

Final Documentation

Smart Windows

5/3/2010
EE Senior Design 2010
Professor Schafer

Kelley Daniels
Tommy Haunert
David Shilling
Andy Spangler



Table of Contents

1 Introduction.....	3
2 Detailed Project Description.....	10
2.1 Theory of Operation.....	10
2.2 System Block diagram.....	12
2.3 PC Control Unit (PCU).....	14
2.4 On-Window Unit (OWU).....	36
2.5 Remote Control (RCU).....	47
3 System Integration Testing.....	51
3.1 General Software Testing Paradigms.....	51
3.2 Pure PC Software Testing.....	53
3.3 Pure Microcontroller Software Testing.....	53
3.4 PC-Microcontroller Interaction Testing.....	53
4 Installation Manual and User's Guide.....	55
4.1 Installation Guide.....	55
4.1.1 PC Unit Setup (<i>optional</i>).....	59
4.1.2 Window Unit Setup.....	60
4.1.1 Remote Control Setup (<i>optional</i>).....	63
4.2 User's Guide.....	64
4.2.1 Manual Mode.....	64
4.2.2 Eco Mode.....	64
4.2.3 Timer Mode.....	65
5 Conclusions.....	68
6 Appendices.....	69
6.1 Complete Hardware Schematics.....	70
6.2 Complete Software Listing.....	73
6.2.1 Microcontroller Software.....	73
6.2.2 PC Software.....	161
6.2.3 Android Software.....	187
6.3 Bill of Materials.....	194
6.4 Data Sheets.....	196

1 Introduction

In collaboration with Solar Shades, Smart Windows has sought to create a bridge between Notre Dame EE Senior Design and Notre Dame's Innovation Park. In addition, Smart Windows sought to solve a very important technical problem.

In a world faced with rising energy demand and depleting energy supply, the next generation of technology must take into account energy usage at every level. According to the United States Energy Information Administration's (EIA) Residential Energy Consumption Survey (RECS), air conditioning consumes a significant and growing portion of US electric power. From 1997 to 2005, the RECS shows that the percent of US residential electric power used for air conditioning rose from 14% to 20.2%.¹

To fight rising energy usage, the US Department of Energy's Energy Savers program recommends reducing the stress on air conditioners. One important Department of Energy guideline is to use window treatments like blinds and shades to reduce the thermal gain through the windows due to radiant solar energy: "Install window shades or other window treatments and close the shades. Shades will help block out not only direct sunlight, but also radiated heat from the outdoors, and insulated shades will reduce the conduction of heat into your home through your windows." In fact, the Department of Energy says that reflective shades can reduce heat gain up to 45%.²

In addition to reducing air conditioning energy consumption, window treatments have other functions. For example, the September 23, 2009 edition of the University of Notre Dame and Saint Mary's College newspaper *The Observer* recommends window treatments as a crime prevention tool. In an article entitled "Burglars Target Off-Campus Housing," South Bend Police Captain Phil Trent notices, "There's people with their front windows right open and I can see a 50-inch plasma screen from the street. You can see someone with the lights on in their house and they're working on a laptop computer...A burglar can do an assessment of what they can steal just by walking down the street looking in the windows." The article states, "To prevent burglaries, students should keep their windows and curtains closed."³

These benefits of window treatments are only effective, however, if the homeowner is diligent in opening and closing them. To access the energy benefits of window treatments, the homeowner must constantly monitor sunlight exposure. To access the security benefits, the homeowner must close every treatment prior to leaving the house.

¹ EIA online RECS 2005 Status Report. <<http://www.eia.doe.gov/emeu/recs/contents.html>>.

² DOE Energy Savers.
<www.energysavers.gov/your_home/space_heating_cooling/index.cfm/mytopic=12353>.

³ Mervosh, Sarah. "Burglars Target Off-Campus Housing." *The Observer*. 23 Sept 2009.
<<http://media.www.ndsmcobserver.com/media/storage/paper660/news/2009/09/23/News/Burglars.Target.OffCampus.Housing-3780142.shtml>>.

Since few homeowners can afford to give this level of attention to their window treatments, the benefits of properly operated window treatments are rarely utilized. For these benefits to be tapped into, the windows must operate automatically in the homeowner's stead. It is this problem of maximizing window treatment utility through automation and electronic intelligence that the Smart Windows design addresses.

In order to address this problem of maximizing window treatment utility, we proposed a window treatment automation system called Smart Windows. As originally proposed, the Smart Windows system is centered on a PC Control Unit (PCU), which consists of a custom-designed PC application with graphical user interface (GUI) and appropriate wireless interfacing hardware. Through this wireless interface, the user can issue commands to individual On-Window Units (OWUs). Users not at their computer can issue commands to individual windows from a wireless remote control unit (RCU) or from directly from a window-mounted panel.

Users have the choice to manually control individual windows, to implement sensor-based control, or operate windows on a timer. These choices are defined as modes. The OWU, operating in one of these three modes, can drive the window treatment using a direct current (DC) motor. In sensor-controlled mode, or Eco Mode, the OWU uses a light sensor to decide if the window blind should be opened or closed to maximize the efficiency of the household HVAC. In Timer Mode, the window treatment open or close at preset times, typically at night or during working hours. In manual mode, the user operates the windows individually from the PC application, remote control, or on-window buttons. Users choose from opened, closed, or half opened window treatments. The OWU derives its intelligence from an embedded microprocessor.

In order to solve the problem of maximizing window treatment utility, this Smart Windows system must meet certain requirements. These requirements are shown in **Table 1.1**. These are the same requirements that were proposed along with the Smart Windows system in the fall. **Table 1.1** indicates whether or not the final Smart Windows design met each requirement.

Table 1.1 Smart Windows system requirements

Subsystem and Interface Requirements		
<i>On Window Unit (OWU)</i>		
Requirement	Description	Feedback
General	Must be able to control window treatments intelligently Must continue to operate when wireless communication is broken	Completed; Requirement Achieved
Size	Must have total dimensions less than or equal to 8" x 5" x 4"	Completed; Requirement Achieved
Weight	Must be less than 5 pounds Must operate with Solar Shades windows Must operate with at least on set of venetian blinds Preferably operates with all window blinds	Completed; Requirement Achieved; Does not operate with general window treatments
Power	Must use 4-8 rechargeable AA batteries lasting a minimum of 14 days	System used 8 rechargeable AA batteries lasting only 5 days
Microcontroller Software	Must use a reasonable amount of program memory Must operate the drive the motor appropriately when necessary Must periodically monitor the light sensor and manual buttons Must decode received wireless messages Must enter power-saving mode when possible	Completed; Requirement Achieved
Motor	Must be a DC motor capable of at least 20 oz-in of torque Must be geared to turn less than 100 rotations per minute Must have a safety clutch to protect the window treatment Must make minimal noise when operating Must meet power requirements (see "Power") above	Completed; Requirement Achieved
Light Sensor	Must be capable of differentiating a sunny day from a cloudy day Must ignore light coming from inside the house Must be report light levels to microcontroller using minimal I/O pins Must meet power requirements (see "Power") above	Completed; Requirement Achieved
Wireless Transceiver	Must send and receive messages at an indoor distance of 100 feet Must be able to address messages to a particular target Must not create interference with other household	Completed; Requirement Achieved

	<p>items</p> <p>Must be able to interface with a microcontroller quickly and reliably</p> <p>Must meet power requirements (see “Power”) above</p>	
Manual Buttons	<p>Must reliably control the system when used</p> <p>Must be easily accessible</p> <p>Must let the user open or close the treatment one increment</p>	Completed; Requirement Achieved
EEPROM Memory	<p>Must be able to hold 1000 bytes of non-volatile memory</p> <p>Must maintain memory for at least a week without power</p> <p>Must interface with a microcontroller</p>	Completed; Requirement Achieved
Real Time Clock	<p>Must be capable of accepting current time from microcontroller</p> <p>Must keep time and date accurately from that point forward as long as power is connected.</p>	Completed; Requirement Achieved
<i>PC Control Unit (PCU)</i>		
General	<p>Must give the user the highest amount of control over the system</p> <p>Must have control over every connected window treatment in the house</p>	Completed; Requirement Achieved
Power	<p>Must be able to draw power from the PC USB connection</p>	Completed; Requirement Achieved
PC Software	<p>Must be able to interface to a microcontroller through USB</p> <p>Must have an intuitive graphical user interface</p> <p>Must be capable of placing each window into one of the three modes</p> <p>Must be capable of controlling individual windows when in manual mode</p> <p>Must store at least Wake-up/Work/Return-from-work/Sleep times locally</p>	Completed; Requirement Achieved
EEPROM Memory	<p>Must be able to hold 1000 bytes of non-volatile memory</p> <p>Must maintain memory for at least a week without power</p> <p>Must interface with a microcontroller</p>	Completed; Requirement Achieved
Real Time Clock	<p>Must be capable of accepting current time from microcontroller</p> <p>Must keep time and date accurately from that point forward as long as power is connected.</p>	Completed; Requirement Achieved
Microcontroller Software	<p>Must decode messages received through the USB connection</p> <p>Must pass these messages to the wireless transceiver</p>	Completed; Requirement Achieved

Wireless Transceiver	<p>Must send and receive messages at an indoor distance of 100 feet</p> <p>Must be able to address messages to a particular target</p> <p>Must not create interference with other household items</p> <p>Must be able to interface with a microcontroller quickly and reliably</p> <p>Must meet power requirements (see “Power”) above</p>	Completed; Requirement Achieved
<i>Remote Control Unit</i>		
General	<p>Must give the user remote control over a particular window treatment</p> <p>Must be capable of converting a particular window to manual mode</p> <p>Must be able to select an active window treatment to control</p>	Completed; Requirement Achieved
Power	Must use 2-4 rechargeable AA batteries lasting a minimum of 14 days	
Microcontroller Software	<p>Must monitor the user input buttons</p> <p>Must pass user commands to the wireless transceiver</p> <p>Must conserve power when possible</p>	Completed; Requirement Achieved
Manual Buttons	<p>Must reliably control the system when used</p> <p>Must be easily accessible</p> <p>Must let the user open or close the treatment one increment</p> <p>Must let the user select the active window to communicate with</p> <p>Must interface with the microcontroller directly through 8 or less I/O pins</p>	Completed; Requirement Achieved
Wireless Transceiver	<p>Must send and receive messages at an indoor distance of 100 feet</p> <p>Must be able to address messages to a particular target</p> <p>Must not create interference with other household items</p> <p>Must be able to interface with a microcontroller quickly and reliably</p> <p>Must meet power requirements (see “Power”) above</p>	Completed; Requirement Achieved
EEPROM Memory	<p>Must be able to hold 1000 bytes of non-volatile memory</p> <p>Must maintain memory for at least a week without power</p> <p>Must interface with a microcontroller</p>	Completed; Requirement Achieved
LCD Screen	<p>Must display the actively selected window on-screen</p> <p>Must be consistent with Power requirements above (See “Power”)</p>	Completed; Requirement Achieved

<i>Wireless Interface</i>		
Distance	Must connect the various units at a distance of 100 feet	Completed; Requirement Achieved
Power	Must not consume more power than the various units can provide	Completed; Requirement Achieved
Interfacing	Must be able to address specific units while ignoring others Must operate on several channels to avoid inter-house interference	Completed; Requirement Achieved
Speed	Must be fast enough to send and receive a message in less than 1 second	Completed; Requirement Achieved

The expectations and requirements set for Smart Windows early on were extensive. However, these requirements were kept in general terms, allowing us the freedom of multiple possible solutions. Therefore, as the design process unfolded, our design requirements changed very little. However, the specific design decisions made to fill these requirements were constantly changing.

While there are many examples of the Smart Windows design changing and adapting, perhaps the best is in the design of the motor limiting mechanism. The requirement for this feature was simply that the motor would be able to stop accurately when the window was open, closed, or half open. There are many ways to accomplish this level of control. Originally, we proposed a design that used a stepper motor to accurately position the blinds. This solution was eventually abandoned in favor of cheaper, more available DC motors. As our design unfolded, this limiting mechanism went through many stages of design. Motor shaft encoders, carefully timed motors, and motor current monitoring were all proposed as possible solutions. In the end, we chose to use a physical limit switch for the motor's open and closed positions and an optical interrupter for the motor's half open position. This design was preferred because it is simple and lightweight. However, the use of limit switches did limit to specific window treatments, causing us to abandon our hopes of a general window treatment solution.

Overall, the system outperformed our minimum expectations. With only one exception, **Table 1.1** shows that our design was able to meet all of the minimum expectations set forth for Smart Windows. Our only failure was battery life. Our project was not able to meet the expected two weeks of battery life between charges. This was due to a variety of reasons. First, did not anticipate the large energy consumption by the wireless network. Second, we had hoped to use several sets of batteries in parallel. However, because the motor needed such a large voltage we were forced to place all of our batteries in series. A more advanced system with a larger budget could have used solar battery charging to circumvent this requirement.

The only other failure of the Smart Windows system is lack of generality. While controlling general window blinds was not a requirement of our original solution, it was a goal of ours. In the end, our design decisions limited us to specific window treatments, the Lowes blinds and the Solar Shade. A system that can control any set of window blinds would find more marketability. Some have suggested that our project ought to be able to raise and lower window blinds in addition to opening and closing them. While this would certainly be an attractive feature for venetian blinds, polarizing Solar Shades windows cannot be opened or closed. Since our main focus was always on making a product immediately attractive to Solar Shades, raising and lowering blinds was never included in our expectations or requirements.

In addition to meeting our minimum requirements, the Smart Windows design includes a variety of features not originally proposed. The final Smart Windows product offers not only PC-level control, but also on-the-go control through an Android mobile phone application. Smart Windows is able to monitor and report its own battery life, a feature the user will appreciate. Also, in manual mode Smart Windows is capable of not only moving the windows to an open, middle, and closed state, but also driving the windows to any state in between. Because Smart Windows was able to meet the vast majority of its design requirements, much of the design time was spent adding to and expanding on the Smart Windows feature set.

2 Detailed Project Description

2.1 Theory of Operation

The Smart Windows system consists of a motor block and three main units, the PC control unit (PCU), the on-window unit (OWU), and the remote control (RCU). Each unit is broken down into several subsystems and the interfaces between them. These units communicate through two-way RF communication using ZigBee protocol.

Each unit requires a printed circuit board (PCB) containing a microcontroller, a ZigBee transceiver circuit, and peripherals. Some peripherals are mounted directly on the main board, but others are mounted on secondary boards. Since each unit requires many of the same basic peripherals, a single PCB was designed. This board will be referred to as the main board. The main board is capable of accepting all peripherals in use in this project. However, the main board for any particular unit only has the necessary peripherals attached. All main boards contain a microcontroller, ZigBee transceiver circuit, a microcontroller programmer circuit, and a 20-MHz ceramic oscillator circuit supplying the microcontroller with its clock.

The PCU consists of the PC application and a main board. The PC connects to the microcontroller on the board through a USB interface. In addition the PC application connects to an Android mobile phone application through the Notre Dame Android lab server. The Android application is capable of issuing window instructions through the PC application. The ZigBee circuitry connects to the microcontroller through a standard serial parallel interface (SPI) synchronous serial interface. The PCU main board has three peripherals attached, a real-time clock, a serial EEPROM chip, and a FTDI serial USB/UART device. The real-time clock interfaces with the microcontroller through SPI and will keep the current time and date. The serial EEPROM acts as non-volatile data memory and connects to the microcontroller through SPI. The serial EEPROM is not currently in use, as the on-chip EEPROM is sufficient. However, future enhancements may require the use of the EEPROM. The FTDI serial USB/USART device allows the microcontroller USART to communicate through standard asynchronous serial communication with the PC through USB. The PCU main board receives its power from the USB connection to the PC.

The OWU consists of a main board with a variety of peripherals. These peripherals include a real-time clock, a serial EEPROM, a DC-input regulation circuit, a DC/DC voltage converter, a limit switch jumper, a button harness, and a motor board, and a light sensor board. The DC-input circuit accepts a voltage from a battery stack and regulates it down to the required 3.3V. The DC/DC converter steps up this 3.3V signal to the 5.0V signal used by several of the peripherals. The limit switch jumper contains the pull-down resistors necessary to operate the motor limit switches and the resistors necessary to bias the photo-interrupter. The button harness contains five panel-mount buttons and pull-down resistor and connects to the main board through jumper wires. The motor board contains the h-bridge circuit needed to run the DC motor and connects to the main board through jumper wires. The light sensor board contains a phototransistor that produces a

voltage proportional to ambient light levels. It also interfaces to the main board through jumper wires.

The RCU also consists of a main board with several peripherals. These peripherals include a DC-input circuit, LCD screen, a DC/DC voltage converter, a button harness, and a serial EEPROM chip. The DC input circuit accepts a battery voltage and regulates it down to a usable value. The LCD screen displays the name of the currently selected window. It connects to the microcontroller through an SPI interface. The DC/DC converter provides the 5V signal needed by the LCD. The button harness holds the buttons that cycle through selected window and command the selected window to open or close. The serial EEPROM is not used in this implementation, but is available for future needs.

The motor block consists of a DC motor, window blinds, two limit switches, a photo-interrupter two ID hubs, a rubber spider coupling, and a steel rod. The DC motor drives the window blinds, causing the shaft on the blinds to rotate. The ID hubs and spider coupling will allow the window blind shaft to couple to a bent steel rod. As the shaft turns, the steel rod will come into contact with the limit switches. A press of a limit switch will indicate that the window is fully opened or closed. When the steel rod breaks the beam of the photo-interrupter, the window will be half opened.

2.2 System Block diagram

A block diagram illustration of the system is shown in **Figure 2.2.1**. An in-depth illustration of the PCU is shown in **Figure 2.2.2**. The OWU is shown in **Figure 2.2.3**, and the RCU is shown in **Figure 2.2.4**.

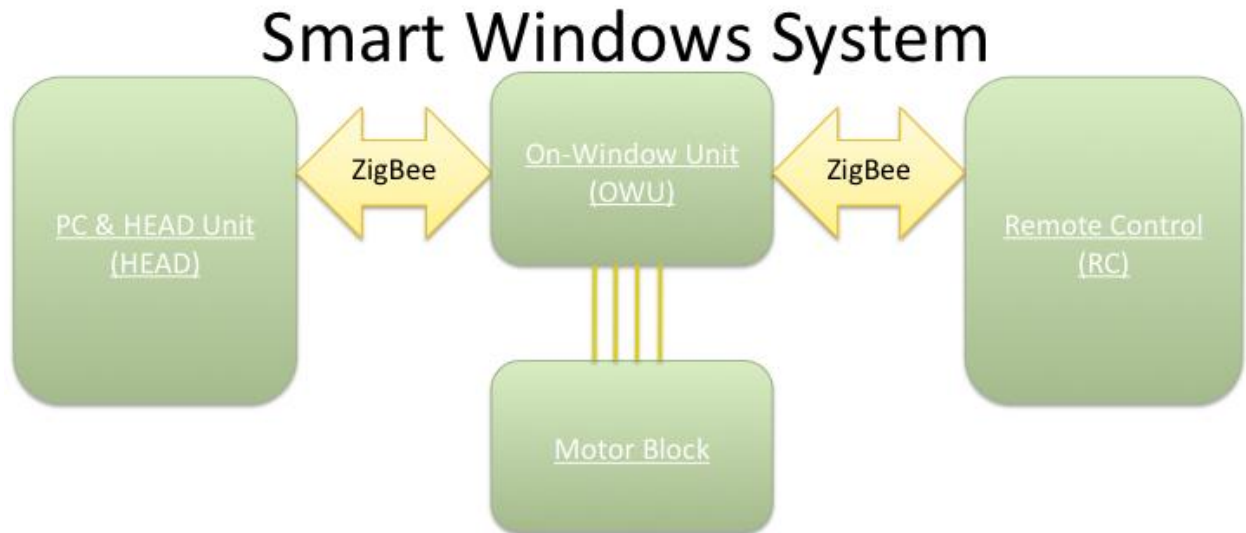


Figure 2.2.1 Block Diagram of the Smart Windows system

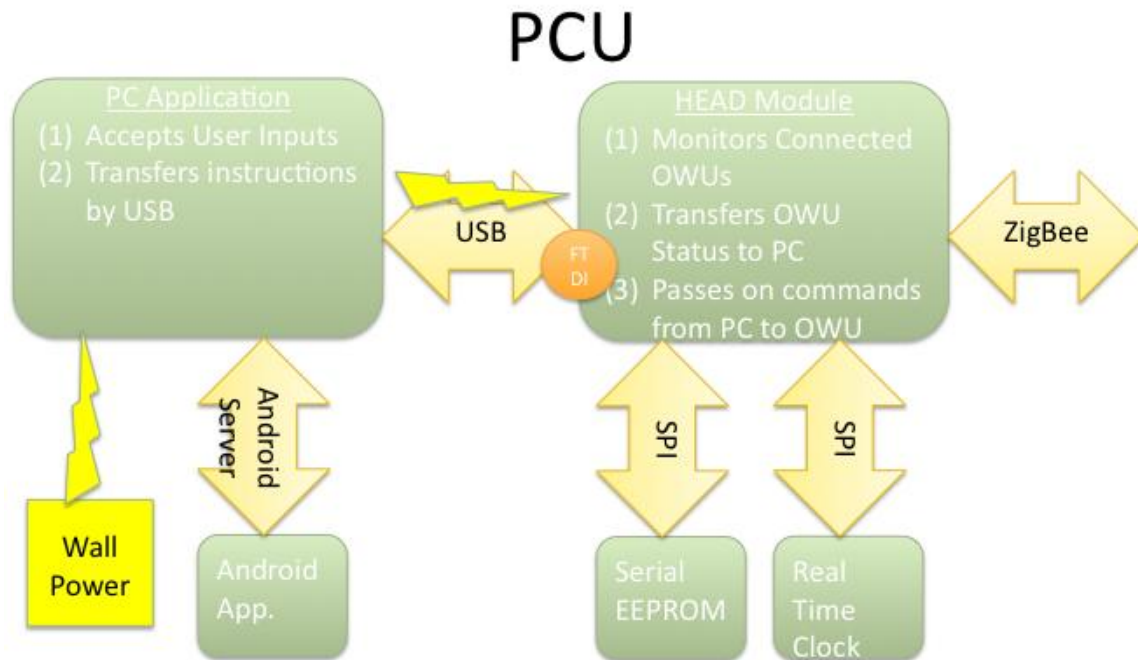


Figure 2.2.2. Block diagram of the PCU.

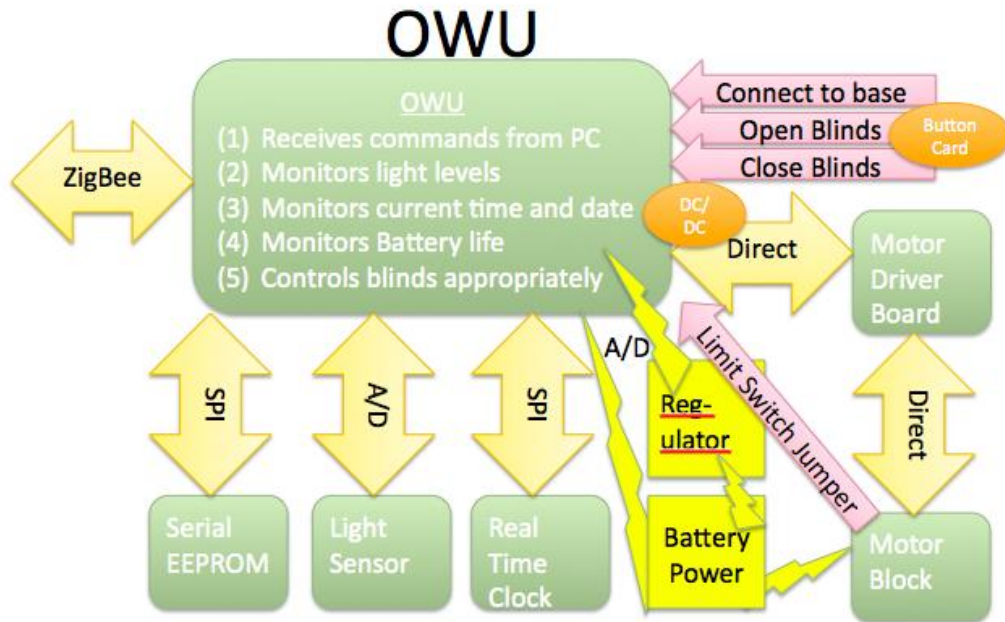


Figure 2.2.3. Block diagram of the OWU.

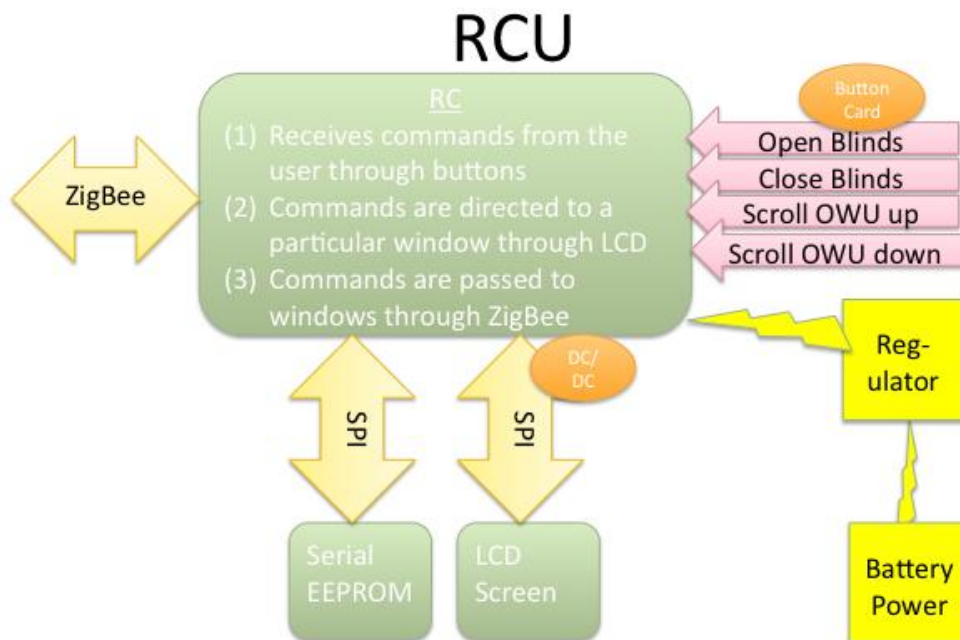


Figure 2.2.4. Block diagram of the RCU

2.3 PC Control Unit (PCU)

The PCU subsystem will allow the user to interface to the system through a personal computer application. The application will give the user a variety of controls over the operation of the window, as described in section 2.3.1. The user will also be able to contact the PC application indirectly through an Android mobile phone application. Commands entered on the PC application will be transmitted through a USB protocol a PC-connected microcontroller board, as described in Section 2.3.2. The microcontroller board will interpret these commands and forward them on to the appropriate windows using a ZigBee wireless interface, as described in Section 2.3.3.

2.3.1 PC application

For the development of our SmartWindow PC interface, we chose to use the language Python, with PyQt and PyUSB libraries. Our decision began with how to best develop a professional looking GUI that we could rapidly develop and that would be powerful enough for our task. (We did not consider USB communication at first, assuming this could be done in every major application language.) Towards this goal, we considered the several languages and graphics plugins. The pros and cons of each considered language are shown in **Table 2.3**. Ultimately, we decided on Python as our language for this project.

Table 2.3. Summary of considered languages.

Language (Graphics Libraries)	Pros	Cons
C++ (with QT)	Professional and powerful. Cross-platform. Free.	QT with C++ is probably overkill and has steep learning curve.
C++ with C, assembly and native Win32 API calls.	Very efficient and clean. Free.	For Windows platform only, and way overkill for an application that will never actually be sold.
C++/C# (with Microsoft Visual Studio libraries)	Relatively easy to code. MFC classes are professional grade.	Deployable only on Windows. MFC classes have licensing fee. May require user to install Visual Studio DLL's.
JAVA (with standard Swing GUI libraries)	Professional and cross-platform. Free.	Hard to compile (without extra tools and significant effort). May require user to install Java Virtual Machine.
Python (with QT, called PyQt)	As professional looking as C++ with QT, but slightly easier and faster to program. Cross-platform. Free.	Usually an interpreted language, but we can use external tools (such as py2exe) to easily make executable.

After we decided to use Python with PyQt, we searched for USB libraries. We initially found “UsbLib” and then later “PyUsb,” a simpler thin wrapper around the “UsbLib” API. As far as open-source libraries, these two were the only options we found.

The PC application will operate in three classes. The first class occurs during startup and initializes the application by loading a settings file stored locally on the PC. The second class represents standard operation. The third class consists of child windows that allow the user to input additional information about a specific window. **Figure 2.3.1** documents the flow of the startup class. **Figure 2.3.2** shows a graphical illustration of the program creating the settings file. **Figure 2.3.3** documents the flow of the main class. **Figure 2.3.4** documents the flow of the window-child class.

PC SOFTWARE: START-UP Class
<ul style="list-style-type: none">• <i>Displays</i><ul style="list-style-type: none">• Loading Screen
<ul style="list-style-type: none">• <i>User Interaction</i><ul style="list-style-type: none">• None
<ul style="list-style-type: none">• <i>Structure</i><ul style="list-style-type: none">• Threads<ul style="list-style-type: none">• Thread 1 (UI Thread)<ul style="list-style-type: none">• Loading Bar/Animation• Thread 2 (Background Thread)<ul style="list-style-type: none">• USB Communication with HEAD-UNIT module• <i>Preconditions</i><ul style="list-style-type: none">• An instance of the SmartWindows application is not already running• <i>Postconditions</i><ul style="list-style-type: none">• Has established which window module are active• Has sent out current time to the HEAD-UNIT module
<ul style="list-style-type: none">• <i>Hierarchy</i><ul style="list-style-type: none">• Instances: Only 1, Parent: None, Children: None

Figure 2.3.1. Flow of the startup class.

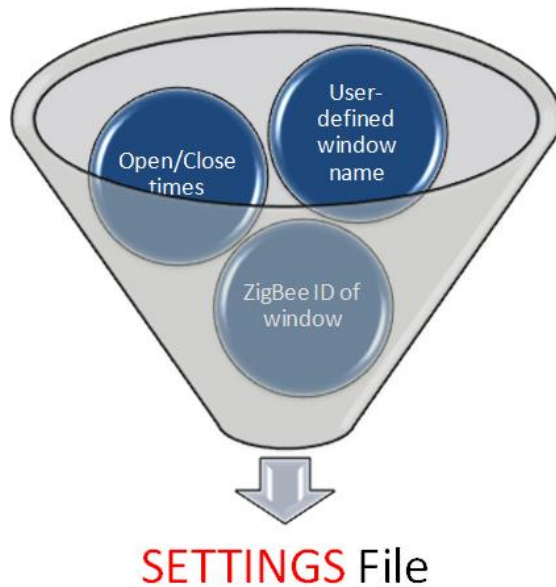


Figure 2.3.2. Settings file data diagram.

PC SOFTWARE: MAIN Class
<ul style="list-style-type: none">• <i>Displays</i><ul style="list-style-type: none">• Window Module Attributes: Name, Open/Close States, Modes
<ul style="list-style-type: none">• <i>User Interaction</i><ul style="list-style-type: none">• Button for each window module<ul style="list-style-type: none">• Clicking online window module starts up WINDOW-CHILD instance• Menu allows<ul style="list-style-type: none">• Emergency Open All, Emergency Close All, Remove Window
<ul style="list-style-type: none">• <i>Structure</i><ul style="list-style-type: none">• Threads<ul style="list-style-type: none">• Thread 1 (UI thread)• Thread 2 (Background Thread)<ul style="list-style-type: none">• Receives intermittent updates on modules that are online• Receives intermittent temperature readings• Preconditions<ul style="list-style-type: none">• START-UP has run and attempted to connect with window modules• Postconditions<ul style="list-style-type: none">• Has loaded all SETTINGS files available and displayed information• On object destroy<ul style="list-style-type: none">• Closes itself and all child (WINDOW-CHILD) instances
<ul style="list-style-type: none">• <i>Hierarchy</i><ul style="list-style-type: none">• Instances: Only 1, Parent: None, Children: WINDOW-CHILD instances

Figure 2.3.3. Flow of the main class.

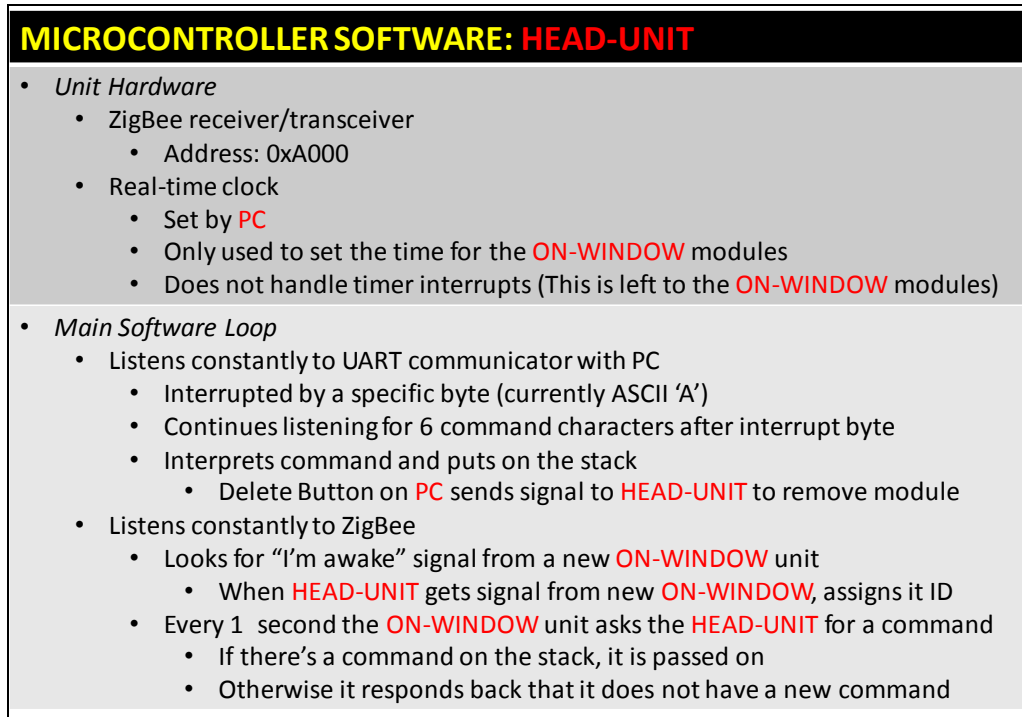


Figure 2.3.4. Flow of the window-child class.

“A designer usually intends an artefact to have some function(s). This influences the way (s)he designs the artefact and chooses to shape its form in such a way that it gives the user clues to the intended functions. Doing so the form itself becomes an intended function. The artefact in itself is just a physical object.” *“Form is Function“ (Bosse Westerlund)*

In designing the SmartWindows PC application, we were intimately aware of how important it was to make the interface intuitive and fun, as well as powerful and customizable. Controlling windows from ones computer far from a life necessity and we knew that users would only do so if it were quick and easy to use.

Towards this end, nearly all controls were put on the main graphical interface alongside friendly icons, with only “delete all windows,” “synchronize current time,” and “exit” (commands not usually necessary) under the menu bar. The buttons to manually open, half-open, and close the windows only became available after the window was put into “manual” mode, to avoid any possible confusion. Setting specific times was put in a separate window which could be opened and closed as desired.

Another important criteria for such a program that sends and receives commands externally, is positive user feedback. After each command was sent to the microcontroller, the program alerted the user (via a message box) that the command was successfully sent. Also, if the head module became disconnected at any point, the user would be notified. The design on the software is shown in **Figure 2.3.5**

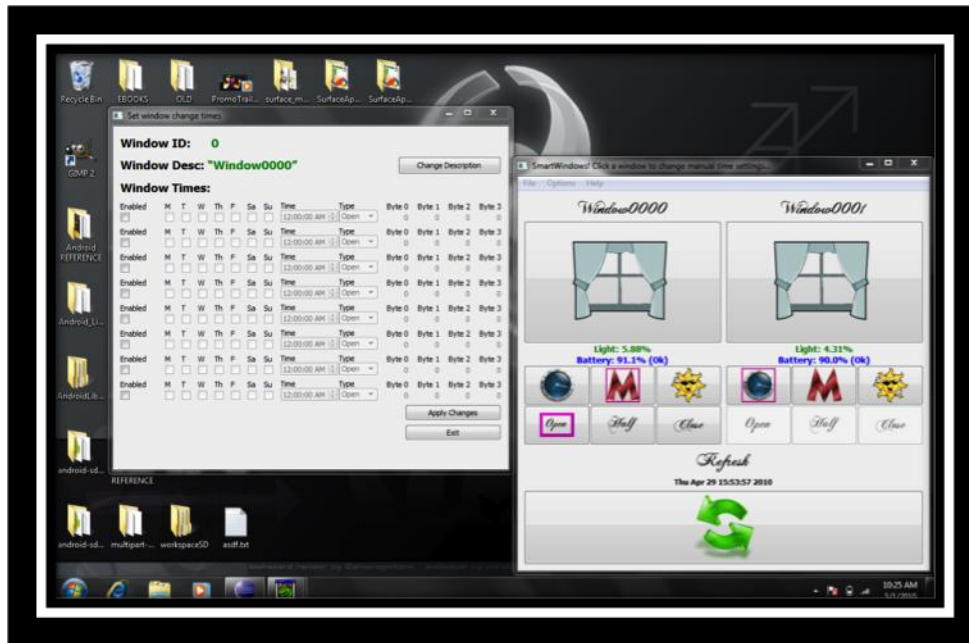


Figure 2.3.5 Screenshot of the PC software

Outside of the USB communication parts of our PC Program there are two major types of code we tested over: deterministic commands and highly variable commands. Out of necessity, we employed two different methods for testing these two types of code.

Deterministic commands, such as “change mode to ECO,” “delete a window,” or “change the name of a window” were tested using comprehensive testing. Every possible command was sent to the microcontroller or PC and to ensure that each command is successfully executed.

Other commands, such as the custom timing for the “Automatic Mode” of our windows, is customizable to the point of being nearly infinitely variable. Thus, we settled for the systematic testing as extensively as our patience allows. For instance, after verifying that all eight open/close times on our form are programmatically identical, we entered extensive day and times and take note of the output bytes. Since we used certain built-in objects, such as the python time-entry widget, we can be confident that it is not possible to enter in unintelligible times (such as 25:61 o’clock), as many other programmers have verified this is not possible with the widget. Whenever possible we have used reliable blocks such as this.

In addition, two members of our team completed an inspection and walkthrough of the source code together, in keeping with the principles of pair programming. A combination of clean programming, carefully checked over, and extensive testing of major cases is sufficient in our case. Writing computer testing routines was not feasible in our timeframe.

The Smart Windows Android Application allows users to open, close, or half-open two windows (Windows “0” and “1”) from their Google Android equipped device. This application works from anywhere in the world as long as the PC application is running, the windows are connected to the PC, and the cell phone has either 3G or WiFi connection.

