National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science

EE-421 Digital System Design

Instructor: Dr Rehan Ahmed

Assignment: 2

Due Date: 22nd May 2022

Total Marks: 100

Submitted By :

Name	ID
Ch. Muhammad Shaheer Yasir	286021
Muhammad Hamza	290951

NuCORE Module:

```
module NuCore (KEY, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, instruction, current_pc, Zero);
input [0:0] KEY;
input [0:0] SW;
output [6:0] HEX0,HEX1, HEX2, HEX3, HEX4, HEX5;
wire clock, reset;
input [38:0] instruction;
output Zero;
wire [63:0] ALUOut;
output reg [5:0] current_pc;
wire [3:0] Address_A, Address_B;
wire [31:0] Write_Data_A, Write_Data_B;
wire [31:0] Read_Data_A, Read_Data_B;
wire [2:0] opcode;


assign clock = KEY[0];
assign reset = SW[0];


always @(posedge clock)
begin
if (reset)
current_pc <= 0;
else
current_pc <= current_pc + 1;
end



//instructionMemory U0 (reset, current_pc, instruction);
assign opcode = instruction [38:36];
decode U1 (clock, instruction, Address_A, Address_B, Write_Data_A, Write_Data_B);
```

```
reg_A U2 (clock, opcode, reset, Address_A, Write_Data_A, Read_Data_A);
reg_B U3 (clock, opcode, reset, Address_B, Write_Data_B, Read_Data_B);
ALU U4 (clock, opcode, Read_Data_A, Read_Data_B, ALUOut, Zero);
display U5(ALUOut[3:0], HEX0);
display U6(ALUOut[7:4], HEX1);
display U7(Read_Data_A[3:0], HEX2);
display U8(Read_Data_A[7:4], HEX3);
display U9(Read_Data_B[3:0], HEX4);
display U10(Read_Data_B[7:4], HEX5);
endmodule

module decode (clock, instruction, Address_A, Address_B, Write_Data_A, Write_Data_B);
input [38:0] instruction;
input clock;
output reg [3:0] Address_A, Address_B;
output reg [31:0] Write_Data_A, Write_Data_B;
wire [2:0] opcode;
assign opcode = instruction [38:36];

always @(*)
begin
if (opcode == 3'b001)
begin
        Address_A = instruction[35:32];
        Write_Data_A = instruction[31:0];
end
else if (opcode == 3'b010)
begin
        Address_B = instruction[35:32];
        Write_Data_B = instruction[31:0];
end
else if (opcode == 3'b011 | opcode == 3'b100 | opcode == 3'b101 | opcode ==  3'b110 | opcode == 3'b111)
```

```
begin
        Address_A = instruction[35:32];
        Address_B = instruction[31:28];
end

end

endmodule

module reg_A (clock, opcode, reset, Address_A, Write_Data_A, Read_Data_A);
input [2:0] opcode;
input [3:0] Address_A;
input [31:0] Write_Data_A;
input reset, clock;
output reg [31:0] Read_Data_A;
integer k;

reg [31:0] reg_file_A [15:0];
always @(*)
begin
        if (reset)
        begin
                for (k = 0; k < 16; k = k + 1);
                begin
                        reg_file_A[k] <= 32'b0;
                end
        end
        else if (opcode == 3'b001)
        begin
                reg_file_A[Address_A] <= Write_Data_A;
        end
        else if (opcode == 3'b011 | opcode == 3'b100 | opcode == 3'b101 | opcode ==  3'b110 | opcode == 3'b111)
```

```
        begin
                Read_Data_A <= reg_file_A[Address_A];
        end
end


endmodule



module reg_B (clock, opcode, reset, Address_B, Write_Data_B, Read_Data_B);
input [2:0] opcode;
input [3:0] Address_B;
input [31:0] Write_Data_B;
input reset, clock;
output reg [31:0] Read_Data_B;
integer k;

reg [31:0] reg_file_B [15:0];
always @(*)
begin
        if (reset)
        begin
                for (k = 0; k < 16; k = k + 1);
                begin
                        reg_file_B[k] <= 32'b0;
                end
        end
        else if (opcode == 3'b010)
        begin
                reg_file_B[Address_B] <= Write_Data_B;
        end
        else if (opcode == 3'b011 | opcode == 3'b100 | opcode == 3'b101 | opcode ==  3'b110 | opcode == 3'b111)
        begin
```

```
                Read_Data_B <= reg_file_B[Address_B];
        end
end

endmodule

module ALU (clock, opcode, Read_Data_A, Read_Data_B, ALUOut, Zero);
input [2:0] opcode;
input [31:0] Read_Data_A, Read_Data_B;
input clock;
output reg [63:0] ALUOut;
output reg Zero;

always @(*)
begin
case(opcode)
3'b000: ALUOut = 63'd0;

3'b011: ALUOut = Read_Data_A + Read_Data_B;

3'b100: ALUOut = Read_Data_A - Read_Data_B;

3'b101: ALUOut = Read_Data_A | Read_Data_B;

3'b110: ALUOut = Read_Data_A & Read_Data_B;

3'b111: ALUOut = Read_Data_A << Read_Data_B;

default: ALUOut = 63'd0;

endcase
```

```
Zero = (ALUOut == 63'd0);


end


endmodule



module display (A, Disp);
input [3:0] A;
output reg [6:0] Disp;
always @(*) begin
case (A)
4'd0 : Disp = 7'b1000000;
4'd1 : Disp = 7'b1111001;
4'd2 : Disp = 7'b0100100;
4'd3 : Disp = 7'b0110000;
4'd4 : Disp = 7'b0011001;
4'd5 : Disp = 7'b0010010;
4'd6 : Disp = 7'b0000010;
4'd7 : Disp = 7'b1111000;
4'd8 : Disp = 7'b0000000;
4'd9 : Disp = 7'b0010000;
4'd10 : Disp = 7'b0001000;
4'd11 : Disp = 7'b0000011;
4'd12 : Disp = 7'b1000110;
4'd13 : Disp = 7'b0100001;
4'd14 : Disp = 7'b0000110;
4'd15 : Disp = 7'b0001110;
endcase
end
endmodule
```

