = 0;
              for (i = 10; i < 18; i++)
                    DAL[j++] = TCPBuffer[i];
              DAL[j] = '\0';

              // Send AT command mode sequence to coordinator
                 XBee
              WriteUSART('+');
              while (BusyUSART());
              WriteUSART('+');
              while (BusyUSART());
              WriteUSART('+');

              // Wait 1/4 second for acknowledgement
              Timer = TickGet();
              do
              {
                    if (DataRdyUSART())
                    {
                          getsUSART(&UReply, 3);

                          // Set command mode flag
                          LED2_IO = 1;
                          break;
                    }
              } while (TickGet()-Timer < TICK_SECOND/4);

              // If command mode flag is not set, halt further
                 processing
              if (!LED2_IO)
              {
                    // Set error flag
                    LED3_IO = 1;
              }
              else
              {
                    // In command mode...
```

```
// Pass DESTINATION ADDRESS HIGH to
   coordinator XBee
sprintf(UBuffer, "ATDH %s\r", DAH);
putsUSART(UBuffer);

// Set error flag - it will be cleared if the
   XBee returns OK
LED3_IO = 1;

// Wait 1 second for acknowledgement
Timer = TickGet();
do
{
      if (DataRdyUSART())
      {
            if (ReadUSART() == 'O')
            {
                  // OK - clear error flag
                  getsUSART(&UReply, 2);
                  LED3_IO = 0;
            }
            else
            {
                  // ERROR
                  getsUSART(&UReply, 5);
            }
            break;
      }
} while (TickGet()-Timer < 1*TICK_SECOND);

// Continue processing only if there was no
   error
if (!LED3_IO)
{
      // Pass DESTINATION ADDRESS LOW to
         coordinator XBee
      sprintf(UBuffer, "ATDL %s\r", DAL);
      putsUSART(UBuffer);

      // Set error flag - it will be cleared
         if the XBee returns OK
      LED3_IO = 1;

      // Wait 1 second for acknowledgement
      Timer = TickGet();
      do
      {
            if (DataRdyUSART())
            {
                  if (ReadUSART() == 'O')
                  {
                        // OK - clear error
                           flag
                        getsUSART(&UReply,
                                  2);
                        LED3_IO = 0;
                  }
                  else
                  {
                        // ERROR
```

```
                              getsUSART(&UReply,
                              5);
                        }
                        break;
                  }
            } while (TickGet()-Timer <
                     1*TICK_SECOND);
      }

      // Send command mode exit sequence to XBee
      // Sending individual characters stops a nul
         char being sent to the destination XBee
      //
      // If exit command fails, command mode will
         eventually timeout anyway
      //
      WriteUSART('A');
      while (BusyUSART());
      WriteUSART('T');
      while (BusyUSART());
      WriteUSART('C');
      while (BusyUSART());
      WriteUSART('N');
      while (BusyUSART());
      WriteUSART('\r');

      // Set error flag - it will be cleared if the
         XBee returns OK
      LED3_IO = 1;

      // Wait 1 second for acknowledgement
      Timer = TickGet();
      do
      {
            if (DataRdyUSART())
            {
                  if (ReadUSART() == 'O')
                  {
                        getsUSART(&UReply, 2);

                        // Clear command mode and
                           error flags
                        LED2_IO = 0;
                        LED3_IO = 0;
                  }
                  else
                  {
                        getsUSART(&UReply, 5);
                  }
                  break;
            }
      } while (TickGet()-Timer < 1*TICK_SECOND);

}     // endif (LED1_IO)

// Prepare TCP reply
if (LED2_IO || LED3_IO)
{
      // There was an error
      TCPPut(server_socket, 'E');
```

```
                TCPPut(server_socket, 'R');

                // Clear the flags
                LED2_IO = 0;
                LED3_IO = 0;
          }
          else
          {
                // Command procedure was successful
                TCPPut(server_socket, 'O');
                TCPPut(server_socket, 'K');
          }

