Dijesh Pradhan|1001516650

CSE 2441-001

TERM DESIGN PROJECT

(Executing INC, CLR, JMP, LDA, STA, ADD instructions)

Project Completed: 2nd May 2019

Submitted on:3rd May 2019

Submitted by: Dijesh Pradhan
ID:1001516650
Submitted to: Dr. Bill D. Carroll

TABLE OF CONTENTS

LIST OF FIGURES

• Introduction

○ Project Overview

Various different techniques have been applied to teach us about a particular subject but the most effective one is doing something practically. Learning effectiveness of advanced engineering problems is lot easier with practical experience. This project is a mandatory project to complete the course, CSE 2441. This course is the introduction to Digital Logic which is required by all the electrical engineering and computer engineering programs in University of Texas at Arlington. This project has been selected as one of the sources of the documentation that students can design a system and components of a system. In this project we are required to execute programs consisting of INC, CLR, JMP, LDA, STA and ADD instructions. This project is divided in two different parts- Part A and Part B. In Part A we integrate the components of TRISC Ram in which we execute programs consisting only of INC, CLR and JMP instructions. In Part B we realize a TRISC ram in which we execute all the INC, CLR, JMP, LDA, STA and ADD instructions. To perform this project, we use a software named Quartus.

○ Project Status

The project was completed within the given time period and was demoed on May 2, 2019. I realized there was no unresolved problems, but I would like to develop a much more better design which could execute more instructions.

○ Report Overview

In this report, I will go through all the details and the parts that was used to design the processor. I would give a brief explanation about the controller, ALU and Accumulator used in this project.