Instruction Memory Module:

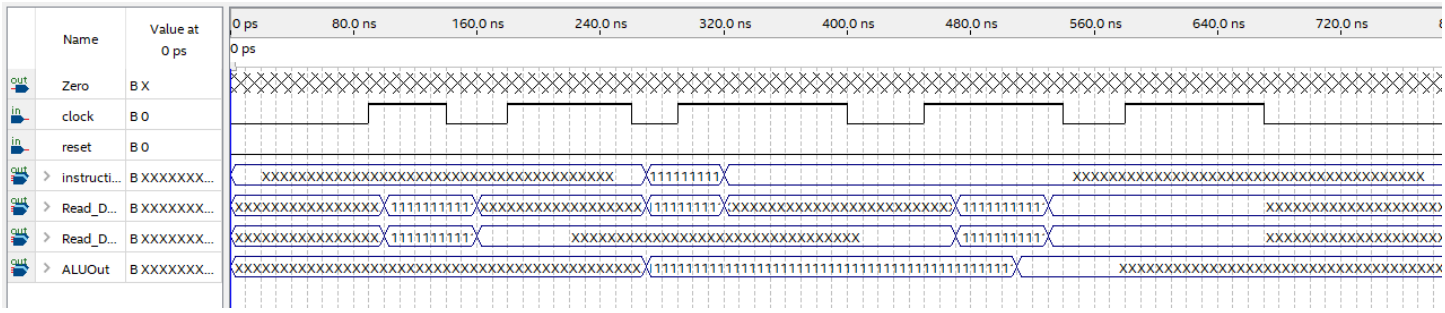
```
module instructionMemory (reset, current_pc, instruction);
input reset;
input [5:0] current_pc;
output [38:0] instruction;
reg [38:0] Imemory [63:0];
integer k;
assign instruction = Imemory[current_pc];
always @(posedge reset)
begin
        for (k = 40; k < 64; k = k + 1)
        begin
            Imemory[k] = 39'b0;
        end
        Imemory [0] <= 39'b0010001000000000000000000000000000000001;
        Imemory [1] <= 39'b0010010000000000000000000000000000000010;
        Imemory [2] <= 39'b0010011000000000000000000000000000000011;
        Imemory [3] <= 39'b0010100000000000000000000000000000000100;
        Imemory [4] <= 39'b0010101000000000000000000000000000000101;
        Imemory [5] <= 39'b0010110000000000000000000000000000000110;
        Imemory [6] <= 39'b0010111000000000000000000000000000000111;
        Imemory [7] <= 39'b0011000000000000000000000000000000001000;
        Imemory [8] <= 39'b0011001000000000000000000000000000001001;
        Imemory [9] <= 39'b0011010000000000000000000000000000001010;
        Imemory [10] <= 39'b0100001000000000000000000000000000001010;
        Imemory [11] <= 39'b0100010000000000000000000000000000001001;

        Imemory [12] <= 39'b0100011000000000000000000000000000001000;
        Imemory [13] <= 39'b0100100000000000000000000000000000000111;
        Imemory [14] <= 39'b0100101000000000000000000000000000000110;
        Imemory [15] <= 39'b0100110000000000000000000000000000000101;
```

