CND 111: Introduction to Digital Design

FINAL PROJECT

Microprocessor Design

Section #: 16

Submitted by:

Student Name	ID
Aya Reda Osman	V23010305
Fatma Abdelmonsef	V23010243
Hadeer Khaled	V23010623
Hayat Gamal Abdel Hady	V23010201

Submitted to:

- **Dr: Islam Yehia**
- **TA: Mohammed Elshafey**

i. Design Steps:

1. Defining Specification of the microprocessor.
2. Defining Input and output ports of the microprocessor.
3. Defining Control Signals generated from the control unit.
4. Defining the Block Diagram and Programming Guide of the microprocessor.
5. Defining Timing Diagram for each instruction.
6. Writing RTL Code for each sub-block in the block diagram.
7. Integrating all sub-blocks together.
8. Putting Testing Plan to test all instructions of the microprocessor.
9. Generating Design Matrix Reports.

ii. Specification:

- A 8-bit microprocessor with the following operations:
  - **Four** Arithmetic operations (ADD, SUB, MUL and DIV).
  - **Three** Logical operations (AND, OR and XOR).
  - **One** Jump operation (JZ).
- Internal 16x8 ROM to hold instructions and data.
  - Instructions saved in ROM starting from address 0H.

iii. I/O :

- 2 inputs (CLK, CLR).
- 1 Output (Output Register Output).

iv. Control Signals:

- we have **15 control** signals listed as follows:

Control Signal	Function	Active Low/High
CP	Incrementing Program Counter (PC).	HIGH
EP	Enable Program Counter (PC).	HIGH
$\overline{Lm}$	Load Memory Address Register (MAR).	LOW
$\overline{CE}$	Enable reading data from ROM.	LOW
$\overline{LI}$	Load Instruction Register (IR).	LOW
$\overline{EI}$	Enable output from IR.	LOW
$\overline{LA}$	Load Accumulator.	LOW

EA	Enable output from Accumulator.	HIGH
$\overline{LB}$	Load Register B.	LOW
$\overline{LO}$	Load Register OUT.	LOW
Alu_Controller (3 Bits)	3'b000 : No Operation.	HIGH
	3'b001 : Add Operation.	
	3'b010 : Sub Operation.	
	3'b011 : MUL Operation.	
	3'b100 : DIV Operation.	
	3'b101 : AND Operation.	
	3'b110 : OR Operation.	
	3'b111 : XOR Operation.	
HLT_E	Indicating HLT instruction.	HIGH
JmpZ	Indicating Jump Operation.	HIGH

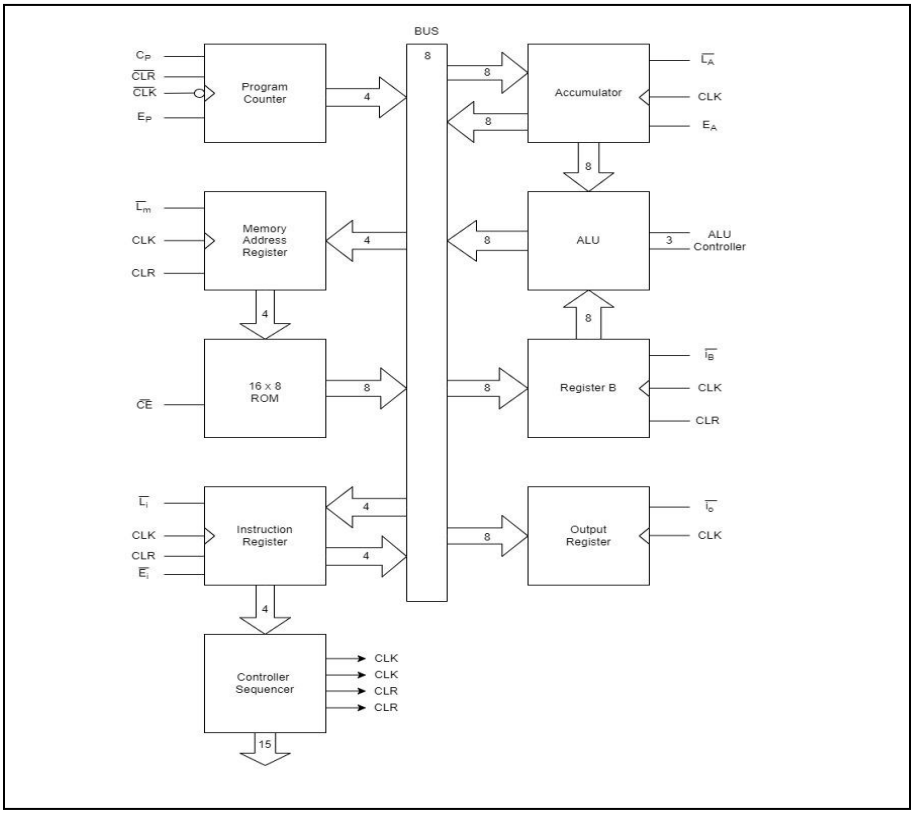
v. Programming Guide:

- **Opcode:**

Opcode	Instruction
0000	ADD
0001	SUB
0010	MUL
0011	DIV
0100	LDA
0101	JZ
0110	AND
0111	OR
1000	XOR
1110	OUT
1111	HTL

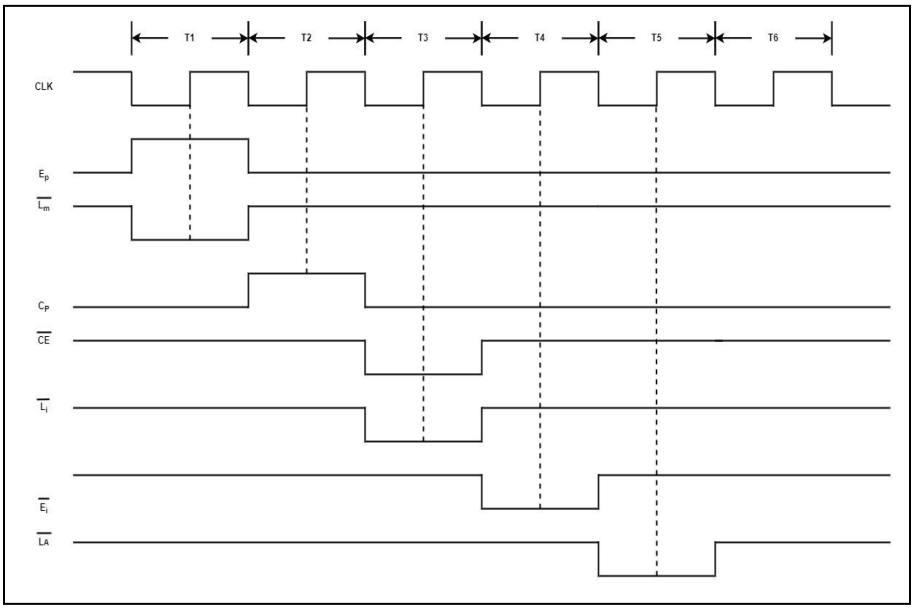
- **Addressing Mode:**
  - we are using direct addressing mode.

vi. Block Diagram:

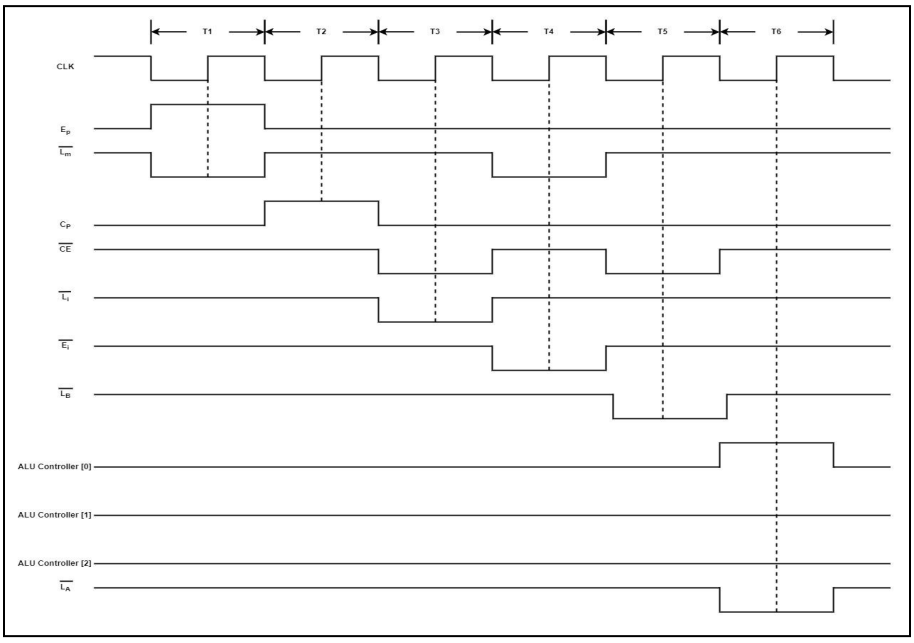


vii. Timing Diagram:

- **Fetch and LDA Instruction Timing Diagram:**

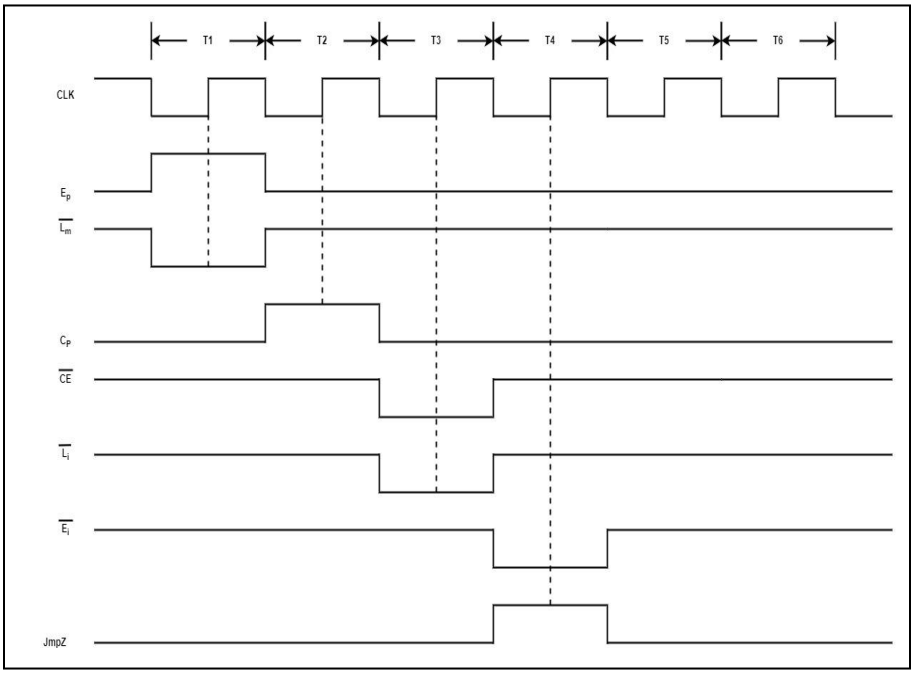


- **Fetch and ADD Instruction Timing Diagram:**

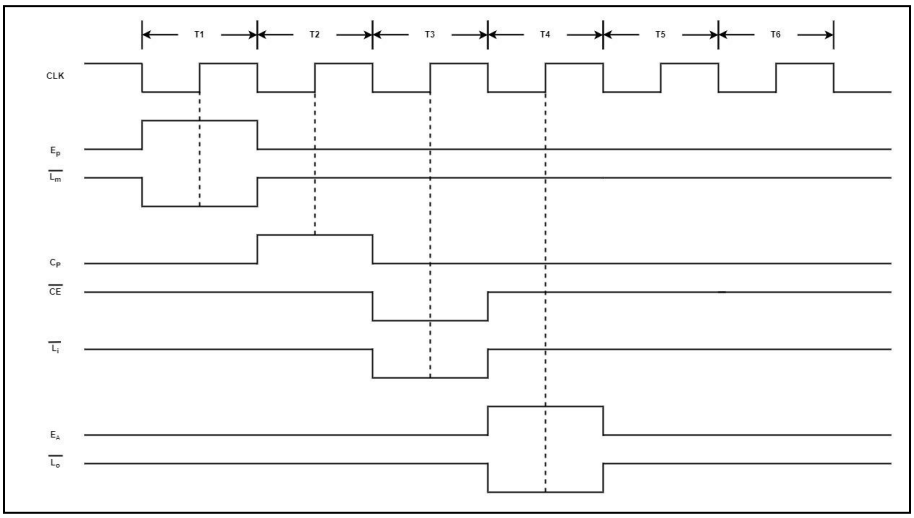


- **Note:**
  - SUB, MUL, DIV, AND, OR, XOR Instructions' Timing diagram is similar to ADD timing diagram but with changing ALU Controller Bits.

- **Fetch and JZ Instruction Timing Diagram:**



- **Fetch and OUT Instruction Timing Diagram:**



viii. RTL Code for each module:

- **Top Module:**

```
module Top_Module (
    input clk,clr,
    output [7:0] topOUT);

    wire CLK_I,CLKn_I;
    wire CP,EP,Lm_n,CE_n,LI_n,EI_n,LA_n,EA,LB_n,LO_n,HLT_E;
    wire [2:0] Alu_Controller;
    wire [3:0] MAR_out,INSR_A,INSR_B;
    wire [7:0] RB_out,ACC_OUT;
    wire [5:0] T;
    wire HLTn_E,j,Z_flag;
    wire [7:0] bus;
    wire [7:0] alu_Out;


    // Controller Sequencer Instantiation
    controller_sequencer CS(.clk(clk),.clr_n(clr),.ins_in(INSR_B),.CP(CP),.EP(EP),.Lm_n(Lm_n),.CE_n(CE_n),
    .LI_n(LI_n),.EI_n(EI_n),.LA_n(LA_n),.EA(EA),.LB_n(LB_n),.LO_n(LO_n),.HLT_E(HLT_E),.j(j),.Alu_Controller(Alu_Controller),.T(T));

    // Clock buffer
    not g0(HLTn_E,HLT_E);
    bufif1 g1(CLK_I,clk,HLTn_E);
    assign CLKn_I = ~CLK_I;

    // Program Counter Instantiation
    PCunit PC(.CLK(CLK_I), .CLR_n(clr), .Cp(CP), .Ep(EP),.PC_out(bus[3:0]),.zero_flag(Z_flag),.j(j),.Jumb_In(bus[3:0]));

    // Memory Address Register Instantiation
    MAR_Processor MAR(.clk(CLK_I),.clr(clr),.Lm_n(Lm_n),.inAddr(bus[3:0]),.outAddr(MAR_out));

    // RAM Instantiation
    ram RAM(.CE_n(CE_n),.address(MAR_out),.data_out(bus));

     // Instruction Register Instantiation
     insReg INSREG(.clk(CLK_I),.LI_n(LI_n),.EI_n(EI_n),.clr(clr),.w(bus),.a(INSR_A),.b(INSR_B));
     // TRI Buffer to connect InsReg output to BUS
     tribuf_4bit bufb (.in(INSR_A), .out(bus[3:0]), .low_enable(EI_n));


     // Accumulator Instantiation
     Accumulator A(.CLK(CLK_I),.LA_n(LA_n),.EA(EA),.A_BUS_IN(bus),.A_BUS_OUT(ACC_OUT),.Zero(Z_flag));
     not (not_EA,EA);
     // TRI Buffer to connect Accumulator output to BUS
     tribuf_8bit AccBuf(.in(ACC_OUT), .out(bus), .low_enable(not_EA));

     // ALU Instantiation
     wire aluOut_en;
     assign aluOut_en = (Alu_Controller != 000 )? 0 : 1;
     ALU_Processor Alu(.A(ACC_OUT),.B(RB_out),.AluController(Alu_Controller),.aluOut(alu_Out));
     tribuf_8bit AluBuf(.in(alu_Out), .out(bus), .low_enable(aluOut_en));

     // Register B Instantiation
     regB RB(.clr(clr), .clk(CLK_I), .D(bus), .Q(RB_out),.LB_n(LB_n));

     // Outou Register Instantiation
     out OUT(.L_o(LO_n),.clk(CLK_I),.t(bus),.g(topOUT));

endmodule
```