• System Design

o System-level description and diagrams

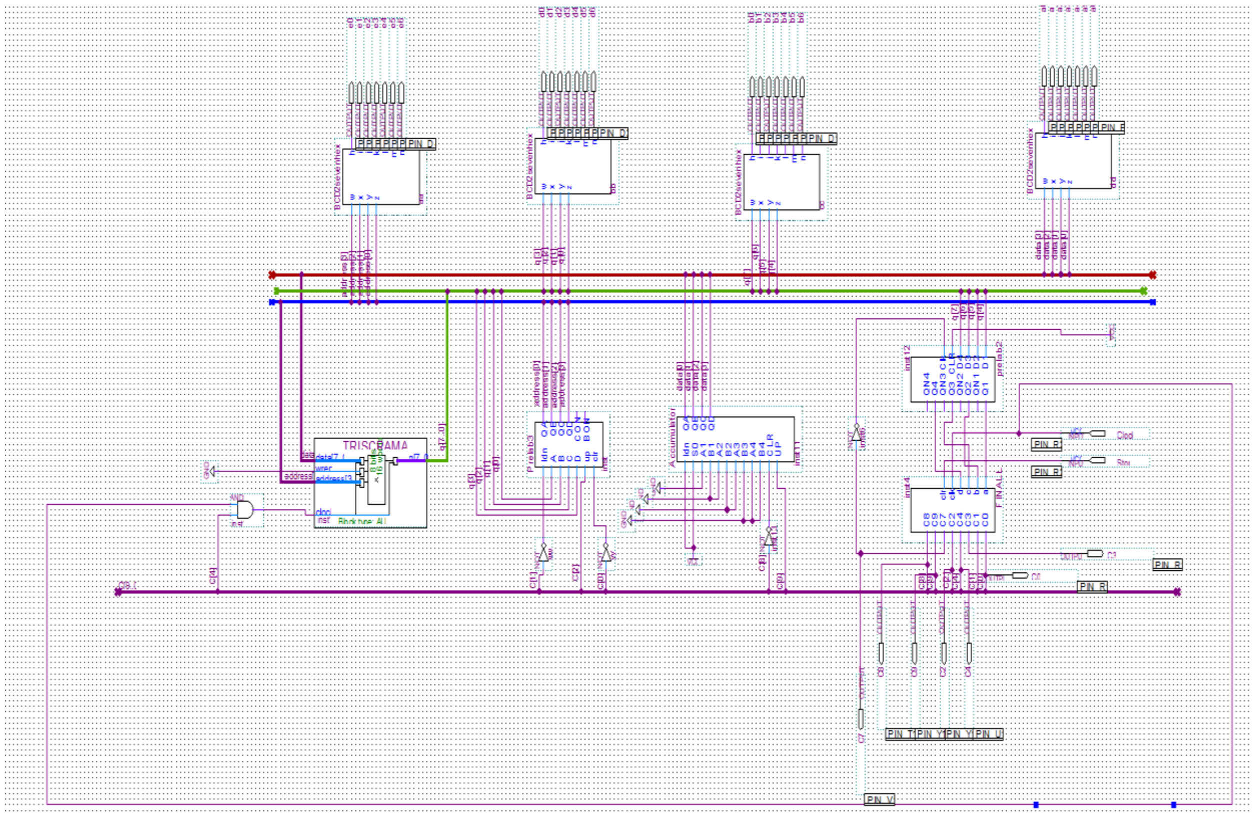


Figure 1- Part A Project

The processor above gives us the required result for the project. It has all the components which helps in reach the required execution instructions. At first, we included the TRISC RAM Part A which was provided us to use to perform the execution. We extracted the instructions from the TRISC RAM and passed to the Memory Data Out (MDO) Bus. From this bus the instructions are loaded to instruction register. From there the instructions are passed to the Control Unit, from where the control signals are passed for various operations. Then the control signals are passed to their respective execution units. The Clear Accumulator (C8) and the Load Instruction Register (C7) are passed to the accumulator, the (C0), (C1) and (C2) are passed to the Program Counter and the (C4) is connected to the RAM. These are passed using a Control Bus. These are requirements and the system- level description for the project's PART A.