The app worked by interpreting the user clicks as one of following six commands:

- Open Window 1
- Half-open Window 1
- Close Window 1
- Open Window 2
- Half-open Window 2
- Close Window 2

These commands are then routed to 6 slightly different PHP “\$_GET” commands. PHP is a popular web scripting language, and “\$_GET” is the shorter and simpler of the two major methods of passing data to the server: “\$_GET” and “\$_POST.” Once the server interpreted the PHP command, it would modify an XML file, which included a tag for whether the current state of the window should be opened, half-opened, or closed. Then, back on the PC computer running the Smart Windows application, a background thread would check this XML file 2 times per second. If it detected that the desired state was different from the current state, it would send out a command to switch to the new state. The operation of this Android application is shown in **Figure 2.3.6**.

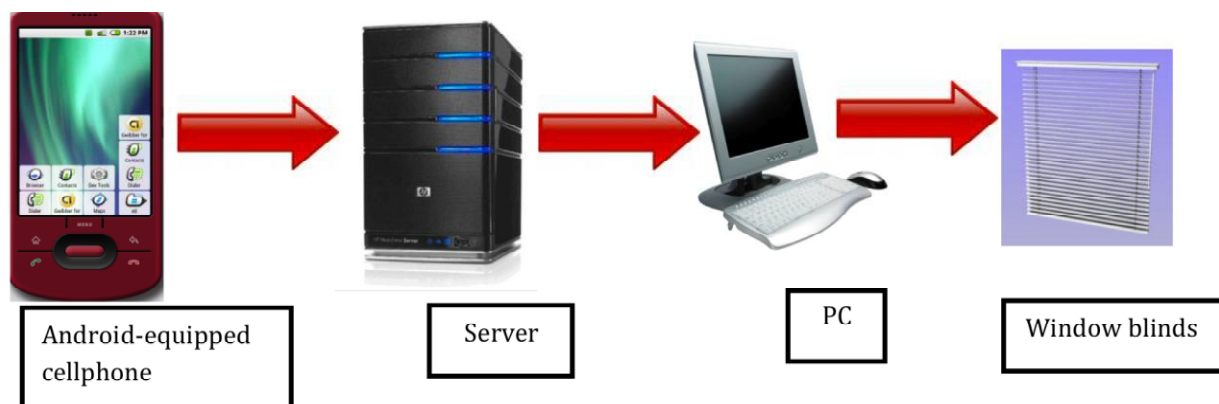


Figure 2.3.6 Operation of the Smart Windows Android Application

2.3.2 USB Interface

The USB interface consists of asynchronous serial communication between the PC and the main board. The PC interacts with the USB channel through a virtual COM port. This USB signal is converted into simple asynchronous serial communication through the FTDI part described in Section 2.3.3.11. The microcontroller will send and receive bytes from the FTDI part through its on-board USART. **Figure 2.3.4 a-g** shows a summary of the communication protocol between the microcontroller and the PC.

Message Type A: Startup

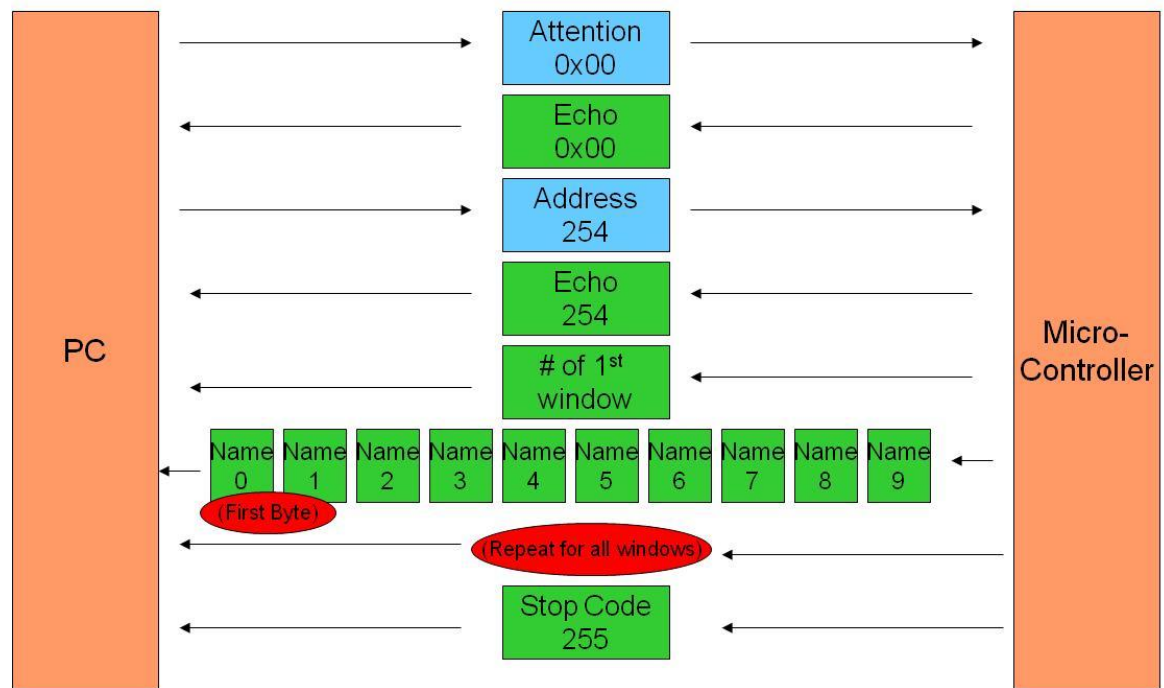


Figure 2.3.4.a. Communications protocol for USB link.

Message Type B: Runtime Messages

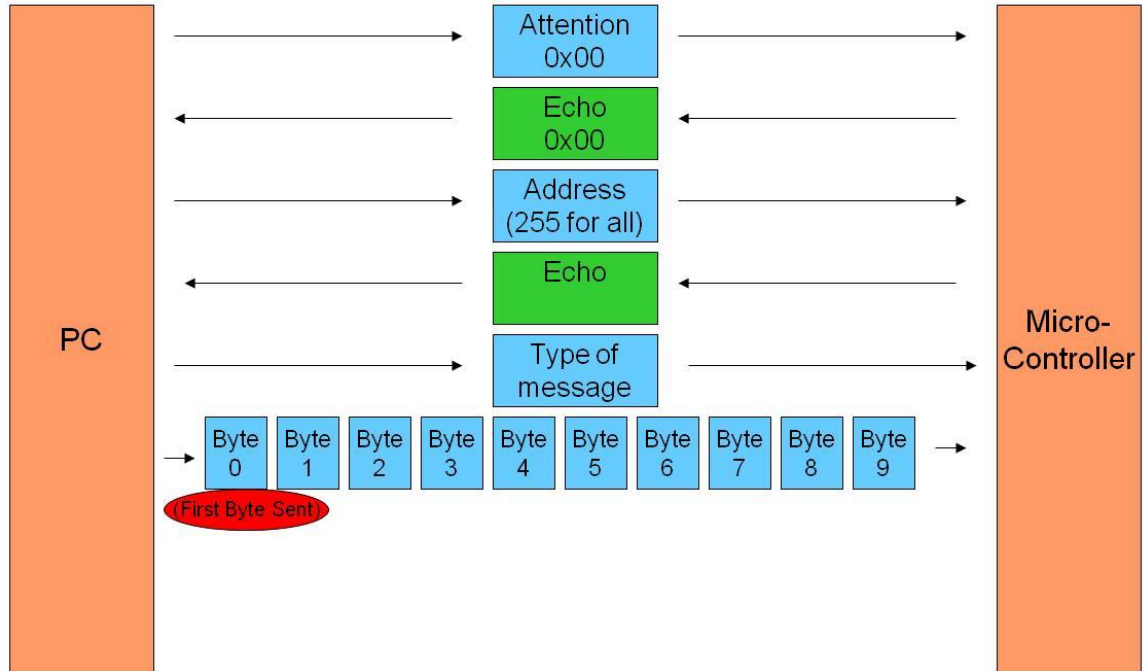


Figure 2.3.4.b. Communications protocol for USB link.

Types of Messages:	
C	Basic Command
1	Send new alarm times for alarms 1 and 2
2	Send new alarm times for alarms 3 and 4
3	Send new alarm times for alarms 5 and 6
4	Send new alarm times for alarms 7 and 8
N	Send a new name for a window
T	Send the system time
L	Request light readings and battery voltages
E	Erase all windows from the system

Figure 2.3.4.c. Communications protocol for USB link.

Type	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
c	cmd	0	0	0	0	0	0	0	0	0
1	type1	sec1	min1	hr1	day1	type2	sec2	min2	hr2	day2
2	type1	sec1	min1	hr1	day1	type2	sec2	min2	hr2	day2
3	type1	sec1	min1	hr1	day1	type2	sec2	min2	hr2	day2
4	type1	sec1	min1	hr1	day1	type2	sec2	min2	hr2	day2
t	sec	min	hour	day	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0

Cmd
0: Open
1: Middle
2: Close
3: Eco
4: Timer

Type:
0: Open
1: Middle
2: Close

Figure 2.3.4.d. Communications protocol for USB link.

Response to 'I' command

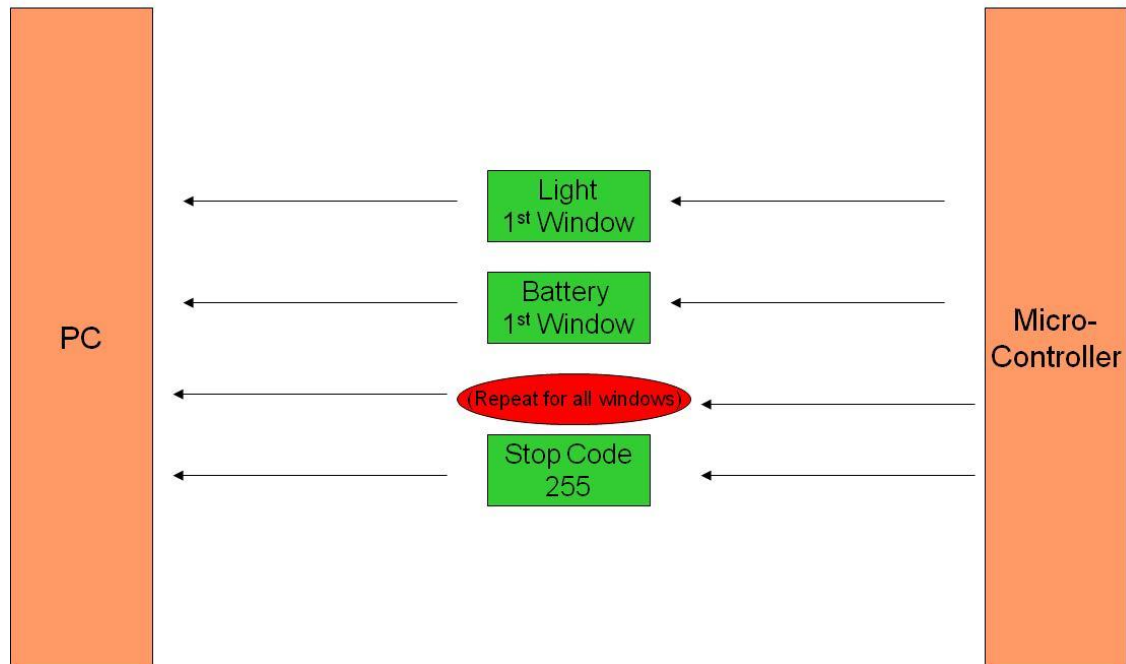


Figure 2.3.4.e. Communications protocol for USB link.

To test this USB protocol, a variety of bytes were sent from the microcontroller to a terminal. Since all of these bytes were received correctly, than microcontroller transmission and PC reception were verified. Then, a variety of bytes were sent from the PC program to a microcontroller and displayed on an LCD screen. Since these bytes were received correctly, the microcontroller reception and PC transmission were verified. Finally, the microcontroller and the PC program were connected together for two-way communication. Once successful, the USB protocol passed this tested this test.

2.3.3 Main Board

The purpose of the main board is to maintain USB communication with the PC and forward messages to the other units through ZigBee wireless communication.

Radio-frequency (RF) communication was chosen for our wireless link for a variety of reasons. RF communication is preferable to infrared communication because it is non-directional. RF communication is reliable and fast, at relatively low power levels. For our RF protocol, we chose ZigBee. The ZigBee standard is optimized for home automation

products, offering an acceptable range at low power levels. ZigBee devices are well understood and readily available.

The main board was designed as a printed circuit board using surface mount technology (SMT). This allowed us to concentrate a large number of parts in a well organized, small area. As described in Section 1 above, this main board has certain standard parts and unit-specific peripherals. The motivation and design of these standard parts and the peripherals are described below. The main board testing consists of testing each of the individual subsystems and interfaces between them. These test plans are also described below.

2.3.3.1 Microcontroller

The microcontroller used as the embedded intelligence for this project must have 33 I/O pins, a universal asynchronous receiver transmitter (UART), non-volatile memory, and capable of 3.3V operation. Since the microcontroller software is written with the BoostC compiler, a Microchip PIC18 model microcontroller was selected. The lowest cost PIC18 that meets our design requirements is the PIC18LF4620. Therefore, to minimize cost while meeting our requirements, the PIC18LF4620 was chosen as our microcontroller. **Figure 2.3.5** shows a schematic of this microcontroller with the pins connected to I/O signals. The naming conventions applied to these I/O signals will be applied throughout the document. The reasons for each pin assignment are explained in the following sections.

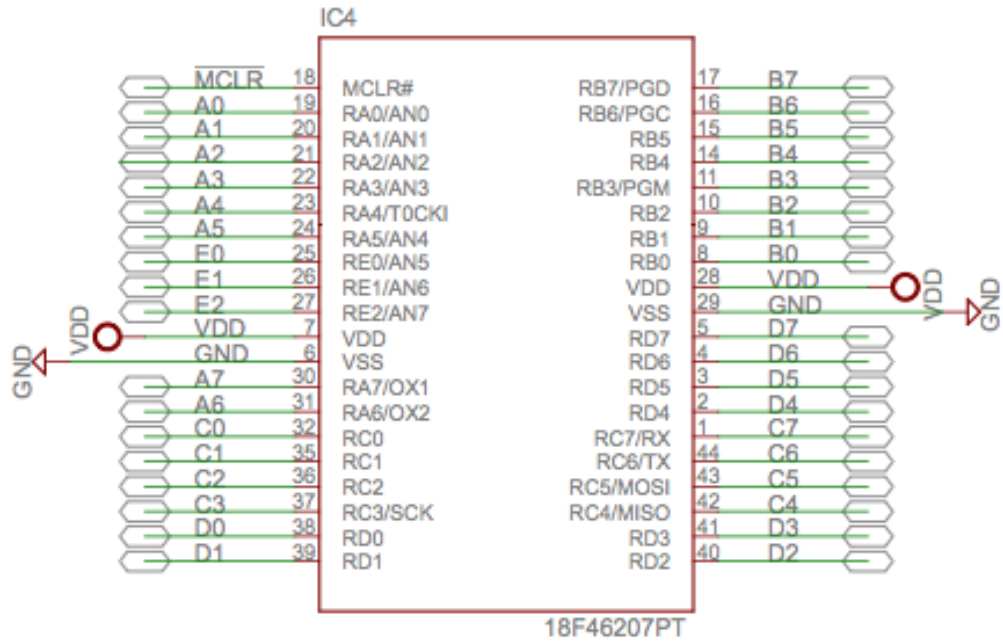


Figure 2.3.5. Microcontroller Schematic.

Since each subsystem relies on proper microcontroller operation, testing the operation of this microcontroller consisted of many implicit tests along the way. To specifically test the soldering connections of the microcontroller, the ports were configured as inputs and connected to V_{DD} and ground successively. An LCD screen was used to display the values on the ports and insure they are operating as expected. This method could only be applied to pins which had solid external pinouts. Other pins were verified by the simple fact that the subsystems connected to them were functioning.

4.3.3.2 Microcontroller Software

The design for the microcontroller software for the PCU is shown in **Figure 2.3.6**. The software consists of a main loop that listens ZigBee communication. USART communication with the PC is interrupt-driven. When a new USART message is received, the microcontroller is interrupted.

When a PC instruction is received, it is placed on a stack indicating which window the instruction is for. Once a communication link is achieved with that window, the instructions are removed from the stack and sent to the window. If the window requests to join a network, the PCU orchestrates this process and stores the window's address. Later, this connection can be reported to the PC application upon request.

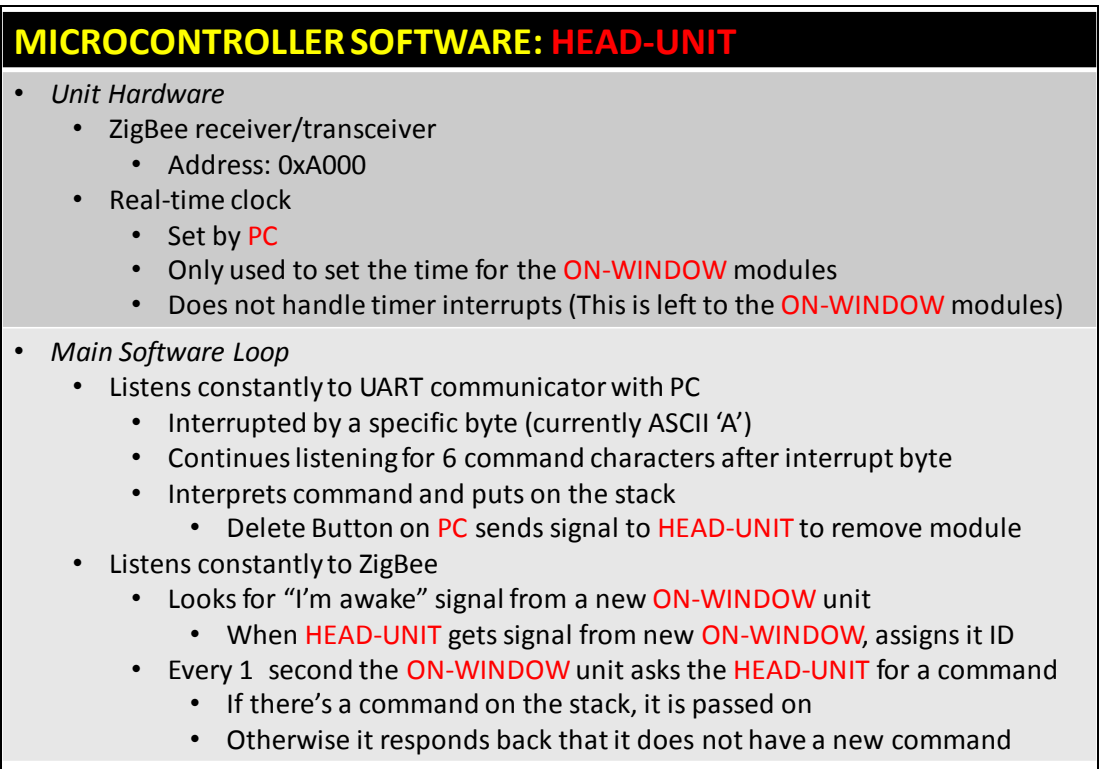


Figure 2.3.6. Microcontroller software design for PCU.

To test the microcontroller code, the code was run many times with a variety of inputs and scenarios. All attempts were made to create unusual situations for the program to handle. Program crashes were monitored and corrected.

The complete microcontroller code and libraries are included in the Appendix at the end of this document. The code is also listed in an executable form on the Smart Windows project website.

2.3.3.3 ZigBee Circuit

The ZigBee circuit must contain an integrated circuit capable of performing ZigBee radio frequency (RF) communication. This IC must be able to interface to the microcontroller through an SPI interface. To perform these functions, we chose the ATMEL ZigBee part AT86R231. The Atmel ZigBee transceiver part was chosen mainly out of familiarity. Senior design groups have worked with this part in the past to great success. We chose the part on the recommendation of past experience. The ZigBee circuit also contains an antenna, an oscillator, and associated impedance-matching traces. These components were chosen as per the instructions on the Atmel datasheet, shown in Appendix 6.4. These ZigBee circuit components were designed by Professor Mike Schafer of

the University of Notre Dame and are reproduced here exactly. The schematic for this ZigBee circuit is shown in **Figure 2.3.7**.

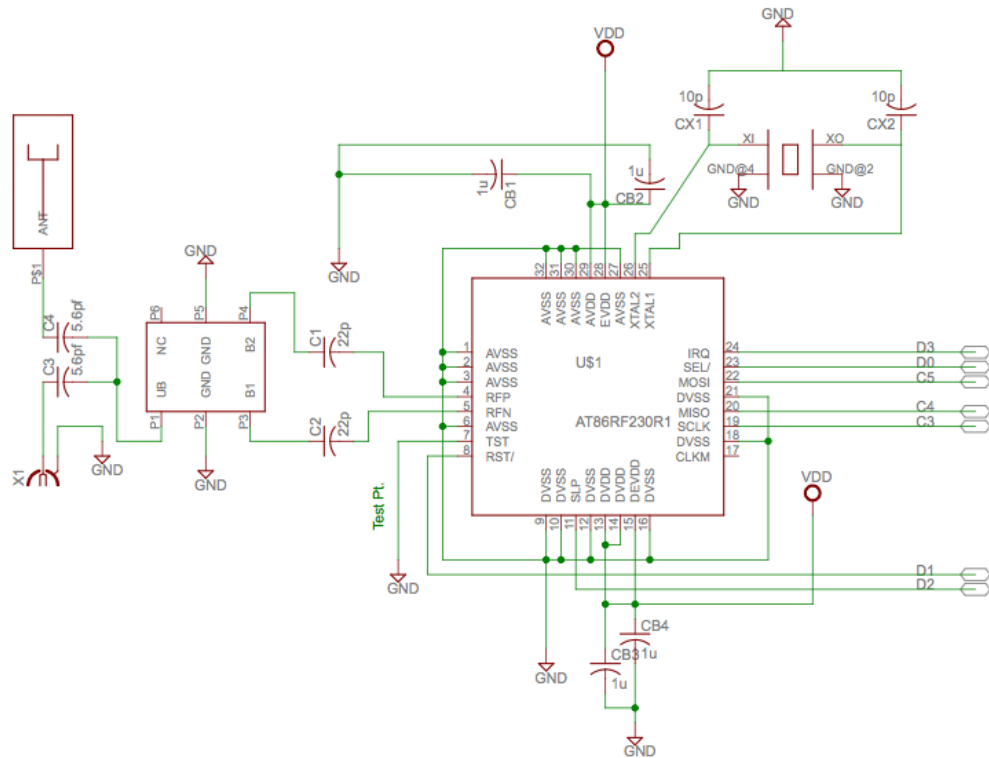


Figure 2.3.7. ZigBee Circuit Schematic.

To test this ZigBee circuit, a two-step process was used. First, the SPI interface was tested by writing and reading to registers on the ATMEL chip. Once this can was confirmed, the ZigBee transmission was tested. Messages were sent over ZigBee to a packet sniffer. This confirmed transmission. Then, packets were received from a packet sender to test reception. Finally, two ZigBee devices were connected in two-way communication. These tests were performed for each ZigBee transceiver involved in the project.

2.3.3.4 ZigBee Wireless Interface

The ZigBee circuit has two interfaces. The SPI interface connects the ATMEL ZigBee IC to the microcontroller. This interface is a standard protocol SPI interface. Standards describing SPI operation and the sending of bytes through SPI are readily available from a variety of sources. Because the ATMEL ZigBee transceiver demands an SPI interface, this decision was out of our control.

The ATMEL ZigBee IC communicates with the ZigBee units on other main boards through a ZigBee protocol RF wireless interface. The ZigBee

protocol we are using is compliant with the IEEE 802.15.4 standard. Our reasons for choosing this standard were laid out above. ZigBee units connected to the network are given a unique two-byte short address by the PCU. The ZigBee modules are set to extended mode, given them the capability for automatic address filtering, automatic message acknowledgement, and automatic retry until acknowledgment.

In any particular message, the first byte represents the type of instruction and subsequent bytes represent additional information associated with that instruction. In general, these messages follow the protocol set forward for USB communication set forth in **Figure 2.3.4** above. However, four additional message types are required for ZigBee communication that were not used for USB communication. The message protocol for these ZigBee-specific message types is laid out in **Figure 2.3.7a**.

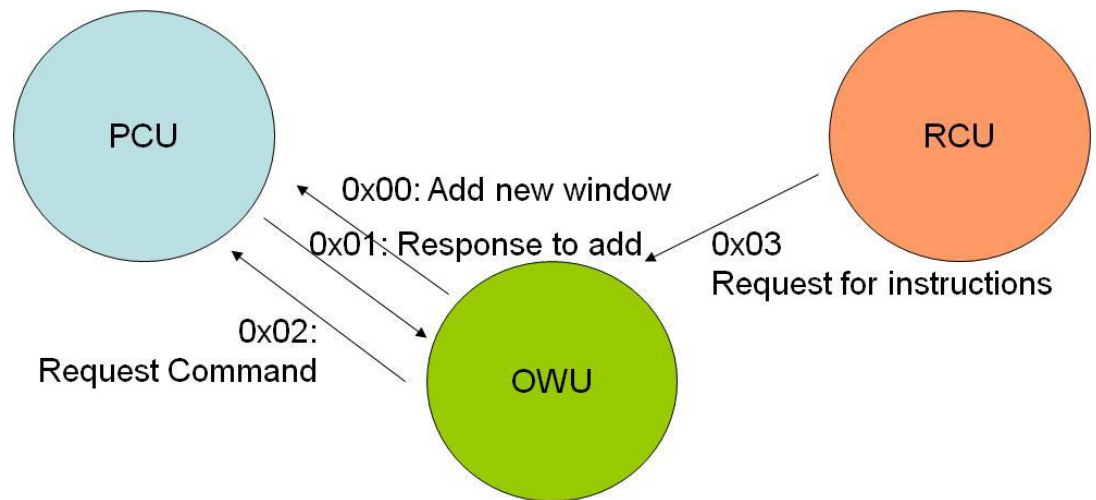


Figure 2.3.7a ZigBee-specific message protocol. The hex numbers listed replace the message type in the protocol of **Figure 2.3.4**. The first and second byte following the message type will contain the current light and battery readings, respectively. All other fields are set to zero.

The ZigBee interface was tested as part of the ZigBee circuit test in the previous section.

2.3.3.5 Ceramic Resonator

Our microcontroller requires a clock source, internal or external. The design of this clock source was subject to several constraints. To save power, our microcontrollers are set to sleep mode in between computation bursts. To decrease power usage, the time per instruction must be minimal so as to increase the sleep duty cycle. Therefore, our clock source must be as fast as possible, while maintaining reliable operation. To ensure efficient operation of the microcontroller, we have used an external

ceramic resonator providing a 20MHz clock to the microcontroller. A ceramic resonator was chosen over a crystal oscillator because of its superior simplicity. The appropriate oscillator is available from Murata through Digi-Key. The part number and ordering information are provided in the Bill of Materials shown in the Appendix. The oscillator circuit is shown in **Figure 2.3.8**. The circuit includes 2 30pF capacitors and a 1M ohm resistor. These elements are included for balancing should the resonator signal be unsatisfactory. In our implementation, the balancing resistor and capacitors were not needed.

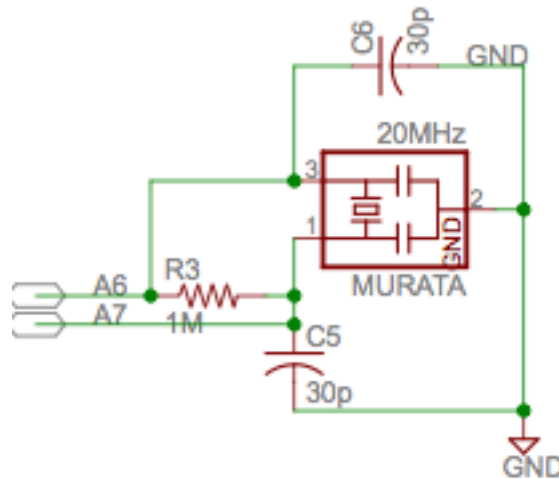


Figure 2.3.8. Oscillator Circuit. The balancing resistor and capacitors were unneeded in our implementation.

This circuit did not need to be tested explicitly since correct microcontroller operation indicated successful clock generation.

2.3.3.6 Programmer Circuit

The programmer circuit allows the Melabs programmer card to interface to our PIC microcontroller. The Melabs programmer was chosen for its availability and ease of use. This circuit was designed by Professor Mike Schafer of the University of Notre Dame and is included in **Figure 2.3.9** with permission. The diode and resistor are intended to protect the programmer from large reverse voltages during programming. The switch is included for resetting microcontroller operation after programming.

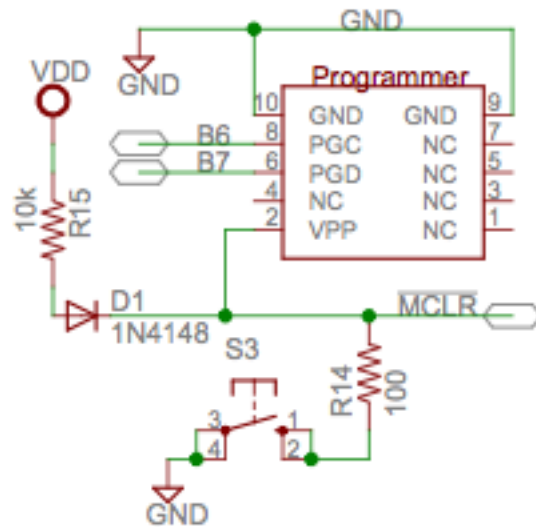


Figure 2.3.9. Programmer Circuit

This circuit did not need to be tested explicitly since correct microcontroller operation indicated successful programming operation.

2.3.3.7 Serial EEPROM

The main board must be able to maintain information about connected window units when power is disconnected. For this design, we have chosen EEPROM memory as our non-volatile memory source. Other forms of memory, such as SD cards, offer larger storage capacity at greater price and complexity. For our design, only 1000 Bytes of memory are required.

The PIC18 microcontroller has limited non-volatile EEPROM memory available. This memory met the needs of our project, and no external memory was required. However, external EEPROM memory in the form of a serial EEPROM chip was included in our design should future versions of the project require it. The serial EEPROM chip chosen is the Microchip 25LC640, which provides an additional 64kbits of memory. This chip was chosen because it is from a trusted manufacturer. It uses a standard SPI interface, which was already in use in other areas of our project. The circuit connecting this EEPROM chip to our microcontroller is shown in **Figure 2.3.10**.

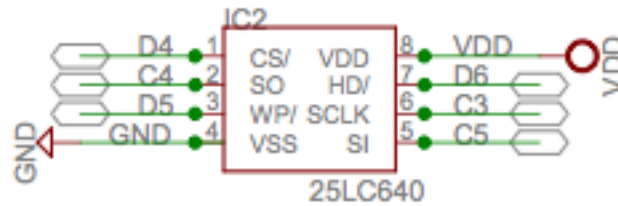


Figure 2.3.10. Serial EEPROM schematic.

As the EEPROM device was not needed in our final design, no EEPROM chips were ever ordered. Without EEPROM chips available, no testing was done on this circuit. However, the on-board microcontroller EEPROM was used in our final design and was tested extensively. Each test involved writing to and reading from the device a variety of times. Data was written to the device, and the power was shut off. The power was then turned back on later, first several minutes and then several days, and the data existence was confirmed.

2.3.3.8 Serial EEPROM SPI Interface

The serial EEPROM chip connects to the microcontroller through a standard protocol SPI interface. Standards describing SPI operation and the sending of bytes through SPI are readily available from a variety of sources. Since the EEPROM chip was never included in the final design, this SPI interface was never tested. However, other working SPI modules give us confidence that this SPI interface is correctly designed should it be needed in future designs.

2.3.3.9 Real Time Clock

When in automatic mode, the OWUs will be asked to open and close at certain times and days of the week. In order to do this, they will need an IC capable of keeping track of the current date and time. Since the PCU is responsible for syncing the time on the various windows to the PC clock, the PCU will also need time keeping abilities.

Devices capable of storing and incrementing the current real time are called real time clocks. For our design, we preferred a real time clock using an SPI interface, due to our familiarity with that interface. We also wanted a real time clock capable of storing alarm times and interrupting the microcontroller. In the final design this alarm feature was not utilized, although the circuitry is available for future designs.

As a result of these design specifications, we selected the Dallas DS1305 real time clock device since it met all of our requirements and offered an

appropriate surface mount package. The DS1305 connects to the microcontroller through an SPI interface.

Figure 2.3.11 shows the schematic that governs the real time clock operation. Pins 1 and 2 of this device are used for secondary power sources. Since none are available, these pins are tied low. This real time clock requires a 32.768 kHz crystal oscillator connected to pins 3 and 5. The SER3205 has been used for this purpose. The datasheet for this oscillator is included in the appendix. Pins 4, 6, 8, 13, and 19 of these device are not connected. Pins 7 and 9 represent the interrupt signals fed back to the microcontroller. They require external pull-up resistors shown. INT1 is unused. INT0 on pin 7 is attached to microcontroller port B5, since this port is capable of interrupt-on-change. Pin 10 is grounded.

Pin 11 of this device determines the mode of serial communication. It is pulled high to indicate SPI communication. The SPI signals on pins 12, 14, 15, and 16 are wired to the respective MSSP ports on the microcontroller as shown. D7 is chosen as the enable port for this device. Pin 17 is the logic-level supply pin and is given V_{DD} . Pin 18 offers an interrupt when power to the device fails. It is wired to port A3 for monitoring. Pin 20 is the power supply pin and is connected to the power trace on the board.

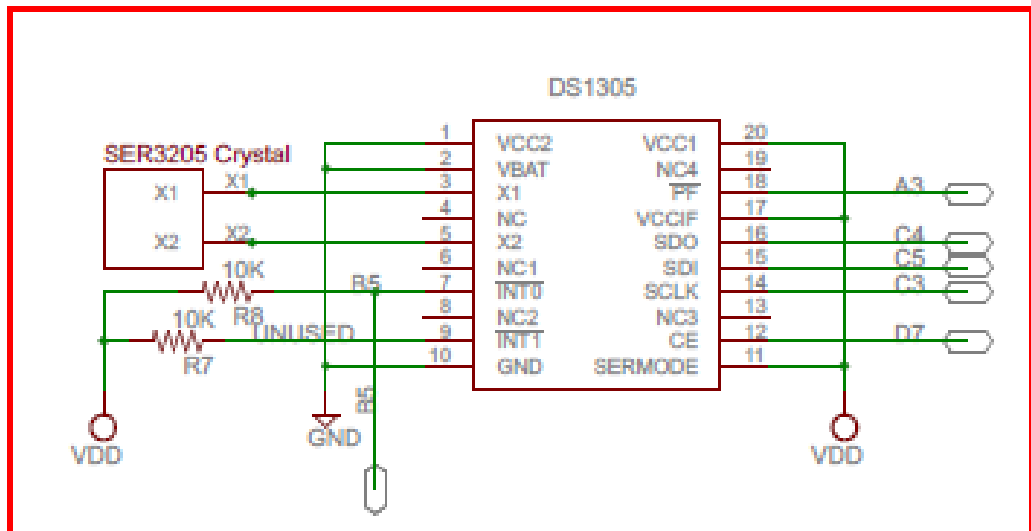


Figure 2.3.11. Real Time Clock Schematic.

The real time clock was tested by placing the current time on it through the SPI interface. Then, the device was allowed to run for a significant interval of time. Meanwhile, the elapsed time was measured with an external clock. After the interval of time has passed, the time was read from the device and compared to the known actual time.

2.3.3.10 Real Time Clock SPI Interface

The real time clock connects to the microcontroller through an SPI interface. Standards describing SPI operation and the sending of bytes through SPI are readily available from a variety of sources. The SPI interface will be tested as part of the real time clock device test listed above.

2.3.3.11 USB/UART FTDI Circuit

To convert the asynchronous serial messages transmitted by the microcontroller into the standard USB signals transmitted by the PC through the COM port, an intermediary is needed. In the first portion of senior design, the group gained experience using the FT232 for USB to serial conversion. This device is widely used for this purpose in industry, and its operation is well documented in online resources. For these reasons, we choose the FT232RL from FTDI. Professor Mike Schafer of the University of Notre Dame designed the circuit governing the operation of this device. The circuit is used with permission and shown in **Figure 2.3.12**.

USB signals enter the circuit through the USB connector shown. The bus voltage is filter using a ferrite bead and stabilized with capacitors as per the data sheet. For boards with no other power source, this USB voltage is regulated down and used as the power source as shown below. The data signals are passed through to the input of the FTDI chip. The outputs from pins 1 and 5 are attached to the USART port on the microcontroller port c as shown. The appropriate pins are driven high and low and shown on the FTDi datasheet included in the appendix.

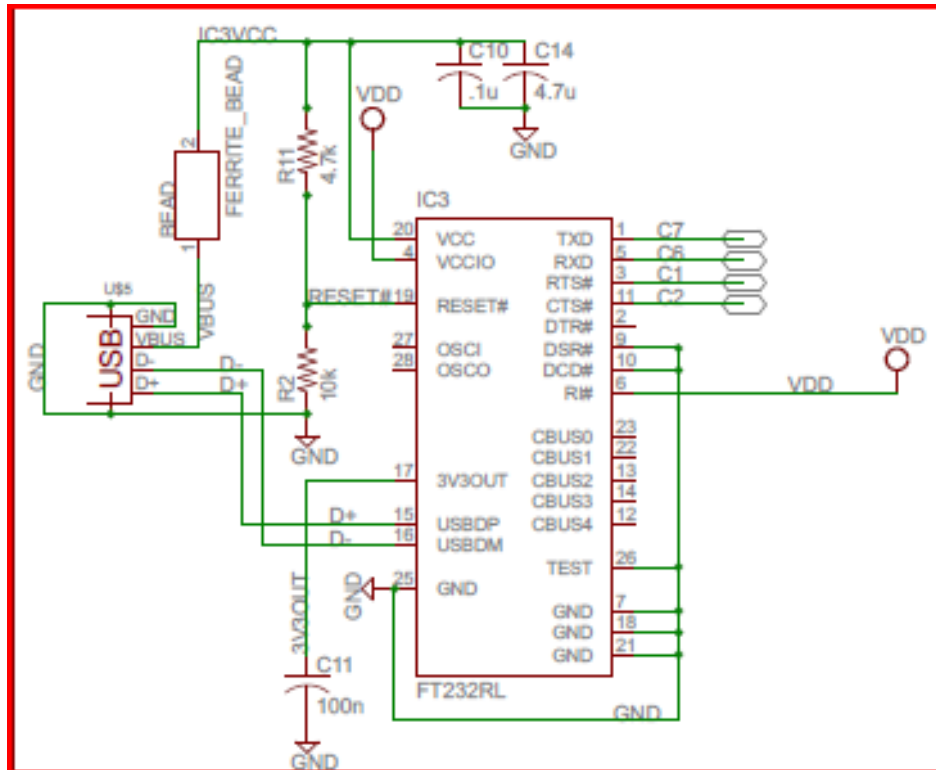


Figure 2.3.12. FT232RL schematic.

