

# Building an LED Binary Clock Z8 Board

Darby Thompson

CS339

4/25/2006

## Design Overview

The goal of this project was to build a custom Z8 board with the functionality of an LED binary clock and alarm. The clock is designed around the Zilog Z8F6401 40-pin DIP microprocessor on a 2.5"x3.8" circuit board. The board is designed to provide the functionality of a simple alarm clock, with the following modes: display the time, set the time, set the alarm, alarm on and alarm off. The DS1307 real time clock module from SparkFun was interfaced over I2C with the Z8 to keep the time. Software to implement the display and alarm capabilities is written in C in the Zilog Developer Studio II.

## Hardware Components

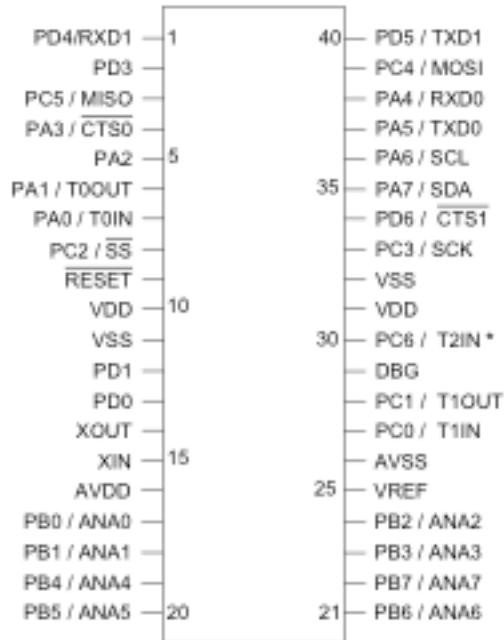
The primary components of this project are the Z8F6401 microprocessor and the real time clock module from SparkFun. The project was restricted to a 2.5"x3.8" PCB ordered from ExpressPCB. Other components required included: 21 LEDs, 2 push-button switches, 2 toggle switches, 1 speaker, power connector, 5V power adapter, voltage regulator, crystal, various resistors, various capacitors, debug interface pins (2x3) and a 40-pin DIP socket (for the microprocessor). The following is a more detailed description of each component used.

### Z8F6401 Microprocessor

*Relevant Features:*

- eZ8 CPU, 20 MHz operation
- 64KB Flash memory with in-circuit programming capability
- 4KB register RAM
- Serial communication protocols
  - Serial Peripheral Interface
  - I2C
- 24 interrupts with programmable priority
- 3 16-bit timers with capture, compare, and PWM capability
- Single-pin On-Chip Debugger
- Watch-Dog Timer (WDT) with internal RC oscillator
- Up to 31 I/O pins
- Voltage Brown-out Protection (VBO)
- Power-On Reset (POR)
- 3.0-3.6V operating voltage with 5V-tolerant inputs

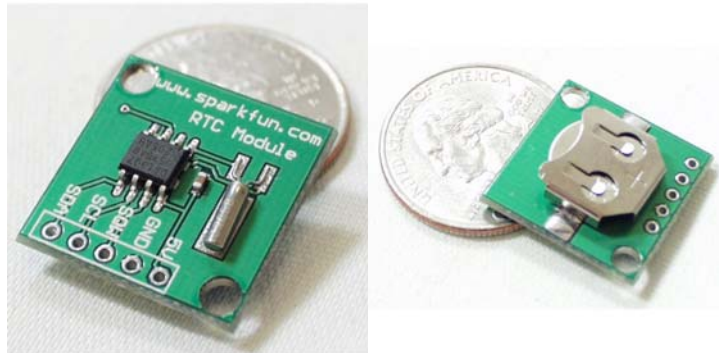
*Pin Assignment:*



A 40-pin DIP socket was used for the microprocessor, to allow for easy debugging and removal.

### Real Time Clock Module

This is a custom designed module for the DS1307 Real Time Clock. The module is shipped pre-programmed with the current MST time, and includes a Lithium coin cell battery which will run the module for a minimum of 9 years without external 5V power. The DS1307 is accessed via the I2C protocol.

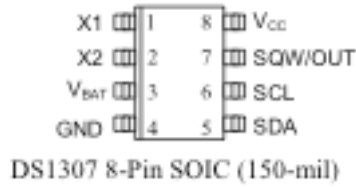


#### Features:

- Two wire I2C interface
- Hour : Minutes : Seconds AM/PM
- Day Month, Date – Year
- Leap year compensation
- Accurate calendar up to year 2100
- Battery backup included

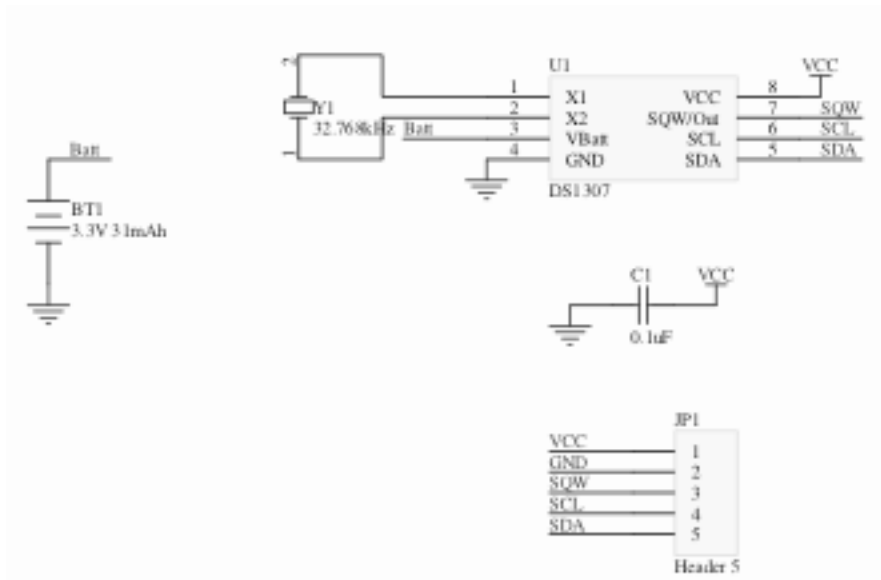
- 1Hz output pin
- 56 Bytes of Non-volatile memory available to user

*DS1307 Pin Assignment and Description:*



- VCC - Primary Power Supply
- X1, X2 - 32.768kHz Crystal Connection
- VBAT - +3V Battery Input
- GND - Ground
- SDA - Serial Data
- SCL - Serial Clock
- SQW/OUT - Square Wave/Output Driver

*Module Schematic:*



**Light Emitting Diodes**

20 LEDs were required for the clock interface, and 1 to indicate whether or not the alarm is set. The LEDs used are T1, 3mm diameter, diffused standard-output LEDs, operating at 10mA with a wavelength of 565nm.

**Push-Button Switches**

2 SPST on-(off) push buttons were used to increment the minutes and hours when in ‘timeset’ or ‘alarmset’ mode. The switches feature 2 pins, with a 1/4” mounting hole and a contact rating of 3A @ 125VAC.

## Toggle Switches

2 miniature toggle switches were used to toggle the alarm on and off, and timeset/alarmset on and off. They are SPST on-off with 2 pins and a contact rating of 6A @ 125VAC and 3A @ 250VAC.

## Voltage Regulator

The Z8 requires 3V and the real time clock requires 5V, so I used a 3.3V low dropout voltage regulator (LD1086). The pin assignment and typical application circuit are shown below:



2 10  $\mu$ F capacitors were also ordered to implement the above circuit.

## Power Supply and Connector

A 5V 1A AC-DC regulated, center-positive power supply was used, along with a standard power connector.

## Speaker

A 1.5" speaker (from Lab 2) was used to implement the alarm. This was not fixed to the board due to size constraints.

## Crystal

A 18.432Mhz Crystal was used, along with 2 15pF ceramic disk capacitors and a 1Mohm resistor as input to the XOUT and XIN pins on the microprocessor.

## Resistors

In addition to the previous resistors mentioned, 10Mohm resistors were required as pull-ups for the switches, RESET pin and DEBUG pin. 330Ohm resistors were required to restrict the current flow to the LEDs in the display and, since there were so many, 3 16-pin Dual-In-Line resistor networks were used. To connect the SCL and SDA pins on the clock, 4.7Kohm resistors were used.

