Jackson Frankfurt

Microcomputer Design

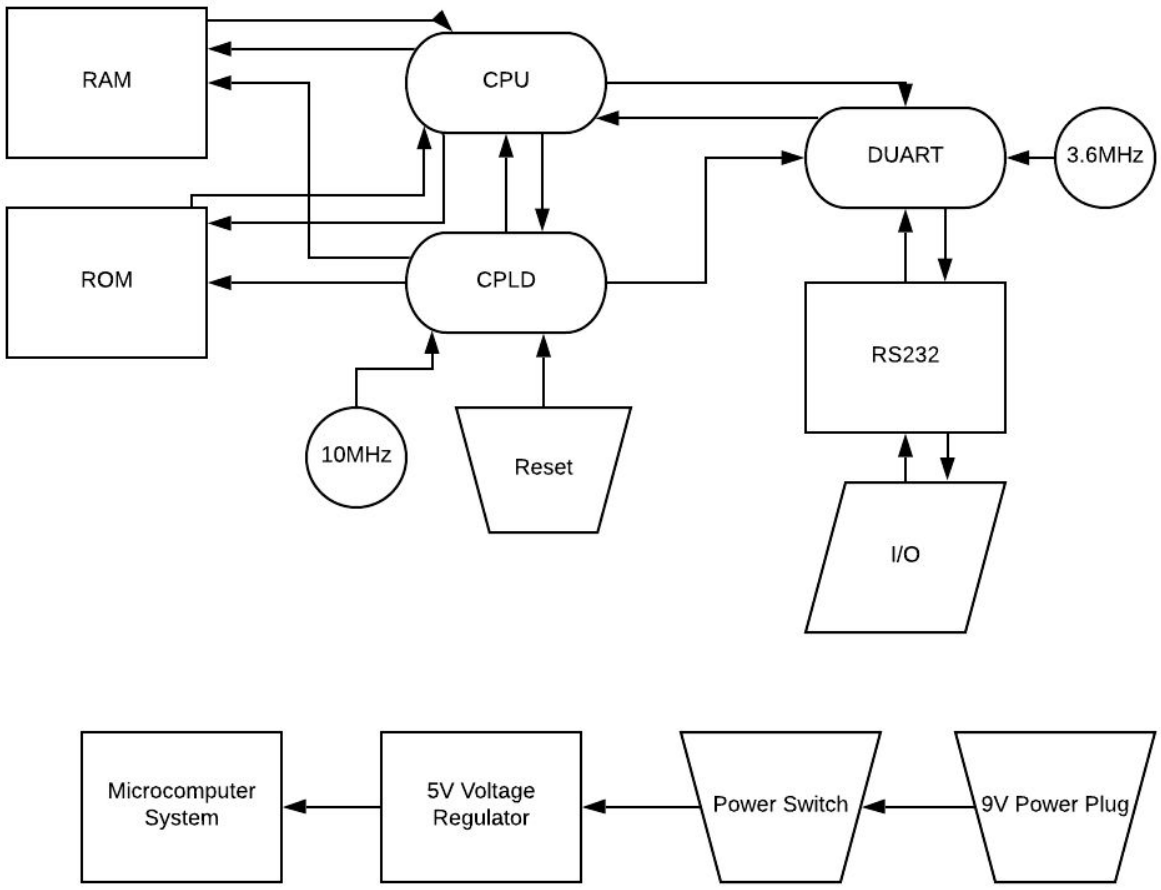
Spring 2019

Dr. Kim

Planning and reflection:

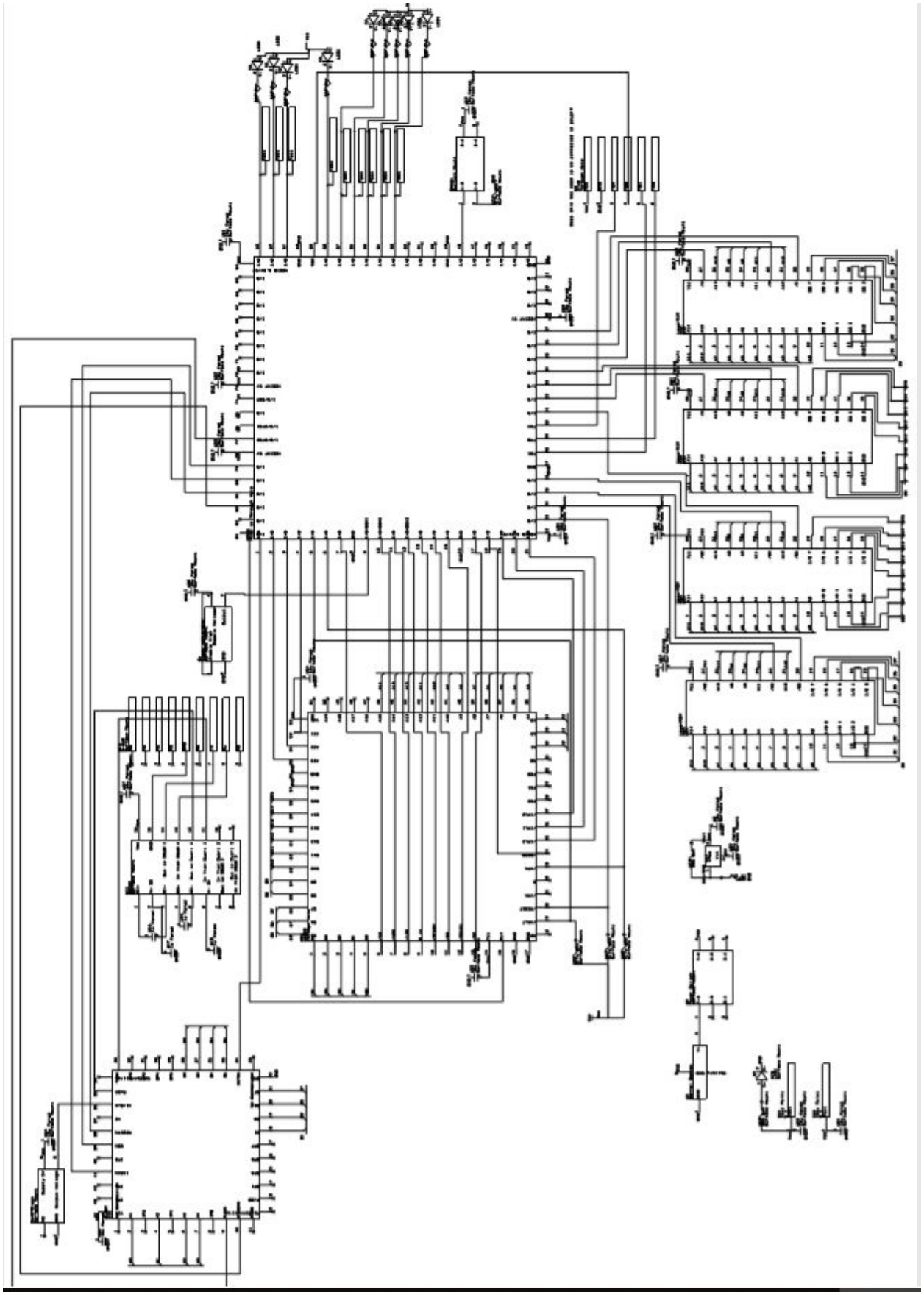
I started by selecting components for my microcomputer. Many of the parts I chose were recommended by lab instructors. I used a 68k CPU. I used a Xilinx CPLD to control chip activation and signals because it is programmable. I used a DUART and RS232 to communicate through a serial connection. I used 32kB of EEPROM to store the monitor program because it can be reprogrammed multiple times electronically.. I used 32kB of SRAM for working memory of the programs. I got these parts mainly from ebay, digikey, and mouser. If I were remaking the computer, I would either get a through-hole 10MHz clock or I would get solder paste to attach the clock more easily.

