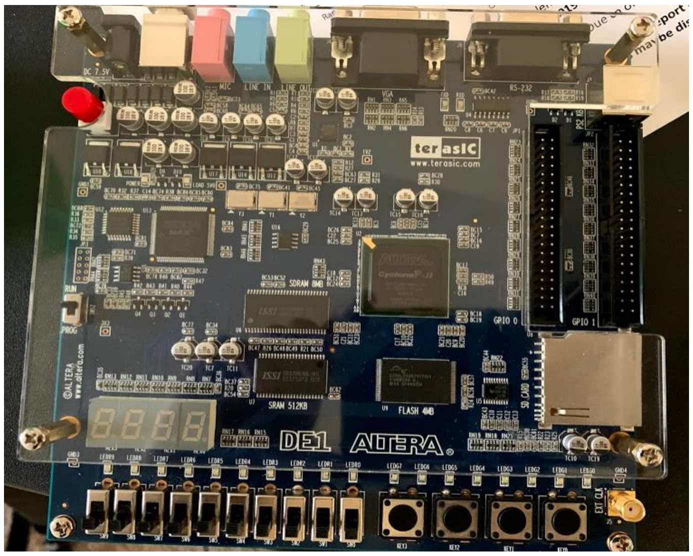


TERM PROJECT REPORT

CSE 2441- FALL 2019

Name: Abhishek Dhital
Student ID: 1001548204

Submitted to:

Dr. Bill D Carroll

UT Arlington

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

1.1. Project Overview

The term project consists of design and implementation task of the Tiny Reduced Instruction Set Computer (TRISC) on the Altera DE1. We use several components to be able to implement the task on the DE1 as required. The main objective of the task is to put together the components such as RAM, Program Counter, Accumulator, ALU, Instruction Register, and the Control Unit to be able to read, write or perform the task as per the received control signals from the Control Unit. The schematic for the completed TRISC looks as shown in Figure 1. The figure shows the final schematic for the TRISC. However, we divided our project into two parts: A and B, therefore in part A, the ALU part was not present and there were less control signals used than in part B, which was the fully functioning version of the TRISC.

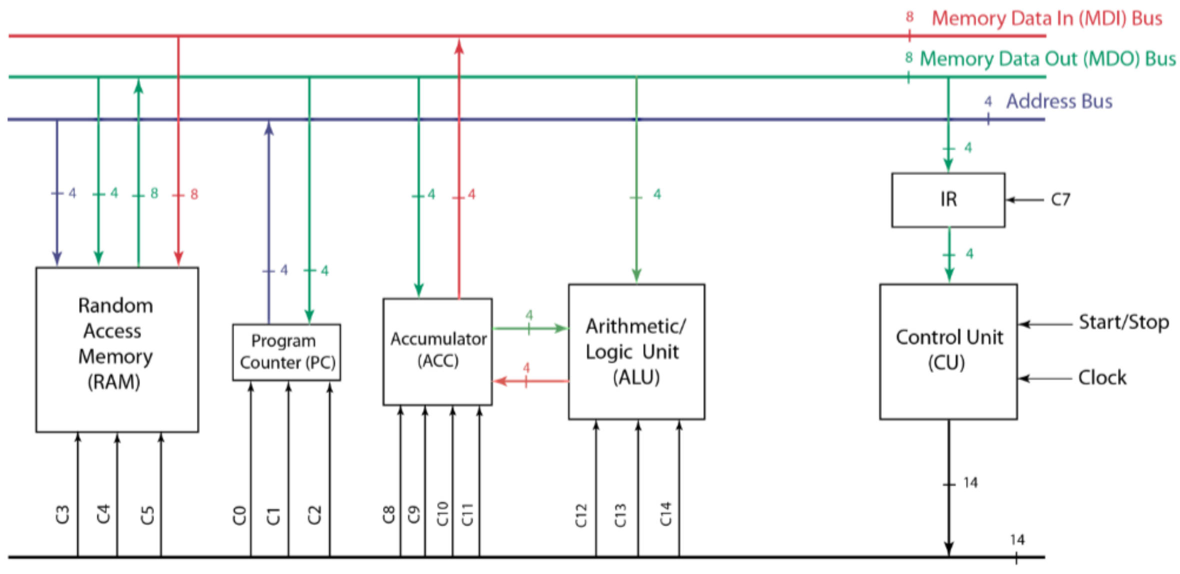


Figure 1 TRISC with the INC, CLR, LDA, STA, ADD and JMP

1.2. Project Status

By the time this report was compiled, the TRISC design was completed and covers all the bases required by the scope of the project.

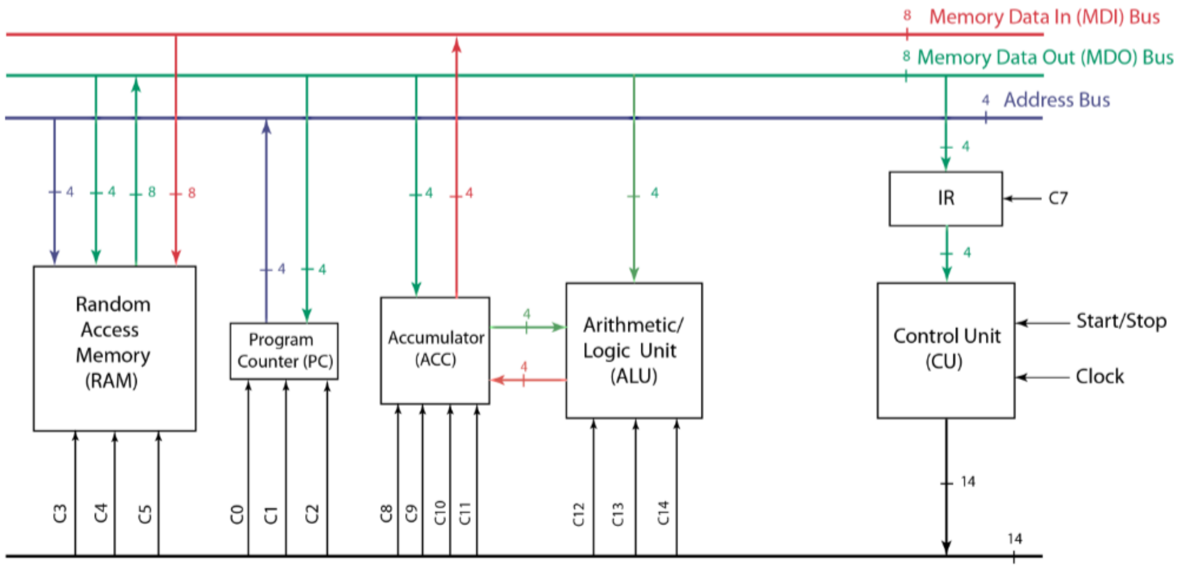
1.3. Report Overview

This report will take us through the system design and procedure, the controller design procedure, any alternative designs that were considered, and the integration and test plan for the design project. It will describe, in detail, the working mechanism of each and every hierarchical component included in the project design and by the end of the report, the reader should be able to understand thoroughly about TRISC and its components.

2. System Design

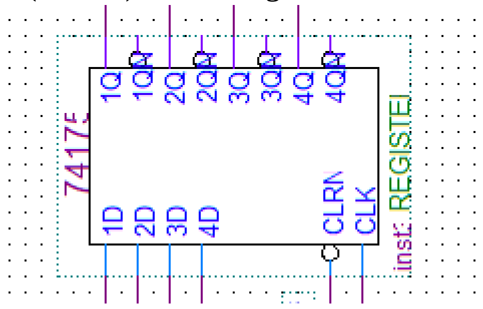
2.1. System-Level Description and Diagrams

As seen on figure 1, the components of the TRISC were put together to build what the project required, and it had various system level components each of which had its own specific task in the functioning of the TRISC. It can also be seen in the figure below.