## Hardware Interfacing

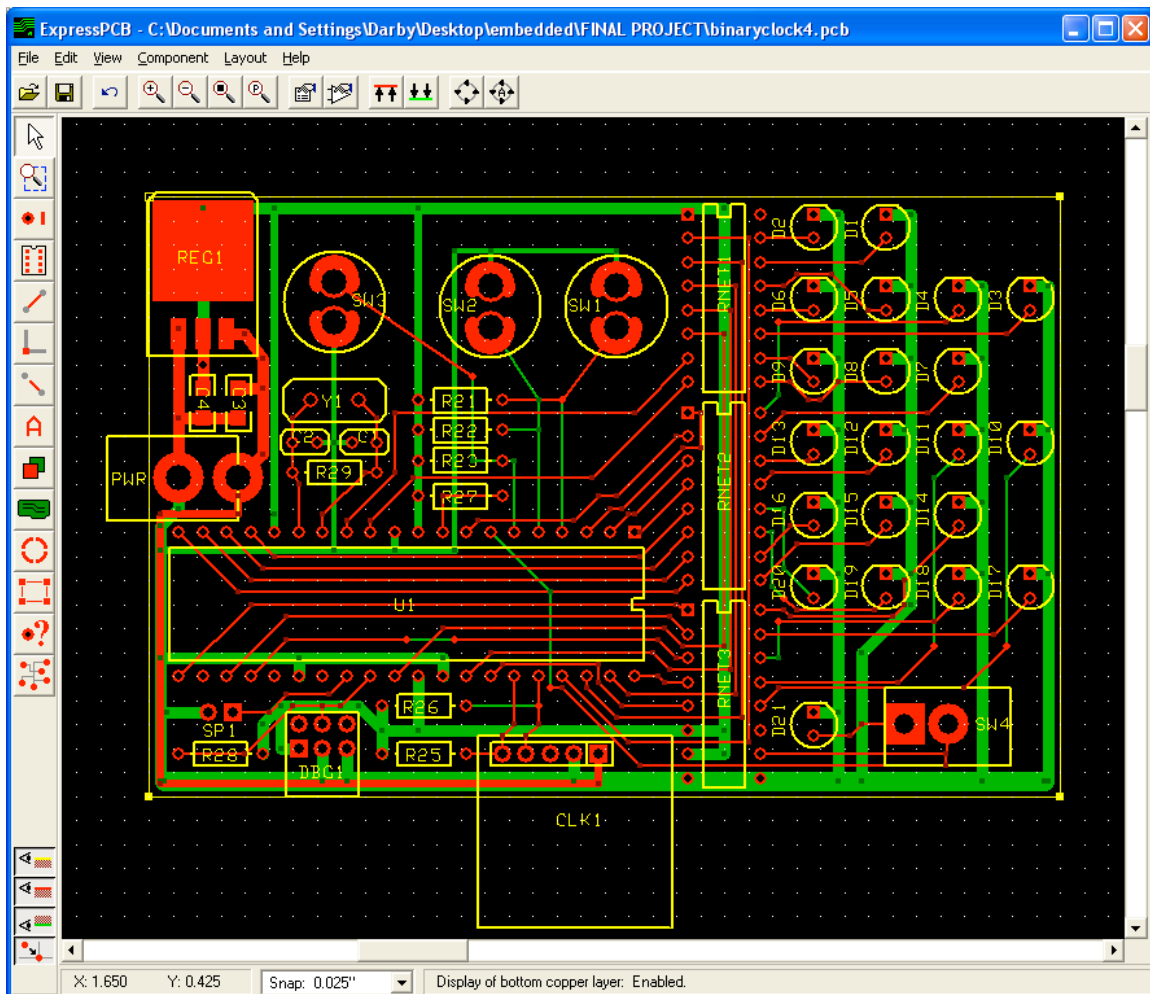
### Schematic

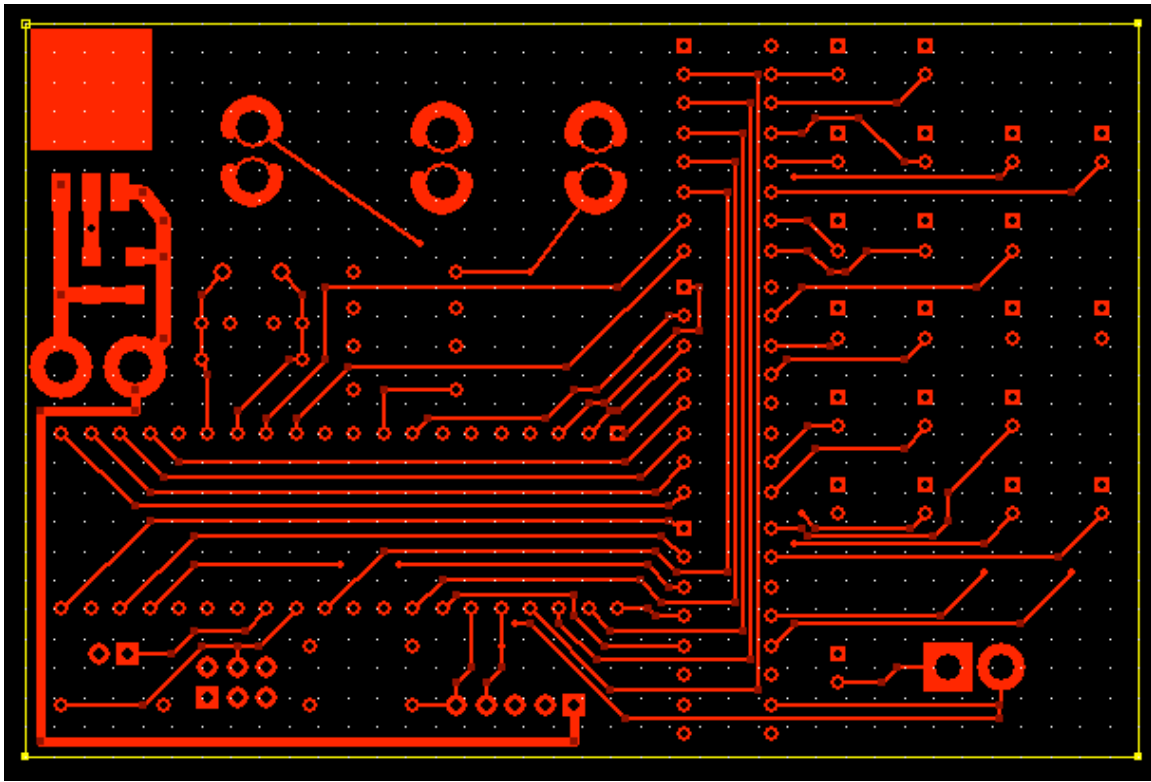
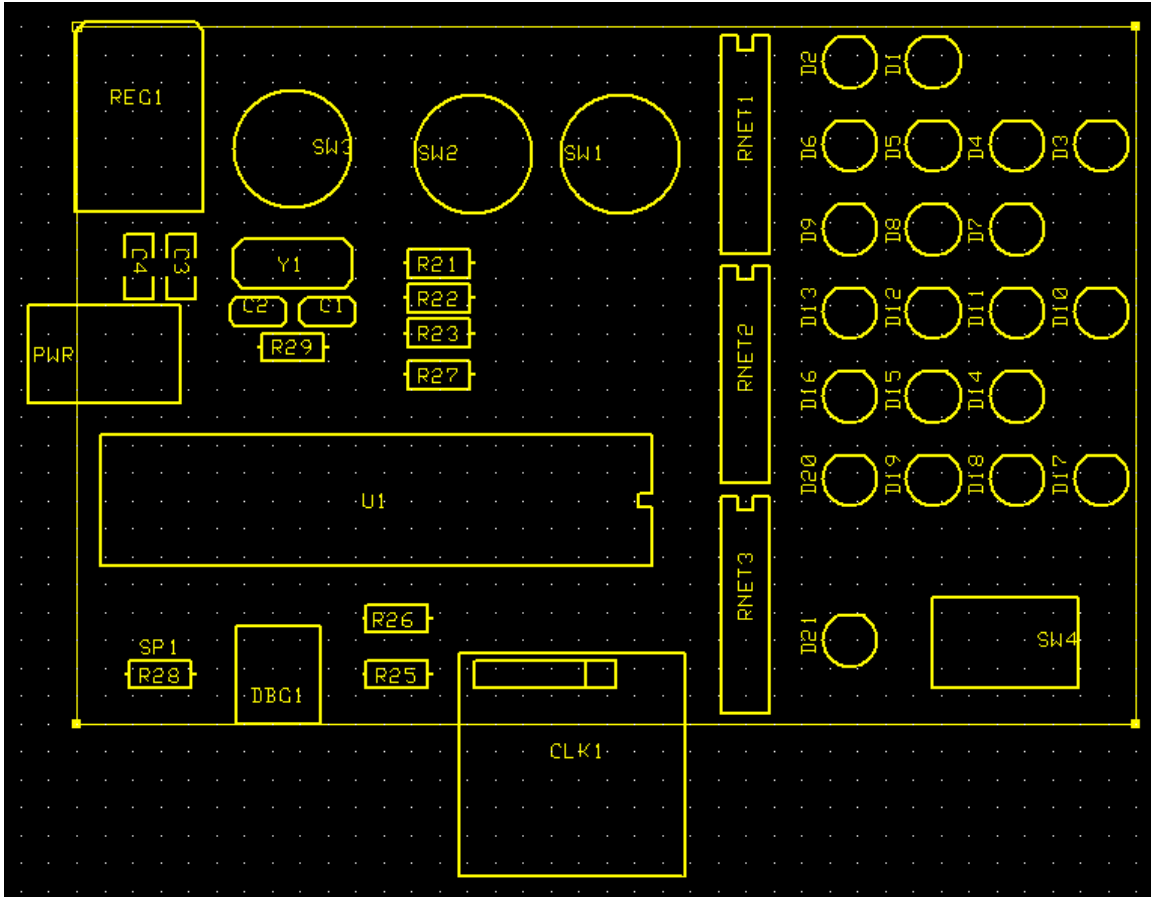
Below is the schematic developed for the project using ExpressSCH. The switches were connected to pins A0-A3 so that they could be configured to send interrupts. Each 'group' of LEDs were connected to sequential pins in the same port (if possible) so that setting and incrementing the clock display would be done efficiently in software. The speaker was connected to PC1/T1OUT so that a tone for the alarm could be played using

