|  |
| --- |
| Heritage College |
| ConnectedControl |
| Final Project II |
|  |
| **Daniel Joannis** |
| **5/7/2012** |

|  |
| --- |
|  |

# Executive Summary

ConnectedControl is a wireless programmable thermostat. What differentiates this thermostat from commercial options is the wireless connectivity to a computer. This allows the thermostat to be programmed from a computer, without being tethered to it in anyway. Although many thermostats are programmable with weekly schedules, the ability to program, view current temperature, and temporarily override settings from a computer provides a new level of control over your environment, with the potential for easy incorporation in home automation schemes.

The circuit has two human input sources: pushbuttons and a temperature sensor. The pushbuttons allow users to navigate the menus and configure the thermostat on the device. The temperature sensor detects ambient temperature and the temperature is read by the microcontroller, an ATMega1284p. The microcontroller compares the measured temperature with the current scheduled temperature, and will trigger a relay to turn on a heater if the temperature is too low. Status is displayed on a 128 by 64 pixel graphical LCD. A pair of XBee wireless modules connects the thermostat to a computer, which is running Windows software that facilitates the configuration of a weekly temperature schedule.

The circuit diagram (figures 2 through 5) attempts to maintain robust simplicity. The microcontroller’s internal pull-ups are used for the push-buttons, minimizing additional circuitry. Mainstream hardware, such as the DS1307 real-time clock, is used to simplify hardware and software design, as there is plenty of support available, either in libraries or example code and schematics. Because of the prevalence of this hardware, cost is, as a result, minimized.

The major components of the first semester timeline are hardware prototyping, circuit board manufacturing, firmware development, and hardware assembly. Second semester work introduced computer connectivity to the firmware, developed Windows software to interact with the hardware, and packaged the hardware in a manner that resembles a commercial product. The actual project completion date was April 30th, 2012.

The biggest challenges for this project was designing a functional, user-friendly interface that takes advantage of the graphical LCD, and implementing a communications scheme between the hardware and the Windows application. However, these challenges were met successfully, and ConnectedControl is a complete, functional product.

Table of Contents

[Executive Summary 2](#_Toc324178859)

[Introduction 4](#_Toc324178860)

[Functional Description 5](#_Toc324178861)

[Circuit Design 6](#_Toc324178862)

[Power Supply 7](#_Toc324178863)

[Microcontroller 7](#_Toc324178864)

[Peripherals 7](#_Toc324178865)

[Graphical LCD Display 8](#_Toc324178866)

[Absolute Maximum Ratings 9](#_Toc324178867)

[User Manual 9](#_Toc324178868)

[Circuit Diagram 12](#_Toc324178869)

[PCB Layout 16](#_Toc324178870)

[Bill of Materials 18](#_Toc324178871)

[Approximate Timeline 19](#_Toc324178872)

[Semester 1 19](#_Toc324178873)

[Semester 2 19](#_Toc324178874)

[Conclusion 20](#_Toc324178875)

# Introduction

Although common in buildings with a dedicated building management department, computer controlled temperature has not made it into residential environments in any big way. However, in a rapidly digitizing world, and with the prevalence of smart phones, home automation has the potential to make a big entrance to a mass market. The ability to control lights, television channels and which song is on your stereo, right from your Smartphone, is no longer fiction. Temperature is another parameter which could be incorporated in home automation.

Every modern thermostat includes the ability to configure a schedule. Users are able to pre-configure what temperature they would like their home to be, at what time of day, and what day of the week. Often advertised as being an energy saving feature, scheduling ultimately means convenience for the user. My final project, ConnectedControl, takes scheduling to another level. It allows a computer to configure the schedule from user input. This also leads to the ability to incorporate temperature control into a home automation setup. However, not only can a computer communicate with the thermostat, it does so wirelessly.

ConnectedControl features an LCD display to indicate relevant information, including current and target temperatures, current time, and menus configure the device. User input can be entered through the 5 buttons on the unit, or through the Windows GUI application. The load (heating apparatus) is controlled using a relay rated up to 16A.

There are manufacturers who do currently produce wireless thermostats for home use. However, these thermostats do not interface with a computer – they are simply remote controlled, like a television. Prices range from just under $100 to over $200, and some do not include necessary control units [1].

There are commercial offerings of wireless thermostats with a web interface. The Ecobee is one example [2]. It allows users to track energy consumption over the internet. However, this software is not designed to be modified by the user. Users must rely on an external service. My hardware and software can be adapted to cooperate with other system designs.

Important circuit design considerations were to ensure that the PCB traces can carry the high current necessary for the load. These high current traces were made short, fat, and on both sides of the board to meet this requirement. In terms of components, the circuit is relatively simple. A microcontroller reads input from buttons, a real-time clock, and a temperature sensor, and in turn triggers a relay. It interfaces with the computer using an Xbee wireless serial link.

# Functional Description

DS1307 RTC

**Figure 1**

Relay

Graphical LCD Display

XBee

JTAG

TMP36

Pushbuttons

ATMega1284p

The thermostat has two physical inputs: the 5 buttons (directional and confirmation - treated as a collective), and the temperature sensor (a TMP36). These are connected to the microcontroller, which interprets the inputs to perform tasks. The buttons are used to navigate the on-screen menu system, allowing the user to adjust temperature scheduling, override current temperature temporarily, and diagnostic features. The TMP36 sensor outputs an analog voltage that varies with the temperature. It will take input voltages from 2.7V to 5.5V, and can measure temperatures from -40 C to 125 C. The voltage increases in steps of 10mV/C, and ranges from 100mV to 2V [3].

The microprocessor used is an ATMega1284p. Maximum operating temperatures for this IC are from -55 C to 125 C [5]. Its input supply voltage range is from 1.8V to 5.5V, with an absolute maximum of 6V [5]. Input from the temperature sensor is compared to the programmed schedule, and when the temperature drops below the threshold, the microcontroller triggers the relay, which powers to load (likely a heater). The relay is rated to handle up to 16A at 277VAC, and consumes 40mA when energized [4].

Current status and configuration options are displayed on the 128\*64 graphical LCD. This display is controlled using an ST7565 IC, and requires input voltage of 2.4V – 3.3V [14]. It operates using 3.3V logic levels, and must be level shifted in order to work with this microcontroller at 5V. It can be operated from -10\*C to 60\*C [16], making this the least temperature tolerant component for the entire build.