8 Memory Data In (MDI) Bus
8 Memory Data Out (MDO) Bus
4 Address Bus
Random Access Memory (RAM)
Program Counter (PC)
Accumulator (ACC)
Arithmetic/ Logic Unit (ALU)
IR
C7
Control Unit (CU)
Start/Stop
Clock
C3 C4 C5 C0 C1 C2 C8 C9 C10 C11 C12 C13 C14
14

2.1.1. Instruction Register (IR)

It is a 4-bit pin in pin out register which loads the data from the Memory Data Out (MDO) bus into the instruction decoder of the control unit. For the project, a 74175 IC was used as an instruction register which loads the data at pins (1D-4D) to (1Q-4Q) when a signal is received at the CLK pin.



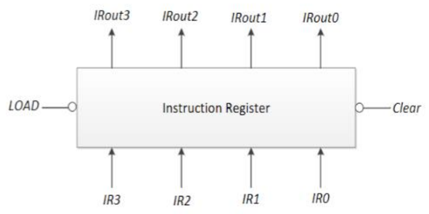


Figure 2 IC 74175

2.1.2. Control Unit (CU)

It is a Finite State Machine which gets the data from MDO bus loaded from the instruction register and accordingly gives out the control signal to the Control Bus (CB). It consists of a 4-to-16bit instruction decoder which decodes the 4-bits received from the instruction register and loads it into the controller unit, which then further performs its operation to output the actual control signal.

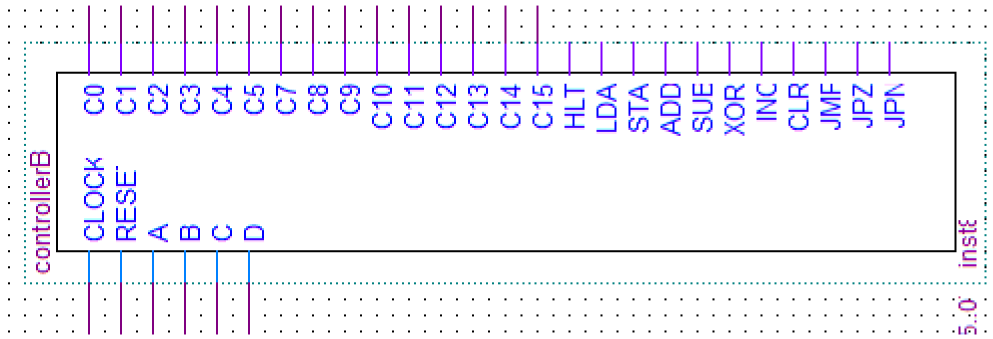


Figure 4 Controller symbol

At each clock, the pins A,B,C,D receive the input from the instruction register which are decoded by the instruction decoder and then the control signal C0-C15 are turned on according to the instruction received from the MDO. The instructions and their subsequent Op Codes are shown in the figure below.

Instruction	Function	Register Transfer	Op Code
LDA	Load ACC	ACC ← (MDR)	0000
STA	Store ACC	MDR ← (ACC)	0001
ADD	Add ACC	ACC ← (ACC) + (MDR)	0010
SUB	Subtract ACC	ACC ← (ACC) – (MDR)	0011
XOR	XOR ACC	ACC ← (ACC) ⊕ (MDR)	0100
INC	Increment ACC	ACC ← (ACC) + 1	0110
CLR	Clear ACC	ACC ← 0	0111
JMP	Jump	PC ← (MDR)	1000
JPZ	Jump if 0	PC ← (MDR) if Z = 1	1100
JPN	Jump if < 0	PC ← (MDR) if N = 1	1001
HLT	Halt	PC ← 0	1111

The opcodes are what the instruction register receives from the MDO, which is then loaded into the controller unit. For the part A of the project, we successfully implemented the INC, CLR and JMP instructions and after the completion of part B, the LDA, STA and ADD instructions were successfully added into the controller FSM.

Also, the input pins unused were there just in need of debugging the design, and it would make it easier for us to determine whether the accurate instructions were being decoded and loaded into the control unit.

2.1.3. Program Counter (PC)

The program counter is a four-bit binary counter which is used to get the data from the Memory Data Out (MDO) and load it by incrementing (if required) into the Address Bus (AB). A 74193 IC was used as a program counter in the project block diagram. As seen in figure 6, the input pins A-D receive the data from MDO, and the it outputs the data when a signal is received at the input pin LDN and the data is incremented before output if there is a control signal received at the input pin UP. The input pin CLR is used to clear the program counter whenever the control signal for CLR is received.

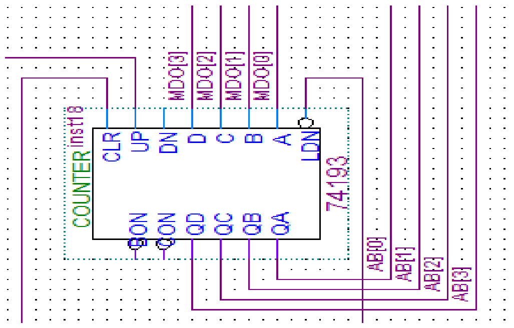


Figure 5 Program Counter

2.1.4. Accumulator (ACC)

The main purpose of the accumulator is to receive data from the MDO bus, and the ALU, and select only out of them to load it into the Memory Data In (MDI) bus. The input pins ALU1-ALU4 receive the output of the ALU and the input pins MDO1-MDO4 receive the data from the MDO bus. Now based on the signal received at the input pin *selector*, at each *Load*, the accumulator outputs either of the ALUs or the MDO data and loads it into the MDI bus. The accumulator selects the data using a multiplexer. The Control Signal 10 (CB10) determines which of the MDO or ALU to output. When CB10=0, the ALU is loaded into the MDI, and when CB10=1, the MDO is loaded into the MDI.

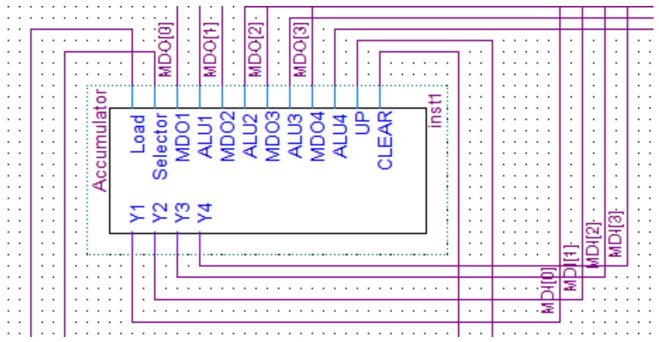


Figure 6 Accumulator showing input and output pins

2.1.5. Arithmetic/ Logic Unit (ALU)

The ALU is introduced in the part B of the project because part A did not consist of the LDA, STA and ADD instructions. So, the accumulator would only load the MDO into MDI. However, in the part B, ALU is essential on many levels. The input pins A3-A0 receive the data from the MDI while the input pins B3-B0 receive the data from MDO. Now, based on the signals on pins S1 and S0, the ALU determines whether to Add, Subtract, XOR or AND the two data. After it has been determined, the output data is received at pins R4-R1 which are then loaded into the input pins of accumulator using a flag register for further proceedings.
The function code for the ALU input pins S1S0 were as follows **(00: ADD, 10: SUB, 01: AND, 11: XOR)**.

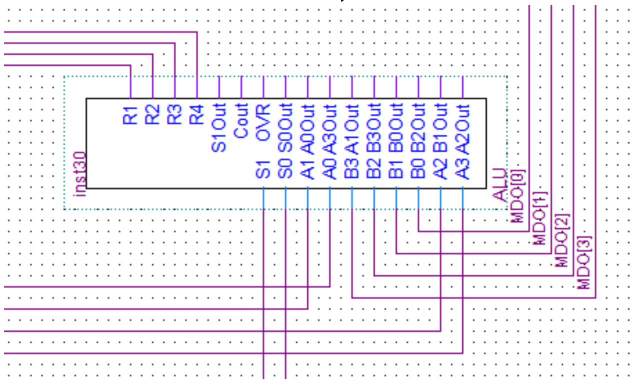


Figure 7 Arithmetic/ Logic Unit (ALU) showing input and output pins

2.1.6. Random Access Memory (RAM)

This is the RAM for out Reduced Instruction Set Computer which functions as a normal RAM. For our project, two different RAMs were provided for part A, and part B respectively. The basic functioning of the RAM was to receive the data from the MDI bus, and the address from the address bus (AB), perform the operation, and output the data and load it into the MDO bus. The control signal CB[5] determined whether to enable the RAM Read/ Write cycle, and therefore it was connected to the *wren* pin of the RAM. Also, the RAM was clocked when both system clock signal and the clock signal from the controller (i.e. CB[4]) were received. So, the two signals were ANDed and the subsequent output was connected to the *clock* input pin of the RAM.

