EEL 3701 — Digital Logic & Computer Systems
Revision **1**
Lab **6** Report: **Elementary CPU Design: Fetching Instructions From a ROM**

Dupuis, Connor
Class #: 12478
Kevin Lovell
Month Day, Year

PRE-LAB QUESTIONS OR EXERCISES

Part 1:

1. Why did we require the new instruction register in this design?
    a. So that we wouldn't have to manually have inputs.
2. In this section of the lab you are setting the INPUT bus by hand. If you wanted to read or fetch this value from memory, what could you add to do this automatically for you every CLK cycle?
    a. You could add a ROM that it reads information from.
3. How would you add more instructions (i.e., 8 instead of 4) to the controller?
    a. By adding more IR slots. Having 2 gives you 4 instructions, 3 gives you 8, 4 gives you 16, etc.

Part 2:

1. Why do we need the extra states in the LDAA and JMP instruction paths?
    a. For LDAA it first must read the instruction to load a value and then the value itself. As for JMP, it must read the instruction to jump and then the values at the location it jumped to.
2. What do you need to do to the address lines to get your program to start at address $4370 (instead of $3E70)?
    a. You need to ground or vcc address bits A[14..4] to the respective value of $3E70. Then it will start at $3E70 and increment from there.

PROBLEMS ENCOUNTERED

N/A

REQUIREMENTS NOT MET

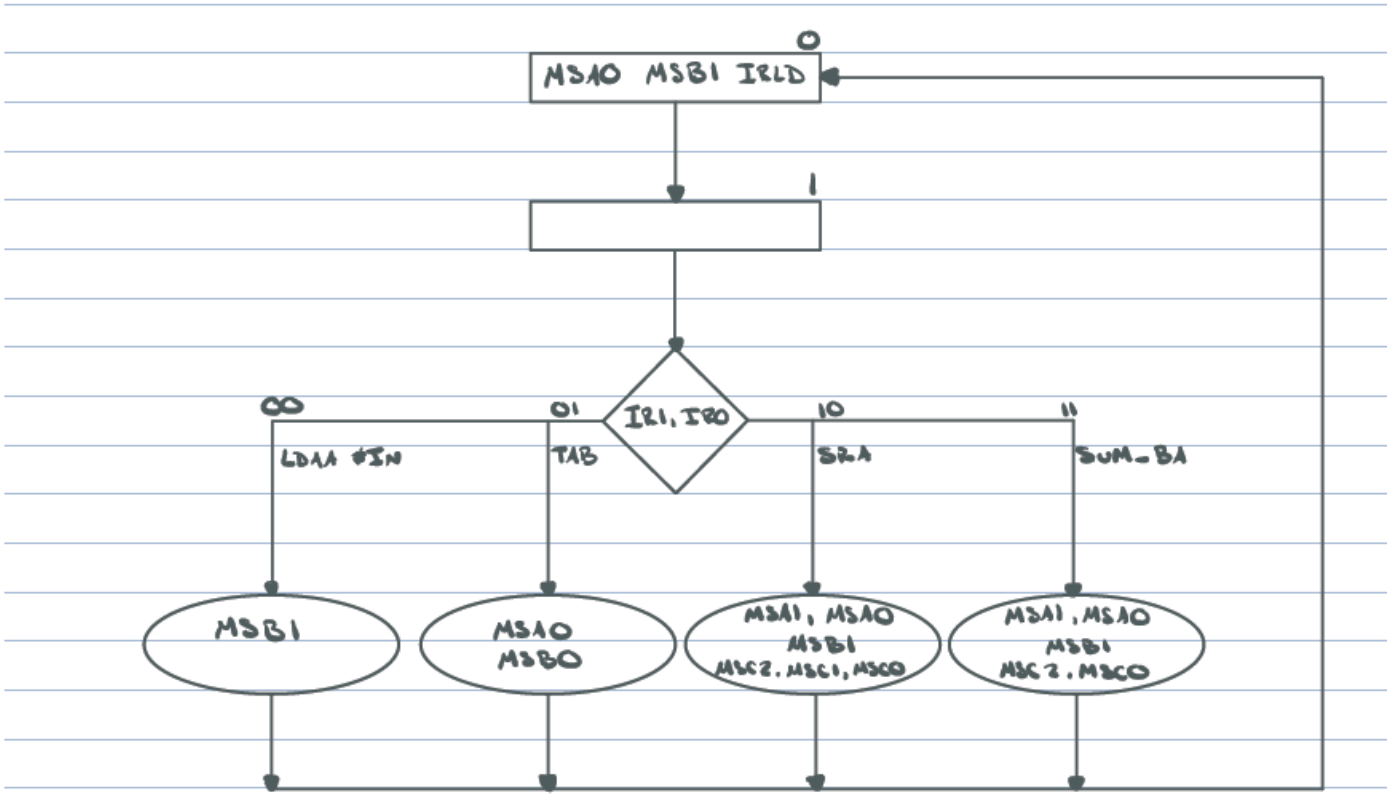
N/A

FUTURE WORK/APPLICATIONS

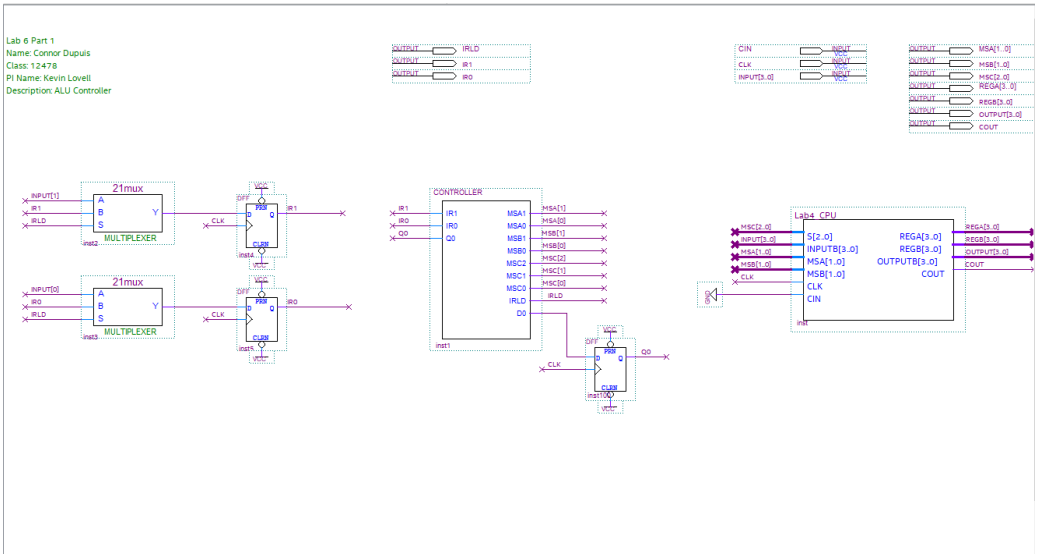
This lab had us create our own CPU which can read instructions from ROM. Having the knowledge of how a basic CPU works and functions is important for moving towards any field of technology. Understanding how something like the CPU, which is the foundations of many forms of technology, is something what is immensely valuable. I could apply this knowledge to virtually any field or work applications involving some sort of computer.