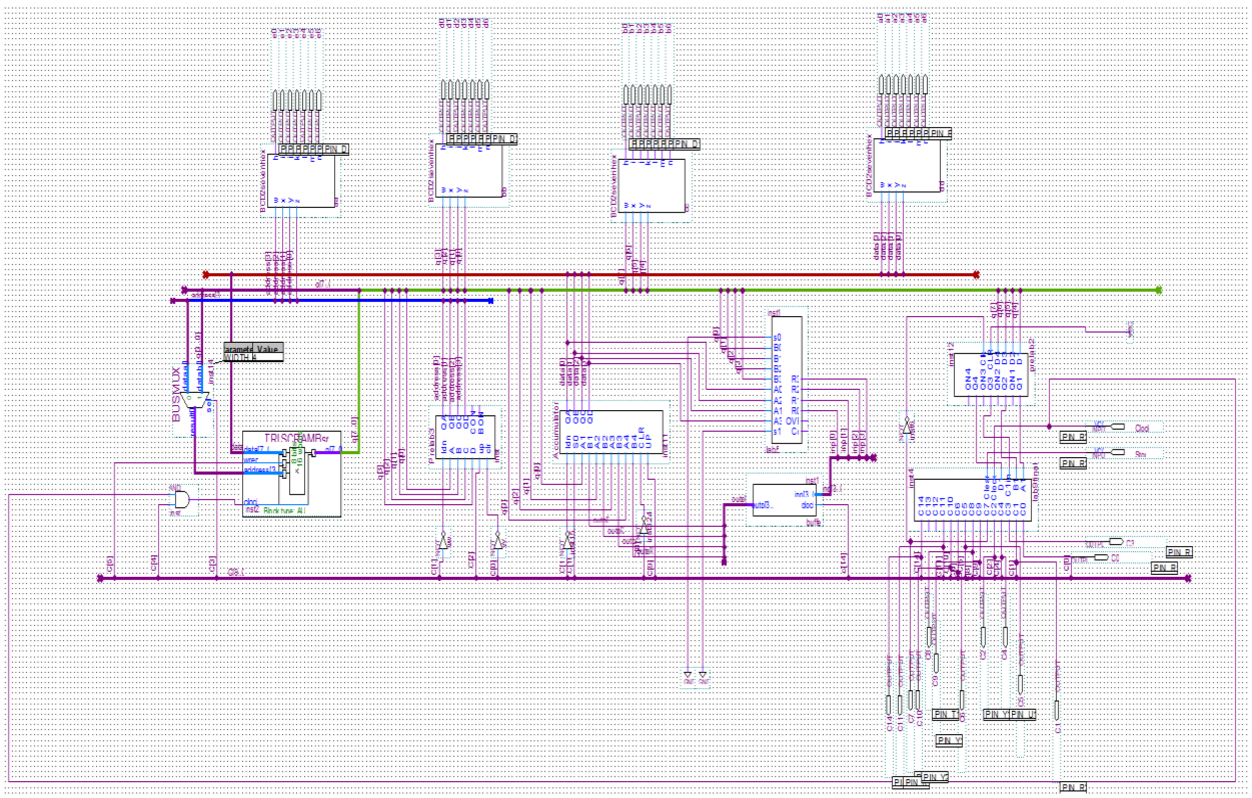


Figure 2 – Part B Project

The above diagram is the block diagram of the PART B of the project. In this part, we use the same diagram of PART A and improvise in the diagram. The control unit from the part A was changed and was improvised so that it can perform the extra execution orders that is LDA, STA and ADD instructions. From the control unit the control signals C12, C13 and C14 are passed in the ALU of the diagram and two other control signals C10 and C11 are passed in the Accumulator. The three instructions are passed in the RAM that is C3, C4 and C5.

○ Subsystem description and diagrams

⇒ TRISC RAM

The TRISC RAM is given to us so that we can execute the instructions from it.

⇒ Parallel in/ Parallel Out (PIPO)

PIPO is a register in which the input is stored for one clock cycle and the C7 instruction is passed as clock in the instruction register.

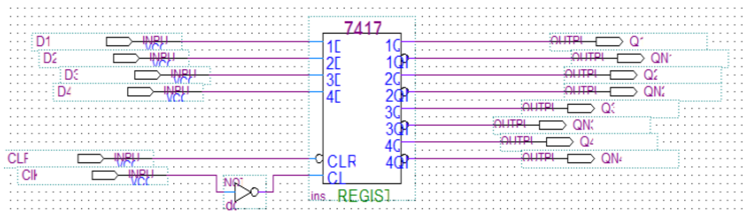


Figure 3- PIPO

⇒ Binary to Hexadecimal

It is a unit which is executed using a Verilog code which changes the binary instructions into hexadecimal instructions. This unit is used to output the LED signals in the DE1 board.

```
module BCD2sevenhex (
input w,x,y,z,
output reg h,i,j,k,l,m,n);
always @ (w,x,y,z) begin
case ({w,x,y,z})
 4'b0000: {h,i,j,k,l,m,n} = 7'b0000001; //0
 4'b0001: {h,i,j,k,l,m,n} = 7'b1001111; //1
 4'b0010: {h,i,j,k,l,m,n} = 7'b0010010; //2
 4'b0011: {h,i,j,k,l,m,n} = 7'b0000110; //3
 4'b0100: {h,i,j,k,l,m,n} = 7'b1001100; //4
 4'b0101: {h,i,j,k,l,m,n} = 7'b0100100; //5
 4'b0110: {h,i,j,k,l,m,n} = 7'b0100000; //6
 4'b0111: {h,i,j,k,l,m,n} = 7'b0001111; //7
 4'b1000: {h,i,j,k,l,m,n} = 7'b0000000; //8
 4'b1001: {h,i,j,k,l,m,n} = 7'b0001100; //9
 4'b1010: {h,i,j,k,l,m,n} = 7'b0001000; //A
 4'b1011: {h,i,j,k,l,m,n} = 7'b1100000; //B
 4'b1100: {h,i,j,k,l,m,n} = 7'b0110001; //C
 4'b1101: {h,i,j,k,l,m,n} = 7'b1000010; //D
 4'b1110: {h,i,j,k,l,m,n} = 7'b0110000; //E
 4'b1111: {h,i,j,k,l,m,n} = 7'b0111000; //F
endcase
 end
endmodule
```

⇒ Program Counter

Program Counter is used to locate where the program is being executed. In this unit C0, C1 and C2 are passed.