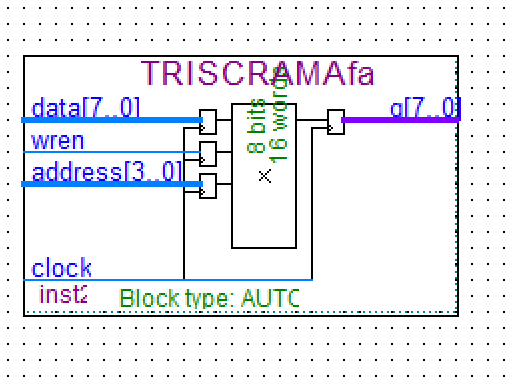


Figure 9 RAM used for part A

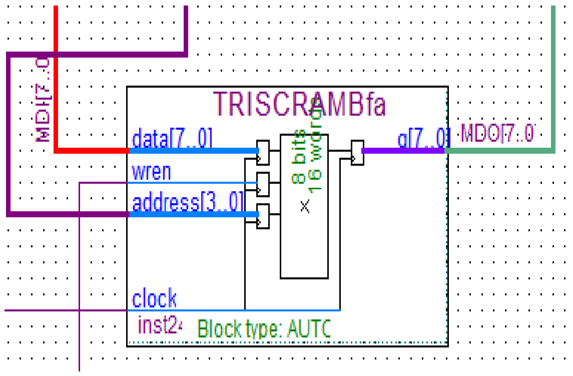


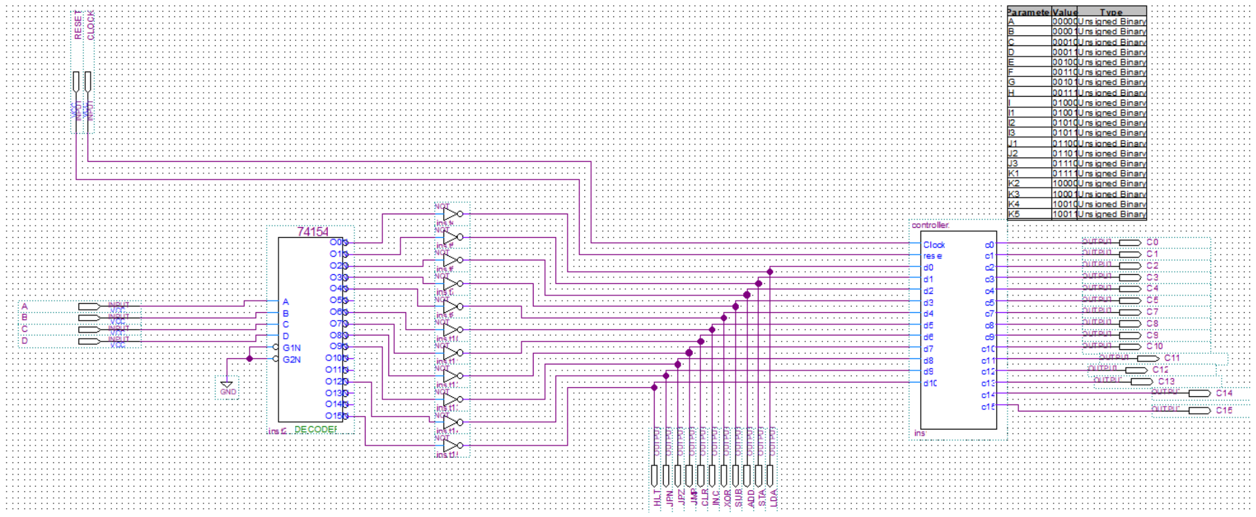
Figure 8 RAM used for part B

2.2. Subsystem descriptions and diagrams

As seen in the system level components, some of them consisted of further components of their own for their accurate functioning. This section describes their composition and functioning.

2.2.1. Instruction decoder for the Control Unit

As we saw earlier, the Control Unit gets the instruction from the instruction register which loads the MDO into the control unit. However, the instruction from the MDO is not exactly what the controller needs to execute. The instruction from the MDO needs to be further decoded into the opcodes (*figure 5*). This is done by the instruction decoder. As seen in *figure 11*, the instruction decoder gets its inputs from the instruction register. The instruction register is a 4 to 16 bit decoder, and for that purpose we use a 74154 IC as an instruction decoder. The output pins of the 74154 IC are connected to NOT so that the opcode could be Active-High instead of Active-Low.



Parameter	Value	Type
A	00000	Unsigned Binary
B	00001	Unsigned Binary
C	00010	Unsigned Binary
D	00011	Unsigned Binary
E	00100	Unsigned Binary
F	00110	Unsigned Binary
G	00101	Unsigned Binary
H	00111	Unsigned Binary
I	01000	Unsigned Binary
I1	01001	Unsigned Binary
I2	01010	Unsigned Binary
I3	01011	Unsigned Binary
J1	01100	Unsigned Binary
J2	01101	Unsigned Binary
J3	01110	Unsigned Binary
K1	01111	Unsigned Binary
K2	10000	Unsigned Binary
K3	10001	Unsigned Binary
K4	10010	Unsigned Binary
K5	10011	Unsigned Binary

Figure 10 Controller with the Instruction Decoder

2.2.2. Accumulator (ACC) Components

As seen earlier, the Accumulator is a vital component in our TRISC. It collects the data from both the MDO bus and the ALU and loads data into the MDI bus according to the control signals received. The figure below shows how an accumulator is designed.

It uses a 74157 IC multiplexer to select between two different sets of inputs A's and B's respectively. Based on the signal on the *Selector* input, it selects one of those data and the output is connected to the input of a 74193-binary counter. So, when the accumulator is loaded, it loads the data at the input pins of that counter to the MDI bus. This is how an accumulator is constructed and how it works.

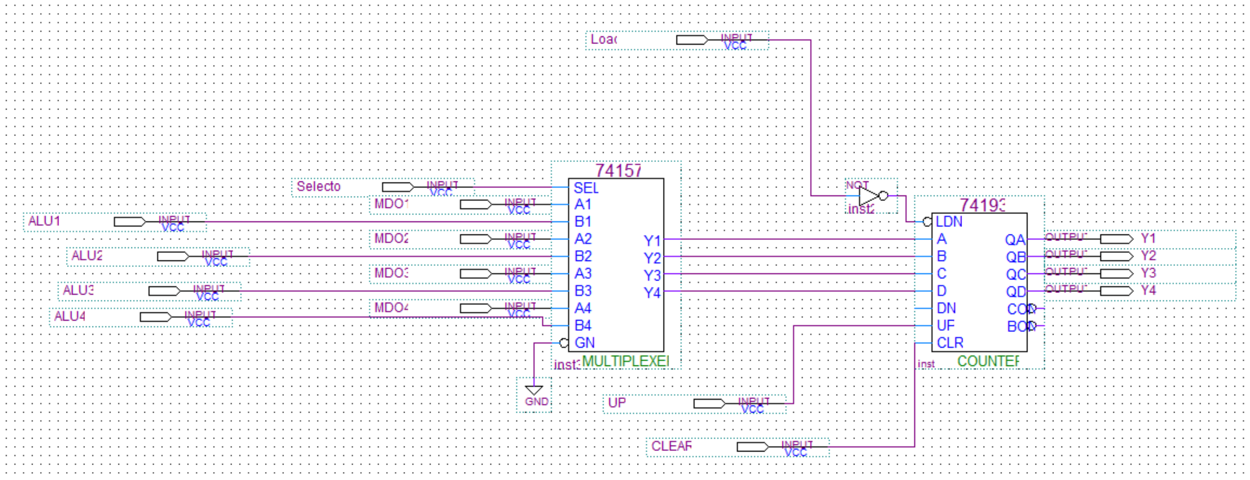


Figure 11 Accumulator (ACC) in detail

2.2.3. Flag Register

It is an intermediate component between the ALU and the accumulator. Its primary function is to load the output of the ALU to the accumulator which is then used for further processing. For that, we again use a 74175 IC as the flag register. It is clocked by control signal 15 (CB[15]).