- **Program Counter:**

```
module PCunit(
      input  CLK, CLR_n, Cp, Ep,zero_flag,j,
      input [3:0] Jumb_In,
      output tri [3:0] PC_out
     );

    reg [3:0] PC_Counter;

    not (not_EP, Ep);
    tribuf_4bit buf2(.in(PC_Counter), .out(PC_out), .low_enable(not_EP));

    always @(posedge CLK or negedge CLR_n)
        begin
            if (!CLR_n)
                begin
                    PC_Counter <= 4'b0;
                end
            else if(j&&zero_flag)
                begin
                    PC_Counter <= Jumb_In;
                end
            else if(Cp)
                begin
                    PC_Counter <= PC_Counter+1;
                end
        end
endmodule
```

- **Tri Buffer 4-Bit:**

```
module tribuf_4bit(in, out, low_enable);
    input [3:0] in; // Input word
    input low_enable; // Enable (active low)
    output  tri [3:0] out; // Output word

    bufif0(out[0], in[0], low_enable);
    bufif0(out[1], in[1], low_enable);
    bufif0(out[2], in[2], low_enable);
    bufif0(out[3], in[3], low_enable);

endmodule
```

- **Tri Buffer 8-Bit:**

```
module tribuf_8bit(in, out, low_enable);

    input [7:0] in; // Input word
    input low_enable; // Enable (active low)
    output tri [7:0] out; // Output word
    tribuf_4bit b0(.in(in[3:0]), .out(out[3:0]), .low_enable(low_enable));
    tribuf_4bit b1(.in(in[7:4]), .out(out[7:4]), .low_enable(low_enable));

endmodule
```

- **Ring Counter:**

```
module ringCounter (
    input clk,clr,
    output reg [5:0] T);

    always@(negedge clk or negedge clr)
        begin
            if(!clr)
                begin
                    T <= 6'b000001;
                end
            else
                begin
                    if(T == 6'b100000)
                        T = 6'b000001;
                    else
                        T = T << 1;
                end
        end

endmodule
```

- **Memory Address Register:**

```
module MAR_Processor (
    input wire clk,clr,Lm_n,
    input wire [3:0] inAddr,
    output reg [3:0] outAddr);

    always@(posedge clk or negedge clr)
    begin
        if(!clr)
            begin
                outAddr <= 4'b0000;
            end
        else if(!Lm_n)
            begin
                outAddr <= inAddr;
            end
    end


endmodule
```

- **16x8 ROM:**

```
module ram (
    input CE_n,
    input [3:0]address,
    output reg [7:0]data_out
);

    reg [7:0] ram_block [15:0];

    initial begin
        $readmemh("Program_Machine_Code.txt",ram_block,0,15);

    end

    always@(*)
        begin
            if(CE_n)
                begin
                    data_out = 8'bzzzzzzzz;
                end
            else
                begin
                    data_out = ram_block[address];
                end
        end

endmodule
```

- **Instruction Register:**

```
module insReg( clk, LI_n, EI_n,clr,w,a ,b);
    input clk;
    input LI_n;
    input EI_n;
    input clr;
    input [7:0] w;
    output reg [3:0] a;
    output reg [3:0] b;


    always@(posedge clk or negedge clr)
        begin
            if (!clr)
                begin
                    a<=4'b0;
                    b<=4'b0;
                end
            else if(!LI_n)
                begin
                    a <= w[3:0];
                    b <= w[7:4];
                end
        end

endmodule
```

- **Accumulator:**

```
module Accumulator(
      input wire CLK, LA_n, EA,
      input wire [7:0] A_BUS_IN,
      output reg [7:0] A_BUS_OUT,
      output reg Zero
     );


     always @(posedge CLK)
         begin
             if(!LA_n)
                 begin
                     A_BUS_OUT <= A_BUS_IN;
                 end
             Zero = ~|(A_BUS_OUT);
         end
endmodule
```

- **Controller Sequencer:**

```
module controller_sequencer (
    input clk,clr_n,
    input [3:0] ins_in,
    output reg CP,EP,Lm_n,CE_n,LI_n,EI_n,LA_n,EA,LB_n,LO_n,HLT_E,j,
    output reg [2:0] Alu_Controller,
    output [5:0] T
    );

    wire [5:0]T_State;
    ringCounter R (.clk(clk),.clr(clr_n),.T(T));

    //assign T = T_State;
    always@(*)
        begin
            if(!clr_n)
                begin
                    {CP,EP,Lm_n,CE_n,LI_n,EI_n,LA_n,EA,Alu_Controller,LB_n,LO_n,HLT_E,j} <= 15'b001111100001100;
                end
            else
                begin
                    if(ins_in == 4'b1111)
                        begin
                            {CP,EP,Lm_n,CE_n,LI_n,EI_n,LA_n,EA,Alu_Controller,LB_n,LO_n,HLT_E,j} <= 15'b001111100001110;
                        end
                    else
                        begin
                            {CP,EP,Lm_n,CE_n,LI_n,EI_n,LA_n,EA,Alu_Controller,LB_n,LO_n,HLT_E,j} <= 15'b001111100001100;
                            case(T)
                                6'b000001: {EP,Lm_n} <= 2'b10;
                                6'b000010: CP <= 1'b1;
                                6'b000100: {CE_n,LI_n} <= 2'b00;
                                6'b001000:  begin
                                                case(ins_in)
                                                    4'b0000: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0001: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0010: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0011: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0100: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0110: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0111: {Lm_n,EI_n} <= 2'b00;
                                                    4'b1000: {Lm_n,EI_n} <= 2'b00;
                                                    4'b0101: {EI_n,j}    <= 2'b01;
                                                    4'b1110: {EA,LO_n}   <= 2'b10;
                                                endcase
                                            end
                                6'b010000: begin
                                                casez(ins_in)
                                                    4'b0???: {CE_n,LB_n} <= 2'b00; //ALU Operations
                                                    4'b0110: {CE_n,LB_n} <= 2'b00;
                                                    4'b0111: {CE_n,LB_n} <= 2'b00;
                                                    4'b1000: {CE_n,LB_n} <= 2'b00;
                                                    4'b0100: {CE_n,LA_n} <= 2'b00;
                                                endcase
                                            end
                                6'b100000: begin
                                                case(ins_in)
                                                    4'b0000: {Alu_Controller,LA_n} <= 4'b0010;      //ADD
                                                    4'b0001: {Alu_Controller,LA_n} <= 4'b0100;      //SUB
                                                    4'b0010: {Alu_Controller,LA_n} <= 4'b0110;      //MUL
                                                    4'b0011: {Alu_Controller,LA_n} <= 4'b1000;      //DIV
                                                    4'b0100: {Alu_Controller,LA_n} <= 4'b0001;      //LOAD
                                                    4'b0110: {Alu_Controller,LA_n} <= 4'b1010;      //AND
                                                    4'b0111: {Alu_Controller,LA_n} <= 4'b1100;      //OR
                                                    4'b1000: {Alu_Controller,LA_n} <= 4'b1110;      //XOR
                                                endcase
                                            end

                                endcase
                        end
                end
        end
endmodule
```