Computer Block Diagram:



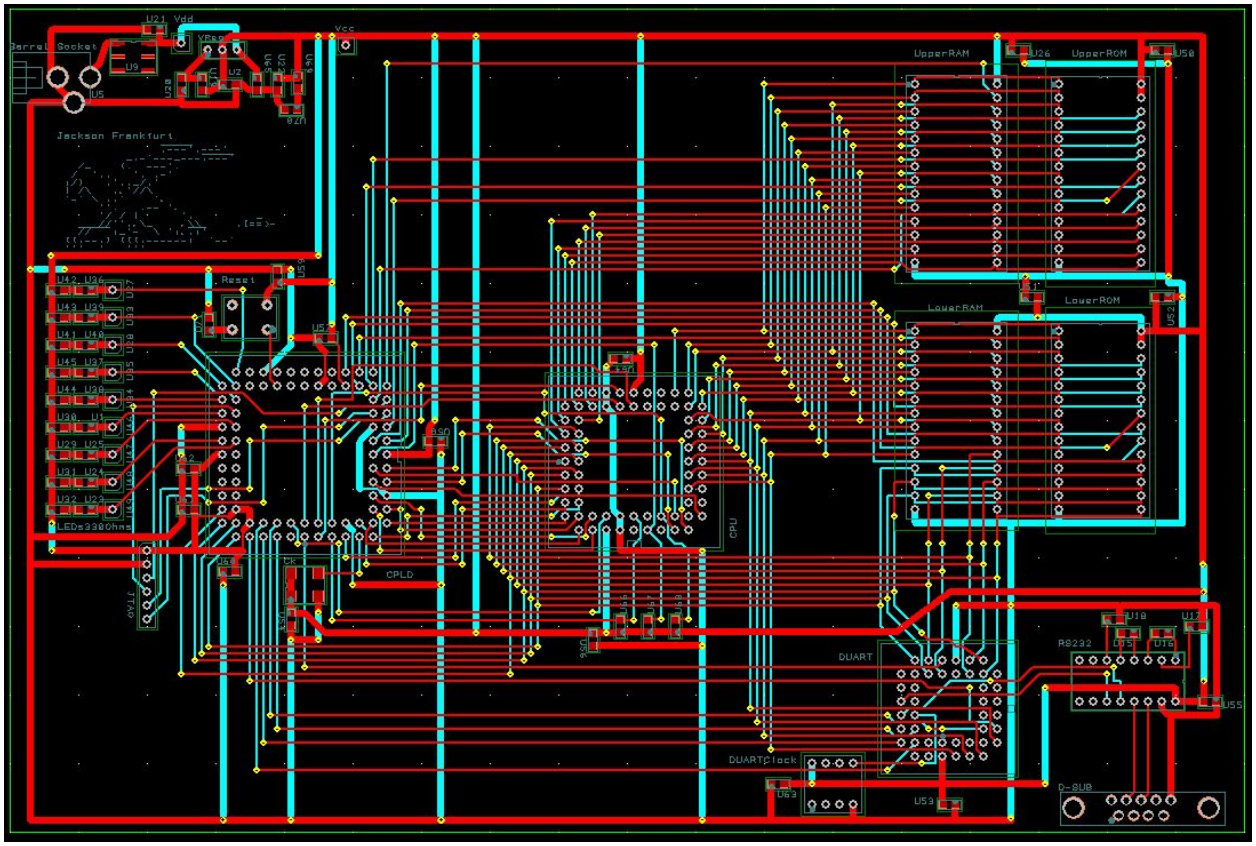
After part selection, it was time to design my PCB. I considered PCB Artist and KiCad to design my PCB. I decided on PCB Artist because I had past experience with PCB Artist and that would simplify the process. I designed symbols for the selected parts. When the part symbols were finished, I made the schematic design and the PCB design within PCB artist.

Schematic Diagram:



There are a few things I would change if I were making my PCB again. I would put the voltage regulator in a position where it could be mounted to the board for heat dissipation. I would add copper pads to the vias used to hold the D-Sub connector in place. With those copper pads, the D-Sub could be soldered in place by more points and would be more secure. I would move the voltage regulator further from my power switch to avoid touching the hot regulator when reaching for the switch. I would add more test points to important signals like the CPU's Halt and Reset. I would make sure probes could be attached to all signals from the top of the board whether through vias or test points. I could connect the lower byte of my databus to the CPLD to allow for extended peripheral functionality. I would connect the rest of my CPLD's I/O pins to test points. I would make built-in stand-offs for the PCB.