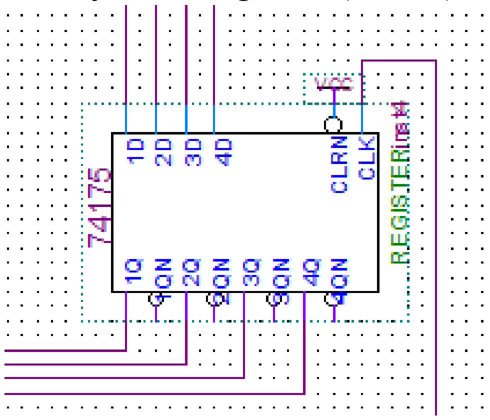


Figure 12 Flag Register

2.3. Hierarchical Design Structure

The schematic block diagram of the fully functional TRISC (after completion of part B)

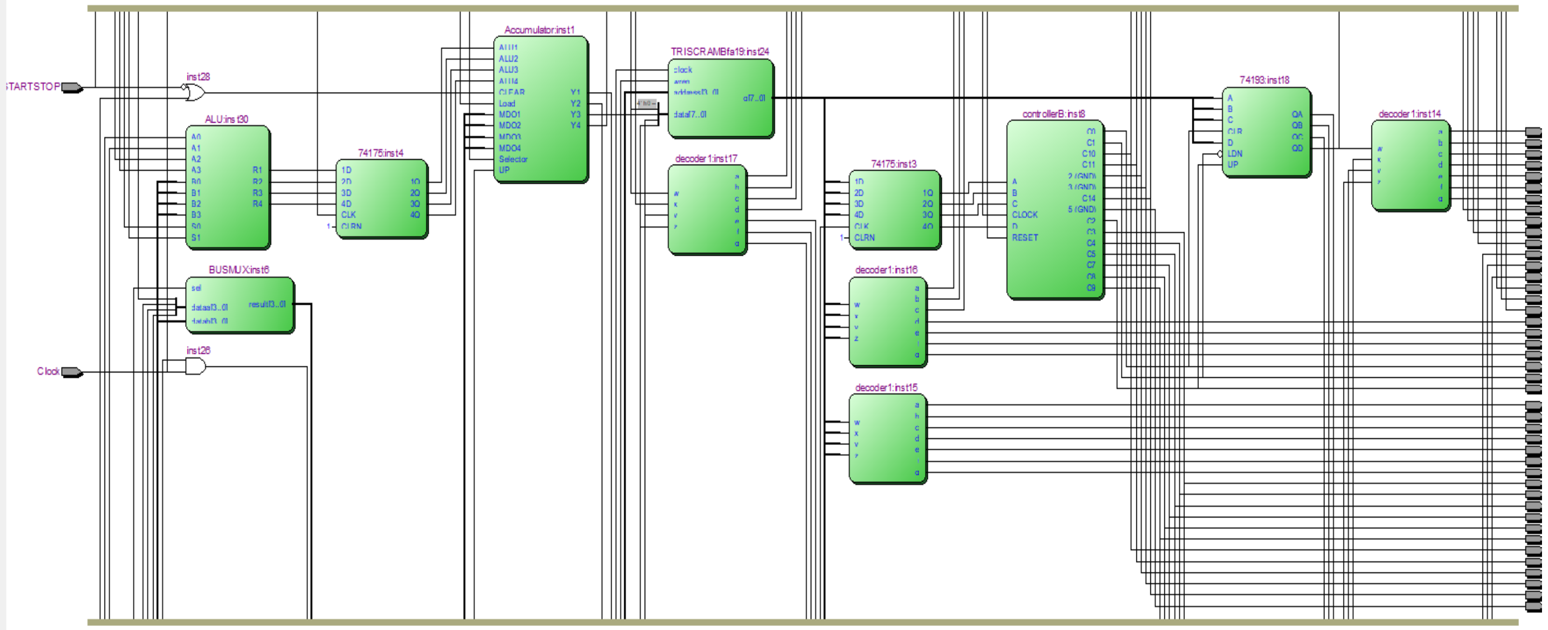


Figure 13 Schematic of the fully functional TRISC

is shown in the figure.

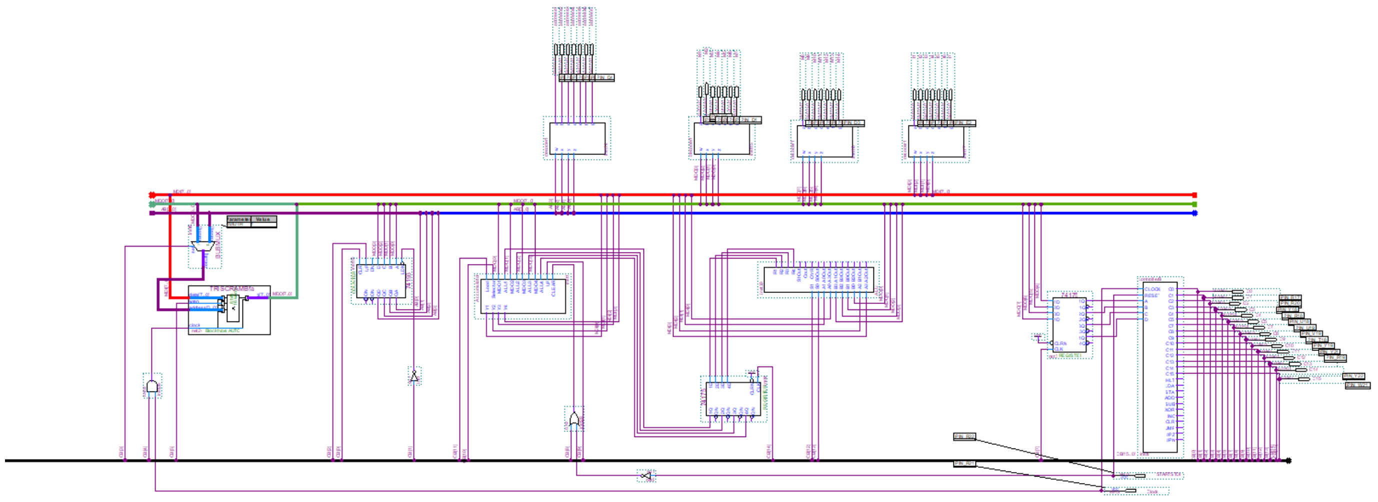
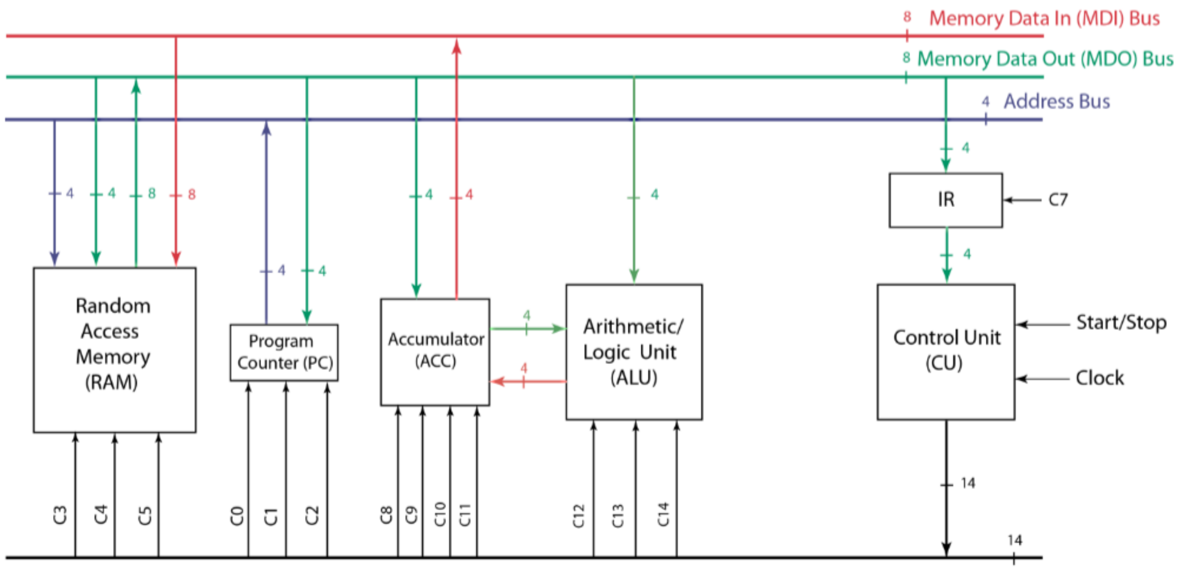


Figure 14 Block Diagram implemented on DE1 using QUARTUS II

2.4. Operating procedure

To understand the operating procedure of the TRISC, let us again refer to *figure 1*.



