Name: Muhammad Aziz Haider

EE-272L Digital Systems Design

Reg. No: 2020-EE-172

Marks Obtained: ____________

CEP Manual

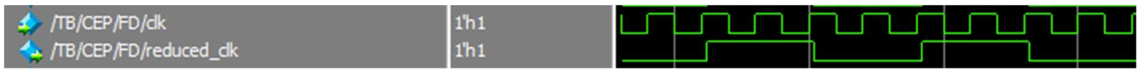
Single Cycle RISC-V Processor for GCD

Truth Table for Controller

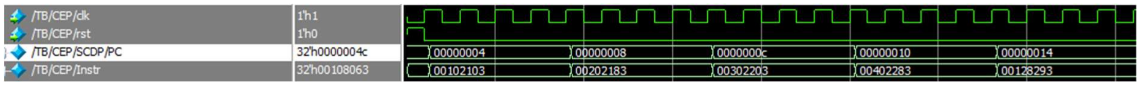
Instruction	opcode	RegWrite	ImmSrc	ALUSrc	ResultSrc	ALUOp
I-Type lw	0000011	1	0	1	1	00
I-Type addi	0010011	1	0	1	0	10
R-Type	0110011	1	X	0	0	10
B-Type	1100011	0	1	0	x	01

Instruction	ALUOp	func3	RtypeSub = op[5] & func7[5]	ALUControl
lw	00	x	X	000
add - addi	10	000	0	000
sub	10	000	1	001
sltu	10	000	X	101
bne	01	X	X	001
beq	01	X	X	001

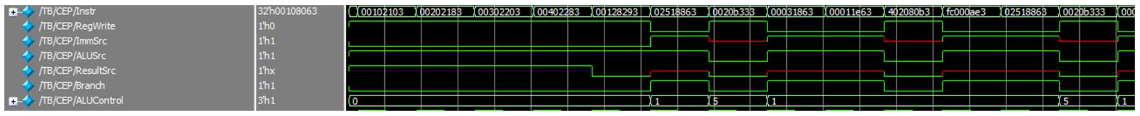
Decreasing frequency of the clock



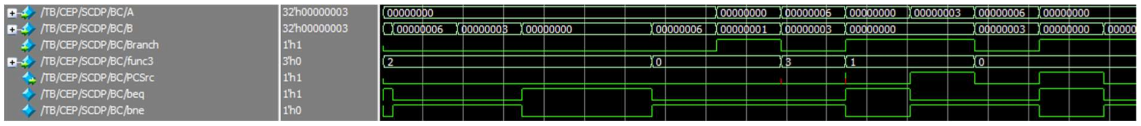
Reading Instructions from Instruction Memory



