RFID Library Management System

Final Report
November 17, 2008

University of British Columbia

EECE 375/474

Prepared By:

Group 9

Adnan Jiwani

Jason Poon

Jay Wakefield

Manasi Kulkarni

Mohammed Taher

Abstract

This report describes the method of designing and implementing a Radio Frequency Identification (RFID) to replace the current bar coding system of UBC libraries. As RFID tags contain identifying information, such as a book’s title or author, the information can be directly read from an RFID reader without the aid of a database. The RFID-enabled library catalogue management system will aid patrons in finding a book, help libraries recover misplaced books, and manage check-outs, check-ins, returns, and fines. Additionally, RFID tags can also act as a security device that can replace the electromagnetic electronic article surveillance (EAS) currently being used by UBC libraries. Our report provides a technical description of the hardware and software involved in designing such a system.

Table of Contents

List of Tables

List of Figures

Introduction

As radio frequency identification (RFID) technology continues to mature, the technology is becoming increasingly affordable. As a direct result of the reduced cost for the technology, RFID is being applied in a variety of forms. One area where RFID technology is being utilized is in libraries.

The majority of libraries maintain their catalogue using barcodes. Although barcodes are relatively low cost and extremely accurate compared to manual key entry, there are many limitations to the technology including:

1. Bar-coded items may only be scanned one at a time
2. If the barcode becomes scratched or damaged, the barcode reader will be unable to read it
3. There is no simple way to detect misplaced books

RFID poses numerous benefits over the current barcode system being used in University of British Columbia (UBC) libraries including faster scanning of items, a theoretical longer scanning distance, easier to maintain large catalogues, and better theft detection rate. Additionally, as RFID tags are capable of being scanned through objects, circulation staff no longer need to repetitively open a book cover to scan a book's barcode; this can substantially decrease the amount of repetitive-motion injuries.

Our implementation of the RFID Library Management system will provide both patrons and librarians with a simplified self check-out/check-in procedure, a high-speed book inventory tracking system, detection of miss-shelved books, and the ability to guide a user to a book in the library.

This report is sub-divided into 6 sections and will provide an in-depth description of our implementation of an RFID library management system. We describe background and related works in section 1 and the overall project design in section 2. The design requirements and goals for our implementation will be provided in section 3. The results of our implementation including any testing results are described in section 4. Section 5 summaries the testing results. The report is concluded in section 6 with an assessment and analysis of the project.

1.0 Background and Related Work

First introduced in 1970, radio frequency technology is often viewed as a future replacement to the ubiquitous Universal Product Code (UPC) bar coding system. A RFID system is typically composed of three components: a reader antenna, a transceiver to decode data, and a RFID tag (transponder). When a RFID tag's antenna passes through the electromagnetic field of a reader's antenna, the tag transmits radio waves back to the reader where it can be translated into meaningful data. RFID tags come in two general varieties: passive and active. Passive tags have no internal power supply and must harvest power from the reader's electromagnetic field in order to transmit data to the reader. Active RFID tags, on the other hand, contain an internal power source that is used to transmit data to the scanning antenna. As a result of having its own onboard power supply, active RFID tags are capable of broadcasting at higher frequencies that can be read at greater distances than passive tags.

Although RFID adoption in libraries has been slow due to huge cost in converting the library system to support RFID, one area of industry that has been quick to adopt the technology is the security industry. RFID tags and keys are quickly replacing normal 'keys' in allowing users access through doors. Numerous countries have also begun issues RFID-embedded passports.

Research into previous RFID-related projects yielded numerous projects including a project by a group of UBC students titled "RFID Library Circulation System". Their overall design goals are similar to those of our RFID Library Management System. Another EECE 374/474 project we came across was done by a group from UBC during the January 2006 term. The goal of this group was to develop a RFID bus pass system. This group used low frequency as they were scanning one pass at a time. However, they used a Series 2000 micro reader from TI instead of building the RFID reader from scratch. This saved them considerable time as they did not have to make many different circuits for the reader.

2.0 Project Design

The RFID Library Management System can be subdivided into three physical components: the handheld unit, the security gate, and the back-end server. The handheld unit is comprised of multiple blocks including the microcontroller, input/output system, RFID reader module, and the power supply. The back-end server houses the database, web server, and drivers to communicate to the handheld device.

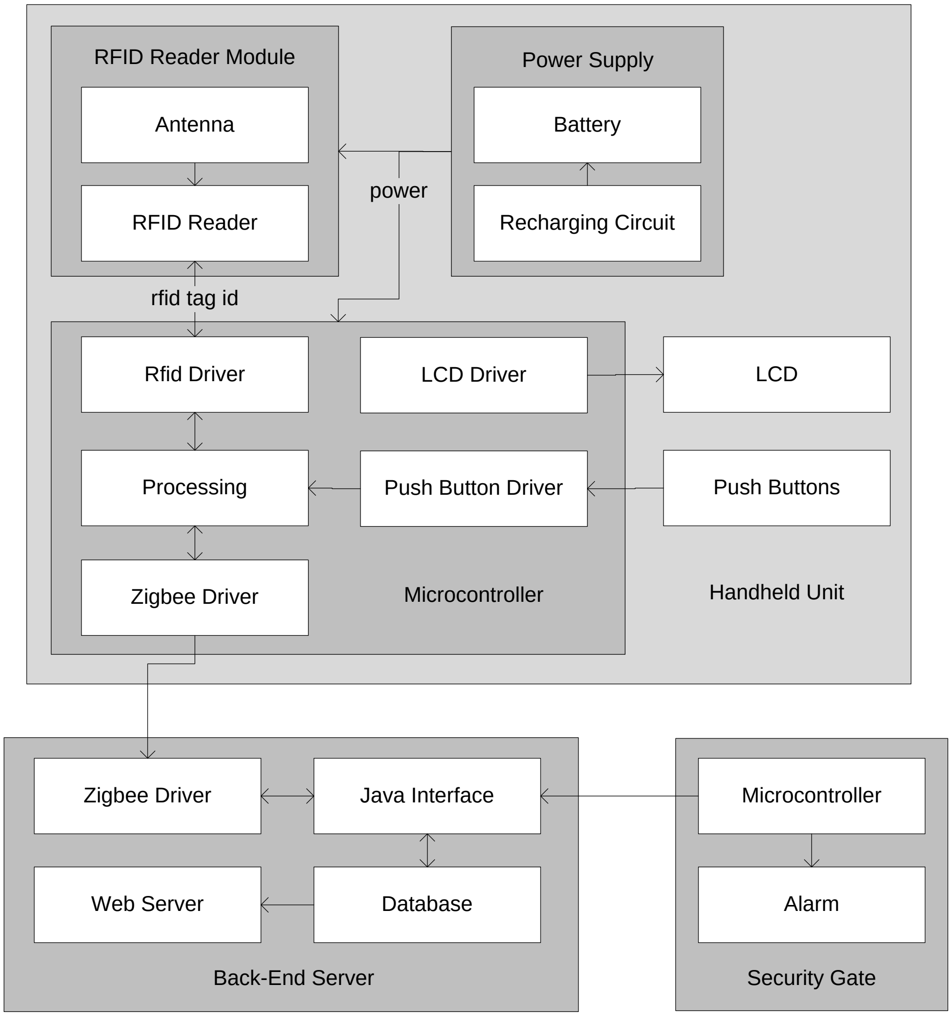


Figure 1 - System Block Diagram

3.0 Requirements and Goals

The hardware and software requirements and goals are described below.

3.1 Hardware Requirements

The below sections outline the requirements of the key hardware sections of the RFID Library Management System.

3.1.1 Remote Handheld Unit

This device will be able to:

1. Read the RFID tags attached to the books and:

- compare the tag information with information stored in its memory (this will happen when a library user is looking for certain book and that list has been downloaded into the handheld) and let the user know of a positive match

- Display information about the book on the LCD screen

- a librarian may use it to add new books to the library and write material information in the tag

2. Receive information from the database via wireless communication (Zigbee).

3. Send RFID tag information to the database

3.1.2 Zigbee module attached to the computer

This unit should be able to:

1. Receive information from the handheld

- The Zigbee module connected to the development kit will be able to receive information from the Zigbee module attached to the hand held device. The information received will then be passed to our database / software via the development board.

2. Send information to the handheld

- The software/database will be able to send information to the other Zigbee module on the handheld via the development board.

Under ideal conditions, the remote handheld unit and the Zigbee module attached to the computer should be able to communicate up to 30m apart (indoor).

3.1.3 Security Gate

Because of budget constraints, we plan on using single RFID reader for both security gate and the hand held device. Therefore we plan to use the hand held device which contains the RFID reader and the microcontroller to act as a security gate for the purpose of demonstration.

3.1.4 Enclosure

We require an enclosure that would be able house the main PCB, the RFID reader module, the Zigbee module, the Antenna circuit and the battery recharger circuit. The enclosure should be made of plastic as a metal enclosure would make the RFID antenna's behavior unpredictable.

3.2 Software Requirements

The below list outlines the requirements that the software component of the RFID library management should achieve:

3.2.1 Web Interface

The web interface should provide the user with the following functionality:

- Check-in/Check-out an item from the catalogue
- Mark a book as missing or damaged
- Renew overdue books
- Add/remove/edit books in the catalogue
- Search the catalogue
- Download a list of books to the handheld unit in order to aid with finding the book in the library
- Register as a library patron and view/edit user specific details
- View all books that the signed-in user has on loan

3.2.2 Handheld Unit

The handheld unit should be able to provide the user with the following functionality:

- Search and mark misplaced books
- Guide user to a book
- Check-in/Check-out books

3.2.3 Security Gate

The security gate should have the following functionality:

- Accurately detect if a book that has been scanned has been checked-out; if book has not been checked-out, sound an alarm

4.0 Project Results Summary

4.1 Handheld Unit

4.1.1 Software Development Environment

Software for the microcontroller is written in the language of C. The Atmel AVRISP mkll programmer, in combination with AVRStudio, the official Atmel integrated development environment, and WinAVR, which contains the GNU GCC compiler, were used to program the microcontroller.

4.1.2 Microcontroller

The first microcontroller we looked at was ATMEL ATMEGA88P. This microcontroller had 8K bytes of flash programmable memory, 512 bytes in EEPROM, 1K bytes in internal SRAM, and one programmable serial USART. The microcontroller met all our requirements apart from the number of programmable serial USART ports. According to our design, the microcontroller needs to communicate with the RFID reader module and the wireless communication module via USART. Therefore, we decided to go with ATMEL ATMEGA162. This microcontroller has similar specifications to ATMEGA88P but had two programmable serial USARTs. The microcontroller supported clock speeds up to 16 MHz which was more than adequate for our project. We chose to work with a DIP packaged microcontroller with a 11.0592MHz external crystal. The microcontroller will be powered with a 5V DC power supply coming from the battery recharger/power supply circuit.

4.1.3 Universal Asynchronous Receiver/Transmitter (UART)

The ATmega 162 contains two Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) that are capable of communicating both synchronously and asynchronously. Both USARTs were set in asynchronous normal mode with 1 start bit, 8 data bits, and 1 stop bit for communicating with the Zigbee chip and the RFID reader.

An external 11.0592 MHz crystal oscillator was utilized allowing us to achieve a baud rate of 9600 with a 0% chance of error. According to the ATmega datasheet, the baud value needed to be written to the USART Baud Rate Register (UBRR) can be calculated by the following formula:

$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

Figure 2 - UBRR Formula

With f_{osc} being 11.0592MHz and the baud rate being 9600, the value of 71 is equated for the UBRR.

4.1.3.1 Zigbee

For wireless communication we considered 2 options: WIFI and Zigbee. WIFI is comparatively more difficult to implement as it requires the writing of many complicated drivers. As such, we decided to go with Zigbee. For Zigbee, we had 2 chips available to us, the Xbee 802.15.4 and Xbee Pro 802.15.4. The main difference between the two chips was that Xbee provided an indoor range of up to 100ft and the Xbee Pro provided an indoor range of 300ft. Since the cost of Xbee pro was almost twice compared to that of Xbee and 100ft was sufficient for our application, we decided to go with Xbee 802.15.4. Furthermore, the Xbee chip comes in two different forms, one with a chip antenna and the other with a wire antenna. Since, the Xbee will be part of the handheld device we decided to buy the chip with a chip antenna on it. Having bought the chip, we then realized that in order to program it, we will need a development board. Therefore, we decided to get a starter development kit that comes with all the necessary parts, like USB development board, USB cable, adapters, and 9V battery clip.

We used Zigbee with both types of antenna. Our Zigbee module looks like the one shown in below.

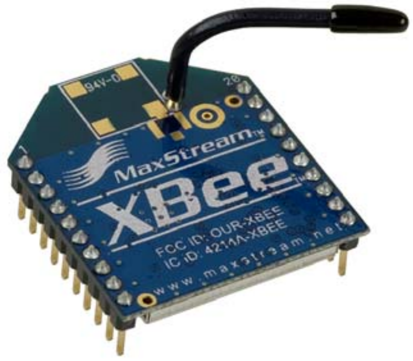


Figure 3 - Zigbee Module with External Antenna

Since the Xbee module requires a supply voltage of 3.3V, the output of the microcontroller's UART transmit pin had to be stepped down from 5V to 3.3V. As 3.3V is exactly 66.67% of 5V, this can simply be achieved by using a voltage divider with three equal resistive values.

With the Zigbee RF module operating in transparent mode, it is capable of both receiving and transmitting data at any one time. As such, it is necessary for the UART to be in asynchronous mode. In order to prevent the Zigbee UART from blocking, both the transmitting and receiving of data are interrupt-driven. To transmit data to the Zigbee module, data is first placed onto a circular buffer where the data on the buffer will be obtained and sent to the module byte by byte. Receiving, on the otherhand, requires the packet to be sent according to the protocol outlined in

Part	Part Number	Cost per Unit	Quantity Purchased	Part Total
Xbee 802.15.4 OEM RF Modules	XB24-ACI-001	$26.53	2	$53.06
Xbee Starter Kit	XBP24-DKS	$179.00	1	$179.00
Atmega 162 Microcontroller	ATMEGA162-16PU-ND	$8.40	2	$16.80
RFID Transponder In-Lay	Free Trial	$1.79	5	$8.95

RFID Transponder In-Lay	481-1067-1-ND	$1.79	5	$8.95
RFID Reader	481-1052-ND	$104.31	1	$104.31
Liquid Crystal Display	LCM-S01604DSR	$25.19	1	$25.19
Voltage Regulator		$0.35	1	$0.35
ISP (Programmer)	ATAVRISP2-ND	$38.89	1	$38.89
Tantulum Capacitor (LE33)	497-4258-1-ND	$1.07	2	$2.14
		$1.11		$0.00
IC Batt Fast Charger	MAX712CPE+-ND	$9.98	1	$9.98
Energizer Rechargeable battery	N705-ND	$15.00	1	$15.00
PNP Transistor	2N6109GOS-ND	$1.55	2	$3.10
Rectifier	1N4001DICT-ND	$0.39	2	$0.78
1.0UF 50V Mini Alum Elect	P824-ND	$0.18	2	$0.36
0.01UF Capacitor	399-4150-ND	$0.21	2	$0.42
Magnetic antenna wire	Active electronics	$12.31	1	$12.31
Header pins	MRO Electronics	$10.08	1	$10.08
SMA Connector	MRO Electronics	$9.59	1	$9.59
Power Adapter	MRO Electronics	$18.03	1	$18.03
Miscellaneous				$8.00
Shipping and Handling		$8.00	4	$32.00
Sub-Total*				**$378.29**
Taxes (GST)		**5.00%**		**$18.91**
Total*				**$397.20**
Total with Xbee starter kit				$557.29

* As the Xbee Starter Kit will not be reimbursed, the value can be removed from the total

Appendix B – Communication Protocol. Once an entire packet has been successfully received, a software interrupt will parse the packet and perform any required instruction.