          // Send reply
          TCPFlush(server_socket);
    }
    else if (TCPBuffer[0] == 'D' && TCPBuffer[1] == '-')
    {
          // D- signals RF data

          // Forward message to destination XBee
          putsUSART(TCPBuffer+2);

          // Set error flag - it will be cleared if the
             destination XBee returns OK
          LED4_IO = 1;

          // Wait 5 seconds for acknowledgement
          Timer = TickGet();
          do
          {
                if (DataRdyUSART())
                {
                      getsUSART(&UReply, 4);
                      UReply[3] = '\0';

                      // Forward reply to server_socket
                         buffer
                      for (j = 0; j < 3; j++)
                      {
                            TCPPut(server_socket, UReply[j]);
                      }
                      state = UReply[2];

                      for (i=0; i<16; i++)
                      {
                            temp[i] = LCDText[i];
                      }
                      strcpypgm2ram(LCDText, "TCP Request");
                      for (i=0; i<16; i++)
                      {
                            LCDText[i+16] = temp[i];
                      }
                      LCDUpdate();

                      // OK - clear the flag
                      LED4_IO = 0;

                      break;
                }
          } while (TickGet()-Timer < 2*TICK_SECOND);
```

```
                        if (LED4_IO)
                        {
                              // The XBee response timed out - prepare
                                 error reply
                              TCPPut(server_socket, 'E');
                              TCPPut(server_socket, 'R');

                              // Clear the flag
                              LED4_IO = 0;
                        }

                        // Send reply
                        TCPFlush(server_socket);
                  }
                  else if (TCPBuffer[0] == '^' && TCPBuffer[1] == 'Z')
                  {
                        // ^Z signals remote host disconnect
                        //
                        // Controller will automatically detect remote
                           disconnect
                        // Nothing to do here
                  }
                  else
                  {
                        // Invalid data was received
                        strcpypgm2ram(LCDText, "Invalid data");
                        LCDUpdate();
                  }

                  TCPHandlerState = SM_LISTENING;
                  break;
      }
}
```

D.2 UHandler.c

```
#define __UHANDLER_C

#include "TCPIPConfig.h"
#include "TCPIP Stack/TCPIP.h"

#define CLIENT_PORT 3490
#define ERRMSG "Unhandled Notif."

static BYTE NotifyAddr[] = "10.1.1.4";

void UHandler(void)
{
      static TICK Timer;
      static TCP_SOCKET client_socket = INVALID_SOCKET;
      static BYTE UBuffer[17];
      BYTE temp[16];
      BYTE i;

      static enum _UHandlerState
      {
            SM_LISTENING = 0,
            SM_NOTIFYING1,
            SM_NOTIFYING2,
            SM_NOTIFYING3
      } UHandlerState = SM_LISTENING;

      switch (UHandlerState)
      {
            case SM_LISTENING:
                  if (DataRdyUSART())
                  {
                        Timer = TickGet();

                        i = 0;
                        do
                        {
                              if (DataRdyUSART())
                                    getsUSART(&UBuffer[i++], 1);
                        } while (TickGet()-Timer < (TICK_SECOND/4));
                        UBuffer[i] = '\0';

                        if (UBuffer[0] == 'N' && UBuffer[1] == '-')
                        {
                              // N- signals an incoming notification
                              UHandlerState = SM_NOTIFYING1;
                        }
                        else if (UBuffer[0] == 'A' && UBuffer[1] == 'D')
                        {
                              // AD signals a switch advertisement
                              UHandlerState = SM_NOTIFYING1;
                        }
                        else
                        {
                              for (i=0; i<16; i++)
                              {
                                    temp[i] = LCDText[i];
                              }
```

```
				strcpypgm2ram(LCDText, ERRMSG);
				for (i=0; i<16; i++)
				{
					LCDText[i+16] = temp[i];
				}
				LCDUpdate();
			}
		}
		break;

	case SM_NOTIFYING1:
		client_socket = TCPOpen((DWORD)&NotifyAddr[0],
				TCP_OPEN_RAM_HOST, CLIENT_PORT,
				TCP_PURPOSE_GENERIC_TCP_CLIENT);
		if (client_socket == INVALID_SOCKET)
			break;