A DS1307 real-time clock (RTC) is used to keep track of time, and communicates over I2C. It takes an input voltage from 4.5V – 5.5V, and is also connected to a 3V lithium cell battery to keep time even when power is lost [13].

Communication with the computer is performed wirelessly using two XBee modules. One module is connected to the computer using a USB-FTDI adapter. The other module is connected to the ATMega’s serial RX and TX ports. These modules require between 2.8V – 3.4V input, and can operate over a distance of 100 ft indoors at 1mW (2.4GHz) and in temperatures ranging from -40 C to +85 [6]. The wireless link is transparent to the computer and the micro – as far as they are concerned they are communicating over a serial link. The computer is to run Windows software that will transmit and read configuration data and status information from the thermostat.

# Circuit Design

The circuit schematic diagram has been divided into four pages, thus this section will analyze each section individually.

## Power Supply

The power supply in this circuit will output 5V and 3.3V. The LCD display and the XBee require 3.3V. The input of the power supply can be between 7.5V and 35V (absolute maximum, + about 0.7V for diode) [7], and is passed through a 1N4001 diode. This diode can handle up to 1A current [8]. It serves to protect the circuit from improper input polarity. From the diode, the voltage is filtered by a 10uF electrolytic capacitor, to remove any ripple and some noise from the input. The power then enters the input of a 7805 5V, 1A [7] voltage regulator. This regulator was chosen because it is quite ubiquitous and has enough current supply to run this circuit. Output from the 5V regulator is them filtered using a 1uF capacitor.

Vcc is now input to the MCP1700-33 3.3V LDO regulator. The MCP1700 can take input up to 6V [9]. The 3.3V output is finally filtered by a 1uF capacitor.

This circuit contains 3 additional filtering capacitors, for the ATMega1284p, the DS1307 RTC, and for the XBee. These serve to reduce noise and spikes in the input signal, a problem that may be present during the switching of the relay.

## Microcontroller

The microcontroller used in this design is an ATMega1284p. It was chosen, simply, for the performance and feature set. The LCD display chosen requires 1KB RAM, which on many micros leaves very little RAM to software, and I was concerned that a micro such as the ATMega32 would not be sufficient, as it only has 2KB of RAM [10]. The ATMega1284p has 16KB of RAM, as well as 128KB of flash memory [5]. As well, with a higher clock speed of 20MHz [12] it should provide a higher refresh rate for the display.

The clock source chosen is a FOX H5C-2E 20MHz clock oscillator. It is in a square 8 pin DIP package that takes up less space on a PCB than other packages. It operates on 4.5V - 5.5V, and outputs a 20MHz clock [11].

A JTAG header will be present in the circuit to allow for programming and debugging, while the microcontroller is in the circuit. This will aid troubleshooting, as well as allow modifications to the code to be easily transferred to the micro.

## Peripherals

The pushbuttons used for menu navigation on the thermostat will make use of the internal pull-up resistors. As such, they are at one end connected to the microcontroller on PA1 through PA5, and on the other to ground.

The TMP36 temperature sensor was chosen because of the wide range of temperature operation and the convenient 10mV/\*C output [3]. It is connected to PA0 on the micro, which is an ADC pin. Although the first version of the PCB connected AREF on the ATMega to 5V, version 2 connects it to 3.3V, in order to provide more resolution to the ADC, and thus the temperature readings.

A 2N2222 NPN transistor serves as a pre-driver to the ALE75B05 relay. The relay consumes 40mA when active, which is well within the 2N2222’s capabilities [12]. A 4.7k resistor at the base of the transistor limits current drawn from the microcontroller, which outputs 5V to enable the relay. A 1N4001 diode is used as a flyback diode to suppress back EMF when the relay is de-energized. The relay is connected inline to the AC load.

In order to keep scheduling information in sync, a DS1307 real-time clock (RTC) is used. It communicates using I2C [13], and provides accurate timing for low cost. It is also a fairly ubiquitous IC, making library support very easy to obtain.

An XBee 2.4GHz radio is used to establish serial communication between the microcontroller and the computer. The XBee operates at 3.3V logic levels. Although the datasheet states that the pins are 5V tolerant [6], because there is a pin left over on the 4050 logic buffer/level converter (described later), the RX pin (which is connected to the micro TX pin) is still level converted to reduce risk of damage. There is no level conversion necessary for the XBee’s TX pin, because 3.3V logic HIGH is still a logic HIGH for 5V level logic. Two LEDs are also incorporated into the circuit, to indicate connection and transmission status.

## Graphical LCD Display

The LCD display in this circuit is controlled by an ST7565, a 3.3V logic level IC. As such, in order to control the display using the ATMega1284p at 5V, logic level shifting is required. To do this, a 4050 buffer is used, which also serves to shift the logic level down to 3.3V [15]. The buffer is connected to the MOSI, SCLK, A0, $RES$, and $CS1$ pins, which are all inputs to the display. There is one remaining pin on the 4050, which is used for the XBee as described above.

The LCD used also has an RGB backlight. These are connected through 330 Ohm resistors to PD2-PD4 on the microcontroller, and can be controlled to display different colours.

# Absolute Maximum Ratings

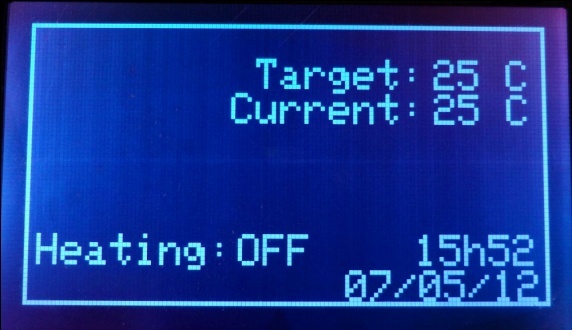
The absolute maximum ratings of the individual components are already outlined in the Functional Description section. However, I will provide the absolute maximum ratings of the project as a whole, accounting for all the components’ individual ratings.