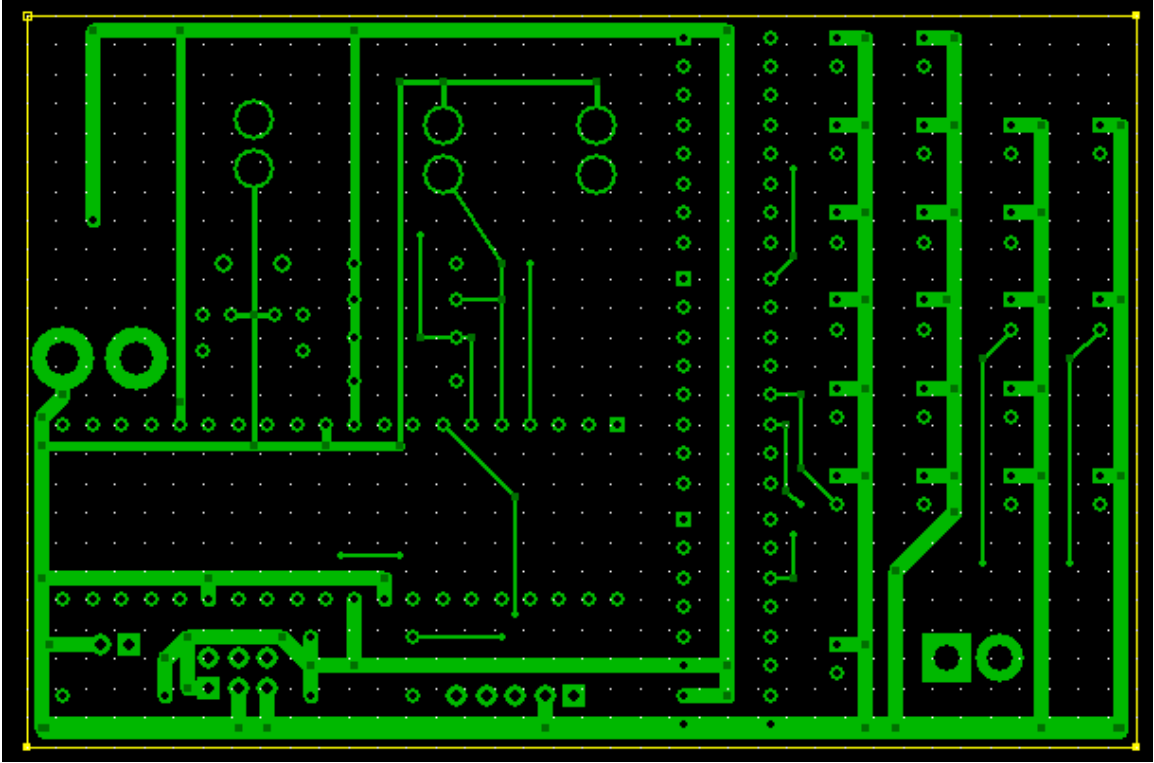


## PCB Design

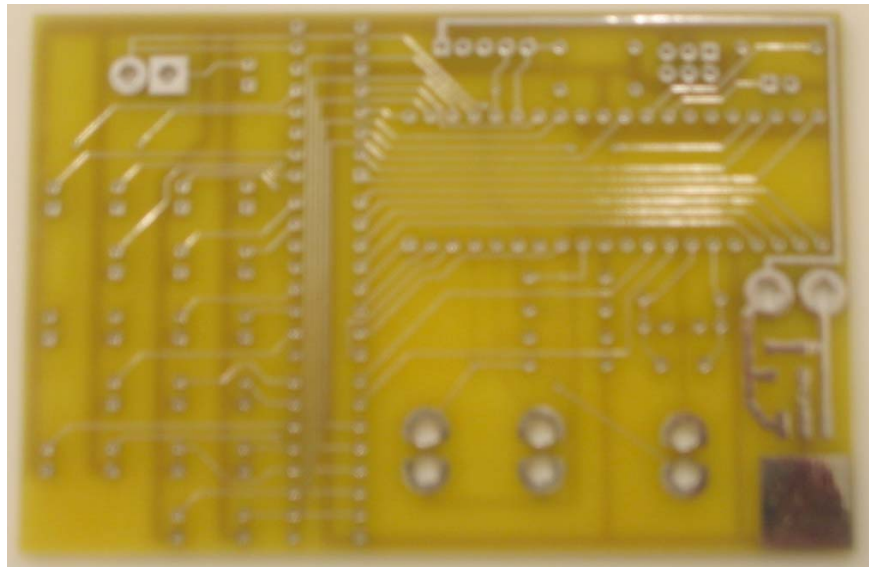
Below is the final PCB design created using ExpressPCB. The LED display had a fixed orientation, apart from that there were no specific constraints on the placement of components. The real time clock module pins were placed such that the clock could hang over the edge of the board, and the speaker was also left unattached to the board to save space. Major GND and VCC tracks were laid on the underside of the board where possible. In the process of creating the PCB design, many different arrangements and orientations of the components were tested. The following design reflects the most efficient arrangement where there is minimal switching between board layers, maximum spacing between tracks, and minimal distances between connections.



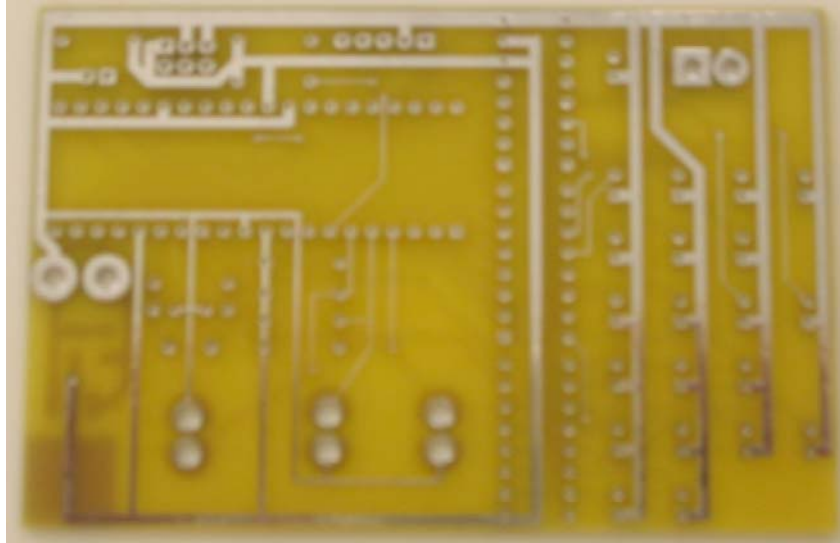




**Putting It All Together**

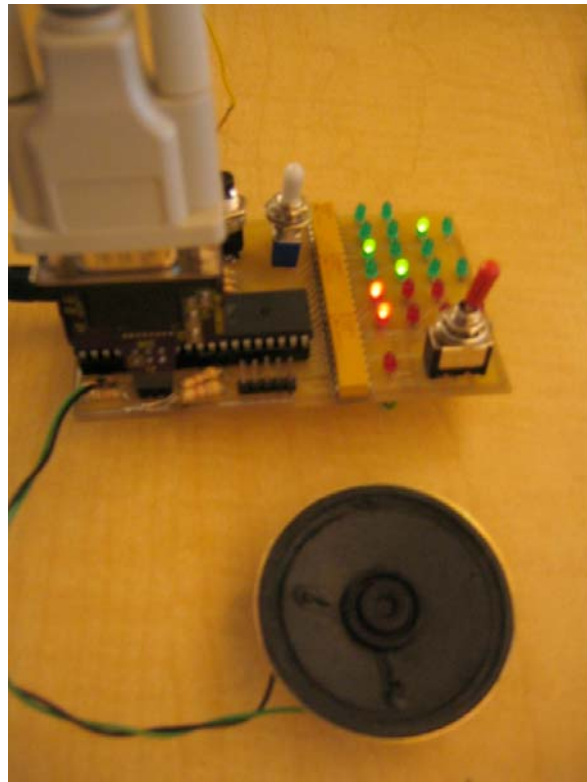


[Top]