4.2 The communication protocol used between the Zigbee module module connected to the back-end server is outlined in section 4.3

As the web interface and the handheld device are often in communication, and as the handheld device is constantly communicating with the database, an interface was required to deal with requests from both the web interface and the handheld device. In order to accomplish this, the design for the interface had to be able to handle controlling the XBee Driver through the serial port of the base station. After a failed trial and error with several PHP plug-ins for the serial port, a java library was discovered that made communicating with the serial port fairly straight forward. Thus, java was chosen as the primary backend interface language for several reasons:

- Provided the easiest way to send and receive information from the serial port
- Provided multithreading capabilities, that allow both reading and writing from the serial port at what seems like the same time
- Provided networking sockets for communication between several threads running at any given time

Sun Microsystems has supported and developed an API for communicating with the serial port. This API was commonly known as sun's javax.comm.* API. However, this API has been discontinued for windows operating systems. On further research it was determined that Sun still allows download of this API for windows, however it does not provide the implementation classes needed for use. There is an open source group named RXTX, that provide this implementation. Once Sun's API and RTXT's implementation has been installed into the base stations Java Runtime Environment (JRE), it became very easy and reliable to communicate with the serial port of the computer, and send information to the XBee card. This was the primary reason for choosing java as the backend language, as well as the other reasons listed above. Information on RXTX can be found on their website at http://users.frii.com/jarvi/rxtx/.

In order to work as a backend communication server, the java interface requires multithreading. It is expected that the handheld device may send data to the server's XBee device at any time. In addition, the database or web interface may need to send data to the handheld device at any time. In order to accomplish this, three main threads are started and are run constantly to allow communication with the XBee device connected to the serial port. These three main threads are a writing thread, a reading thread, and a thread that handles writing from the web interface. These threads handle the various forms of communication that is needed with the XBee device. The extra thread for writing from the web interface is needed, because it is a special case of writing. The main write thread writes data from the system input stream. The web interface does not write to this input stream, but rather sends data from it's own new java thread. Therefore, the extra thread is needed to handle writing information that does not come from the system input stream.

Any communication that does not require the web interface is handled all within the already running, three main threads. These operations include checking in and out books from the library via the handheld device, adding misplaced books to the misplaced book check list, and the security gate checking. These simply communicate with the database, and do not need to communicate with other running threads on the system. The protocol would simply be received, and the appropriate database operation would be run using the current thread, and the JDBC (Java Database Connector). There is no need for any other threads to be run, or communicated for such a process. However, there are some operations that are either started by or need to communicate with the web interface. This situation is slightly more complicated, and requires net communication via java sockets.

As part of PHP, there is a command that allows outside programs to be run outside the script, as if they were started on the command line of the underlying system. This command is “shell_exec (String command).” This command takes a string as its parameter that is the exact command that you would execute from the shell of the operating system, and it returns a string containing all of the output from the program that is run. We utilize this command on our system to send and receive information from the handheld device to or from the web interface. Whenever this occurs, a new java program is started from the PHP web interface that deals with the exact request from the user on the web page. It then sends the information out through the serial port. However, because this new program is a new thread on the Java Virtual Machine (JVM), and the thread that communicates with the XBee device is also its own thread in the JVM, the program launched from the web interface needs some way to send information to the communication thread, and out to the XBee device. This is accomplished by using java sockets. We have written a server that deals with communications between threads, and runs as its own thread in the background. This server passes information from thread to thread and allows communication from the web interface to the serial port communication thread. Whenever a PHP-java thread is started, it sets up a socket with the server. At the same time, the receiving end either starts, or has already started its own socket with the server to send and receive any information that involves the web interface. Doing this allows the communication between the Java backend serial port/Xbee communication thread, and any threads that are started and ended via the web interface, thus allowing communication between the web interface and the handheld device.

Apart from managing different operations of functionality, the java interface is also responsible for parsing an 8 byte RFID number into a long integer for storage on the database. Because there is no such variable as an unsigned byte in Java, integers are used to store the 8 byte RFID number revieved from the handheld device to allow the range of the byte, by mimicking an unsigned byte. The number is then parsed into an integer using the following code:

```
public static long bytesToLong( int[] data ){

            long RFIDnum = 0;

            for( int i = 0; i < data.length; i++ ){
                  RFIDnum = RFIDnum << 8; //shift left by two HEX digits
                  RFIDnum += data[i];
            }
            return RFIDnum;
    }
```

The reverse operation parsing can also be achieved by parsing the integer into an array of integers in much the same fashion. Note that because there are no unsigned variables in Java or in MySQL, this algorithm results in a negative long integer if the most significant bit of the 8 byte RFID packet is set.

Information is passed and received between the handheld and java interface using a set protocol. This protocol provides information to the java interface about what operation to perform, and the data needed for that operation. For details on this protocol, please see appendix B.

The challenges with this approach were mostly allowing communication between backend processes that could be started and stopped at any time from the web interface. However once we experimented with the network sockets, it was confirmed that java was sufficient for allowing a backend constantly running process that handles communication between the XBee and handheld devices, the MySQL server, and the web interface.

Communication Protocol.

4.2.1.1 Radio Frequency Identification (RFID) Reader

4.2.1.1.1 Hardware

Initially, we wanted to build the RFID reader from scratch by connecting parts to the reader chip. The first reader chip we considered was the TRF7960 from TI. The TRF7960 was a fully integrated 13.56MHz RFID reader system and was capable of reading and writing to HF RFID tags. The only drawback it had was that it was only 5.15mm X 5.15mm in size with 32 pins on it. Due to the small size and high number of pins, we decided that it will be very difficult and time consuming to work with it and therefore, rejected it. The second reader chip we considered was the S6700 multi-protocol transceiver IC from TI. The S6700 had similar specification to that of the TRF7960 and was comparatively bigger in size which would make its implementation easier. However, after some more research and on the advice of our TA, we decided not to use it in our project as it required complicated circuitry for it to work.

Taking into account out time constraints, we decided to buy a RFID reader *module* instead of a reader chip. This would eliminate creation of any complicated circuit for the RFID reader. We decided to go for the S4100 Multifunction reader module from Texas Instruments.

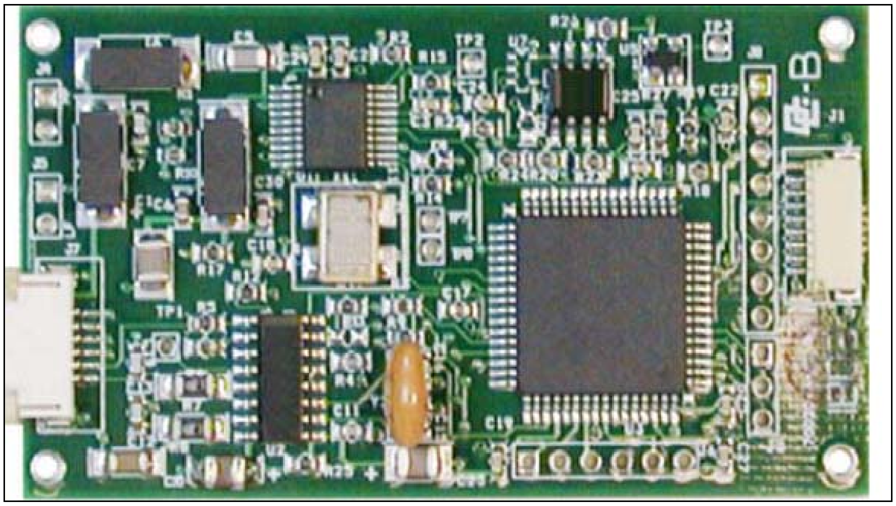


Figure 4 - S4100 Multi-Function Reader Module (Texas Instruments)

The S4100 reader module supports both, high frequency and low frequency applications. The module also contains a microprocessor, which does the final decoding of the signal received and provides an output that can be connected to a microcontroller/computer. The module requires an antenna with 50 Ohms impedance at 13.56 MHz. Using a 1.5” diameter antenna and TI’s ISO 15693 compliant rectangular inlay, the read range can be 4” or greater, which is sufficient for our application. Since, only an antenna and no external complicated circuitry are required to start using this module, we decided that this will be the best option as it will save time and make our device more efficient.

The schematic for the module is shown below:

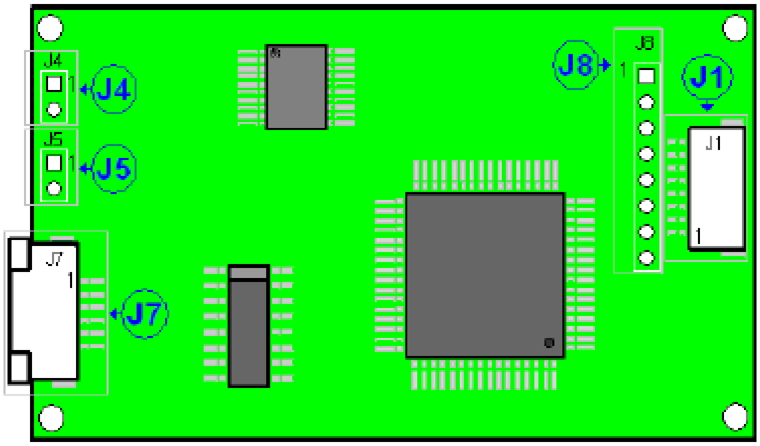
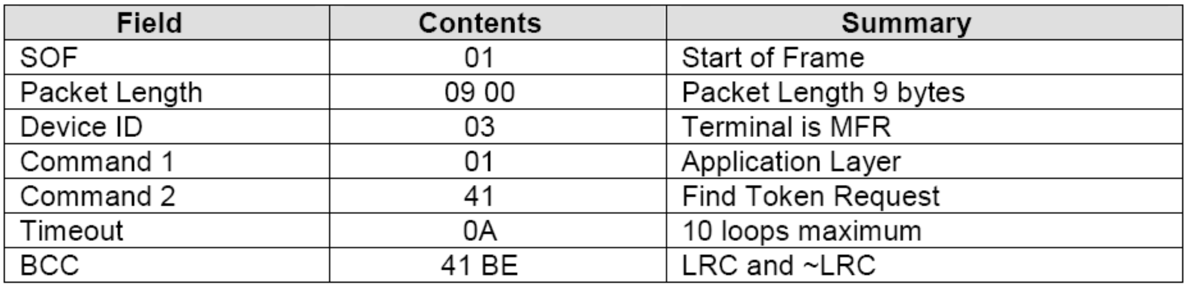


Figure 5 - S4100 Reader Module with Different Connectors

Connectors J8 is used for power (5V), ground and UART communication. The J4 connector is used for attaching the antenna to the module via coaxial cable. No other connectors were used.

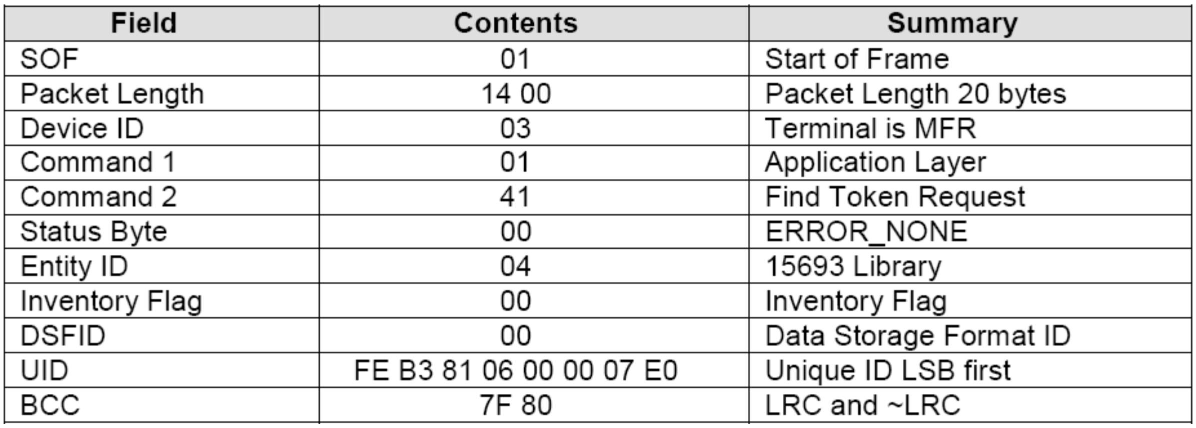
4.2.1.1.2 Software

Similar to Zigbee module the S4100 RFID reader module is capable of both receiving and transmitting data at any one time, which therefore forced the USART to be in asynchronous mode. Also to prevent RFID UART from blocking, both the transmitting and receiving of data are interrupt-driven. When the user presses any function that requires the reading of the RFID tag, the request packet (Figure 6) is first put in a circular array and than sent byte by byte. For the request packet we had many choices. Since we are using Tag-it tags, which are also compatible with ISO-15693, we could change the command 1 in Figure 6 to 05 and look for the Tag-it tag only. However, using this we would get only 4 byte long tags and as a group we had decided in our design to have 8 byte long tags. We could also replace the command 1 with 04 and look for ISO-15693 tags only. This would unable the detection of multiple tags and possibly speed the whole process of searching for books. However, our antenna is not very consistent in detecting all tags in its range and we therefore decided to scan one tag at a time only. The advantage of the packet in Figure 6 is that if multiple tags are detected we get a collision error, which enables us to remove the ambiguity of whether all tag are successfully scanned or not. For receiving, things are a lot a simpler. The response packet (Figure 7) is first put in a array than the tag starting from byte 10 is parsed and returned to the calling function. There is also a find function in RFID that simply compares 2 tags, the tag scanned and the tag of the book user is trying to find, and returns the result in Boolean form.



Field	Contents	Summary
SOF	01	Start of Frame
Packet Length	09 00	Packet Length 9 bytes
Device ID	03	Terminal is MFR
Command 1	01	Application Layer
Command 2	41	Find Token Request
Timeout	0A	10 loops maximum
BCC	41 BE	LRC and ~LRC

Figure 6 – RFID common request packet (Texas Instruments, 2003)



Field	Contents	Summary
SOF	01	Start of Frame
Packet Length	14 00	Packet Length 20 bytes
Device ID	03	Terminal is MFR
Command 1	01	Application Layer
Command 2	41	Find Token Request
Status Byte	00	ERROR_NONE
Entity ID	04	15693 Library
Inventory Flag	00	Inventory Flag
DSFID	00	Data Storage Format ID
UID	FE B3 81 06 00 00 07 E0	Unique ID LSB first
BCC	7F 80	LRC and ~LRC

Figure 7 – RFID ISO-15693 response packet (Texas Instruments, 2003)

4.2.2 User Interface

A user is able to visually toggle through the various states of the handheld by using one of the four push buttons. Once a push button has been pressed, it will cause a certain output to be displayed on the liquid crystal display.

4.2.2.1 Liquid Crystal Display (LCD)

In order to extend the battery life of the handheld, we decided to purchase a liquid crystal display (LCD) without the backlight feature. Additionally, a 4-line character-based display would be sufficient to display all required information to the user. The Lumex S01604DSR 16x4 character-based LCD fit these requirements.