		Timer = TickGet();

		UHandlerState = SM_NOTIFYING2;
		break;

	case SM_NOTIFYING2:
		if (!TCPIsConnected(client_socket))
		{
			if (TickGet()-Timer > 5*TICK_SECOND)
			{
				TCPDisconnect(client_socket);
				client_socket = INVALID_SOCKET;

				UHandlerState = SM_LISTENING;
			}
			break;
		}

		if (!TCPIsPutReady(client_socket))
			break;

		i = 0;
		while (UBuffer[i] != '\0')
		{
			TCPPut(client_socket, UBuffer[i++]);
		}
		TCPFlush(client_socket);

		for (i=0; i<16; i++)
		{
			temp[i] = LCDText[i];
		}
		strcpy(LCDText, UBuffer);
		for (i=0; i<16; i++)
		{
			LCDText[i+16] = temp[i];
		}
		LCDUpdate();

		UHandlerState = SM_NOTIFYING3;
		break;

	case SM_NOTIFYING3:
		TCPDisconnect(client_socket);
```

```
                    client_socket = INVALID_SOCKET;

                    UHandlerState = SM_LISTENING;
                    break;
            }
            return;
    }
```

E. Switch Firmware Source Code

E.1 Switch 1 (End_device/ex1.c)

```
#include <p18f1220.h>
#include <usart.h>
#include <delays.h>

/* Set configuration bits for use with ICD2 / PIC18F1220:
 *  - set internal oscillator with I/O on RA6 and RA7
 *  - disable watchdog timer
 *  - disable low voltage programming
 *  - enable background debugging
 */
#pragma config OSC = INTIO2
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = ON

/* Interrupt routines */
void rx_handler(void);
void tmr_handler(void);

/* High priority ISR */
#pragma code rx_interrupt = 0x8
void rx_int(void)
{
      _asm goto rx_handler _endasm
}
#pragma code

/* Low priority ISR */
#pragma code tmr_interrupt = 0x18
void tmr_int(void)
{
      _asm goto tmr_handler _endasm
}
#pragma code

#pragma interrupt rx_handler
void rx_handler(void)
{
      unsigned char comm[5];

      /* Get data received from the USART */
      getsUSART((char*)&comm, 5);   // Expect SW-X\0 ([action]-
                                       [state][nul])

      if (comm[0] == 'S' && comm[1] == 'W')
      {
            if (comm[3] == '0')
            {
                  /* Turn relay OFF */
                  PORTAbits.RA1 = 0;
                  /* Device server expects 4 characters in ACK (3 + nul) */
                  putrsUSART((const far rom char*)"OK0");
            }
            else if (comm[3] == '1')
```

```
        {
              /* Turn relay ON */
              PORTAbits.RA1 = 1;
              /* Device server expects 4 characters in ACK (3 + nul) */
              putrsUSART((const far rom char*)"OK1");
        }
        else
        {
              /* Invalid switch state - do nothing and ACK with error
                 */
              putrsUSART((const far rom char*)"ERR");
        }
  }
  else if (comm[0] == 'I' && comm[1] == 'D')
  {
        /* Store device id in FLASH memory (0x000FF0).
         * This corresponds to the device id in the database
         * and ensures that the correct entry is updated
         * following a notification.
         * The id remains stored even if the device is reset.
         */