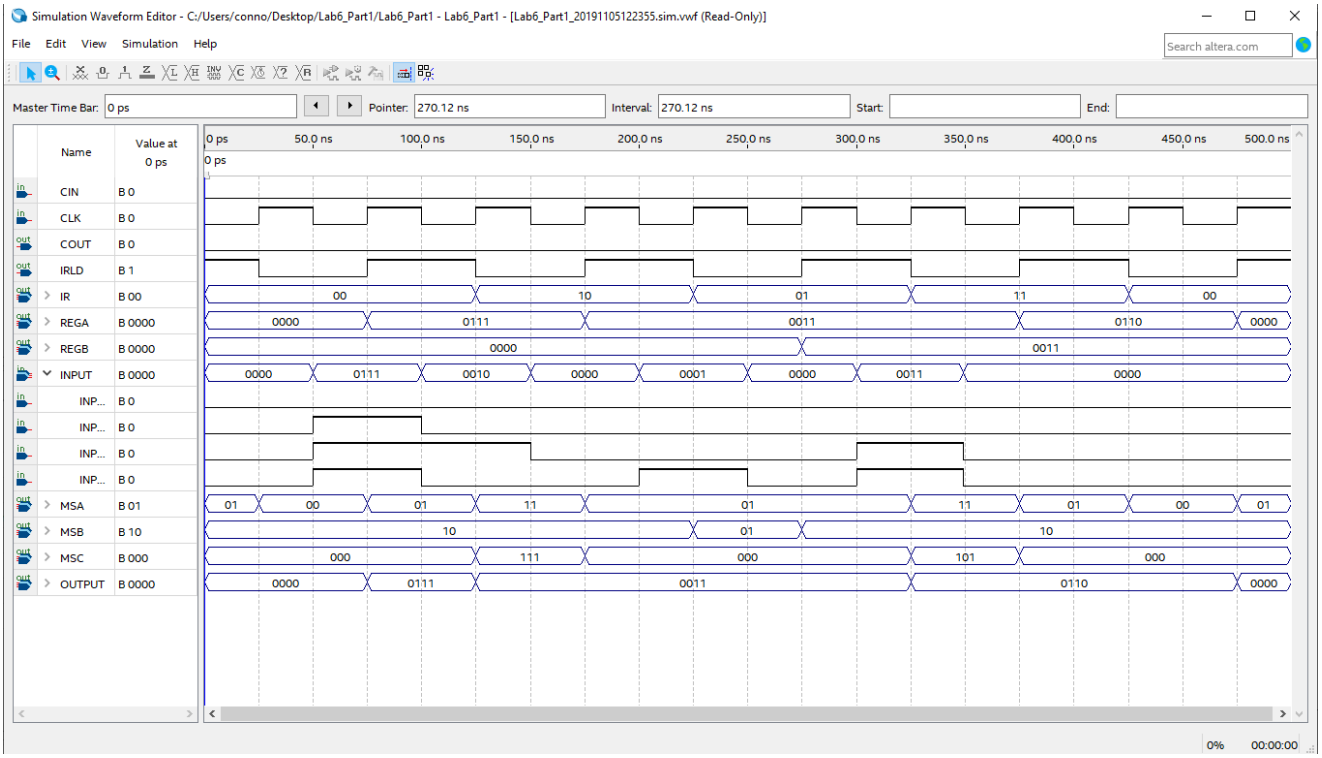
PRE-LAB REQUIREMENTS (Design, Schematic, ASM Chart, VHDL, etc.)

Part 1:



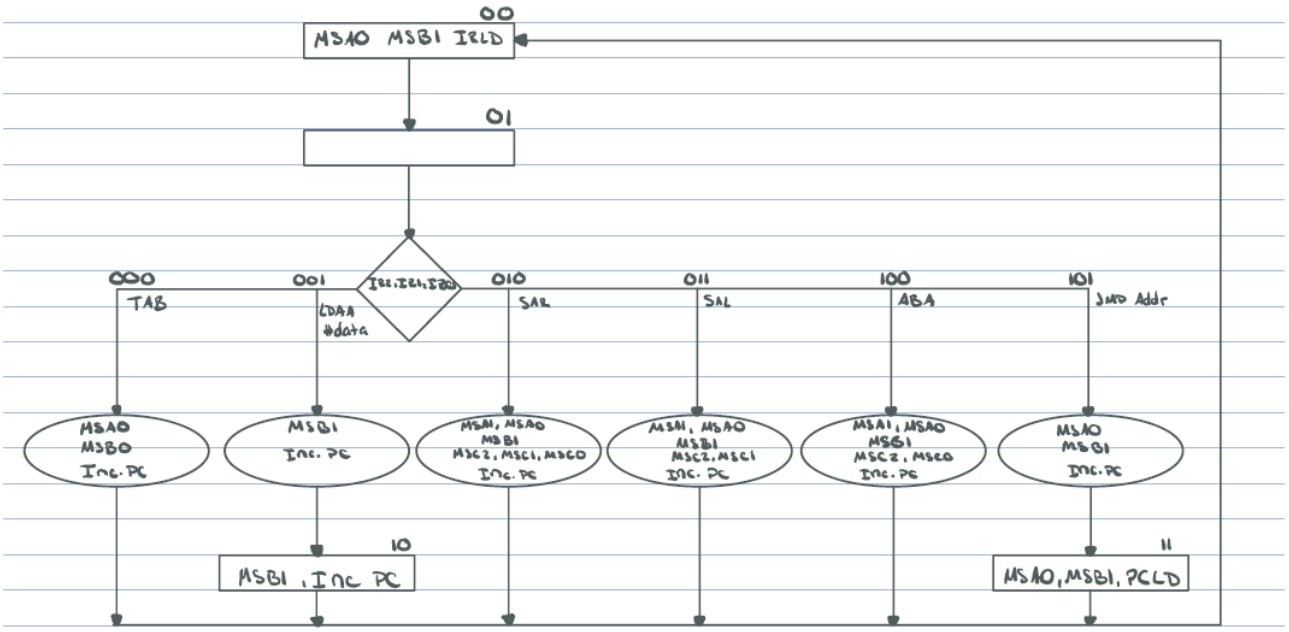
	Q_0	IR1	IR0	Q_0^+ (D)	MSA1	MSA0	MSB1	MSB0	MSC2	MSC1	MSC0	IRLD
Input	0	x	x	1	0	1	1	0	0	0	0	1
LDAA	1	0	0	0	0	0	1	0	0	0	0	0
TAB	1	0	1	0	0	1	0	1	0	0	0	0
SRA	1	1	0	0	1	1	1	0	1	1	1	0
SUM	1	1	1	0	1	1	1	0	1	0	1	0