4.2.2.1.1 Pin Configuration

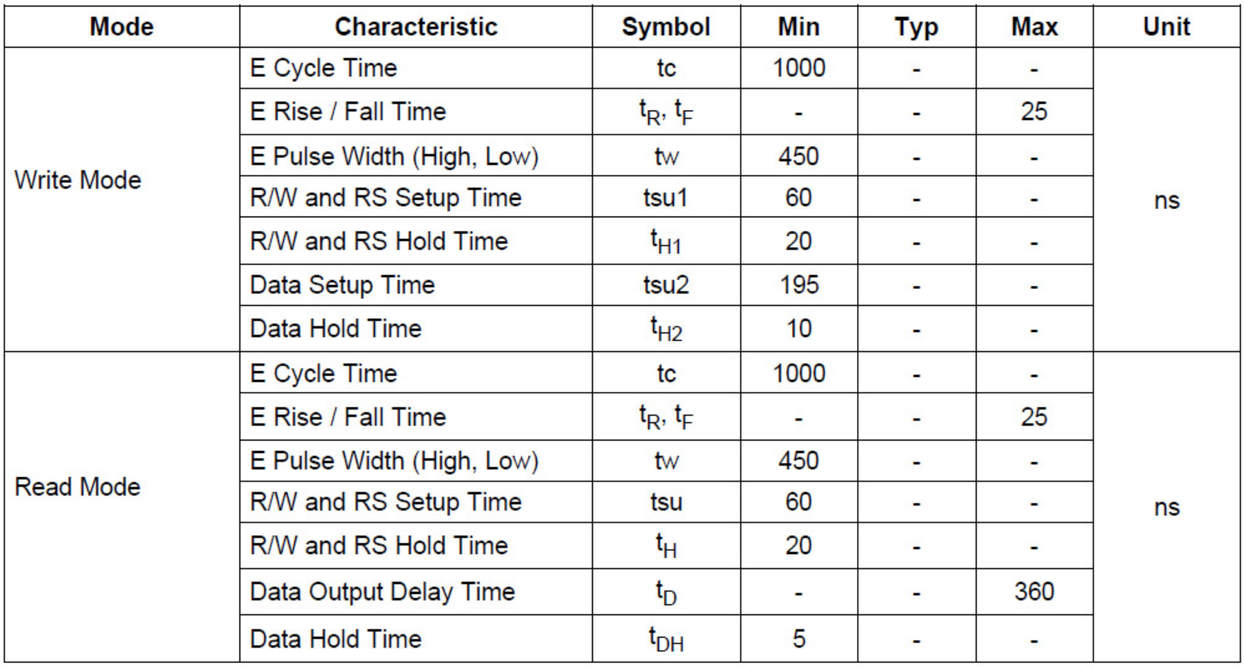
As described by Table 1, the Lumex S01604DSR has a total 14 connections: 3 power lines, 3 control lines, and 8 data lines.

Pin No.	Name	Function
1	Vss	Ground
2	Vdd	+5V Power Supply
3	Vo	Driving Voltage (Contrast)
4	RS	Register Select - Low – data is treated as instructions - High – data is treated as text to display
5	R/W	Read/Write - Low – write commands or character data - High – read LCD registers
6	E	Enable - High – start of data transmission - Low – end of data transmission
7-14	DB[0..7]	Data bits

Table 1 - Lumex S011604DSR Pin Configuration

4.2.2.1.2 Software Driver

The operation of the S01604DSR is similar to that of other LCDs. Namely, the three control lines (RS, RW, EN) need to be set in to certain low/high values before a certain operation may be performed. A unique issue in designing the LCD driver was to ensure that all timing requirements were met. The below table describe the timing requirements for the S01604DSR.



Mode	Characteristic	Symbol	Min	Typ	Max	Unit
Write Mode	E Cycle Time	tc	1000	-	-	ns
	E Rise / Fall Time	t_R, t_F	-	-	25	
	E Pulse Width (High, Low)	tw	450	-	-	
	R/W and RS Setup Time	tsu1	60	-	-	
	R/W and RS Hold Time	t_{H1}	20	-	-	
	Data Setup Time	tsu2	195	-	-	
	Data Hold Time	t_{H2}	10	-	-	
Read Mode	E Cycle Time	tc	1000	-	-	ns
	E Rise / Fall Time	t_R, t_F	-	-	25	
	E Pulse Width (High, Low)	tw	450	-	-	
	R/W and RS Setup Time	tsu	60	-	-	
	R/W and RS Hold Time	t_H	20	-	-	
	Data Output Delay Time	t_D	-	-	360	
	Data Hold Time	t_{DH}	5	-	-	

Figure 8 - LCD Timing Requirements (Lumex, 1998)

In addition to the above timing requirements, the initialization of the LCD also required timing considerations. The initialization of the display includes the set function, display on/off, display clear, and entry mode set operations and each operation required a certain time delay prior to proceeding.

4.2.2.2 Push Buttons

Four buttons (left/back, up, down, right/enter) on the handheld unit provide the user with full accessibility to the handheld functions.

As push-buttons are mechanical switches they exhibit ‘switch bounce’ where the electrical signal will bounce for several microseconds before reaching stability. Figure 9 shows the effects of switch bounce.

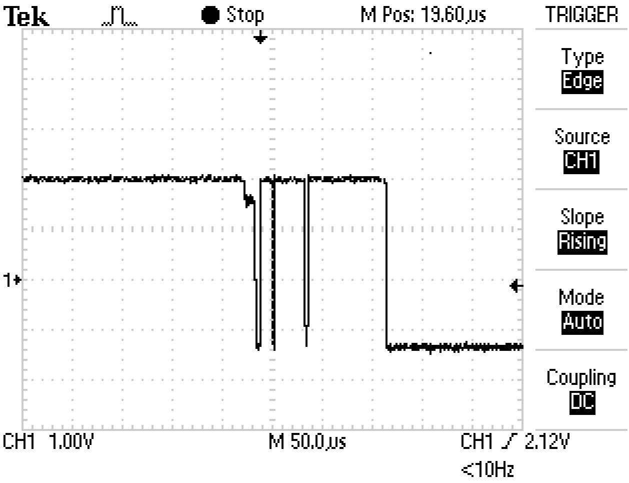


Figure 9 - Switch Bounce

The effects of switch bouncing can be mitigated through various means. In order to detect any transition changes with the push buttons, the state of the push buttons were constantly being polled by the microcontroller through the use of a timer overflow interrupt. As a result of the constant polling, switch de-bouncing can be easily handled in software. When the software detects a change in the push button state, it starts incrementing a counter, each time re-reading the push button state, until it reaches a safe, bounce-free count. If the state is not stable, the counter is reset to its initial value.

Having the push-buttons interrupt-driven also enables the user to be able to “push-and-hold” the push buttons. As a result, the user can simply push and hold down a certain button instead of repeatedly pushing down on the button to toggle between the different menus.

4.2.2.3 Menu System

In designing the menu system, a large emphasis was placed on making the menu system easily maintainable and able to support a hierarchical system (i.e. submenus, submenus of submenus, etc.). The menu system accepts inputs from the four push buttons and executes the currently selected function. The ‘up’, ‘down’ buttons are primarily used from scrolling up or down a page to view further menu items. The ‘left’ button serves two purposes; pressing the ‘left’ button would either cancel the currently executed function or view the previous menu. Similar to the ‘left’ button, the ‘right’ button will either execute the function or show the currently selected item’s submenu.

Storage of the menu system and its corresponding submenus and functions were stored in two structures: menuItems and menu.

```
typedef struct menuItems {
    const char *title;
    struct menu *subMenu;
    pFunc command;
} menuItems;

typedef struct menu {
    const char *title;
    int numItems;
    menuItems *pMenuItems;
    struct menu *prevMenu;
} menu;
```

The menu structure holds the menu’s title, the number of items in its submenu, a pointer to its submenu, and a pointer to its previous menu. menuItem, on the other hand, contains the title of the menu item, a pointer to its submenu, and a function pointer to a corresponding function.

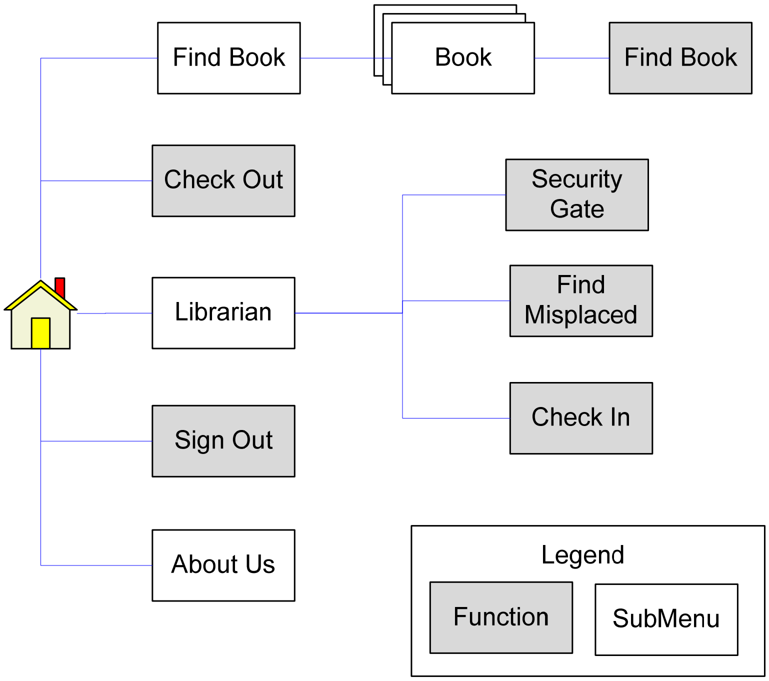


Figure 10 - Handheld Menu System

The hierarchy of the handheld menu system is described in Figure 10. The main menu contains 5 options: find book, check out, librarian, sign out, and about us.

The find book contains a sub menu that, when pressed, will display a list of books that user has requested to find. Upon selecting a book from the list, the user will initiate the find book function which will guide the user to the book.

The check out function will scan a book and send a check out request to the back-end database.

The librarian menu is only accessible if librarian credentials have been downloaded to the handheld unit. Credential downloads automatically occur when a user has logged into the web interface. If the user has been properly authenticated, the user will be prompted with three menu options: security gate, find misplaced, and check in. The security gate option turns on security gate mode where the handheld is constantly scanning books. If a scanned book was not checked out, an alarm will sound off. The find misplaced option will allow a librarian to find misplaced book and the check in option will scan a book and send a check in request to the bank-end database.

When a user has completed their tasks on the handheld, the user can erase all user data from the handheld by selecting the sign out option.

The about us option lists the authors of the system.

4.2.3 Battery/Power Supply

The power supply consists of a 9 volt, 175 mAh Rechargeable Nickel Metal Hydride (NiMH) Energizer battery and a recharger circuit for the battery. NiMH batteries are sensitive to over-charging. In order to recharge the batteries, fast charging method must be used to recharge the battery in conjunction with a “smart charger”. The smart charger will prevent overcharging of the batteries and prevent battery damage (Wikipedia). The most common way to prevent battery damage from happening is by discharging the battery to an almost empty state (which is determined by the battery voltage) and then recharging it using a timer. This prevents the NiMH battery from over-charging or under-charging.

We used the Maxim 712 chip to build the smart recharger circuit. The circuit requires an external PNP transistor, a blocking diode, three resistors and three capacitors to meet its minimum operation requirement. The circuit diagram for our recharger is shown in Figure 11.

The Max 712 chip fast-charges the Nickel Metal Hydride battery from a DC source that must be least 1.5V higher than the maximum battery voltage. The chip can be programmed to charge up to 11 cells (at 1.4V per cell) by using the simple mode of the charger (Maxim).

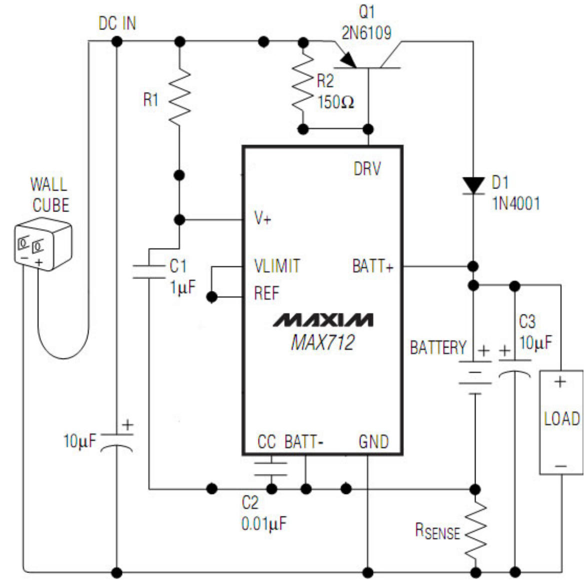


Figure 11 - Operating Circuit for Maxim 712 (Maxim)

12V DC from a wall cube as the input. The value for R1 is calculated using the formula:

R1 = Minimum wall cube voltage – 5V / 7mA = 1 kOhm

R2 = 150Ω (according to Maxim)

The values for the external capacitors are C1 = 1µF, C2 = 0.01µF and C3 = 10µF and are all standard values according to the datasheet [7]. RSENSE is used to calculate the fast-charge current (IFAST).

IFAST = (capacity of battery in mAh)/(charge time in hours)

Since, the capacity of our battery (C) is 175 mA, we wanted to recharge our battery at rates between 2C (350mA) and 4C (700mA) which would take 30mins to 15mns to recharge a drained battery. Charging at 2C requires a RSENSE of 0.83 Ohms and charging at 4C requires a RSENSE of 0.417 Ohms. We selected

RSENSE to be 0.56 Ohms as it was readily available at the parts store to give us a charging rate of 3.33C (583mA).

IFAST = 0.25 V / RSENSE = 0.25V/ 0.56 = 582.75 mA

4.2.3.1 Fast-Charge Mode

The Max 712 enters the fast-charge state when the battery is installed and there is some battery current detection (Ground voltage is less than Battery voltage). Also, the cell voltage should be higher than the undervoltage lockout (UVLO) which is approximately 0.4V per cell.

The charger detects the status of the battery by monitoring its voltage level. As the charge in the battery increases, the voltage increases linearly. When the battery is charged to its maximum, the voltage reaches a saturation level and remains steady at one value for a period of time and then steadily starts decreasing, as the battery starts discharging again.

The following graph gives an idea of the NiMH cells being charged/discharged using a Max 712 chip.

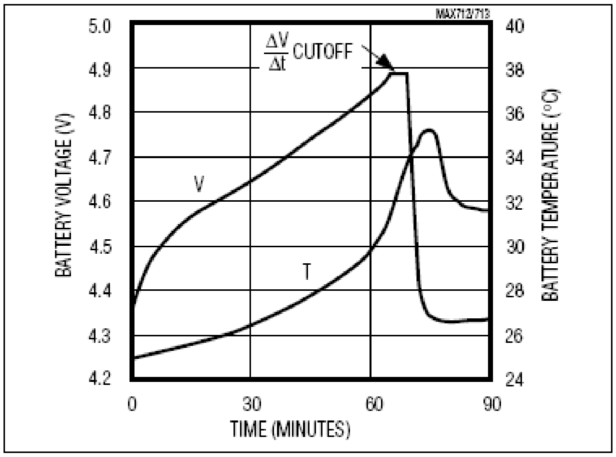


Figure 12 - NiMH Cells charged with Max 712 (Maxim)

The above graph shows the relationship between voltage (V) and time (minutes) for 3 NiMH cells. Our battery has 6 cells and follows the same trend except that it would take longer to charge our battery.

4.2.4 Antenna

The S4100 module requires an antenna that has an impedance of 50Ω at 13.56MHz. We decided to build the antenna rather than buy an antenna for budget purposes.

The theory behind the operation of the antenna relies on Faraday's law to induce a time varying voltage on the tag's antenna caused by the time varying magnetic field produced by the reader's antenna. This varying magnetic field in the reader antenna is caused due to varying current that is passed through the antenna (Ampere's law). A simple diagram displaying the above mechanism is shown Figure 13 - How a Reader Communicates with a RFID Tag.

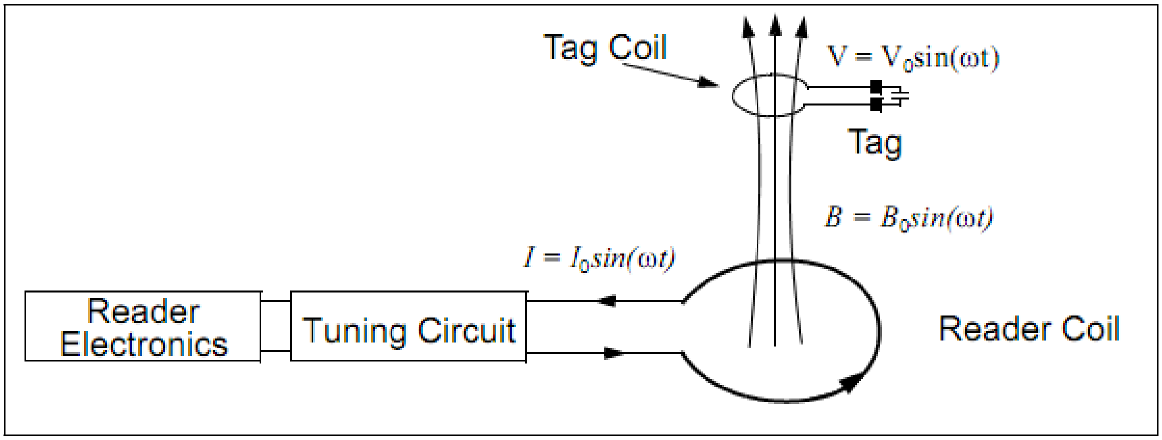


Figure 13 - How a Reader Communicates with a RFID Tag

For making the antenna, we decided to follow the HF Antenna Cookbook (Texas Instruments, 2004). This document contains information on how to design antennas for various readers manufactured by TI on a Printed Circuit Board (PCB). Since our antenna is attached to the handheld device, we decided to use a small antenna that would provide sufficient reading range. From the various designs in the cookbook we decided to go with the 50mm X 30mm antenna. The antenna requires a small impedance matching circuit attached to the antenna. The layout provided by the cookbook includes a place for SMA connector, as shown in Figure 14.