        /* Disable interrupts */
        INTCONbits.GIE = 0;
        /* Load TBLPTR with address 0x000FF0 */
        TBLPTRU = 0x00;
        TBLPTRH = 0x0F;
        TBLPTRL = 0xF0;
        /* Point to FLASH program memory */
        EECON1bits.CFGS = 0;
        EECON1bits.EEPGD = 1;
        /* Enable write to memory */
        EECON1bits.WREN = 1;
        /* Enable row erase operation */
        EECON1bits.FREE = 1;
        /* Write sequence (CPU stall) */
        EECON2 = 0x55;
        EECON2 = 0xAA;
        EECON1bits.WR = 1;
        Nop();
        /* Load TABLAT with device id (hex) */
        TABLAT = comm[3];
        _asm TBLWT _endasm
        /* Write sequence (CPU stall) */
        EECON2 = 0x55;
        EECON2 = 0xAA;
        EECON1bits.WR = 1;
        Nop();
        /* Disable write to memory */
        EECON1bits.WREN = 0;
        /* Re-enable interrupts */
        INTCONbits.GIE = 1;

        /* ACK with current switch state */
        if (PORTAbits.RA1)
        {
              putrsUSART((const far rom char*)"OK1");
        }
        else
        {
              putrsUSART((const far rom char*)"OK0");
```

```
			}
		}
		else
		{
			/* Invalid command - do nothing and ACK with error */
			putrsUSART((const far rom char*)"ERR");
		}

		/* Clear interrupt flag */
		PIR1bits.RCIF = 0;
	}

	#pragma interrupt tmr_handler
	void tmr_handler(void)
	{
		/* Toggle LED */
		PORTAbits.RA2 ^= 1;
		/* Clear interrupt flag */
		INTCONbits.TMR0IF = 0;
	}

	void main(void)
	{
		/* Configure the internal oscillator as system clock (8MHz) */
		OSCCON = 0x72;
		OSCTUNE = 0x00;

		/* Configure PORTA - set RA2 and RA5 as inputs */
		PORTA = 0x00;
		TRISA = 0x21;

		/* Configure A/D for digital inputs */
		ADCON1 = 0x7F;

		/*
		 * Open the USART configured as
		 * 8N1, 19200 baud, in polled mode.
		 * Note that 25 is the appropriate
		 * value for BRGH=1 and Fosc=8MHz.
		 */
		OpenUSART(USART_TX_INT_OFF &
		              USART_RX_INT_ON &
		              USART_ASYNCH_MODE &
		              USART_EIGHT_BIT &
		              USART_CONT_RX &
		              USART_BRGH_HIGH, 25);

		/* Enable interrupt priority */
		RCONbits.IPEN = 1;
		/* Make receive interrupt high priority */
		IPR1bits.RCIP = 1;
		/* Enable all high priority interrupts */
		INTCONbits.GIEH = 1;

		/* Configure Timer0 - 16-bit, 1:16 prescaler, internal clock source
		   */
		T0CON = 0x93;
		/* Enable Timer0 interrupt */
		INTCONbits.TMR0IE = 1;
		/* Make Timer0 interrupt low priority */
		INTCON2bits.TMR0IP = 0;
	/* Enable all low priority interrupts */
	INTCONbits.GIEL = 1;

	/* Loop forever */
	while (1)
	{
		if (!PORTAbits.RA0)
		{
			/* The switch button was pressed - toggle the relay */
			PORTAbits.RA1 ^= 1;

			/* Notify server of change */
			putcUSART('N');
			while (BusyUSART());
			putcUSART('-');
			/* Read device id from FLASH memory */
			TBLPTRU = 0x00;
			TBLPTRH = 0x0F;
			TBLPTRL = 0xF0;
			_asm TBLRD _endasm
			while (BusyUSART());
			putcUSART((char)TABLAT);
			while (BusyUSART());
			putcUSART('-');
			while (BusyUSART());
			putcUSART('S');
			while (BusyUSART());
			putcUSART('W');
			while (BusyUSART());
			putcUSART('-');
			while (BusyUSART());
			if (PORTAbits.RA1)
			{
				putcUSART('1');
			}
			else
			{
				putcUSART('0');
			}

			/* Delay to combat switch bounce */
			Delay10KTCYx(100);
		}
	}
}
```

E.2 Switch 2 (End_device2/ex1.c)

```
#include <p18f1220.h>
#include <delays.h>
#include <string.h>
#include <usart.h>