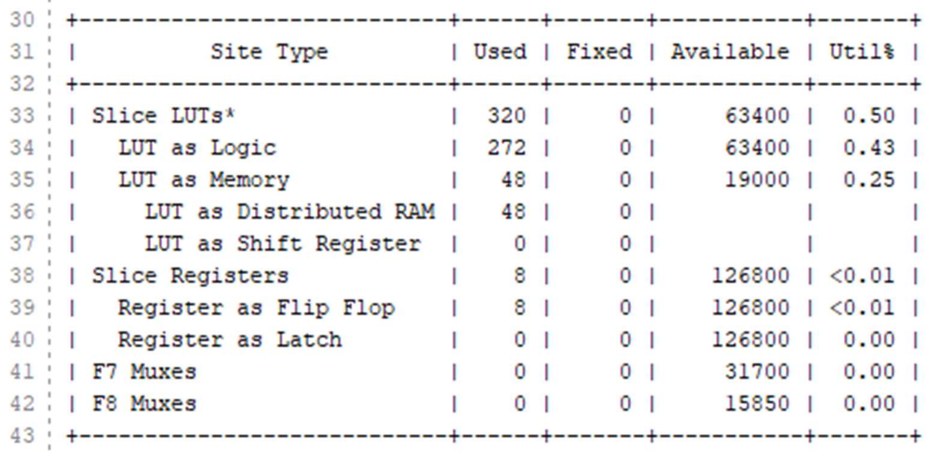
Controller Signals



Branch Comparator

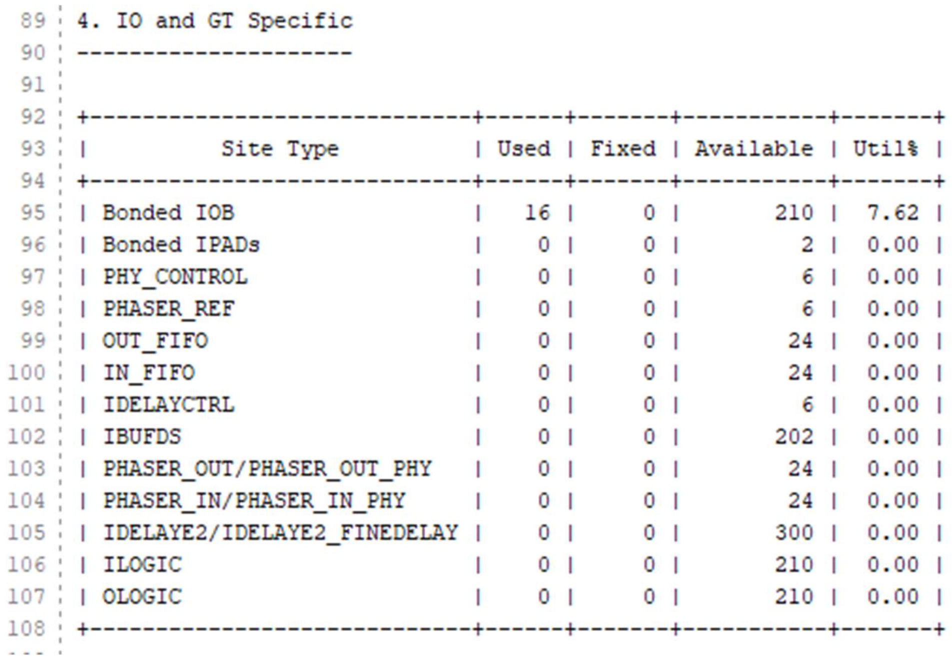


LUTs Used



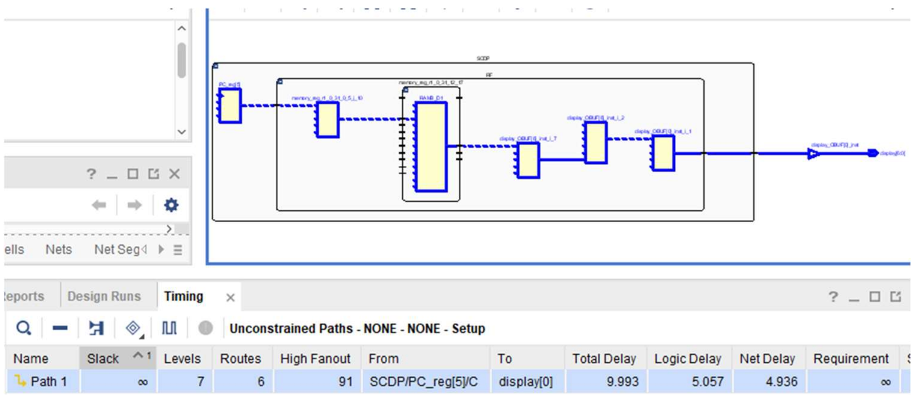
```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |  320 |     0 |     63400 |  0.50 |
|   LUT as Logic             |  272 |     0 |     63400 |  0.43 |
|   LUT as Memory            |   48 |     0 |     19000 |  0.25 |
|     LUT as Distributed RAM |   48 |     0 |           |       |
|     LUT as Shift Register  |    0 |     0 |           |       |
| Slice Registers            |    8 |     0 |    126800 | <0.01 |
|   Register as Flip Flop    |    8 |     0 |    126800 | <0.01 |
|   Register as Latch        |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                   |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                   |    0 |     0 |     15850 |  0.00 |
+----------------------------+------+-------+-----------+-------+
```