When the system is clocked, the control unit is clocked. So, whatever data is present in MDO is loaded by IR into the control unit. The instruction decoder in the control unit further decodes that data into the opcodes for the controller to process. Then, the controller outputs a control signal which then controls what the other components in the interface do. For instance, if the signal is C1, the program counter is clocked. So, it increments the data from the MDO and loads it into the address bus. Thus, we now move to the next memory location. Similarly, control signal C8, C9, C10, and C11 decide the functioning of the accumulator. Control signals C3, C4, C5 control the functioning of the RAM, and the control signals C12, C13, C14 control the functioning of the ALU.
Also, we have four seven-segment displays connected to the buses. From the left, the first display shows the memory location, second and third displays are for the data in MDO bus, and the fourth one (the one on the right) displays the data in the MDI bus.

3. Controller Design Details

3.1. Functional Description and diagram showing I/O

The block diagram of the controller used in the design can be seen in the figure below. The input pins A, B, C, and D are the input pins for the instruction decoder in the control unit which will then decode the instruction into opcodes and load it into the controller to get the appropriate control signals. Those control signals are output at the pins C0-C15 respectively. Other remaining output pins were installed for debugging purpose. This way, it makes it easier to detect whether the instruction decoder is sending correct opcodes to the controller.

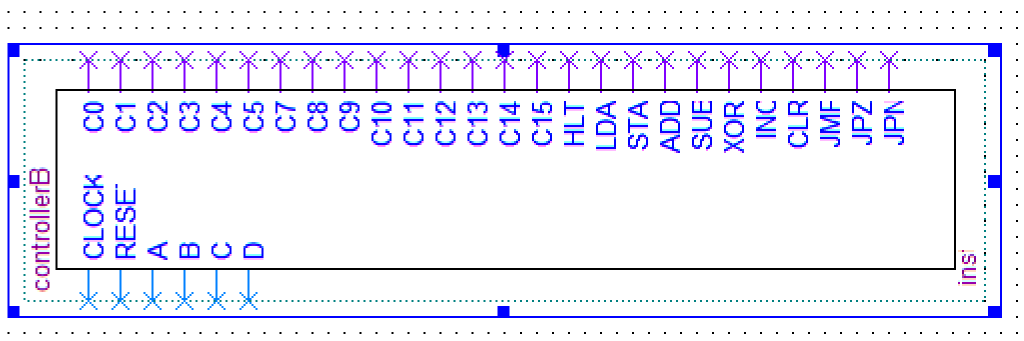


Figure 15 Controller showing input and output pins

3.2. State Diagrams

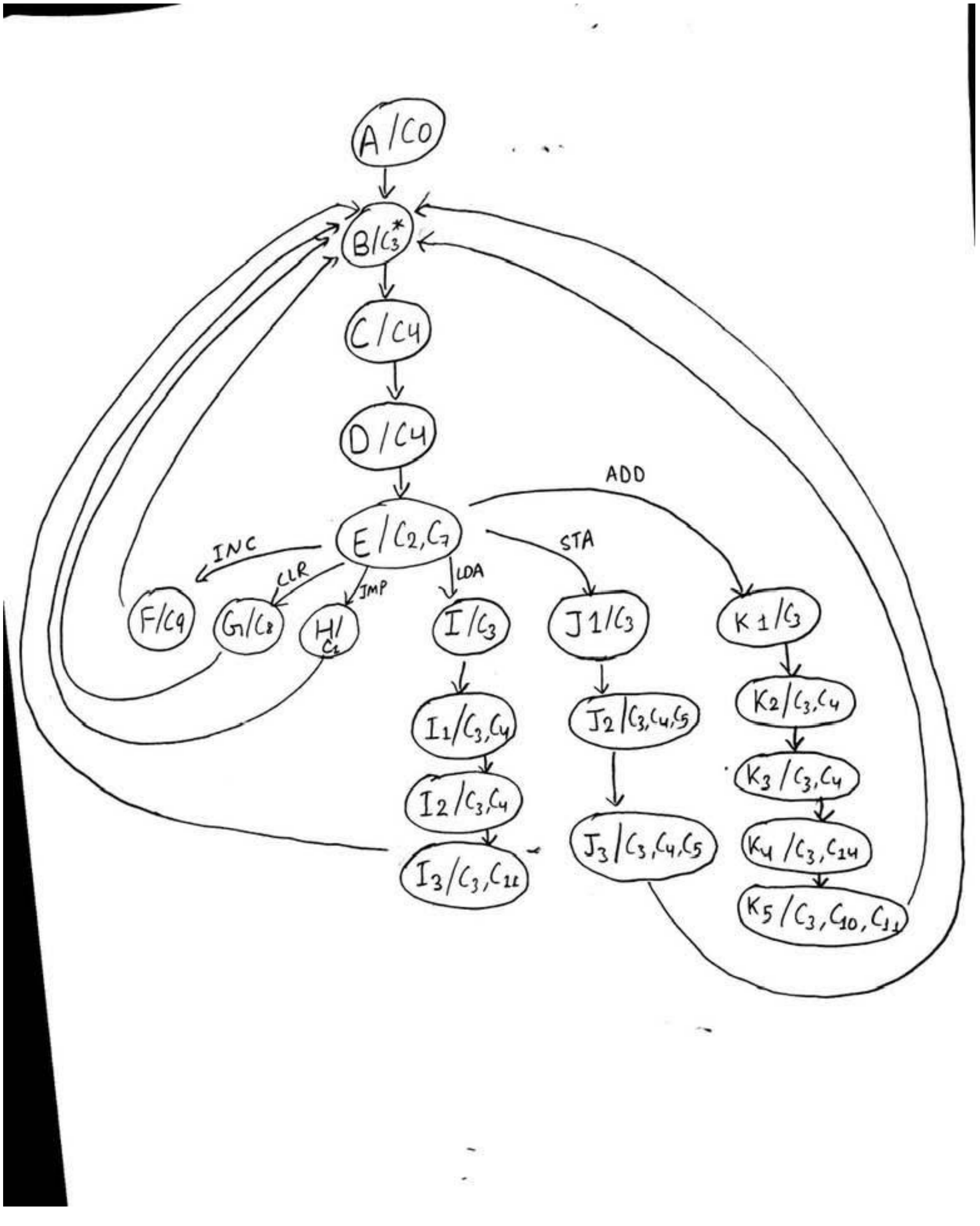
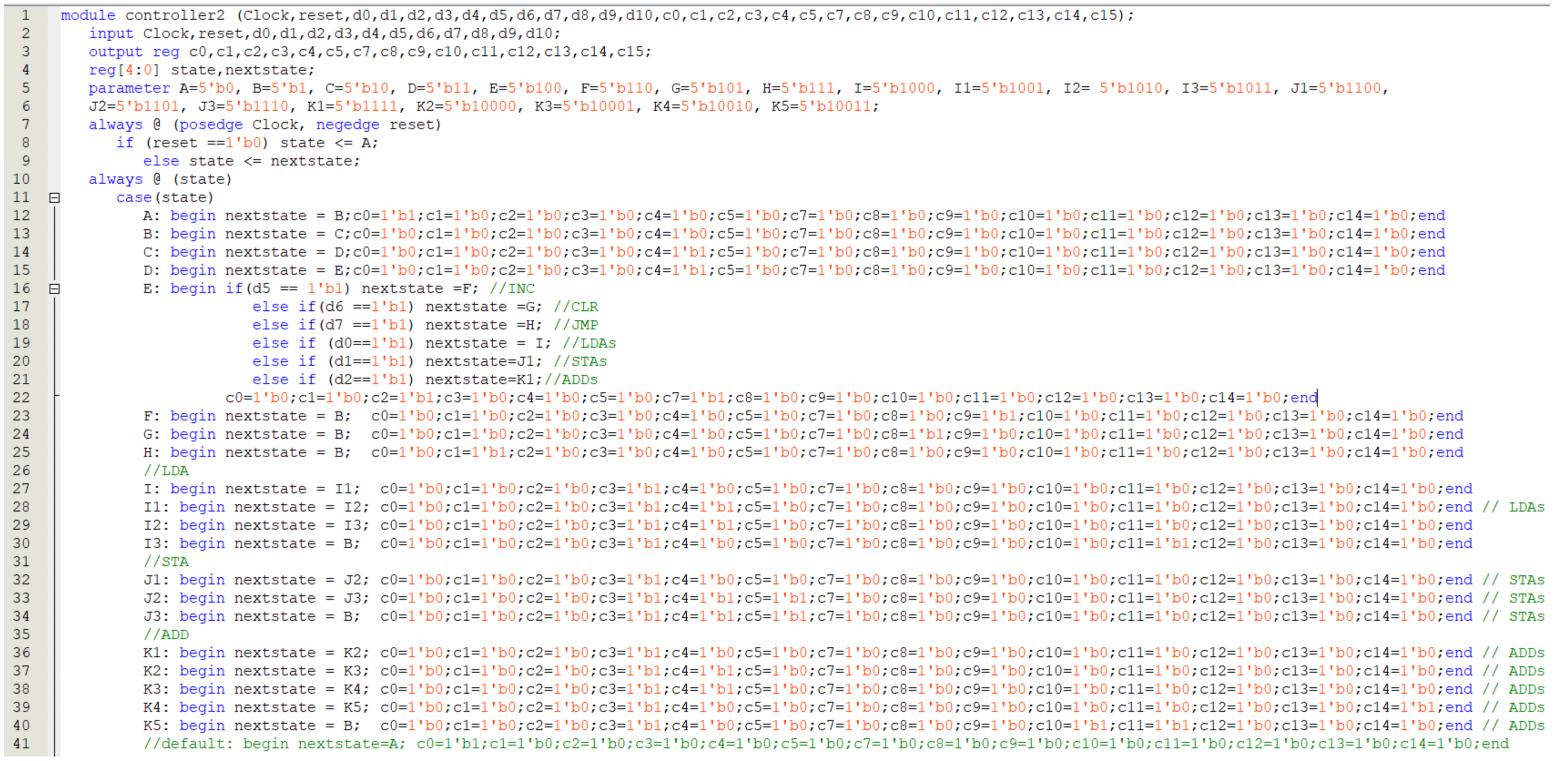