**Input Voltage:** -50 – 35VDC

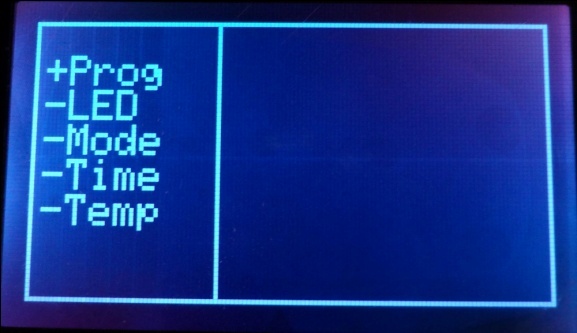
**Operating Temperature:** 0 – 60\*C

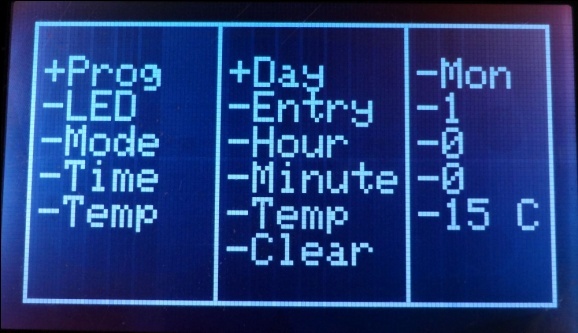
**Storage Temperature:** -20 – 70\*C

# User Manual

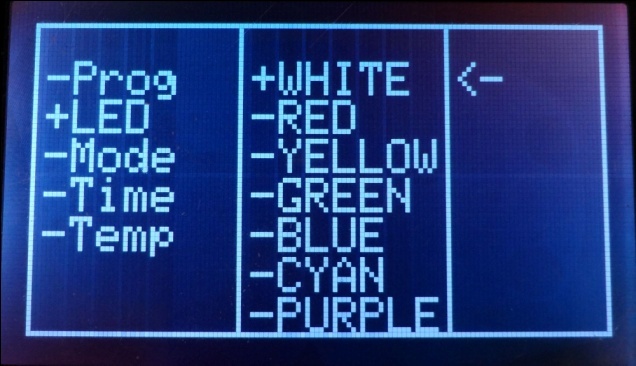
**Idle Screen:** This is the main screen, displaying current status information. You can see the current and target temperatures, operating mode, power status, and the time and date.

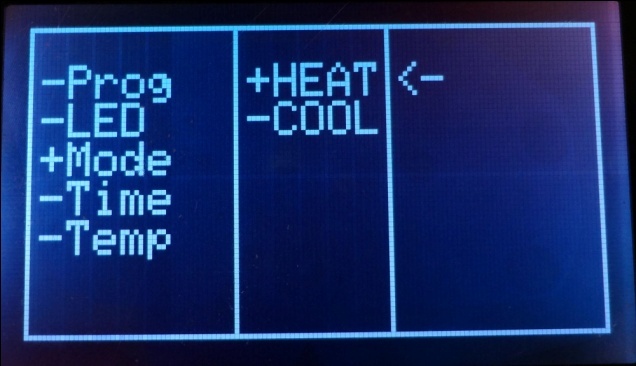
Pressing the center button will bring you into the menu.

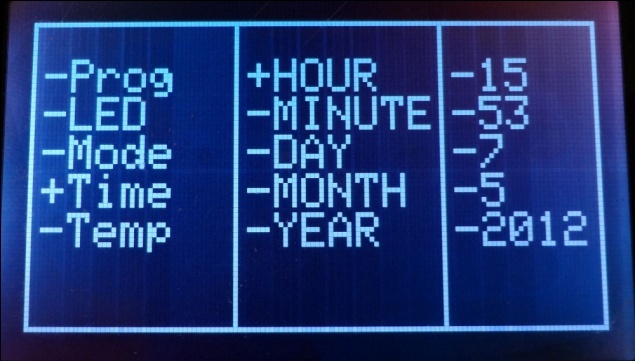
**Main Menu:** To navigate the menu, the up/down buttons will move the cursor up and down, and the right button will enter the currently selected menu. The back and center buttons will exit the menu.

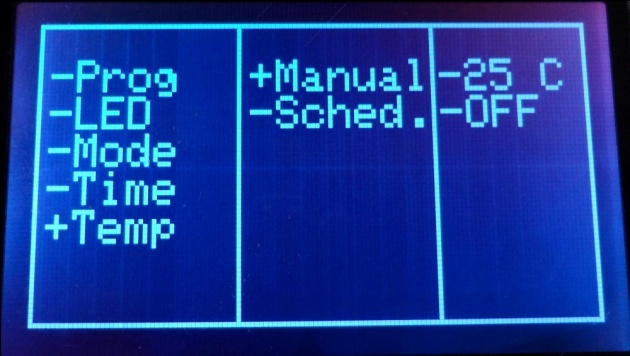
**Program Menu:** This menu is where you can enter your schedule into the thermostat. Navigation is the same, where you enter the next level with the right button, and go back a level with the left button.

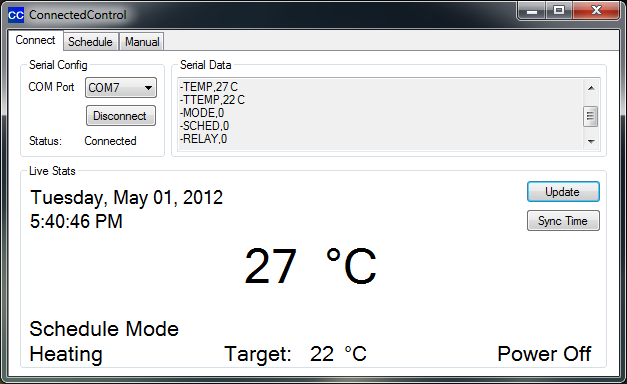
The Clear feature allows you to wipe all schedule information and start from scratch.

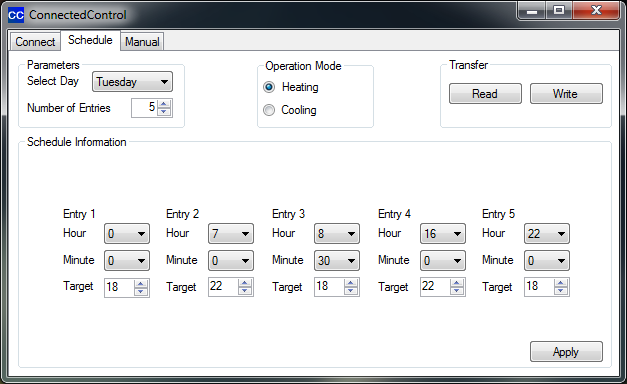
**LED Menu:** The backlight colour can be set in this menu. The <- arrow indicates the current setting. This setting is remember, even when power is lost.

**Mode Menu:** The mode menu allows you to switch from heating to cooling mode. This setting is also remembered on power loss.

**Time Menu:** The Real Time Clock can be set in this menu. Note that you can hold the up/down buttons to rapidly scroll through the numbers.