This USB signal from the PC also carries a 5V signal along with it. This 5V signal will be used to power the PCU main board since the main board of the PCU is not attached to a battery like other boards. This 5V voltage is run directly into a 3.3V regulator, as shown in **Figure 2.3.13**. For the regulator, the ZLD01117 was chosen. While a variety of regulators are available, we required one that would support input voltages anywhere between 5 and 15 volts. We also required the regulator to supply at least 125 mA of current and be available in-stock from Digi-Key. The only regulator meeting these constraints was the ZLD01117.

The capacitors in the regulator circuit are included to filter AC noise out of the power signal provided by the regulator. The values are those recommended by the data sheet included in the appendix. The switch is used to switch the input to the regulator from the USB bus voltage to the battery voltage provided on the DC input jack. The resistor network shown is not implemented on the PC board and will be explained in a later section.

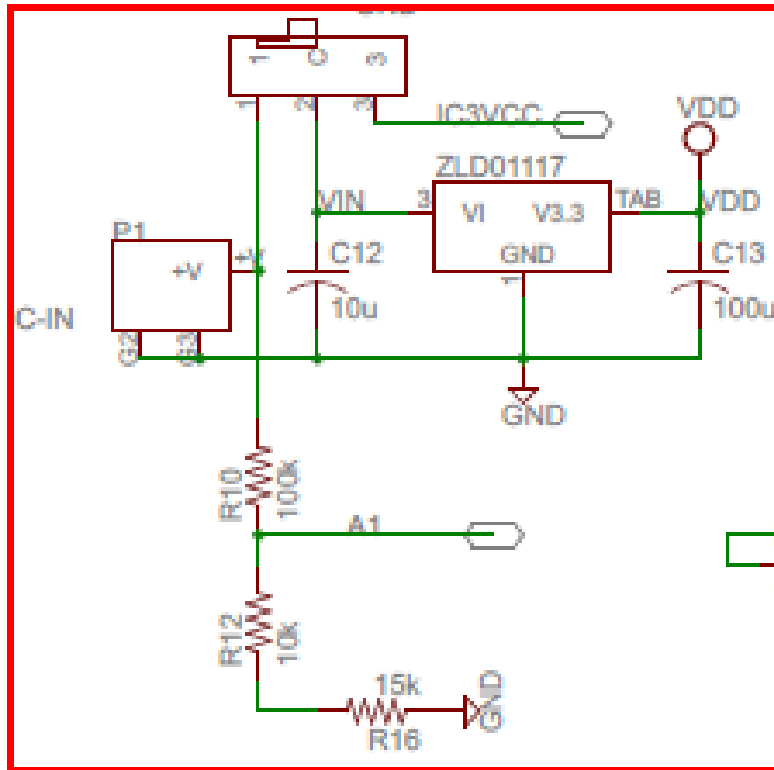


Figure 2.3.13. Regulator circuit.

For testing the communications properties of this circuit, bytes were sent and received to the PC. This process was part of the USB interface test already described above. To test the power generation portion of the circuit, we simply checked to make sure the board is receiving power using a voltmeter. At every step, an ohm meter was used to ensure that there was adequate resistance between the power and ground traces on the board.

2.4 On-Window Unit (OWU)

Each on-window subsystem needed to have two features: intelligence and control. The OWU needs the ability to control the window blinds through an appropriate actuator. Intelligence is necessary in order to make control decisions in real time, with or without direct user input.

For this purpose, we have divided the OWU into two main subsystems. The motor block contains all the necessary hardware to drive the window blinds. Its operation is described below. The main board contains the intelligence needed to control the motor block and maintain communication with the rest of the system.

The motor block interfaces to the main board in several ways. The motor block will use an h-bridge circuit (described below) to directly control the motor. This circuit will have two input signals fed to it from the microcontroller representing forward and reverse operation. Also, the main board will monitor the motor's progress through two limit switches and an optical photo-interrupter. These devices are described below.

2.4.1 Motor Block

The motor block consists of a DC motor, window blinds, two limit switches, a photo-interrupter, two ID hubs, a rubber spider coupling, and a steel rod.

The DC motor drives the window blinds, causing the shaft on the blinds to rotate. Many motors were considered for this project. Servo motors and stepper motors offer the greatest amount of control over motor rotation. DC motors with shaft encoders offer the ability to monitor motor rotation directly. However, the simplest and least expensive solution is a DC motor with no control device. For this purpose, we chose the SolarBotics G3 motor because of its built in protection clutch, low voltage operation, and geared down rotation.

However, it is imperative that we know how long the motor has rotated. For this, we used simple limit switches located at each end of the motor's rotation cycle. When the spinning steel rod strikes these switches, the microcontroller can receive a signal indicating that the motor has reached its bounds. This limit switch solution, however, is not able to alert the microcontroller when the motor has reached the middle of its rotation. For this, we mounted a photo-interrupter in the center of the rotation path. As the steel rod attached to the motor turns through this interrupter, the optical beam is broken and the interrupter sends a logic high to the microcontroller. The circuit running these limiting devices is shown below in **Figure 2.4.a**. The 10k-ohm resistors pull down the limit switches. The 47-ohm resistor limits the current to the photodiode in the photo-interrupter. The 1k-ohm resistor limits the current into the collector of the

Inputs to this circuit from the microcontroller consist of V_EN1 and V_EN2. When V_EN1 is driven high, the motor outputs drive the motor forward. When V_EN2 is driven low, the motor outputs drive the motor in reverse. The DC input jack provides ~10V power to the h-bridge and motor. The capacitor stabilizes these voltages to a DC level.

To test this circuit, the motor was operated several hundred times over the span of weeks. For a successful test the motor must start and stop exactly 100% of the time. No motor faults were tolerated in this design.

2.4.2 Main Board

The vast majority of the design and testing of this system is identical to Section 2.3.3. Some OWU-specific elements are added where necessary. The design and testing of these OWU-specific peripherals are described below.

2.4.2.1 Microcontroller

Identical to Section 2.3.3.1

2.4.2.2 Microcontroller Software

The microcontroller software operation has been described above. On each cycle, the microcontroller will issue an alert indicating that it is awake. Then it listens briefly for new instructions from the PCU. If instructions are received, they are decoded and handled appropriately. This interaction happens in accordance with the ZigBee protocol described above.

After talking to the PCU, the microcontroller will request instructions from the RCU. If instructions are received, they will be handled appropriately. If none are received, the software will move on with no change. This interaction happens in accordance with the ZigBee protocol described above.

After attempting both forms of ZigBee communication and polling the on-window buttons, the software will make mode-dependent decisions. In eco mode, the software polls the light levels and makes a decision. In timer mode, the software polls the real time clock and checks the value against the stored alarm times.

Following these steps, the microcontroller sleeps for a 1 second cycle before repeating. This sleep cycle increases the battery life of the system dramatically.

The design for the microcontroller software for the OWU is shown in **Figure 2.4.1**. The complete microcontroller code is listed in Appendix 6.2.1. The main program operates at a high level. Individual functions are handled by a variety of task-specific function libraries. These function libraries include the software to run the ZigBee, LCD, real time clock, ADCON, EEPROM, serial communication, and motor functions.

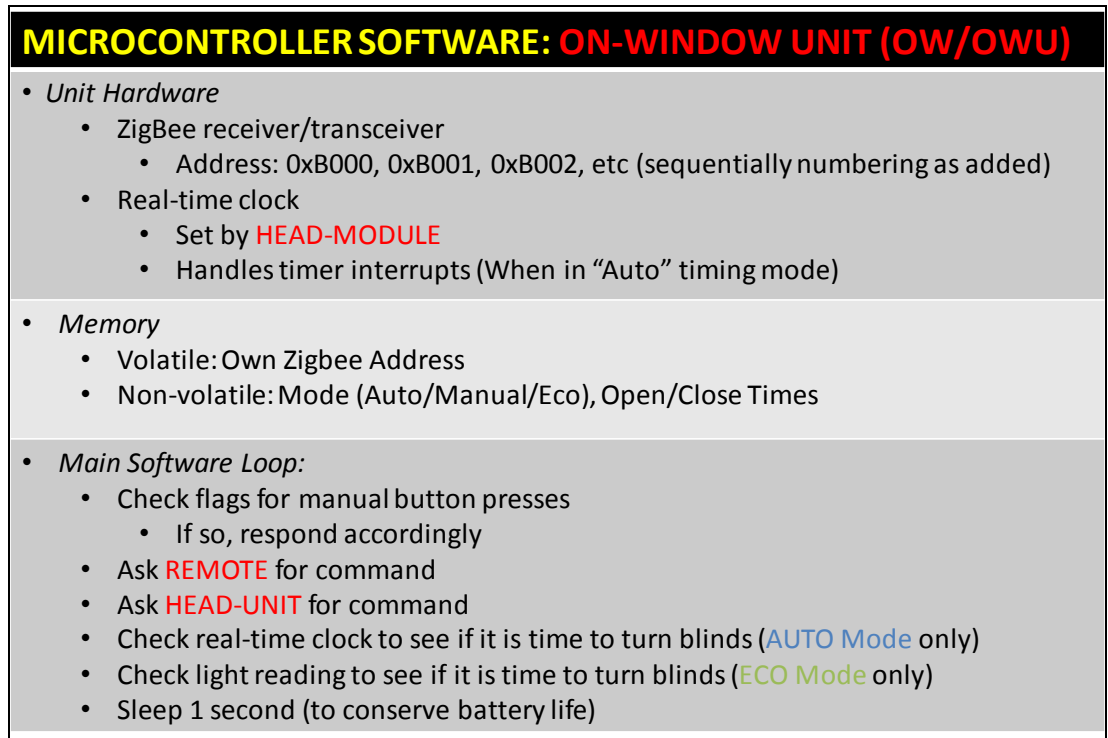


Figure 2.4.1. Microcontroller software design for OWU.

To test the microcontroller code, the code was be run many times with a variety of inputs and scenarios. All attempts will be made to create unusual situations for the program to handle. Program crashes will be monitored and corrected.

2.4.2.3 ZigBee Circuit

Identical to Section 2.3.3.3

2.4.2.4 ZigBee Wireless Interface

Identical to Section 2.3.3.4

2.4.2.5 Ceramic Resonator

Identical to Section 2.3.3.5

2.4.2.6 Programmer Circuit

Identical to Section 2.3.3.6

2.4.2.7 Serial EEPROM

Identical to Section 2.3.3.7

2.4.2.8 Serial EEPROM SPI Interface

Identical to Section 2.3.3.8

2.4.2.9 Real Time Clock

Identical to Section 2.3.3.9

2.4.2.10 Real Time Clock SPI Interface

Identical to Section 2.3.3.10

2.4.2.11 DC Input Circuit

Because there is no USB connection to the OWUs, they must derive their power from another source. Connecting each window to a wall outlet would use a disproportionate number of outlets. These units could be hardwired to the household power line, but that would require professional installation. Our design preference for Smart Windows is for simple do it yourself installation. Therefore, rechargeable batteries must power the system. While a solar charging circuit would be ideal, the complexity is too great for this first version design. Due to their combination of safety and longevity, we have chosen NiMH AA batteries. These batteries can be charged with any household charger. To reach the voltage required by the DC motor, we will need 8 batteries in series.

To regulate these voltages down to the 3.3V required by the main board, a voltage regulator will be used. The required regulator design has already been explained in Section 2.3.3.11 along with most of this power circuit.

This power circuit must be capable of monitoring battery voltage. To do this, we have chosen to run the battery voltage into the analog to digital converter present in the microcontroller. To monitor battery voltage, the battery voltage must be scaled down to a safe level before entering the microcontroller. At new battery life, battery voltages near 15 volts are possible. For safety, we have chosen a 5:1 resistor divider as shown below. When monitoring battery voltage, we have simply remember that

the true battery voltage is 5 times greater than the measured value. Port A1 is used to connect to the analog to digital converter.

This DC input circuit is shown in **Figure 2.4.2**.

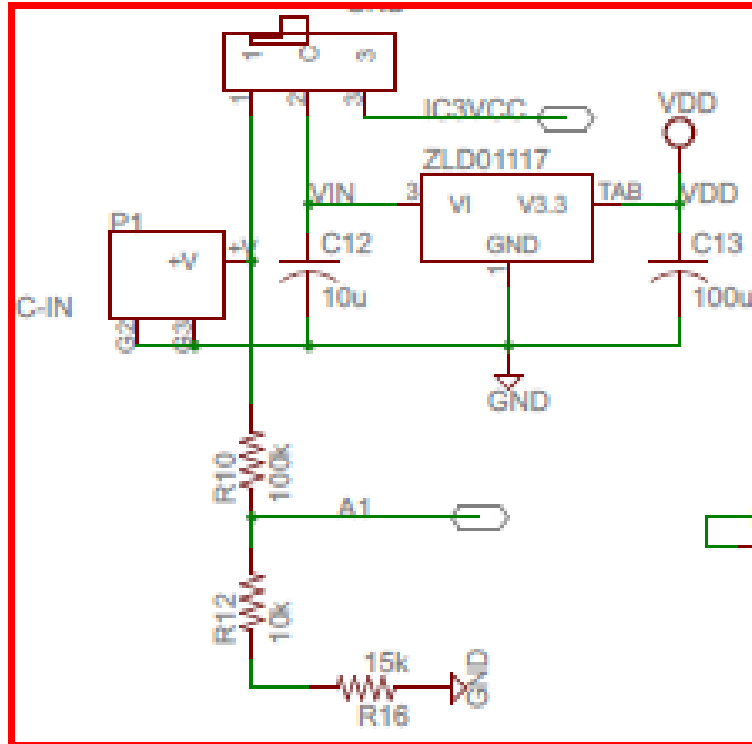


Figure 2.4.2. DC input circuit.

Testing the DC input circuit was straightforward. A voltmeter was used to insure that the board is receiving power. Then, the voltage measured by the analog to digital converter was scaled up by 5 and compared to this reading.

2.4.2.12 Light Sensor Board

When the OWU is in eco mode, it must be able to respond to changing light levels. Therefore, a small light sensing board has been designed. This board will be fixed to the window surface. Jumper wires connect this board to the main board.

The light sensor board contains a TEMT6000 phototransistor. This transistor uses light levels in place of its gate. We have chosen a simple phototransistor because of its size and simplicity. We want the light sensor to have a small size impact on the project. More complicated sensors are available, but they are larger and more costly. The voltage created by the light-gated phototransistor is read by the analog to digital converter using port A0 and reported to the user. This voltage is roughly

proportional to ambient light levels. The TEMT6000 was chosen as the most widely used ambient light phototransistor.

The fully designed light sensor board is shown **Figure 2.4.3**. The jumper on the main board that connects to the light sensor board is shown in **Figure 2.4.4**.

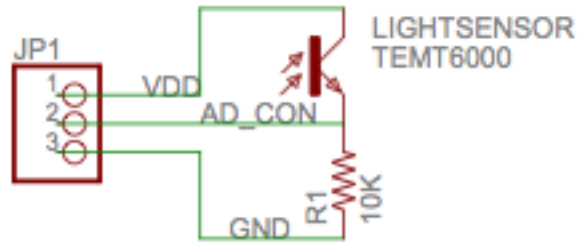


Figure 2.4.3. Light sensor board.

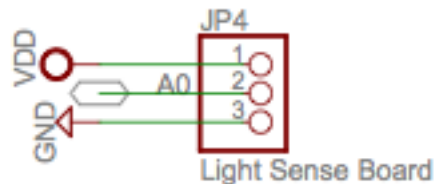


Figure 2.4.4. Main board light sensor jumper.

To test the light sensor board, the voltage output was converted to a digital signal using the A/D converter. This digital reading was displayed on the LCD screen. The light levels on the sensor were varied by changing the room lighting conditions. We verified that the light readings on the screen agreed with the light levels on the sensor.

2.4.2.13 Button Card

The OWU needs three buttons. One button will be used to connect the OWU to the base station. The other two buttons will be used open and close the window blinds. Since these buttons must be in a place accessible to the user, they cannot be attached to the main board. Instead, the buttons are placed on the panel of the project box containing the OWU main board. A wiring harness connects these buttons to their pull-down resistors on the board. This card supports up to 5 buttons, but only three will be needed for the OWU. **Figure 2.4.6** contains the main board jumper to which this button card can be attached. 10-kohms was chosen as an ordinary value for a pull-down resistor.

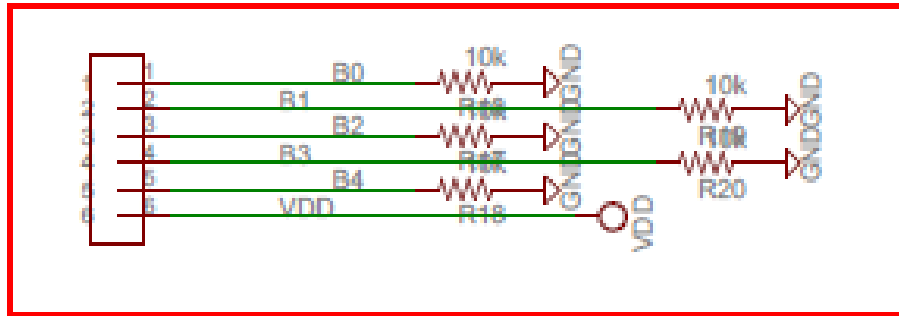


Figure 2.4.7. Button Card Main Board Jumper.

To test the button card, we will display the value on the Port B on the LCD screen. We will press various combinations of the buttons to ensure that they all work as expected.

2.4.2.14 DC/DC Converter

The motor control board requires a 5V signal. To convert the main board 3.3V signal to a 5.0V signal, a DC-DC converter device is necessary. A suitable device is the MCP1252 charge pump. This device was chosen because it is produced by a trusted manufacturer and has worked acceptably on previous projects. The LCD screen and motor logic will draw current from this device. Both devices were tested to insure that they did not draw more than the maximum allowed current of the charge pump.

The schematic governing the operation of this device was designed by Professor Mike Schafer at the University of Notre Dame and is reproduced here with permission as **Figure 2.4.8.**

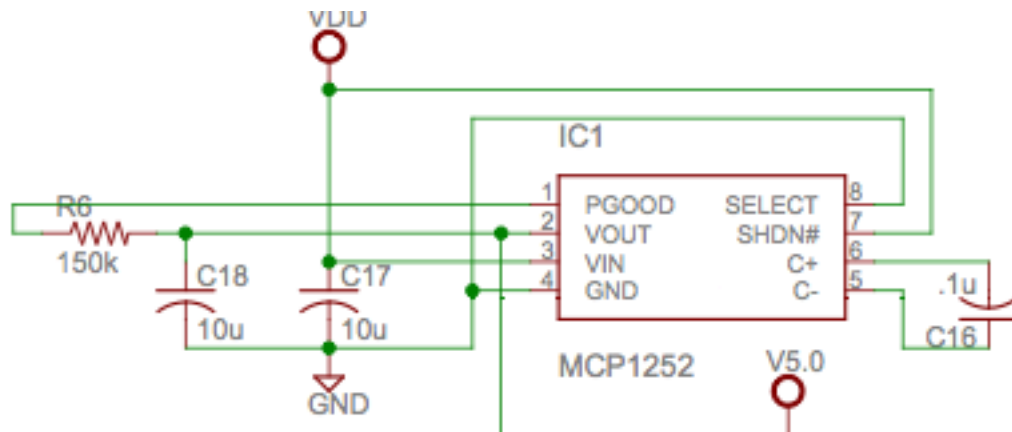


Figure 2.4.8. DC/DC converter circuit.

To test the charge pump DC/DC converter, a voltmeter was used to make sure the appropriate 5V signal is being generated.

2.4.2.15 Motor Control Board

To control the DC motor from the microcontroller, an h-bridge and related protection circuitry are needed. The h-bridge converts logic outputs from the microcontroller into the voltages needed to drive a motor. These choice of this h-bridge design has been described above. Because the motor deals with high voltage signals that are dangerous to the microcontroller, this motor driving circuit is placed on a physically separate board.

Figure 2.4.9 shows the motor board. The protection circuitry around the h-bridge was designed using the datasheet for the LM298. The jumper connection that interfaces to the motor control board is shown in **Figure 2.4.10**.

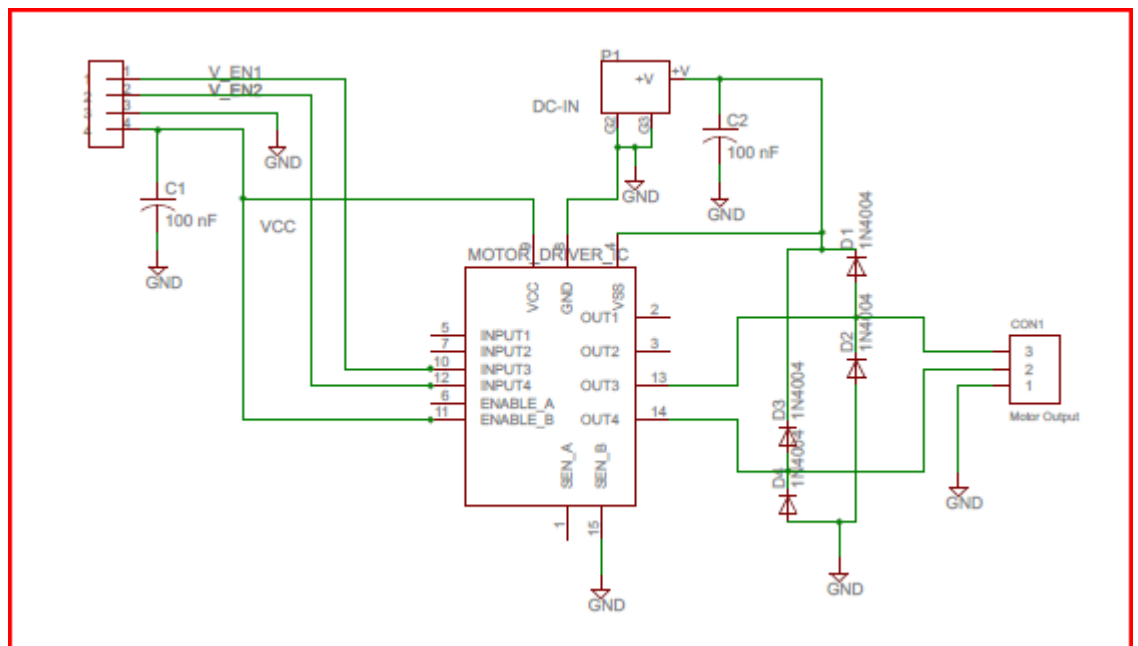


Figure 2.4.9. Motor control board.

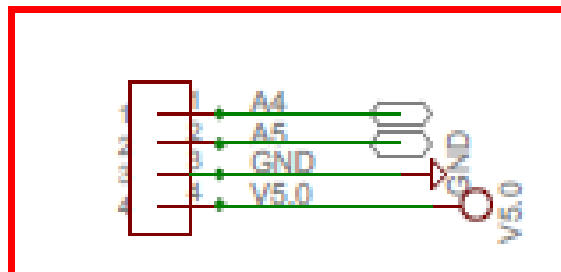


Figure 2.4.10. Motor control board to main board jumper connection.

No specific test was needed for the h-bridge circuit since the operation of the motor is being tested above in the motor block test. If the motor block is able to respond appropriately with certainty to microcontroller signals, it can be assumed that the driver board is working correctly.

2.4.2.16 Limit Switch Jumper Circuit

When operating the motor, a limit switch is depressed when the motor becomes fully opened or fully closed. The microcontroller will read this voltage to control motor movement. Therefore, pull-down resistors are necessary to create a working switch circuit. The necessary resistors are included on the main board and connected to the switches through a jumper. The circuit is shown in **Figure 2.4.11**.

As described above, the microcontroller must also sense when the motor has reached its half opened state. To do this, we have chosen a photointerrupter, since it does not impede the path of the motor. For this design, we chose the Sharp GP1S52. Our specific photointerrupter was chosen because it was produced by a trusted manufacturer and had an appropriate sized slit.

The photointerrupter has a photodiode to produce a beam of light and a phototransistor to receive it. The photodiode needs a current-limiting resistor, and the phototransistor needs a collector resistance. These resistors values were chosen according to the data sheet shown in the Appendix.

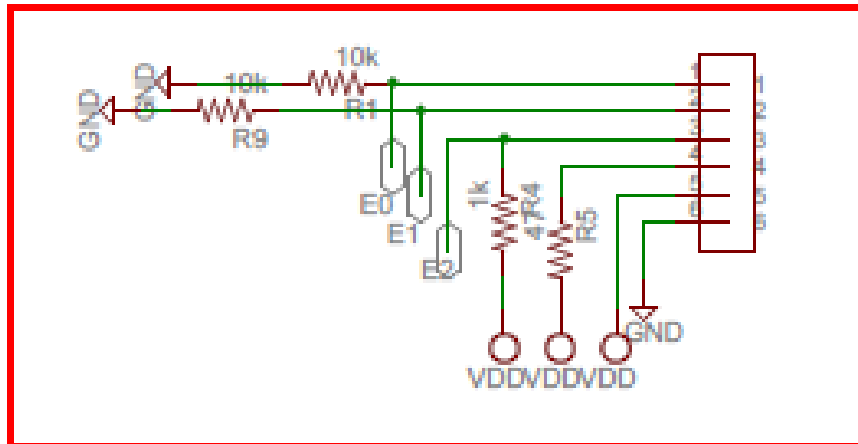


Figure 2.4.11. Limit switch jumper circuit.

No specific test was needed for the limit switch circuit since the operation of the motor is being tested above in the motor block test. If the motor

block is able to stop appropriately when it reaches the limits of its motion, it can be assumed that the limit switch circuit is working correctly. However, in the event of limit switch malfunction, a voltage meter could have been used to insure proper switch operation.

2.5 Remote Control (RCU)

The user may choose to control his or her Smart Windows without the use of the PC. It would be inconvenient for the user to have to use the on-window buttons in this event. Instead, our design includes a wireless remote control subsystem. This system interfaces to the rest of the system using ZigBee, as previously described. As was the case before, the remote control uses a main board with certain RCU-specific peripherals as described below.

2.5.1 Main Board

The vast majority of the design and testing of this system is identical to Section 2.3.3. Some RCU-specific elements are added where necessary. The design and testing of these RCU-specific peripherals are described below.

2.5.1.1 Microcontroller

Identical to Section 2.3.3.1

2.5.1.2 Microcontroller Software

The design for the microcontroller software for the RCU is shown in **Figure 2.4.3**. Since no on or off button was included for the remote control, it must be capable of putting itself to sleep during periods of inactivity. This is handled through a 1 Hz interrupt and counter. The counter is incremented once per second in software. After 180 counts, or 3 minutes, the microcontroller goes to sleep until a button is pressed. Any time a button is pressed, the counter is reset to zero. In this way, the remote control shuts itself off after three minutes of inactivity.

Specific ZigBee messages are sent out in response to a request from the OWU, as described in the ZigBee protocol above. The software listens for requests and responds appropriately once per cycle. At the end of the cycle, the software puts the ZigBee to sleep for 1 second in order to save battery life. Button presses are stored on a stack until they are requested by the appropriate OWU.

MICROCONTROLLER SOFTWARE: REMOTE ("RC")	
• <i>Unit Hardware</i>	<ul style="list-style-type: none">• ZigBee receiver/transceiver<ul style="list-style-type: none">• Address: 0xC000• Only one REMOTE unit will be supported under our implementation
• <i>Memory</i>	<ul style="list-style-type: none">• Volatile<ul style="list-style-type: none">• List of window names with corresponding ID's
• <i>Main Software Loop:</i>	<ul style="list-style-type: none">• Update LCD with currently selected window (if changed)• Look for "Open window" or "Close window" button flag<ul style="list-style-type: none">• Send corresponding open or close command to HEAD-MODULE (to be forwarded to ON-WINDOW)• Look for "Show next window (up)" or "Next Window (down)" button presses<ul style="list-style-type: none">• Update which window is to be displayed, based on internal list• Every 5 seconds receive new list of ON-WINDOW modules

Figure 2.5.1 Microcontroller software design for RCU

To test the microcontroller code, the code will be run many times with a variety of inputs and scenarios. All attempts were made to create unusual situations for the program to handle. Program crashes were monitored and corrected.

2.5.1.3 ZigBee Circuit

Identical to Section 2.3.3.3

2.5.1.4 ZigBee Wireless Interface

Identical to Section 2.3.3.4

2.5.1.5 Ceramic Resonator

Identical to Section 2.3.3.5

2.5.1.6 Programmer Circuit

Identical to Section 2.3.3.6

2.5.1.7 Serial EEPROM

Identical to Section 2.3.3.7

2.5.1.8 Serial EEPROM SPI Interface

Identical to Section 2.3.3.8

2.5.1.9 DC Input Circuit

Identical to Section 2.4.2.11

2.5.1.10 Button Card

Identical to Section 2.4.2.13

2.5.1.11 DC/DC Converter

Identical to Section 2.4.2.14

2.5.1.12 LCD Screen

The remote control can only address one OWU at a time. To display the currently addressed window to the user, an LCD screen was included on the remote control. For this design, we required an LCD screen capable of making SPI communication, the standard for this project, and capable of fitting onto the panel of our remote control.

For this purpose have chosen the 2x16 character display offered by New Haven Displays. This display uses a one-directional SPI interface to display characters on the screen. **Figure 2.5.2** shows the schematic for the main board jumper that enables connection to the LCD screen. The LCD screen requires a 5V signal for power. This signal is obtained from the DC/DC converter in Section 2.4.1.11.

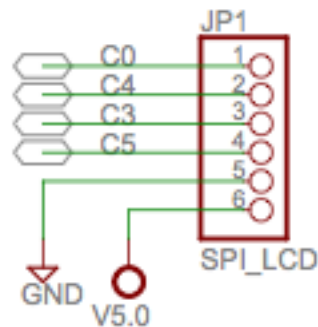


Figure 2.5.2. LCD screen jumper.

To test the LCD screen, a variety of characters were sent to the screen through the SPI interface. Once these characters were displayed correctly, we sent more complicated commands such as clear the screen and advance one line. Once these also worked correctly, the LCD had passed the test.

2.5.1.13 LCD Screen SPI Interface

The SPI screen communicates with the microcontroller through a one-way SPI protocol. This protocol operates in the same manner as the standard SPI protocol available in a variety of places. The difference is that for one-way communication the slave (the LCD screen) does not return a byte after the master sends one. Instead, the master sends bytes corresponding to the ASCII codes to display on the screen. The SPI interface was tested with the LCD screen test listed above.

3 System Integration Testing

There were three main forms of software system integration required by this project. First the various ZigBee-enabled main board units had to make reliable communication with each other over ZigBee. Each command of the ZigBee protocol described above needed to be tested. Second, the USB-enabled PCU had to make reliable USB communication with the PC application. Each command of the USB protocol described above needed to be tested. Finally, the Android mobile phone application had to connect to the PC application through the Notre Dame Android server.

Separate hardware system integration testing was not necessary for two reasons. First, the subsystem testing described in Section 2 included tests of each hardware interface. Since each hardware interface was already tested, it has been proved that the hardware is integrated. Second, the software integration testing described here was completed by sending messages and enacting the instructions described by those messages. This process implicitly tests each piece of necessary hardware as individual messages and commands are carried out by the hardware systems.

Therefore, our software testing was grouped into the following categories:

- ❖ Pure PC-side testing
- ❖ Pure Microcontroller-side testing
- ❖ PC-Microcontroller interactions

Android testing was included in the pure PC-side testing as it measures the Android's ability to communicate instructions only to the PC. The PC's ability to forward these messages to individual OWU's is tested elsewhere.

ZigBee testing was accomplished in the pure-microcontroller-side testing. Here, we measured one microcontroller's ability to reliably send messages to other microcontroller units.

In each of these 3 testing categories, general testing paradigms were applied. These principles are described in Section 3.1.

3.1 General Software Testing Paradigms

The main testing paradigm applied to this project was repeatability. One successful test was never taken to mean a successful interface. Instead, each test was repeated many times with different variations. Other important paradigms are summarized here:

- ❖ Deterministic (or "fixed") commands
 - Deterministic commands were tested using comprehensive testing, since doing so was easily possible. Every possible command was sent

to the microcontroller/PC and we ensured each command was successfully executed.

- Examples:
 - “Change mode to ECO”
 - “Open window”
 - “Delete a window”
- ❖ Highly variable (or “infinitely variable”) commands.
 - Commands such as the custom timing for the “Timing Mode” of our windows, was customizable to the point of being nearly infinitely variable
 - Thus, we had to settle for systematic testing as extensively as our time constraints allowed. For instance, carefully verifying that all eight open/close times on our form were programmatically identical, we entered extensive days and times and took note of the output bytes generated.
- ❖ Well accepted packages:
 - By using certain built-in objects, such as the python time-entry widget, we were confident that it would not be possible to enter in unintelligible times (such as 25:61 o’clock), as many other programmers and testers have verified this is not possible with the widget. Whenever possible we used reliable blocks such as this in our design.
 - Another example of this was our USB communication module which had been developed by other programmers and tested extensively before us

Overall, a combination of clean and well-regarded programming practices, which we carefully checked over, and extensive testing of major cases is what we deemed sufficient in our case. Since we are not dealing with dangerous equipment or huge financial transactions, minor flaws are, although unfortunate, not disastrous. Writing computer testing routines was not feasible in our timeframe, or, we felt, necessary for a non-safety related application. Having a completely separate testing team unrelated to our design team was not possible.

Ideas for future improvements in the software testing progress:

- ❖ Separate testing team:
 - For every “programmer” there will be an “antigrammer”
- ❖ Code reviews:
 - Putting all, or nearly all, of our code on a large screen or projector
 - We can then critique and verify each other’s code
- ❖ Pair programming:

- Statistics show that 2 programmers programming together on one computer program at 70% of the speed with 70% fewer errors

3.2 Pure PC Software Testing

The PC software was tested extensively, both through specific testing and by general run time observation. Some examples of this pure PC software testing are provided here. Examples of pure PC-side testing include:

- ❖ Verifying that the various date and time selections matched up to the correct bytes
 - Since the program visually outputs the byte codes to the right of the date and time selection, this part can be verified before without the microcontroller.
- ❖ Verifying that saved window names and times save to a file, and load from a file, correctly.
- ❖ Verifying routing of clicks and graphical confirmation.
- ❖ Most of GUI minus back-end USB communication
- ❖ Getting the current time
- ❖ Formatting the time to “binary coded decimal”

3.3 Pure Microcontroller Software Testing

The microcontroller software was tested in two steps. First each submodule of the main board was tested without the use of ZigBee to insure proper operation. Then, messages were sent from one board to another of ZigBee, while being monitored by a ZigBee packet sniffer. The team watched the messages closely to be sure that each submodule was able to receive the correct instructions reliably. Examples of pure Microcontroller testing include:

- ❖ Sending the time from the head module to the secondary modules
- ❖ Sending light readings from the secondary modules to the head module
- ❖ Sending battery readings from the secondary module to the head module
- ❖ Verifying that the window responds to changes in sunlight while in Eco mode

3.4 PC-Microcontroller Interaction Testing

The PC-microcontroller interface was the most difficult testing we performed. The microcontroller hardware, the pc software, and the microcontroller software were all involved in the successful operation of this test. First, the USB testing described in Section 2 above was applied to ensure a reliable USB link between the two subsystems. Then, each of the protocol-described message formats were tested one at a time, sticking to the testing paradigms described above. Examples of PC-Microcontroller interactions that were tested include:

- ❖ Receiving light and battery readings back from the head module

- ❖ Sending the current time from the PC to the head module and verifying that it is correct
- ❖ Sending mode-change commands from the PC to the head module, and having them correctly forwarded on and executed.

4 Installation Manual and User's Guide

Thank you for purchasing a Smart Windows system. If you have purchased a Smart Windows Starter Kit, you have everything you need to install and control one Smart Window in your home. If you want additional windows, please purchase a Smart Windows Expansion Pack. Your new product will:

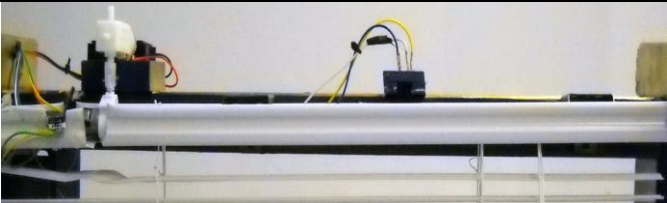
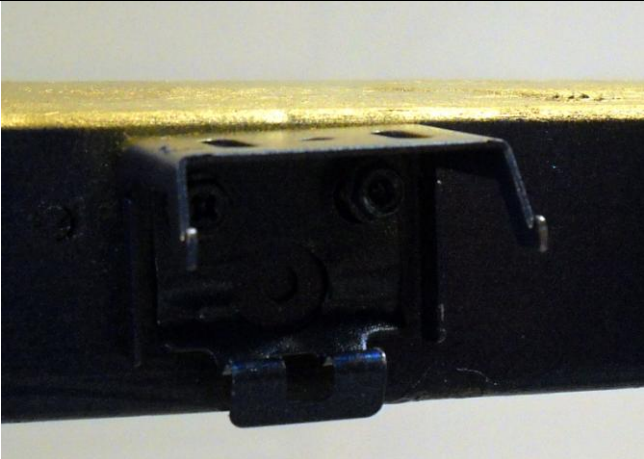
- allow you to open and close your window blinds at the touch of a button,
- control your window treatments with a wireless remote control,
- and coordinate all your windows with a PC application.







If you have purchased a Smart Windows expansion pack, your new product includes everything you need to setup and connect an additional Smart Window to your current Smart Windows installation.


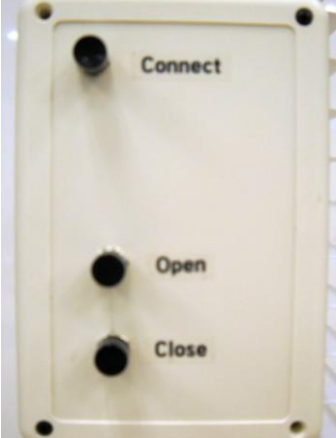


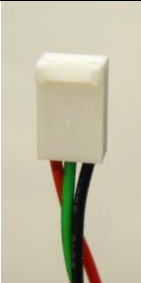
4.1 Installation Guide





Table 4.1.1 shows the contents of your new Smart Windows Starter Kit. Smart Windows Expansion Packs contain only those elements listed in the “**Window Unit**” section of **Table 4.1.1**. Before beginning, please ensure that these elements were included in your Smart Windows kit.

Table 4.1.1 Smart Windows contents

Smart Windows Contents		
Part	#	Picture
Window Unit		
Window blind assembly	1	
Blinds mounting bracket	2	
Wood Screws	10	

Motor connector	1	
Window assembly connector	1	
Battery Pack	1	
Rechargeable AA battery	8	
Battery mounting bracket	3	
Power cable	1	

Picture hanger	2	
Nail	2	
Window control box	1	
Motor control box	1	
Light sensor	1	
Light sensor connector	1	
Remote Control		

Wireless Remote Control	1	
9V Battery	1	
PC Unit		
PC Communication Box	1	
5-pin Full-Size A-to-B USB Cable	1	

In order to complete the assembly, you will need the tools and supplies shown in **Table 4.1.2**. Please gather these items before beginning assembly.