First, we set the Input to 0000 to load in the number 7 into REGA on the next clock cycle. After the number is loaded, we shift it right and put it in REGA (Input – 0010). Then we put the number in REGA into REGB (Input - 0001). Finally, we add REGA and REGB and put that number in REG A (Input - 0011).

Part 2:

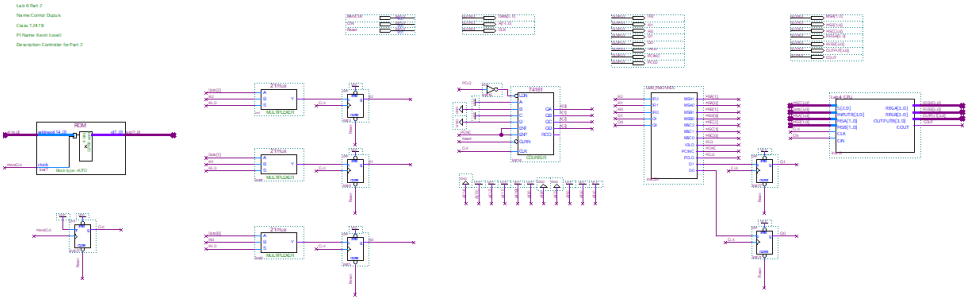


					D_1	D_0										
Q_1	Q_0	IR2	IR1	IR0	Q_1^+	Q_0^+	MSA1	MSA0	MSB1	MSB0	MSC2	MSC1	MSC0	IRLD	PC.Inc	PC.LD
0	0	X	X	X	0	1	0	1	1	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0
0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	1	0
0	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	0
0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	1	0
0	1	1	0	0	0	0	1	1	1	0	1	0	1	0	1	0
0	1	1	0	1	1	1	0	1	1	0	0	0	0	0	1	0
1	0	X	X	X	0	0	0	0	1	0	0	0	0	0	1	0
1	1	X	X	X	0	0	0	1	1	0	0	0	0	0	0	1