I/O Specifics



```
4. IO and GT Specific
---------------------


+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Bonded IOB                  |   16 |     0 |       210 |  7.62 |
| Bonded IPADs                |    0 |     0 |         2 |  0.00 |
| PHY_CONTROL                 |    0 |     0 |         6 |  0.00 |
| PHASER_REF                  |    0 |     0 |         6 |  0.00 |
| OUT_FIFO                    |    0 |     0 |        24 |  0.00 |
| IN_FIFO                     |    0 |     0 |        24 |  0.00 |
| IDELAYCTRL                  |    0 |     0 |         6 |  0.00 |
| IBUFDS                      |    0 |     0 |       202 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |        24 |  0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |        24 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |       300 |  0.00 |
| ILOGIC                      |    0 |     0 |       210 |  0.00 |
| OLOGIC                      |    0 |     0 |       210 |  0.00 |
+-----------------------------+------+-------+-----------+-------+
```

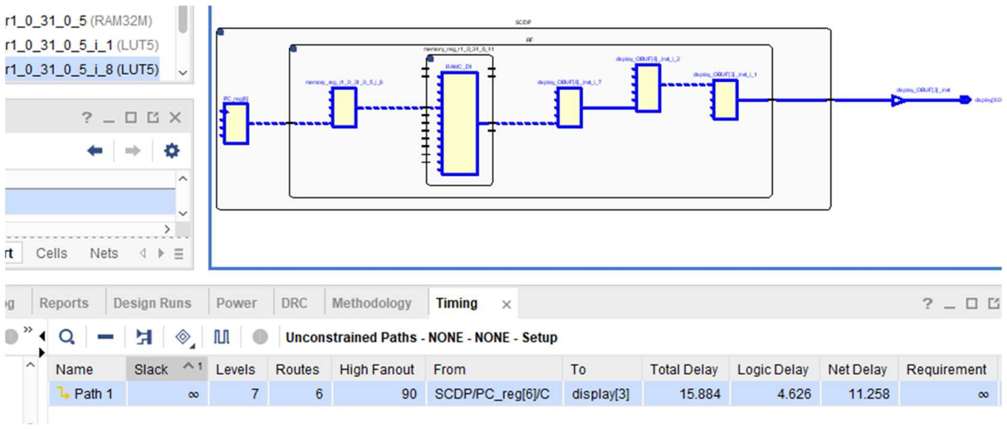
Pre Implementation Delay



Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	7	6	91	SCDP/PC_reg[5]/C	display[0]	9.993	5.057	4.936	∞