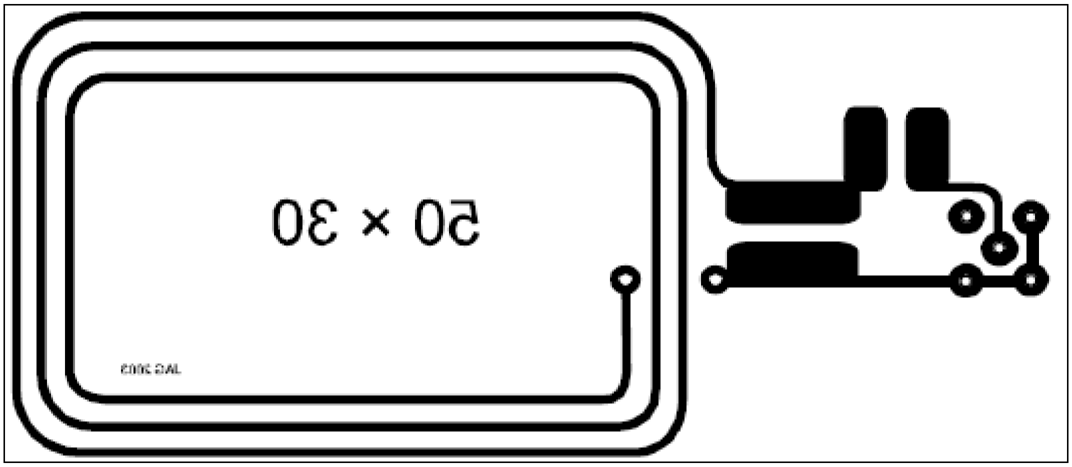


Figure 14 - PCB Layout for Antenna (Texas Instruments, 2004)

The impedance matching circuit is a parallel RC circuit with a series capacitor as shown in Figure 15. The Antenna cookbook suggested a parallel 10kΩ resistor with a parallel capacitor that had a value of 120pF + 6-30pF to vary the resonance frequency and the series capacitor with 6-30 pF to match the impedance of 50Ω.

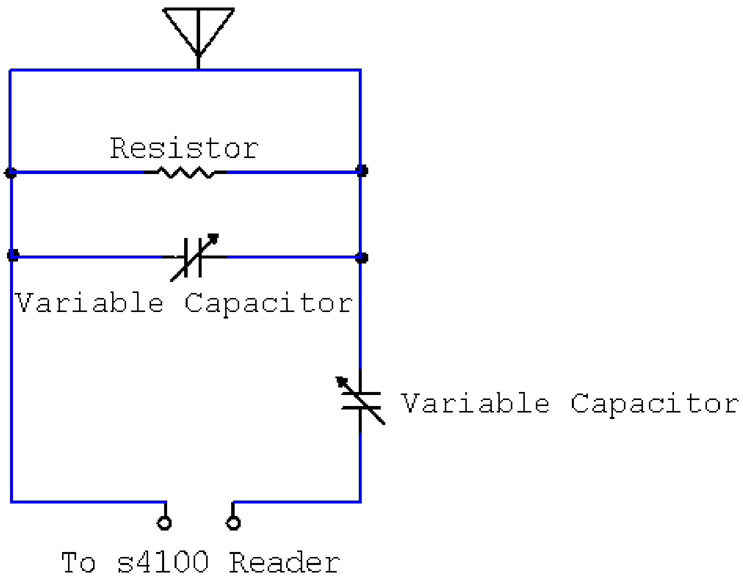


Figure 15 - Impedance Matching

When we tested this PCB antenna, we found that the antenna was tuned to the 3rd harmonic of 13.56 MHz i.e. 40.7 MHz and we were not able to make it change its resonance frequency to 13.56 MHz by varying the capacitance.

Based on the TA's advice, we decided to make antennas using magnetic wire instead of making the antenna on PCB. Our methodology for choosing the components for the antenna circuit was both theoretical calculations and trial and error. Initially, we made several antennas with arbitrary amount of loops. Most of them are rectangular in shape and were between 7-15cm in length and 4-10cm in width. We built our resonance circuit on breadboard with variable capacitors and resistors.

To find the values for the capacitance, we measured the inductance of the antenna (at 1KHz) using a RLC meter. Based on the inductance, appropriate capacitors were chosen.

The resonant frequency of an LC circuit is given by Figure 16 - Resonant Frequency of an LC Circuit:

$$f_o = \frac{1}{2\pi\sqrt{L_T C}}$$

Figure 16 - Resonant Frequency of an LC Circuit

Where f_o is the resonant frequency, L_T is the inductance and C is the capacitance.

Since the inductance is fixed for the antenna and the resonant frequency is fixed at 13.56 MHz, we can only vary the capacitance. This can be found using the formula found in Figure 17.

$$C = \frac{1}{L(2\pi f_o)^2}$$

Figure 17 - Capacitance for Resonant Frequency

Once we found the correct value for the resonant frequency, we can calculate the bandwidth of the RLC circuit by using the following Figure 18 - Bandwidth of RLC Circuit.

$$B = \frac{1}{2\pi RC}$$

Figure 18 - Bandwidth of RLC Circuit

We can also find the Quality factor Q by using the formula shown in Figure 19 - Quality Factor.

$$Q = R\sqrt{\frac{C}{L}}$$

Figure 19 - Quality Factor

To make the antenna more efficient, we would like to have the widest bandwidth while maintaining a certain Q. This proves to be a challenge because the Bandwidth is inversely related to the resistance and Q is directly proportional to the resistance.

For measuring the effectiveness of the antenna we used a setup similar to the one shown in Figure 20.

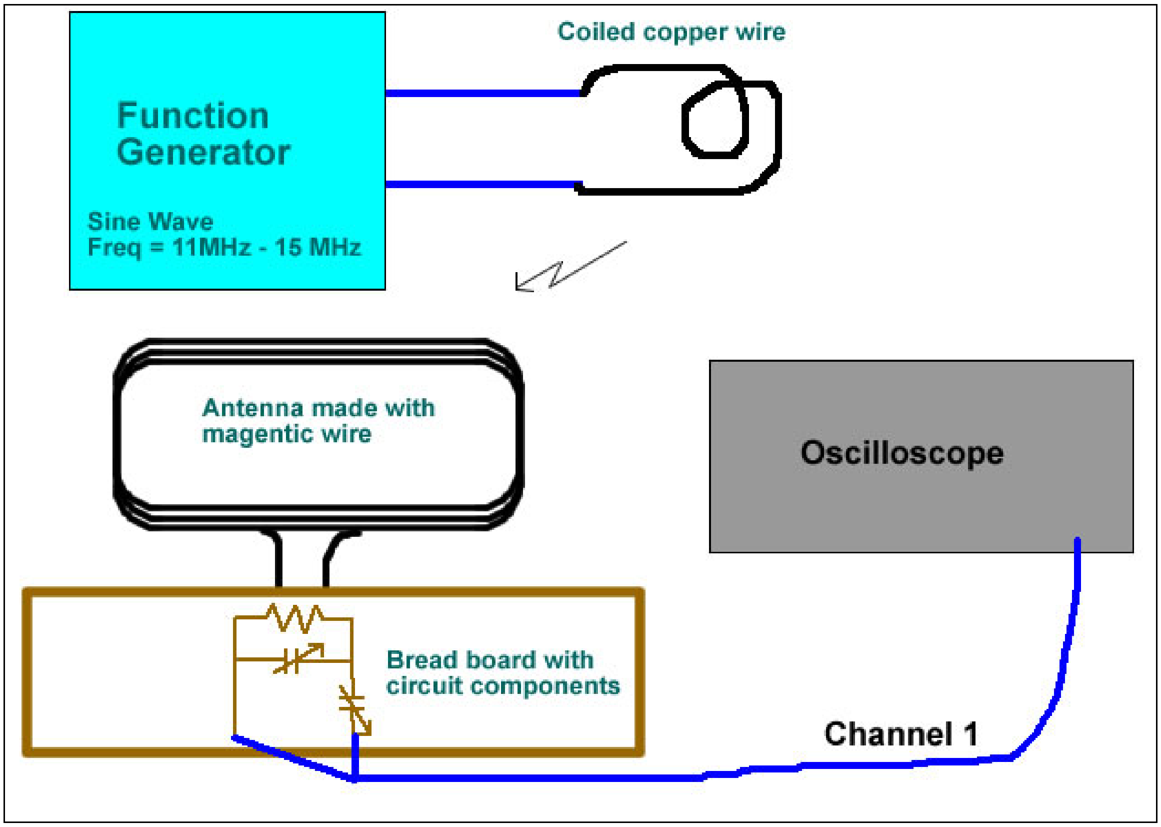


Figure 20 - Setup to find the resonance frequency of the antenna

The oscilloscope was connected to the circuit's terminals to measure the resonance of the antenna circuit. The function generator was connected with a coiled copper wire and was used to output a 20V peak to peak sine wave. The antenna attached to the circuit board would pickup the sine wave emitted by the function generator and it could be seen on the oscilloscope. When the circuit was at resonance for a particular frequency, we would observe the highest amplitude on the oscilloscope. And if the frequency was changed in either direction, the amplitude measured by the oscilloscope would decrease. We varied the capacitance of the circuit until we made the resonant frequency of the circuit to be 13.56 MHz.

Based on the TA's advice, we decided to make our final antenna with a thick 14 gauge magnetic wire with two loops. The impedance matching circuit was put on a prototype board along with a SMA connector. An 18" coaxial cable was used to connect to the SMA connector and the RFID reader module. The use of the coaxial cable would allow us to place the antenna away from other circuit components and make the antenna less susceptible to interference caused by them.

Currently, our antenna looks like the following.

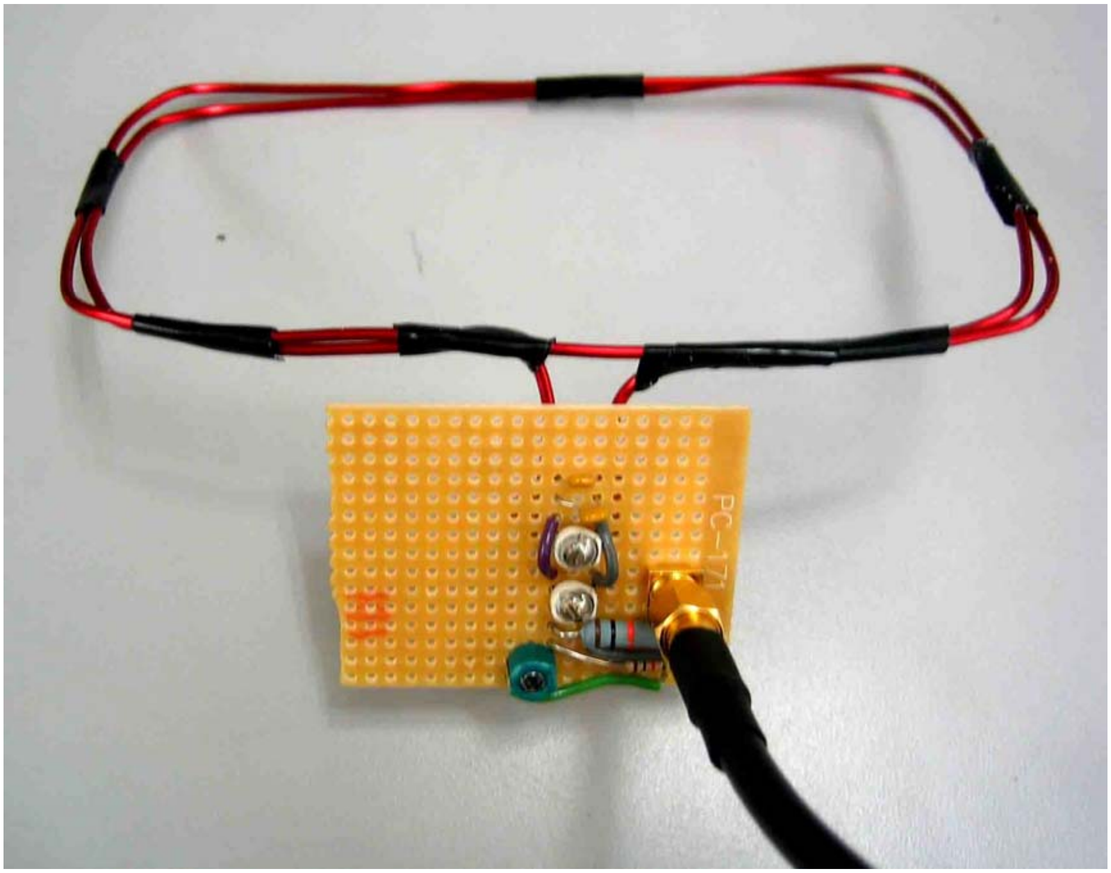


Figure 21 - Final Antenna for the RFID Module

The final antenna had an inductance of 1.8uH (measured at 1KHz) which allowed us to have a rough estimate of the capacitance required for making the circuit resonant at 13.56 MHz. We used the same setup as shown in Figure 22 to make the circuit resonant at 13.56 MHz. ISO 10373-6 was used to match the impedance of the circuit to 50 ohms. A setup similar to Figure 22 was used where the phase difference between the signal source and the load is monitored.

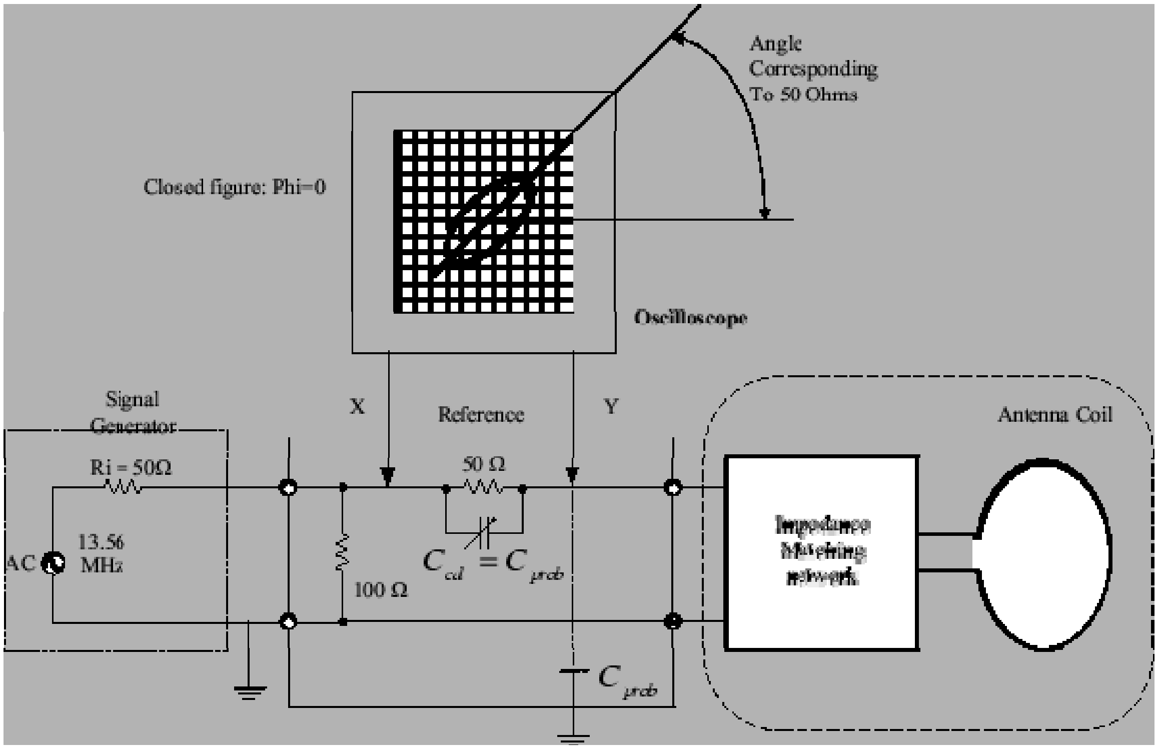


Figure 22 - Setup to match the impedance of the antenna circuit

In the first step, we calibrate our testing set formed by a 100 ohm resistor connected in parallel to the signal generator. A 50 ohm resistor and a variable capacitor are connected in series to the signal generator. The output of the circuit is connected to a 50 ohm resistor initially to simulate a 50 ohm load. We note the shape and angle of the Lissajous figure. The fatness represents the phase difference and the angle of the ellipse represents the impedance. Then we replace the 50 ohm calibration resistor and replace it with our actual circuit. Then we vary the series capacitance of the antenna circuit until we notice similar shaped ellipse with the same angle and phase difference.