⇒ Accumulator

In this unit two set of data are passed, one from the ALU and one from the program unit. The accumulator chooses one of the data and executes the required operations. Then the instruction is passed to the Binary to Hexadecimal unit and is displayed to the DE1 board.

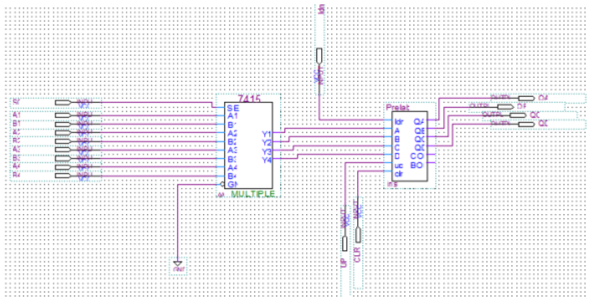


Figure 4- Accumulator

⇒ ALU

ALU stands for Arithmetic Logic Unit. A set of data is passed to this unit from the Accumulator and the unit performs add, subtract, XOR and AND operations. This instruction is then passed to the buffer which acts like a PIPO where the data is stored for one complete cycle. After one complete cycle the instruction is then passed to the Accumulator.

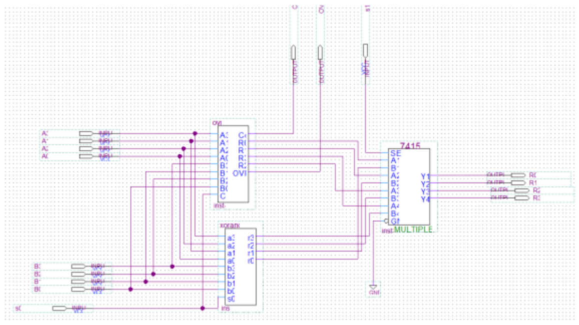


Figure 5- ALU

- Hierarchical design structure

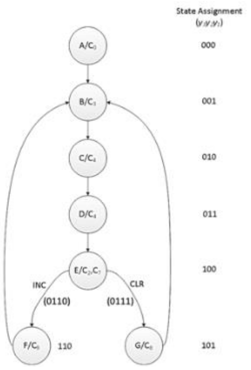
  This project is based on the concept of hierarchy. The course was started with the basic lab works in which we performed the basic logic operations and designed the block diagrams in which we could perform simple binary additions. Then the labs were tilted to the deeper parts in which we created the ALU in Lab 5 which is used in this project as well. We also then came to get the knowledge of the control unit in Lab 9 which is also used to improvise the design of the project and the control unit is the most important part of the design which is changed in Part A and Part B of the project. We used program counter and PIPO in other labs which is also a very important part of this project. In these small steps I was able to combine all of them and perform the project.

- ## Controller design details

  - ### Functional description

The Controller is the main part of the project which is changed twice. At first the Verilog code is designed in such a way that it performs the three instructions INC, CLR and JMP instructions for Part A and the Verilog of the controller is changed to execute the extra instructions as well which is LDA, STA and ADD. The instructions is passed from the instruction register to the Decoder which is then passed to the control Unit, where the program runs and the Control Unit sends out the correct control signals. For example when the instruction CLR is selected then the control signal C0 is passed from the Control Unit.

- State diagrams



- DE1 pin assignments

The control signals passed are assigned from LEDR9-LEDR0 and LEDG7-LEDG4(C0-C14). Key0 and Key1 are assigned for Clear and Clock respectively.