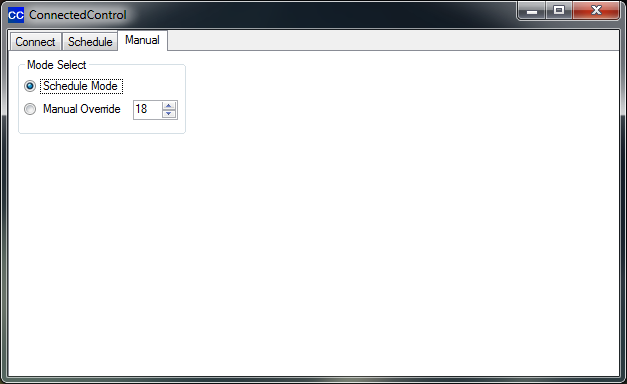
**Temperature Menu:** In this menu, you can switch from Scheduled to Manual target temperature. This allows you to override the schedule with a different temperature. When you no longer wish to manually override the temperature, you can turn on the Schedule again.

**Connect Tab**: This is the main tab, which allows you to connect to the device’s COM port. Here, you will see raw serial data for debugging, as well as live stats. There is a button to synchronize the time on the device with the computer time.

****

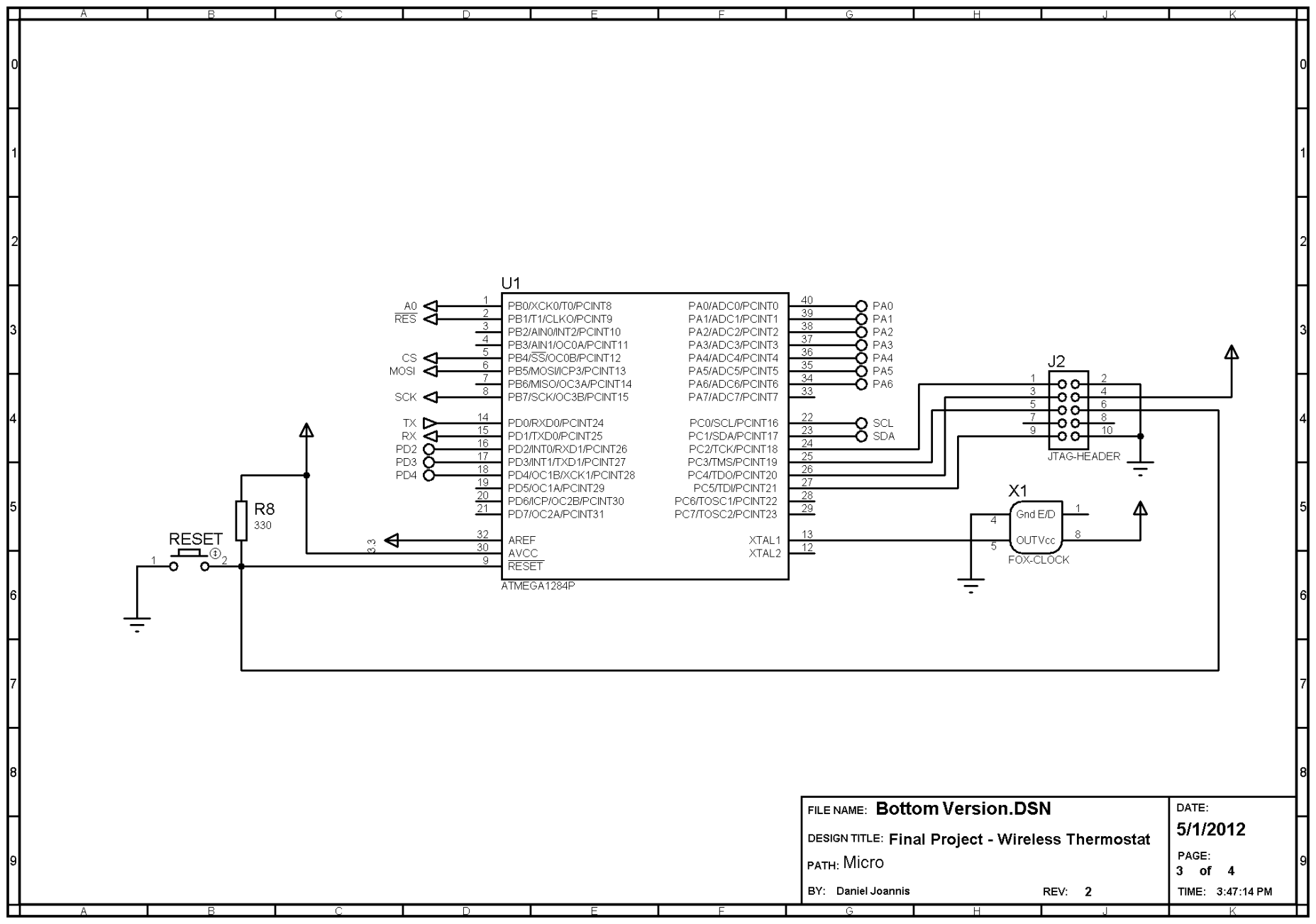
**Schedule Tab:** Here you can configure your schedule. Select the day to configure from the drop down, choose the number of entries you want for that day, and fill it out. Click Apply before switching to another day.

To simplify usage, hit Read to download the existing schedule from the device. This way, you can make small adjustments without having to make a complete schedule from nothing.