Once we matched the impedance, we took our antenna circuit to the Radio Science lab of UBC to test our circuit. With the help of the TA, we used a Vector Network Analyzer and found that our antenna was very well tuned to 13.56 MHz and had impedance very close to 50 ohms. The final antenna was able to read up to 4 tags (that are parallel to the plane of the antenna) from up to 4”.

4.2.5 Printed Circuit Board (PCB)

The main PCB was designed to integrate the microcontroller, the Zigbee module, user input (push buttons). The final PCB measured 6.4cm x 11.5cm and the PCB design is shown in Figure 23 - Group 9 PCB Circuit.

The PCB design for Zigbee module was retrieved from Xbee Reference (Systems, 2008) and integrated into our main PCB design.

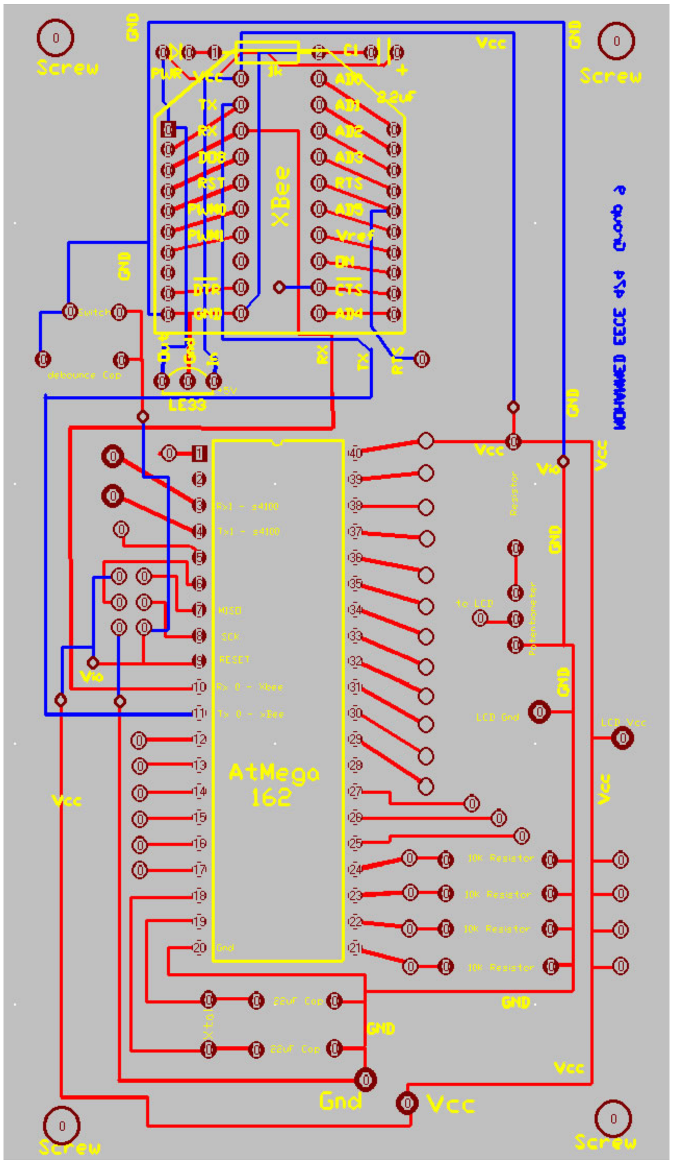


Figure 23 - Group 9 PCB Circuit

4.2.6 Enclosure

The plastic enclosure houses all our PCBs, the reader module, the antenna circuit, the recharger circuit and the LCD screen. We also needed four push buttons, two status indicator LEDs (on/off and charger status), a primary switch to switch the hand held on and off and female AC-DC wall charger connector to connect to the recharger circuit inside. The enclosure measures 18.5 cm X 11 cm X 4 cm from the inside is more than adequate to fit all our handheld components in.

4.3 Back-End Server

The back-end server is comprised of an Apache web server, a MySQL database, and a Java Interface to communicate with the handheld unit.

4.3.1 Database

The MySQL database was designed to store significant library information, as well as to store interim data when the user is operating in specific modes capable by the system. Seven tables were used to store this information, as shown in Figure 24 below.

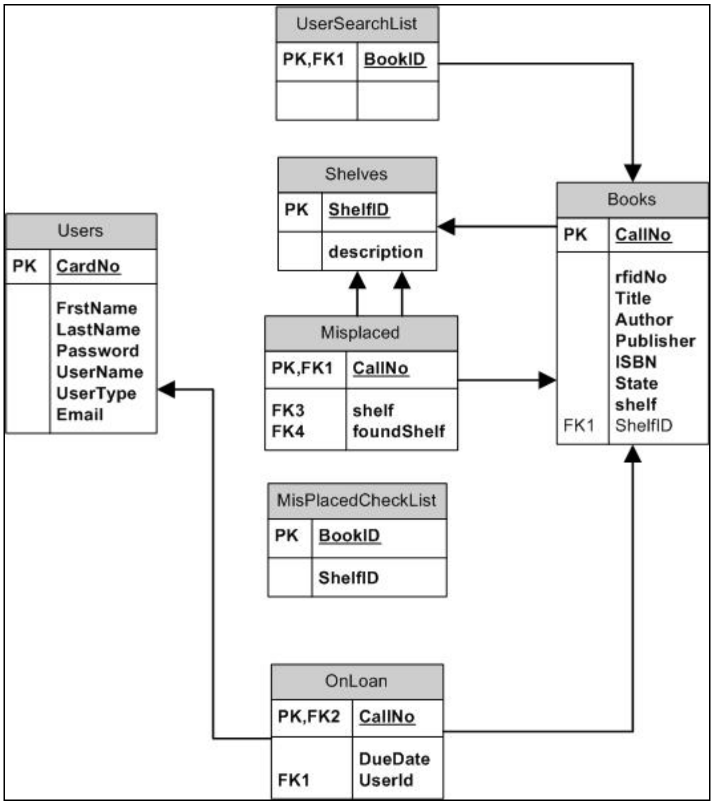


Figure 24 - Database Design

The Books table consists of the necessary information for storing books and other library items in the database. Columns included in this table are for the item's Call Number, RFID Number, Title, Author, Publisher, ISBN Number, and State. In this design, the Call Number and RFID Numbers are unique entities. Also, we chose to the store the RFID number as a long integer, for simplicity. In doing so the java back end converts the 8 byte RFID input into a long integer and stores in this table. This conversion will be described in more detail in section 4.3 of this report. Note that because RFID numbers are stored as a long integer, they are sometimes stored as a negative number, if the most significant bit of the 8 byte number is set. In Java and MySQL, there is no such variable that is unsigned. Thus, if the most significant bit is set, the number will always be stored as a negative number.

A user table is used to keep track of user information such as name, password, username, authorization type, e-mail address and library card number. These credentials can be edited from the User Interface. This table mostly provides the library system a name and information to associate a name and information with a book that has been checked out of the library. It also allows the capability of having e-mail overdue and due date notifications. Although that has not been implemented, there is the possibility of implementation in the future because it has been included in the database design.

A shelf table is used to store information about the shelves in the library. The table stores the shelf's given RFID number, as a well as a 12 character description of where in the library the shelf is located. This description is used on the handheld device, when a user is locating books that he/she has downloaded as part of his/her search list. It is also used as the destination of a foreign key for the shelf credential in the books table.

Two tables are used to handle misplaced books. The MisplacedCheckList table is a temporary storage table while a librarian is performing inventory of the library. As the librarian scans books in inventory mode, the RFID number of the book scanned and the shelf it is currently on is entered in this table temporarily. Once the librarian has completed the inventory, a comparison with the books in this table, and the book's true shelf location are compared. If the book is on the wrong shelf, the information for that book is then moved to the misplaced table, where it is stored there with information as to where the book belongs, and where the book is currently. At this point, all the entries from the MisplacedCheckList table are removed, and only truly misplaced books remain in the Misplaced table, where they will remain until the librarian marks them as found.

Lastly, a UserSearchList table is used as temporary storage of items the user wishes to search for when he/she is logged in. As the user selects books to search for, they are stored in rows in this table, until the user logs out of the system. At any point when the user wishes to physically search for these items, the java backend uses this information to tabulate a search list, and send to the handheld device.

As noted in Figure 24 above, most of the tables have foreign keys to other tables in the system. These foreign keys are used to abstract information from tables in order to make them simpler. For example, in the books table, the shelf entity is a foreign key to a shelf in the shelf table. This way, the books table only needs to store the RFID number of the shelf, and can retrieve the shelf description through the foreign key.

4.3.2 Web Server

The Apache web server serves as the first point of contact for the user. The entirety of the web user interface is coded using a mixture of PHP, hypertext markup language (HTML), and cascading style sheets (CSS). The web interface can be broken into several components: user management, catalogue, and librarian tasks. Shown in Figure 25 is a web site map detailing the various pages a user is able to access.

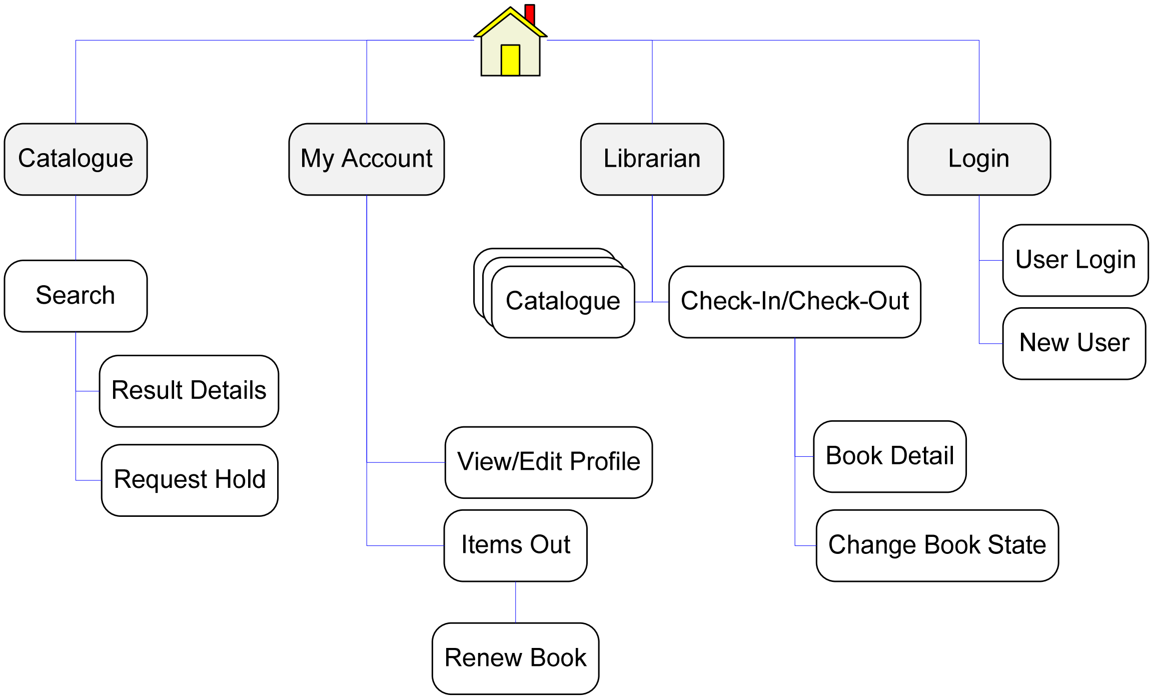
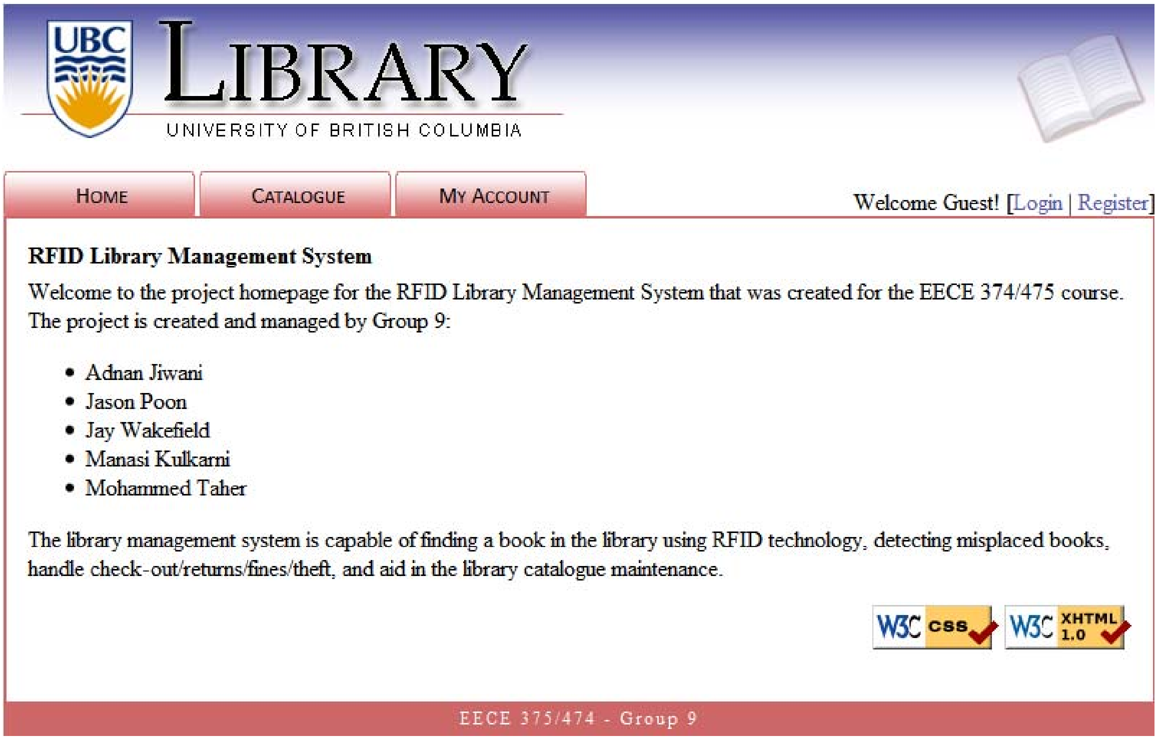


Figure 25 - Library web interface map



RFID Library Management System

Welcome to the project homepage for the RFID Library Management System that was created for the EECE 374/475 course. The project is created and managed by Group 9:

- Adnan Jiwani
- Jason Poon
- Jay Wakefield
- Manasi Kulkarni
- Mohammed Taher

The library management system is capable of finding a book in the library using RFID technology, detecting misplaced books, handle check-out/returns/fines/theft, and aid in the library catalogue maintenance.

Figure 26 - Web Interface Starting Page

4.3.2.1 User Management

In the upper right corner of the interface, the user has the ability to either login to their account or register for a new account. When the user has completed the login process, a cookie will be created and until the cookie's expire, it will allow the user to skip the login procedure on subsequent visits to the website.