- **ALU:**

```
module ALU_Processor (
    input [7:0] A,B,
    input [2:0] AluController,
    output reg [7:0] aluOut);


    always@(*)
        begin
            case(AluController)
                3'b000: aluOut= aluOut;
                3'b001: aluOut= A+B;
                3'b010: aluOut= A-B;
                3'b011: aluOut= A*B;
                3'b100: aluOut= A/B;
                3'b101: aluOut= A&B;
                3'b110: aluOut= A|B;
                3'b111: aluOut= A^B;
                default: aluOut= 0;
            endcase

        end
endmodule
```

- **Register B:**

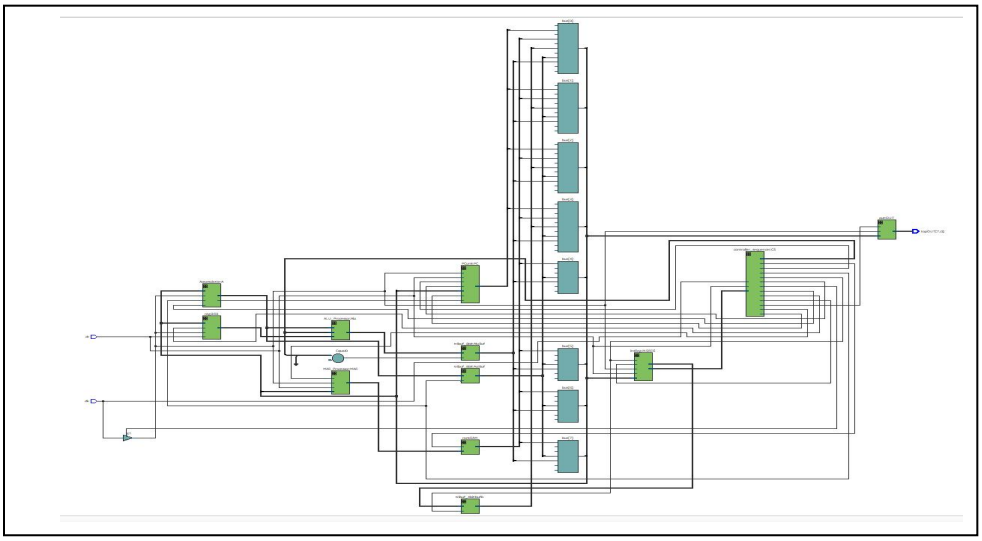
```
module regB (clr, clk, D, Q,LB_n);
    input clr;
    input clk,LB_n;
    input [7:0] D;
    output reg [7:0] Q;

    always @(posedge clk or negedge clr)
        begin
            if (!clr)
                Q <= 0;
            else if(!LB_n)
                Q <= D;
        end
endmodule
```

- **Output Register:**

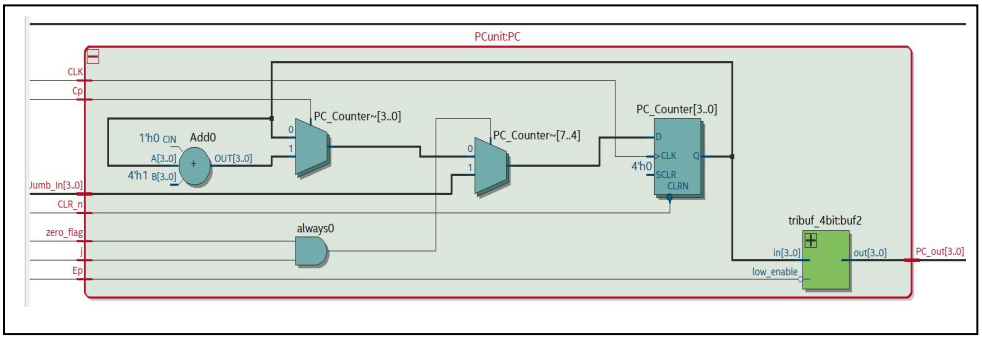
```
module out(L_o,clk,t,g);
    input L_o;
    input clk;
    input [7:0] t;
    output reg [7:0] g;
    always @(posedge clk)
        begin
            if (L_o==0)
                begin
                    g<=t;
                end
            else
                begin
                    g<=g;
                end
        end

endmodule
```

ix. Netlist View for each Module:

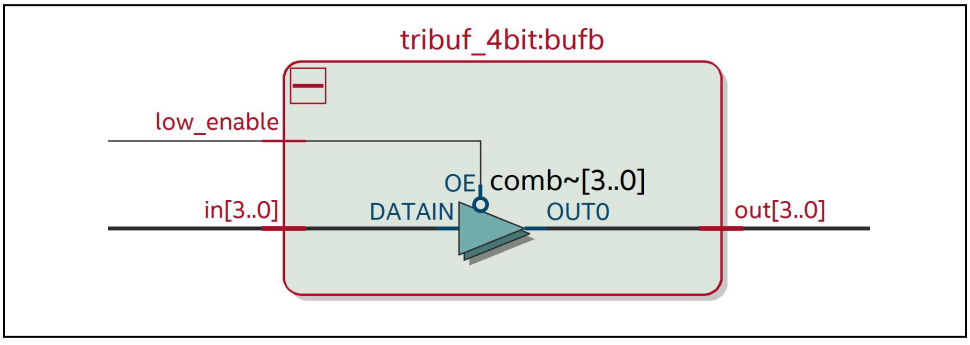
- **Top Module:**



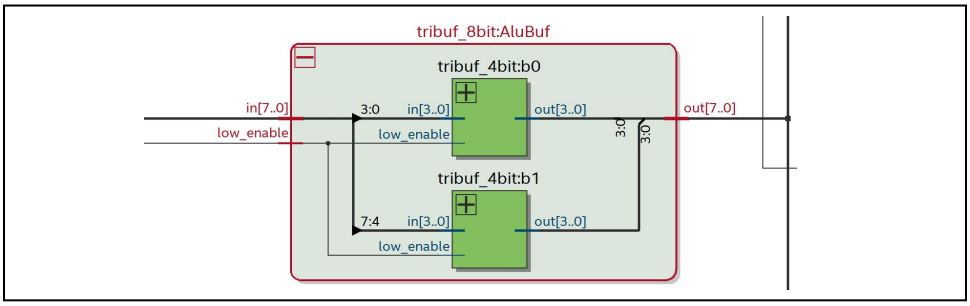
- **Program Counter:**



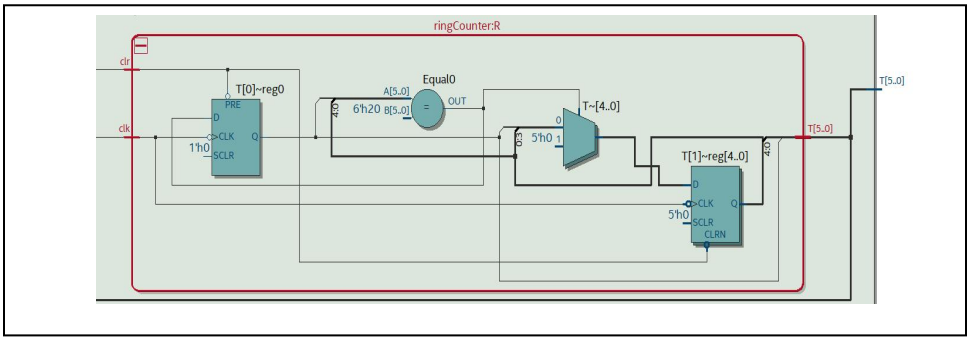
- **Tri Buffer 4-Bit:**



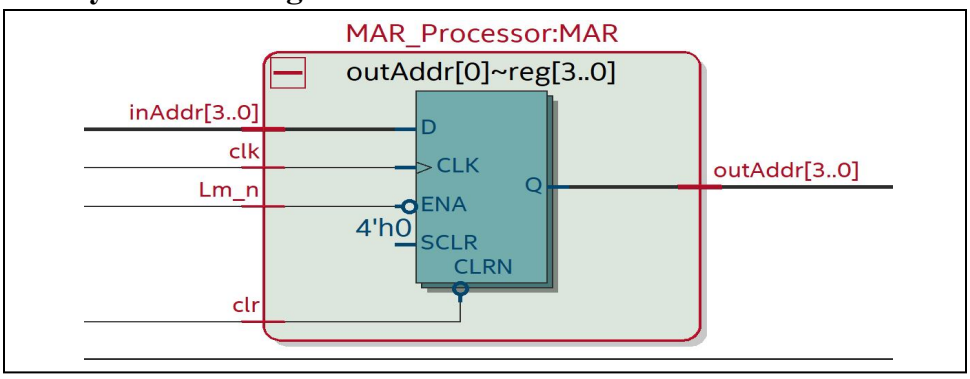
- **Tri Buffer 8-Bit:**



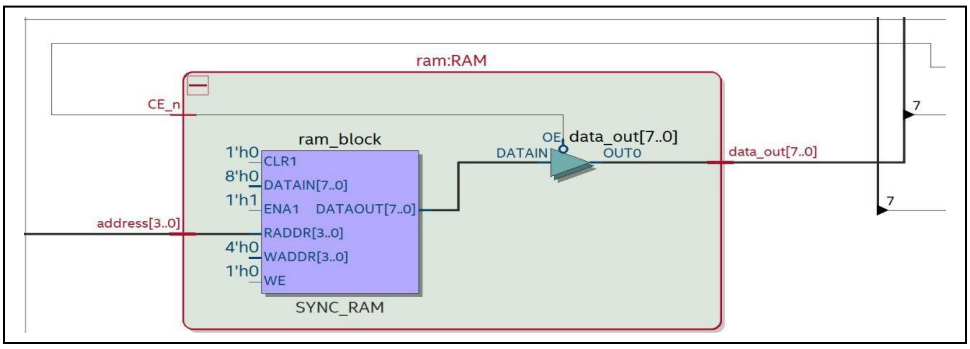
- **Ring Counter:**



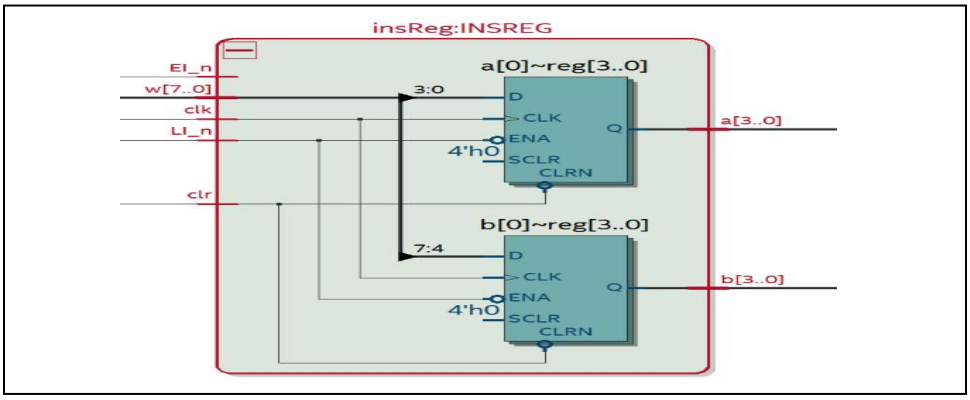
- **Memory Address Register:**