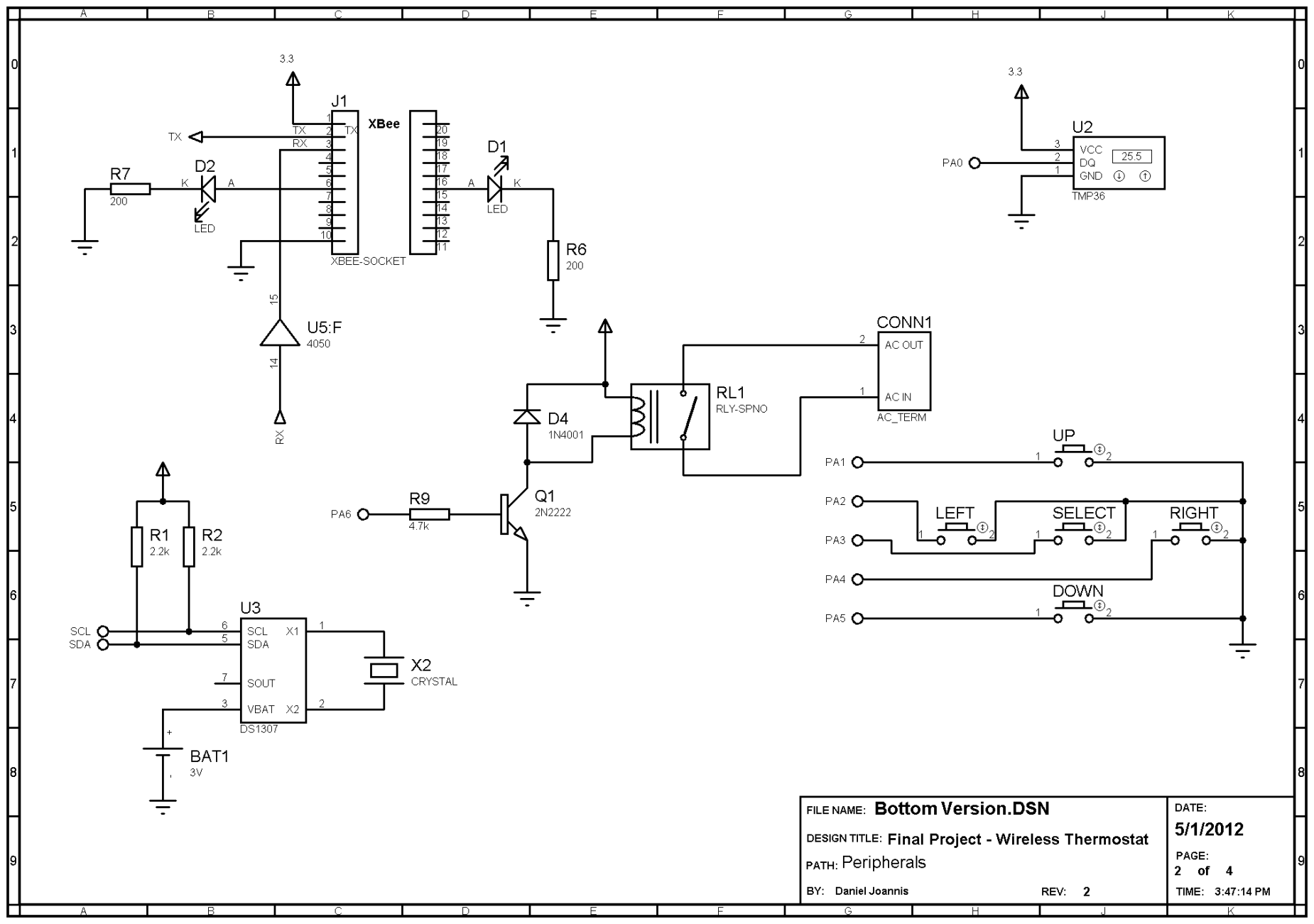
**Manual Tab:** This is a simple tab. You can switch from Schedule to Manual Override mode, and vice versa. Set the desired Override temperature in the numeric box, and switch to Manual Override. The new target temperature will be set and the mode will change to Manual.

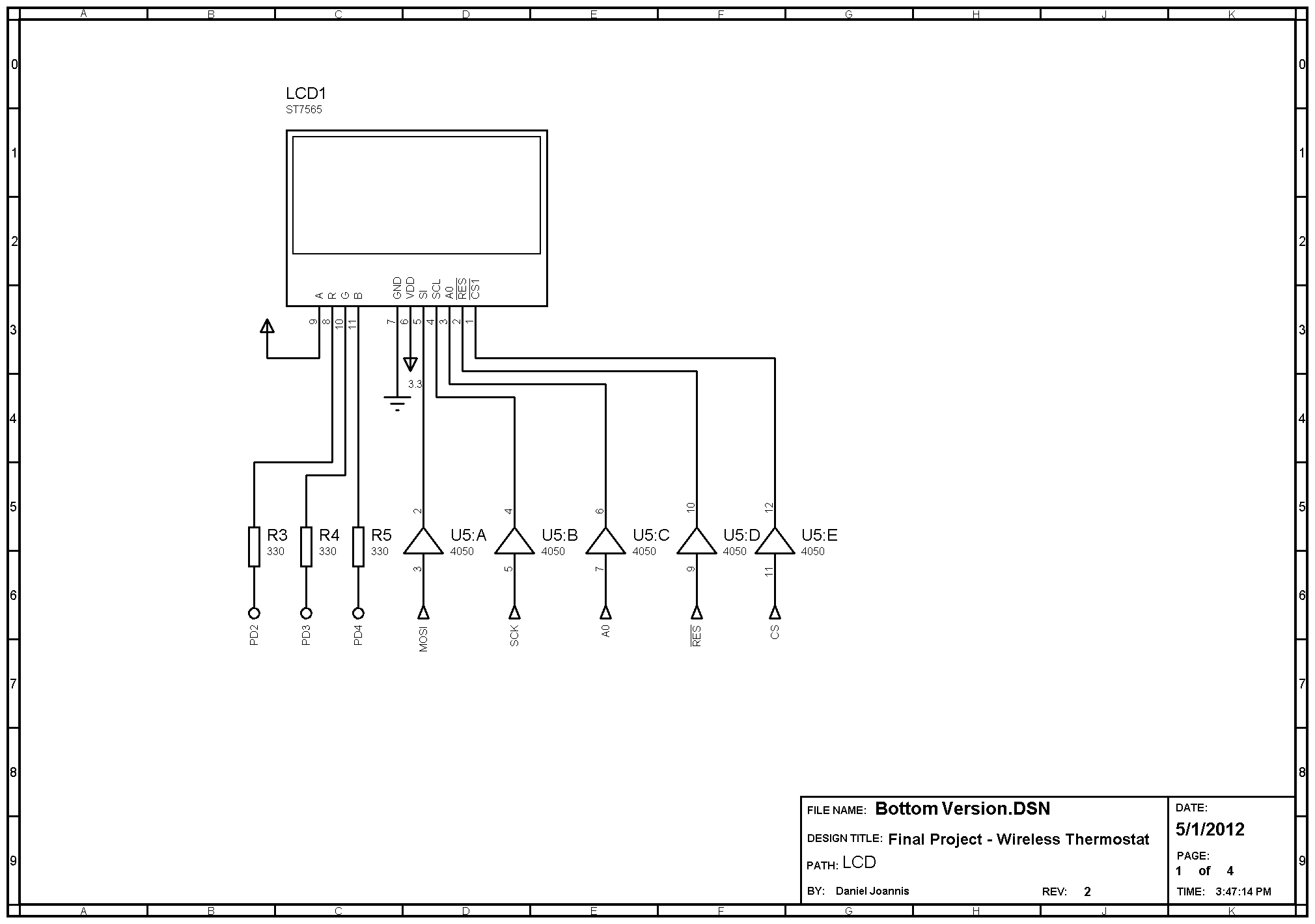
# C:\Users\Daniel\Desktop\psu.bmpCircuit Diagram

Figure 2



**Figure 3**

**Figure 4**

**Figure 5**

# PCB Layout

Figure 16 shows the PCB layout in ARES. There isn’t too much circuitry, so auto-routing was possible with minimal manual intervention. Keep-out zones were employed to prevent traces from running in the high-voltage area by the relay, and underneath the RTC battery.