Once a user has properly logged in, the user will have the ability to view and edit their user information as well as view and renew any items that currently have on loan.

4.3.2.2 Catalogue

Searching through the catalogue is implemented using MySQL's built-in full-text search feature. Full-text searching utilizes indexes to examine all the words in every stored database entry to match the user query. As full-text searching is designed for large databases, running a full-text search on a small database containing a couple entries will lead to unfavorable results. However, as a library's catalogue contains many hundreds or thousands of books, this will not be an issue in a real-life scenario.

4.3.2.3 Librarian Tasks

Once a librarian logs into the website, he/she will gain access to additionally functionality including the ability to view and modify the catalogue.

The user interface allows the librarian to perform many administration tasks, such as adding, editing, and removing books in the catalogue. There are tabs for each of these tasks, as seen in Figure 27. There are also functionalities that allow a librarian to check out a book in a patron's name from the web interface rather than the handheld, and to view the entire library catalogue and all books that have been flagged as misplaced.

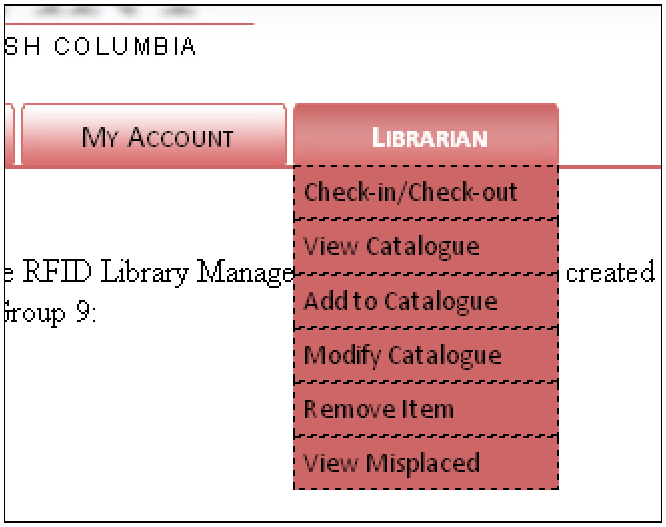
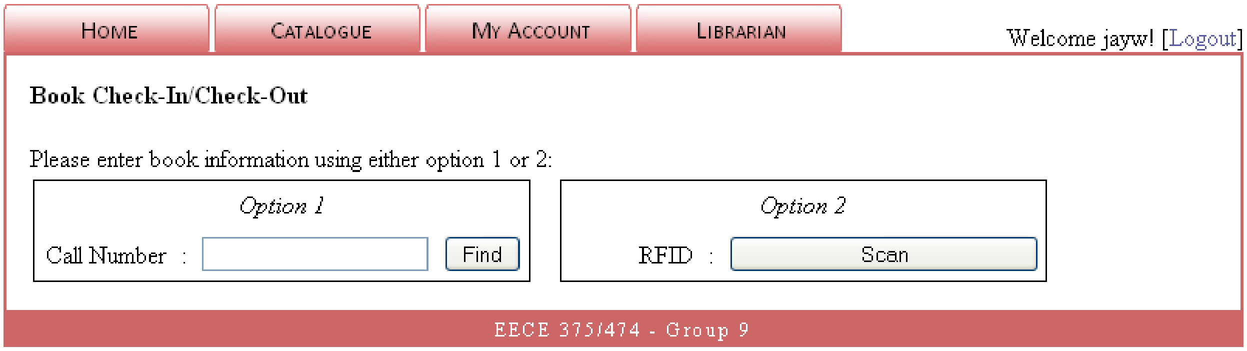


Figure 27 - User Interface Librarian Features

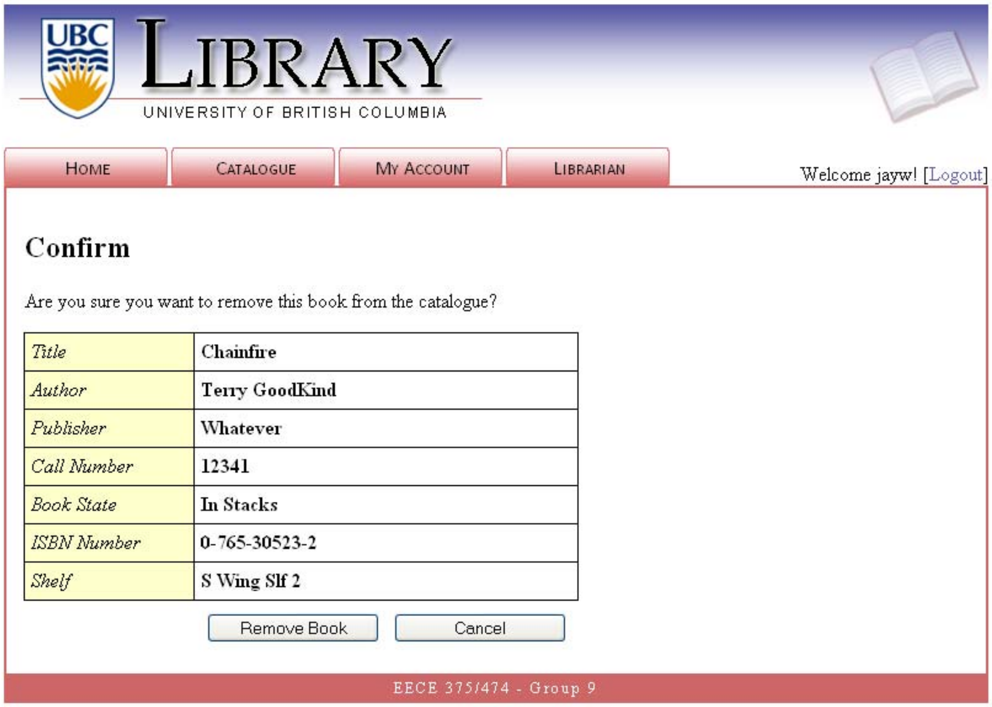
In each of the Check-in/Check-out, Add to Catalogue, Modify Catalogue, and Remove Item options, the librarian has two options of selecting a book to operate on. They may enter a books' call number, or may scan the book with the handheld device. In either situation, the librarian is forwarded to a screen where they see the book's details and may modify or confirm their actions. These examples can be seen below:



Book Check-In/Check-Out

Please enter book information using either option 1 or 2:

Figure 28 - Two Book Modification Options



Confirm

Are you sure you want to remove this book from the catalogue?

Title	Chainfire
Author	Terry GoodKind
Publisher	Whatever
Call Number	12341
Book State	In Stacks
ISBN Number	0-765-30523-2
Shelf	S Wing Slf 2

Figure 29 - Librarian confirmation of actions

The librarian is also able view the entire catalogue, and to view all books that have been marked as misplaced. In each of these pages, the librarian is presented with a list of details of each book in the table, as well as links to perform actions on the books. While viewing all items in the library, the librarian may click on the "Modify" link, where they will be forwarded to page in which they are able to modify item information. As well, in the misplaced books list, they are able to mark the book as found, and remove is from the misplaced table in the database, as discussed in section 4.2.1.

4.4 Java Interface

As the web interface and the handheld device are often in communication, and as the handheld device is constantly communicating with the database, an interface was required to deal with requests from both the web interface and the handheld device. In order to accomplish this, the design for the interface had to be able to handle controlling the XBee Driver through the serial port of the base station. After a failed trial and error with several PHP plug-ins for the serial port, a java library was discovered that made communicating with the serial port fairly straight forward. Thus, java was chosen as the primary backend interface language for several reasons:

- Provided the easiest way to send and receive information from the serial port
- Provided multithreading capabilities, that allow both reading and writing from the serial port at what seems like the same time
- Provided networking sockets for communication between several threads running at any given time

Sun Microsystems has supported and developed an API for communicating with the serial port. This API was commonly known as sun's javax.comm.* API. However, this API has been discontinued for windows operating systems. On further research it was determined that Sun still allows download of this API for windows, however it does not provide the implementation classes needed for use. There is an open source group named RXTX, that provide this implementation. Once Sun's API and RTXT's implementation has been installed into the base stations Java Runtime Environment (JRE), it became very easy and reliable to communicate with the serial port of the computer, and send information to the XBee card. This was the primary reason for choosing java as the backend language, as well as the other reasons listed above. Information on RXTX can be found on their website at http://users.frii.com/jarvi/rxtx/.

In order to work as a backend communication server, the java interface requires multithreading. It is expected that the handheld device may send data to the server's XBee device at any time. In addition, the database or web interface may need to send data to the handheld device at any time. In order to accomplish this, three main threads are started and are run constantly to allow communication with the XBee device connected to the serial port. These three main threads are a writing thread, a reading thread, and a thread that handles writing from the web interface. These threads handle the various forms of communication that is needed with the XBee device. The extra thread for writing from the web interface is needed, because it is a special case of writing. The main write thread writes data from the system input stream. The web interface does not write to this input stream, but rather sends data from it's own new java thread. Therefore, the extra thread is needed to handle writing information that does not come from the system input stream.

Any communication that does not require the web interface is handled all within the already running, three main threads. These operations include checking in and out books from the library via the handheld device, adding misplaced books to the misplaced book check list, and the security gate checking. These simply communicate with the database, and do not need to communicate with other running threads on the system. The protocol would simply be received, and the appropriate database operation would be run using the current thread, and the JDBC (Java Database Connector). There is no need for any other threads to be run, or communicated for such a process. However, there are some operations that are either started by or need to communicate with the web interface. This situation is slightly more complicated, and requires net communication via java sockets.

As part of PHP, there is a command that allows outside programs to be run outside the script, as if they were started on the command line of the underlying system. This command is “shell_exec (String command).” This command takes a string as its parameter that is the exact command that you would execute from the shell of the operating system, and it returns a string containing all of the output from the program that is run. We utilize this command on our system to send and receive information from the handheld device to or from the web interface. Whenever this occurs, a new java program is started from the PHP web interface that deals with the exact request from the user on the web page. It then sends the information out through the serial port. However, because this new program is a new thread on the Java Virtual Machine (JVM), and the thread that communicates with the XBee device is also its own thread in the JVM, the program launched from the web interface needs some way to send information to the communication thread, and out to the XBee device. This is accomplished by using java sockets. We have written a server that deals with communications between threads, and runs as its own thread in the background. This server passes information from thread to thread and allows communication from the web interface to the serial port communication thread. Whenever a PHP-java thread is started, it sets up a socket with the server. At the same time, the receiving end either starts, or has already started its own socket with the server to send and receive any information that involves the web interface. Doing this allows the communication between the Java backend serial port/Xbee communication thread, and any threads that are started and ended via the web interface, thus allowing communication between the web interface and the handheld device.

Apart from managing different operations of functionality, the java interface is also responsible for parsing an 8 byte RFID number into a long integer for storage on the database. Because there is no such variable as an unsigned byte in Java, integers are used to store the 8 byte RFID number revieved from the handheld device to allow the range of the byte, by mimicking an unsigned byte. The number is then parsed into an integer using the following code:

```
public static long bytesToLong( int[] data ){

            long RFIDnum = 0;

            for( int i = 0; i < data.length; i++ ){
                  RFIDnum = RFIDnum << 8; //shift left by two HEX digits
                  RFIDnum += data[i];
            }

            return RFIDnum;
      }
```

The reverse operation parsing can also be achieved by parsing the integer into an array of integers in much the same fashion. Note that because there are no unsigned variables in Java or in MySQL, this algorithm results in a negative long integer if the most significant bit of the 8 byte RFID packet is set.

Information is passed and received between the handheld and java interface using a set protocol. This protocol provides information to the java interface about what operation to perform, and the data needed for that operation. For details on this protocol, please see appendix B.

The challenges with this approach were mostly allowing communication between backend processes that could be started and stopped at any time from the web interface. However once we experimented with the network sockets, it was confirmed that java was sufficient for allowing a backend constantly running process that handles communication between the XBee and handheld devices, the MySQL server, and the web interface.

4.5 Communication Protocol

A communication protocol between the back-end server and the handheld unit was devised in order to effectively communicate between the two entities.

The specifics of the protocol are defined in

Part	Part Number	Cost per Unit	Quantity Purchased	Part Total
Xbee 802.15.4 OEM RF Modules	XB24-ACI-001	$26.53	2	$53.06
Xbee Starter Kit	XBP24-DKS	$179.00	1	$179.00
Atmega 162 Microcontroller	ATMEGA162-16PU-ND	$8.40	2	$16.80
RFID Transponder In-Lay	Free Trial	$1.79	5	$8.95
RFID Transponder In-Lay	481-1067-1-ND	$1.79	5	$8.95
RFID Reader	481-1052-ND	$104.31	1	$104.31
Liquid Crystal Display	LCM-S01604DSR	$25.19	1	$25.19
Voltage Regulator		$0.35	1	$0.35
ISP (Programmer)	ATAVRISP2-ND	$38.89	1	$38.89
Tantulum Capacitor (LE33)	497-4258-1-ND	$1.07	2	$2.14
		$1.11		$0.00
IC Batt Fast Charger	MAX712CPE+-ND	$9.98	1	$9.98
Energizer Rechargeable battery	N705-ND	$15.00	1	$15.00
PNP Transistor	2N6109GOS-ND	$1.55	2	$3.10
Rectifier	1N4001DICT-ND	$0.39	2	$0.78
1.0UF 50V Mini Alum Elect	P824-ND	$0.18	2	$0.36
0.01UF Capacitor	399-4150-ND	$0.21	2	$0.42
Magnetic antenna wire	Active electronics	$12.31	1	$12.31
Header pins	MRO Electronics	$10.08	1	$10.08
SMA Connector	MRO Electronics	$9.59	1	$9.59
Power Adapter	MRO Electronics	$18.03	1	$18.03
Miscellaneous				$8.00
Shipping and Handling		$8.00	4	$32.00
Sub-Total*				**$378.29**
Taxes (GST)		**5.00%**		**$18.91**
Total*				**$397.20**
Total with Xbee starter kit				$557.29

* As the Xbee Starter Kit will not be reimbursed, the value can be removed from the total

Appendix B – Communication Protocol.

4.6 Security Gate

As mentioned in the requirements, the main aim of the security gate is to detect and sound a buzzer if someone is trying to leave with a book that is not checked out. The buzzer that we had available was an AC buzzer. To convert the 5V DC voltage provided by the microcontroller when the pin is set to high to an AC wave we used an LM555 timer/oscillator. The LM555 circuit we used is similar to the one provided in the datasheet for a stable operation with one exception, the reset pin is not attached to anything as reset is not required in our case. The circuit diagram we used and the pin description of LM555 are provided in Figure 31 and Figure 32 respectively. Since we needed 500Hz to tune the buzzer to the correct volume and had many 10K resistors, we calculated the capacitance using the formula provided in the datasheet:

$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2\,R_B)\,C}$$

Figure 30 – Frequency Formula LM555

This gave us 0.096µF for the capacitance. Not having any 0.096µF capacitors we decided to use 0.1µF capacitor instead, giving us a frequency of 480Hz.