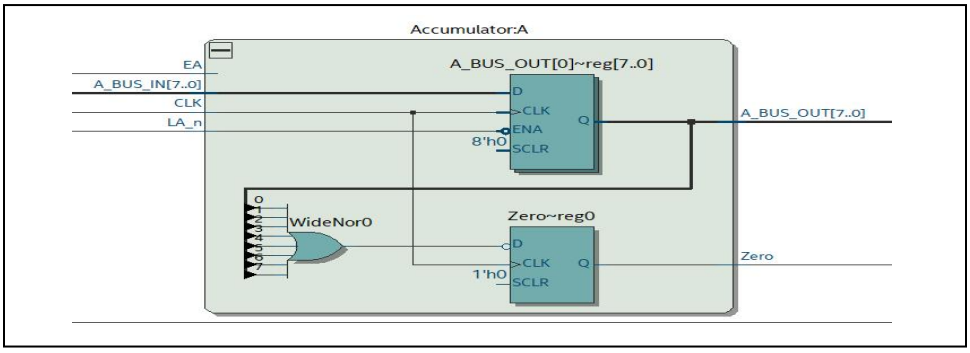
- **16x8 ROM:**



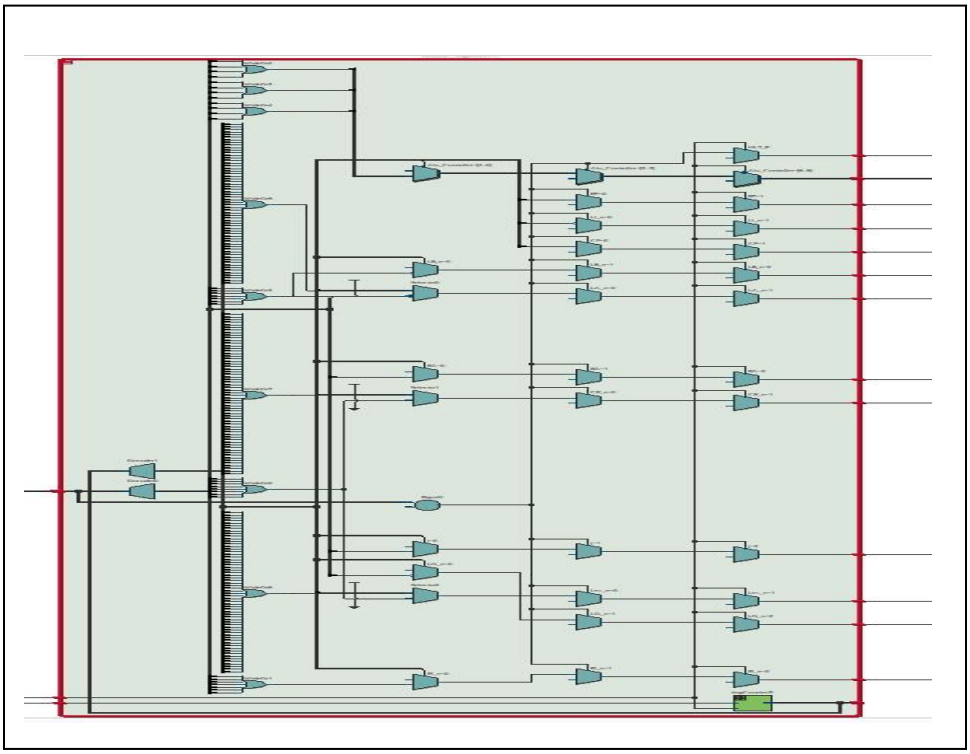
- **Instruction Register:**



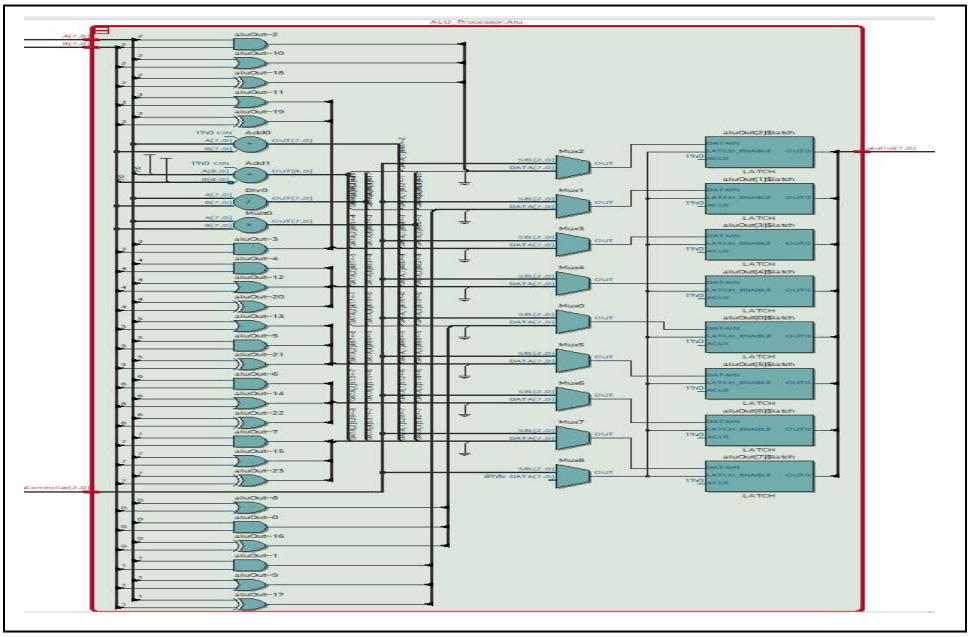
- **Accumulator:**



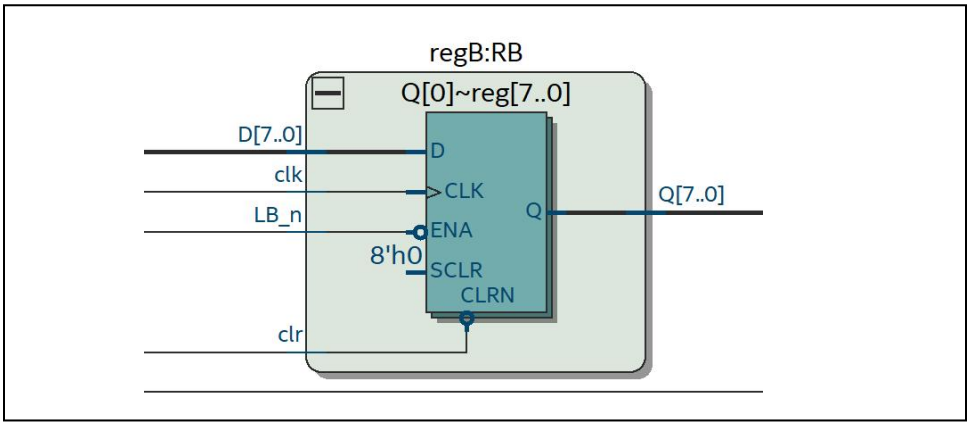
- **Controller Sequencer:**



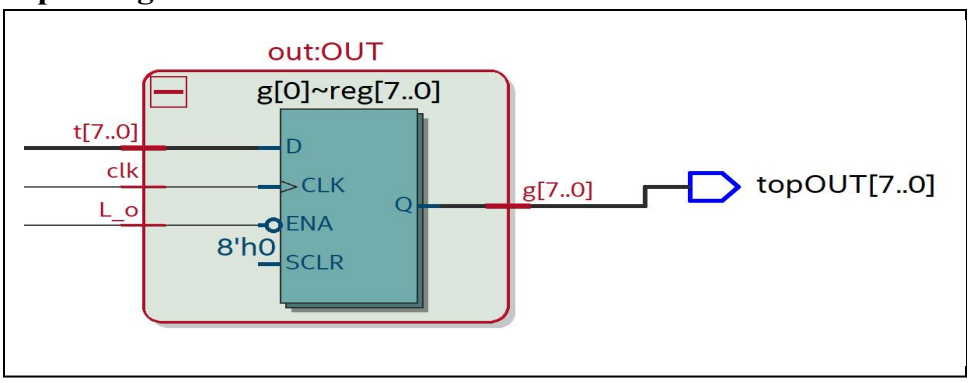
- **ALU:**



- **Register B:**



- **Output Register:**



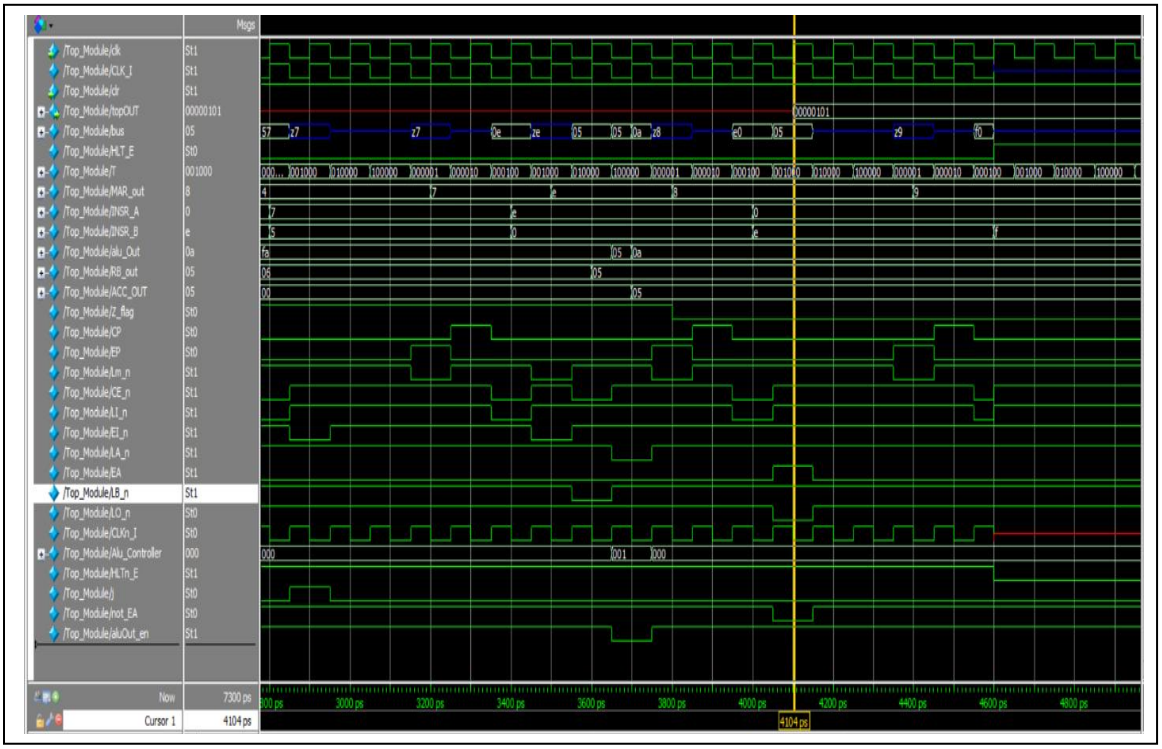
x. Testing Plan:

a. Program 1:

- The following program is Testing Arithmetic Instructions and Branch Instruction:
  - Out = 1+2+3-6+5 = 5 "Jump Occurred"

```
Address      Content
  0H           4A          // LDA AH
  1H           0B          // ADD BH
  2H           0C          // ADD CH
  3H           1D          // SUB DH
  4H           57          // JZ  7H
  5H           E0          // OUT
  6H           F0          // HLT
  7H           0E          // ADD 0E
  8H           E0          // OUT
  9H           F0          // HLT

  AH           01
  BH           02
  CH           03
  DH           06
  EH           05
  FH           FF
```

  - Simulation

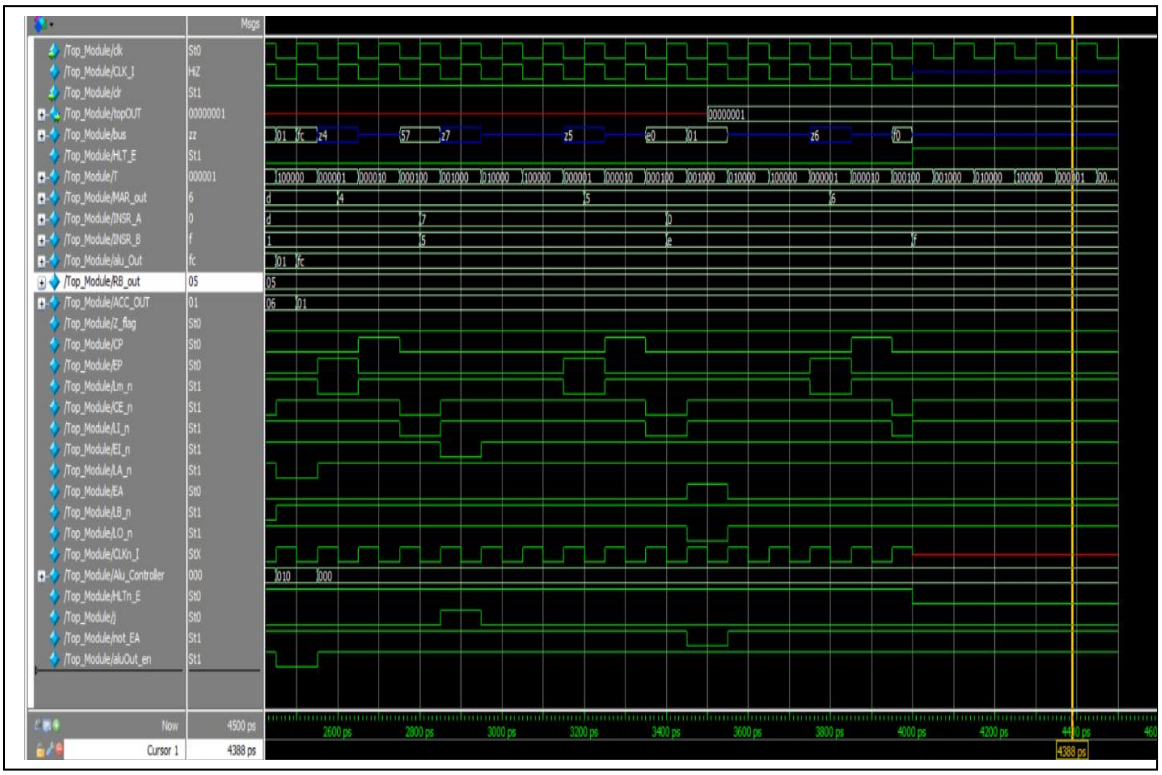


b. Program 2:

- The following program is Testing Arithmetic Instructions and Branch Instruction:
  - Out = 1+2+3-5 = 1 "Jump Ignored"

```
Address      Content
  0H           4A          // LDA AH
  1H           0B          // ADD BH
  2H           0C          // ADD CH
  3H           1D          // SUB DH
  4H           57          // JZ  7H
  5H           E0          // OUT
  6H           F0          // HLT
  7H           0E          // ADD 0E
  8H           E0          // OUT
  9H           F0          // HLT

  AH           01
  BH           02
  CH           03
  DH           05
  EH           05
  FH           FF
```

  - Simulation

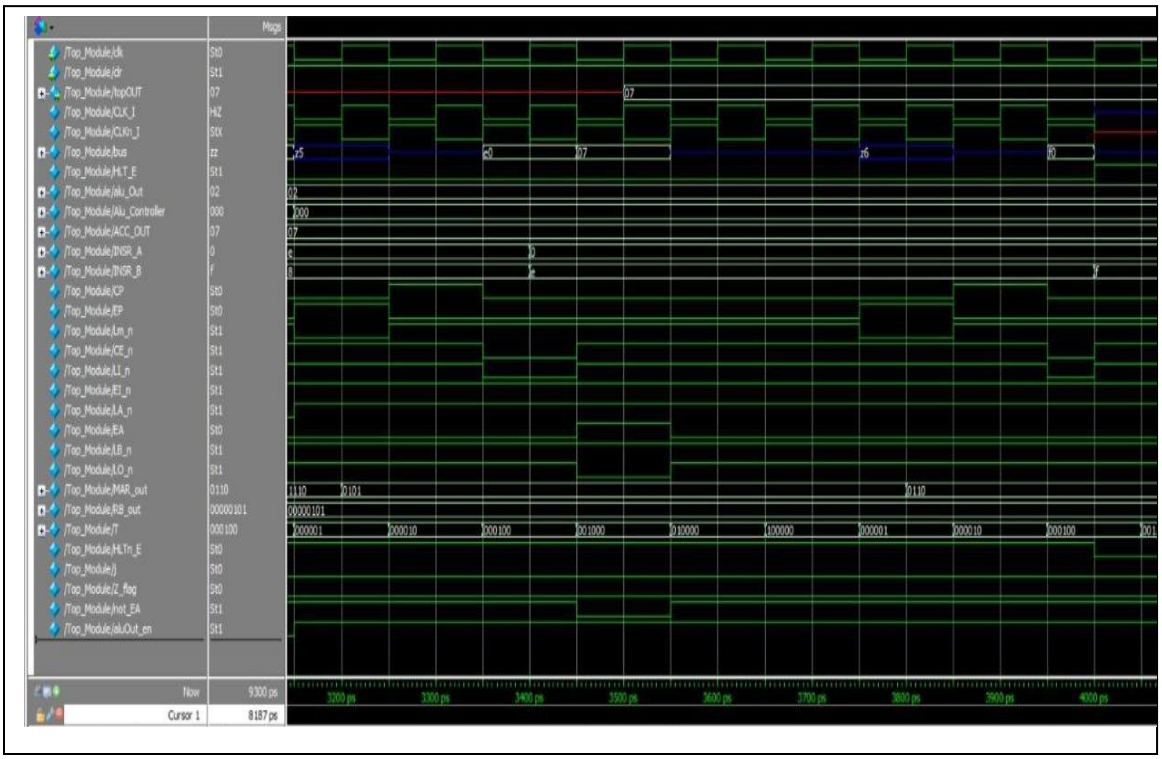


c. Program 3:

- The following program is Testing Logic Instructions:
  - Out = (((1 & 2) | 3) & 2) ^ 5 = 7

```
Address      Content
   0H          4A          // LDA AH
   1H          6B          // AND BH
   2H          7C          // OR  CH
   3H          6D          // AND DH
   4H          8E          // XOR 7H
   5H          E0          // OUT
   6H          F0          // HLT
   7H          FF
   8H          FF
   9H          FF

   AH          01
   BH          02
   CH          03
   DH          02
   EH          05
   FH          FF
```

  - Simulation

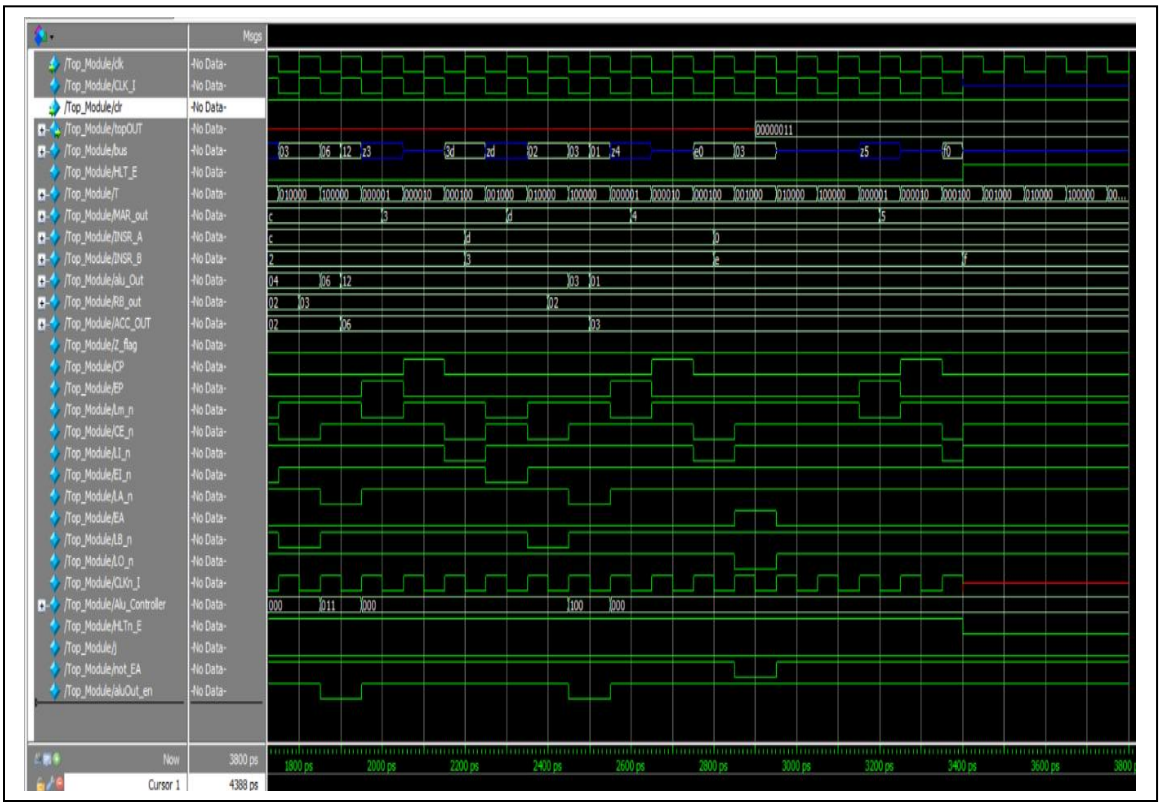


d. Program 4:

- The following program is Testing Arithmetic Instructions (MUL & DIV):
  - Out = ((1 * 2) * 3) /2 = 3

```
Address      Content
  0H           4A          // LDA AH
  1H           2B          // MUL BH
  2H           2C          // MUL CH
  3H           3D          // DIV DH
  4H           E0          // OUT
  5H           F0          // HLT
  6H           FF
  7H           FF
  8H           FF
  9H           FF

  AH           01
  BH           02
  CH           03
  DH           02
  EH           FF
  FH           FF
```

  - Simulation



xi. Design Matrix Reports:

- **Timing Reports:**
  - Default CLK:

Clocks

<<Filter>>

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by
1	clk	Base	12.000	83.33 MHz	0.000	6.000		

  - Maximum Frequency "For Worst Case Delay":

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	90.32 MHz	90.32 MHz	clk	

  - Setup Summary:

Slow 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.464	0.000

  - Hold Summary:

Slow 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.355	0.000

- **Flow Summary Report:**

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Dec 18 22:31:20 2023
Quartus Prime Version	22.1std.2 Build 922 07/20/2023 SC Lite Edition
Revision Name	finalproject
Top-level Entity Name	Top_Module
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	114 / 56,480 (< 1 %)
Total registers	47
Total pins	10 / 268 (4 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	1 / 156 (< 1 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

- **Analysis and Synthesis Report:**

Analysis & Synthesis Summary

<<Filter>>

Analysis & Synthesis Status	Successful - Mon Dec 18 22:30:27 2023
Quartus Prime Version	22.1std.2 Build 922 07/20/2023 SC Lite Edition
Revision Name	finalproject
Top-level Entity Name	Top_Module
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	47
Total pins	10
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	1
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

- **Resources Usage Report:**

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	124
2		
3	▼ Combinational ALUT usage for logic	202
1	-- 7 input functions	1
2	-- 6 input functions	41
3	-- 5 input functions	63
4	-- 4 input functions	47
5	-- <=3 input functions	50
4		
5	Dedicated logic registers	47
6		
7	I/O pins	10
8		
9	Total DSP Blocks	1
10		
11	Maximum fan-out node	clr~input
12	Maximum fan-out	55
13	Total fan-out	1088
14	Average fan-out	4.03

- **Power Report:**

Power Analyzer Summary

<<Filter>>

Power Analyzer Status	Successful - Mon Dec 18 22:57:29 2023
Quartus Prime Version	22.1std.2 Build 922 07/20/2023 SC Lite Edition
Revision Name	finalproject
Top-level Entity Name	Top_Module
Family	Cyclone V
Device	5CGXFC7C7F23C8
Power Models	Final
Total Thermal Power Dissipation	358.82 mW
Core Dynamic Thermal Power Dissipation	1.95 mW
Core Static Thermal Power Dissipation	349.48 mW
I/O Thermal Power Dissipation	7.38 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

xii. Teamwork Plan:

- First, we put design specification, I/O ports, Control Signals, Block Diagram, Timing Diagram and Programming Guide together.
- Second, we divided sub-blocks between us 2 sub-blocks for each (except control unit) and wrote the RTL of each block.
- Third, we wrote the RTL for the control unit together.
- Forth, we integrated the sub-blocks together.
- Fifth, we put testing plan contains 4 programs that test all instructions of the processor.
- Sixth, we generated design matrix reports.