/* Set configuration bits for use with ICD2 / PIC18F1220:
 *  - set internal oscillator with I/O on RA6 and RA7
 *  - disable watchdog timer
 *  - disable low voltage programming
 *  - enable background debugging
 */
#pragma config OSC = INTIO2
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = ON

/* Interrupt routines */
void rx_handler(void);

/* High priority ISR */
#pragma code rx_interrupt = 0x8
void rx_int(void)
{
      _asm goto rx_handler _endasm
}
#pragma code

#pragma interrupt rx_handler
void rx_handler(void)
{
      if (INTCON3bits.INT2IF)
      {
            /* This interrupt sometimes spontaneously occurs.
             * Check that the button is indeed pressed to combat this.
             */
            if (!PORTBbits.RB2)
            {
                  /* The switch button was pressed - toggle the relay */
                  PORTAbits.RA0 ^= 1;

                  /* Notify server of change */
                  putcUSART('N');
                  while (BusyUSART());
                  putcUSART('-');
                  /* Read device id from FLASH memory */
                  TBLPTRU = 0x00;
                  TBLPTRH = 0x0F;
                  TBLPTRL = 0xF0;
                  _asm TBLRD _endasm
                  while (BusyUSART());
                  putcUSART((char)TABLAT);
                  while (BusyUSART());
                  putcUSART('-');
                  while (BusyUSART());
                  putcUSART('S');
                  while (BusyUSART());
                  putcUSART('W');
                  while (BusyUSART());
                  putcUSART('-');
                  while (BusyUSART());
                  if (PORTAbits.RA0)
                  {
                        putcUSART('1');
                  }
                  else
                  {
                        putcUSART('0');
                  }
            }

            /* Delay to combat switch bounce */
            Delay10KTCYx(100);

            /* Clear interrupt flag */
            INTCON3bits.INT2IF = 0;
      }
      else if (PIR1bits.RCIF)
      {
            unsigned char comm[5];

            /* Get data received from the USART */
            getsUSART((char*)&comm, 5);   // Expect SW-X\0 ([action]-
                                             [state][nul])

            if (comm[0] == 'S' && comm[1] == 'W')
            {
                  if (comm[3] == '0')
                  {
                        /* Turn relay OFF */
                        PORTAbits.RA0 = 0;
                        /* Device server expects 4 characters in ACK (3 +
                           nul) */
                        putrsUSART((const far rom char*)"OK0");
                  }
                  else if (comm[3] == '1')
                  {
                        /* Turn relay ON */
                        PORTAbits.RA0 = 1;
                        /* Device server expects 4 characters in ACK (3 +
                           nul) */
                        putrsUSART((const far rom char*)"OK1");
                  }
                  else
                  {
                        /* Invalid switch state - do nothing and ACK with
                           error */
                        putrsUSART((const far rom char*)"ERR");
                  }
            }
            else if (comm[0] == 'I' && comm[1] == 'D')
            {
                  /* Store device id in FLASH memory (0x000FF0).
                   * This corresponds to the device id in the database
                   * and ensures that the correct entry is updated
                   * following a notification.
                   * The id remains stored even if the device is reset.
                   */
```

```
                  /* Disable interrupts */
                  INTCONbits.GIE = 0;
                  /* Load TBLPTR with address 0x000FF0 */
                  TBLPTRU = 0x00;
                  TBLPTRH = 0x0F;
                  TBLPTRL = 0xF0;
                  /* Point to FLASH program memory */
                  EECON1bits.CFGS = 0;
                  EECON1bits.EEPGD = 1;
                  /* Enable write to memory */
                  EECON1bits.WREN = 1;
                  /* Enable row erase operation */
                  EECON1bits.FREE = 1;
                  /* Write sequence (CPU stall) */
                  EECON2 = 0x55;
                  EECON2 = 0xAA;
                  EECON1bits.WR = 1;
                  Nop();
                  /* Load TABLAT with device id (hex) */
                  TABLAT = comm[3];
                  _asm TBLWT _endasm
                  /* Write sequence (CPU stall) */
                  EECON2 = 0x55;
                  EECON2 = 0xAA;
                  EECON1bits.WR = 1;
                  Nop();
                  /* Disable write to memory */
                  EECON1bits.WREN = 0;
                  /* Re-enable interrupts */
                  INTCONbits.GIE = 1;