Table 4.1.2 Smart Windows required tools and supplies

Smart Windows Contents	
Tools	Supplies
Hammer	Hot glue, double-sided tape, or other adhesive
Phillips Screwdriver	
Personal Computer (optional)	
NiMH Battery Charger	

4.1.1 PC Unit Setup (*optional*)

Note: If you do not have a personal computer or choose not to install the PC-based portion of this product, your windows will still be operational. However, the available features will be greatly reduced. Installing the PC portion of this product is highly recommended.

(1) Connect the hardware: Remove the communication box (as shown in **Table 4.1.1**) from its packaging. Using the included A-to-B USB cable (as shown in **Table 4.1.1**), plug the communication box into an available USB port on your windows-based personal computer. Your computer should recognize the connection of an “FTDI USB to serial converter.” You may be prompted to install a driver for this device. Follow the on-screen instructions.

(2) Install the software: Insert the Smart Windows CD-ROM into your personal computer. A dialog box will appear prompting you to install the Smart Windows software. Please follow on-screen installation instructions.

(3) Start the software: At the conclusion of the installation, a Smart Windows icon will be placed on your desktop. Use this icon to start the Smart Windows software application. At startup, the software will search for connected windows. If no window units have been installed yet, the software will alert you that there are no connected windows.

(4) How to use the software: Now that you have installed your PC software, you can use it only when you need it. Individual windows will operate with or without the software. However, new windows will not be able to be added to the system if the PC communication box is not plugged into a USB port on your computer. The software does not have to be running to install new windows.

4.1.2 Window Unit Setup

(1) Install the window blind assembly: Smart Windows window blinds are installed like any ordinary window blind. Use a Phillips screw driver and wood screw to mount the two mounting brackets (as shown in **Table 4.1.1**) to the top of your window frame as shown in **Figure 4.1.2a**. Then, using firm pressure, lock the window blind assembly into place in the mounting brackets. Be sure that the open side of the blind assembly is facing up.

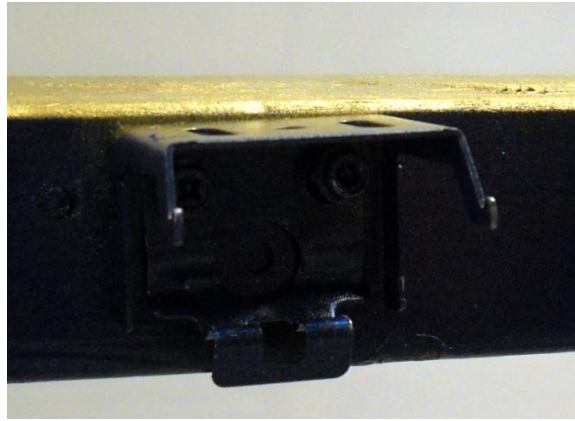


Figure 4.1.2a Proper installation of the window blind assembly mounting brackets

(2) Install the batteries: Smart Windows blinds operate on eight rechargeable AA batteries (included). Remove the batteries from their packaging and charge them using a NiMH AA battery charger (available wherever batteries are sold). Once the batteries are charged, insert them into the provided battery pack (as shown in **Table 4.1.1**). Be sure to insert the batteries with the correct polarity.

(3) Install the battery mounting assembly: To hold your batteries, 3 battery mounting brackets have been provided (as shown in **Table 4.1.1**). Using a Phillips screwdriver and wood screws, install these brackets near the top of your window frame in such a way as to hold your battery pack securely, as shown in **Figure 4.1.2b**. Consider installing your battery pack in a location where it will not be visible, such as behind the valence above your window frame. Slide your battery pack into the assembled battery mounting frame.



Figure 4.1.2b Proper installation on the battery mounting brackets.

(4) Install the window control box: The window control box contains the buttons you will use to control your window blinds directly. Choose a spot for this box near your window where it will be visible but non-intrusive. The control box will be hung from your wall or window frame using picture hangers (as shown in **Table 4.1.1**). Using a hammer and nails, fix the picture hangers to the wall. Be sure that the picture hangers are spaced in such a way that they will fit into the holes on the back of the control box. Hang the control box from the picture hangers.

(5) Mount the light sensor: Smart Windows uses a light sensor to monitor the sunlight outside your window. Use hot glue, double-stick tape, or other adhesive to mount this light sensor (as shown in **Table 4.1.1**) to the surface of your window. Be sure that the light-sensing side of the sensor board is facing out of your home, as shown in **Figure 4.1.2c**. Using the three-wire light sensor connector, connect the light sensor to the port on the top of the window control box.

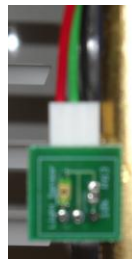


Figure 4.1.2c The light-sensing side of the light sensor MUST face out of your home.

(6) Install the motor control box: Choose a spot above your window frame to install the black motor control box. Try to choose a spot where the control box will not be visible, such as behind the valence above your window frame. Also, choose a spot as close as possible to the motor on your window blind assembly (see Section 4.1.2.1). Using a Phillips screwdriver and two wood screws, fix the motor control box to the wall or window frame through the screw holes on the mounting flanges. The motor control box

has one red and one black wire coming out of it. Attach these wires to the similar colored connector on the motor assembly.

(7) Connect the power cords: The battery pack has two power plugs. Attach the shorter cord to the receptacle on the side of the motor control box. Attach the longer cord to the receptacle on the side of the window control box. Leave the battery switch in the off position for now.

(8) Connect the control wiring: Attach the 6-wire window assembly connector coming from your window blind assembly to the associated port on the top of your window control box. Using the 4-wire motor connector, attach the motor control box to the associated port on the top of the window control box, as shown in **Figure 4.1.2d**.



Figure 4.1.2d Port placement on the top of the window control box. From left to right: light sensor, window assembly connector, motor connector.

(9) Turn on the system: Using the switch on the battery pack, turn on the power to the system. You should now be able to use the window control buttons on the window control box to open and close your window blinds.

Recommended: If you have installed a PC unit in Section 4.1.1, you can now connect this window to the PC unit. Check to be sure your PC communication box is plugged into a USB port on your computer. The software does not need to be running in order to connect a new window. Press and hold the “Connect” button on the window control box for 2 seconds. The window is now connected to the system and will remain connected unless it is manually removed from the system using the PC application. If you ever need to reconnect the window, repeat this process.

(10) Repeat steps 1-9 for all window units

4.1.1 Remote Control Setup (optional)

Note: The remote control can operate independent of the PC software. However, individual window units cannot connect to the system without the PC communication box being plugged into a USB drive. Therefore, the remote control will not be able to find connected windows if the PC communication box is plugged into a USB port.

(1) Insert a battery: Using a Philips screwdriver, remove the faceplate of the remote control. Attach the provided 9V battery to the battery connector inside of the remote as shown in **Figure 4.1.3a**. If at any point the remote battery dies, it can be replaced by following this same procedure.

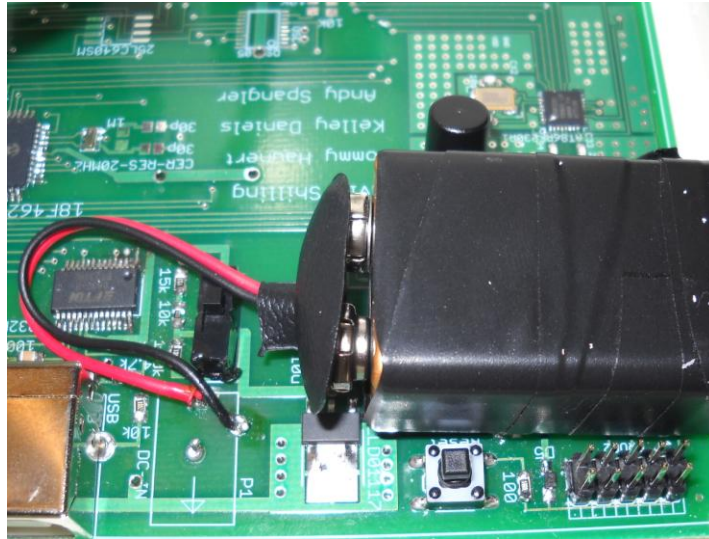


Figure 4.1.3a Connection of the 9V battery

(2) The remote is now operational

4.2 User's Guide

The Smart Windows system you have purchased is designed to be easy to install and easier to operate. As described in Section 4.1 (Installation Guide), simply pressing the connect button on your new window will connect it to the system.

Now that your window is connected, you are free to operate it as you choose. Your window has three modes of operation: manual mode, eco mode, and timer mode. Each mode of operation is described below. Following these descriptions, a troubleshooting guide is included for your benefit.

4.2.1 Manual Mode

In manual mode, you have control over your window blind. To enter manual mode, load the PC software and connect the OWU. Be sure that you have connected the OWU to the system by pressing connect during installation. Press the refresh button on your screen, and your OWU should appear. Click on the icon of the window to give this window a unique name. This name will help you identify which window is which as your collection grows. Names can be up to 10 characters in length. To move the window into manual mode, click on the red M below the window.

Now that the window is in manual mode, you can control it one of three ways. First, you can open, close, or half-open the window blind right from your PC by simply clicking the appropriate box under the window. You can also control the Smart Window at the press of the button. Press and hold the open or close buttons on your window control box. When the window blind is opened or closed to your liking, release the button.

Finally, you can control your window blind using the provided wireless remote control. First, ensure that your PCU is plugged into a computer. Then, press any button on the remote to wake up the remote and switch it on. The remote will automatically search for connected windows through your PC. The PC application does not have to be running for this process. Once the windows are displayed on your remote screen, use the up and down arrows to select the window you are interested in. Then, press either the open or close button to change the state of that window blind. Be sure to press the button firmly and hold it for about 1 second. If you connect new windows to your system, the remote will automatically find the names of these windows in a few minutes.

4.2.2 Eco Mode

In eco mode, your window blinds will adapt to changing sunlight. This mode is ideal if you are running your home's air conditioning. As sunlight levels increase, your air conditioning unit has to work harder to cool the house. Smart Windows will automatically close your window blinds when it becomes too sunny, saving you money.

To enter eco mode, load the PC software and connect the OWU. Be sure that you have connected the OWU to the system by pressing connect during installation. Press the refresh button on your screen, and your OWU should appear. Click on the icon of the window to give this window a unique name. This name will help you identify which window is which as your collection grows. Names can be up to 10 characters in length. To move the window into manual mode, click on the sun icon below the window.

Once your window is in eco mode, it will adapt to changing sunlight. If at any time you press an open or close button associated with this Smart Window, either on the window control box or on the remote control, the window blind will automatically leave eco mode and enter manual mode.

4.2.3 Timer Mode

In timer mode, your Smart Window has the ability to open or close at certain times of the day. You choose these times from a simple menu on your PC application. Each Smart Window can have its own set of alarm times.

To enter timer mode, load the PC software and connect the OWU. Be sure that you have connected the OWU to the system by pressing connect during installation. Press the refresh button on your screen, and your OWU should appear. Click on the icon of the window to give this window a unique name. This name will help you identify which window is which as your collection grows. Names can be up to 10 characters in length. To move the window into manual mode, click on the clock icon below the window.

Once your window is in timer mode, it will open or close according to the preferences you have set on the PC application. The PC application does not have to be running for the window to operate in timer mode. If you wish to change your preferences for a particular window at any time, simply open the PC application and click on the window icon you wish to modify. You can store up to 8 alarm times for any given Smart Window.

If at any time you press an open or close button associated with this Smart Window, either on the window control box or on the remote control, the window blind will automatically leave timer mode and enter manual mode. The window will no longer react to the alarm preferences you have set.

4.2.4 Troubleshooting Guide

If you are having problems with your system, first shut off the power and restart the device. Most times, this will correct any issues. Try restarting the software or reconnecting the window to the system. When connecting new windows or re-loading the remote control, be sure that the PCU is connected to a computer using the provided USB cable. If one part of your system is not working, be sure that the batteries do not need charged. The PC application will report the battery voltage of all connected

windows. OWU batteries can be charged with any home NiMH AA charger. The remote battery is a common 9V and must be replaced. Do not charge the 9V battery.

If none of the above general solutions solve your problem, try the application specific solutions listed below. The sections below also list instructions for verifying that your Smart Windows installation is working correctly.

When manually setting window to opened, closed, or middle:

- After a brief transmission delay, the PC application should then give a message box indicating the command was successfully sent.
- After another brief delay, the window should respond to the command.
- If the window is already in that mode it will not throw any error, but will simply not visually respond (since it immediately hits the limit switch).
- Troubleshooting:
 - Is the window already in the mode (open, closed, middle) which you are trying to change it into? If so, you will not see a visual change.
 - Try hitting “Refresh” on the PC
 - Does the microcontroller still show as there?
 - If not, push the “connect” button on the window module. Then hit the “Refresh” button again.
 - Did the PC application never give a message box indicating the command was successfully sent?
 - If not, the PC and/or microcontroller are hung up
 - Restart the PC application.
 - The microcontroller should automatically break out of its hang within 2 minutes.
 - If there is still a problem, start the microcontroller again. Then press the “connect” button to add it.
 - Other ideas:
 - Try clicking the “manual mode” button again
 - Try restarting your computer

When changing window mode:

- After a brief transmission delay, the PC application should then give a message box indicating the command was successfully sent.
- After another brief delay, the window should switch its mode, but it may not be immediately obvious.
- To test **Eco** mode:
 - Take a flashlight, or portable light source, and put it close to the light sensor. The window blinds should close. Then remove the light source, and the window blinds should open again.
- To test **Timing** mode:

- Open your local system time (in the lower right on a Window machine), and note the current time (the real time is irrelevant, the time for your window modules will be taken from this).
- Switch the mode for the target microcontroller to Timer mode (see directions above).
- Open the time settings by clicking on the window. Choose a time that is some short but reasonable period of time out (such as 2 minutes from what you see is the current time).
- Click “Apply Times”
- Wait until the correct time and watch the window module for the change to occur.

When deleting all windows from PC:

- Go to menu bar and select “Delete all window modules”
- Remember to hit refresh button

5 Conclusions

While most people choose to ignore their window blinds, our research has shown that this is a poor choice. At the very least, improper window treatment operation can cost homeowner's money on their energy bill. At the very worst, negligent operation of window blinds can lead to increased likelihood of burglary and theft. It is clear that we should all pay more attention to our window blinds.

However, this is not an easy proposal. Owning a home brings with it a full list of daily tasks and chores. As any homeowner knows, the summation of these chores can become tiring and tedious. Who has time to worry about meticulously opening and closing window blinds several times a day?

Smart Windows solves the problem of improperly operated window blinds without adding more work for the homeowner. Smart Windows systems can be installed quickly and easily. Once installed, the user can set his or her preferences right from the PC. Then, the windows will operate themselves with no extra effort from the user. If the user chooses to take a more active role in the operation of his window treatments, he or she has a variety of options. The remote control adds an impressive piece of technology into your home at little cost. The PC application allows you to control your window blinds as part of your morning ritual – without walking all around the house.

The Smart Windows system has shown commercial viability. Solar Shades, a company producing polarizing window films has expressed interest in adding Smart Windows technology for their product demos. As windows move towards a more technological future, the Smart Windows technology is well placed to become a part of everyday our everyday window experience.

This project brings to the forefront a series of design issues and serious engineering challenges. However, a solution exists to each of these various problems. In discovering and implementing these solutions, we have gained a great deal of technical knowledge as well as problem solving strategies.

In an effort to best minimize costs and maximize functionality, this design has been carefully considered and revised. A wide variety of sources and experts were consulted in constructing this design. To this end, several members of the Notre Dame engineering faculty, especially Professor Schafer and Will McCleoud, the owner of Solar Shade, deserve thanks for their valuable tools and guidance.

The Smart Windows system, as it is designed here, is a widely applicable and adaptable system. However, it has also been targeted as a valuable marketing tool for the Solar Shades window treatment.

6 Appendices

These four appendices are found on the following pages:

6.1 Complete Hardware Schematic

6.2 Complete Software Listing

6.2.1 Microcontroller Software

6.2.2 PC Software

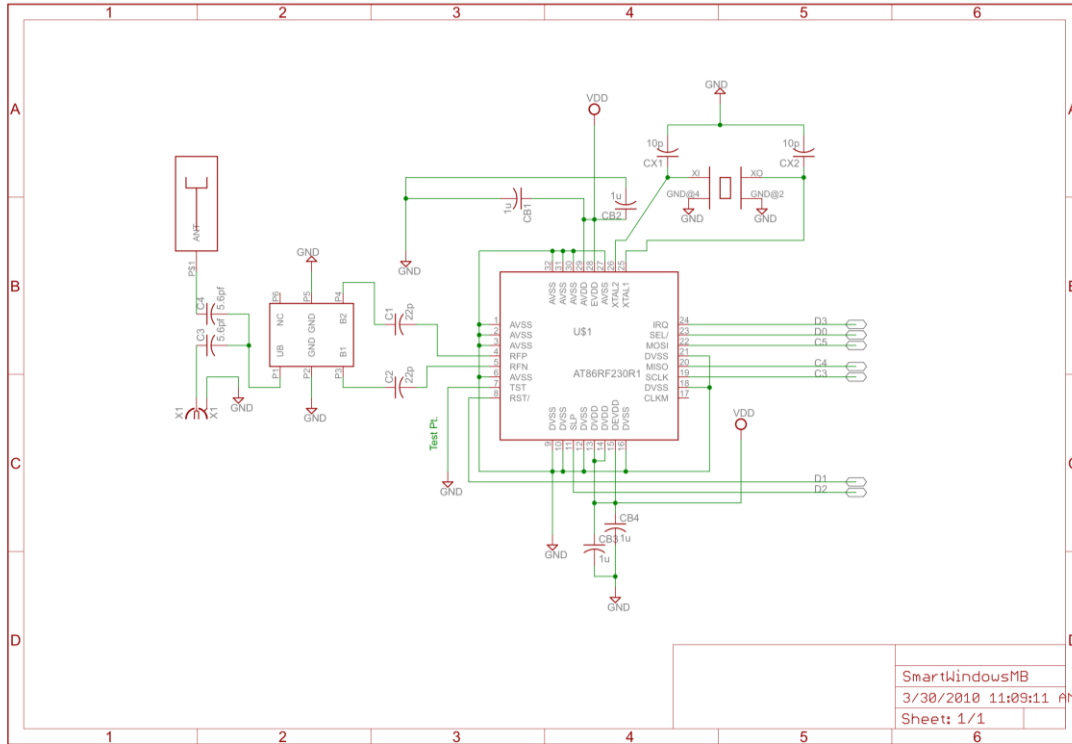
6.3 Bill of Materials

6.4 Data Sheets

6.1 Complete Hardware Schematics

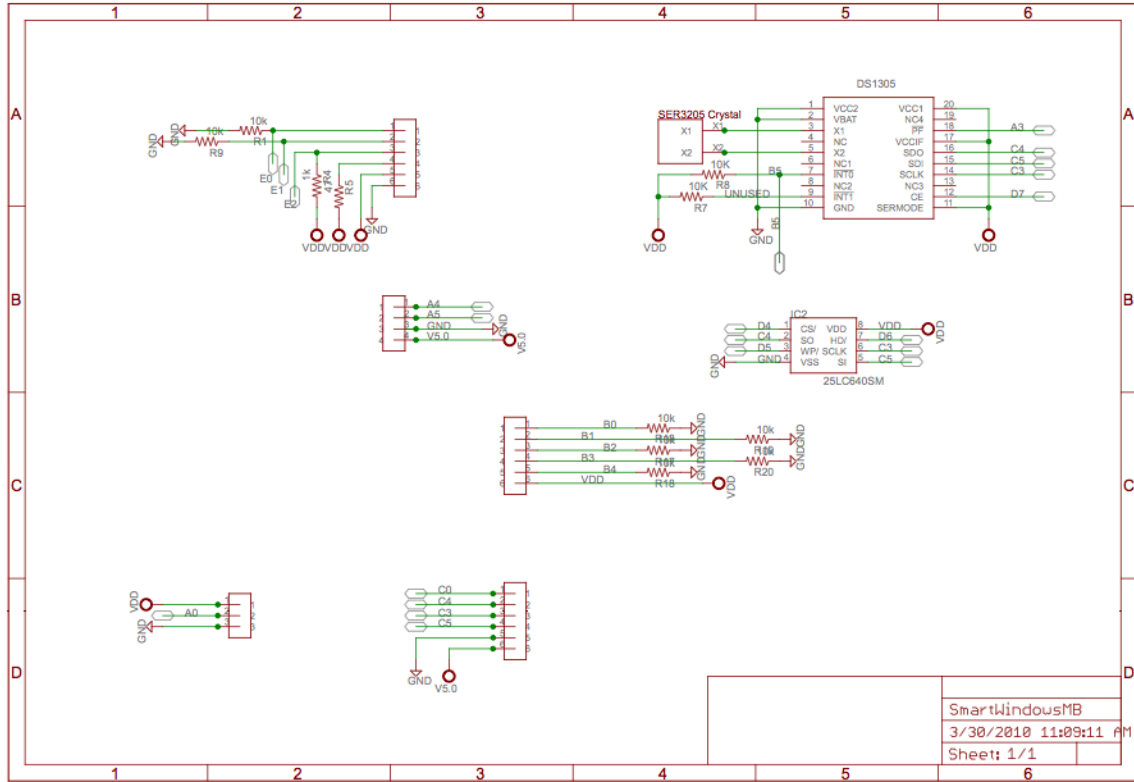
The hardware consists of three circuit boards: the main board, the motor board, and the light sensor board.

Main Board: Page 1

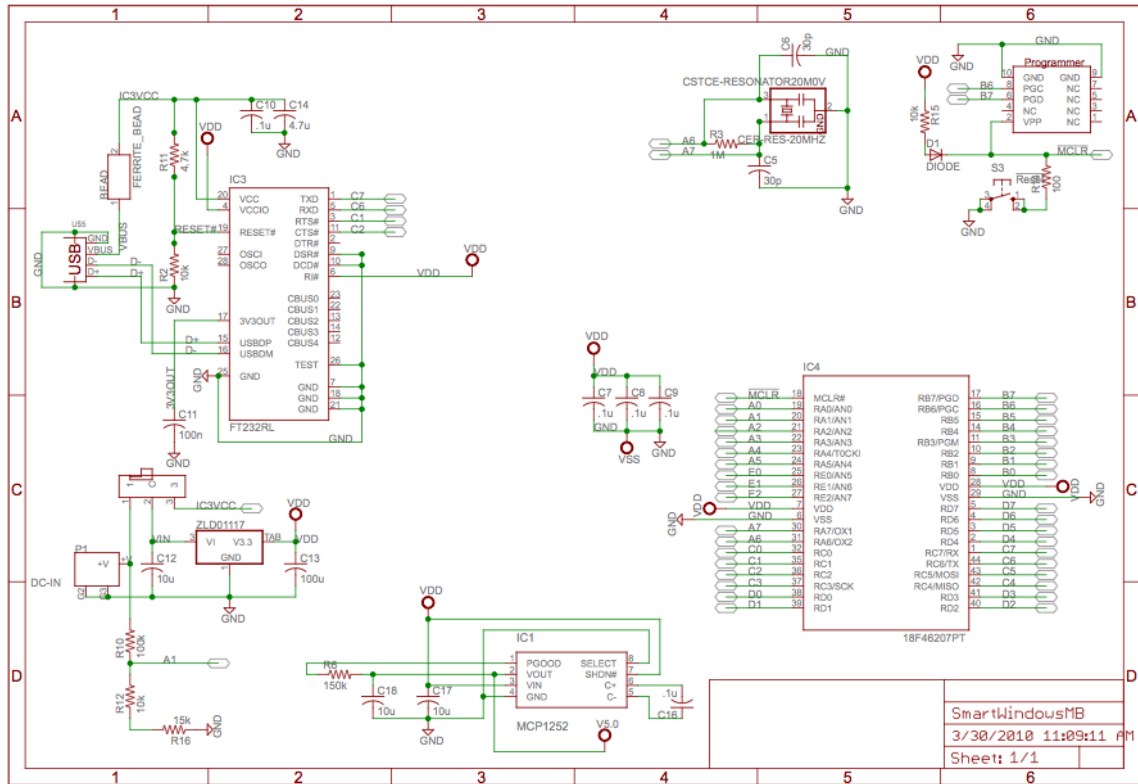


5/3/2010 12:33:40 AM N:\Public\SmartWindows\Eagle\CurrentBoards\SmartWindowsMB\SmartWindowsMB.sch (Sheet: 1/1)

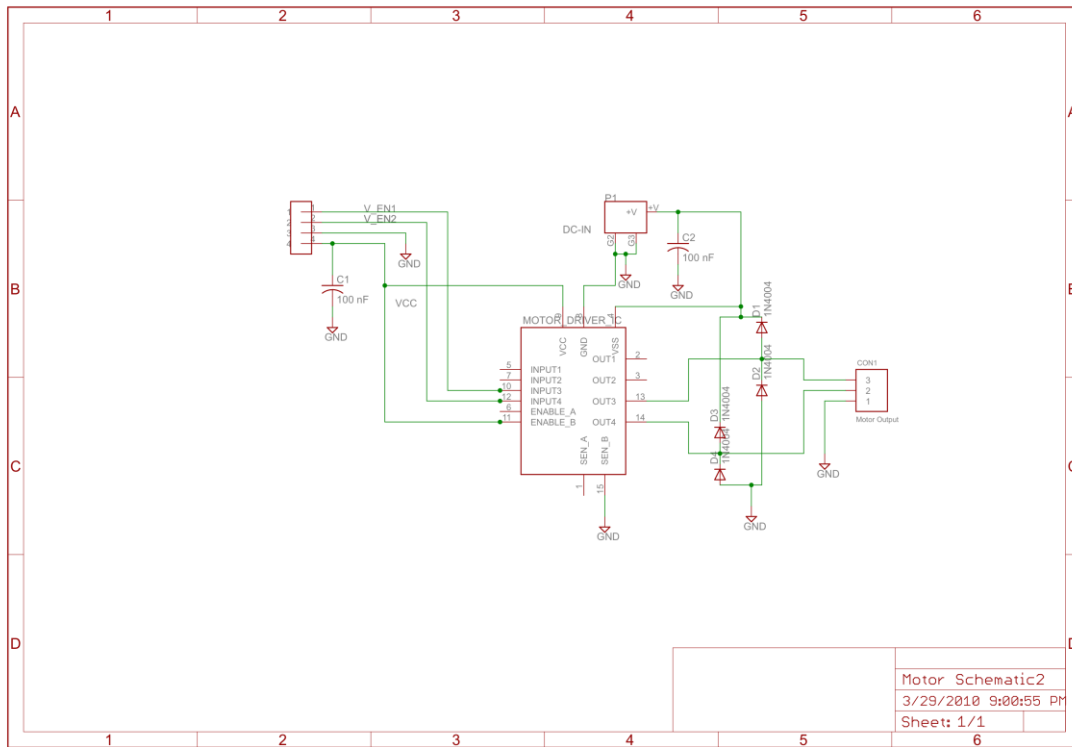
Main Board: Page 2



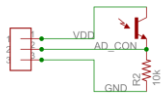
Main Board: Page 3



Motor Board



Light Sensor Board



6.2 Complete Software Listing

The software consists of three main submodules: microcontroller software (embedded C), PC software (Python QT), and Android software (Java).

6.2.1 Microcontroller Software

The microcontroller software consists of a main function and several libraries. These are listed below. The main program contains of a software switch. Switch position 0 indicates the software is for a PCU, switch position 1 indicates the software is for a OWU, and switch position 2 indicates the software is for a RCU.

Main Program

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

#pragma DATA _CONFIG1H, _OSC_HS_1H
#pragma DATA _CONFIG2H, _WDT_OFF_2H
#pragma DATA _CONFIG4L, _LVP_OFF_4L & _XINST_OFF_4L
#pragma DATA _CONFIG3H, _MCLRE_ON_3H
#pragma CLOCK_FREQ 20000000

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      SmartWindows.c              //
// Main code for the project        //
////////////////////////////////////

/--// Communication Definitions
#define pc_attn  0
#define pc_poweron 1
#define win_max  10

/--// Function prototypes
void PIC_sleep(void);           // Puts the Microchip in idle mode
```

//--// Volatile bit Declarations

```
// Timers
volatile bit tmr0if@INTCON.2;           // Timer 0 Interrupt Flag

// USART
volatile bit rcif@PIE1.5;               // USART2 Receive Interrupt Flag

// ZIGBEE
volatile bit zb_rs @PORTD.1;            // ZigBee Reset
volatile bit zb_slp@PORTD.2;           // ZigBee Sleep Signal
volatile bit zb_irq@PORTD.3;           // ZigBee Interrupt Signal

// Buttons
volatile bit b0@PORTB.0;                 // Button attached to B0
volatile bit b1@PORTB.1;                 // Button attached to B1
volatile bit b2@PORTB.2;                 // Button attached to B2
volatile bit b3@PORTB.3;                 // Button attached to B3
volatile bit b4@PORTB.4;                 // Button attached to B4
volatile bit connect_button@PORTB.2;    // Connect button

// Motor block
volatile bit open_command @PORTA.4;     // output
volatile bit close_command @PORTA.5;    // output
volatile bit close_button @PORTB.1;     // input
volatile bit open_button @PORTB.0;      // input
volatile bit open_stop @PORTE.0;        // input
volatile bit close_stop @PORTE.1;       // input
volatile bit middle_stop @PORTE.2;      // input

// RCU
volatile bit RCU_open @PORTB.0;
volatile bit RCU_close @PORTB.1;
volatile bit RCU_up @PORTB.2;
volatile bit RCU_down @PORTB.3;

//--// Global Interrupt Semiphores
bool tmr0is;                             // Timer 0 Rollover Interrupt
bool rcis;                                // USART2 Character
Receiver
```

//--// MAIN FUNCTION

```
void main(void){

    //--// Setup I/O Pins
    adcon1 = 00001101b;
    intcon2.7 = 1;

    portb = 0;
    trisb.5 = 1;
    trisb.4 = 1;
    trisb.3 = 1;
    trisb.2 = 1;
    trisb.1 = 1;
    trisb.0 = 1;

    trisa.5 = 0;
    trisa.4 = 0;

    //--// General Variables
    unsigned short i; // Loop variable used
multiple places
    unsigned short j; // Loop variable used
multiple places
    unsigned short k; // Loop variable used
multiple places
    char debounce;

    //--// USART Initialization
    usart_init(57600); // Initializes USART2
@ 57.6k baud rate
    char c = 'l';

    //--// Real Time Clock Initialization
    RTC_init();
    struct time t;
    struct time* tp = &t;
    t.hr10 = 1;
    t.hr1 = 1;
    t.hr = 11;
    t.min10 = 1;
    t.min1 = 1;
    t.min = 11;
    t.sec10 = 1;
    t.sec1 = 1;
    t.sec = 11;
    t.day = 2;
```

```
t.mon10 = 1;
t.mon1 = 1;
t.mon = 11;
t.date10= 1;
t.date1 = 1;
t.date = 11;
t.yr10 = 1;
t.yr1 = 1;
t.yr = 11;
RTC_set(tp);
delay_ms(100);
t.day = 3;
RTC_get(tp);
RTC_print_terminal(tp);

char alarm[8][5];
for (i=0;i<8;i++){
    for (j=0;j<5;j++) {
        alarm[i][j] = 255;
    }
}
char check1 = 0;
char check2 = 0;
char check3 = 0;
char check4 = 0;
char days = 0;
char sec_bcd=0;
char min_bcd=0;
char hr_bcd=0;
usart_putc(c);

//-- ZigBee Initialization
spi_init(0); // Initialize the SPI
usart_putc(c);

zb_init(11); // Initialize for ZigBee
usart_putc(c);
char seq = 0; // Sequencing field for
messages char success; // Success of an
operation
char message[85]; // Holds messages to
send unsigned short message_length; // Holds length of current
message
```

```

char data[70]; // Holds data received
unsigned short data_length; // Holds length of current
data
transmitted char pc_instructions[win_max][11]; // Instruction from the PC to be
instructions are available char new_instructions[win_max]; // Indicates that new
for(i=0;i<win_max;i++){ // Initialize to zero
    new_instructions[i] = 0;
    for(j=0;j<41;j++){
        pc_instructions[i][j]=0;
    }
}
char current_window = 254; // Currently addressed
window 254 - none, 255 - all

char pan0; // LS Byte of
PAN address char pan1; // MS Byte of
PAN address char add0; // LS Byte of
device address char add1; // MS Byte of
device address char source_add1; // Received message
source char source_add0; // Received message
source

unsigned short window_count; // Number of connected windows
char name[10][10];
char status[10];
char light[10];
char battery[10];
char disp[11];

//--// USART Initialization
//usart_init(57600); // Initializes USART2
@ 57.6k baud rate
//char c;

//--// Interrupts Initialization
intcon |= 11000000b; // Global and Peripheral
Interrupts enables
tmr0is = 0;
rcis = 0;

```

```

//
Enable specific interrupts separately

//--// Type-dependant Code
// CASE 0: PCU Base Station -- Do not modify (D.C.S.)
// CASE 1: OWU Window Station -- Do not modify (D.C.S)
// CASE 2: RCU Remote Control -- Do not modify (D.C.S)
// CASE 3: Product Testing and Evaluation
switch (2) {

//--// PCU
case 0:
BASE STATION
    open_command = 0;
    close_command = 0;
//--// ZigBee Addressing
    pan1 = 0xBA;
    pan0 = 0x5E;
    add1 = 0xA0;
    add0 = 0x00;
    zb_write_reg(PAN_ID_1,pan1); // PAN ID 0xBA5E
    zb_write_reg(PAN_ID_0,pan0);
    zb_write_reg(SHORT_ADDR_1,add1); // Device Address
0xA000
    zb_write_reg(SHORT_ADDR_0,add0);
//--// Hold up to 10 Windows with name-length 10
    window_count = EEPROM_read(0,0);// Get window count from
EEPROM
    if (window_count==255) {
        window_count = 0;
    }
    for (i=0;i<window_count;i++){
// For each window...
        status[i] = EEPROM_read(0,11*(i+1)+0);
// ...Read in the status
        light[i] = 0;
// Default light reading
        for (j=0;j<10;j++){
            name[i][j] = EEPROM_read(0,11*(i+1)+1+j);
// ...Read in the name
        }
    }

// ENABLE RECEIVER
    usart_receive_enable();

```

```
while(1) {
    //---// INTERRUPT CHECK ROUTING
    if (tmr0is==1) {
        tmr0is=0;
    }
    if (rcis==1) {
// Received an attention signal
        rcis=0;
// Clear the semaphore
        c = usart_getc();
        delay_ms(400);
        usart_putc(c);

//---// Startup Message
        if (c==254) {
// Startup Code
            for (i=0;i<window_count;i++){
// Loop through connect windows
                delay_ms(800);
                usart_putc(i);

// Put window number
                delay_ms(200);
                for (j=0;j<10;j++){
                    usart_putc(name[i][j]);

// Put window name
                    delay_ms(100);
                }
            }
            usart_putc(255);

// Put success code
        }

//---// Regular Message
        else { // ELSE: Not a startup code

            //---// Obtain the rest of the instructions...
            current_window = c;

            //---// ...for 255 signals
            if (c==255) {

                for(i=0;i<11;i++){pc_instructions[0][i]=usart_getc();}
                    for(i=0;i<win_max;i++){
                        for(j=0;j<11;j++){
                            pc_instructions[i][j] =
pc_instructions[0][j];

```

```
        }
        new_instructions[i] = 1;
    }

    //--// Special Cases
    switch (pc_instructions[0][0]) {
    case 'a':
        // NOT FILLING FOR

        break;
    case 'n':
        // Record the names

        for(i=0;i<window_count;i++){
            for (j=0;j<10;j++){
                EEPROM_write(pc_instructions[i][j+1],0,11*(i+1)+1+j);
            }
        }
        for
            new_instructions[i] =

        }

        break;
    case 't':
        // Store time locally
        RTC_init();
        t.sec10 =

        t.sec1 =

        t.min10 =

        t.min1 =

        t.hr10 =

        t.hr1 =

        t.day =

        RTC_set(tp);
        delay_ms(50);
        spi_init(0);

SIMPLICITY (SHOULD BE '1','2','3','4')

for(i=0;i<window_count;i++){
    (i=0;i<window_count;i++){
    0;

    (pc_instructions[0][1]>>4) & 00001111b;
    pc_instructions[0][1] & 00001111b;
    (pc_instructions[0][2]>>4) & 00001111b;
    pc_instructions[0][2] & 00001111b;
    (pc_instructions[0][3]>>4) & 00001111b;
    pc_instructions[0][3] & 00001111b;
    pc_instructions[0][4];
```



```

break;
case 'l':
    delay_s(1);
    for
        if
        if
        }
    for
        usart_putc(light[i]);
        delay_ms(200);
        usart_putc(battery[i]);
        delay_ms(200);
        }
    usart_putc(255);
    for
        new_instructions[i] =
        }
break;
case 'e':
    for
        new_instructions[i] =
        }
        window_count = 0;
        EEPROM_write(0,0,0);
        // Erase Window count
        for
            // For each window...
            EEPROM_write(0,0,11*(i+1)+0);
            // ...overwrite in the status
            for (j=0;j<10;j++){
            EEPROM_write(0,0,11*(i+1)+1+j);
            // ...overwrite in the name
            }
            }
break;
case 'c':
    // Nothing needed

```

```

(i=0;i<window_count;i++) {
(light[i]==255){light[i]=254;}
(battery[i]==255){battery[i]=254;}

(i=0;i<window_count;i++) { // Put the readings

// Signal eng of message

(i=0;i<window_count;i++){
0; // Clear the new message

(i=0;i<window_count;i++){
0;

// Erase Window count
(i=0;i<window_count;i++){
EEPROM_write(0,0,11*(i+1)+0);
EEPROM_write(0,0,11*(i+1)+1+j);

```

```
        break;
        default:
            // Nothing needed
        break;
    }
}

//--// ... for individual windows
else if (c<win_max) {

    for(i=0;i<11;i++){pc_instructions[current_window][i]=usart_getc();}
    new_instructions[current_window] =
1;

    //--// Special cases
    switch
(pc_instructions[current_window][0]){

        case '1':

            new_instructions[current_window] = 2;
            break;
        case '2':

            new_instructions[current_window] = 2;
            break;
        case '3':

            new_instructions[current_window] = 2;
            break;
        case '4':

            new_instructions[current_window] = 2;
            break;
        case 'n':
            for (j=0;j<10;j++){

                EEPROM_write(pc_instructions[current_window][j+1],0,11*(i+1)+1+j);

                name[current_window][j] = pc_instructions[current_window][j+1];
            }

            new_instructions[current_window] = 0;

            break;
        default:
            // Nothing
            break;
    }
}
```

```

    }
    }
}
usart_receive_enable();
// Reenable random communication
}
//---// INTERRUPT CHECK ROUTINE

    success = 0; // Reset the
success flag to avoid mixups
    data[0] = 0xF0; //
Reset data flag to avoid mixups
    success = zb_rx(10000); // Listen for a
while

    if (success){
        message_length = zb_read_fb(message);
// Message Length
        data_length = message_length - header_length - 2;
        for (i=0; i<data_length; i++){
// Data extraction
            data[i] = message[header_length+i];
        }
        source_add0 = message[header_length-2];
// Source extraction
        source_add1 = message[header_length-1];