One issue with this layout is that the relay, when triggering, creates interference that causes the XBee to transmit garbage data. This can lead to corrupted transmissions when sending data and the relay triggers. Though this is very unlikely to happen, as the device essentially pauses while transmitting data, it has happened in practice.

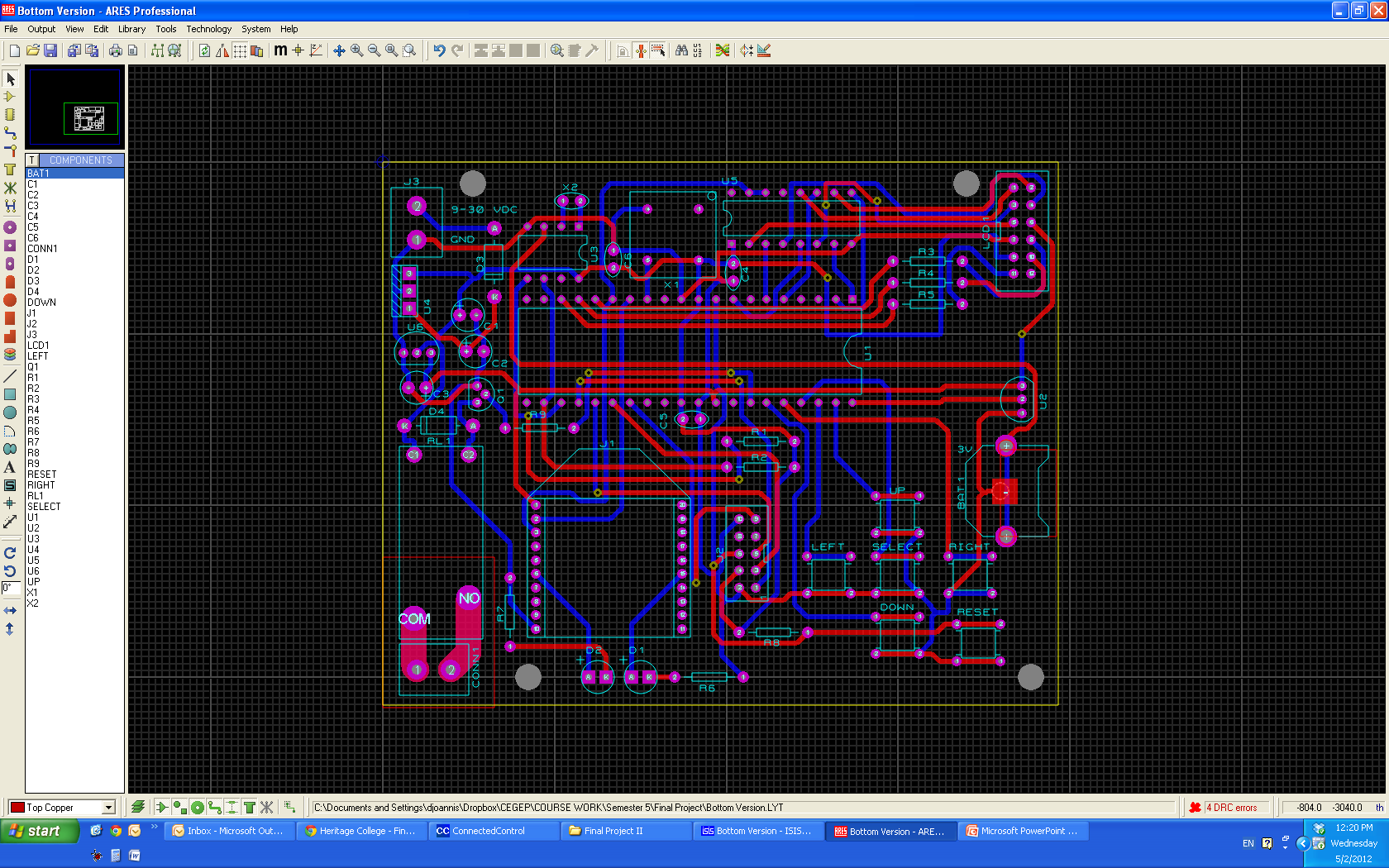


Figure 16

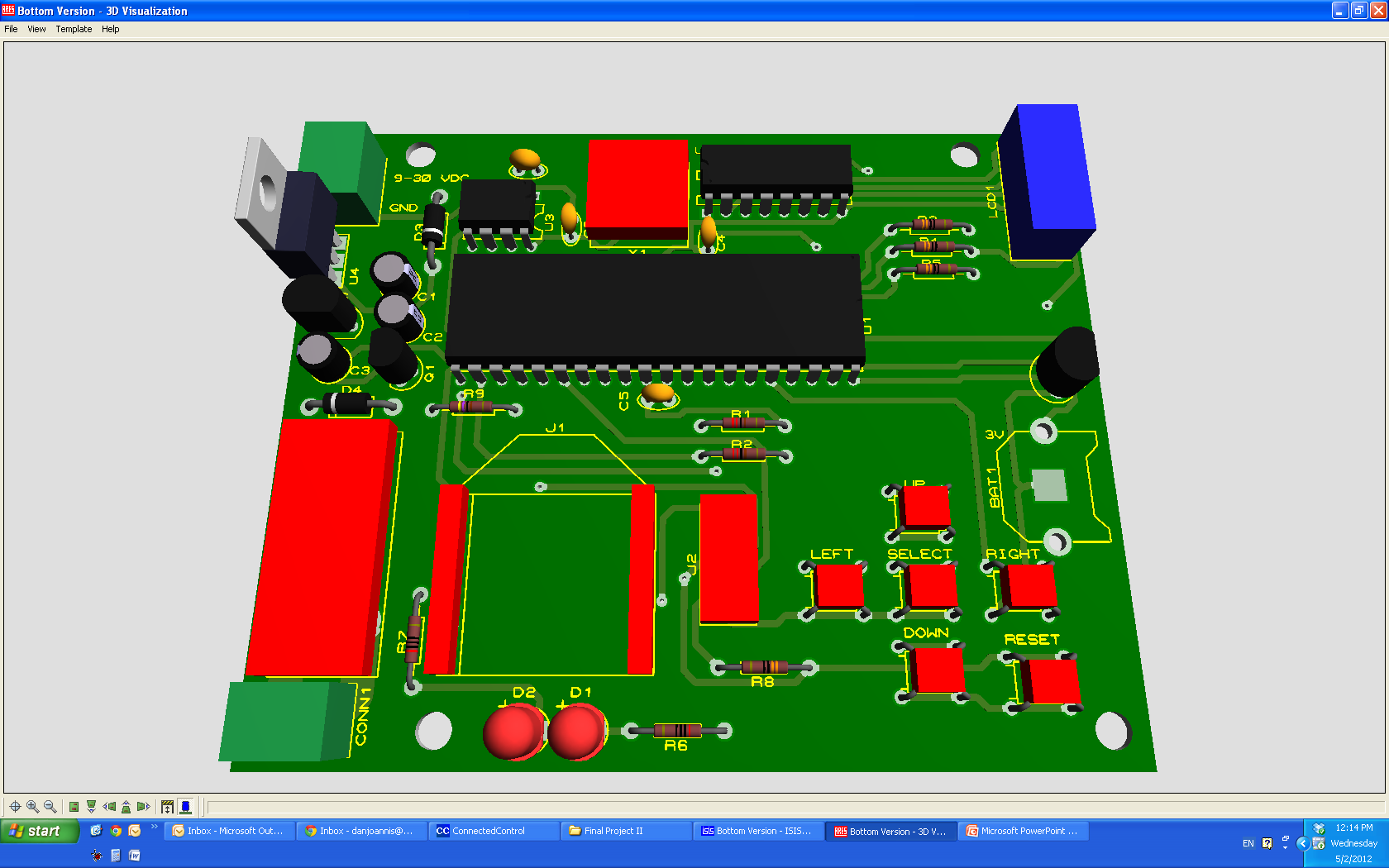


Figure 17

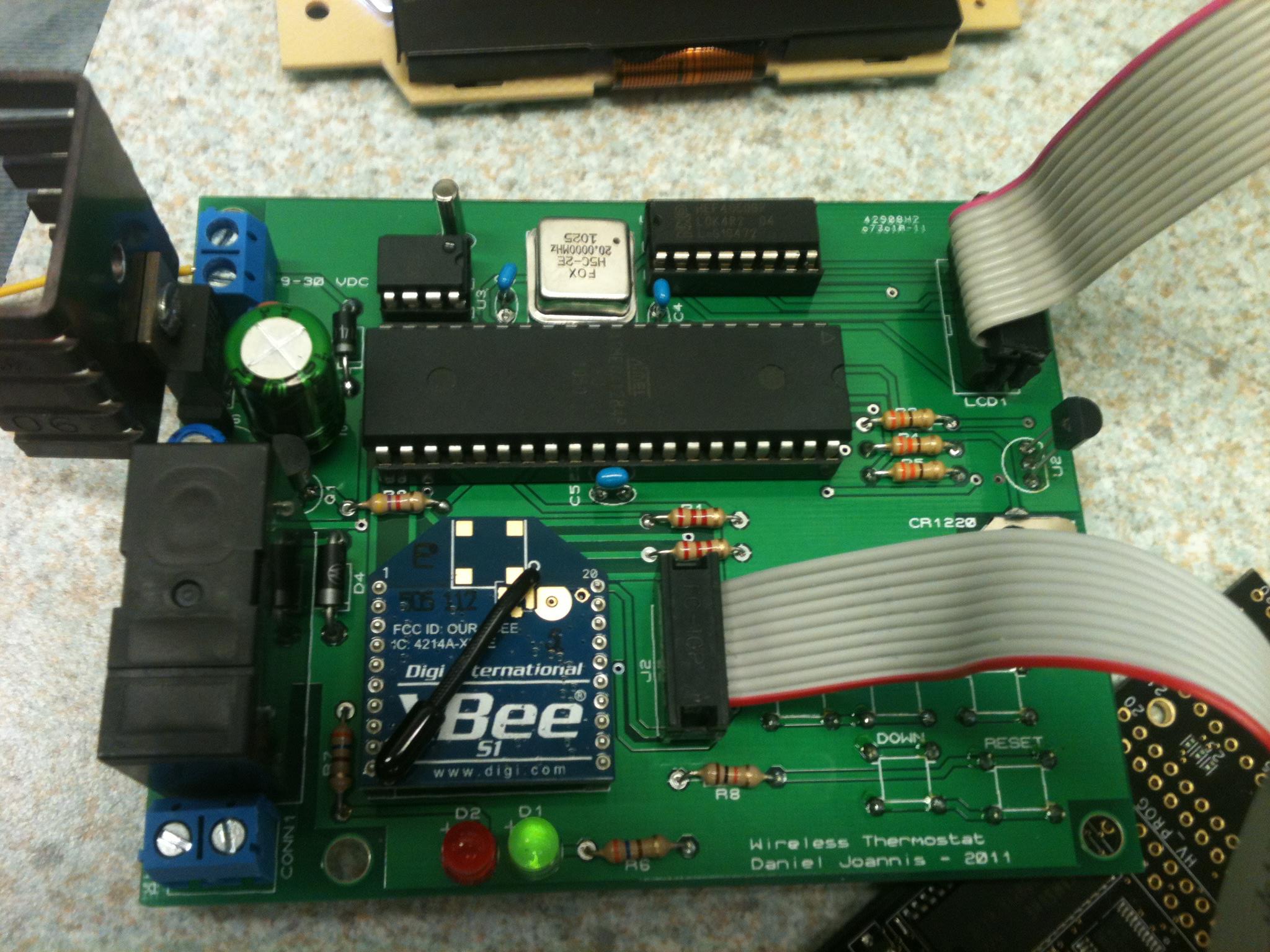


Figure 18

# Bill of Materials

Table 1

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Quantity | Reference | Value |
| Resistors | 2 | R1,R2 | 2.2k |
| Resistors | 3 | R3-R5 | 330 |
| Resistors | 2 | R6,R7 | 200 |
| Resistors | 1 | R8 | 10k |
| Resistors | 1 | R9 | 4.7k |
| Capacitors | 2 | C1,C3 | 1uF |
| Capacitors | 1 | C2 | 10uF |
| Capacitors | 2 | C4,C6 | 0.1uF |
| Capacitors | 1 | C5 | 0.47uF |
| Integrated Circuits | 1 | U1 | ATMEGA1284P |
| Integrated Circuits | 1 | U2 | TMP36 |
| Integrated Circuits | 1 | U3 | DS1307 |
| Integrated Circuits | 1 | U4 | 7805 |
| Integrated Circuits | 1 | U5 | 4050 |
| Integrated Circuits | 1 | U6 | MCP1700-33 |
| Transistors | 1 | Q1 | 2N2222 |
| Diodes | 2 | D1,D2 | LED |
| Diodes | 2 | D3,D4 | 1N4001 |
| Miscellaneous | 2 | AC IN,AC OUT | TERMINAL |
| Miscellaneous | 1 | BAT1 | 3V |
| Miscellaneous | 2 | J1,J2 | XBEE-SOCKET |
| Miscellaneous | 1 | J3 | JTAG-HEADER |
| Miscellaneous | 1 | LCD1 | ST7565 |
| Miscellaneous | 1 | RL1 | RLY-SPNO |
| Miscellaneous | 1 | X1 | FOX-CLOCK |
| Miscellaneous | 1 | X2 | CRYSTAL |

# Approximate Timeline

## Semester 1

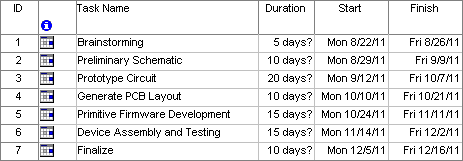


Figure 19

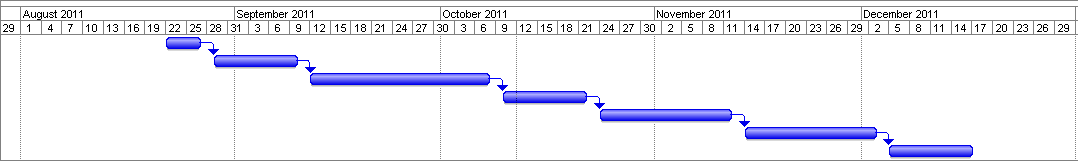


Figure 20

## Semester 2

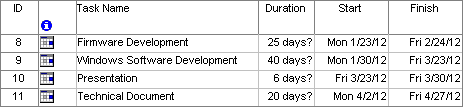


Figure 21

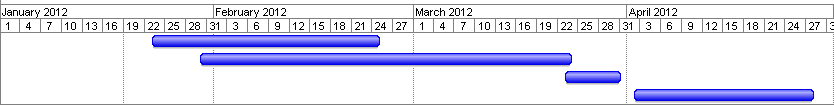


Figure 22