                  /* ACK with current switch state */
                  if (PORTAbits.RA0)
                  {
                        putrsUSART((const far rom char*)"OK1");
                  }
                  else
                  {
                        putrsUSART((const far rom char*)"OK0");
                  }
            }
            else
            {
                  /* Invalid command - do nothing and ACK with error */
                  putrsUSART((const far rom char*)"ERR");
            }

            /* Clear interrupt flag */
            PIR1bits.RCIF = 0;
      }
}

unsigned char UReply[17];
unsigned char UBuffer[17];

void main(void)
{
      /* Configure the internal oscillator as system clock (8MHz) */
      OSCCON = 0x72;
      OSCTUNE = 0x00;
```

```
/* Configure PORTA - set RA5 as input, RA0 as output */
PORTA = 0x00;
TRISA = 0x20;

/* Configure PORTB - set RB2 and RB4 as inputs, RB1 as output */
PORTB = 0x00;
TRISB = 0x14;

/* Configure A/D for digital inputs */
ADCON1 = 0x7F;

Delay10KTCYx(100);

/*
 * Open the USART configured as
 * 8N1, 19200 baud, in polled mode.
 * Note that 25 is the appropriate
 * value for BRGH=1 and Fosc=8MHz.
 */
OpenUSART(USART_TX_INT_OFF &
              USART_RX_INT_ON &
              USART_ASYNCH_MODE &
              USART_EIGHT_BIT &
              USART_CONT_RX &
              USART_BRGH_HIGH, 25);

WriteUSART('+');
while (BusyUSART());
WriteUSART('+');
while (BusyUSART());
WriteUSART('+');

while (!DataRdyUSART());
getsUSART((char*)&UReply, 3);
UReply[3] = '\0';

WriteUSART('A');
while (BusyUSART());
WriteUSART('T');
while (BusyUSART());
WriteUSART('S');
while (BusyUSART());
WriteUSART('H');
while (BusyUSART());
WriteUSART('\r');

while (!DataRdyUSART());
getsUSART((char*)&UBuffer+2, 7);

WriteUSART('A');
while (BusyUSART());
WriteUSART('T');
while (BusyUSART());
WriteUSART('S');
while (BusyUSART());
WriteUSART('L');
while (BusyUSART());
WriteUSART('\r');

while (!DataRdyUSART());
getsUSART((char*)&UBuffer+8, 9);
```

```
      UBuffer[0] = 'A';
      UBuffer[1] = 'D';
      UBuffer[16] = '\0';

      WriteUSART('A');
      while (BusyUSART());
      WriteUSART('T');
      while (BusyUSART());
      WriteUSART('C');
      while (BusyUSART());
      WriteUSART('N');
      while (BusyUSART());
      WriteUSART('\r');

      while (!DataRdyUSART());
      getsUSART((char*)&UReply, 3);
      UReply[3] = '\0';

      putsUSART((char*)&UBuffer);

      /* Clear interrupt flag */
      PIR1bits.RCIF = 0;

      /* Enable interrupt priority */
      RCONbits.IPEN = 1;
      /* Make receive interrupt high priority */
      IPR1bits.RCIP = 1;

      /* Enable INT2 external interrupt */
      INTCON3bits.INT2IE = 1;
      /* Interrupt on falling edge */
      INTCON2bits.INTEDG2 = 0;
      /* Make interrupt high priority */
      INTCON3bits.INT2IP = 1;
      /* Enable peripheral interrupts */
      INTCONbits.PEIE = 1;

      /* Enable all high priority interrupts */
      INTCONbits.GIEH = 1;

      /* Loop forever */
      while (1);
}
```