// Use the data
        switch (source_add1) {
//--// Remote request
            case 0xC0:

// Remote control unit
                if (data[0] == 0xFF) { //
Request for names
                    message[0] = window_count;
                    success =
zb_tx(0xC0,0x00,message,1,seq);
                    seq++;
                    for
(i=0;i<window_count;i++) {
                        for (j=0;j<10;j++) {
                            disp[j] =
name[i][j];
                        }
                    }
                    success = 0;

```

```

                                while (!success){
                                    success =
z_b_tx(0xC0,0x00,disp,10,seq);
                                }
                                seq++;
                                }
                                }
                                break;
                                //--// Window unit request
                                default:
                                // On window unit

                                //--// Request to join
                                if (data[0] == 0x00) {

                                    window_count++;

                                    // Send the window its new
                                address
                                    message[0] = 0x01;
                                    message[1] = pan1;
                                    message[2] = pan0;
                                    message[3] = 0xB0;
                                    message[4] = window_count-
                                1;

                                    while (data[0] != 0x02) {
                                        seq =
                                        success =
z_b_broadcast(message,5,seq);
                                        if (success){
z_b_rx(30000);
                                        // Listen for a while

                                        message_length = z_b_read_fb(message);
                                        // Message
                                        Length
                                        data_length =
                                        message_length - header_length - 2;
                                        for (i=0;
                                        // Data extraction
                                        i<data_length; i++){
                                            data[i]
                                        }
                                        source_add0 =
                                        = message[header_length+i];
                                        // Source extraction
                                        source_add1 =
                                        message[header_length-2];
                                        message[header_length-1];

```

```

    }
    }
    name>window_count-1][0] =
    name>window_count-1][1] =
    name>window_count-1][2] =
    name>window_count-1][3] =
    name>window_count-1][4] =
    name>window_count-1][5] =
    name>window_count-1][6] =
    name>window_count-1][7] =
    name>window_count-1][8] =
    name>window_count-1][9] =

'W';
'i';
'n';
'd';
'o';
'w';
'0';
'0';
'0';
window_count-1 +'0';

// Update the records

EEPROM_write(window_count,0,0);
i=window_count-1;

EEPROM_write(window_count-1,0,11*(i+1)+0);
for (j=0;j<10;j++){

EEPROM_write(name>window_count-1][j],0,11*(i+1)+1+j);
    }

// Clean the new_instructions

new_instructions>window_count-1] = 0;
}

//--// Request for instructions
else if (data[0] == 0x03) {
    light[source_add0] = data[1];

// Second byte is the light data
```

```

data[2];          // Third byte is battery data
                                battery[source_add0] =

                                // New Instructions available

for correct window
                                if
(new_instructions[source_add0]){
                                for (i=0;i<11;i++){
                                message[i] =

                                }
                                success =

                                seq++;
                                if
                                uart_putc('a');
                                }

                                new_instructions[source_add0] = 0;
                                }

                                // No new Instructions

available for the correct window
                                else{
                                message[0] = 0x00;
                                success =

                                seq++;
                                }
                                }

                                break;
                                }
                                }
                                if (rcsta.1){
                                uart_receive_disable();          // Clear
                                uart_receive_enable();
                                }
                                }
                                // DISABLE RECEIVER
                                uart_receive_disable();
                                break;

                                //--// OWU

```

```

case 1: //
REMOTE STATION
    // OWU Variables
    motor_init();
    char mode = 0; // 0
(1,2) manual, 3 green, 4 security
    char state; // 0
closed, 1 middle, 2 open
        if (porte.0){state=2;}
        else if (porte.1){state=0;}
        else {state=1;}

    // ZigBee addressing
    pan1 = EEPROM_read(0x00,0x00); // PAN ID from
EEPROM
    pan0 = EEPROM_read(0x00,0x01);
    add1 = EEPROM_read(0x00,0x02); // Device Address
from EEPROM
    add0 = EEPROM_read(0x00,0x03);
    zb_write_reg(PAN_ID_1,pan1);
    zb_write_reg(PAN_ID_0,pan0);
    zb_write_reg(SHORT_ADDR_1,add1);
    zb_write_reg(SHORT_ADDR_0,add0);

    // Initialize Timer1
    t0con = 10000111b; // Initialize T0
    tmr0h = 46005/256; // Set T0 to

interrupt @ 1 Hz
    tmr0l = 46005%256;

    // Initialize adcon for light sensor
    ad_init(0);
    char light_reading = 0;
    char bat_reading = 0;

    // Technical Readout
    usart_printf("\n\rTechnical Read Out:\n\rPower On");

    while(1) {
        //---// INTERRUPT CHECK ROUTING
        if (tmr0is==1) {
            tmr0is=0;
        }
        if (rcis==1) {
            rcis=0;
            usart_receive_enable();
        }
    }

```

```
    }

// Technical Readout
usart_printf("\n\r>");

/--// Check the light-sensor and battery
    light_reading = ad_conv();
    ad_init(1);
    delay_ms(50);
    usart_printf("{L:}");
    usart_putShort(light_reading);
    usart_printf(" B:");
    bat_reading = ad_conv();
    usart_putShort(bat_reading);
    usart_printf("} ");
    ad_init(0);

/--// STATE MACHINE
    switch (mode) {
    case 3: // Green Mode
        //Check Buttons
        if(open_button||close_button){
            debounce = 0;
            for (i=0;i<50;i++){
                if
(open_button||close_button) {debounce++;}
            }
            if (debounce>25) {mode = 0;}
        }
        while(open_button && (!open_stop ))
{open_command = 1;}

        open_command = 0;
        while(close_button && (!close_stop))
{close_command = 1;}

        close_command = 0;
        //Check Light
        if (porte.0){state=2;} // Open
        else if (porte.1){state=0;} // Close
        else {state=1;} //

Middle

        switch (state) {
        case 0:
            if (light_reading < 75)
{motor_open();}

            break;
        case 2:
```



```

        if (light_reading >= 75)
{motor_close();}
        break;
        default:
        if (light_reading < 75)
{motor_open();}
        else
{motor_close();}
        break;
        }
break;
case 4: // Security Mode
//Check Buttons
if(open_button||close_button){
    debounce = 0;
    for (i=0;i<100;i++){
        if
(open_button||close_button) {debounce++;}
        }
        if (debounce>50) {mode = 0;}
        }
while(open_button && (!open_stop ))
{open_command = 1;}
    open_command = 0;
while(close_button && (!close_stop))
{close_command = 1;}
    close_command = 0;
//Check Alarms
RTC_init();
RTC_get(tp);
/*
//RTC_print_terminal(tp);
sec_bcd = (t.sec10<<4) | t.sec1;

min_bcd = (t.min10<<4) | t.min1;
hr_bcd = (t.hr10<<4) | t.hr1;
uart_putc(' ');
uart_putByte(sec_bcd);
uart_putc(':');
uart_putByte(min_bcd);
uart_putc(':');
uart_putByte(hr_bcd);
uart_putc(':');
uart_putShort(t.day);
uart_putc(' ');
*/

```

```
for (i=0;i<8;i++) {
    check1 = 0;
    check2 = 0;
    check3 = 0;
    check4 = 0;
    if (alarm[i][1] != 255){
        usart_printf("{A}");
        usart_putShort(i);
        usart_printf(":");
        if (alarm[i][1] == sec_bcd) {
            check1=1;
            usart_putc('1');
        }
        if (alarm[i][2] == min_bcd) {
            check2=1;
            usart_putc('2');
        }
        if (alarm[i][3] == hr_bcd) {
            check3=1;
            usart_putc('3');
        }
        days = alarm[i][4];
        switch (t.day){
            case 1:
                check4 =
                if (check4)
                    break;
            case 2:
                check4 =
                if (check4)
                    break;
            case 3:
                check4 =
                if (check4)
                    break;
            case 4:
                check4 =
                if (check4)
                    break;
        }
    }
}

days.0;
{usart_putc('4');}

days.1;
{usart_putc('4');}

days.2;
{usart_putc('4');}

days.3;
{usart_putc('4');}
```

```
days.4;
{usart_putc('4');}
```

```
days.5;
{usart_putc('4');}
```

```
days.6;
{usart_putc('4');}
```

```
check3 && check4){
```

```
    motor_open();
```

```
    motor_middle();
```

```
    motor_close();
```

```
        }
        spi_init(0); // Initialize the SPI
    }
break;
```

```
        break;
    case 5:
        check4 =
            if (check4)
                break;
    case 6:
        check4 =
            if (check4)
                break;
    case 7:
        check4 =
            if (check4)
                break;
    default:
        break;
    }
    if (check1 && check2 &&
        switch (alarm[i][0]) {
            case 0:
                break;
            case 1:
                break;
            case 2:
                break;
            default:
                break;
        }
        usart_printf("}");
    }
}
```

```
default: // Manual Mode
    if (middle_stop) {usart_putc('m');}
    while(open_button && (!open_stop))

{open_command = 1;}

    open_command = 0;
    while(close_button && (!close_stop))

{close_command = 1;}

    close_command = 0;
break;
}

/--// Wake up the ZigBee
    zb_wake();

/--// Contact the PC
    message[0] = 0x03;
    message[1] = light_reading;
    message[2] = bat_reading;
    success = zb_tx(0xA0,0x00,message,3,seq);
    seq++;
    success = 0;
    success = zb_rx(2000);

// Listen for a message
    if (success){
        message_length = zb_read_fb(message);

// Once one is received, read FB

// Extract the data
        data_length = message_length -
header_length - 2;

        data[0] = message[header_length];
        usart_printf(" {M: ");
        for (i=0; i<data_length; i++){
            data[i] = message[header_length+i];
        }
        usart_putc(data[0]);
        usart_printf(")");
        switch(data[0]) {
            case 'c':
                switch (data[1]) {
                    case 0:
                        mode = 0;
                        motor_open();
                    break;
                    case 1:
                        mode = 0;
```

```
motor_middle();  
  
break;  
case 2:  
    mode = 0;  
    motor_close();  
break;  
case 3:  
    mode = 3;  
break;  
case 4:  
    mode = 4;  
break;  
}  
break;  
case '1':  
    k = 1;  
    for (i=0;i<2;i++) {  
        for (j=0;j<5;j++) {  
            alarm[i][j] =  
  
                k++;  
            }  
        }  
break;  
case '2':  
    k = 1;  
    for (i=2;i<4;i++) {  
        for (j=0;j<5;j++) {  
            alarm[i][j] =  
  
                k++;  
            }  
        }  
break;  
case '3':  
    k = 1;  
    for (i=4;i<6;i++) {  
        for (j=0;j<5;j++) {  
            alarm[i][j] =  
  
                k++;  
            }  
        }  
data[k];  
data[k];  
data[k];
```

```
break;
case '4':
    k = 1;
    for (i=6;i<8;i++) {
        for (j=0;j<5;j++) {
            alarm[i][j] =
data[k];
                                k++;
                                }
        }

/*
");
    usart_printf("\n\r\n\rAlarm 1:

    usart_putByte(alarm[0][0]);
    usart_putc(' ');
    usart_putByte(alarm[0][1]);
    usart_putc(' ');
    usart_putByte(alarm[0][2]);
    usart_putc(' ');
    usart_putByte(alarm[0][3]);
    usart_putc(' ');
    usart_putByte(alarm[0][4]);
    usart_printf("\n\r\n\rAlarm 2:

");
    usart_putByte(alarm[1][0]);
    usart_putc(' ');
    usart_putByte(alarm[1][1]);
    usart_putc(' ');
    usart_putByte(alarm[1][2]);
    usart_putc(' ');
    usart_putByte(alarm[1][3]);
    usart_putc(' ');
    usart_putByte(alarm[1][4]);
    usart_printf("\n\r\n\rAlarm 3:

");
    usart_putByte(alarm[2][0]);
    usart_putc(' ');
    usart_putByte(alarm[2][1]);
    usart_putc(' ');
    usart_putByte(alarm[2][2]);
    usart_putc(' ');
    usart_putByte(alarm[2][3]);
    usart_putc(' ');
    usart_putByte(alarm[2][4]);
*/
```

```
break;
case 't':
    RTC_init();
    t.sec10 = (data[1]>>4) &
    t.sec1 = data[1] &
    t.min10 = (data[2]>>4) &
    t.min1 = data[2] &
    t.hr10 = (data[3]>>4) &
    t.hr1 = data[3] &
    t.day = data[4];
    RTC_set(tp);
    delay_ms(10);
    spi_init(0); // Initialize the
                //zb_init(11); // Initialize for
break;
case 'e':
    pan1 = 0x00;
    pan0 = 0x00;
    add1 = 0x00;
    add0 = 0x00;
    zb_write_reg(PAN_ID_1,pan1); // PAN ID from EEPROM
    zb_write_reg(PAN_ID_0,pan0);
    zb_write_reg(SHORT_ADDR_1,add1); // Device Address from EEPROM
    zb_write_reg(SHORT_ADDR_0,add0);
    EEPROM_write(pan1,0x00,0x00);
    EEPROM_write(pan0,0x00,0x01);
    EEPROM_write(add1,0x00,0x02);
    EEPROM_write(add0,0x00,0x03);
break;
default:
```

```

        break;
    }

}

/--// Contact the Remote
    message[0] = 0x03;
    success = zb_tx(0xC0,0x00,message,1,seq);
    seq++;
    success = 0;
    success = zb_rx(1000);
// Listen for a message
    if (success){
        message_length = zb_read_fb(message);
// Once one is received, read FB
        // Extract the data
        data_length = message_length -
header_length - 2;

        data[0] = message[header_length];
        for (i=0; i<data_length; i++){
            data[i] = message[header_length+i];
        }
        if (porte.0){state=2;} // Open
        else if (porte.1){state=0;} // Close
        else {state=1;} //

Middle

        switch (data[0]) {
            case 0x01:
                if (state==0)
                    {motor_open();}
                else if (state==1)
                    {motor_open(); }
                break;
            case 0x02:
                if (state==2)
                    {motor_middle();}
                else if (state==1)
                    {motor_close();}
                break;
        }
    }

/--// CONNECT TO BASE STATION
if (connect_button){
    // Send broadcast message with
request/panid/address

```



```
message[0] = 0x00;
message[1] = pan1;
message[2] = pan0;
message[3] = add1;
message[4] = add0;
seq = zb_broadcast(message,5,seq);

// Listen for response with new panid/address
data[0]=0;
while(data[0] != 0x01){
    success = 0;
    success = zb_rx(0);
    if (success){
        message_length =
zb_read_fb(message);
        // Extract the data
        data_length = message_length -
header_length - 2;
        for (i=0; i<data_length; i++){
            data[i] =
message[header_length+i];
        }
    }
}
// Use the data
zb_write_reg(PAN_ID_1,data[1]);          // PAN
ID from EEPROM

zb_write_reg(PAN_ID_0,data[2]);
zb_write_reg(SHORT_ADDR_1,data[3]);    //
Device Address from EEPROM

zb_write_reg(SHORT_ADDR_0,data[4]);
EEPROM_write(data[1],0x00,0x00);
EEPROM_write(data[2],0x00,0x01);
EEPROM_write(data[3],0x00,0x02);
EEPROM_write(data[4],0x00,0x03);
// Send connected message
message[0] = 0x02;
success = zb_tx(0xA0,00,message,1,seq);
seq++;
// Tell Technical Readout
usart_printf("{ Connected as # ");
usart_putShort(data[4]);
usart_printf("} ");
delay_s(5);
}
```

```

        //--// Sleep cycle
        zb_sleep();
        PIC_sleep();
    }
    break;

//--// RCU
case 2:
    //--// Timer0 Init
    t0con = 10000111b; //
Initialize T0
    intcon |= 00100000b; // Enable the
T0 interrupt to occur @ tmr0if
    tmr0h = 46005/256; // Set
T0 to interrupt @ 1 Hz
    tmr0l = 46005%256;
    char tmr0count = 0; //
Reset the timer interrupt counter
    char sleep_timer = 0; // Fall asleep

//--// Set Device Addressing
    pan1 = 0xBA;
    pan0 = 0x5E;
    add1 = 0xC0;
    add0 = 0x00;
    zb_write_reg(PAN_ID_1,pan1); // PAN ID
0xBA5E
    zb_write_reg(PAN_ID_0,pan0);
    zb_write_reg(SHORT_ADDR_1,add1); // Device
Address 0xC000
    zb_write_reg(SHORT_ADDR_0,add0);

//--// Prepare to accept Current Windows from Base Station
    window_count=1; //
Number of connected windows
    current_window = 0; // Currently
addressed window
    for (i=0;i<10;i++){
        name[i][0] = 'E';
        name[i][1] = 'm';
        name[i][2] = 'p';
        name[i][3] = 't';
        name[i][4] = 'y';
        name[i][5] = ' ';
        name[i][6] = ' ';
        name[i][7] = ' ';
    }

```

```

        name[i][8] = ' ';
        name[i][9] = ' ';
        status[i] = 0;
    }
    //--// Request up to date window information
    data[0] = 0xFF;
    success = zb_tx(0xA0,0x00,data,1,seq);
// Are you there
    seq++;
    if (success) {
        // If a head station responds
        source_add1 = 0;
        while (source_add1 != 0xA0) {
// Listen for return byte from HEAD
            success = zb_rx(0);

            message_length = zb_read_fb(message);
            source_add1 = message[header_length-1];
// Source extraction
        }
        for (i=0; i<message_length; i++){
// Data extraction
            data[i] = message[header_length+i];
        }
        window_count = data[0];
// Returned byte is the window count
        for (i=0; i<window_count; i++) {
// Receive the window names
            source_add1 = 0;
            while (source_add1 != 0xA0) {
// Listen for byte from HEAD
                success = zb_rx(0);
                message_length =
zb_read_fb(message);
                source_add1 =
message[header_length-1]; // Source extraction
            }
            for (j=0; j<message_length; j++){
// Data extraction
                data[j] = message[header_length+j];
                name[i][j] = data[j];
                // byte string is the window name
            }
            name[i][message_length] = '\0';
// End of string character
        }
    }

```

```
    }
    name>window_count][0] = 'S';
    name>window_count][1] = 'o';
    name>window_count][2] = 'l';
    name>window_count][3] = 'a';
    name>window_count][4] = 'r';
    name>window_count][5] = 'S';
    name>window_count][6] = 'h';
    name>window_count][7] = 'a';
    name>window_count][8] = 'd';
    name>window_count][9] = 'e';
    name>window_count][10] = '\0';
    }

//--// Setup the LCD
    SPILCD_init();
    delay_ms(100);
    SPILCD_brightness(6);
    delay_ms(100);
    SPILCD_clear();
    delay_ms(100);
    SPILCD_putShort(window_count);
    SPILCD_printf(" windows.");
    delay_ms(100);
    SPILCD_home();
    delay_ms(100);
    spi_init(0);
    delay_ms(2);

//--// Main Loop
while(1) {
    //---// INTERRUPT CHECK ROUTINE
        if (tmr0is==1) {
            tmr0is=0;
            tmr0count++;
            sleep_timer++;
            if (sleep_timer>180){
                SPILCD_init();
                SPILCD_brightness(0);
                spi_init(0);
                delay_ms(100);
                zb_sleep();

while(!(RCU_up||RCU_down||RCU_open||RCU_close));
                zb_wake();
```

```

        delay_ms(100);
        SPILCD_init();
        delay_ms(100);
        SPILCD_brightness(6);
        delay_ms(100);
        SPILCD_clear();
        spi_init(0);
        sleep_timer=0;
    }
    if (tmr0count>60) {
        tmr0count = 0;
        // Get current windows again
        data[0] = 0xFF;
        success =
zb_tx(0xA0,0x00,data,1,seq);    // Are you there
        seq++;
        if (success) {
            // If a head station responds
                source_add1 = 0;
                while (source_add1
!= 0xA0) {    // Listen for return byte from HEAD
                    success =
zb_rx(0);

                message_length = zb_read_fb(message);
                    source_add1 =
message[header_length-1];    // Source extraction
                }
            for (i=0;
i<message_length; i++){    // Data extraction
                data[i] =
message[header_length+i];
            }
            window_count =
data[0];    // Returned byte is the window count
            for
            (i=0;i<window_count;i++) {    // Receive the window names
                source_add1 =
0;
                while
                (source_add1 != 0xA0) {    // Listen for byte from HEAD
                    success
= zb_rx(0);

                message_length = zb_read_fb(message);

```

```
source_add1 = message[header_length-1]; // Source extraction

                                }
                                for (j=0;
j<message_length; j++){          // Data extraction
                                data[j]
= message[header_length+j];
    name[i][j] = data[j];          // byte string is the window
name
                                }

    name[i][message_length] = '\0'; // End of string
character
                                }

    name[window_count][0] = 'S';
    name[window_count][1] = 'o';
    name[window_count][2] = 'l';
    name[window_count][3] = 'a';
    name[window_count][4] = 'r';
    name[window_count][5] = 'S';
    name[window_count][6] = 'h';
    name[window_count][7] = 'a';
    name[window_count][8] = 'd';
    name[window_count][9] = 'e';
    name[window_count][10] = '\0';

                                }
                                }
                                }

//---// INTERRUPT CHECK ROUTINE

//--// Check the Buttons
if (RCU_up) {
    current_window++;
}
```

```

        if (current_window > window_count)
{ current_window = 0;}
        sleep_timer = 0;
        }
    else if (RCU_down) {
        if (current_window == 0) { current_window =
window_count;}
        else { current_window--;}
        sleep_timer = 0;
        }
    else if (RCU_open) {
        status[current_window] = 1;
        sleep_timer = 0;
        }
    else if (RCU_close) {
        status[current_window] = 2;
        sleep_timer = 0;
        }

    //--// Pass along the most recent command
    success = 0; // Reset the
success flag to avoid mixups
    success = zb_rx(1000); // Listen for a
while
    if (success) {
        message_length = zb_read_fb(message);
        source_add0 = message[header_length-2]; //
Source extraction
        source_add1 = message[header_length-1];

        if
((source_add1 == 0xB0) && (status[source_add0])) {
            data[0] = status[source_add0];
            status[source_add0] = 0;
            success =
zb_tx(source_add1, source_add0, data, 1, seq);
            seq++;
        }
        else if
((source_add1 == 0xD0) && (status[window_count])) { // Solar Shade
            data[0] = status[window_count];
            status[window_count] = 0;
            while (RCU_open || RCU_close) {
                success = 0;
                success = zb_rx(1000);
                if (success) {

```

```

                                                                    zb_tx(0xD0,0x00,data,1,seq);
// 1 open, 2 close
                                                                    seq++;
                                                                    }
                                                                    }
                                                                    success = 0;
                                                                    i = 0;
                                                                    while((!success)&&(i<100)){ success =
zb_rx(1000);}
                                                                    data[0] = 0;
                                                                    for (i=0;i<10;i++) {
                                                                    success =
zb_tx(0xD0,0x00,data,1,seq); // 0 stop
                                                                    seq++;
                                                                    }
                                                                    }
                                                                    }

SPILCD_init();
delay_ms(50);
SPILCD_clear();
delay_ms(35);
for (i=0;i<10;i++){
    disp[i] = name[current_window][i];
}
disp[10] = '\0';
SPILCD_printf(disp);
delay_ms(10);
spi_init(0);
delay_ms(50 );
}

break;

/--// Product Testing
case 3:
    char thecount = 0+'a';
    while (1) {
        usart_putc(thecount);
        thecount++;
        delay_s(1);
        if (thecount>10+'a') {
            thecount = 0+'a';
        }
    }

break;
```



```
//--// Default Mode
    default:
        SPILCD_init();
        while(1){
            //SPILCD_brightness(5);
            //SPILCD_clear();
            SPILCD_printf("Catch");
            //SPILCD_putShort(window_count);
            //spi_init(0);
        }
        break;
    }
}

////////////////////////////////////
//    Function: interrupt
//    Purpose: The interrupt service routine
//    Input:  Void
//    Output: Void
////////////////////////////////////
void interrupt(void) {

    //////////////////////////////////////
    // IMPORTANT //////////////////////////////////////
    // Semiphores Needed:      tmr0is --> timer0 rollover interrupt
    // Ensure that these semiphores are globally defined before using
    //////////////////////////////////////

    if(tmr0if != 0) {
        tmr0is = true;           // Overflow on timer0
        tmr0if = 0;             // Set the semiphore
        tmr0h = 46005/256;      // Reset the flag
        tmr0l = 46005%256;      // Reset the timer0 counter to 46005
    }

    if(rcif != 0) {
        char c;                 // USART Received Character
        c = usart_foundc();     // Reset the flag by reading the buffer
        if (c==pc_attn) {
            rcis = true;        // If the message signal has been given
            usart_receive_disable(); // Set the semiphore
            delay_ms(400);
            usart_putc(0);
        }
        c = '\0';              // Reset c
    }
}
```

```
    }  
  
    return;  
}
```



```
////////////////////////////////////  
// Function: sleep_PIC  
// Purpose: puts the microcontroller into idle_primary mode  
// Inputs: Void  
// Outputs: Void  
////////////////////////////////////  
void PIC_sleep(void){  
    intcon |= 00100000b;           // Enable the T0 interrupt to occur @  
tmr0if                             // Set for idle mode on sleep command  
    // Set T0 to interrupt @ 1 Hz  
    tmr0h = 46005/256;  
    tmr0l = 46005%256;  
  
    oscon |= 10000000b;  
    // ASSUMING SCS = 00 (DEFAULT)  
  
    // Execute sleep command  
    nop();  
    nop();  
    nop();  
    sleep();  
  
    // Wake on interrupt  
    nop();  
    nop();  
    nop();  
    intcon &= 11011111b;         // Disable T0 interrupt  
    return;  
}
```

ZigBee Library: ZigBeelib.c

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      ZigBeelib.c                    //
//      SPI functions for ZigBee       //
////////////////////////////////////

// Function Prototypes are in ZigBeelib.h

// Global Volatile Bits
volatile bit zb_cs @PORTD.0;    // Chip Select (low true)
volatile bit zb_rs @PORTD.1;    // Reset (Low true)
volatile bit zb_slp@PORTD.2;    // Sleep Signal

////////////////////////////////////
//      Function: spi_init
//      Purpose:  Initializes MSSP for SPI
//      Inputs:   rate -> currently unused
//      Output:   Void
////////////////////////////////////
void spi_init(unsigned short rate)
{
    // Reset
    sspcon1 = 0b00000000;

    // SSP1STAT
    //bit7:  0: sample time, middle of period
    //bit6:  1: CKE clock select, (Words-0, Picture-1)
    //bit5-0: 0: I2C mode only, Default is 0
    sspstat = 0b01000000;

    // SSP1CON1
    //bit7:  0:  write collision, clear in software
```

```

//bit6: 0: receiver overflow, clear in software, avoid by always reading
buffer
//bit5: 1: mssp enable bit
//bit4: 0: CKP - idle state is 0 or 1
//bit3-0: 0010: Clock speed - Fos/64
sspcon1 = 0b00100010;
// Sets data in, data out, and clock ports
// BUT: must clear trisc.5 For data out
//      must clear trisc.3 For clock

// I/O Ports
/* SEL: D0 --> Output 0
Reset: D1 --> Output 0
SLP: D2 --> Output 0
IRQ: D3 --> Input 1
SCK: C3 --> Output 0
MISO: C4 --> Input 1
MOSI: C5 --> Output 0 */
trisc.5 = 0;
trisc.3 = 0;
trisd.0 = 0;
trisd.1 = 0;
trisd.2 = 0;
trisd.3 = 1;

// Set Default Output Values
zb_cs = 1; // Set chip select high
zb_slp = 0; // Set slp signal low
zb_rs = 1; // Set reset high
}

```

```

////////////////////////////////////
// Function: spi_byte
// Purpose: Sends a byte on MOSI and recieves a byte on MISO
// Inputs: data -> the byte to be sent on MOSI
// Output: The byte received on MISO
////////////////////////////////////
char spi_byte(char data)
{
    volatile bit sspif@PIR1.3; // Define a bit For interurpt flag
    char return_byte;
    // Collision Check not required yet
    sspif = 0; // Set interrupt flag to low
    sspbuf = data; // Write data to buffer to start trasmission
}

```

```

while(!sspif);           // Wait until interrupt flag goes high
return_byte = sspbuf;
return return_byte;     // Return the new contents of the buffer
}

/////////////////////////////////////////////////////////////////
//      Function:      zb_init
//      Purpose:       Initializes the zigbee card for transmission
//      Inputs:        channel
//      Output:        Void
/////////////////////////////////////////////////////////////////
void zb_init(char channel)
{
    char reg;

    // Reset the device
    zb_rs = 0;
    delay_ms(20);
    zb_rs = 1;
    delay_ms(20);

    zb_write_reg(IRQ_MASK,0b00000000); // Mask all interrupts
    reg = zb_read_reg(IRQ_STATUS);     // Read the interrupt
register to clear it
    zb_write_reg(PHY_CC_CCA,channel); // Set the
communication channel
    zb_state_trans(TRX_OFF,TRX_OFF); // Move ZigBee card
to clock state with command TRX_OFF
    zb_state_trans(PLL_ON,PLL_ON);    // Move to PLL-ON
state

    // Initialize for Extended Mode
    unsigned short i;                 // Loop Variable
    char read_reg;                    // Temporary Register
    char write_reg =0;                // Temporary Register
    for (i=0;i<4;i++) {               // Produce an 8-bit random
number from register 6
        read_reg = zb_read_reg(PHY_RSSI);
        switch (i){
            case 0:
                write_reg.0 = read_reg.6;
                write_reg.1 = read_reg.5;
            break;
            case 1:

```

```

        write_reg.2 = read_reg.6;
        write_reg.3 = read_reg.5;
    break;
    case 2:
        write_reg.4 = read_reg.6;
        write_reg.5 = read_reg.5;
    break;
    case 3:
        write_reg.6 = read_reg.6;
        write_reg.7 = read_reg.5;
    break;
    }
    nop();
    nop();
    nop();
}
zb_write_reg(0x2D,write_reg);           // Write the random number to 0x2D
write_reg = 01000111b;                  // Write the random number
to 0x2E <2:0>, keep bit 6 set
zb_write_reg(0x2E,write_reg);
}

```

```

////////////////////////////////////
//      Function: zb_read_reg
//      Purpose: Reads a register on the ZigBee card
//      Inputs:  add -> the register address to read from
//      Output:  The byte sitting in the register that was read
////////////////////////////////////
char zb_read_reg(char add)
{
    char ans;                               // Temporary 1-byte
variable
    zb_cs = 0;                               // Chip select low to begin
process
    ans = spi_byte((add&0x3F)|0x80); // Send command byte 10__add_
    ans = spi_byte(0);                 // Send dummy byte to
collect contents
    zb_cs = 1;                           // Chip select high to end
transmission
    return ans;
}

```

```
////////////////////////////////////  
//      Fucntion: zb_write_reg  
//      Purpose: Writes to a register on the ZigBee card  
//      Inputs:  add  -> Address of the register to write to  
//              data -> Byte to put in the register  
//      Output:      Void  
////////////////////////////////////  
void zb_write_reg(char add, char data)  
{  
    char ans;                                // Temporary 1-byte  
variable  
    zb_cs = 0;                               // Chip select low to begin  
transmission  
    ans = spi_byte((add&0x3F)|0xC0); // Send command byte 11__add_  
    ans = spi_byte (data);           // Send data to be written  
    zb_cs = 1;                         // Chip select high to end  
transmission  
}
```

```
////////////////////////////////////  
//      Function:    zb_write_fb  
//      Purpose:     Writes a 1-byte payload to the ZigBee card frame buffer  
//                  with the appropriate 802.15.4-2003 format  
//      Input:       data -> The data bytes to be included in the buffer  
//                  size -> The size in bytes of the data  
//                  seq -> The current seq value  
//                  add0 -> First byte of the address to send to  
//                  add1 -> Second byte of the address to send to  
//      Output:      The updated seq value  
////////////////////////////////////  
char zb_write_fb(char add1,char add0,char data[],char size,char seq)  
{  
    char ans;                                // Temporary 1-byte variable  
    int i;                                  // Loop variable  
  
    char my_pan_0 = zb_read_reg(PAN_ID_0);  
    char my_pan_1 = zb_read_reg(PAN_ID_1);  
    char my_add_0 = zb_read_reg(SHORT_ADDR_0);  
    char my_add_1 = zb_read_reg(SHORT_ADDR_1);  
  
    zb_cs = 0;                               // Chip select low to begin transmission  
    ans = spi_byte(0x60); // Send command byte 0110 0000  
    ans = spi_byte(13+size); // Send PHR (frame length) -> 13 + size of data
```

```

// Send PSDU
ans = spi_byte(0b00100001); // Send FCF(control bytes) 7654 3210 FEDC BA98
ans = spi_byte(0x88);      // 0010 0001 1000 1000

ans = spi_byte(seq);      // Send sequence field

ans = spi_byte(my_pan_0); // Destination Send addressing fields
ans = spi_byte(my_pan_1);
ans = spi_byte(add0);
ans = spi_byte(add1);

ans = spi_byte(my_pan_0); // Source Send addressing fields
ans = spi_byte(my_pan_1);
ans = spi_byte(my_add_0);
ans = spi_byte(my_add_1);

/* */                      // No Security Fields

for (i=0;i<size;i++)      // Send data bytes
{
    ans = spi_byte(data[i]);
}

/* */                      // Auto sends the 2-byte FCS

zb_cs = 1;                // Chip select high to end
transmission

seq++;                    // Increment seq
return seq;
}

```

```

////////////////////////////////////
//      Function: zb_read_fb
//      Purpose: Reads the ZigBee frame buffer assuming the format used
//                used in the function zb_write_fb (see above)
//      Input:   Pointer to the memory where frame buffer will be stored
//      Output:  The length of the framebuffer read out
////////////////////////////////////
char zb_read_fb(char* tmp)
{
    char length;          // Holds the contents of PHR field
    zb_cs = 0;           // Chip select low to begin transmission
    length = spi_byte(0x20); // Send command byte 0010 0000
}

```



```
length = spi_byte(0);          // Read and store the PHR
tmp[0] = length;

char ans;                      // Temporary variable
char i;                        // Loop counter variable
for (i=0; i<length; i++)
{
    ans = spi_byte(0);        // Read the next byte
    tmp[i+1] = ans;          // Store that byte
}

zb_cs = 1;                    // Chip select high to end
transmission
return length+1;
}

////////////////////////////////////
// Function:    zb_state_trans
// Purpose:     Moves the zb card into a new state
// Inputs:      state -> the state you want to move into
//              command -> the command to give to the register
// Output:      Void
////////////////////////////////////
void zb_state_trans(char state, char command)
{
    char reg;
    zb_write_reg(TRX_STATE,command);    // Move ZigBee card to PLL-
LOCK
    reg = zb_read_reg(TRX_STATUS);
    while((reg&0b00011111) != state)
    {
        reg = zb_read_reg(TRX_STATUS);
    }
}

////////////////////////////////////
// Function:    zb_tx
// Purpose:     Transmits a char string over ZigBee
// Inputs:      data -> the data to send over ZigBee
//              size -> the size of the data to send
//              seq -> the current seq value
```

```

//          add0 -> First byte of the address to send to
//          add1 -> Second byte of the address to send to
//   Output:      the incremented seq value
///////////////////////////////////////////////////////////////////
char zb_tx (char add1,char add0,char data[],char size,char seq)
{
    char reg;
    char success = 0;
    zb_state_trans(TX_ARET_ON,TX_ARET_ON);           // Move to Extended
TX state
    seq = zb_write_fb(add1,add0,data,size,seq); // Write the data into framebuffer
    zb_state_trans(BUSY_TX_ARET,TX_START);           // Begin the
transmission
    reg = zb_read_reg(TRX_STATUS);                   // Wait until transmit
is finished by returning from Busy
    while(reg != TX_ARET_ON){
        reg = zb_read_reg(TRX_STATUS);
    }
    reg = zb_read_reg(TRX_STATE);                     // Extended-mode
transmission result
    if ((reg&11110000b)==0) {
        // Reg 7:5 will be 0 on success
        success = 1;
    }

    zb_state_trans(PLL_ON,PLL_ON);                   // Move back to the
TRX_OFF state
    return success;
}

```

```

/////////////////////////////////////////////////////////////////
//   Function:    zb_rx
//   Purpose:     Receives a byte over ZigBee
//   Inputs:      Length to listen for (0 for infinite)
//   Output:      Whether a frame was received
///////////////////////////////////////////////////////////////////
char zb_rx (unsigned short length)
{
    char reg;
    char success = 0;

    zb_write_reg(IRQ_MASK,0b00101100);              // Enable interrupts 2,
3, and 5
}

```

```

    reg = zb_read_reg(IRQ_STATUS);           // Refresh the intrupt
buffer
    zb_state_trans(RX_AACK_ON,RX_AACK_ON);   // Move to the AACK
Receive state

    // On message detect, auto move to BUSY_RX_AACK state
    // IRQ_2: When RX begins
    // IRQ_3: When TRX ends
    bool IRQ3= 0;
    // Puts FCS in PHY_RSSI bit 7
    // IRQ_5: If address match
    // Moves back into RX_AACK_ON

    switch(length) {
        case 0:                               //
Unending listen
            while (!IRQ3) {
                reg = zb_read_reg(IRQ_STATUS);
                if (reg.3==1){
                    IRQ3=1;
                }
            }
            reg = zb_read_reg(TRX_STATUS);     // Wait until transmit
is finished by returning from Busy
            while(reg != RX_AACK_ON){
                reg = zb_read_reg(TRX_STATUS);
            }
            reg = zb_read_reg(TRX_STATE);     // Extended-mode
transmission result
            if (((reg&11100000)==0) | ((reg&11100000)==32)) {
                // Reg 7:5 will be 0 or 1 on success
                success = 1;
            }
            break;

        default:                               //
Limited listen
            unsigned short i;
            for (i=0;i<length;i++) {
                reg = zb_read_reg(IRQ_STATUS);
                if (reg.3==1){
                    reg = zb_read_reg(TRX_STATUS); // Wait
until transmit is finished by returning from Busy
                    while(reg != RX_AACK_ON){
                        reg = zb_read_reg(TRX_STATUS);
                    }
                }
            }
    }