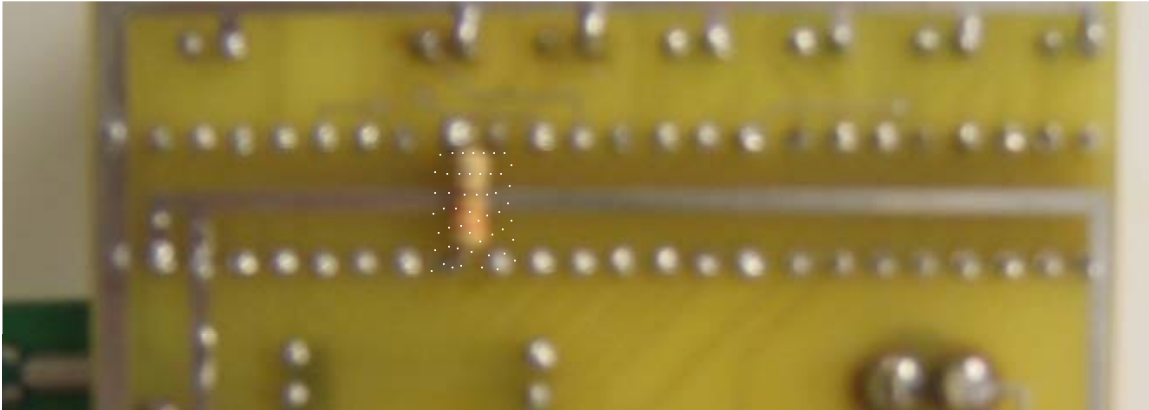


[Bottom]

The first thing soldered to the board was the regulator and power connector. These were added and tested successfully (the regular correctly outputted 3.3V). Next, I added the processor socket, processor, crystal and its capacitors and resistor. I also added the pull-up resistor for the RESET and DEBUG pins. I added the pins for the target interface module, and tried to connect to the Z8 through the Zilog Developer Studio II environment. At this point it became apparent that the TIM pins were flipped horizontally; however, this was a quick fix (rotating the connector board). Once this was discovered, the Z8 could successfully be programmed.

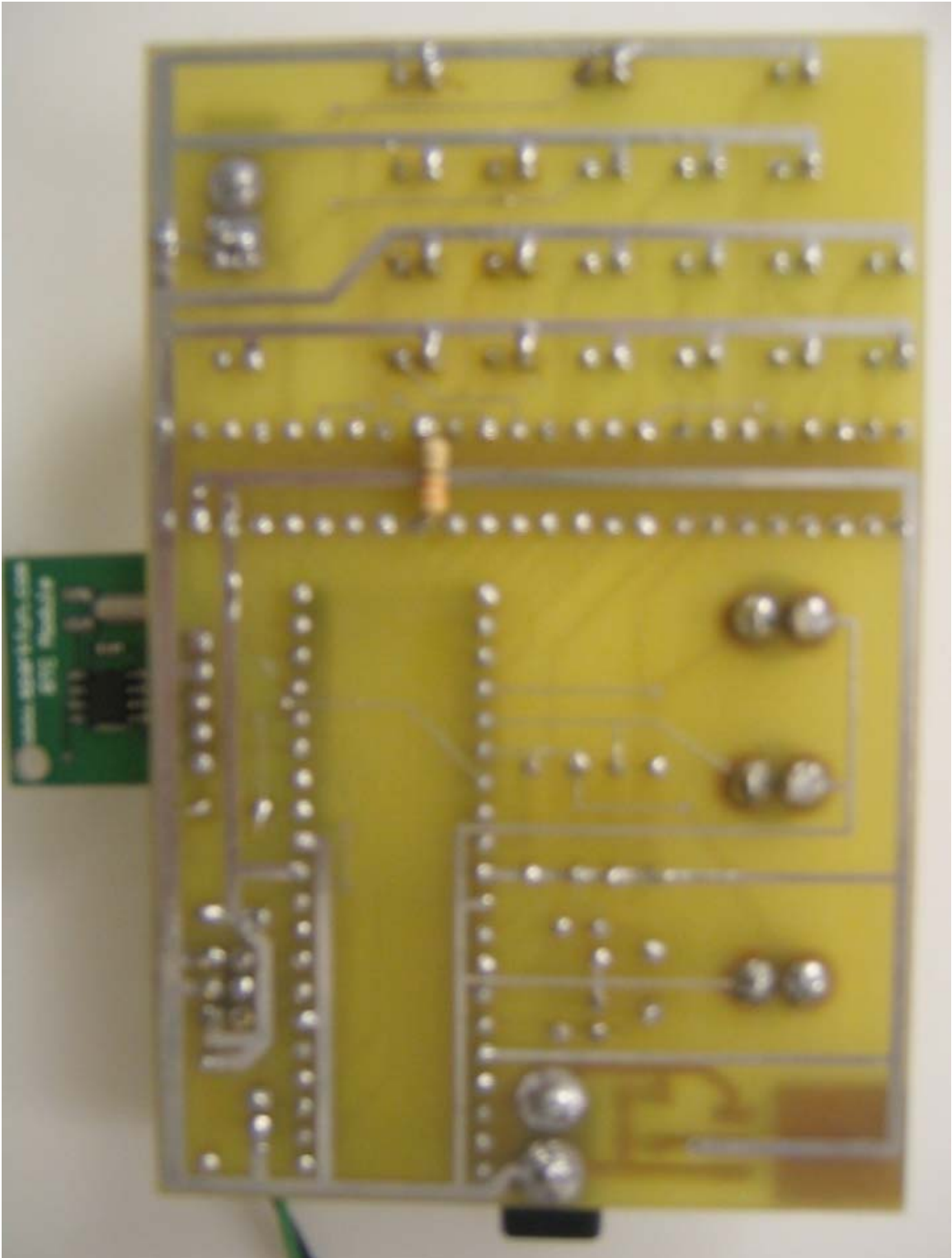


Next the resistor networks and LEDs were added. The plastic casing of the resistor networks turned out to be approximately 1mm longer than the standard PCB footprint that I had used (lesson learned!). To resolve the issue, I filed down the plastic coating on the ends of the networks so that they could fit. Once added, I programmed the z8 to turn on all GPIO ports to test the LEDs. Testing was successful for all but one LED who's resistor appeared to be damaged (possibly by the filing down or the end of the resistor). After adding a single 330Ohm resistor across the underside of the corrupt pin on the resistor network, testing was successful.



Next, I added the switches and their related pull-up resistors. Testing the switches turned out to be much more problematic than the previous components. The push-button switches had to be replaced since they provided unreliable results, but the most obvious problem occurred with SW4, the alarm toggle switch. This switch used a 330Ohm pull-up resistor rather than a 10Kohm resistor, so that it could drive an LED to indicate whether it was on or off. Unfortunately, this meant that the switch only pulled the voltage down to 1.75V when turned on, which is not low enough to register logic 0 by the Z8. The fix was to increase the resistor to 10Kohm and to bypass the LED. To use the LED, it had to be connected to one of the two free GPIO ports (PC0) on the Z8 along with a 330Ohm resistor. The switch de-bouncing issues are addressed in the software section of this report.

Lastly, the real time clock, resistors, and speaker were added. During the hardware testing it became apparent that the resistors ordered for the I2C interface (SDA and SCL lines) were 4.7Ohms instead of 4.7Kohms, and the problem was resolved quickly by purchasing the correct resistors. Another change to the original schematic was that the resistors were pulled up to the 5V power line instead of 3V.



## Software

The Z8 microprocessor was programmed using the Zilog Developer Studio II through the serial debug target interface that came with the lab Z8 evaluation board. Simple

programs were originally developed to test the LEDs and switches. To test the LEDs, the appropriate GPIO pins were configured as data output, and outputs set to logic 1 to turn on all clock LEDs for testing.

Since the switches were connected to Port A, I configured them as interrupt pins. The two toggle switches had no need for de-bouncing since they were SPST on-off: I only had to check the appropriate Port A input pin to see if was switched on or off, which was very helpful. The two push-button switches were momentary SPST on-(off) switches. This required some de-bouncing (a counter was used), however it can still be tricky to get the precise correct number of button presses from them.