- Verilog Code

```
module controller(
        input a,b,c,d,e,f,g,h,i,j,k,clock,clear,
        output reg C0,C1,C3,C4,C2,C7,C9,C8,C5,C6,C10,C11,C12,C13,C14);
        reg[4:0] state,nextstate;
        parameter
A=5'b00000,B=5'b00001,C=5'b00010,D=5'b00011,E=5'b00100,F=5'b00101,G=5'b00110,H=5'b00111,I=5'b01000,J=5'b01001,K=5'b01010,L=5'b01
100,M=5'b01101,N=5'b01110,O=5'b01111,P=5'b10000,Q=5'b10001,R=5'b10010,S=5'b10110,T=5'b11000;
        always@(posedge clock, negedge clear)
                if(clear==0) state<=A;
                        else state<= nextstate;
        always@(state,a,b,c,d,e,f,g,h,i,j,k)
                case(state)
                        A: nextstate<=B;
                        B: nextstate<=C;
                        C: nextstate<=D;
                        D: nextstate<=E;
                        E: if(f) nextstate<=F; else if(g) nextstate<=G; else if(h) nextstate<=H; else if(a) nextstate<=I; else if(b)
nextstate<=M; else if(c) nextstate<=Q; else nextstate <=A;
                        F: nextstate<=B;
                        G: nextstate<=B;
                        H: nextstate<=B;
                        I: nextstate<=J;
                        J: nextstate<=K;
                        K: nextstate<=L;
                        L: nextstate<=B;
                        M:nextstate<=N;
                        N:nextstate<=O;
                        O:nextstate<=P;
                        P:nextstate<=B;
                        Q: nextstate<=R;
                        R: nextstate<=S;
                        S: nextstate<=T;
```

```
                                T: nextstate<=B;
                        endcase
                always@(state)
                        case(state)
                        A:begin C0 = 1; C1=0; C3=0; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        B:begin C0 = 0; C1=0; C3=0; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        C:begin C0 = 0; C1=0; C3=0; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        D:begin C0 = 0; C1=0; C3=0; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        E:begin C0 = 0; C1=0; C3=0; C4=0; C2=1; C7=1; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        F:begin C0 = 0; C1=0; C3=0; C4=0; C2=0; C7=0; C9=1; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        G:begin C0 = 0; C1=0; C3=0; C4=0; C2=0; C7=0; C9=0; C8=1;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        H:begin C0 = 0; C1=1; C3=0; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        I:begin C0 = 0; C1=0; C3=1; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        J:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        K:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        L:begin C0 = 0; C1=0; C3=1; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=1; C12=0; C13=0; C14=0;end
                        M:begin C0 = 0; C1=0; C3=1; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        N:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        O:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        P:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=1; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        Q:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        R:begin C0 = 0; C1=0; C3=1; C4=1; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=0;end
                        S:begin C0 = 0; C1=0; C3=1; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=0; C11=0; C12=0; C13=0; C14=1;end
                        T:begin C0 = 0; C1=0; C3=1; C4=0; C2=0; C7=0; C9=0; C8=0;C5=0; C6=0; C10=1; C11=1; C12=0; C13=0; C14=0;end
                        endcase
endmodule
```

- ## Integration and Test Plan
  - ### Integration Strategy

    For the integration strategy, at first I got all the required bdf and bsf files for the units that are required in the block diagram into one folder and included all the files into the project. At first, I designed all the smaller units and simulated them to make sure they were running properly without any errors.

  - ### Test Strategy

    When I started making the project, I simulated every time I added a smaller unit so that I won't have to find the problem in the end with a lot of the units combined which would have been difficult to figure out the problem. In this way I was able to complete the project in an efficient way.

- ## Conclusion

  In this project I learnt how to build a processor which can execute different instructions. I learnt how to build a project in a hierarchy order. This is my first project in which I programmed a full functioning process for six different instructions. In this project I learnt how a simple processor works.