```

```
    }  
    reg = zb_read_reg(TRX_STATE);          //  
Extended-mode transmission result  
    if (((reg&11100000)==0) | ((reg&11100000)==32))  
{  
        // Reg 7:5 will be 0 or 1 on success  
        success = 1;  
    }  
    break;  
}  
    break;  
}  
  
zb_state_trans(PLL_ON,PLL_ON);  
zb_write_reg(IRQ_MASK,0x00);  
return success;  
}
```

```
////////////////////////////////////  
//    Function:    zb_sleep  
//    Purpose:     Puts the Zigbee card to sleep  
//    Inputs:      Void  
//    Output:      Void  
////////////////////////////////////  
void zb_sleep(void)  
{  
  
    char reg;          // Read current state  
    reg = zb_read_reg(TRX_STATUS);  
    if((reg&0b00011111) != TRX_OFF)      // If not in TRX_OFF...  
    {  
        zb_state_trans(TRX_OFF, TRX_OFF); // ...Move to TRX_OFF  
        reg = zb_read_reg(TRX_STATUS);  
    }  
    zb_slp = 1; // Once in TRX_OFF, go to sleep with sleep signal port  
    delay_ms(1);  
}
```

```
////////////////////////////////////  
//    Function:    zb_wake  
//    Purpose:     Wakes up the Zigbee card  
//    Inputs:      Void
```

```
//      Output:      Void
//      //////////////////////////////////////////////////
void zb_wake(void)
{
    zb_slp = 0;
    delay_ms(1);
}

//      //////////////////////////////////////////////////
//      Function:    zb_broadcast_fb
//      Purpose:     Writes a broadcast payload to the ZigBee frame buffer
//                  with the appropriate 802.15.4-2003 format
//      Input:      data -> The data bytes to be included in the buffer
//                  size -> The size in bytes of the data
//                  seq -> The current seq value
//      Output:     The updated seq value
//      //////////////////////////////////////////////////
char zb_broadcast_fb(char data[],char size,char seq)
{
    char ans;                // Temporary 1-byte variable
    int i;                   // Loop variable

    char my_pan_0 = zb_read_reg(PAN_ID_0);
    char my_pan_1 = zb_read_reg(PAN_ID_1);
    char my_add_0 = zb_read_reg(SHORT_ADDR_0);
    char my_add_1 = zb_read_reg(SHORT_ADDR_1);

    zb_cs = 0;              // Chip select low to begin transmission
    ans = spi_byte(0x60); // Send command byte 0110 0000
    ans = spi_byte(13+size); // Send PHR (frame length) -> 13 + size of data

    // Send PSDU
    ans = spi_byte(0x01);   // Send FCF(control bytes) 7654 3210 FEDC BA98
    ans = spi_byte(0x88);   // 0000 0001 1000 1000

    ans = spi_byte(seq);    // Send sequence field

    ans = spi_byte(0xFF);   // Destination Send addressing fields
    ans = spi_byte(0xFF);
    ans = spi_byte(0xFF);
    ans = spi_byte(0xFF);

    ans = spi_byte(my_pan_0); // Source Send addressing fields
    ans = spi_byte(my_pan_1);
    ans = spi_byte(my_add_0);
```

```

ans = spi_byte(my_add_1);

/* */                               // No Security Fields

for (i=0;i<size;i++)                // Send data bytes
{
    ans = spi_byte(data[i]);
}

/* */                               // Auto sends the 2-byte FCS

zb_cs = 1;                           // Chip select high to end
transmission

seq++;                               // Increment seq
return seq;
}

////////////////////////////////////
//   Function:    zb_broadcast
//   Purpose:     Transmits a char string over ZigBee
//   Inputs:      data -> the data to send over ZigBee
//                size -> the size of the data to send
//                seq -> the current seq value
//   Output:      the incremented seq value
////////////////////////////////////
char zb_broadcast (char data[],char size,char seq)
{
    char reg;

    zb_state_trans(TX_ARET_ON,TX_ARET_ON);           // Move to Extended
TX state
    seq = zb_broadcast_fb(data,size,seq);           // Write the data into framebuffer

    zb_state_trans(BUSY_TX_ARET,TX_START);         // Begin the transmission
    reg = zb_read_reg(TRX_STATUS);                 // Wait until transmit is
finished by returning from Busy
    while(reg != TX_ARET_ON){
        reg = zb_read_reg(TRX_STATUS);
    }

    /*
    reg = zb_read_reg(TRX_STATE);                   // Extended-mode
transmission result
    if (((reg&11100000)==0) | ((reg&11100000)==32)) {
        // Reg 7:5 will be 0 or 1 on success

```

```
        success = 1;
    }
    */
    zb_state_trans(PLL_ON,PLL_ON);           // Move back to the
TRX_OFF state
    return seq;
}
```

ZigBee Library: ZigBeelib.h

```
#ifndef _ZIGBEELIB_H_
#define _ZIGBEELIB_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      ZigBeelib.h                    //
//      SPI function headers for ZigBee //
////////////////////////////////////

// Constants
#define header_length 12           // Length of the message before data

// Structures
struct window {
    char status;                  // Window status
    char name[16];                // Name of the window
};

// Registers
#define TRX_STATUS                0x01
#define TRX_STATE                 0x02
#define TRX_CTRL_0                0x03
#define TRX_CTRL_1                0x04
#define PHY_TX_PWR                0x05
#define PHY_RSSI                 0x06
#define PHY_ED_LEVEL             0x07
#define PHY_CC_CCA               0x08
#define CCA_THRES                 0x09
#define RX_CTRL                  0x0A
#define SFD_VALUE                0x0B
#define TRX_CTRL_2               0x0C
#define ANT_DIV                   0x0D
#define IRQ_MASK                 0x0E // Set to enable an interrupt
#define IRQ_STATUS               0x0F // Check to see if interrupt happened
#define VREG_CTRL                0x10
#define BATMON                   0x11
#define XOSC_CTRL                0x12
#define SHORT_ADDR_0             0x20
#define SHORT_ADDR_1             0x21
#define PAN_ID_0                 0x22
#define PAN_ID_1                 0x23
```



```
// States
#define P_ON 0x00
#define BUSY_RX 0x01
#define BUSY_TX 0x02
#define RX_ON 0x06
#define TRX_OFF 0x08
#define PLL_ON 0x09
#define SLEEP 0x0F
#define BUSY_RX_AACK 0x11
#define BUSY_TX_ARET 0x12
#define RX_AACK_ON 0x16
#define TX_ARET_ON 0x19
#define RX_ON_NOCLK 0x1C
#define RX_AACK_ON_NOCLOCK 0x1D
#define BUSY_RX_AACK_NOCLK 0x1E
#define STATE_TX_IN_PROGRESS 0x1F

// State Commands
#define NOP 0x00
#define TX_START 0x02
#define FORCE_TX_OFF 0x03
#define FORCE_PLL_ON 0x04
#define RX_ON 0x06
#define TRX_OFF 0x08
#define PLL_ON 0x09
#define RX_AACK_ON 0x16
#define TX_ARET_ON 0x19

// ZigBee SPI1 Function Prototypes
// See ZigBeelib.c for actual functions
void spi_init (unsigned short rate);
    // Initializes the SPI
char spi_byte(char value);
    // Sends and receives a byte by SPI
void zb_init(char channel);
    // Initializes the Atmel ZigBee card
char zb_read_reg(char add);
    // Reads a register on ZigBee card
void zb_write_reg(char add, char data);
    // Writes a register on the ZigBee card
char zb_read_fb(char* tmp);
    // Reads the framebuffer on the ZigBee card
```

```
char zb_write_fb(char add1, char add0,char data[],char size,char seq);    // Writes to the
framebuffer on the ZigBee card
void zb_state_trans(char state, char command);
    // Causes the ZigBee card to change states
char zb_tx (char add1, char add0, char data[], char size, char seq); // Transmits a packet
over ZigBee
char zb_rx (unsigned short length);
    // Receives a data packet
void zb_sleep(void);
    // Puts the ZigBee chip to sleep
void zb_wake(void);
    // Wakes the ZigBee chip from sleep
char zb_broadcast_fb(char data[],char size,char seq);
    // Prepares a broadcast paket for ZigBee
char zb_broadcast(char data[], char size, char seq); //
Transmits a broadcast packet over ZigBee

#endif // _ZIGBEELIB_H_
```

SPI LCD Library: SPILCDlib.c

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      LCDlib_SPI.c                  //
//      SPI functions For LCD         //
////////////////////////////////////

// Global Volatile Bits
volatile bit lcd_cs@LATC.0;           // Chip Select (low true)

////////////////////////////////////
//      Function: SPILCD_init        //
//      Purpose: initializes the SPI LCD
//      Input:  Void
//      Output: Void
////////////////////////////////////
void SPILCD_init(void){

    // Reset
    sspcon1 = 0b00000000;

    // SSP1STAT
    //bit7: 0: sample time, middle of period
    //bit6: 0: CKE clock select (rising)
    //bit5-0: 0: I2C mode only, Default is 0
    sspstat = 0b00000000;

    // SSP1CON1
    //bit7: 0: write collision, clear in software
    //bit6: 0: receiver overflow, clear in software, avoid by always reading
buffer
    //bit5: 1: mssp1 enable bit
    //bit4: 1: CKP - idle state is 0 or 1
    //bit3-0: 0010: Clock speed - Fos/64
```

```

//          0011: TMR2/2
sspcon1 = 0b00110010;
// Sets data in, data out, and clock ports
// BUT: must clear trisc.5 For data out
//          must clear trisc.3 For clock

// I/O Ports
/* SCK:  C3 --> Output    0
   MISO: C4 --> Input     1
   MOSI: C5 --> Output    0
   Select:C0 --> Output 0 */
trisc.5 = 0;
trisc.3 = 0;
trisc.0 = 0;

// Set Default Output Values
lcd_cs = 1;          // Set chip select high
}

////////////////////////////////////
//   Function: SPILCD_putc
//   Purpose:  Puts a char to the SPI LCD
//   Input:    Void
//   Output:   c -> the string to put to the screen
////////////////////////////////////
void SPILCD_putc(char c){
    char ans;
    delay_ms(1);
    lcd_cs = 0;
    delay_ms(1);
    ans = spi_byte(c);
    delay_ms(1);
    lcd_cs = 1;
    delay_ms(1);
    return;
}

////////////////////////////////////
//   Function: SPILCD_putShort
//   Purpose:  prints an unsigned short to the lcd in decimal form
//   Input:    dec -> that value to be printed
//   Output:   Void
////////////////////////////////////
```

```
////////////////////////////////////  
void SPILCD_putShort(unsigned short dec) {  
    unsigned short dec1;      // Ten Thousands place  
    unsigned short dec2;      // Thousands place  
    unsigned short dec3;      // Hundreds place  
    unsigned short dec4;      // Tens place  
    unsigned short dec5;      // Ones place  
  
    dec1 = dec/10000;          // Get the Ten Thousands place with division  
    dec = dec%10000;          // Calculate how much is left  
    dec2 = dec/1000;          // Get the Thousands place with division  
    dec = dec%1000;          // Calculate how much is left  
    dec3 = dec/100;           // Get the Hundreds place with  
division  
    dec = dec%100;            // Calculate how much is left  
    dec4 = dec/10;            // Get the Tens place with division  
    dec5 = dec%10;            // The Ones place is how much is left  
  
    if(dec1 != 0)              // If there is a Ten Thousands place, print it  
        SPILCD_putc(dec1 + '0');  
    if(dec2 != 0)              // If there is a Thousands place, print it  
        SPILCD_putc(dec2 + '0');  
    if(dec3 != 0)              // If there is a Hundreds place, print it  
        SPILCD_putc(dec3 + '0');  
    if(dec4 != 0)              // If there is a Tens place, print it  
        SPILCD_putc(dec4 + '0');  
    SPILCD_putc(dec5 + '0');    // Print the one's place  
}
```

```
////////////////////////////////////  
//    Function: SPILCD_printf  
//    Purpose: Puts a string to the SPI LCD  
// Input:  c -> the char to put to the screen  
// Output: Void  
////////////////////////////////////  
void SPILCD_printf(char c[]){  
    int i;  
    for(i=0; c[i] != '\0'; i++)    // Loop through each character in the array  
        SPILCD_putc(c[i]);        // Put the character to the terminal  
        delay_ms(1);  
    return;  
}
```

```
////////////////////////////////////  
// Function: SPILCD_cmd  
// Purpose: Sends a command to the screen via SPI  
// Input: The command length and the command  
// Output: Void  
////////////////////////////////////  
void SPILCD_cmd(char length, char* cmd) {  
    char ans;  
    int i;  
    SPILCD_putc(0xFE);  
    for (i=0;i<length;i++) {  
        SPILCD_putc(cmd[i]);  
        delay_ms(1);  
    }  
}
```

```
////////////////////////////////////  
// Function: SPILCD_clear  
// Purpose: Clears the screen and returns cursor  
// Input: Void  
// Output: Void  
////////////////////////////////////  
void SPILCD_clear(void) {  
    char cmd[1];  
    cmd[0] = 0x51;  
    SPILCD_cmd(1,cmd);  
}
```

```
////////////////////////////////////  
// Function: SPILCD_newline  
// Purpose: Moves the LCD cursor to line 2  
// Input: Void  
// Output: Void  
////////////////////////////////////  
void SPILCD_newline(void) {  
    char cmd[2];  
    cmd[0] = 0x45;  
    cmd[1] = 0x40;  
    SPILCD_cmd(2,cmd);  
}
```

```
////////////////////////////////////  
// Function: SPILCD_home  
// Purpose: Moves the LCD cursor to line 1  
// Input: Void  
// Output: Void  
////////////////////////////////////  
void SPILCD_home(void) {  
    char cmd[1];  
    cmd[0] = 0x46;  
    SPILCD_cmd(1,cmd);  
}  
  
////////////////////////////////////  
// Function: SPILCD_brightness  
// Purpose: Sets the LCD backlight brightness  
// Input: The brightness setting 1-8  
// Output: Void  
////////////////////////////////////  
void SPILCD_brightness(char brightness) {  
    char cmd[2];  
    cmd[0] = 0x53;  
    cmd[1] = brightness;  
    SPILCD_cmd(2,cmd);  
}
```

SPI LCD Library: SPILCDlib.h

```
#ifndef _LCDLIB_SPI_H_
#define _LCDLIB_SPI_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      LCDlib_SPI.h                //
//      SPI function headers for LCD//
////////////////////////////////////

// Function Prototypes
void SPILCD_init(void);           // Initializes the SPI
LCD
void SPILCD_putc(char c);         // Puts a char to the SPI LCD
void SPILCD_putShort(unsigned short dec); // Prints a short to the LCD
void SPILCD_printf(char c[]);    // Prints a string to the screen
void SPILCD_cmd(char length, char cmd[]); // Sends a command to the LCD
void SPILCD_clear(void);         // Clears the screen and
returns cursor
void SPILCD_newline(void);       // Moves the cursor to the
beginning of line 2
void SPILCD_home (void);        // Moves the cursor to the
beginning of line 1
void SPILCD_brightness(char brightness); // Sets the LCD brightness
// Utilizes spi1_byte from ZigBeelib.c to send and receive spi bytes

#endif // _LCDLIB_SPI_H_
```


Real Time Clock Library: RTClib.c

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTClib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      RTClib.c                       //
// Functions for the RTC               //
////////////////////////////////////

// Global Volatile Bits
volatile bit rtc_en@LATD.7;           // Chip Enable

// Function Prototypes in RTClib.h

////////////////////////////////////
//      Function: RTC_init
//      Purpose: initializes the SPI RTC
// Inputs:  Void
// Output:  Void
////////////////////////////////////
void RTC_init(void){

    // Reset
    sspcon1 = 0b00000000;

    // SSPSTAT
    //bit7: 0: sample time, middle of period
    //bit6: 0: CKE clock select (Falling if CKP is 0)
    //bit5-0: 0: I2C mode only, Default is 0
    sspstat = 0b00000000;

    // SSPCON1
    //bit7: 0: write collision, clear in software
    //bit6: 0: receiver overflow, clear in software, avoid by always reading
buffer
    //bit5: 1: mssp1 enable bit
```

```

//bit4: 0: CKP - idle state is 0 or 1
//bit3-0: 0010: Clock speed - Fos/64
sspcon1 = 0b00100010;
// Sets data in, data out, and clock ports
// BUT: must clear trisc.5 For data out
//      must clear trisc.3 For clock

// I/O Ports
/* Enable:D7 --> Output    0
   SCK: C3 --> Output      0
   MISO: C4 --> Input      1
   MOSI: C5 --> Output     0    */
trisc &= 11010111b;
trisd.7 = 0;

// Set Default Output Values
rtc_en = 0;          // Set chip select high

// Set up the initial parameters
// Control Register
// Bit 7:      Enable Oscillator (Low True) 0
// Bit 6:      Write Protect                1
// Bit 5-3:    Reserved                    000
// Bit 2:      Int Con (Use both Int Pins?) 1
// Bit 1:      Alarm Enable 1              0
// Bit 0:      Alarm Enable 0              0

char ans;
ans = RTC_reg(0x8F,0b01000100);

}

////////////////////////////////////
//      Function: RTC_reg
//      Purpose: Writes or reads an RTC register
//      Input:   Address to access, Byte to write
//      Output:  Byte read
////////////////////////////////////
char RTC_reg(char add, char input){
    char output = 0;
    rtc_en = 1;          // Set the chip enable low
    output = spi_byte(add); // Send an spi byte with address
    output = spi_byte(input); // Send an spi byte with the data
    rtc_en = 0;          // Set the chip enable low
    return output;
}

```

```
////////////////////////////////////
// Function: RTC_set
// Purpose: Sets the current time to the RTC
// Inputs: Pointer to a time structure
// Outputs: Void
////////////////////////////////////
void RTC_set(struct time* tp) {
    char i;           // Loop variable
    char ans;        // Temporary register placeholder
    char time[7];    // The time information to be sent by SPI

    time[0] = ((tp->sec10 <<4) & 0b01110000) | (tp->sec1 & 0b00001111); // See
datasheet
    time[1] = ((tp->min10 <<4) & 0b01110000) | (tp->min1 & 0b00001111);
    time[2] = ((tp->hr10 <<4) & 0b00110000) | (tp->hr1 & 0b00001111);
    time[3] = tp->day & 0b00001111;
    time[4] = ((tp->date10<<4) & 0b00110000) | (tp->date1 & 0b00001111);
    time[5] = ((tp->mon10 <<4) & 0b00110000) | (tp->mon1 & 0b00001111);
    time[6] = ((tp->yr10 <<4) & 0b11110000) | (tp->yr1 & 0b00001111);

    ans = RTC_reg(0x8F,0b00000100); // Disable write protect

    rtc_en = 1;                       // Send time bytes over SPI
(Burst Mode)
    ans = spi_byte(0x80);               // First address to write to
    for (i=0;i<7;i++) {
        ans = spi_byte(time[i]);      // Write the 7 bytes (address inc.
automatically)
    }
    rtc_en = 0;                       // End burst mode

    ans = RTC_reg(0x8F,0b01000100); // Enable write protect
    return;
}
```

```
////////////////////////////////////
// Function: RTC_get
// Purpose: Gets the current time from the RTC
// Input: Pointer to a time structure
// Output: Void
////////////////////////////////////
void RTC_get(struct time* tp) {
```

```
char tmp;           // Temporary storage variable
struct time t;     // Structure to hold the output

tmp = RTC_reg(0x00,0);           // Read the seconds
register
tp->sec10 = (tmp>>4) & 0b00000111; // Decode the 10s place
tp->sec1  = tmp & 0b00001111;     // Decode the 1s place
tp->sec   = 10*tp->sec10 + tp->sec1; // The total seconds

tmp = RTC_reg(0x01,0);           // Read the minutes
register
tp->min10 = (tmp>>4) & 0b00000111; // Decode the 10s place
tp->min1  = tmp & 0b00001111;     // Decode the 1s place
tp->min   = 10*tp->min10 + tp->min1; // The total minutes

tmp = RTC_reg(0x02,0);           // Read the hours
register
tp->hr10  = (tmp>>4) & 0b00000011; // Decode the 10s place
tp->hr1   = tmp & 0b00001111;     // Decode the 1s place
tp->hr    = 10*tp->hr10 + tp->hr1;  // The total hour

tmp = RTC_reg(0x03,0);           // Read the days
register
tp->day   = tmp & 0b00001111;     // Decode the 1s place

tmp = RTC_reg(0x04,0);           // Read the date
register
tp->date10 = (tmp>>4) & 0b00000011; // Decode the 10s place
tp->date1  = tmp & 0b00001111;     // Decode the 1s place
tp->date   = 10*tp->date10 + tp->date1; // The total date

tmp = RTC_reg(0x05,0);           // Read the month
register
tp->mon10 = (tmp>>4) & 0b00000011; // Decode the 10s place
tp->mon1  = tmp & 0b00001111;     // Decode the 1s place

tp->mon   = 10*tp->mon10 + tp->mon1; // The total month

tmp = RTC_reg(0x06,0);           // Read the year
register
tp->yr10  = (tmp>>4) & 0b00001111; // Decode the 10s place
tp->yr1   = tmp & 0b00001111;     // Decode the 1s place
tp->yr    = 10*tp->yr10 + tp->yr1;  // The total year
return;
}
```

```
////////////////////////////////////  
// Function: RTC_print_terminal  
// Purpose: Prints a time to the screen through usart2  
// Input: A Pointer to a time structure  
// Output: Void  
////////////////////////////////////  
void RTC_print_terminal(struct time* tp) {  
    usart_printf("\n\r");  
    usart_putShort(tp->hr10); // Put the hours to the screen  
    usart_putShort(tp->hr1 ); // Put the hours to the screen  
    usart_printf(":");  
    usart_putShort(tp->min10); // Put the minutes to the screen  
    usart_putShort(tp->min1 ); // Put the minutes to the screen  
    usart_putc(':');  
    usart_putShort(tp->sec10); // Put the seconds to the screen  
    usart_putShort(tp->sec1 ); // Put the seconds to the screen  
    usart_printf(" ");  
  
    switch (tp->day) { // Put the day of the week to the screen  
        case 1:  
            usart_printf("Sunday ");  
            break;  
        case 2:  
            usart_printf("Monday ");  
            break;  
        case 3:  
            usart_printf("Tuesday ");  
            break;  
        case 4:  
            usart_printf("Wednesday ");  
            break;  
        case 5:  
            usart_printf("Thursday ");  
            break;  
        case 6:  
            usart_printf("Friday ");  
            break;  
        case 7:  
            usart_printf("Saturday ");  
            break;  
        default:  
            usart_printf("Error ");  
            break;  
    }  
}
```

```
    }  
  
    switch (tp->mon) {          // Put the month to the screen  
        case 1:  
            usart_printf("January ");  
            break;  
        case 2:  
            usart_printf("February ");  
            break;  
        case 3:  
            usart_printf("March ");  
            break;  
        case 4:  
            usart_printf("April ");  
            break;  
        case 5:  
            usart_printf("May ");  
            break;  
        case 6:  
            usart_printf("June ");  
            break;  
        case 7:  
            usart_printf("July ");  
            break;  
        case 8:  
            usart_printf("August ");  
            break;  
        case 9:  
            usart_printf("September ");  
            break;  
        case 10:  
            usart_printf("October ");  
            break;  
        case 11:  
            usart_printf("November ");  
            break;  
        case 12:  
            usart_printf("December ");  
            break;  
        default:  
            usart_printf("Error ");  
            break;  
    }  
  
    usart_putShort(tp->date10); // Put the date to the screen  
    usart_putShort(tp->date1 ); // Put the date to the screen
```

```
    usart_printf(", 20");
    usart_putShort(tp->yr10);    // Put the year to the screen
    usart_putShort(tp->yr1 );    // Put the year to the screen
}

////////////////////////////////////
// Function: RTC_print_lcd
// Purpose: Prints a time to the lcd through spi
// Input:   A Pointer to a time structure
// Output:  Void
////////////////////////////////////
void RTC_print_lcd(struct time* tp) {
    SPILCD_clear();
    SPILCD_putShort(tp->hr10); // Put the hours to the screen
    SPILCD_putShort(tp->hr1 ); // Put the hours to the screen
    SPILCD_printf(":");
    SPILCD_putShort(tp->min10);    // Put the minutes to the screen
    SPILCD_putShort(tp->min1 );    // Put the minutes to the screen
    SPILCD_putc(':');
    SPILCD_putShort(tp->sec10);    // Put the seconds to the screen
    SPILCD_putShort(tp->sec1 );    // Put the seconds to the screen
    SPILCD_newline();

    switch (tp->day) {            // Put the day of the week to the screen
        case 1:
            SPILCD_printf("Sun ");
            break;
        case 2:
            SPILCD_printf("Mon ");
            break;
        case 3:
            SPILCD_printf("Tues ");
            break;
        case 4:
            SPILCD_printf("Wed ");
            break;
        case 5:
            SPILCD_printf("Thur ");
            break;
        case 6:
            SPILCD_printf("Fri ");
            break;
        case 7:
            SPILCD_printf("Sat ");
            break;
    }
}
```

```
    default:
        SPILCD_printf("Err ");
        break;

    }

switch (tp->mon) {           // Put the month to the screen
    case 1:
        SPILCD_printf("Jan");
        break;
    case 2:
        SPILCD_printf("Feb ");
        break;
    case 3:
        SPILCD_printf("Mar ");
        break;
    case 4:
        SPILCD_printf("Apr ");
        break;
    case 5:
        SPILCD_printf("May ");
        break;
    case 6:
        SPILCD_printf("June ");
        break;
    case 7:
        SPILCD_printf("July ");
        break;
    case 8:
        SPILCD_printf("Aug ");
        break;
    case 9:
        SPILCD_printf("Sept ");
        break;
    case 10:
        SPILCD_printf("Oct ");
        break;
    case 11:
        SPILCD_printf("Nov ");
        break;
    case 12:
        SPILCD_printf("Dec ");
        break;
    default:
        SPILCD_printf("Err ");
        break;
}
```



```
    }  
  
    SPILCD_putShort(tp->date10); // Put the date to the screen  
    SPILCD_putShort(tp->date1 ); // Put the date to the screen  
    SPILCD_printf(", 20");  
    SPILCD_putShort(tp->yr10); // Put the year to the screen  
    SPILCD_putShort(tp->yr1 ); // Put the year to the screen  
}
```

Real Time Clock Library: RTClib.h

```
#ifndef _RTCLIB_H_
#define _RTCLIB_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      LCDlib_SPI.h                //
//      SPI function headers for LCD//
////////////////////////////////////

// Structures
struct time {
    char sec10;      // 10s place of seconds
    char sec1;      // 1s place of seconds
    char sec;       // Decimal form of seconds

    char min10;     // 10s place of minutes
    char min1;     // 1s place of minutes
    char min;      // Decimal form of minutes

    char hr10;     // 10s place of hours
    char hr1;     // 1s place of hours
    char hr;      // Decimal form of hours

    char day;      // Day of the week (Sun=1)

    char date10;   // 10s place of date
    char date1;   // 1s place of the date
    char date;    // Decimal form of hours

    char mon10;   // 10s place of the month
    char mon1;   // 1s place of the month
    char mon;    // Decimal form of month

    char yr10;   // 10s place of the year
    char yr1;   // 1s place of the year
    char yr;    // Decimal form of year
};

// Function Prototypes
void RTC_init(void);           // Initializes the RTC
char RTC_reg(char add, char input); // Writes or read a RTC register
void RTC_set(struct time *tp); // Sets a time to the RTC
```

```
void RTC_get(struct time *tp); // Gets the time from the RTC
void RTC_print_terminal(struct time *tp); // Prints a time to the screen
void RTC_print_lcd(struct time *tp); // Prints a time to the screen

#endif // _RTCLIB_H_
```

ADCON Library: ADlib.c

```

#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      ADlib.c                        //
// Functions for A/D conversion        //
////////////////////////////////////

// Function Prototypes in ADlib.h

////////////////////////////////////
//      Function: ad_init
//      Purpose: Configures the A/D converter module
//      Input: the channel you want to use
//      Output: Void
////////////////////////////////////
void ad_init(char channel){
    char temp;    // Temporary value

    //ADCON1:
        // bit 7-6: unimplemented          00      00
        // bit 5-4: voltage ref config      00      Vdd-Vss
        // bit 3-0: A/D port config        1101    AN0, AN1 Analog;
Else digital
    adcon1 = 0b00001101;

    // AD0 PORTA      bit0
    // AD1 PORTA      bit1
    // AD2 PORTA      bit2
    trisa.0 = 1;
    trisa.1 = 1;

    //ADCON0:
        // bit 7-6: unimplemented          00
        // bit 5-2: analog channel select  0000    AN0

```

```

//
0001  AN1
// bit 1: A/D GO/DONE (0=idle, 1=in prog)      0
// bit 0: A/D enable (0=disabled, 1=enabled) 0      Disabled
adcon0 = (channel<<2) & 00111100b;

//ADCON2:
//bit 7: A/D result format                      0      Left-
Justified
//bit 6: unimplemented                          0
//bit 5-3: A/D acquisition time                 010      4
oscillations of the A/D clock = 6.4us > 2us (arbitrary)
//bit 2-0: A/D conversion select                010      32*Tosc =
1.6us => See data sheet
adcon2 = 0b00010010;

}

```

```

////////////////////////////////////
//      Function: ad_conv
//      Purpose: A/D conversion
//      Input:   Void
//      Output:  Unsigned short converted value
////////////////////////////////////
char ad_conv(){
    char store[30];
    char i;
    for(i=0;i<30;i++) {
        volatile bit godone @ADCON0.1; // GO/DONE bit

        adcon0 |= 0b00000001;           //turn on A/D module
        delay_us(10);                   // Wait an aquisition time

        godone = 1;                      // starts the
conversion process
        while (godone);                  // wait until
conversion bit is idle

        store[i] = adresh;
        //unsigned short outl = adresl;
        //unsigned short out = ((outh<<8) & (0b1111111100000000)) + outl;

        adcon0 &= 0b11111110;           //turn off A/D module

```

```
    }  
  
    long sum = 0;  
    for (i=0;i<30;i++) {  
        sum = sum + store[i];  
    }  
    char out = sum/30;  
    return out;  
}
```

ADCON Library: ADlib.h

```
#ifndef _ADLIB_H_
#define _ADLIB_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      ADlib_SPI.h                 //
//      SPI function headers for AD //
////////////////////////////////////

// Function Prototypes
void ad_init(char channel);          // Sets up the A/D for a particular channel
char ad_conv();                     // Performs A/D conversion

#endif // _ADLIB_H_
```

Serial Library: usartlib.c

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      usartlib.c                      //
// Serial functions for usart          //
////////////////////////////////////

// Function Prototypes are in usartlib.h

// Global Volatile Bits

////////////////////////////////////
//      Function: usart_init
//      Purpose: initializes usart
//      Input:  rate -> Must be 57600 for stable operation
//      Output: Void
////////////////////////////////////
void usart_init(unsigned short rate)
{
    trisc |= 11000000b;      // Set TX and Receive pins for eusart
    baudcon = 00000000b;    // 8-bit baud rate

    if (rate == 57600)
        spbrg = 21;        // Set correct spbrg2 correctly For 57.6k rate
    else
        spbrg = (1250000/rate)-1; // Otherwise, attempt to find the "n value" as
best as possible

// THIS CODE HAS AN
OVERFLOW ERROR -> DON'T USE

/**/ //rcsta2 default is 00000000b
//rcsta2 = 10000000b;
//bit 7: enable serial port - 1
//bit 6: 8 bit reception - 0
//bit 5: N/A - x
//bit 4: enable receiver - 0
```



```
//bit 3: N/A          - x
//bit 2: No framing error. - x
//bit 1: No overrun error. - x
//bit 0: N/A          - x
rcsta = 10000000b;    // Enable serial port

/**/ //txsta default is 00000010b
//txsta = 00100100b;
//bit 7: x
//bit 6: 8-bit transmit => 0
//bit 5: transmit enable => 1
//bit 4: asynch => 0
//bit 3: not "sync break on transmission completed" => 0
//bit 2: high speed => 1
//bit 1: "TSRx empty" => x
//bit 0: Parity bit => x
txsta = 00100100b;    // High speed baud rate mode
}

////////////////////////////////////
//      Function: usart_putc
//      Purpose: puts a character to terminal via usart
// Input:  c -> character to be typed (ascii)
// Output: Void
////////////////////////////////////
void usart_putc(char c)
{
    volatile bit transmitReady@PIR1.4; // transmission signal bit
    while(!transmitReady);             // Wait until usart is ready
    (bit goes high)
    txreg = c;                          // Put the data to be
    written into the Register
}

////////////////////////////////////
//      Function: usart_getc
//      Purpose: gets a character from terminal via usart
// Input:  Void
// Output: The character captured from the terminal
////////////////////////////////////
char usart_getc(void)
{
```

```
char value; // Allocate memory
For a character
volatile bit receiveReady@PIR1.5; // Monitor receive signal flag
unsigned long ii = 0;
char error = 0;
rcsta |= 00010000b; // Enable the receiver
while(!receiveReady){ // Wait until usart2 is ready
(bit goes high)
    if (ii>750000) {
        error = 1;
        break;
    }
    ii++;
}
value = rcreg; // Get that value in the buffer
rcsta &= 11101111b; // Disable the receiver to
prevent overflow
if (error) {value = 0;}
return value; // Return the character
}
```

```
////////////////////////////////////
// Function: usart_receive_enable
// Purpose: Enables the receiver for character finding
// Input: Void
// Output: Void
////////////////////////////////////
void usart_receive_enable(void) {
    pie1 |= 00100000b; // Enable receiver interrupts
    rcsta |= 00010000b; // Enable the receiver
    return;
}
```

```
////////////////////////////////////
// Function: usart_receive_disable
// Purpose: Enables the receiver for character finding
// Input: Void
// Output: Void
////////////////////////////////////
void usart_receive_disable(void) {
    rcsta &= 11101111b; // Disable the receiver to
prevent overflow
```

```
        pie1 &= 11011111b;                // Disable the receiver
interrupts
    return;
}

/////////////////////////////////////////////////////////////////
// Funtion:    usart_foundC
// Purpose:    Interrupt driven mechanism for finding cs on terminal
// Input:      Void
// Output:     The character captured from the terminal
/////////////////////////////////////////////////////////////////
char usart_foundc(void)
{
    char value = rcreg;
    return value;
}

/////////////////////////////////////////////////////////////////
//    Function: usart_printf
//    Purpose:  prints a character string to the terminal
//    Input:   c[] -> variable length character array
//    Output:  Void
/////////////////////////////////////////////////////////////////
void usart_printf(char c[])
{
    for(int i=0; c[i] != '\0'; i++)        // Loop through each character in the array
        usart_putc(c[i]);                // Put the character to the
terminal
}

/////////////////////////////////////////////////////////////////
//    Function: usart_putShort
//    Purpose:  prints an unsigned short to the terminal in decimal form
//    Input:   dec -> that value to be printed
//    Output:  Void
/////////////////////////////////////////////////////////////////
void usart_putShort(unsigned short dec)
{
    unsigned short dec1;                // Ten Thousands place
    unsigned short dec2;                // Thousands place
}
```

```
unsigned short dec3;      // Hundreds place
unsigned short dec4;      // Tens place
unsigned short dec5;      // Ones place

dec1 = dec/10000;         // Get the Ten Thousands place with division
dec = dec%10000;         // Calculate how much is left
dec2 = dec/1000;         // Get the Thousands place with division
dec = dec%1000;         // Calculate how much is left
dec3 = dec/100;          // Get the Hundreds place with
division
dec = dec%100;           // Calculate how much is left
dec4 = dec/10;          // Get the Tens place with division
dec5 = dec%10;          // The Ones place is how much is left

if(dec1 != 0)            // If there is a Ten Thousands place, print it
    usart_putc(dec1 + '0');
if(dec2 != 0)            // If there is a Thousands place, print it
    usart_putc(dec2 + '0');
if(dec3 != 0)            // If there is a Hundreds place, print it
    usart_putc(dec3 + '0');
if(dec4 != 0)            // If there is a Tens place, print it
    usart_putc(dec4 + '0');
usart_putc(dec5 + '0');// Print the one's place
}
```

```
////////////////////////////////////
//      Function: usart_getShort
//      Purpose: reads in an decimal number from the terminal
// Input:  Void
// Output: The decimal input in unsigned short format
////////////////////////////////////
unsigned short usart_getShort(void)
{
    char  userInput;      // User's input as a character
    int   userInt;        // User's current input converted to
integer
    unsigned int add = 0; // The user's whole input as an integer
    unsigned short short_add; // The user's whole input as a unsign. Short
    int i;                // Loop variable

    usart_printf("Please enter your digits:"); // Requests a decimal number
from user
    for(i=0; i<5; i++) // Get a
digit a max of 5 times (For an unsign. Short)
```

```

    {
        userInput = usart_getc();                // Get the
user's next digit
        if((userInput=='\r')||(userInput=='\n')){ // If the digit is a "return" or
"new line," Break from the loop
            break;
        }
        else if((userInput>='0')&&(userInput<='9')){// If the user has added a
valid digit
            userInt = userInput - '0';          // Save the
current digit in interger form
        }
        else {
            // If the user did not enter a valid digit
            usart_printf("That is not a valid digit.");// Print an error message
            return 0;
            // Return a zero
        }
        add = add*10 + userInt;                  // Add
the new digit to old total (weighed appropriately)
    }

    if(add < 32768)
        // If the number is valid
        short_add = add;                        // Store
it as a Short
    else {
        // If the number is too large
        usart_printf("Too large for a short.");// Print an error
        short_add = add;                        // Save
it (truncated) as a Short
    }

    return short_add;                           //
Return the user's Short
}

```

```

////////////////////////////////////
//      Function: usart_getByte
//      Purpose: reads in a byte in binary form
// Input:  Void
// Output: The input byte in char format
////////////////////////////////////
char usart_getByte(void) {

```

```
char out_byte = 10000000b; // User's input as a character
char input; // User's current input
character
int i; // Loop variable

for(i=0; i<8; i++) { // Get a digit a max of 5 times (For an
unsign. Short
    input = usart_getc(); // Get the user's next character

    if (input == '1') { // Character is a 1
        out_byte = (out_byte<<1) & 11111110b; // Move all previous
bits left
        out_byte++; // Add a one to the right-most
bits
    }
    else if (input=='0'){ // Character is a 0
        out_byte = (out_byte<<1) & 11111110b; // Move all previous
bits left
    }
    else { // If the character is not a
valid 1 or 0
        out_byte = 0; // Return a zero
        break; // Stop asking for bits
    }
    return out_byte; // Return the user's Short
}
}
```

```
////////////////////////////////////
// Function: usart_putByte
// Purpose: puts in a byte in binary form
// Input: The byte to output
// Output: Void
////////////////////////////////////
void usart_putByte(char reg) {
    if (reg.7 ==1) {usart_putc('1');}
    else {usart_putc('0');}

    if (reg.6 ==1) {usart_putc('1');}
    else {usart_putc('0');}

    if (reg.5 ==1) {usart_putc('1');}
    else {usart_putc('0');}
}
```

```
if (reg.4 ==1) {usart_putc('1');}  
else {usart_putc('0');}  
  
if (reg.3 ==1) {usart_putc('1');}  
else {usart_putc('0');}  
  
if (reg.2 ==1) {usart_putc('1');}  
else {usart_putc('0');}  
  
if (reg.1 ==1) {usart_putc('1');}  
else {usart_putc('0');}  
  
if (reg.0 ==1) {usart_putc('1');}  
else {usart_putc('0');}  
}
```

Serial Library: usartlib.h

```
#ifndef _USARTLIB_H_
#define _USARTLIB_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      usartlib.h                  //
//  USART function headers         //
////////////////////////////////////