Post Implementation Delay



Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	7	6	90	SCDP/PC_reg[6]/C	display[3]	15.884	4.626	11.258	∞

```
lw x1, 0(x0)
lw x2, 1(x0)
lw x3, 2(x0)
lw x4, 3(x0)
lw x5, 4(x0)
addi x5, x5, 1
check:  beq x3,  x5, result
sltu x6, x1, x2
bne x6, x0, swap
bne x2, x0, subtract
addi x3, x3,1
beq x0, x0, check
swap:   add x7, x1, x0
add x1, x2, x0
add x2, x7, x0
beq x0, x0, check
subtract:   sub x1, x1, x2
beq x0, x0, check
result: add x4, x1, x0
beq x1,x1,0
```

```
module RiscV_SS_Processor(input logic clk,rst,
output logic [7:0] seg,
output logic [6:0] display);

    logic [31:0] Instr,outz;
    logic RegWrite, ImmSrc, ALUSrc, ResultSrc,Branch;
    logic [2:0] ALUControl;
    logic clk_25MHz;

    //we need 25MHz clock for our single cycle processor
    FrequencyDivider FD(clk,clk_25MHz);

    DataPath
    SCDP(outz,Instr,ALUControl,Branch,RegWrite,ImmSrc,ResultSrc,ALUSrc,clk_25MHz,rst);
    Controller SCC(Instr,ALUControl,ResultSrc,RegWrite,ImmSrc,Branch,ALUSrc);

    //4x7 Decoder for Character Display on FPGA
    always @(*) begin
        case(outz)
            0: display = 7'b1000000;
            1: display = 7'b1111001;
            2: display = 7'b0100100;
            3: display = 7'b0110000;
            5: display = 7'b0011001;
            6: display = 7'b0010010;
            7: display = 7'b0000010;
            8: display = 7'b1111000;
            9: display = 7'b0000000;
            default: display = 7'b1111111;
        endcase
    end

    //Assigning seg like this because we need to use only one character on FPGA
    assign seg = 8'b11111110;

endmodule
```

```
module FrequencyDivider(input logic clk, output logic reduced_clk);

    //FPGA gives us a clock frequency of 100MHz
    //We need 25 MHz for our single cycle risc v processor
    //We use two T-flip flops for that purpose
    //Basically T flip flop is a bit counter (synchronous) with enable

    logic [1:0] t = 0;
    always_ff @(posedge clk) begin
        t[0] <= #1 t[0] + 1;
    end

    always_ff @(posedge t) begin
        t[1] <= #1 t[1] + 1;
    end
    assign reduced_clk = t[1];

endmodule
```

```
module DataPath(
output logic [31:0] y,
output logic [31:0] Instr,
input logic [2:0] ALUControl,
input logic Branch,RegWrite,ImmSrc,ResultSrc,ALUSrc,clk,rst);

    logic [31:0] PCNext;
    logic [31:0] PCPlus4,PCTarget;

    //PC Counter
    logic [31:0] PC = 0;
    always_ff @(posedge clk) begin
        PC <= #1 PCNext;
    end

    //Adder for PC
    always_comb begin
        PCPlus4 = PC + 4;
    end

    InstructionMemory IM(PC,Instr); //Instruction Memory

    logic [31:0] SrcA,SrcB,RD2;
    logic [31:0] Result;

    RegisterFile
    RF(SrcA,RD2,Result,Instr[19:15],Instr[24:20],Instr[11:7],RegWrite,clk,rst);
    //Register File

    //Immediate Generator and extender to 32 bits
    logic [31:0] ImmExt;
    ImmediateGen IG(ImmSrc,Instr,ImmExt);

    //Adder for Branching
    always_comb begin
        PCTarget = PC + ImmExt;
    end

    //Comparator for branching if beq or bne occurs
    logic PCSrc;
    BranchComparator BC(SrcA,RD2,Branch,Instr[14:12],PCSrc);

    //mux for Source 2 of ALU
    always_comb begin
        case(ALUSrc)
        0: SrcB = RD2;
        1: SrcB = ImmExt;
        endcase
    end

    //mux for PC Counter
    always_comb begin
        case(PCSrc)
        0: PCNext = PCPlus4;
        1: PCNext = PCTarget;
        endcase
    end

    //Algorithmic Logic Unit
    logic [31:0] ALUResult;
    ALU LogicUnit(SrcA,SrcB,ALUControl,ALUResult);

    //Data Memory
    logic [31:0] ReadData;
    DataMemory DM(ALUResult,ReadData);

    //Mux for Result Source of Instruction
    always_comb begin
        case(ResultSrc)
```

```
            0: Result = ALUResult;
            1: Result = ReadData;
            endcase
        end