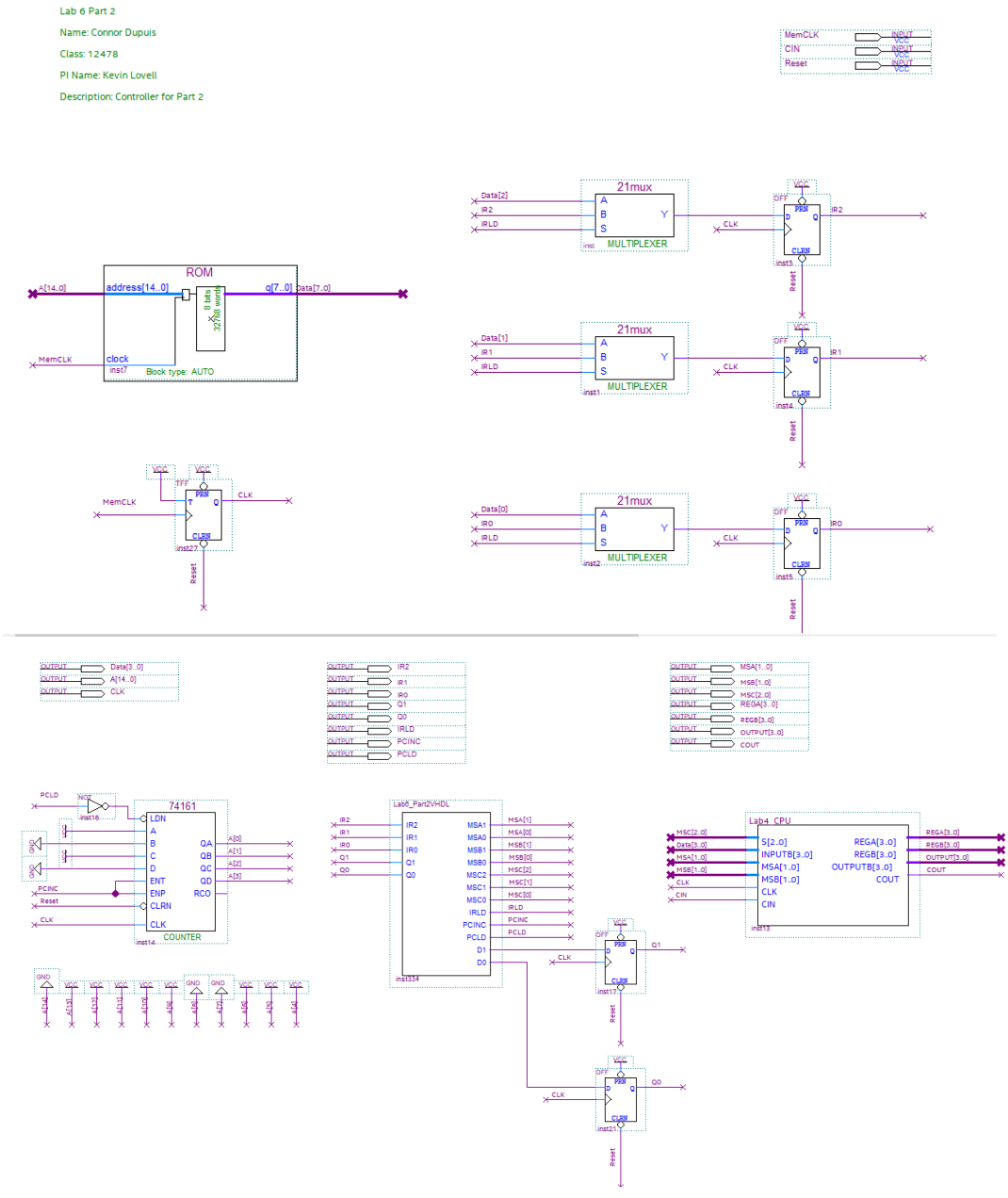
Addr		Mach code	A	B	A	B	A	B	A	B
$3E70-$3E71	LDAA #7	001	0111	X	X	X	X	X	X	X
$3E72	TAB	000	0111	0111	X	X	X	X	X	X
$3E73-$3E74	LDAA #3	001	0011	0111	X	X	X	X	X	X
$3E75	SAL	011	0110	0111	1100	0110	0010	0001	0010	0001
$3E76	ABA	100	1101	0111	0010	0110	0011	0001	0011	0001
$3E77	SAR	010	0110	0111	0001	0110	0001	0001	0001	0001
$3E78	TAB	000	0110	0110	0001	0001	0001	0001	0001	0001
$3E79	JMP 5	101	0110	0110	0001	0001	0001	0001	0001	0001
$3E7A-$3E7B	LDAA $F	000	X	X	X	X	X	X	X	X