// USART Function Prototypes
// See usartlib.c for actual functions

void usart_init(unsigned short rate); // Initializes the EUSART
void usart_putc(char c);              // Puts a character to the
Terminal                              // Gets a character from the
char usart_getc(void);                // Enables the receiver for char
void usart_receive_enable(void);      // Disables the receiver
finding                                // Handles a found character;
void usart_receive_disable(void);     // Prints a character string to
void usart_foundc(void);              // Prints a decimal number to Terminal
driven by interrupt                   // Accepts a decimal number from the
void usart_printf(char c[]);          Terminal
void usart_putShort(unsigned short dec); // Prints a decimal number to Terminal
unsigned short usart_getShort(void);   // Accepts a decimal number from the
Terminal                               // Accepts a byte bit by bit
char usart_getByte(void);             // Puts a byte in binary form to
from Terminal                          Terminal
void usart_putByte(char reg);
#endif // _USARTLIB_H_
```


EEPROM Library: EEPROMlib.c

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      EEPROMlib.c                    //
//      Functions for data EEPROM      //
////////////////////////////////////

// Function Prototypes are in EEPROMlib.h

char EEPROM_read(char addH, char addL)
{
    eadrh = 00000011b & addH;    // Write the 2 MSB
    eadr = addL;                // Write the rest of the address
    eecon1.7 = 0;               // Data memory mode
    eecon1.6 = 0;               // Access the EEPROM
    eecon1.0 = 1;               // Read mode
    char data;
    data = eedata;
    return data;
}

void EEPROM_write(char data,char addH, char addL)
{
    volatile bit writeIF@PIR2.4;

    eadrh = 00000011b & addH;    // Write the 2 MSB
    eadr = addL;                // Write the rest of the address
    eedata = data;              // data to be written
    eecon1.7 = 0;               // Data memory mode
    eecon1.6 = 0;               // Access the EEPROM
    eecon1.2 = 1;               // Enables writing
    char temp;                  // Record global interrupt state
    temp = intcon.7;
    intcon.7 = 0;               // Diable global interrupts
}
```

```
    eecon2 = 0x55;
    eecon2 = 0x0AA;
    eecon1.1 = 1;
    while(!writeIF);
    writeIF = 0;

    intcon.7 = temp;
    eecon1.2 = 0;
    return;
}
```

// Write sequence
// Write start

// Return interrupts to previous state
// Diables writing

EEPROM Library: EEPROMlib.h

```
#ifndef _EEPROMLIB_H_
#define _EEPROMLIB_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      EEPROMlib.h                 //
//      Functions for data EEPROM   //
////////////////////////////////////

// Function prototypes for EEPROMlib.c
char EEPROM_read(char addH, char addL);
void EEPROM_write(char data,char addH, char addL);

#endif // _EEPROMLIB_H_
```

Motors Library: motorlib.c

```
#include <system.h>
#include "ZigBeelib.h"
#include "LCDlib_SPI.h"
#include "RTCLib.h"
#include "ADlib.h"
#include "usartlib.h"
#include "EEPROMlib.h"
#include "motorlib.h"

////////////////////////////////////
//      Smart Windows Project          //
//                                     //
//      motorlib.c                      //
//      Functions for motor control      //
////////////////////////////////////

// Function Prototypes in motorlib.h

// Global volatile bits
volatile bit open_cmd      @PORTA.4;
volatile bit open_limit    @PORTE.0;
volatile bit close_cmd     @PORTA.5;
volatile bit close_limit@PORTE.1;
volatile bit middle_limit  @PORTE.2;

////////////////////////////////////
//      Function: motor_init           //
//      Purpose: Initializes the ports for motors
//      Input: Void
//      Output: Void
////////////////////////////////////
void motor_init(void){
    trisa.4 = 0;
    trisa.5 = 0;
    porta.4 = 0;
    porta.5 = 0;
    trise.0 = 1;
    trise.1 = 1;
    trise.2 = 1;
    open_cmd = 0;
    close_cmd = 0;
}

////////////////////////////////////
```

```
//      Function: motor_open
//      Purpose: Opens the window
//      Input: Void
//      Output: Void
//////////
void motor_open(void){
    open_cmd = 1;
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    while (!open_limit);
    open_cmd = 0;
}
```

```
//////////
//      Function: motor_close
//      Purpose: Closes the window
//      Input: Void
//      Output: Void
//////////
void motor_close(void){
    close_cmd = 1;
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    while (!close_limit);
    close_cmd = 0;
}
```

```
//////////
//      Function: motor_middle
//      Purpose: Middles the window
```

```

//      Input: Void
//      Output: Void
//////////
void motor_middle(void){
    if (open_limit) {
        close_cmd = 1;
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        close_cmd = 0;
    }
    else if (close_limit) {
        open_cmd = 1;
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        open_cmd = 0;
    }
    else {
        motor_open();
        close_cmd = 1;
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        while (!middle_limit);
        close_cmd = 0;
    }
}

```

}
}

Motors Library: motorlib.h

```
#ifndef _MOTORLIB_H_
#define _MOTORLIB_H_
#include<system.h>

////////////////////////////////////
//      Smart Windows Project      //
//                                  //
//      motorlib.h                  //
//      Functions for motor control //
////////////////////////////////////

// Function prototypes for motorlib.c
void motor_init(void);
void motor_open(void);
void motor_close(void);
void motor_middle(void);

#endif // _MOTOR_H_
```


6.2.2 PC Software

Desktop Application Code- Classes Listed Alphabetically (Python)

AllWindows.py

```
from __future__ import division
import sys
import time
import os
import platform
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.uic import *
from math import *
#My classes:
from SingleWindow import *
from SendToMicro import *
from RigUp import *
#Multithreading:
from multiprocessing import Process, Lock, Value, Array, Pipe

class AllWindows(QMainWindow): #QWidget):
#     def setMyTitle(self, xxxx):
#         self.setWindowTitle(xxxx)

#####
##### NEW BUTTON STUFF #####
#####
#     Commands().sendSimpleCommand(windowNumber, cmd_open, cmd_middle,
cmd_close, cmd_green, cmd_security)

    def myShowMsg(self, msg):
        qq = QMessageBox()
        qq.setText(msg)
        qq.setWindowTitle("Update!")
        qq.exec_()
    def handleAutoModeClick(self, n):
        Commands().sendSimpleCommand(windowNumber=n, cmd_security=True)
        self.myShowMsg("Window " + str(n) + " is now in timing mode!")
    def handleManualModeClick(self, n):
        print "manual mode click for window " + str(n)
    def handleEcoModeClick(self, n):
        Commands().sendSimpleCommand(windowNumber=n, cmd_green=True)
        self.myShowMsg("Window " + str(n) + " is now in eco mode!")
    def handleOpenModeClick(self, n):
        Commands().sendSimpleCommand(windowNumber=n, cmd_open=True)
        self.myShowMsg("Window " + str(n) + " is now opening!")
    def handleHalfModeClick(self, n):
        Commands().sendSimpleCommand(windowNumber=n, cmd_middle=True)
        self.myShowMsg("Window " + str(n) + " is now going to the
middle!")
    def handleCloseModeClick(self, n):
        Commands().sendSimpleCommand(windowNumber=n, cmd_close=True)
        self.myShowMsg("Window " + str(n) + " is now closing!")
    def modeClickHandler(self, obName, myInstance):
        me = obName + str(myInstance)
```

```
notme = []
if(obName.count('auto') < 1):
    notme.append('autoButton' + str(myInstance))
else:
    self.handleAutoModeClick(myInstance)
if(obName.count('manual') < 1):
    notme.append('manualButton' + str(myInstance))
else:
    self.handleManualModeClick(myInstance)
if(obName.count('eco') < 1):
    notme.append('ecoButton' + str(myInstance))
else:
    self.handleEcoModeClick(myInstance)
qpb = self.findChild(QPushButton, name=me)
qpb2 = self.findChild(QPushButton, name=notme[0])
qpb3 = self.findChild(QPushButton, name=notme[1])
qpb.setIcon(QIcon(obName + 'X.png'))
qpb2.setIcon(QIcon(notme[0][0:-1] + '.png'))
qpb3.setIcon(QIcon(notme[1][0:-1] + '.png'))
shouldEnable = (obName.count('manual') > 0)
temp1 = self.findChild(QPushButton, name="open" +
str(myInstance))
temp1.setEnabled(shouldEnable)
temp2 = self.findChild(QPushButton, name="half" +
str(myInstance))
temp2.setEnabled(shouldEnable)
temp3 = self.findChild(QPushButton, name="close" +
str(myInstance))
temp3.setEnabled(shouldEnable)

def modeClickHandler2(self, obName, myInstance):
    me = obName + str(myInstance)
    notme = []
    if(obName.count('open') < 1):
        notme.append('open' + str(myInstance))
    else:
        self.handleOpenModeClick(myInstance)
    if(obName.count('half') < 1):
        notme.append('half' + str(myInstance))
    else:
        self.handleHalfModeClick(myInstance)
    if(obName.count('close') < 1):
        notme.append('close' + str(myInstance))
    else:
        self.handleCloseModeClick(myInstance)
    qpb = self.findChild(QPushButton, name=me)
    qpb2 = self.findChild(QPushButton, name=notme[0])
    qpb3 = self.findChild(QPushButton, name=notme[1])
    qpb.setIcon(QIcon(obName + 'X.png'))
    qpb2.setIcon(QIcon(notme[0][0:-1] + '.png'))
    qpb3.setIcon(QIcon(notme[1][0:-1] + '.png'))

def createMyButton(self, objName, myInstance):
    modeButton = QPushButton()
    modeButton.setIcon(QIcon(objName + '.png'))
    if(objName.count('auto')>0): #START OUT IN AUTO MODE
```

```
        modeButton.setIcon(QIcon(objName + 'X.png'))
modeButton.setIconSize(QSize(50,50)) #70
modeButton.setObjectName(objName + str(myInstance))
def miniClickHandle():
    self.modeClickHandler(objName, myInstance)
self.connect(modeButton, SIGNAL("clicked()"), miniClickHandle)
return modeButton

def createMyButton2(self, objName, myInstance):
modeButton = QPushButton()
modeButton.setEnabled(False)
modeButton.setIcon(QIcon(objName + '.png'))
modeButton.setIconSize(QSize(50,50)) #70
modeButton.setObjectName(objName + str(myInstance))
def miniClickHandle():
    self.modeClickHandler2(objName, myInstance)
self.connect(modeButton, SIGNAL("clicked()"), miniClickHandle)
return modeButton

def makeWholeVlayoutButtonSet(self, myInstance):
vlayout = QVBoxLayout()
hlayout = QHBoxLayout()
hlayout.addWidget(self.createMyButton('autoButton',
myInstance))
hlayout.addWidget(self.createMyButton('manualButton',
myInstance))
hlayout.addWidget(self.createMyButton('ecoButton', myInstance))
hlayout2 = QHBoxLayout()
hlayout2.addWidget(self.createMyButton2('open', myInstance))
hlayout2.addWidget(self.createMyButton2('half', myInstance))
hlayout2.addWidget(self.createMyButton2('close', myInstance))
vlayout.addLayout(hlayout)
vlayout.addLayout(hlayout2)
return vlayout

#####
#####
#####

def pullDescriptionFromFile(self, myFile):
    if os.path.exists(myFile) == False: #if something went wrong
        return "(None)"
    with open(myFile, "r") as f:
        lns = f.read()
        f.close()
    lns = lns.split("\n")
    if(len(lns) > 1):
        return str(lns[0])
    return "(None)"
def reloadNames(self):
    print "reloadNames"
    f_list = []
#     for f in os.listdir(os.getcwd()):
#         if str(f).split('.')[1] == "settings":
#             f_list.append(f)
#     for i_f in range(len(f_list)):
```

```

#         with open(f_list[i_f],"r") as f:
#             linez = f.read()
#             f.close()
#             linez = linez.split("\n")
#             if(len(linez) > 1):
#                 myTempTitle = str(linez[0])
#                 self.findChild(QLabel, "myLabel_" +
str(i_f)).setText(myTempTitle)
def myClickHandler(self):
    for z in range(self.myNumberOfButtons):
        if self.findChild(QPushButton,
QString("myWindowButtons_"+str(z))) == self.focusWidget():
            f2 = SingleWindow(winID = self.listOfWindowIDs[z])
            f2.show()
            #self.connect(self.visibleRegion(),
SIGNAL("clicked()"), self.reloadNames)
            return
def __init__(self, parent=None, i_am_reloading=False,
refresh_text=' '):
    #####
    ws = Commands().setup()
    RigUp().matchSettingFiles(ws)
    light_readings = Commands().getLightSensorReadings()
    #####
    if(i_am_reloading == False):
        super(AllWindows, self).__init__(parent)
        self.setWindowTitle("SmartWindows! Click a window to change
manual time settings...")
        self.resize(QSize(600,200))
        self.addMyMenuBar()
        bigvlayout = QVBoxLayout()
        hlayout = QHBoxLayout()
        hlayout.setObjectName("myVLayout")
        listOfSettingsFiles = []
        listOfWindowIDs = []
        listOfWindowNames = []
        for f in os.listdir(os.getcwd()):
            if str(f).split('.')[1] == "settings":
                listOfSettingsFiles.append(f)
                listOfWindowIDs.append(str(f).split('.')[0])

listOfWindowNames.append(self.pullDescriptionFromFile(f))
    myWindowTuple = zip(listOfSettingsFiles, listOfWindowIDs,
listOfWindowNames)
    self.myNumberOfButtons = len(listOfSettingsFiles)
    self.listOfWindowIDs = listOfWindowIDs #save this for later (in
click handler)
    ##### If we have no windows #####
    if(len(myWindowTuple) == 0):
        myButton = QPushButton()
        myButton.setIcon(QIcon("no_windows.jpg"))
        myButton.setIconSize(QSize(500,312))
        hlayout.addWidget(myButton)
    #####
    # "getLightSensorReadings" returns light sensor 0, battery
sensor 0, light sensor 1, battery sensor 1, etc....

```

```

for mwt in range(len(myWindowTuple)):
    miniVerticalLayout = QVBoxLayout()
    myButton = QPushButton()
    myButton.setIcon(QIcon("windowPic.png"))
    myButton.setIconSize(QSize(170,170)) #was 150,150
    myButton.setObjectName("myWindowButtons_"+str(mwt))
    self.connect(myButton, SIGNAL("clicked()"),
self.myClickHandler)
    myLabel = QLabel()
    myLabel.setObjectName("myLabel_" + str(mwt)) #36 / 48 / 60
    myLabel.setFont(QFont("ChopinScript"))
#####
    myLabel.setText("<font size=\"48\">" + ws[mwt] + "</font>")
    myLabel.setAlignment(Qt.AlignHCenter)
    # Light readout... #
    myLabel2 = QLabel()
    myLabel2.setObjectName("light_" + str(mwt)) #36 / 48 / 60
    myD = 1.0*light_readings[2*mwt] #i just made a variable
called my D. lolz. it stands for double tho.
    myD = 100.*myD/255.
    myLabel2.setText("<font color=\"green\"><h3>Light: " +
str(myD) [0:4] + "%" + "</h3></font>")
    myLabel2.setAlignment(Qt.AlignHCenter)
    # Battery readout... #
    myLabel3 = QLabel()
    myLabel3.setObjectName("battery_" + str(mwt)) #36 / 48 / 60
    myD2 = 1.0*light_readings[2*mwt+1] #i just made a variable
called my D. lolz. it stands for double tho.
    myD2 = 100.*myD2/170.
    cut_off = 75;
    if(myD2 > cut_off):
        myLabel3.setText("<font color=\"blue\"><h3>Battery: " +
(str(myD2) + " ") [0:4] + "% (Ok)" + "</h3></font>")
    else:
        myLabel3.setText("<font color=\"blue\"><h3>Battery: " +
(str(myD2) + " ") [0:4] + "%<font color=\"red\"> (LOW!)</font>" +
"</h3></font>")
    myLabel3.setAlignment(Qt.AlignHCenter)
#####
    miniVerticalLayout.addWidget(myLabel)
    miniVerticalLayout.addWidget(myButton)
    miniVerticalLayout.addWidget(myLabel2)
    miniVerticalLayout.addWidget(myLabel3)

miniVerticalLayout.addLayout(self.makeWholeVlayoutButtonSet(mwt))
#BUTTONS
#####
    #////////// add border ?
    hlayout.addItem(miniVerticalLayout)
    qw = QWidget()
    #####
    bigvlayout.addLayout(hlayout)
    refreshButton = QPushButton()
    refreshButton.setBaseSize(100, 100)
    refreshButton.setIcon(QIcon("refresh2.png"))
    refreshButton.setIconSize(QSize(100,100))
    myLabel2 = QLabel()

```

```

myLabel2.setFont(QFont("ChopinScript"))
myLabel2.setText("<font size=\"48\">Refresh</font>")
myLabel2.setAlignment(Qt.AlignHCenter)
myLabel2.setObjectName("refreshLabel")
bigvlayout.addWidget(myLabel2)
refLabel = QLabel()
refLabel.setAlignment(Qt.AlignHCenter)
refLabel.setText(refresh_text)
bigvlayout.addWidget(refLabel)
bigvlayout.addWidget(refreshButton)
def refreshClickHandle():
    xxyy = "<h4>" + time.asctime() + "</h4>"
    self.__init__(parent, i_am_reloading=True,
refresh_text=xxyy)
    self.connect(refreshButton, SIGNAL("clicked()"),
refreshClickHandle)
    qw.setLayout(bigvlayout) #was "qw.setLayout(hlayout)"
    self.setCentralWidget(qw)

def addMyMenuBar(self):
    #First create all actions:
    quitAction = self.createAction("&Quit",
self.close, "Ctrl+Q", "filequit", "Close the application")
    delWinAction = self.createAction("&Delete ALL Windows",
self.deleteWindowClick, "Ctrl+D", "deletewindow", "Remove a Window")
    #was self.syncTimeClick
    syncTimeAction = self.createAction("&Sync Module's Time",
self.sendMyCurrentTime, "Ctrl+S", "synctime", "Synchronize the Time")
    howToUseAction = self.createAction("&How to Use",
self.howToUseClick, "Ctrl+H", "howtouse", "User help content")
    #Now create all menu categories:
    fileMenu = self.menuBar().addMenu("&File")
    optionsMenu = self.menuBar().addMenu("&Options")
    helpMenu = self.menuBar().addMenu("&Help")
    #Now add actions to menu categories:
    fileMenu.addAction(quitAction)
    optionsMenu.addAction(delWinAction)
    optionsMenu.addAction(syncTimeAction)
    helpMenu.addAction(howToUseAction)

    #A helper function:
    def createAction(self, text, slot=None, shortcut=None, icon=None,
tip=None, checkable=False, signal="triggered()"):
        action = QAction(text, self)
        if icon is not None:
            action.setIcon(QIcon(":/%s.png" % icon))
        if shortcut is not None:
            action.setShortcut(shortcut)
        if tip is not None:
            action.setToolTip(tip)
            action.setStatusTip(tip)
        if slot is not None:
            self.connect(action, SIGNAL(signal), slot)
        if checkable:
            action.setCheckable(True)
        return action
    def sendMyCurrentTime(self):

```

```

Commands().sendCurrentTime(waitAfter=False)
qq = QMessageBox()
qq.setText("Current time set!")
qq.setWindowTitle("Update!")
qq.exec_()
def deleteWindowClick(self):
    Commands().sendEraseAllWindowsCommand()
    qq = QMessageBox()
    qq.setText("All windows have been removed");
    qq.setWindowTitle("Update!")
    qq.exec_()
def howToUseClick(self):
    qmb = QMessageBox()
    qmb.setText("<h1><font color=\"blue\">HOW TO
USE</font></h1>\n\n\n<img src=\"imconfus.jpg\" width=\"346\"
height=\"300\" />")
    qmb.setWindowTitle("Connection")
#     qmb.setIcon(QMessageBox.Warning)
    qmb.addButton(QString("Alright!"), QMessageBox.AcceptRole)
    qmb.addButton(QString("Explains it all!"),
QMessageBox.RejectRole)
    qmb.addButton(QString("Terrific!"), QMessageBox.ActionRole)
#     qmb.addButton(QString("Amazing!"), QMessageBox.HelpRole)
    qmb.exec_()

```

Commands.py

```

import time
from SendToMicro import *

#Commands to call from other classes:
#--sendSimpleCommand
#--sendNewAlarmTimes
#--sendWindowName
#--sendCurrentTime
#--getLightSensorReadings
#--sendEraseAllWindowsCommand

class Commands:

    #-----CONSTANTS-----
    #For window time setting:
    WINDOW_TIMES_PER_WINDOW = 8
    WINDOW_TIMES_FILLER = 255
    #For sending simple commands:
    CMD_OPEN      = 0
    CMD_MIDDLE    = 1
    CMD_CLOSE     = 2
    CMD_GREEN     = 3
    CMD_SECURITY  = 4

    #-----TOOLS-----

    # "strTimeToBCD" aids in converting times (strings) to BCD
    def strTimeToBCD(self, strTime):
        if(len(strTime) < 2):

```

```

        strTime = '0' + strTime
        intTime_part1 = int(strTime[0])
        intTime_part2 = int(strTime[1])
        myBCD = 16*intTime_part1 + intTime_part2
        return myBCD

def intTimeToBCD(self, intTime):
    intTime_part1 = int(intTime/10)
    intTime_part2 = int(intTime%10)
    return int(16*intTime_part1 + intTime_part2)

#-----Category A Commands-----
#"setup" returns an array of window name
def setup(self):
    stme = SendToMicroEfficient()
    stme.setup()
    c = ''
    c2 = ''
    windowNumbers = []
    windowNames = []
    print 'STARTING SETUP'
    #send a zero, pause, then a one
    stme.send(chr(0)) # 0 = listen for command
    x = stme.receive(1)
    print "x = " + str(ord(x))
    time.sleep(1.3)
#    time.sleep(3)
    stme.send(chr(254)) # 254 = start up
    y = stme.receive(1)
    print "y = " + str(ord(y))
    print 'ready to receive windows...',
    for i in range(10):
        print '.',
        c = stme.receive(1)
        print "c = " + str(ord(c))
        if(c == chr(255)):
            break
        print '1',
        c2 = stme.receive(10)
        windowNumbers.append(c)
        windowNames.append(c2)
    #stme.receive(1) #final stop byte
    print 'FINISHED WITH SETUP'
    return windowNames
#    return windowNames, windowNumbers
#-----

#-----Category B Commands-----
#"sendSimpleCommand" sends either open/middle/close/green/security
def sendSimpleCommand(self, windowNumber, cmd_open=False,
cmd_middle=False, cmd_close=False, cmd_green=False,
cmd_security=False):
    mType = 'c'
    mWho = windowNumber
    mDataArr = [0,0,0,0,0,0,0,0,0,0,0]
    #Figure out which command:

```



```

cmd = -1
if(cmd_open == True):
    cmd = self.CMD_OPEN
elif(cmd_middle == True):
    cmd = self.CMD_MIDDLE
elif(cmd_close == True):
    cmd = self.CMD_CLOSE
elif(cmd_green == True):
    cmd = self.CMD_GREEN
elif(cmd_security == True):
    cmd = self.CMD_SECURITY
if(cmd == -1):
    print "sendSimpleCommand error: no valid command sent"
    return
mDataArr[0] = cmd

self.send_generic_b_command(type=mType,who=mWho,dataArr=mDataArr)

#"sendNewAlarmTimes" sends all the alarm times for a window
#times is an array (of length 8) of a 5-tuple of command, sec, min,
hr, days
#NOTE: sec, min, hr will be taken in normal, then converted to BCD
#NOTE: IT REDIRECTS TO sendNewAlarmTimes2, THIS IS NO LONGER USED!
def sendNewAlarmTimes(self, windowNumber, times):
    self.sendNewAlarmTimes2(windowNumber, times)
    if(0 == 1):
        if len(times[0]) != 5:
            print "WRONG NUMBER OF TIMES! I WANT A TUPLE WITH 5
ELEMENTS"

            #tell it that we are setting alarms:
            print "about to send generic b command...",
            self.send_generic_b_command(type='a', who=windowNumber)
            print "done."
            #set up communication class
            stme2 = SendToMicroEfficient()
            stme2.setup()
            #get a confirmation 'a':
            if(stme2.receive(1) != 'a'):
                print 'Warning: got unexpected alarm response'
            else:
                print 'Got expected alarm response'
            #####
            #check object types
            for k in range(5):
                if (isinstance(times[0][k], int) == False):
                    print "WARNING: expected tuple element " + str(k) +
" to be an int"
            #####
            happycounter = 1
            #tell it what alarms we are setting:
            for t in times:
                #Convert to BCD. open/close, sec, min, hr, day
                t2 = chr(t[0]), chr(self.intTimeToBCD(t[1])),
chr(self.intTimeToBCD(t[2])), chr(self.intTimeToBCD(t[3])), chr(t[4])
                for tt in t2:
                    stme2.send(tt)

```

```

        #//////////
        print "send byte number " + str(happycounter)
        happycounter += 1
        #//////////
        time.sleep(.2)
    #fill it up to 8:
    extraFillerTimes = self.WINDOW_TIMES_PER_WINDOW -
len(times)
    for i in range(extraFillerTimes):
        for j in range(5):
            stme2.send(chr(self.WINDOW_TIMES_FILLER))
            #//////////
            print "send FILLER byte number " +
str(happycounter) + " that is equal to the byte value " +
str(self.WINDOW_TIMES_FILLER)
            happycounter += 1
            #//////////
            time.sleep(.2)

    def sendNewAlarmTimes2(self, windowNumber, times):
        # Pad "times" to be a 2D 8 x 5 matrix
        if len(times[0]) != 5:
            print "WRONG NUMBER OF TIMES! I WANT A TUPLE WITH 5
ELEMENTS"
            while len(times) < 8:
                times.append((self.WINDOW_TIMES_FILLER,
self.WINDOW_TIMES_FILLER, self.WINDOW_TIMES_FILLER,
self.WINDOW_TIMES_FILLER, self.WINDOW_TIMES_FILLER))
            # Send the first command that we are sending an alarm
            print "starting..."
            print "type is " + str(type(times[0][0]))
            for n in range(4): #up to 4 because it sends two each time
                print "-----"
                print "sending alarms " + str(2*n) + " and " + str(2*n + 1)
+ "...",
                print "BEFORE bcd...",
                print str(times[2*n])
                print "AND"
                print str(times[2*n+1])
                print "then after..."
                two_alarms = []
                if(times[2*n][0] != 255): #if it's not a filler
                    two_alarms.append(times[2*n][0])
# 1 open/close        do NOT convert
                    two_alarms.append(self.intTimeToBCD(times[2*n][1]))
# 1 sec                convert to BCD
                    two_alarms.append(self.intTimeToBCD(times[2*n][2]))
# 1 min                convert to BCD
                    two_alarms.append(self.intTimeToBCD(times[2*n][3]))
# 1 hr                 convert to BCD
                    two_alarms.append(times[2*n][4])
# 1 day                do NOT convert
                else: #if it IS a filler
                    two_alarms.append(255)
                    two_alarms.append(255)

```

```

        two_alarms.append(255)
        two_alarms.append(255)
        two_alarms.append(255)
    if(times[2*n+1][0] != 255):
        two_alarms.append(times[2*n+1][0])
# 2 open/close    do NOT convert
        two_alarms.append(self.intTimeToBCD(times[2*n+1][1]))
# 2 sec          convert to BCD
        two_alarms.append(self.intTimeToBCD(times[2*n+1][2]))
# 2 min          convert to BCD
        two_alarms.append(self.intTimeToBCD(times[2*n+1][3]))
# 2 hr           convert to BCD
        two_alarms.append(times[2*n+1][4])
# 2 day          do NOT convert
    else: #if it IS a filler
        two_alarms.append(255)
        two_alarms.append(255)
        two_alarms.append(255)
        two_alarms.append(255)
        two_alarms.append(255)
    print str(two_alarms) #####
    c = '1'
    if(n == 1):
        c = '2'
    elif(n == 2):
        c = '3'
    elif(n == 3):
        c = '4'
    self.send_generic_b_command(type=c, who=windowNumber,
dataArr=two_alarms, endConfirmation=True) #type will be 1,2,3,4 (which
are all 'alarm' commands too)
    print "listening for response..."
    print "Done send alarms"

#"sendWindowName"
def sendWindowName(self, windowNumber, windowName):
    print "~~~~~SENDING WINDOW NAME~~~~~"
    #make windowName at least 10 characters long:
    windowName = windowName + "          "
    myDataArr = []
    for ii in range(10):
        myDataArr.append(ord(str(windowName)[ii]))
    self.send_generic_b_command(type='n', who=windowNumber,
dataArr=myDataArr)
    print "done sending name"

#"sendCurrentTime" sends the current time to the head module
def sendCurrentTime(self, waitAfter=True):
    print "~~~~~SENDING CURRENT TIME
~~~~~"
    lt = time.localtime()
    sec_bcd = self.strTimeToBCD(time.strftime('%S',lt))
    min_bcd = self.strTimeToBCD(time.strftime('%M',lt))
    hour_bcd = self.strTimeToBCD(time.strftime('%H',lt))
    day = int(time.strftime('%w',lt)) + 1
    mType = 't' # 't' = time

```

```

mWho = 255 # 255 = all modules
mDataArr = [0,0,0,0,0,0,0,0,0,0,0]
mDataArr[0] = sec_bcd # data[0] = sec
mDataArr[1] = min_bcd # data[1] = min
mDataArr[2] = hour_bcd # data[2] = hr
mDataArr[3] = day

self.send_generic_b_command(type=mType, who=mWho, dataArr=mDataArr)
##### TODO: don't always wait here
if(waitAfter == True):
    time.sleep(3)

#"getLightSensorReadings" returns light sensor 0, battery sensor 0,
light sensor 1, battery sensor 1, etc....
def getLightSensorReadings(self):
    self.send_generic_b_command(type='l', who=255)
    windowReadings = []
    stme = SendToMicroEfficient()
    stme.setup()
    x = -1
    while True:
        x = ord(stme.receive(1))
        if(x == 255):
            break
        else:
            windowReadings.append(x)
    print "light sensor readings = " + str(windowReadings)
    return windowReadings

#"sendEraseAllWindowsCommand" erases all windows
def sendEraseAllWindowsCommand(self):
    self.send_generic_b_command(type='e', who=255)

#"send_generic_b_command" sends out a category b command
# type must always be specified
# who is the window # (by default all windows)
# dataArr is the 10 bytes to send (by default all zeros)
def send_generic_b_command(self, type, who=255,
dataArr=[0,0,0,0,0,0,0,0,0,0], endConfirmation=False):
    #format first 3 bytes
    byte_c = chr(0)
    byte_b = chr(who)
#    byte_b = chr(ord(who)) #int cast, in case passed as string
#####
    byte_a = type
#    if isinstance(type, int):
#        byte_a = chr(type)
#    else:
#        byte_a = type
#####
    #prep for send
    stme = SendToMicroEfficient()
    stme.setup()
    #Send Byte C / Get Confirmation
    stme.send(byte_c)
    stme.receive(1)

```

```
time.sleep(1.3)
#Send Byte B / Get Confirmation
stme.send(byte_b)
stme.receive(1)
time.sleep(.3)
#Send Byte A / NO Confirmation
stme.send(byte_a)
time.sleep(.3)
#Send the 10 data bytes with delay / NO Confirmation
for i in range(10):
    stme.send(chr(dataArr[i]))
    time.sleep(.2)
if(endConfirmation == True):
    cc = stme.receive(1)
    if(cc == 'a'):
        print "Got positive confirmation of a"
    time.sleep(.2)
```

```
#-----
#-----
#-----
```

ConvertTimes.py

```
class ConvertTimes:
    def convert(self, stuffToWrite):
        stuffToSend = stuffToWrite.split('\n')[1:]
        bigArr = []
        for x in stuffToSend:
            xx = x.split(',')
            if(len(xx) == 4):
                tt = int(xx[3]), 0, int(xx[2]), int(xx[1]), int(xx[0])
                bigArr.append(tt)
        print str(bigArr)
        return bigArr
```

main.py

```
#Libraries:
from __future__ import division
import sys
import time
import os
import platform
import d2xx
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.uic import *
from math import *
#My classes:
from SplashForm import *
from AllWindows import *
from SendToMicro import *
```

```
#import Commands
from Commands import *
from RigUp import *
#####
import urllib
#####

#Multithreading:
from multiprocessing import Process, Lock, Value, Array, Pipe

class WholeProgram:

#####
#####
    def checkForWindowChanges(self):
        def getStateOfWindowX(i):
            STATE_OPENED = -1
            STATE_MIDDLE = 0
            STATE_CLOSED = 1
            theurl =
'http://129.74.154.171/observer/media.php?id=1267301014&uid=5619109'
            if i != 1:
                theurl =
'http://129.74.154.171/observer/media.php?id=1267369485&uid=5619109'
            f = urllib.urlopen(theurl)
            s = f.read()
            f.close()
            #<RATING user_rated="-1" up_count="0" down_count="1"/>
            iStart = s.find('user_rated')
            iEnd = s.find('up_count')
            s = s[iStart:iEnd]
            # print s
            if(s.count('-1') > 0):
                return STATE_OPENED
            elif(s.count('1') > 0):
                return STATE_CLOSED
            else:
                return STATE_MIDDLE
        print 'Starting checkForWindowChanges...'
        while True:
            win1_old_state = getStateOfWindowX(1)
            win1_current_state = getStateOfWindowX(1)
            win2_old_state = getStateOfWindowX(2)
            win2_current_state = getStateOfWindowX(2)
            while((win1_old_state == win1_current_state) and
(win2_old_state == win2_current_state)):
                win1_current_state = getStateOfWindowX(1)
                win2_current_state = getStateOfWindowX(2)
            # print '.',
            time.sleep(.5)
            if(win1_old_state != win1_current_state): #window 1 command
sent
                if win1_current_state == -1:
                    Commands().sendSimpleCommand(0, cmd_open=True)
                elif win1_current_state == 0:
```

```

        Commands().sendSimpleCommand(0, cmd_middle=True)
    else:
        Commands().sendSimpleCommand(0, cmd_close=True)
#         print 'Window 1 has a new state.... ' +
str(win1_current_state)
        elif(win2_old_state != win2_current_state): #window 2
command sent
        if win2_current_state == -1:
            Commands().sendSimpleCommand(1, cmd_open=True)
        elif win2_current_state == 0:
            Commands().sendSimpleCommand(1, cmd_middle=True)
        else:
            Commands().sendSimpleCommand(1, cmd_close=True)
#         print 'Window 2 has a new state.... ' +
str(win2_current_state)

#####
#####

    #This function does opening animation
    def openingAnimation(self):
        app = QApplication(sys.argv)
        app.quitOnLastWindowClosed()
        sform = SplashForm()
        sform.show()
        app.exec_()

#####
#
    #This function constantly listens for incoming microcontroller
commands:
    #     def constantListening(self):
    #         while(1):
    #             xyz = SendToMicro().receive()
    #             print xyz,
    #             sys.stdout.flush()
    #             print str(self.fl)
    #             self.fl.setWindowTitle("TEST")

#####
#

    def __init__(self):
        #####One of these is to listen for android, the other is
the regular program##
        with open('alternate.txt','r') as f:
            type = f.read().strip()

#####
#####
#         type = '1' #//////////////////// TO DISABLE ANDROID LISTENING
////////////////////
        if type == '0':
            with open('alternate.txt','w') as f:
                type = f.write('1')

```

```
        android = Process(target=self.checkForWindowChanges())
        android.start()
    else:
        with open('alternate.txt', 'w') as f:
            type = f.write('0')
        app = QApplication(sys.argv)
        app.quitOnLastWindowClosed()
        fakeComm = Process(target=self.openingAnimation)
        fakeComm.start() #start splash screen
        self.fl = AllWindows() #wait for real stuff to happen
        fakeComm.terminate() #kill splash screen
        self.fl.show()
        app.exec_()
        print "done."

if __name__ == '__main__':
    wp = WholeProgram()
    print "got here"
```

RigUp.py

```
import os
class RigUp:
    noisy = True
    def matchSettingFiles(self, myMicros2, noisy=False):
        print "myMicros2 = " + str(myMicros2)
        if noisy == False:
            self.noisy = True
            noisy = True
        # myMicrosComb = str(myMicros2)
        for w in range(10): #maximum number of window modules is 10
            # if str(myMicrosComb).count(str(w)) > 0: #if that settings
            # file exists
            if w < len(myMicros2):
                if os.path.exists(os.path.join(os.getcwd(), str(w) +
                '.settings')):
                    self.dprint(str(w) + '.settings exists... write
                    name just in case...')
                    with open(str(w)+'.settings', 'r') as f:
                        xx = f.read()
                    f.close()
                    xx2 = xx.split('\n')
                    with open(str(w)+'.settings', 'w') as f2:
                        for j in range(len(xx2)):
                            if(j == 0):
                                f2.write(myMicros2[w] + '\n')
                            else:
```



```
                f2.write(xx2[j])
            f2.close()
        else:
            self.dprint(str(w) + '.settings does NOT exist...
creating it now....')
            with open(str(w) + '.settings', 'w') as fopenfile:
                fopenfile.write(myMicros2[w])
                print "putting in file " + myMicros2[w]

#####
                fopenfile.close()
            else:
                if os.path.exists(os.path.join(os.getcwd(), str(w) +
'.settings')):
                    # self.dprint(str(w) + '.settings exists... i will
have to remove it...')
                    print str(w) + '.settings exists... i will have to
remove it...'
                    os.remove(os.path.join(os.getcwd(), str(w) +
'.settings'))
                else:
                    self.dprint(str(w) + '.settings does NOT exists...
good...')
    def dprint(self, msg):
        if self.noisy == True:
            print msg
        else:
            print ''
```

SendToMicro.py

```
import time
import d2xx

class SendToMicro:
    def send(self, userInput, noisy = False):
        d = d2xx.listDevices(d2xx.OPEN_BY_DESCRIPTION) # list devices
by description, returns tuple
        if(noisy == True):
            print "Devices found: " + str(d)
        try:
            h = d2xx.open(d.index('FT232R USB UART'), ) #get the one we
want
            h.setBaudRate(d2xx.BAUD_57600)
            h.setDataCharacteristics(d2xx.BITS_8, d2xx.STOP_BITS_1,
d2xx.PARITY_NONE)
            for i in range(len(userInput)):
                h.write(userInput[i])
            # h.read(2)
        except ValueError:
            print "Microcontroller not recognized"
    def receive(self, bytesToLookFor = 1, noisy = False):
        if not isinstance(bytesToLookFor, int):
            print "epic fail"
            return -1
        d = d2xx.listDevices(d2xx.OPEN_BY_DESCRIPTION) # list devices
by description, returns tuple
        if(noisy == True):
```

```
        print "Devices found: " + str(d)
    try:
        h = d2xx.open(d.index('FT232R USB UART', )) #get the one we
want
        h.setBaudRate(d2xx.BAUD_57600)
        h.setDataCharacteristics(d2xx.BITS_8, d2xx.STOP_BITS_1,
d2xx.PARITY_NONE)
        #         time.sleep(.01)
        x = h.read(bytesToLookFor)
    except ValueError:
        print "Microcontroller not recognized"
    finally:
        return x
    def microcontrollerIsHookedUp(self):
        d = d2xx.listDevices(d2xx.OPEN_BY_DESCRIPTION) # list devices
by description, returns tuple
        try:
            h = d2xx.open(d.index('FT232R USB UART', )) #get the one we
want
            return True
        except ValueError:
            return False

class SendToMicroEfficient:
    def setup(self):
        d = d2xx.listDevices(d2xx.OPEN_BY_DESCRIPTION) # list devices
by description, returns tuple
        try:
            self.h = d2xx.open(d.index('FT232R USB UART', )) #get the
one we want
            self.h.setBaudRate(d2xx.BAUD_57600)
            self.h.setDataCharacteristics(d2xx.BITS_8,
d2xx.STOP_BITS_1, d2xx.PARITY_NONE)
        except ValueError:
            print "EPIC FAIL"
    def send(self, userInput):
        #         print str(ord(userInput[0]))
        for i in range(len(userInput)):
            self.h.write(userInput[i])
    def receive(self, bytes):
        return self.h.read(bytes)
```

SingleWindow.py