PCB Diagram:



When my PCB was designed, it was time to order it. Once my PCB arrived, I began to solder components to the board. The materials I used for soldering were: flux core solder, a face mask, my large soldering iron, and some metal wool to clean the solder tip. If I were redoing the project, I would purchase liquid flux to make the solder stick more easily to the desired areas. I would also get isopropyl alcohol to clean the board afterwards.

I had two planning issues with my hardware. My first issue was a problem with voltage regulation. My regulator only dropped my voltage down to 8V instead of 5V. The problem was that I misunderstood the datasheet. I removed two resistors to fix that. My second problem was that I did not include two pull-up resistors in my original design. They were 4.7kOhm resistors to pull inactive DUART DTACK and DUART RESET.

Implementation and troubleshooting:

In the process of soldering my board for the first time, I made a few mistakes. The first mistake I found was that I soldered my 0805 LEDs into 0805 capacitor pads. The second mistake I found was that I had a cold-solder on my pull-up resistor to CPU HALT. The third mistake I found was a cold-solder somewhere on my RAM or ROM. After cleaning those solders up, my software worked more reliably. What follows is more of a narrative of my implementation subject by subject.

When my board was shipping and also after my board was assembled, I worked on writing VHDL to program my CPLD to control the different chips on my board. I first wrote a simpler test program to make sure the CPLD was working correctly. Unfortunately, that program used too many macrocells for it to load onto the CPLD. I spoke with a lab instructor and he recommended simplifying the logic and removing processes within the VHDL in order to reduce the number of macrocells used. Instead of processes, I used several ‘when’ statements. I applied the principle to my main VHDL program and it fit correctly onto my CPLD. Unfortunately, I did not get the output I expected from my CPLD. All my output test lines were inactive, regardless

of having hard coded output. I began to investigate.

At some point, I found that my clock signal going into the CPLD was shorted, so it was not producing the correct output. This was probably related to the clock being a small surface mounted component and my tools and skill not being up to the task of soldering it correctly to the board. So, I removed the clock and damaged my PCB in the process. I patched up the mistake with a working clock and jumper wires. Finally, the CPLD was giving the expected output.

The next step was loading the ROM test and measuring my output. Unfortunately, my CPU did nothing. After some searching, I found that my CPU Halt signal was going active. I measured the resistance of the pullup resistor I had pulling CPU Halt inactive and found that it was much greater than expected. It turned out to be a cold solder joint. After fixing the resistor, I had the same problem: CPU Halt was pulled high. After more searching some more, I found that when I fixed my 10MHz clock, I shorted the Halt signal to the clock signal. That drove the CPU into a halt state immediately. I fixed the short and my ROM and RAM tests seemed to work.