The speaker was connected to the output pin of Timer 1 such that the timer could be configured to output a PWM signal to create an audible tone from the speaker for an alarm. This was successfully implemented and tested.

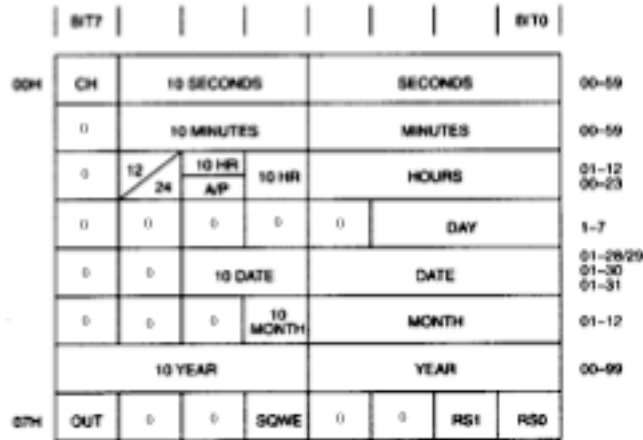
Once these components were tested, a full program was developed to provide the full functionalities of a binary alarm clock by using one of the unused timers on the Z8. Timer 0 was set to interrupt every second, and the second, minute and hour values were incremented and the LED display refreshed. Modes were setup such that the user can set the time by toggling switch 1 on (with the alarm switch (4) off), then using switches 2 and 3 to increment the hour/minute values. Another mode was setup so that the user could set the alarm when both the alarm toggle and timeset/alarmset toggle were on.

Now that I had a working binary clock, I needed to interface it with the SparkFun real time clock module so that the time could be stored and kept correctly (rather than being reset after a power loss). The DS1307 is a low-power, full binary coded decimal clock/calendar which comes with 56 bytes of NV SRAM and interfaces with the Z8 using the I2C protocol. The clock/calendar provides seconds, minutes, hours, day, date, month and year information (although only the seconds, minutes and hours were used for this project). It also operates in either a 24 hour or 12 hour format. The address map for the registers of the clock is shown below:

|     |         |
|-----|---------|
| 00H | SECONDS |
|     | MINUTES |
|     | HOURS   |
|     | DAY     |
|     | DATE    |
|     | MONTH   |
|     | YEAR    |
| 07H | CONTROL |
| 08H | RAM     |
|     | 56 x 8  |
| 3FH |         |

The clock is accessed by reading the appropriate registers (0x00 = seconds, 0x01 = minutes, 0x02 = hours). Similarly, the time can be set by writing to the same registers. The clock also features a clock halt (CH) bit which must be set to 0 to enable the clock,

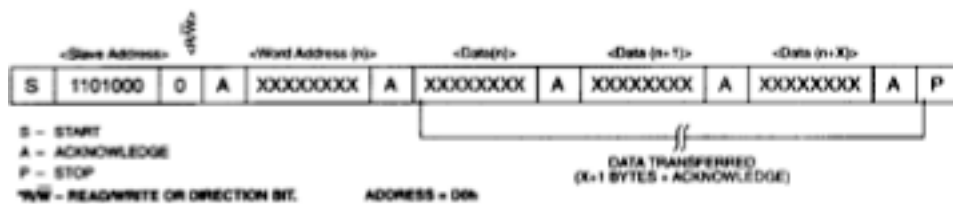
and a 12 hour/24 hour bit which was set low for a 24 hour clock. When a START condition is encountered, the current time is copied to a second set of registers to eliminate the need to re-read the registers in case of an update of the main registers during a read. The format of these 'timekeeper' registers is shown below:



As seen in the above figure, 3 sequential byte reads are required to access the seconds, minutes and hours from the clock.

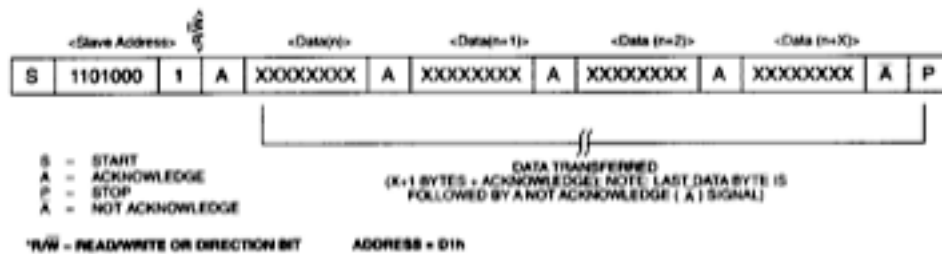
The sequence of events needed to write to the clock are as follows:

- Send a START condition
- Send the DS1307 write address (0xD0)
- The DS1307 responds with an ACK
- Send a register address
- The DS1307 responds with an ACK
- Send data (with the DS1307 responding with an ACK after each byte)
- Generate a STOP condition



To read the time from the clock:

- Set the register pointer by performing a write (as above, without sending data)
- Send a START condition
- Send the DS1307 read address (0xD1)
- The DS1307 responds with an ACK
- The DS1307 begins to transmit data starting with the register address pointed to by the register pointer.
- After reading the appropriate bytes, send a NACK to end the read
- Generate a STOP condition



The functions used to implement these instructions were based on the I2C drivers from the Z8F640 Family Drivers Demo. Many problems were initially encountered when trying to communicate with the clock. After countless debugging of the software, it was discovered that the clock chip was malfunctioning and had to be replaced. Once replaced, the software successfully communicated with the chip. Currently the clock triggers a timer interrupt every second, during which the Z8 reads the time from the clock and refreshes the display. The following are relevant I2C code excerpts from the software:

### Reading the time

```

I2C_write_byte(WriteAddress); // Write ADDRESS
I2C_start(); // I2C Start
I2C_Transmit_Data_Empty();
I2C_write_byte(0x00); // Set Register Pointer
I2C_Transmit_Data_Empty();
I2C_stop(); // I2C Stop

I2C_write_byte(ReadAddress); // Read ADDRESS
I2C_start(); // I2C Start
I2C_Transmit_Data_Empty();
I2C_Receive_Data_Full (); // wait for receive buffer full
data = I2C_read_byte (); // read messed up byte
secH = (data>>4) & 0x0F;
secL = (data & 0x0F);
I2C_Receive_Data_Full (); // wait for receive buffer full
data = I2C_read_byte (); // read seconds
secH = (data>>4) & 0x0F;
secL = (data & 0x0F);
I2C_Receive_Data_Full (); // wait for receive buffer full
data = I2C_read_byte (); // read minutes
minH = ((data>>4) & 0x0F);
minL = (data & 0x0F);
I2C_Receive_Data_Full (); // wait for receive buffer full
data = I2C_read_byte (); // read hours
hourH = ((data>>4) & 0x03);
hourL = (data & 0x0F);
I2C_Send_NACK(); // Send NACK to indicate last read

```

```
I2C_stop(); // I2C Stop
```

*Writing to the minutes/hours register (register is 0x01 for minutes, 0x02 for hours)*

```
I2C_write_byte(WriteAddress); // Write ADDRESS
I2C_start(); // I2C Start
I2C_Transmit_Data_Empty();
I2C_write_byte(0x01); // select REGISTER
I2C_Transmit_Data_Empty();
data=((minH<<4) & 0xF0) | minL;
I2C_write_byte(data); // Write DATA
I2C_Transmit_Data_Empty();
I2C_stop(); // I2C Stop
```

The complete source code for the project can be found in APPENDIX 2.

## Operation

### Turn on the Alarm

Toggle switch 4 to 'on'. The red LED below the switch should light when the alarm is on.

### Turn the Alarm off / Deactivate the Alarm Tone

Toggle switch 4 to 'off'. The red LED below the switch should be turned off.