```
import sys
import time
import os
import platform
import d2xx
#from PyQt4.QtCore import *
from PyQt4.QtCore import Qt
from PyQt4.QtCore import SIGNAL
from PyQt4.QtCore import QSize
from PyQt4.QtCore import QTime
from PyQt4.QtGui import *
from PyQt4.uic import *
from math import *
from Commands import *
```

```

from ConvertTimes import *

class SingleWindow(QWidget):
    #-----Get a bit from a number-----
    def getBit(self, num, bitNumFromLeft):
        if bitNumFromLeft == 0:
            return int(bool(num & 0b10000000))
        if bitNumFromLeft == 1:
            return int(bool(num & 0b01000000))
        if bitNumFromLeft == 2:
            return int(bool(num & 0b00100000))
        if bitNumFromLeft == 3:
            return int(bool(num & 0b00010000))
        if bitNumFromLeft == 4:
            return int(bool(num & 0b00001000))
        if bitNumFromLeft == 5:
            return int(bool(num & 0b00000100))
        if bitNumFromLeft == 6:
            return int(bool(num & 0b00000010))
        if bitNumFromLeft == 7:
            return int(bool(num & 0b00000001))
    #-----Get a "day module"-----
    def getDayModule(self, timestampCounter, whichDay, b0=0, b1=0,
b2=0, b3=0, enabled=False): #returns QVBoxLayout of a day
        myMaxHeight = 18
        myMaxWidth = 25
        myMaxWidth2 = 85
        myMaxWidth3 = 65
        dayNamez =
["Enabled", "M", "T", "W", "Th", "F", "Sa", "Su", "Time", "Type", "Byte 0", "Byte
1", "Byte 2", "Byte 3"]
        #####
        dayName = dayNamez[whichDay%(len(dayNamez))]
        #####
        miniVLayout = QVBoxLayout() #vertical layout
        tempQL = QLabel()
        tempQL.setText(dayName)
        tempQL.setMaximumHeight(myMaxHeight)
        if whichDay == 0: #"enabled"
            x = QCheckBox()
            x.setMaximumHeight(myMaxHeight)
            x.setMinimumWidth(myMaxWidth + 40)
            if(enabled):
                x.setChecked(True)
            else:
                x.setChecked(False)
        if whichDay > 0 and whichDay <= 7: #"M-Su"
            x = QCheckBox()
            x.setMaximumWidth(myMaxWidth)
            #byte 1 tells us how to fill in the days of the week
            #####
            if self.getBit(b0, whichDay) == 1:
                x.setChecked(True)
            #####
            if self.getBit(b0, whichDay+1) == 1:
                x.setChecked(True)

```

```

#####
if whichDay == 8: # "Time"
    x = QTimeEdit()
    x.setMaximumWidth(myMaxWidth2)
    #byte 2 tells us how to fill in the hour. and byte 3 the
minute.
    x.setTime(QTime(b1,b2))
if whichDay == 9: # "Type"
    x = QComboBox()
    x.addItem("Open")
    x.addItem("Middle")
    x.addItem("Close")
    x.setMaximumWidth(myMaxWidth2)
    #byte 3 tells us if it is an open or close operation
    if b3 == 0:
        x.setCurrentIndex(0)
    else:
        x.setCurrentIndex(1)
if whichDay >= 10 and whichDay <= 13: # "Bytes 0-3"
    x = QLabel()
    if whichDay == 10: #byte 0
        x.setText(str(b0))
    if whichDay == 11: #byte 1
        x.setText(str(b1))
    if whichDay == 12: #byte 2
        x.setText(str(b2))
    if whichDay == 13: #byte 3
        x.setText(str(b3))
    x.setMaximumWidth(myMaxWidth3)
    x.setMinimumWidth(45)
    tempQL.setLayoutDirection(Qt.RightToLeft)
    x.setLayoutDirection(Qt.RightToLeft)
x.setMaximumHeight(myMaxHeight)
x.setObjectName(dayName.strip() + "_" + str(timestampCounter))

if whichDay >= 1:
    x.setEnabled(enabled)
def miniAndyClick():
    self.allOtherClicks(x.objectName())
    self.connect(x, SIGNAL("clicked()"), miniAndyClick)
    self.connect(x, SIGNAL("timeChanged(QTime)"), miniAndyClick)
    self.connect(x, SIGNAL("currentIndexChanged(int)"),
miniAndyClick)
    miniVLayout.addWidget(tempQL)
    miniVLayout.addWidget(x)
    return miniVLayout

#-----Get a "week module"-----
def getWeekModule(self, timestampCounter, byte1=0, byte2=0,
byte3=0, byte4=0, enabled=False):
    myWeek = QHBoxLayout() #horizontal layout
    for i in range(14):

myWeek.addItem(self.getDayModule(timestampCounter,i,byte1,byte2,byte3,b
yte4,enabled))
    return myWeek

```

```

#----Deal with Submit button click (save to file, send to
microcontroller)----
def submitButtonClicked(self):
    #figure out stuff to write
    stuffToWrite = self.getDescription() + "\n"
    for i in range(0,100):
        tempChild = self.findChild(QCheckBox,
name="Enabled_"+str(i))
        if tempChild == None:
            break
        else: #the row exists...
            if(tempChild.isChecked()): #and the row is enabled...
                stuffToWrite += self.findChild(QLabel, name="Byte
0_"+str(i)).text() + ","
                stuffToWrite += self.findChild(QLabel, name="Byte
1_"+str(i)).text() + ","
                stuffToWrite += self.findChild(QLabel, name="Byte
2_"+str(i)).text() + ","
                stuffToWrite += self.findChild(QLabel, name="Byte
3_"+str(i)).text() + "\n"
            #write it to file
            myFname = str(self.myWinID) + ".settings"
            with open(myFname,"w") as f:
                f.write(stuffToWrite)
                f.close()
            ##### SEND TO MICROCONTROLLER
            #####
            bigArr = ConvertTimes().convert(stuffToWrite)
            Commands().sendCurrentTime()
            time.sleep(3)
            Commands().sendNewAlarmTimes(int(self.myWinID), bigArr)

            #####
            #####
            #send to microcontroller (DEBUGGING: output messagebox)
            message = QMessageBox()
            message.setText("Finished.\n" \
                +"\nThe following was sent and recorded
locally...\n" + stuffToWrite)
            message.addButton("Accept", QMessageBox.AcceptRole)
            message.exec_()
            #get rid of the "must save changes" warning:
            (self.findChild(QLabel, name="titleLabel")).setText("<h2>Window
Times:</h2>")
#-----Deal with exit click-----
def exitClicked(self):
    self.close()

#-----Get Window Description-----
def getDescription(self):
    if self.myWinDesc != None:
        return self.myWinDesc
    myFname = str(self.myWinID) + ".settings"
    if os.path.exists(myFname) == False:
        return "(None)"
    with open(myFname,"r") as f:
        lns = f.read()

```

```

        f.close()
    lns = lns.split("\n")
    if(len(lns) >= 1):
        self.myWinDesc = str(lns[0])
        return self.myWinDesc
    return "(None)"
#-----Load from file (called within __init__)-----
def loadFromFile(self, vlayout):
    myFname = str(self.myWinID) + ".settings"
    timestampCounter = 0
    if os.path.exists(myFname) == True: #if the settings file
exists
        with open(myFname, "r") as f:
            lns = f.read()
            f.close()
            lns = lns.split('\n')
            if(len(lns) > 1): #if there's more than 1, first is
description, toss it
                lns = lns[1:]
            for ln in lns: #allow ANY number to be loaded from file...
                ln = ln.strip()
                ln = ln.split(",")
                if len(ln) == 4:
                    try:
                        i0 = int(ln[0])
                        i1 = int(ln[1])
                        i2 = int(ln[2])
                        i3 = int(ln[3])

vlayout.addItem(self.getWeekModule(timestampCounter, i0, i1, i2, i3, enabled
=True))
                            timestampCounter += 1
                    except:
                        pass #ignore lines where parsing failed...
                        #And now add some blank ones to get up to 8
                else:
                    QMessageBox.warning(None, 'No settings file found...', 'No
settings file was found in the local folder.\n\nA blank template will
be loaded.')
                    while timestampCounter < 8:
                        vlayout.addItem(self.getWeekModule(timestampCounter))
                        timestampCounter += 1
#-----Deal with all clicks-----
def allOtherClicks(self, obName): #only called when state is
changed
    xtitle = self.findChild(QLabel, name="titleLabel")
    xtitle.setText("<h2>Window Times: <font color = \"red\"
size=\\\"4\\\">(Please click \\\"Apply Changes\\\" to load new times onto
window)</font></h2>")
    timestampNumStr = obName.split("_")[1]
    objectType = obName.split("_")[0]
    dayNamez1 = ["M", "T", "W", "Th", "F", "Sa", "Su"]
#-----If this in an Enable click, do appropriate
enabling/disabling-----
    if objectType == "Enabled": #flip state of associated objects

```

```

        newState = self.findChild(QCheckBox,
name=obName).isChecked()
        for dayName in dayNamez1:
            self.findChild(QCheckBox,
name=dayName+"_" +timestampNumStr).setEnabled(newState)
            self.findChild(QTimeEdit,
name="Time_" +timestampNumStr).setEnabled(newState)
            self.findChild(QComboBox,
name="Type_" +timestampNumStr).setEnabled(newState)
            for iii in range(4):
                self.findChild(QLabel, name="Byte
"+str(iii)+"_" +timestampNumStr).setEnabled(newState)
            return
        #-----If this is a "time" click, update bytes 1 and 2
        if objectType == "Time":
            tempTimeObj = self.findChild(QTimeEdit, name=obName)
            myQTime = tempTimeObj.time() #returns a "QTime
            #byte 1 is hour:
            (self.findChild(QLabel, name="Byte
1_" +timestampNumStr)).setText(str(myQTime.hour()))
            #byte 2 is minute:
            (self.findChild(QLabel, name="Byte
2_" +timestampNumStr)).setText(str(myQTime.minute()))
            return
        #-----If this is an "open/close" click, update byte 3
        if objectType == "Type":
            (self.findChild(QLabel, name="Byte
3_" +timestampNumStr)).setText(str((self.findChild(QComboBox,
name=obName)).currentIndex()))
            return
        #-----If this is M-F click, update byte 0
        if dayNamez1.count(objectType) >= 1:
            M_checked = int(self.findChild(QCheckBox,
name="M_" +timestampNumStr).isChecked())
            T_checked = int(self.findChild(QCheckBox,
name="T_" +timestampNumStr).isChecked())
            W_checked = int(self.findChild(QCheckBox,
name="W_" +timestampNumStr).isChecked())
            Th_checked = int(self.findChild(QCheckBox,
name="Th_" +timestampNumStr).isChecked())
            F_checked = int(self.findChild(QCheckBox,
name="F_" +timestampNumStr).isChecked())
            Sa_checked = int(self.findChild(QCheckBox,
name="Sa_" +timestampNumStr).isChecked())
            Su_checked = int(self.findChild(QCheckBox,
name="Su_" +timestampNumStr).isChecked())
            addItUp = Su_checked + 2*M_checked + 4*T_checked +
8*W_checked
            addItUp += 16*Th_checked + 32*F_checked + 64*Sa_checked
            (self.findChild(QLabel, name="Byte
0_" +timestampNumStr)).setText(str(addItUp))
            return
        def changeDescClicked(self):
            myTuple = QDialogBox.getText(None, 'Change Name', 'Please enter
a Name\n(Name will be truncated to 10
characters)', QLineEdit.Normal, self.getDescription())

```



```
Title_qlabel.setObjectName("titleLabel")
Title_qlabel.setMaximumHeight(30)
vlayout.addWidget(Title_qlabel)
#Load up the old times from file:
self.loadFromFile(vlayout)
#Add "Apply Changes" Button:
submitButton = QPushButton("Apply Changes")
self.connect(submitButton, SIGNAL("clicked()"),
self.submitButtonClicked)
submitButton.setMaximumWidth(140)
submitButton.setLayoutDirection(Qt.RightToLeft)
vlayout.addWidget(submitButton)
#Add "Exit" Button:
exitButton = QPushButton("Exit")
self.connect(exitButton, SIGNAL("clicked()"), self.exitClicked)
exitButton.setMaximumWidth(140)
exitButton.setLayoutDirection(Qt.RightToLeft)
vlayout.addWidget(exitButton)
#Add spacer
mySpacer = QLabel()
vlayout.addWidget(mySpacer)
#Set layout
self.setLayout(vlayout)
```

SplashForm.py

```
from __future__ import division
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.uic import *
from math import *
#new:
import platform
import d2xx

class SplashForm(QWidget):
    def __init__(self, parent=None):
        self.paused = False #####
        super(SplashForm, self).__init__(parent)
        self.current_pic_number = 0
        nAcross = 20
        nDown = 10
        layout = QGridLayout()
        self.QL = []
        self.QL_logo = QLabel()
        self.QL_logo.setPixmap(QPixmap("windowLogo.png"))
        layout.addWidget(self.QL_logo,1,1,nDown-1,nAcross-1)
        for i in range(0,nAcross):
            self.QL.append(QLabel())
            self.QL[-1].setPixmap(QPixmap("dot0.png"))
            layout.addWidget(self.QL[-1],0,i)
        for i in range(0,nDown):
            self.QL.append(QLabel())
            self.QL[-1].setPixmap(QPixmap("dot0.png"))
            layout.addWidget(self.QL[-1],i,nAcross)
        for i in range(nAcross,0,-1):
            self.QL.append(QLabel())
            self.QL[-1].setPixmap(QPixmap("dot0.png"))
```

```
        layout.addWidget(self.QL[-1],nDown,i)
    for i in range(nDown,0,-1):
        self.QL.append(QLabel())
        self.QL[-1].setPixmap(QPixmap("dot0.png"))
        layout.addWidget(self.QL[-1],i,0)
    loadMsg = QLabel("<font color=red size=32><i>Establishing  
window communications. Please Wait...</i></font>")
    layout.addWidget(loadMsg,9,1,1,20)
    self.setWindowFlags(Qt.SplashScreen)
    self.setLayout(layout)
    #for waiting and updating the GUI
    self.timer = QBasicTimer()
    self.step = 20 + 20 + 10 + 5
    self.timer.start(60,self)
def timerEvent(self,event):
    if(self.paused == False): #####
        self.step += 1
        self.advanceSplashScreenGraphic(self.step)
def advanceSplashScreenGraphic(self, i):
    self.QL[i%len(self.QL)].setPixmap(QPixmap("dot" +
str(self.current_pic_number) + ".png"))
    if i%(20 + 20 + 10 + 10) == 0:
        self.current_pic_number = (self.current_pic_number + 1)%5
def pause(self): #####
    self.paused = True
def resume(self): #####
    self.paused = False
```

6.2.3 Android Software

Google Android Code

SmartWindows.java

```
package nd.seniordesign;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;

import android.app.Activity;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.MenuItem.OnMenuItemClickListener;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.Gallery;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

public class SmartWindows extends Activity implements OnClickListener {

    //***** CONSTANTS *****

    public int WIN_1 = 1;
    public int WIN_2 = 2;
    public int MY_OPEN = -1;
    public int MY_MIDDLE = 0;
    public int MY_CLOSE = 1;

    public String WIN_1_ID = "1267301014";
    public String WIN_2_ID = "1267369485";
    public String CMD_OPEN = "&rating=-1";
    public String CMD_MIDDLE = "&rating=0";
    public String CMD_CLOSE = "&rating=1";
    public String CMD_CORE =
"http://129.74.154.171/observer/submit_rating.php?skey=041a09cd46ac0093
246c8917-5619109&uid=5619109&mid=";

    public String COMP_CMD_OPEN_WIN_1 = CMD_CORE + WIN_1_ID +
CMD_OPEN;
    public String COMP_CMD_MIDDLE_WIN_1 = CMD_CORE + WIN_1_ID +
CMD_MIDDLE;
}
```

```

    public String COMP_CMD_CLOSE_WIN_1 = CMD_CORE + WIN_1_ID +
CMD_CLOSE;
    public String COMP_CMD_OPEN_WIN_2 = CMD_CORE + WIN_2_ID +
CMD_OPEN;
    public String COMP_CMD_MIDDLE_WIN_2 = CMD_CORE + WIN_2_ID +
CMD_MIDDLE;
    public String COMP_CMD_CLOSE_WIN_2 = CMD_CORE + WIN_2_ID +
CMD_CLOSE;
    //*****

    Button oButton = null;
    Button mButton = null;
    Button cButton = null;
    TextView win_name_label = null;

    //Set on menu item click
    public int currentWindow = 1;
    public int currentCommand = -1;
    //Set after button click, before Async task begins
    public String currentUrlStr = "";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.window_1_xml);

        win_name_label = (TextView)
this.findViewById(R.id.windowNameLabel);
        win_name_label.setTextSize((float) 26.0);

        oButton = (Button) this.findViewById(R.id.openButton);
        mButton = (Button) this.findViewById(R.id.middleButton);
        cButton = (Button) this.findViewById(R.id.closeButton);

        oButton.setTag("openButton");
        mButton.setTag("middleButton");
        cButton.setTag("closeButton");

        oButton.setOnClickListener(this);
        mButton.setOnClickListener(this);
        cButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if(v.getTag().toString().contains("open")) {
            this.currentCommand = -1;
            new async0().execute("");
            Toast t = Toast.makeText(this, "Open command
sending...", Toast.LENGTH_SHORT);
            t.show();
        }else if(v.getTag().toString().contains("middle")) {
            this.currentCommand = 0;
            new async0().execute("");
            Toast t = Toast.makeText(this, "Middle command
sending...", Toast.LENGTH_SHORT);

```

```

        t.show();
    }else if(v.getTag().toString().contains("close")) {
        this.currentCommand = 1;
        new async0().execute("");
        Toast t = Toast.makeText(this, "Close command
sending...", Toast.LENGTH_SHORT);
        t.show();
    }
}

//***** URL get (within AsyncTask wrapper)
*****
public class async0 extends AsyncTask<String, String, String> {

    URL url = null;
    HttpURLConnection urlConn = null;
    InputStreamReader isr = null;
    BufferedReader in = null;
    String inputLine = null;
    boolean epicFail = false;

    protected void onPreExecute() {

        if((SmartWindows.this.currentCommand == -1) &&
(SmartWindows.this.currentWindow == 1))
            SmartWindows.this.currentUrlStr =
COMP_CMD_OPEN_WIN_1;
        else if((SmartWindows.this.currentCommand == 0) &&
(SmartWindows.this.currentWindow == 1))
            SmartWindows.this.currentUrlStr =
COMP_CMD_MIDDLE_WIN_1;
        else if((SmartWindows.this.currentCommand == 1) &&
(SmartWindows.this.currentWindow == 1))
            SmartWindows.this.currentUrlStr =
COMP_CMD_CLOSE_WIN_1;
        else if((SmartWindows.this.currentCommand == -1) &&
(SmartWindows.this.currentWindow == 2))
            SmartWindows.this.currentUrlStr =
COMP_CMD_OPEN_WIN_2;
        else if((SmartWindows.this.currentCommand == 0) &&
(SmartWindows.this.currentWindow == 2))
            SmartWindows.this.currentUrlStr =
COMP_CMD_MIDDLE_WIN_2;
        else if((SmartWindows.this.currentCommand == 1) &&
(SmartWindows.this.currentWindow == 2))
            SmartWindows.this.currentUrlStr =
COMP_CMD_CLOSE_WIN_2;

        super.onPreExecute();
    }

    protected String doInBackground(String... params) {
        try {
            url = new
URL(SmartWindows.this.currentUrlStr);

```

```

        urlConn      = (URLConnection)
url.openConnection();
        isr          = new
InputStreamReader(urlConn.getInputStream());
        in           = new BufferedReader(isr);
        inputLine    = in.readLine();
        in.close();
        urlConn.disconnect();
    } catch (Exception e3) {
        Log.v("ANDY", "exception 3");
    }
    return "";
}

protected void onPostExecute(String result) {
    String msg;
    if (epicFail) {
        msg = "Failed to deliver command, sorry";
    } else {
        msg = "Sent!";
    }
    Toast t = Toast.makeText(SmartWindows.this, msg,
Toast.LENGTH_SHORT);
    t.show();
    super.onPostExecute(result);
}

//***** MENU
*****
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.layout.menu_all_media, menu);

    MenuItem menu_item_win_1 = menu.getItem(0);
    MenuItem menu_item_win_2 = menu.getItem(1);

    menu_item_win_1.setOnMenuItemClickListener(new
OnMenuItemClickListener() {
        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            if (SmartWindows.this.win_name_label != null) {
                if (currentWindow == 1) { //if we're
already on win 1
                    Toast t =
Toast.makeText(SmartWindows.this, "Already set to window 1",
Toast.LENGTH_SHORT);
                    t.show();
                }
                else {
                    SmartWindows.this.win_name_label.setText("Window 1");
                    currentWindow = 1;
                }
            }
        }
    });
}

```



```

    public static final int closeButton=0x7f050006;
    public static final int menu_button_win_1=0x7f050000;
    public static final int menu_button_win_2=0x7f050001;
    public static final int middleButton=0x7f050005;
    public static final int openButton=0x7f050004;
    public static final int otherLabel=0x7f050002;
    public static final int windowNameLabel=0x7f050003;
}
public static final class layout {
    public static final int menu_all_media=0x7f030000;
    public static final int window_1_xml=0x7f030001;
}
public static final class string {
    public static final int app_name=0x7f040000;
}
}

                                menu_all_media.xml
<menu xmlns:android="http://schemas.android.com/apk/res/android"
android:name="Context Menu">
<item
    android:id="@+id/menu_button_win_1"
    android:icon="@drawable/win_icon"
    android:numericShortcut="1"
    android:title="Window 1"
>
</item>
<item
    android:id="@+id/menu_button_win_2"
    android:icon="@drawable/win_icon"
    android:numericShortcut="2"
    android:title="Window 2"
>
</item>
</menu>

                                window_1_xml.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>
<ImageView
    android:layout_width="200px"
    android:layout_height="100px"
    android:src="@drawable/window_logo"
    android:layout_gravity="center"
/>
<TextView
    android:id="@+id/otherLabel"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Press menu to select alternate window..."
/>
<TextView

```



```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text=" "
    />
    <TextView
        android:id="@+id/windowNameLabel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Window 1"
        android:gravity="center"
    />
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="150px"
        android:src="@drawable/window_pic"
    />
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="200px"
        >
        <!-- android:src="@drawable/open" -->
        <!-- android:src="@drawable/half" -->
        <!-- android:src="@drawable/close" -->

        <Button
            android:id="@+id/openButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Open"
            android:layout_weight="1"
        />
        <Button
            android:id="@+id/middleButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Middle"
            android:layout_weight="1"
        />
        <Button
            android:id="@+id/closeButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Close"
            android:layout_weight="1"
        />
    </LinearLayout>
</LinearLayout>
```

6.3 Bill of Materials

Board	Subsystem	Part Description	Supplier	Part #	PCU	OWU	RCU	#	Price/Unit	Cost	Link	Manufacturer	Manufacturer part number
Light		10k Resistor	Stock	Stock	0	1	0	2	Stock				
Light		Photodiode	Digikey	751-1055-1-ND	0	1	0	2	\$1.44	\$2.88	http://se	Vishay/Semicon	TEM76000X01
Light		Molex 3-pin (M)	Digikey	WM4301-ND	0	1	0	2	\$1.17	\$2.34	http://se	Molex Inc	22-05-3031
Light		Molex 3-pin (F)	Digikey	WM2001-ND	0	1	0	2	\$0.60	\$1.20	http://se	Molex Inc	22-01-3037
Light		Molex Crimp Pins	Digikey	WM1114-ND	0	3	0	6	\$0.27	\$1.62	http://se	Molex Inc	08-50-0114
Main	Buttons	Push Button Switch	Digikey	EG2021-ND	0	3	5	11	\$0.59	\$6.49	http://se	E-Switch	PS1024ALBLK
Main	Buttons	10K resistor	Digikey	P10.0KCCT-ND	0	3	5	11	\$0.01	\$0.11	http://se	Panasonic - ECG	ERJ-6ENF1002V
Main	Buttons	Molex 8-pin (M)	Digikey	WM4306-ND	0	1	1	3	\$1.30	\$3.90	http://se	Molex Inc	22-05-3081
Main	Buttons	Molex 8-pin (F)	Digikey	WM2006-ND	0	1	1	3	\$0.60	\$1.80	http://se	Molex Inc	22-01-3087
Main	Buttons	Molex Crimp Pins	Digikey	WM1114-ND	0	8	8	24	\$0.60	\$14.40	http://se	Molex Inc	08-50-0114
Main	DC/DC	DC/DC Charge Pump	Digikey	MCP1252-33X50I/MS-ND	0	1	1	3	Purchased		http://se	Microchip Technology	MCP1252-33X50I/MS
Main	DC/DC	10 uF Capacitor	Digikey	445-1372-1-ND	0	2	2	6	\$1.43	\$8.58	http://se	TDK Corporation	C2012Y5V0J106Z
Main	DC/DC	100 nF Capacitor	Digikey	PCC1853CT-ND	0	1	1	3	\$0.26	\$0.78	http://se	Panasonic - ECG	ECJ-2VF1E104Z
Main	DC/DC	150K Resistor	Digikey	P150KACT-ND	0	1	1	3	Stock		http://se	Panasonic - ECG	ERJ-6GEYJ154V
Main	EEPROM	EEPROM	Digikey	25LC640A-I/SN-ND	0	0	0	0	Purchased		http://se	Microchip Technology	25LC640A-I/SN
Main	LCD	New Haven Serial Display	Digikey	NHD-0216K3Z-NSW-BBW	0	0	1	1	\$20.75	\$20.75	http://se	Newhaven Display Intl	NHD-0216K3Z-NSW-BBW
Main	LCD	Molex 6-pin (M)	Digikey	WM4304-ND	0	0	1	1	\$1.07	\$1.07	http://se	Molex Inc	22-05-3061
Main	LCD	Molex 6-pin (F)	Digikey	WM2004-ND	0	0	1	1	\$0.68	\$0.68	http://se	Molex Inc	22-01-3067
Main	LCD	Molex Crimp Pins	Digikey	WM1114-ND	0	0	6	6	\$0.27	\$1.62	http://se	Molex Inc	08-50-0114
Main	Light	Molex 3-pin (M)	Digikey	WM4301-ND	0	1	0	2	\$1.17	\$2.34	http://se	Molex Inc	22-05-3031
Main	Light	Molex 3-pin (F)	Digikey	WM2001-ND	0	1	0	2	\$0.60	\$1.20	http://se	Molex Inc	22-01-3037
Main	Light	Molex Crimp Pins	Digikey	WM1114-ND	0	3	0	6	\$0.27	\$1.62	http://se	Molex Inc	08-50-0114
Main	Limit	10K resistor	Digikey	P10.0KCCT-ND	0	2	0	4	Stock		http://se	Panasonic - ECG	ERJ-6ENF1002V
Main	Limit	47 resistor	Digikey	P47.0CCT-ND	0	1	0	2	Stock		http://se	Panasonic - ECG	ERJ-6ENF47R0V
Main	Limit	1k resistor	Digikey	P1.0KACT-ND	0	1	0	2	Stock		http://se	Panasonic - ECG	ERJ-6GEYJ102V
Main	Limit	Molex 6-pin (M)	Digikey	WM4304-ND	0	1	0	2	\$1.07	\$2.14	http://se	Molex Inc	22-05-3061
Main	Limit	Molex 6-pin (F)	Digikey	WM2004-ND	0	1	0	2	\$0.68	\$1.36	http://se	Molex Inc	22-01-3067
Main	Limit	Molex Crimp Pins	Digikey	WM1114-ND	0	6	0	12	\$0.27	\$3.24	http://se	Molex Inc	08-50-0114
Main	ucontroller	MicroController	Digikey	PIC18LF4620-I/PT-ND	1	1	1	4	Purchased		http://se	Microchip Technology	PIC18LF4620-I/PT
Main	ucontroller	10-pin header	Stock	Stock	1	1	1	4	Stock				
Main	ucontroller	10K resistor	Digikey	P10.0KCCT-ND	1	1	1	4	Stock		http://se	Panasonic - ECG	ERJ-6ENF1002V
Main	ucontroller	Push Button Switch	Digikey	450-1655-ND	1	1	1	4	Stock		http://se	Tyco Electronics	FSMCDAH
Main	ucontroller	100 Resistor	Digikey	P100DACT-ND	1	1	1	4	Stock		http://se	Panasonic - ECG	ERA-6AEB101V
Main	ucontroller	1N4148	Digikey	1N4148WS-FDICT-ND	1	1	1	4	Stock		http://se	Diodes Inc	1N4148WS-7-F
Main	ucontroller	0.1uF Capacitor	Digikey	490-1723-1-ND	1	1	1	4	\$0.05	\$0.20	http://se	Murata Electronics North America	GRM219F51H104ZA01D
Main	ucontroller	20MHz Ceramic Oscillator	Digikey	490-4717-1-ND	1	1	1	4	Stock		http://se	Murata Electronics North America	CSTCE20M0V53Z-R0
Main	ucontroller	30pF Capacitor	Digikey	478-3738-1-ND	2	2	2	8	\$0.02	\$0.16	http://se	AVX Corporation	08051A300JAT2A
Main	ucontroller	1M Resistor	Digikey	P1.0MACT-ND	1	1	1	4	Stock		http://se	Panasonic - ECG	ERJ-6GEYJ105V
Main	ucontroller	Poly Case Project Box	Poly Case	SL-64P (yes- mounting boss)	1	1	1	4	\$4.11	\$16.44	http://ww	Polycase	SL640
Main	Power	15K Resistor	Digikey	P15KACT-ND	0	1	1	3	Stock		http://se	Panasonic - ECG	ERJ-6GEYJ153V
Main	Power	10K resistor	Digikey	P10.0KCCT-ND	0	1	1	3	Stock		http://se	Panasonic - ECG	ERJ-6ENF1002V
Main	Power	100K Resistor	Digikey	P100KATR-ND	0	1	1	3	Stock		http://se	Panasonic - ECG	ERJ-6GEYJ104V
Main	Power	DC Power Jack (1.3mm) (M)			0	1	1	3	Stock		http://ww	VARIOUS	420ECS
Main	Power	DC Power Plug (1.3mm) (F)			0	1	1	3	Stock		http://ww	JAMECO VALUE	G1P639-R

Smart Windows
Daniels, Haunert, Shilling, Spangler

Main	Power	100 nF Capacitor	Digikey	490-1723-1-ND	3	3	3	12	\$0.05	\$0.60	http://se	Murata Electronics North America	GRM219F51H104ZA01D
Main	Power	22 uF Capacitor	Digikey	490-1719-1-ND	1	1	1	4	\$0.05	\$0.20	http://se	Murata Electronics North America	GRM21BR60J226ME39L
Main	Power	10 uF Capacitor	Digikey	445-1372-1-ND	1	1	1	4	\$1.43	\$5.72	http://se	TDK Corporation	C2012Y5V0J106Z
Main	Power	3.3V Voltage Regulator	Digikey	ZLDO1117G33DICT-ND	1	1	1	4	\$2.33	\$9.32	http://se	Diodes Inc	ZLDO1117G33TA
Main	Power	Slide Switch	Digikey	EG1903-ND	1	1	1	4	Stock		http://se	E-Switch	EG1218
Main	RTC	Crystal	Digikey	SER3205-ND	1	1	0	3	\$0.32	\$0.96	http://se	Epson Toyocom Corporation	C-002RX 32.7680K-E:PBFFREE
Main	RTC	Real Time Clock	Digikey	DS1305E+-ND	1	1	0	3	Purchased		http://se	Maxim Integrated	DS1305E+
Main	RTC	10K resistor	Digikey	P10.0KCCT-ND	2	2	0	6	Stock		http://se	Panasonic - ECG	ERJ-6ENF1002V
Main	USB	USB - Serial UART	Digikey	768-1007-1-ND	1	1	1	4	\$4.50	\$18.00	http://se	FTDI	FT232RL R
Main	USB	Ferrite Bead			1	1	1	4	Stock				
Main	USB	USB - Type B	Digikey	151-1121-ND	1	1	1	4	\$0.93	\$3.72	http://se	EDAC Inc	690-004-221-023
Main	USB	4.7k Resistor	Digikey	P4.7KACT-ND	1	1	1	4	Stock		http://se	Panasonic - ECG	ERJ-6GEYJ472V
Main	USB	10k Resistor	Digikey	P10.0KCCT-ND	1	1	1	4	Stock		http://se	Panasonic - ECG	ERJ-6ENF1002V
Main	USB	100nF Capacitor	Digikey	490-1723-1-ND	1	1	1	4	\$0.05	\$0.20	http://se	Murata Electronics North America	GRM219F51H104ZA01D
Main	USB	0.1uF Capacitor	Digikey	490-1723-1-ND	1	1	1	4	\$0.05	\$0.20	http://se	Murata Electronics North America	GRM219F51H104ZA01D
Main	USB	4.7uF Capacitor	Digikey	490-3901-1-ND	1	1	1	4	\$0.25	\$1.00	http://se	Murata Electronics	GRM219C81A475KE34D
Main	ZigBee	Antenna (LINX24ANT)			1	1	1	4	Stock				
Main	ZigBee	Transceiver Chip	Digikey	AT86RF230-ZU-ND	1	1	1	4	Purchased		http://se	Atmel	AT86RF230-ZU
Main	ZigBee	BALLIN 0805			1	1	1	4	Stock				
Main	ZigBee	ABRACKY			1	1	1	4	Stock				
Main	ZigBee	1uF Capacitor	Digikey	490-1700-2-ND	4	4	4	16	\$0.05	\$0.80	http://se	Murata Electronics	GRM216R61E105KA12D
Main	ZigBee	10pF Capacitor	Digikey	490-1590-1-ND	2	2	2	8	\$0.22	\$1.76	http://se	Murata Electronics	GRM2195C2A100JZ01D
Main	ZigBee	22pF Capacitor	Digikey	490-1591-1-ND	2	2	2	8	\$0.22	\$1.76	http://se	Murata Electronics	GRM2195C2A220JZ01D
Main	ZigBee	5.6pF Capacitor	Digikey	478-1301-1-ND	2	2	2	8	\$0.23	\$1.84	http://se	AVX Corporation	08055A5R6CAT2A
Motor		H-bridge (L298)	Digikey	497-1395-5-ND	0	1	0	2	Purchased		http://se	STMicroelectronics	L298N
Motor		DC Power Jack (1.3mm) (M)			0	1	0	2	Stock		http://w	VARIOUS	420ECS
Motor		DC Power Plug (1.3mm) (F)			0	1	0	2	Stock		http://w	JAMECO VALUE	G1P639-R
Motor		Molex 4-pin (M)	Digikey	WM4302-ND	0	1	0	2	\$1.14	\$2.28	http://se	Molex Inc	22-05-3041
Motor		Molex 4-pin (F)	Digikey	WM2002-ND	0	1	0	2	\$0.63	\$1.26	http://se	Molex Inc	22-01-3047
Motor		Molex Crimp Pins	Digikey	WM1114-ND	0	6	0	12	\$0.27	\$3.24	http://se	Molex Inc	08-50-0114
Motor		Tyco Screw Terminal	Digikey	A98167-ND	0	1	0	2	Stock		http://se	Tyco Electronics	284392-3
Motor		100 nF Capacitor	Digikey	490-1723-1-ND	0	2	0	4	\$0.05	\$0.20	http://se	Murata Electronics North America	GRM219F51H104ZA01D
Motor		1N4004 Diode	Digikey	1N4004FSCT-ND	0	1	0	2	Stock		http://se	Fairchild Semico	1N4004
Motor		Poly Case Project Box	Poly Cas	BF-1502012 (Lid Option)	0	1	0	2	\$1.01	\$2.02	http://w	Polycase	BF-1502012
None		Motor	Solarbot	GM3	0	1	0	2	Purchased		http://w	Solarbotics	GM3
None		Steal Rod	Lowe's	Lowe's	0	1	0	2	Purchased				
None		Microswitch	Digikey	CKN9940-ND	0	2	0	4	\$2.27	\$9.08	http://se	C&K Component	ZMCHM9L3T
None		23" x 42" White Mini Blin	Lowe's	168349	0	1	0	2	\$8.96	\$17.92	http://w	Levolor	LVYCCDD2304201D
None		Rubber Spider	Jameco	162000	0	1	0	2	\$1.55		http://w	VARIOUS	M01-0004
None		.125 ID Hub	Jameco	162288	0	1	0	2	\$1.49	\$2.98	http://w	VARIOUS	162288
None		.197 ID Hub	Jameco	161998	0	1	0	2	\$1.19	\$2.38	http://w	VARIOUS	M01-0002
None		8 Battery Holder	Digikey	BH48AAW-ND	0	1	0	2	\$1.91	\$3.82	http://se	MPD (Memory P	BH48AAW
None		NiMH AA Battery	Digikey	N703-ND	0	8	0	16	\$3.96	\$63.36	http://se	Energizer	NH15
None		Battery Charger	Stock	Stock	0	1	0	2	Stock				
None		Photo-interruptor	Digikey	425-1971-5	0	1	0	2	Purchased		http://se	Sharp Microelect	GP1S52VJ000F
None		9-V Battery Holder	Digikey	377-1549-ND	0	0	1	1	Stock		http://se	Bud Industries	HH-3449
									\$251.54				

6.4 Data Sheets

The following links lead to data sheet for all major electronic components in our project.
The shorter data sheets have been attached to this report.

Part Description	Digikey Part #	Link
Photodiode	751-1055-1-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=751-1055-1-ND
FTDI Serial UART	768-1007-1-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=768-1007-1-ND+
New Haven Serial Display	NHD-0216K3Z- NSW-BBW	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=NHD-0216K3Z-NSW-BBW+
Crystal Oscillator	SER3205-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=SER3205-ND+
3.3V Voltage Regulator	ZLDO1117G33DIC T-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=ZLDO1117G33DICT-ND
20MHz Ceramic Oscillator	490-4717-1-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=490-4717-1-ND
EEPROM	25LC640A-I/SN-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=25LC640A-I/SN-ND
Photo-interruptor	425-1971-5	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=425-1971-5+
H-bridge (L298)	497-1395-5-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=497-1395-5-ND+
ZigBee Transceiver Chip	AT86RF230-ZU- ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=AT86RF230-ZU-ND
Real Time Clock	DS1305E+-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=DS1305E%2B-ND
DC/DC Charge Pump	MCP1252- 33X50I/MS-ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=MCP1252-33X50I/MS-ND
MicroController	PIC18LF4620-I/PT- ND	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=PIC18LF4620-I/PT-ND