Figure 16 State Diagram of the Controller FSM

3.3. Schematic Diagrams and/or Verilog code

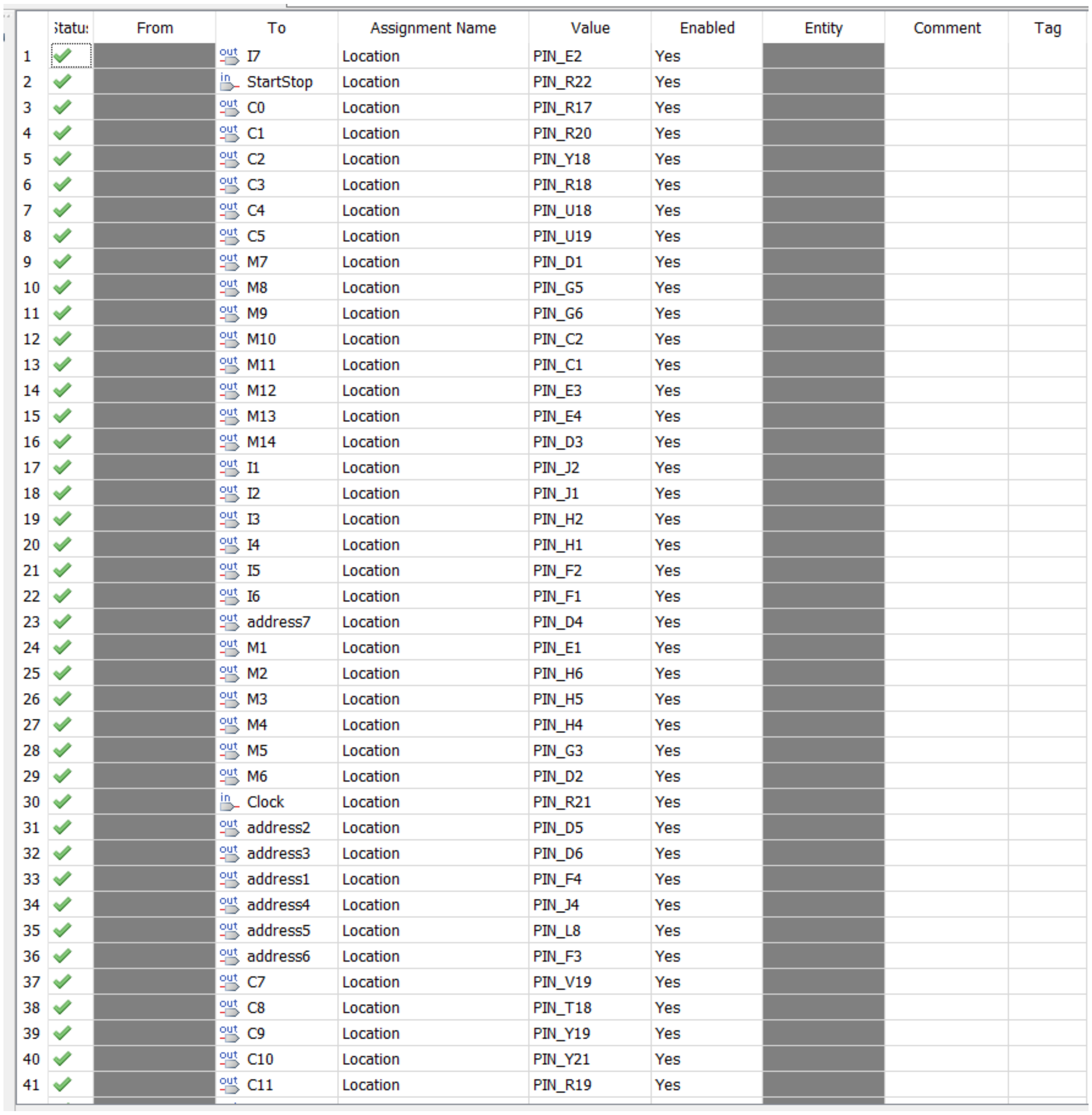


```verilog
module controller2 (Clock,reset,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,c0,c1,c2,c3,c4,c5,c7,c8,c9,c10,c11,c12,c13,c14,c15);
   input Clock,reset,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10;
   output reg c0,c1,c2,c3,c4,c5,c7,c8,c9,c10,c11,c12,c13,c14,c15;
   reg[4:0] state,nextstate;
   parameter A=5'b0, B=5'b1, C=5'b10, D=5'b11, E=5'b100, F=5'b110, G=5'b101, H=5'b111, I=5'b1000, I1=5'b1001, I2= 5'b1010, I3=5'b1011, J1=5'b1100,
   J2=5'b1101, J3=5'b1110, K1=5'b1111, K2=5'b10000, K3=5'b10001, K4=5'b10010, K5=5'b10011;
   always @ (posedge Clock, negedge reset)
      if (reset ==1'b0) state <= A;
         else state <= nextstate;
   always @ (state)
      case(state)
         A: begin nextstate = B;c0=1'b1;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         B: begin nextstate = C;c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         C: begin nextstate = D;c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b1;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         D: begin nextstate = E;c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b1;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         E: begin if(d5 == 1'b1) nextstate =F; //INC
                     else if(d6 ==1'b1) nextstate =G; //CLR
                     else if(d7 ==1'b1) nextstate =H; //JMP
                     else if (d0==1'b1) nextstate = I; //LDAs
                     else if (d1==1'b1) nextstate=J1; //STAs
                     else if (d2==1'b1) nextstate=K1;//ADDs
                  c0=1'b0;c1=1'b0;c2=1'b1;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b1;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         F: begin nextstate = B;  c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b1;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         G: begin nextstate = B;  c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b1;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         H: begin nextstate = B;  c0=1'b0;c1=1'b1;c2=1'b0;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         //LDA
         I: begin nextstate = I1;  c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         I1: begin nextstate = I2; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b1;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // LDAs
         I2: begin nextstate = I3; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b1;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
         I3: begin nextstate = B;  c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b1;c12=1'b0;c13=1'b0;c14=1'b0;end
         //STA
         J1: begin nextstate = J2; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // STAs
         J2: begin nextstate = J3; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b1;c5=1'b1;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // STAs
         J3: begin nextstate = B;  c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b1;c5=1'b1;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // STAs
         //ADD
         K1: begin nextstate = K2; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // ADDs
         K2: begin nextstate = K3; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b1;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // ADDs
         K3: begin nextstate = K4; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b1;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end // ADDs
         K4: begin nextstate = K5; c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b1;end // ADDs
         K5: begin nextstate = B;  c0=1'b0;c1=1'b0;c2=1'b0;c3=1'b1;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b1;c11=1'b1;c12=1'b0;c13=1'b0;c14=1'b0;end // ADDs
         //default: begin nextstate=A; c0=1'b1;c1=1'b0;c2=1'b0;c3=1'b0;c4=1'b0;c5=1'b0;c7=1'b0;c8=1'b0;c9=1'b0;c10=1'b0;c11=1'b0;c12=1'b0;c13=1'b0;c14=1'b0;end
```