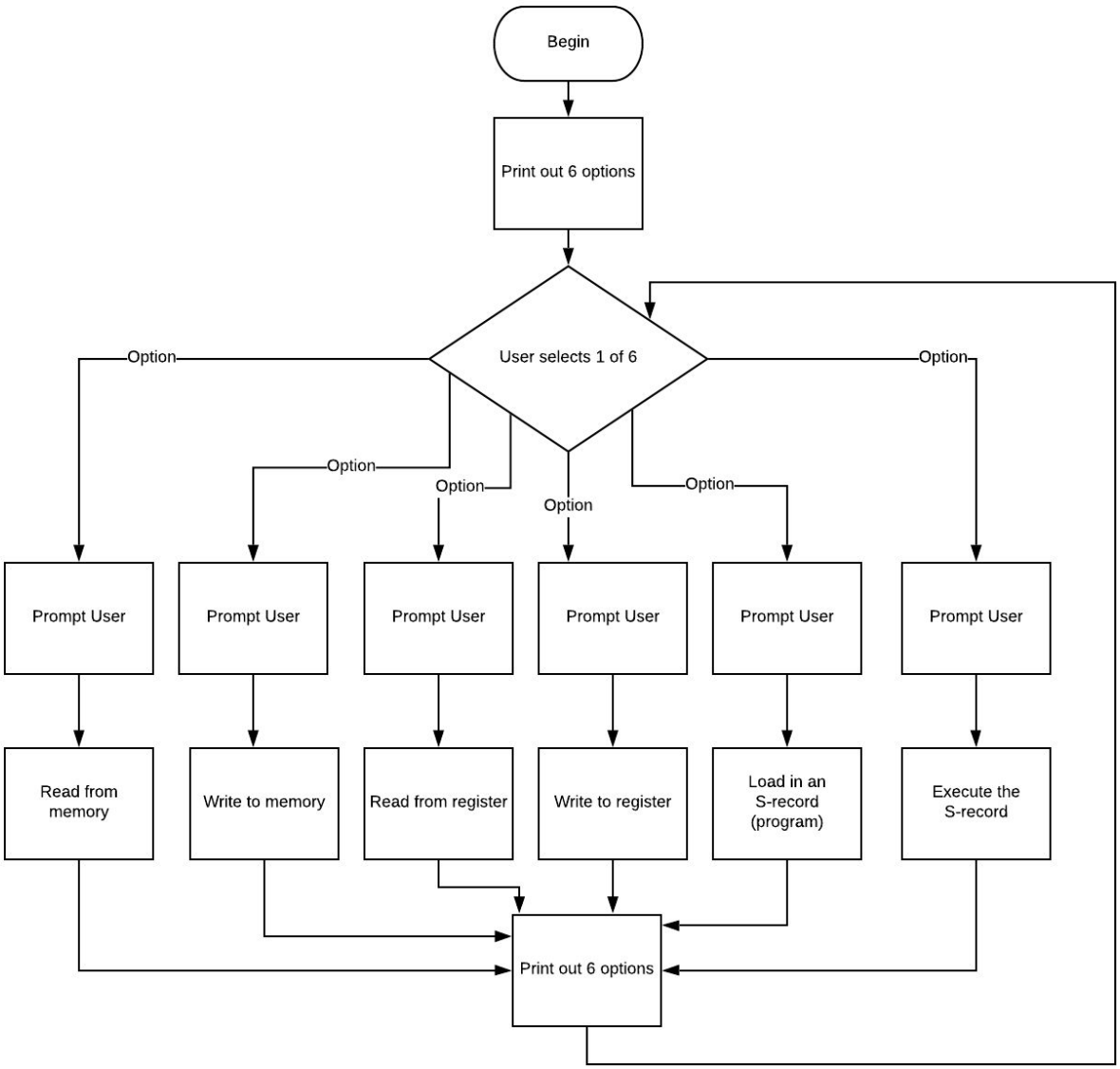
Next, I needed to test my connection to the serial port. I connected my board and desktop computer with a Null Modem Cable. I opened Putty and started a connection at a Baud rate of 19200. I intentionally shorted the Rx/Tx lines before and after my RS232 in order to echo data back to the monitor in order to make sure my connections were good. Then I loaded in a DUART test with the DUART in auto-echo mode. That didn't work at first, so I replaced the DUART. After the replacement, the auto-echo worked.

The next step was a software echo. I had an issue on my computer where the DUART would store a queue of 3 bytes. It would output the same byte 3 times before going to the next value. The solution was to flush my DUART after each input. Once I did that, the software echo worked correctly.

At this point, I needed a monitor program to see if I could get my computer to work. I spent several hours writing a x68 program in the easy68k simulator. I wrote subroutines for

putChar, getChar, getHex, getAscii, getByte, getWord, getLongword, and each of the user-facing routines.

Software diagram:



When the software system worked as intended within the simulator, I tried to put it onto my computer. It failed for a few reasons. I did not initialize my stack correctly, I did not declare variables in the right place. I wrote defects into the code that were undetected because the

simulator initializes memory to FF and registers to zero. Once these problems were solved, the program still didn't work completely. The logic was all correct. But something else was wrong.

My problem was that after a few operations, the CPU would enter a Halt state. This was a mysterious problem. I was unsure whether the issue was software or hardware. It seemed unlikely to be hardware because my hardware tests had gone well. In the end, it was another soldering issue. After resoldering my RAM and ROM, my computer no longer halted unexpectedly.

The final result was a single board computer that could perform 6 user-facing routines:

1. Read from a memory location
2. Write to a memory location
3. Read from a register
4. Write to a register
5. Load in an S-record
6. Execute the S-record