Unfortunately, for the second semester my timeline was too ambitious. I was unable to meet my deadlines, as my project was completed April 30th, 2012. In retrospect, I may have simplified the breakdown of tasks too much as well, and would have been better to have included a task of packaging the project into an enclosure, and finalizing appearance and function.

# Conclusion

The process of creating a product, from conception to completion, has been one of problem solving and creativity. A product on paper, and one on PCB, are two completely different things. It is necessary to come up with solutions to problems that didn’t exist in previous phases, long after they are complete. There were periods of time where motivation was severely lacking, when some of the tasks seemed too big or complicated. However, like every puzzle, time would find the answers. In the end, I am very satisfied with my project, and it has exceeded any expectations I had before.

My proposal outlined two major challenges: creating a very good user interface, and creating a communications protocol between the hardware and the Windows software. I feel I managed to accomplish the first, as my on-device user interface is fully featured, allowing the user the same level of control as the Windows GUI, though a bit less friendly to use. This is a limitation of a small, low resolution screen and having simple buttons as inputs.

For communications, I created a simple that used characters as instructions, with different characters serving to request different communication. This works very well due to the simplicity, however, there is one thing that could be improved upon. The hardware will poll in order to receive data from the GUI once the instruction is received by the interrupt. At this point, interrupts are disabled and the device polls for each byte coming from the computer. In the event a byte is damaged or missing, or communications are cut in the middle, the device will freeze waiting for this data. It must then either be reset, or receive enough data to exit the loops. Ideally, a timeout should exist to break the device out of this kind of situation.