Figure 17 Controller Verilog Code

3.4. DE1 pin assignments

The following pin assignments were used for DE1.



	Statu:	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		I7	Location	PIN_E2	Yes			
2	✓		StartStop	Location	PIN_R22	Yes			
3	✓		C0	Location	PIN_R17	Yes			
4	✓		C1	Location	PIN_R20	Yes			
5	✓		C2	Location	PIN_Y18	Yes			
6	✓		C3	Location	PIN_R18	Yes			
7	✓		C4	Location	PIN_U18	Yes			
8	✓		C5	Location	PIN_U19	Yes			
9	✓		M7	Location	PIN_D1	Yes			
10	✓		M8	Location	PIN_G5	Yes			
11	✓		M9	Location	PIN_G6	Yes			
12	✓		M10	Location	PIN_C2	Yes			
13	✓		M11	Location	PIN_C1	Yes			
14	✓		M12	Location	PIN_E3	Yes			
15	✓		M13	Location	PIN_E4	Yes			
16	✓		M14	Location	PIN_D3	Yes			
17	✓		I1	Location	PIN_J2	Yes			
18	✓		I2	Location	PIN_J1	Yes			
19	✓		I3	Location	PIN_H2	Yes			
20	✓		I4	Location	PIN_H1	Yes			
21	✓		I5	Location	PIN_F2	Yes			
22	✓		I6	Location	PIN_F1	Yes			
23	✓		address7	Location	PIN_D4	Yes			
24	✓		M1	Location	PIN_E1	Yes			
25	✓		M2	Location	PIN_H6	Yes			
26	✓		M3	Location	PIN_H5	Yes			
27	✓		M4	Location	PIN_H4	Yes			
28	✓		M5	Location	PIN_G3	Yes			
29	✓		M6	Location	PIN_D2	Yes			
30	✓		Clock	Location	PIN_R21	Yes			
31	✓		address2	Location	PIN_D5	Yes			
32	✓		address3	Location	PIN_D6	Yes			
33	✓		address1	Location	PIN_F4	Yes			
34	✓		address4	Location	PIN_J4	Yes			
35	✓		address5	Location	PIN_L8	Yes			
36	✓		address6	Location	PIN_F3	Yes			
37	✓		C7	Location	PIN_V19	Yes			
38	✓		C8	Location	PIN_T18	Yes			
39	✓		C9	Location	PIN_Y19	Yes			
40	✓		C10	Location	PIN_Y21	Yes			
41	✓		C11	Location	PIN_R19	Yes			

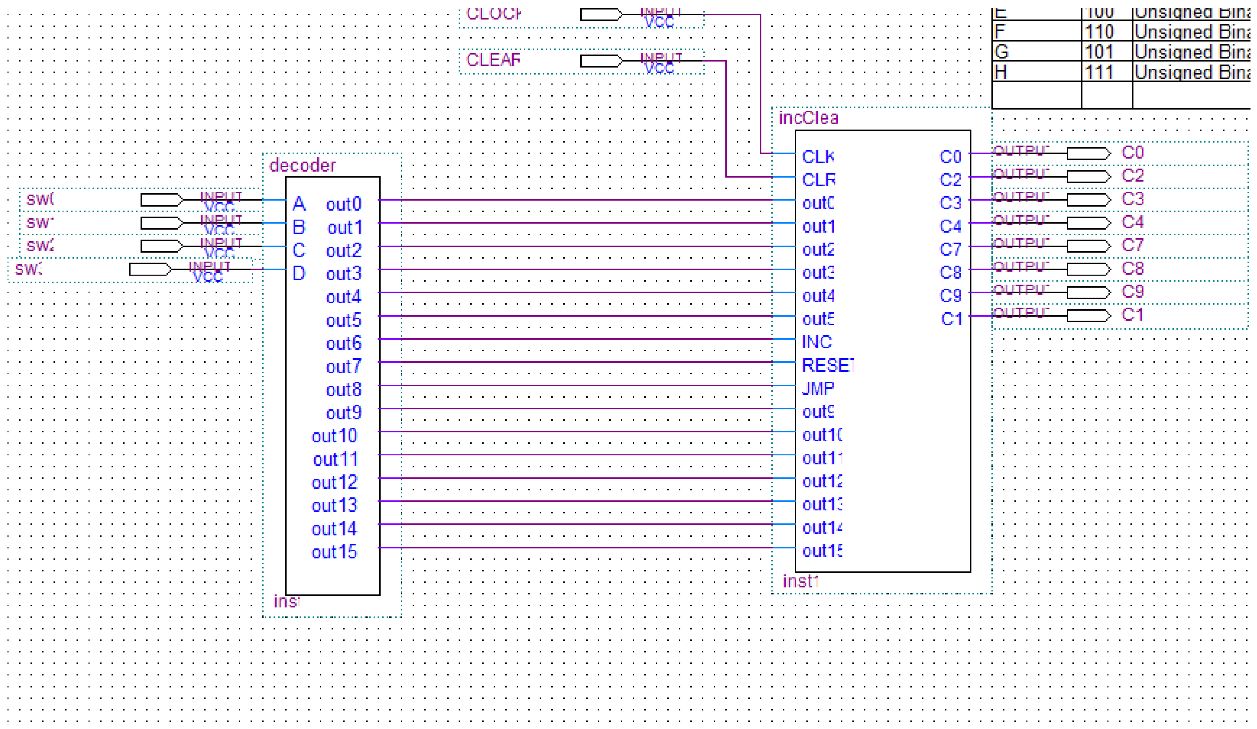
Figure 18 DE1 Pin Assignments

4. Alternative design considerations

4.1. Alternatives considered

When I tried to do things differently, the results were not as expected. So, I had to switch back to the conventional ways, exactly the way the project instructions said. However, lets have a look at some of those cases.

- **Tried to use active low instruction decoder for control unit**

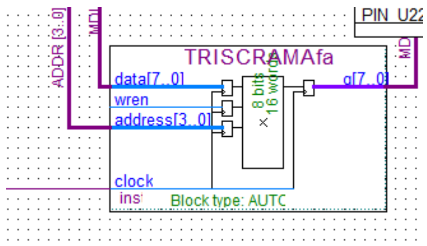


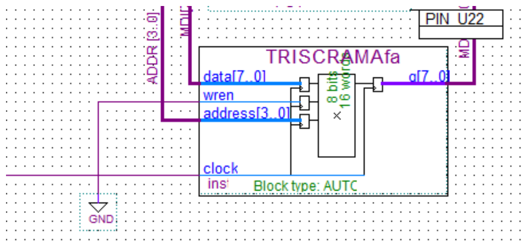
E	100	Unsigned Bina
F	110	Unsigned Bina
G	101	Unsigned Bina
H	111	Unsigned Bina

Figure 19 Alternative Design used for controller

This certainly did not give out the correct output so it had to be changed to active high.