        //Using this variable for output
        always_comb begin
            y = SrcA;
        end

    endmodule
```

```
module InstructionMemory(
input logic [31:0] PC,
output logic [31:0] Instr);

    logic [31:0] InstrMem [31:0];  //2D Array

    initial begin
        $readmemb("C:/Users/PC/OneDrive/Desktop/CEP/Codes/IM_Data.mem",InstrMem);
    end

    always_comb begin
        Instr = InstrMem[PC[31:2]];
    end

endmodule
```

```
00000000000000000010000010000011
00000000000100000010000100000011
00000000001000000010000110000011
00000000001100000010001000000011
00000000010000000010001010000011
00000000000100101000001010010011
00000010010100011000100001100011
00000000001000001011001100110011
00000000000000110001100001100011
00000000000000010001111001100011
00000000000100011000000110010011
11111110000000000000011011100011
00000000000000001000001110110011
00000000000000010000000010110011
00000000000000111000000100110011
11111100000000000000111011100011
01000000001000001000000010110011
11111100000000000000101011100011
00000000000000001000001000110011
00000000000100001000000001100011
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```systemverilog
module RegisterFile(
output logic [31:0] RD1, RD2,
input logic [31:0] WD3,
input logic [4:0] A1, A2, A3,
input  logic RegWrite, clk, rst);

    logic [31:0] memory [31:0];  //2D Array
    //mux 1
    always_comb begin
        if(A1==0) RD1 = 0;
        else RD1 = memory[A1];
    end
    //mux 2
    always_comb begin
        if(A2==0) RD2 = 0;
        else RD2 = memory[A2];
    end

    //Writing data to register file
    always_ff @(posedge clk) begin
        if(rst) begin  //Added reset for the purpose of simulation
            for (int i = 0; i <= 31; i += 1) memory[i] <= #1 0;
        end
        else if (RegWrite) memory[A3] <= #1 WD3;
    end

endmodule
```

```systemverilog
module ImmediateGen(
input logic ImmSrc,
input logic [31:0] Instr,
output logic [31:0] Imm);

    always_comb begin
        case(ImmSrc)
            1'b0: Imm = {{20{Instr[31]}}, Instr[31:20]}; //If the Immediate is I type
            1'b1: Imm =  {{20{Instr[31]}}, Instr[7], Instr[30:25], Instr[11:8], 1'b0};
            //If the Immediate is B type
            default: Imm = 32'bx; // undefined
        endcase
    end

endmodule
```

```
module ALU(
input logic [31:0] A,B,
input logic [2:0] sel,
output logic [31:0] result);

    always_comb begin
        case (sel)
        3'b000:result = A + B; //AND
        3'b001:result = A - B; //SUB
        3'b010:result = A | B; //OR
        3'b011:result = A ^ B; //XOR
        3'b100:result = A & B; //AND
        3'b101:result = A < B; //SLTU
        default: result = A + B;
        endcase
    end

endmodule
```

```
module BranchComparator(input logic [31:0] A,B,
input logic Branch,
input logic [2:0]func3,
output logic PCSrc);

    logic beq,bne;

    always_comb begin
        if (A == B) begin
            beq = 1; //Set beq = 1 if the statements are equa
            bne = 0;
            end
        else begin
            beq = 0; //Otherwise set bne = 1 if statements are not equal
            bne = 1;
            end
    end

    always_comb begin
        case({Branch,func3})
        4'b0_000: PCSrc = 1'b0; //add - addi - sub
        4'b0_010: PCSrc = 1'b0; //lw
        4'b0_011: PCSrc = 1'b0; //sltu
        4'b1_000: PCSrc = beq;
        4'b1_001: PCSrc = bne;
        default: PCSrc = 1'bx;
        endcase
    end

endmodule
```

```systemverilog
module DataMemory(
input logic [31:0] A,
output logic [31:0] RD);

    logic [31:0] DataMem [31:0];  //2D Array

    //Instantiating the data memory
    initial begin
        $readmemh("C:/Users/PC/OneDrive/Desktop/CEP/Codes/DM_Data.mem",DataMem);
    end

    always_comb begin
        RD = DataMem[A[4:0]]; //Reading Data from Data Memory
    end

endmodule
```