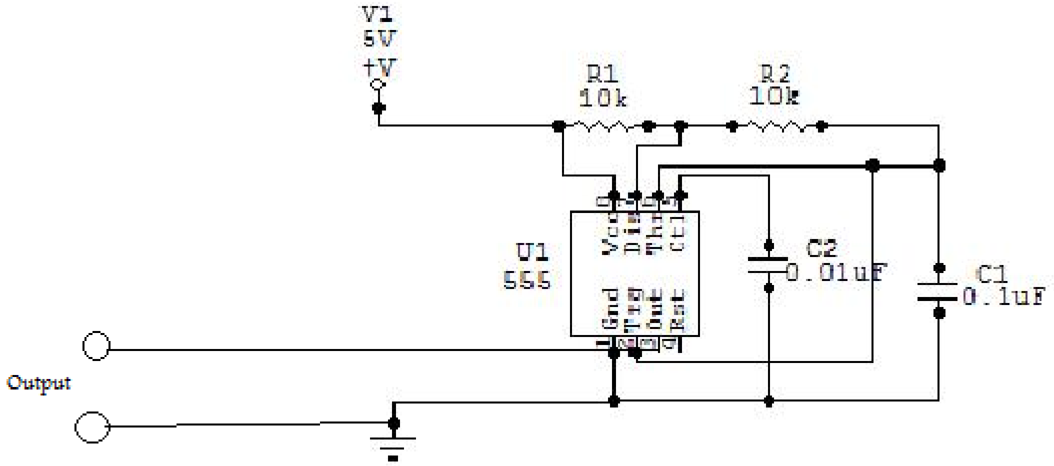


Figure 31 - LM555 Circuit Diagram

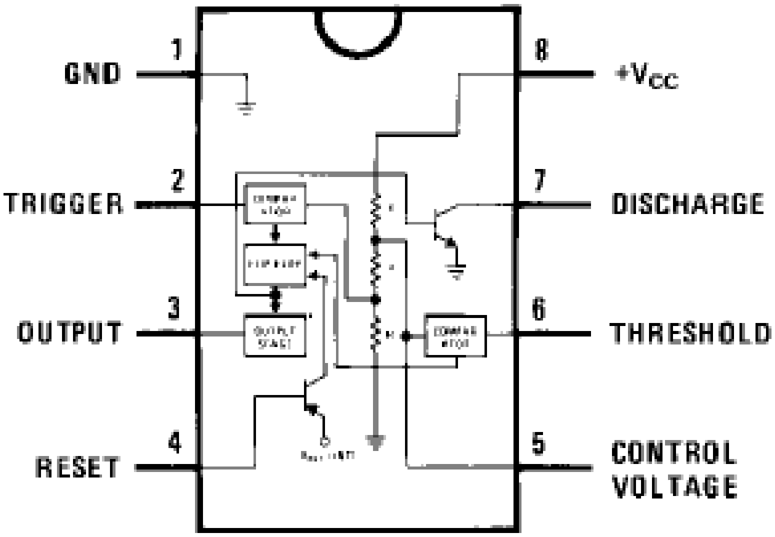


Figure 32 - LM555 pin diagram

On the software side the security gate program continuously scans for a tag. As soon as a tag is detected it sends the security packet to the database and receives a response packet (Appendix B – Communication Protocol) from it. It than turns the buzzer on if the response suggests the book was not checked out.

4.7 RFID Tags

Since the S4100 module gives a good range with ISO 15693 compliant rectangular inlay we decided to get RI-I03-112A-03 RFID transponder Inlay. Apart from being compliant with the module, it is also writable with a user programmable memory of 2K and is appropriate in size and thickness to use in books, as shown in Figure 33 below.

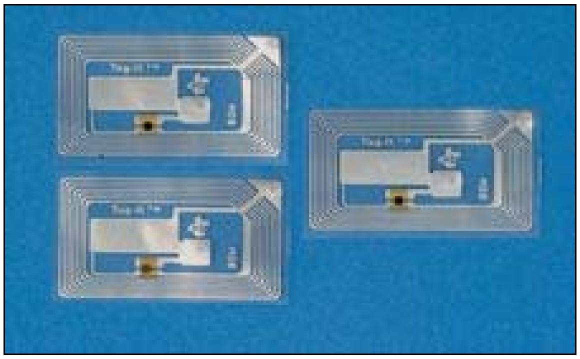


Figure 33 - RI-I03-112A-03 HF-I Plus Inlay

5.0 Testing Results

While developing each module of the RFID library management system, each module was individually tested to ensure that it was functioning properly. Once each module was functioning properly, the different modules were linked together and tested as a whole. This section provides a description on the type of testing that was performed and the results of those tests.

5.1 Software

Each software module was individually tested. Each component was then integrated and black-box testing was performed on the software as a whole. An example of one black-box testing that was performed was:

Test Case: Find Book

1. User logs into the web interface downloading their username and user authentication level to the handheld

2. The user choose books from the catalogue to find in the library

3. The list of books is downloaded to the handheld unit

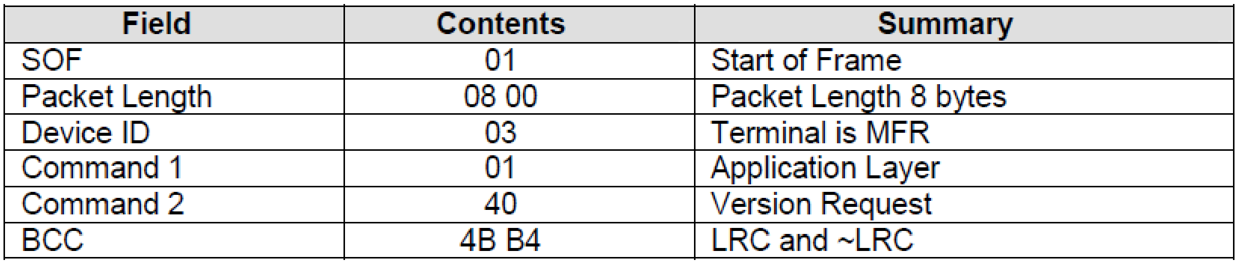
4. The user selects each book and begins to scan the shelf to find the book

Similar test cases were performed for all the possible handheld and back-end server instructions. The following sections outline testing that was performed on the handheld unit and back-end server.

5.1.1 Handheld Unit

As AVRStudio comes complete with a debugger, the debugger was initially used to ensure that the correct values were being pushed onto the pins. Once that was in place, the LCD driver was the first piece of microcontroller software written such that all following software could use the LCD to print debug statements.

In order to test the RFID driver, as no antenna was available until the latter parts of development, we opted to use the version request packet for the S4100 to simulate the RFID tag number in place of the tag request (Figure 34).



Field	Contents	Summary
SOF	01	Start of Frame
Packet Length	08 00	Packet Length 8 bytes
Device ID	03	Terminal is MFR
Command 1	01	Application Layer
Command 2	40	Version Request
BCC	4B B4	LRC and ~LRC

Figure 34 - S4100 Version Request Packet

When sending the version request packet, the S4100 module will respond with a packet that contains the version number of all the various terminal firmware on the S4100. These version numbers were used to simulate the RFID tag ids. However, as the version request will always return the same response packet (i.e. same simulated RFID tag), we were limited in the functions we could test. By using the version request we were able to test scan tag, check in and out, and book list building. However, we were unable to test find book or find misplaced books due to the reason mentioned above. Although this was not a form of thorough testing it gave us a sense of how our program was interfacing with the Java interface and underlying database. This method of testing provided satisfactory results and alerted us to a few major issues in our implementation. One such issue was related to the receive buffer for Zigbee where the overflow of the buffer was being handled incorrectly.

5.1.2 Back-End Server

Testing for the web interface portion of the back-end server is straight-forward. As the web interface was built using scripting languages, PHP and HTML, the changes to the source file could directly be seen by opening the page in a web browser. If the results, were not as planned, then additional changes to the source file were made to correct the problems.

The java interface was tested initially by itself without integration with the microcontroller. This was achieved because it only interacts with the serial port, and the XBee device on the end of it. Therefore, the destination of the wireless information does not need to be the microcontroller, but can be a pseudo replacement. In this case, we used another computer to simulate packets being sent and received to the java interface, using that computers serial port and the other XBee device. Doing this, we were able to test and debug all of the functionality of the java interface before we performed integration testing with the microcontroller. This allowed us to have confidence that any bugs we found during integration testing were isolated to the handheld device, since the java interface operates independently, and was testing and declared functional the day before. This saved us time and frustration when performing integration testing, since we could isolate any bugs to the handheld device.

5.2 Hardware

5.2.1 RFID Reader Module and antenna

The RFID reader module was tested initially with a makeshift antenna. The makeshift antenna was made with 5-6 turns of copper wire connected to the J8 port of the reader. Although the makeshift antenna was not tuned or impedance matched, it allowed us to test the RFID reader module. With the makeshift antenna, we were able to read single tags up to 2 cm away.

When we had the final antenna made, we tested the antenna using a Vector Network Analyzer at the Radio Science Lab. The test showed that our antenna was tuned close to 13.56 Mhz with 50 ohm impedance. The test also showed that the bandwidth for the antenna was very narrow. This has been discussed in section 6.4 in Assesment and Analysis.

5.2.2 Microcontroller

The microcontroller was initially tested by running a small blinking light program. Using this simple program, we were able to verify that all the ports were operational. Later, the microcontroller was tested by using push buttons and LED response.

5.2.3 Power Supply / Battery Recharger

The smart battery charger for the NiMh charger was tested initially with the battery that was ordered from DigiKey. Since the battery that was bought was fully charged, we discharge it slightly by connecting the battery terminals to resistors (not exceeding 0.25W limit for the resistors). A slightly discharged battery was not able to trigger the fast charge mode of the battery charger but we were able to detect trickle charge current.

5.3 Integration

Once both hardware and software were successfully tested, we put everything together to test our entire library system. We attached RFID tags on various books and shelves to create a sort of mock library. Since both hardware and software were tested individually before, this part went pretty smoothly. It however showed us that our antenna is somewhat inconsistent, as it sometimes detects all tags in its range and sometime misses a tag. After running through the complete testing we calculated the success rate for our antenna to be 60% for detecting all tags. With this success rate we concluded that we are better of using the implementation of detecting one tag at a time only as it removes ambiguity and has a success rate of close to 85%.

6.0 Assessment and Analysis

6.1 Handheld Unit

The final firmware for the microcontroller filled 7896 bytes (48.2%) of programming memory and 412 bytes (40.2%) of data space. Although 412 bytes is on the high side, there is sufficient space to allow the download of ten books.

The completed handheld unit draws a total of 82mA.

6.2 Backend Server

6.2.1 Web Interface

The web interface provides a clear and easy way for the library user to perform important library operations. It allows the user to register and manage his profile, and search for library books easily without confusion. It also performs very quickly, while interfacing with either the MySQL database and the java interface which communicated with the handheld device. Because of these reasons, the web interface is an excellent prototype for the library management system. If more time was available, we would have included more functionality to the system, including on hold functionality, and modifying the states of the books to more realistic states, such as missing or damaged. However, given that this library system is s prototype, it performs as well as we expected it to when we were designing it.

6.2.1 Java Interface

The java interface does a better job than expected with handling information from the Xbee device and the PHP web interface. Originally, there were some concerns that with as many threads running that were needed; the java language and environment would be too slow. However, the communication happens without any apparent delay. Also, the performance when communicating with the serial port is very impressive when comparing it to every other language implementation of serial port communication. To this point, we have not experienced an instance in which the java interface has lost information, or did not receive a packet from the handheld device, apart from known bugs when doing implementation testing.

At this point, the java interface is run on the database server as many standalone java applications. Realistically, however, the java interface would need to communicate with a database that was not necessarily on the same computer the application was running on. We did not address this issue in our project, for reasons including time, knowledge base, and that it was not needed to accomplish this prototype. However, if this were used in a real library, the java interface would need to be migrated to a web environment, and run on a servlet such as Apache Tomcat, and have it communicate with databases on remote servers. Concerns involving this would be network speed when communicating with the database, and also the migration itself. However, for the scope of this project, these are not a concern.

6.3 PCB

Our final PCB design failed to capture several smaller circuit parts and is described below.

Battery recharger circuit – We were not able to integrate the battery recharger circuit due to limitation of time and not having finalized the enclosure (space requirements). However, the battery charger is small enough that it was put on a separate prototype board. Having the recharger circuit on a prototype board allowed us to move the smaller circuit any corner of the enclosure and be placed above an existing PCB. This also allowed us to place the female adapter of the AC-DC converter at an appropriate corner of the enclosure where it is convenient for the library user to switch on/off the handheld.

Voltage divider circuit – A voltage divider circuit was required for the microcontroller to communicate with the Zigbee module via UART. The microcontroller works with a 5V power supply and it sends out a 5V signal via the UART port to the Zigbee module. The Zigbee module is unable to interpret the 5V signal it receives because it works on a 3.3V logic level. After consultation with the TA, we decided to use a simple voltage divider circuit between the UART ports of the Zigbee to bring the 5V logic down to 3.3V. The microcontroller faces no problem in receiving 3.3V logic from the Zigbee module as it interprets any signal level above 2.5V correctly.

Additionally, our circuit had an error in connecting the Tx and Rx (UART port) of the Zigbee to the microcontroller. The Tx and the Rx lines were interchanged and was fixed by drilling additional holes and using small segments of copper wires.

6.4 Antenna

The antenna that was built and tested using the Vector Network Analyzer had too narrow bandwidth. As a result, when the antenna was connected to the reader, it was not able to read any tags. Since the bandwidth is inversely related to the resistance according toe following formula:

$$B = \frac{1}{2\pi RC}$$

Figure 35 - Antenna Bandwidth

We added another 10K ohms resistor in parallel to the existing resistor and had a total resistance of 5 kohms.

But by decreasing the resistance, we have also reduced the Quality factor of the antenna according to following formula:

$$Q = R\sqrt{\frac{C}{L}}$$

Figure 36 - Quality Factor

After making changing the resistance, we were able to read single tags from up to 3.5" and could read 3 out of 4 tags. The S4100 implements anti-collision algorithm to read multiple tags. However, our reader was unable to write to tags as there were lots of errors. A probable solution to this would be to have a better antenna design with larger sized magnetic wires.

References

Lumex. (1998, October 9). LCM - X01604DXX.

Maxim. (n.d.). *Max712 - Max713.* Retrieved from Maxim IC: http://datasheets.maxim-ic.com/en/ds/MAX712-MAX713.pdf

Systems, L. f. (2008, April 5). *XBee Interface Board*. Retrieved from Laboratory for Intelligent Mechanical Systems: http://hades.mech.northwestern.edu/wiki/index.php/XBee_Interface_Board

Texas Instruments. (2004). *HF Antenna Cookbook - Technical Application Report.*

Texas Instruments. (2003, October). Reader Series 4000 - Base Application Reference Guide.

Texas Instruments. (n.d.). *S4100 Multi-Function Reader Module*. Retrieved November 5, 2008, from Texas Instruments: http://www.ti.com/rfid/shtml/prod-readers-Rf-mgr-mnmn.shtml

Wikipedia. (n.d.). *Nickel-metal hydride battery*. Retrieved November 11, 2008, from Wikipedia: http://en.wikipedia.org/wiki/Nickel_metal_hydride_battery

Appendix A – Cost Analysis

Part	Part Number	Cost per Unit	Quantity Purchased	Part Total
Xbee 802.15.4 OEM RF Modules	XB24-ACI-001	$26.53	2	$53.06
Xbee Starter Kit	XBP24-DKS	$179.00	1	$179.00
Atmega 162 Microcontroller	ATMEGA162-16PU-ND	$8.40	2	$16.80
RFID Transponder In-Lay	Free Trial	$1.79	5	$8.95
RFID Transponder In-Lay	481-1067-1-ND	$1.79	5	$8.95
RFID Reader	481-1052-ND	$104.31	1	$104.31
Liquid Crystal Display	LCM-S01604DSR	$25.19	1	$25.19
Voltage Regulator		$0.35	1	$0.35
ISP (Programmer)	ATAVRISP2-ND	$38.89	1	$38.89
Tantulum Capacitor (LE33)	497-4258-1-ND	$1.07	2	$2.14
		$1.11		$0.00
IC Batt Fast Charger	MAX712CPE+-ND	$9.98	1	$9.98
Energizer Rechargeable battery	N705-ND	$15.00	1	$15.00
PNP Transistor	2N6109GOS-ND	$1.55	2	$3.10
Rectifier	1N4001DICT-ND	$0.39	2	$0.78
1.0UF 50V Mini Alum Elect	P824-ND	$0.18	2	$0.36
0.01UF Capacitor	399-4150-ND	$0.21	2	$0.42
Magnetic antenna wire	Active electronics	$12.31	1	$12.31
Header pins	MRO Electronics	$10.08	1	$10.08
SMA Connector	MRO Electronics	$9.59	1	$9.59
Power Adapter	MRO Electronics	$18.03	1	$18.03
Miscellaneous				$8.00
Shipping and Handling		$8.00	4	$32.00
Sub-Total*				**$378.29**
Taxes (GST)		**5.00%**		**$18.91**
Total*				**$397.20**
Total with Xbee starter kit				$557.29

* As the Xbee Starter Kit will not be reimbursed, the value can be removed from the total

Appendix B – Communication Protocol

Generic Packet Structure

Start of File	Length (bytes)	Command	…
0x01			…

The commands being sent by database to the microcontroller include:

- Book List Download
- Download Authentication
- Request RFID Tag ID
- Security Book Status RSP

Commands being sent from microcontroller to database include:

1. Modify Book State
2. Check if misplaced
3. Response RFID Tad ID
4. Find Misplaced
5. Security Book Status REQ

Commands ending in “0” are requests, whereas “1” are responses. (i.e. for request RFID Tag ID, the cmd is 0x20 whereas the response is 0x21).