### Set the Alarm

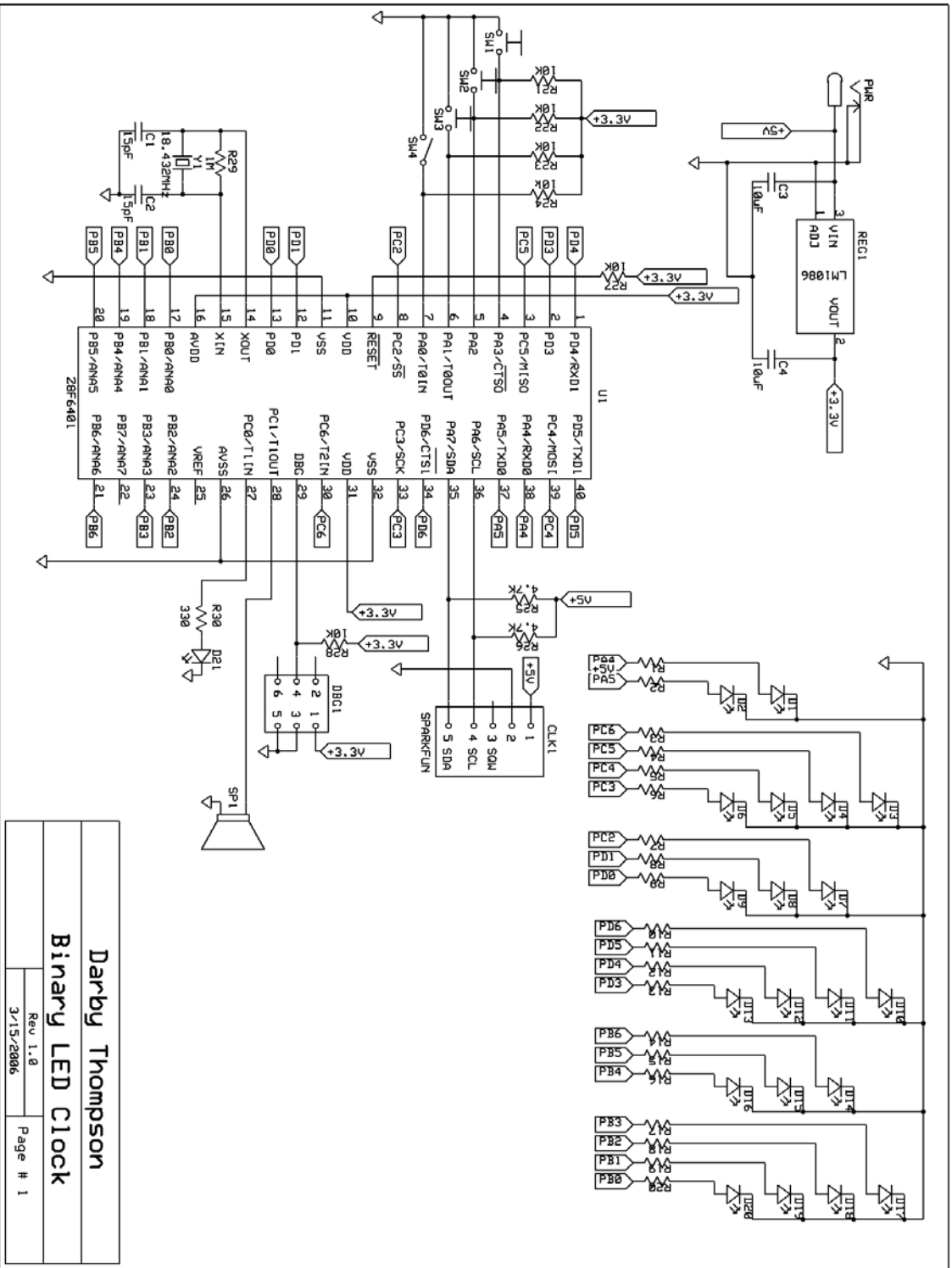
Toggle switch 4 to 'on'. Toggle switch 1 to 'on'. The display will now show the current settings for the alarm. To increment the hour press switch 2. To increment the minutes press switch 3. To return to the normal clock display mode, toggle switch 4 to 'off'.

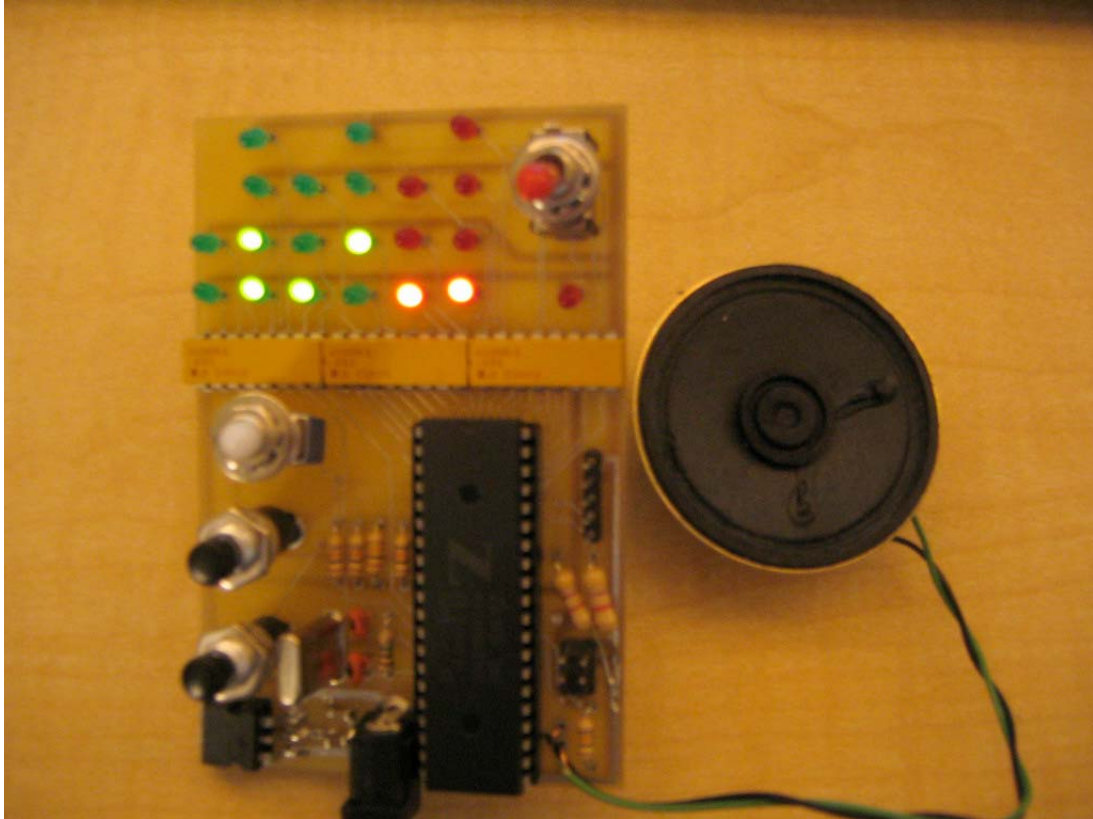
### Set the Time

Make sure that Toggle switch 4 is 'off'. Toggle switch 1 to 'on'. The display will now show the current settings for the time. To increment the hour press switch 2. To increment the minutes press switch 3. To return to the normal clock display mode, toggle switch 4 to 'off'.

## Final Results and Conclusions

The final product is a fully functional binary LED alarm clock. The modified schematic (with the corrections stated in the previous sections) is shown below:





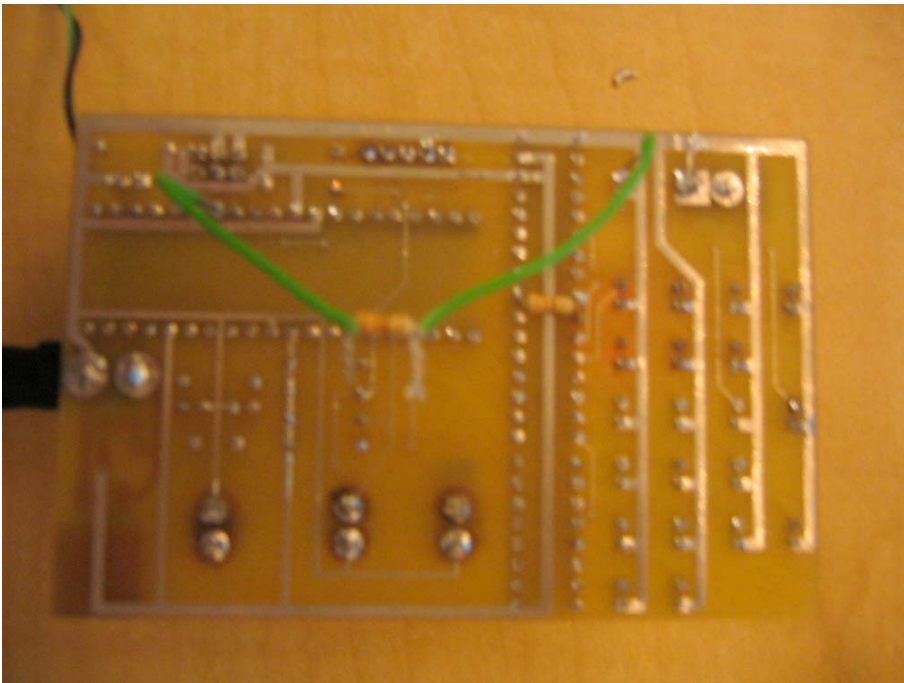
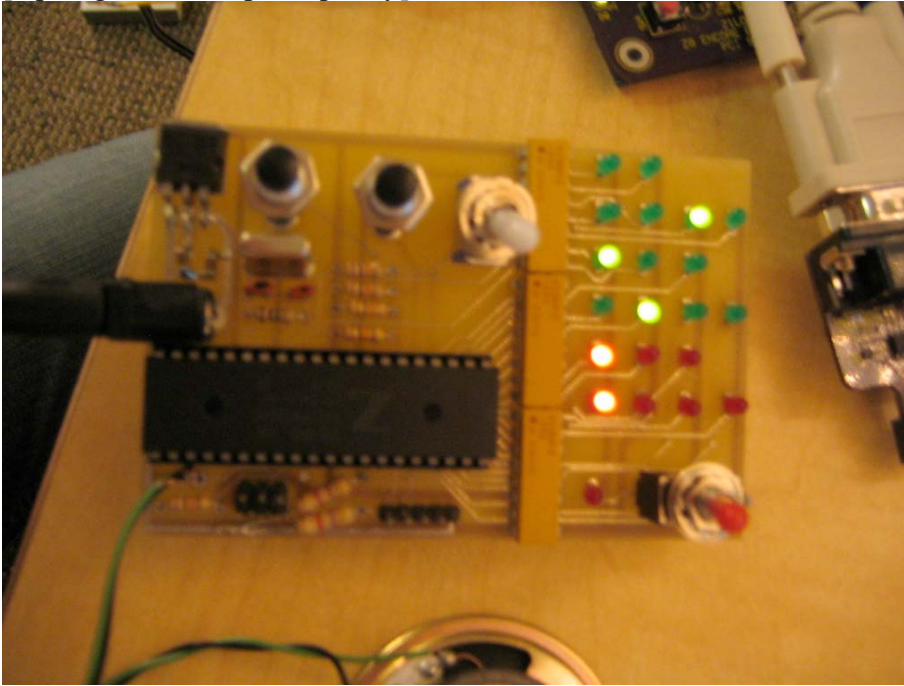
More pictures are available in Appendix 1.

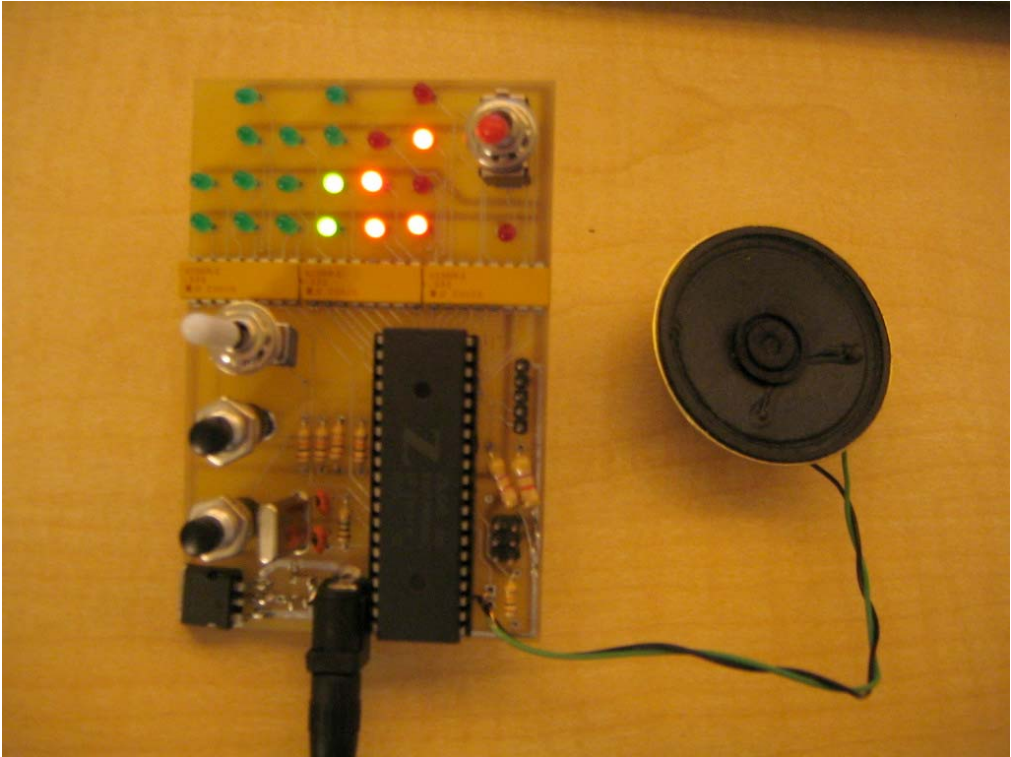
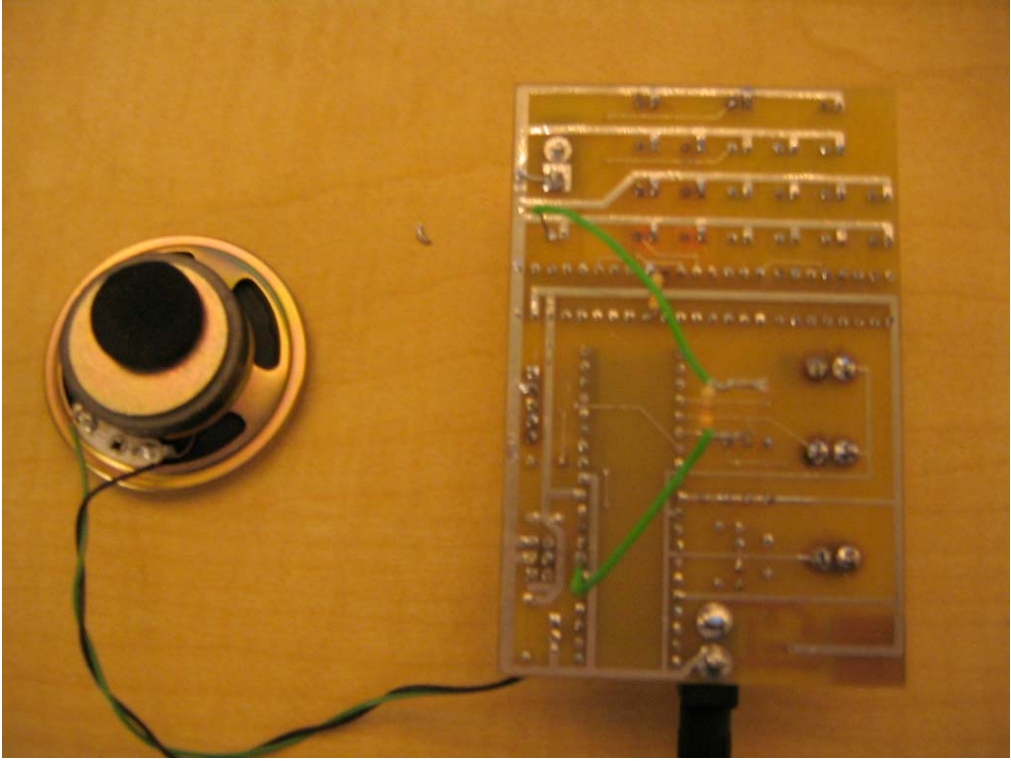
In retrospect, changes that I would make if I were to repeat the project include:

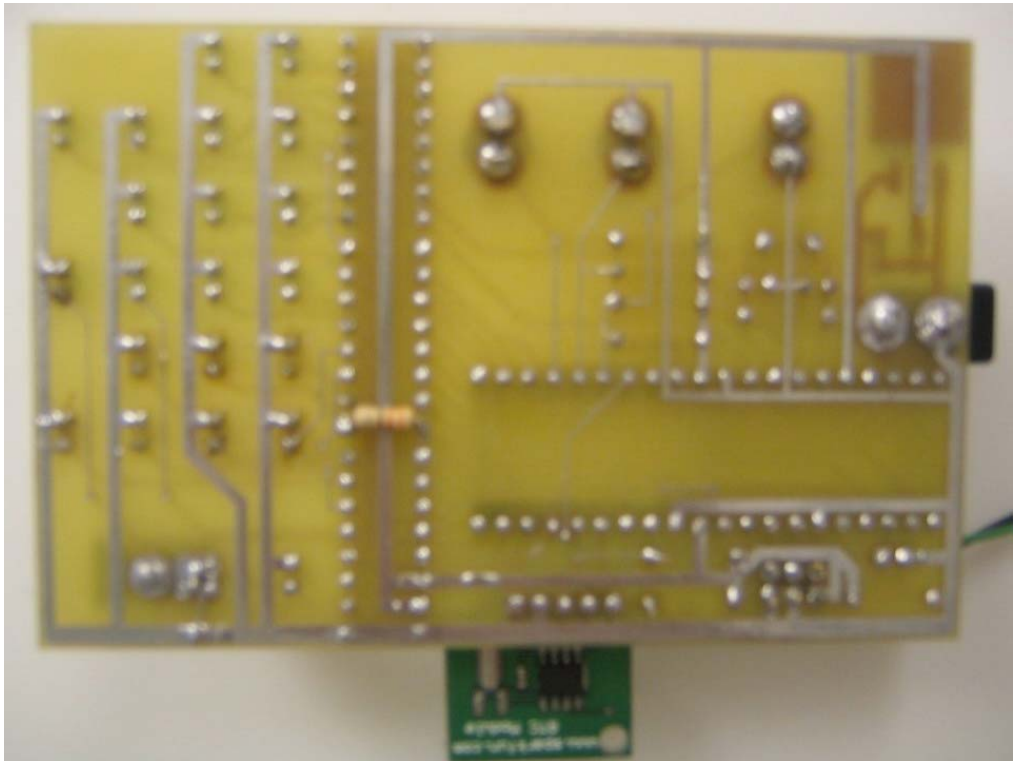
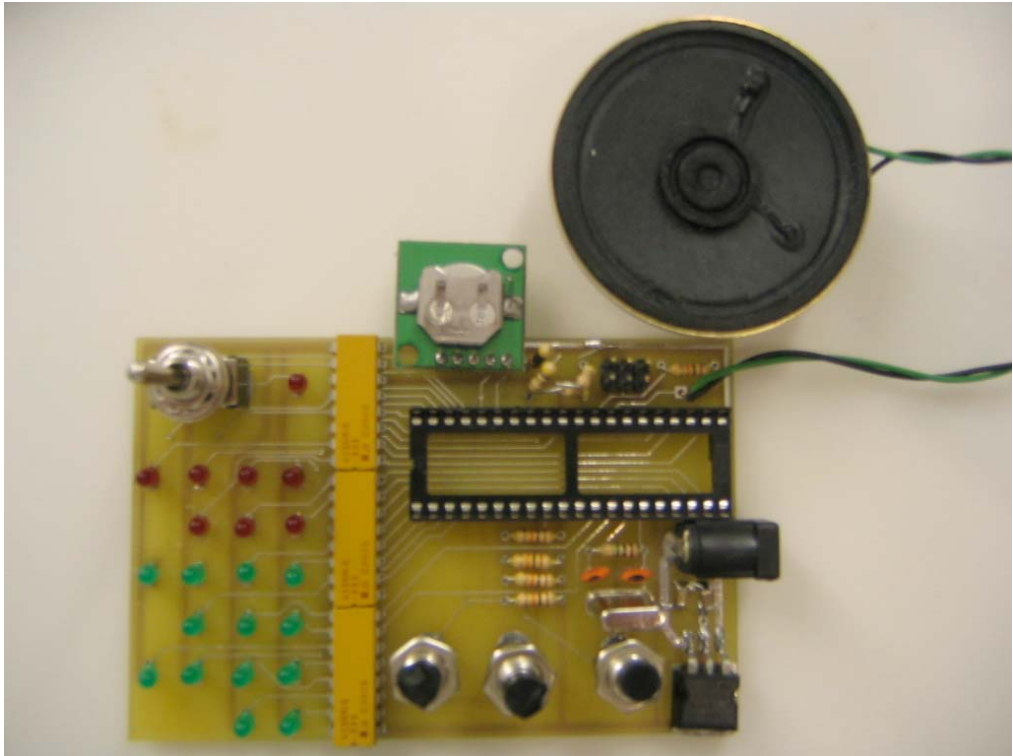
- Making the corrections to the schematic as shown above, and reflecting those changes in the design of the PCB
- Utilizing a breadboard to test components before soldering them to the board, to reduce the amount of de-soldering that I had to do.
- Developing the board around the DS1338/9 clock which uses 3V instead of the DS1307 (5V).
- Testing the RTC using the Z8 development board before connecting it to the board.
- Improve the switch de-bouncing.
- Store the alarm time in Flash so that it is preserved during power loss.
- Another improvement might be to use the square wave out (SQW) from the RTC to trigger an interrupt to 'fake' an increment of the seconds value and then read from the clock less frequently to synch up with the correct time (rather than reading every second as currently implemented).

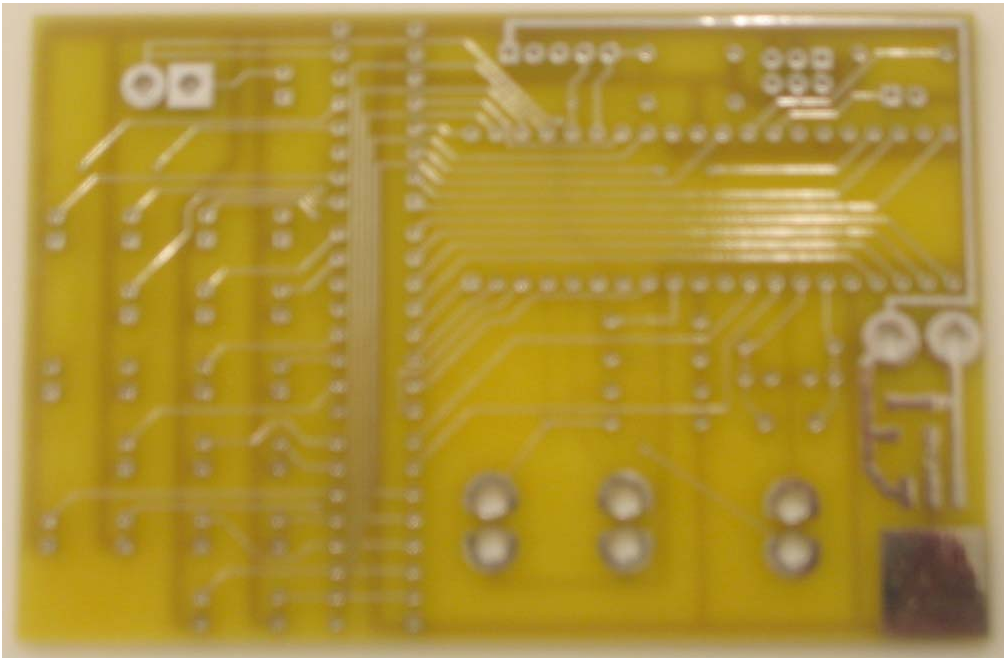
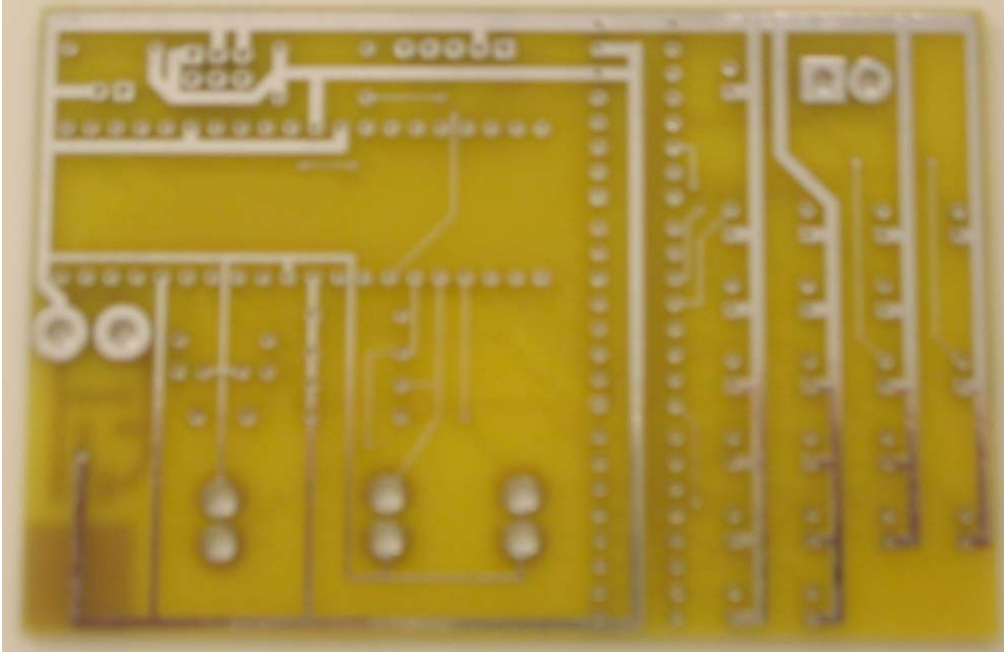
## Appendix 1: Pictures

[Apologies for the poor quality]











**Appendix 2: Source Code**

\* Find attached in a zip file.