Rather than using XBees, I would like to use a single WiFly, and connect the thermostat to a WiFi network and create a web interface. A web UI is platform independent, and can even be accessed from smartphones, making it more practical. Unfortunately, because the WiFly was sold out at the time of design, I quickly dismissed it.

Overall, this was a very successful project. I learnt ways to save time in the future, made mistakes from which I have learnt. An important part of this project for me was to create something useful. I didn’t want to end up with a project that just sat on a shelf once completed. ConnectedControl has met this objective, as being both functional and useful.

References

[1] <http://www.nextag.com/wireless-thermostat/compare-html>

[2] <http://www.ecobee.com/>

[3] <http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf>

[4] <http://pewa.panasonic.com/assets/pcsd/catalog/ale-catalog.pdf>

[5] <http://www.atmel.com/dyn/resources/prod_documents/doc8059.pdf>

[6] <http://www.picaxe.com/docs/xbe001.pdf>

[7] <http://www.national.com/ds/LM/LM340.pdf>

[8] <http://www.diodes.com/datasheets/ds28002.pdf>

[9] <http://www.datasheetcatalog.org/datasheet2/c/0gyw0u7w6jro8xifeqpk6gjti67y.pdf>

[10] <http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf>

[11] <http://www.brookdale.com/Fox/h5c-2e.pdf>

[12] <http://www.stanford.edu/class/ee133/datasheets/2n2222.pdf>

[13] <http://datasheets.maxim-ic.com/en/ds/DS1307.pdf>

[14] <http://www.ladyada.net/learn/lcd/TG12864H3-03A%20EN_V1.1.pdf>

[15] <http://www.nxp.com/documents/data_sheet/HEF4050B.pdf>

[16] <http://www.adafruit.com/datasheets/TG12864H3-05A%20EN_V1.0.pdf>