Initialization

Download user authentication (0x50)

SOF	0x01	1 byte
Packet Length	0x06	1 byte
Command	0x50	1 byte
User Authentication	0x01 – Librarian 0x02 – Student	1 byte
User ID	(User card no)	2 bytes

Initialize Book Find List (0x10)

SOF	0x01	1 byte
Packet Length	0x27	1 byte
Command	0x10	1 byte
User	0x01 – Librarian	1 byte

Authentication	0x02 – Student	
User ID	(User card no)	2 bytes
Book No	(Increment for each book starting with 1)	1 byte
Book Title	ASCII Characters If < 12, pad with 0x20	12 bytes
Book RFID	RFID Tag ID	8 Bytes
Book Location Description	ASCII Characters If < 12, pad with 0x20	12 Bytes

- -

Request RFID Tag Id (0x20)

SOF	0x01	1 byte
Packet Length	0x03	1 byte
Command	0x20	1 byte

Response RFID Tag Id (0x21)

SOF	0x01	1 byte
Packet Length	0x0B	1 byte
Command	0x21	1 byte
Scanned RFID Tag	RFID Tag ID	8 bytes

- -

Modify Book State Request (0x30)

SOF	0x01	1 byte
Packet Length	0x0E	1 byte
Command	0x30	1 byte
User ID	User Card Number	2 bytes
New State	0x10 – Check-Out 0x20 – Check-In	1 byte
RFID Tag	RFID Tag ID	8 bytes

Modify Book State Response (0x31)

SOF	0x01	1 byte
Packet Length	0x04	1 byte
Command	0x31	1 byte
Response Code	0x01 – Success 0x02 – Fail	1 byte

- -

Find Misplaced (0x40)

SOF	0x01	1 byte

Packet Length	0x13	1 byte
Command	0x40	1 byte
Scanned RFID Tag	RFID Tag ID	8 byte
Scanned Shelf Tag	RFID Tag ID	8 byte

End (0x41)

SOF	0x01	1 byte
Packet Length	0x03	1 byte
Command	0x41	1 byte

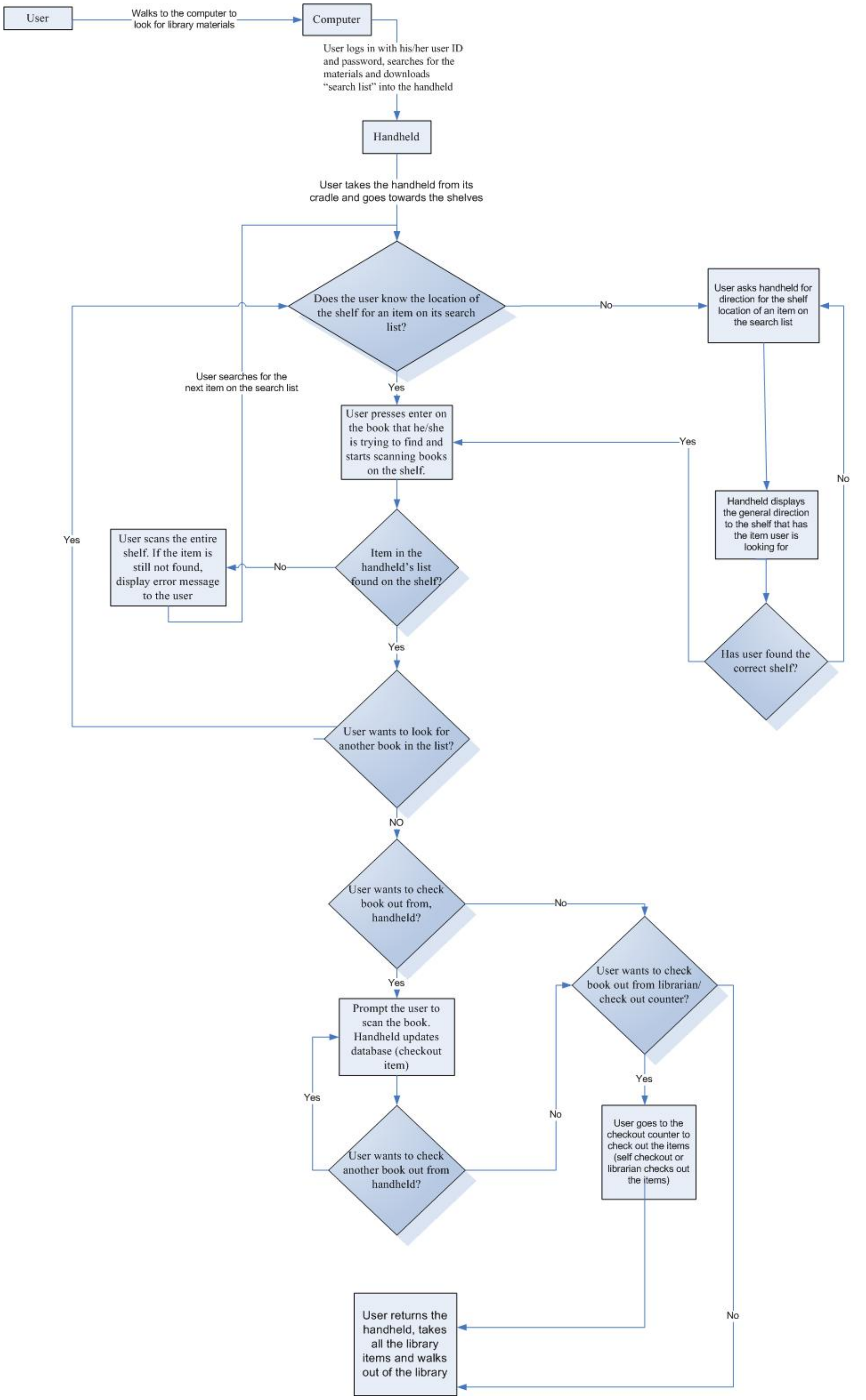
Security Status Request (0x60)

SOF	0x01	1 byte
Packet Length	0x0B	1 byte
Command	0x60	1 byte
Scanned RFID Tag	RFID Tag ID	8 byte

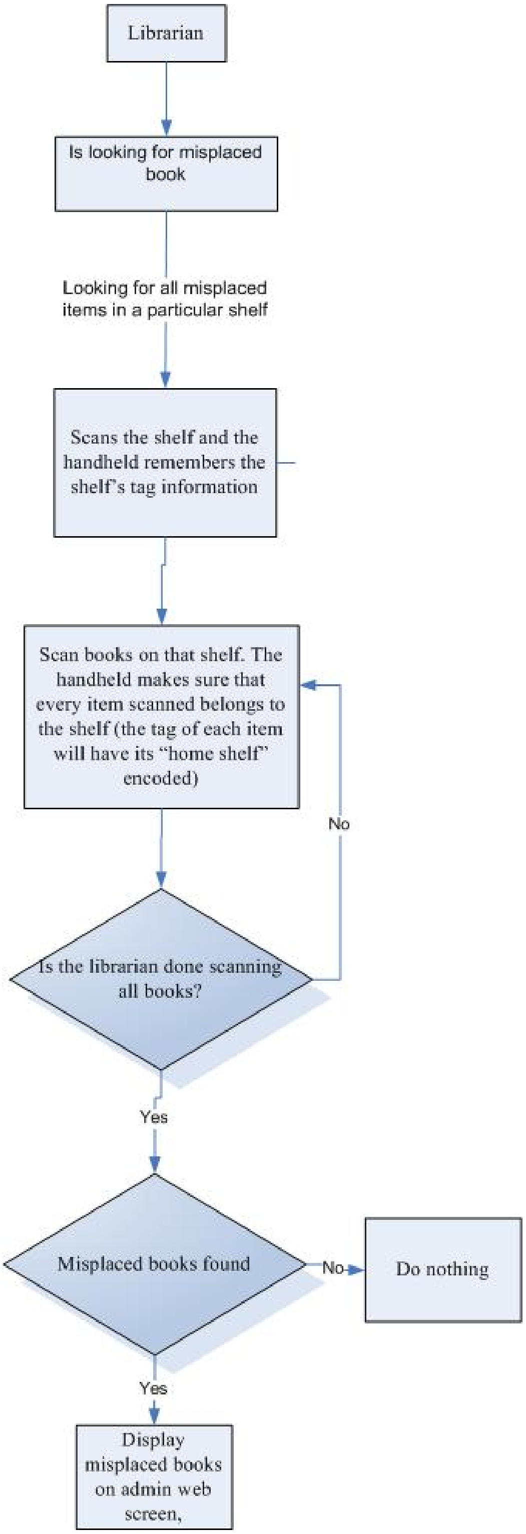
Security Status Response (0x61)

SOF	0x01	1 byte
Packet Length	0x04	1 byte
Command	0x61	1 byte
Book Status	0x10 – Checked-Out 0x20 – Not Checked-Out	1 byte

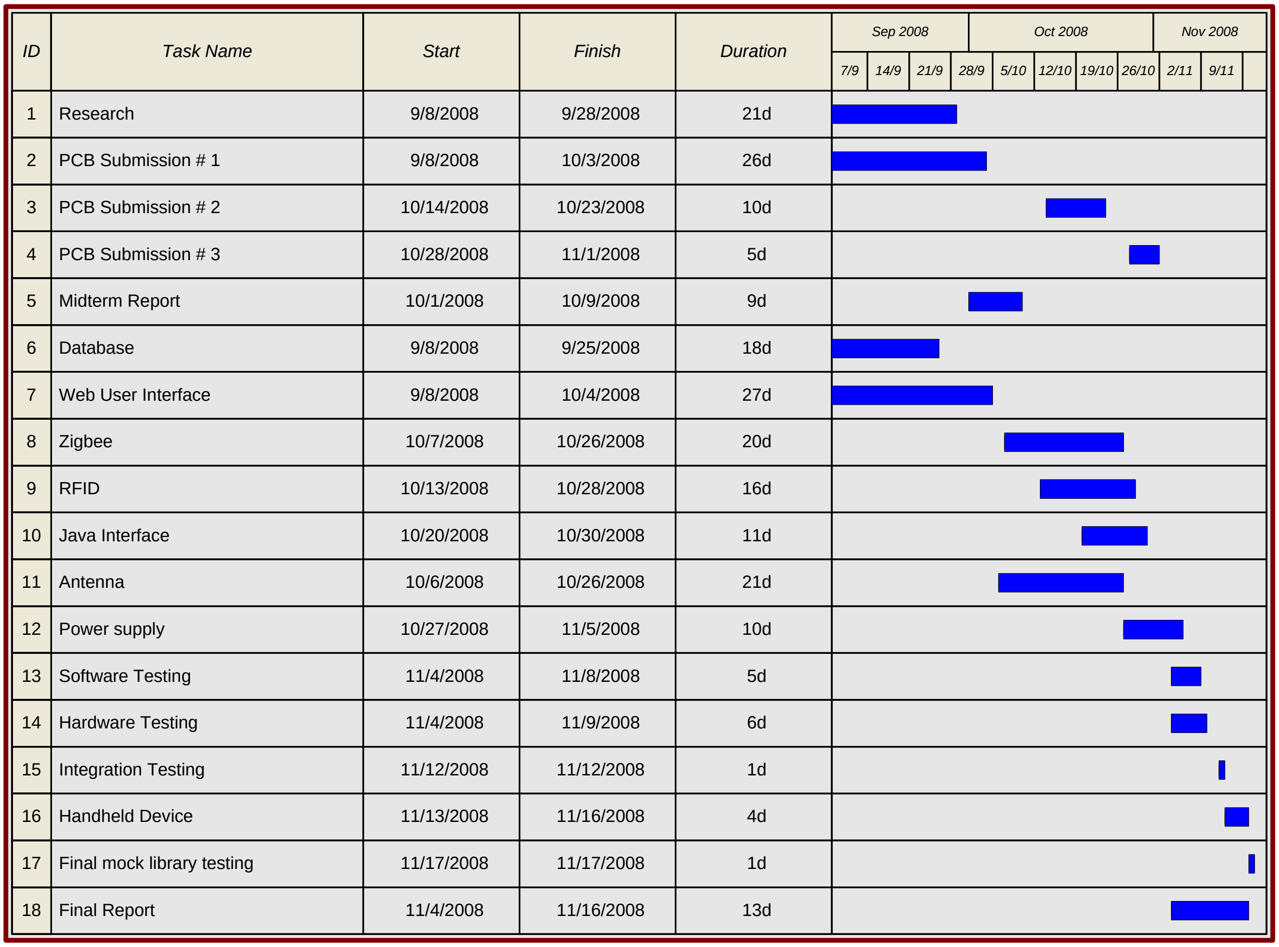
Appendix C – User Flow Chart


User
Walks to the computer to look for library materials
Computer
User logs in with his/her user ID and password, searches for the materials and downloads "search list" into the handheld
Handheld
User takes the handheld from its cradle and goes towards the shelves
Does the user know the location of the shelf for an item on its search list?
No
User asks handheld for direction for the shelf location of an item on the search list
Yes
User searches for the next item on the search list
User presses enter on the book that he/she is trying to find and starts scanning books on the shelf.
Yes
Handheld displays the general direction to the shelf that has the item user is looking for
No
Yes
User scans the entire shelf. If the item is still not found, display error message to the user
No
Item in the handheld's list found on the shelf?
Has user found the correct shelf?
Yes
User wants to look for another book in the list?
NO
User wants to check book out from, handheld?
No
Yes
User wants to check book out from librarian/ check out counter?
Prompt the user to scan the book. Handheld updates database (checkout item)
Yes
Yes
No
User goes to the checkout counter to check out the items (self checkout or librarian checks out the items)
User wants to check another book out from handheld?
No
User returns the handheld, takes all the library items and walks out of the library

Appendix D – Librarian Flow Chart


Librarian
Is looking for misplaced book
Looking for all misplaced items in a particular shelf
Scans the shelf and the handheld remembers the shelf's tag information
Scan books on that shelf. The handheld makes sure that every item scanned belongs to the shelf (the tag of each item will have its "home shelf" encoded)
No
Is the librarian done scanning all books?
Yes
Misplaced books found
No
Do nothing
Yes
Display misplaced books on admin web screen,

Appendix E – Gantt Chart



ID	Task Name	Start	Finish	Duration
1	Research	9/8/2008	9/28/2008	21d
2	PCB Submission # 1	9/8/2008	10/3/2008	26d
3	PCB Submission # 2	10/14/2008	10/23/2008	10d
4	PCB Submission # 3	10/28/2008	11/1/2008	5d
5	Midterm Report	10/1/2008	10/9/2008	9d
6	Database	9/8/2008	9/25/2008	18d
7	Web User Interface	9/8/2008	10/4/2008	27d
8	Zigbee	10/7/2008	10/26/2008	20d
9	RFID	10/13/2008	10/28/2008	16d
10	Java Interface	10/20/2008	10/30/2008	11d
11	Antenna	10/6/2008	10/26/2008	21d
12	Power supply	10/27/2008	11/5/2008	10d
13	Software Testing	11/4/2008	11/8/2008	5d
14	Hardware Testing	11/4/2008	11/9/2008	6d
15	Integration Testing	11/12/2008	11/12/2008	1d
16	Handheld Device	11/13/2008	11/16/2008	4d
17	Final mock library testing	11/17/2008	11/17/2008	1d
18	Final Report	11/4/2008	11/16/2008	13d