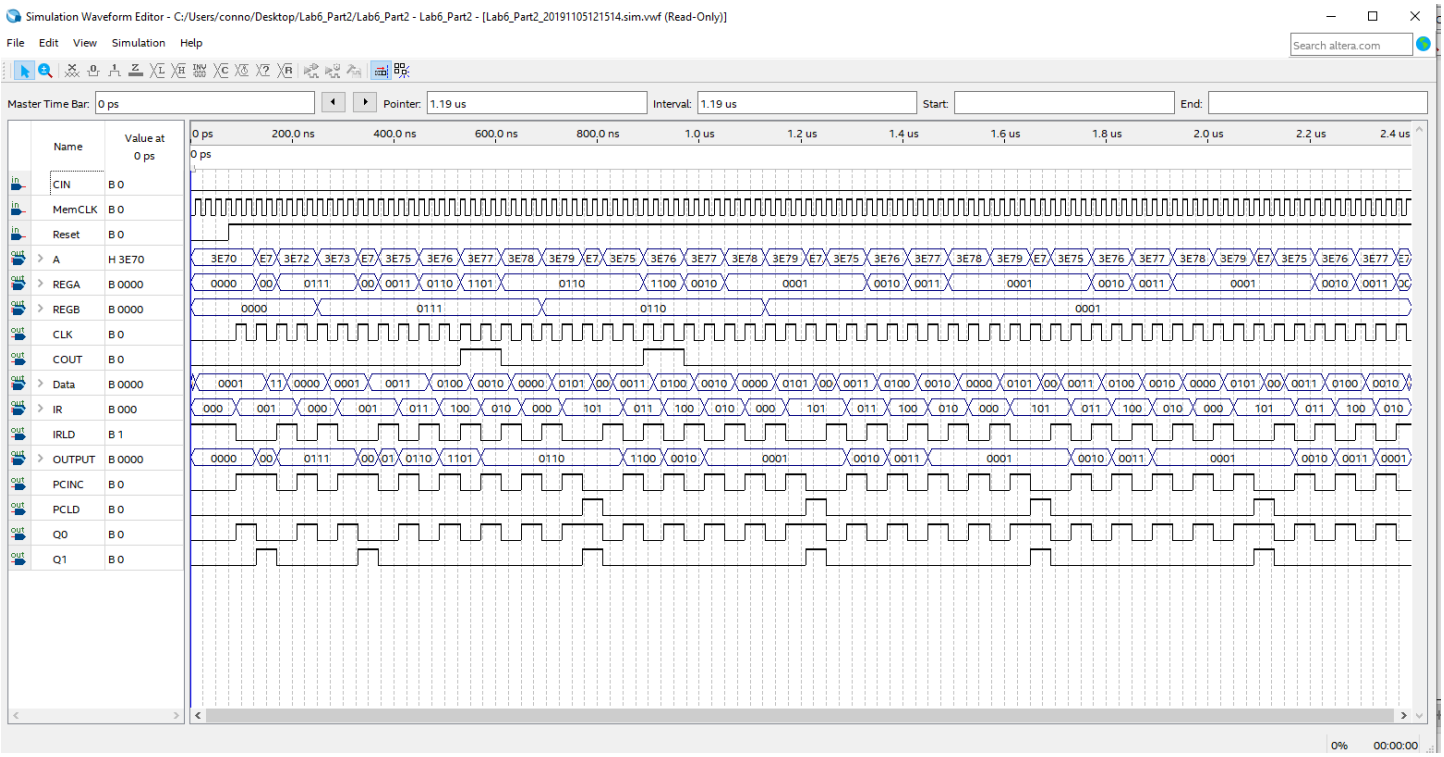
Part 2 MIF:

Full Picture:



Zoomed In:

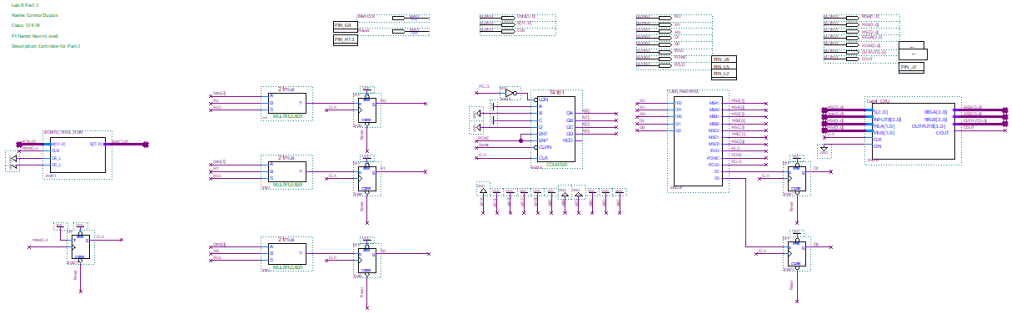




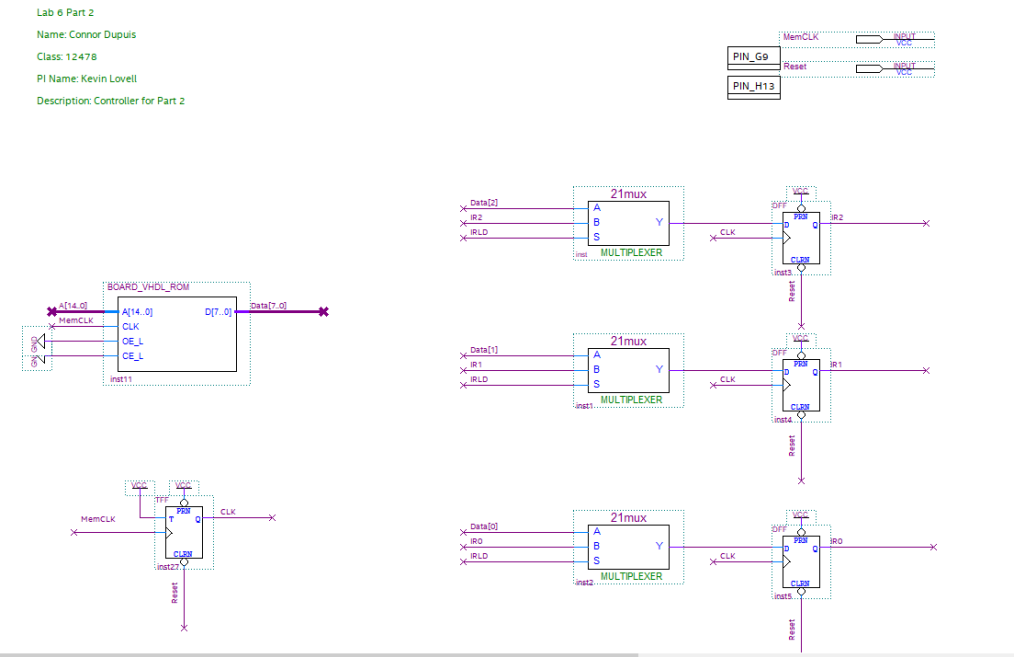
The MIF file has information starting at address $3E70, so for address bits A[14..4] I grounded and vcced them accordingly so that the simulation begins at that address. From there it pulls the information from the ROM and follows the instructions set up in the code I wrote. It first loads 7 into REGA, then pushes it to REGB. Next it loads in 3 to REGA and then shifts REGA left. Then it sums REGA and REGB into REGA. Next it shifts REGA right and pushes that into REGB. Finally, it jumps back to address $3E75 and continues the instructions from there. Eventually REGB gets stuck at 0001, and REGA alternates between 0001, 0010, and 0011 continuously.

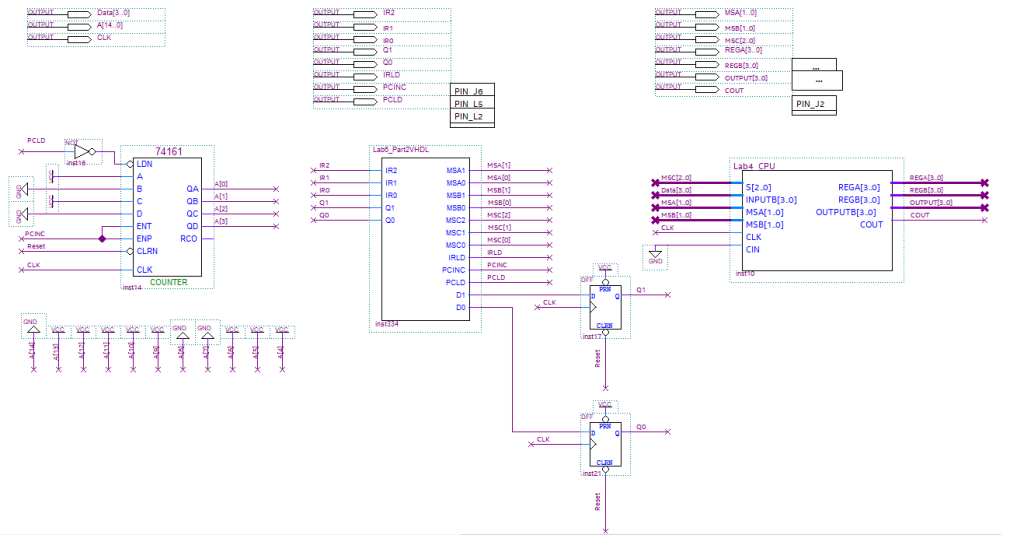
Part 2 VHDL:

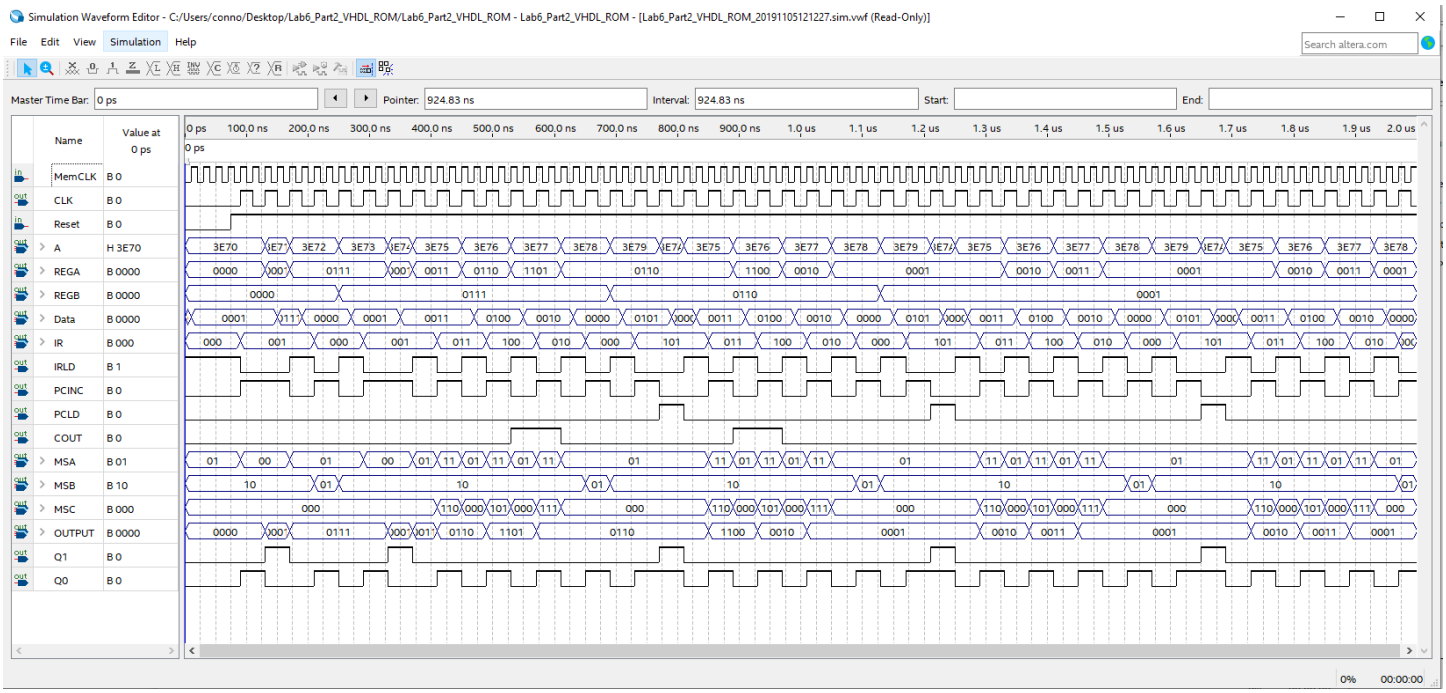
Full Picture:



Zoomed In:







This simulation is identical to the one shown by the MIF file, but instead uses VHDL opposed to a MIF file for the ROM. It pulls the information from the ROM and follows the instructions set up in the code I wrote. It first loads 7 into REGA, then pushes it to REGB. Next it loads in 3 to REGA and then shifts REGA left. Then it sums REGA and REGB into REGA. Next it shifts REGA right and pushes that into REGB. Finally, it jumps back to address $3E75 and continues the instructions from there. Eventually REGB gets stuck at 0001, and REGA alternates between 0001, 0010, and 0011 continuously.