- **WriteEnable was not grounded**
  For Part A, the write enable of the RAM was not grounded at first. However, while debugging, I found the issue and corrected it, thus getting the desired output. We can see it in the images below.





5. Integration and Test Plan

5.1. Integration strategy and Test strategy

As seen in *figure 19*, the pin assignments were assigned to the system clock and clear inputs, the control signal LEDs and the four 7-segment displays. System clock was assigned to Key1 on the DE1 board, and it was used to clock the TRISC in order to move between memory addresses and performing the respective tasks as instructed by the opcodes at the particular control signals.

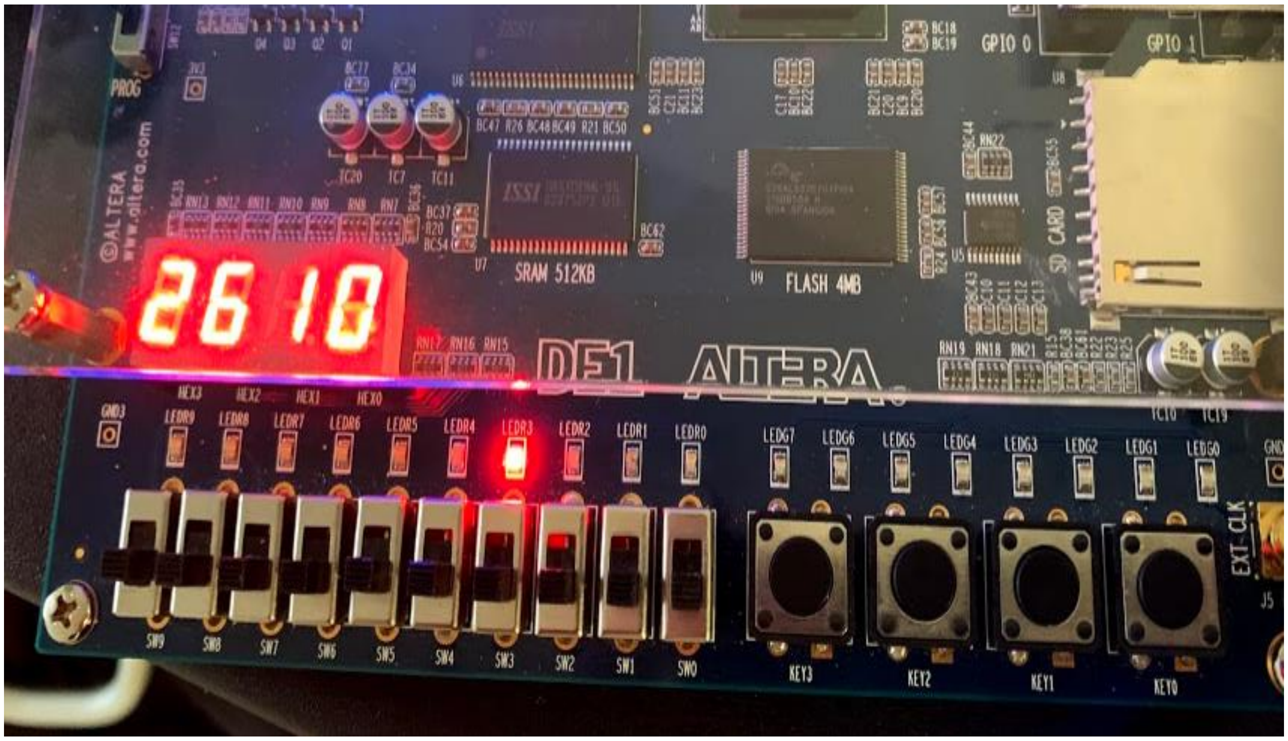


Figure 20 Implementation on the DE1 board (I)

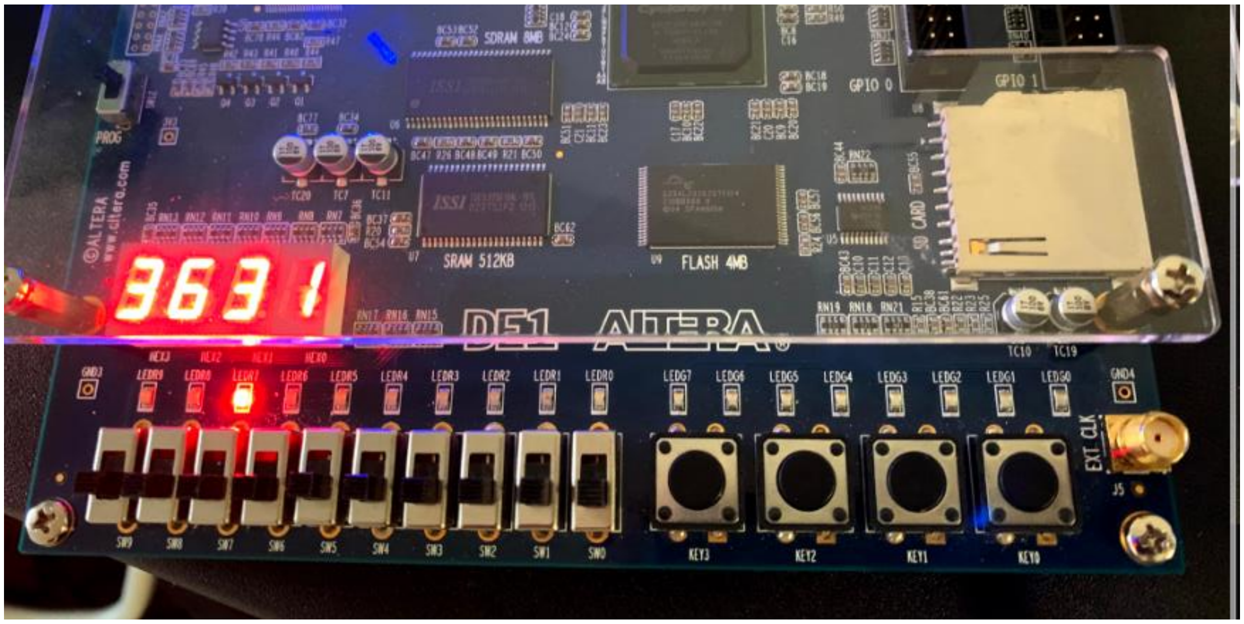


Figure 21Implementation on the DE1 (II)

6. Conclusion

6.1. Resolution of design and/or implementation issues

Any design issues occurred were addressed by calm debugging with substantial assistance from the TA and professor himself. For instance, there was a case when I was using the bits in reversed order in my ALU. I was sending the bits in reversed order into the ALU and thus it did not produce the accurate outputs. Also, there was another instance, when the Address Bus was cleared at each clock. It was because I forgot to insert a NOT gate for the active low clear, due to which it was clearing itself without any signal. Such, minor to major issues were tackled by constant work and referring to the TA and professor after completion of each single step.

6.2. Lessons Learned

By the time the project was completed, I found myself more apt in working with Verilog, working with finite state machines and designing them using Verilog code on the Quartus II. Designing block diagram files and then saving, compiling and creating their symbol files used to be quite difficult task in the previous lab sessions. However, after the completion of the project such things would not worry me anymore. This was not the case when we were using Quartus II for the first or second times during Lab session 2 and 3. Also, all the previous labs were so delicately used in compiling the project as a who which emphasizes on the fact that consistent work should be done at every point of time in order to achieve something significant. For example, Lab 4 dealt with simple instruction decoders. We were asked to design four different kinds of decoders and the lab was an easy task to complete. However, had it not been done properly, the project would have been left without a 4 bit to 7-segment decoder. Therefore, the positives after being able to successfully design and implement the project are certainly there to be realized.