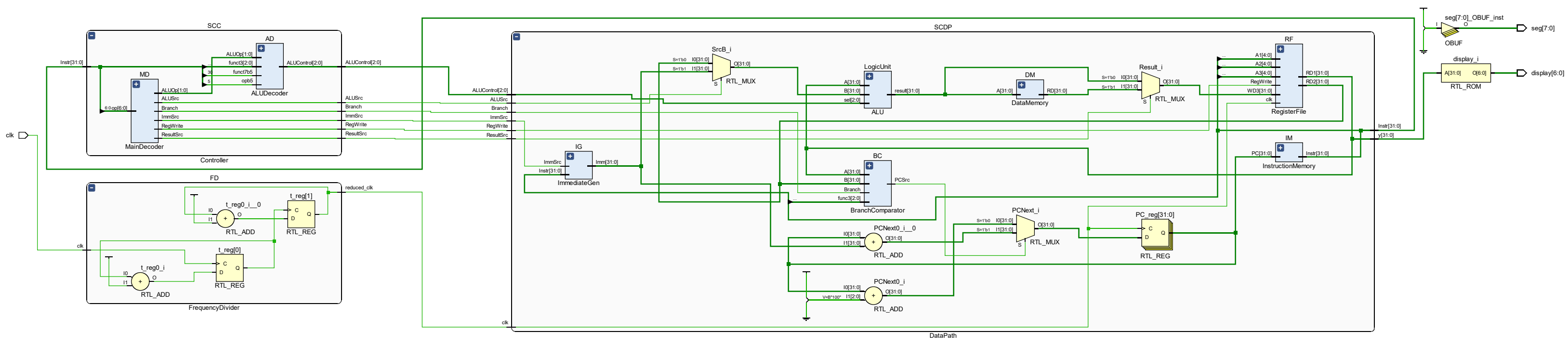
```
00000006
00000003
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
module Controller(
input logic [31:0] Instr,
output logic [2:0] ALUControl,
output logic ResultSrc,RegWrite,ImmSrc,Branch,ALUSrc);

    logic [1:0] ALUOp;
    //Calling two modules into controller
    MainDecoder MD(Instr[6:0],RegWrite,ImmSrc,ResultSrc,Branch,ALUSrc,ALUOp);
    ALUDecoder AD(ALUOp,Instr[5],Instr[30],Instr[14:12],ALUControl);

endmodule
```

```
module MainDecoder(input logic [6:0] op,
output logic RegWrite,ImmSrc,ResultSrc,Branch,ALUSrc,
output logic [1:0] ALUOp);

    logic [6:0] control;
    assign {RegWrite,ImmSrc,ResultSrc,ALUSrc,ALUOp,Branch} = control; //Concatenate the
    output to make coding easy

    always_comb begin
        case(op)
        //I-type lw instruction
        7'b0000011: control = 7'b1_0_1_1_00_0;
        //I-type addi instructions
        7'b0010011: control = 7'b1_0_0_1_10_0;
        //R-type
        7'b0110011: control = 7'b1_x_0_0_10_0;
        //B-type
        7'b1100011: control = 7'b0_1_x_1_01_1;
        default: control = 7'bx_x_x_x_xx_x;
        endcase
    end

endmodule
```

```
module ALUDecoder(input logic [1:0] ALUOp,
input logic opb5,funct7b5,input logic [2:0] funct3,
output logic [2:0] ALUControl);

 logic RtypeSub;
 assign RtypeSub = funct7b5 & opb5; // TRUE for R-type subtract

 always_comb
     case(ALUOp)
         2'b00: ALUControl = 3'b000; // addition
         2'b01: ALUControl = 3'b001; // subtraction
         default: case(funct3) // R-type or I-type ALU
                 3'b000: if (RtypeSub)
                 ALUControl = 3'b001; // sub
                 else ALUControl = 3'b000; // add, addi
                 3'b011: ALUControl = 3'b101; // slt, sltu
                 3'b110: ALUControl = 3'b011; // or, ori
                 3'b111: ALUControl = 3'b010; // and, andi
                 default: ALUControl = 3'bxxx; // ???
                 endcase
     endcase

endmodule
```

```
module TB();

    logic clk,rst;
    logic [6:0] display;
    logic [7:0] seg;

    RiscV_SS_Processor CEP(clk,rst,seg,display);

    initial begin
        clk = 0;
        forever #10 clk = ~clk; //Time Period for 100MHz
    end

    initial begin
        rst = 1;
        @(posedge clk);
        rst = 0;
        repeat (240) @(posedge clk);
        $stop;
    end

endmodule
```



clk
SCC
Controller
AD
ALUDecoder
ALUOp[1:0]
funct3[2:0]
funct7b5
opb5
ALUControl[2:0]
MD
MainDecoder
Instr[31:0]
6:0 op[6:0]
ALUSrc
Branch
ImmSrc
RegWrite
ResultSrc
FD
FrequencyDivider
reduced_clk
t_reg[1]
t_reg[0]
t_reg0_i__0
t_reg0_i
RTL_ADD
RTL_REG
SCDP
DataPath
SrcB_i
RTL_MUX
LogicUnit
ALU
A[31:0]
B[31:0]
sel[2:0]
result[31:0]
DM
DataMemory
RD[31:0]
Result_i
RF
RegisterFile
A1[4:0]
A2[4:0]
A3[4:0]
WD3[31:0]
RD1[31:0]
RD2[31:0]
IG
ImmediateGen
Imm[31:0]
BC
BranchComparator
func3[2:0]
PCSrc
PCNext0_i__0
PCNext0_i
V=B"100"
PCNext_i
PC_reg[31:0]
IM
InstructionMemory
PC[31:0]
y[31:0]
seg[7:0]_OBUF_inst
OBUF
seg[7:0]
display_i
RTL_ROM
O[6:0]
display[6:0]