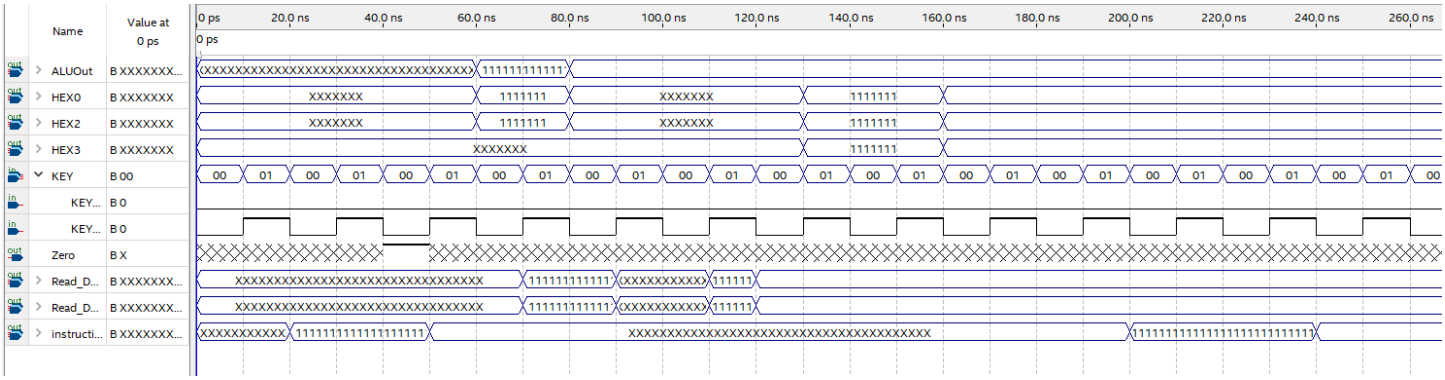
```
        Imemory [16] <= 39'b0100111000000000000000000000000000000100;
        Imemory [17] <= 39'b0101000000000000000000000000000000000011;
        Imemory [18] <= 39'b0101001000000000000000000000000000000010;
        Imemory [19] <= 39'b0101010000000000000000000000000000000001;
        Imemory [20] <= 39'b0110001000100000000000000000000000000000;
        Imemory [21] <= 39'b0110010001000000000000000000000000000000;
        Imemory [22] <= 39'b0110011001100000000000000000000000000000;
        Imemory [23] <= 39'b0110100010000000000000000000000000000000;
        Imemory [24] <= 39'b0110101010100000000000000000000000000000;
        Imemory [25] <= 39'b0110110011000000000000000000000000000000;
        Imemory [26] <= 39'b0110111011100000000000000000000000000000;
        Imemory [27] <= 39'b0111000100000000000000000000000000000000;
        Imemory [28] <= 39'b0111001100100000000000000000000000000000;
        Imemory [29] <= 39'b0111010101000000000000000000000000000000;
        Imemory [30] <= 39'b1100001000100000000000000000000000000000;
        Imemory [31] <= 39'b1100010001000000000000000000000000000000;
        Imemory [32] <= 39'b1100011001100000000000000000000000000000;
        Imemory [33] <= 39'b1100100010000000000000000000000000000000;
        Imemory [34] <= 39'b1100101010100000000000000000000000000000;
        Imemory [35] <= 39'b1100110011000000000000000000000000000000;
        Imemory [36] <= 39'b1100111011100000000000000000000000000000;
        Imemory [37] <= 39'b1101000100000000000000000000000000000000;
        Imemory [38] <= 39'b1101001100100000000000000000000000000000;
        Imemory [39] <= 39'b1101010101000000000000000000000000000000;
end
endmodule
```

Test bench module :

```
module testbench;


reg [0:0] KEY;

reg [0:0] SW;

wire [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

reg [38:0] instruction;

wire [5:0] current_pc;


NuCore DUT(KEY[0], SW[0], HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, instruction, current_pc);


instructionMemory DUT1 (SW[0], current_pc, instruction);


endmodule
```

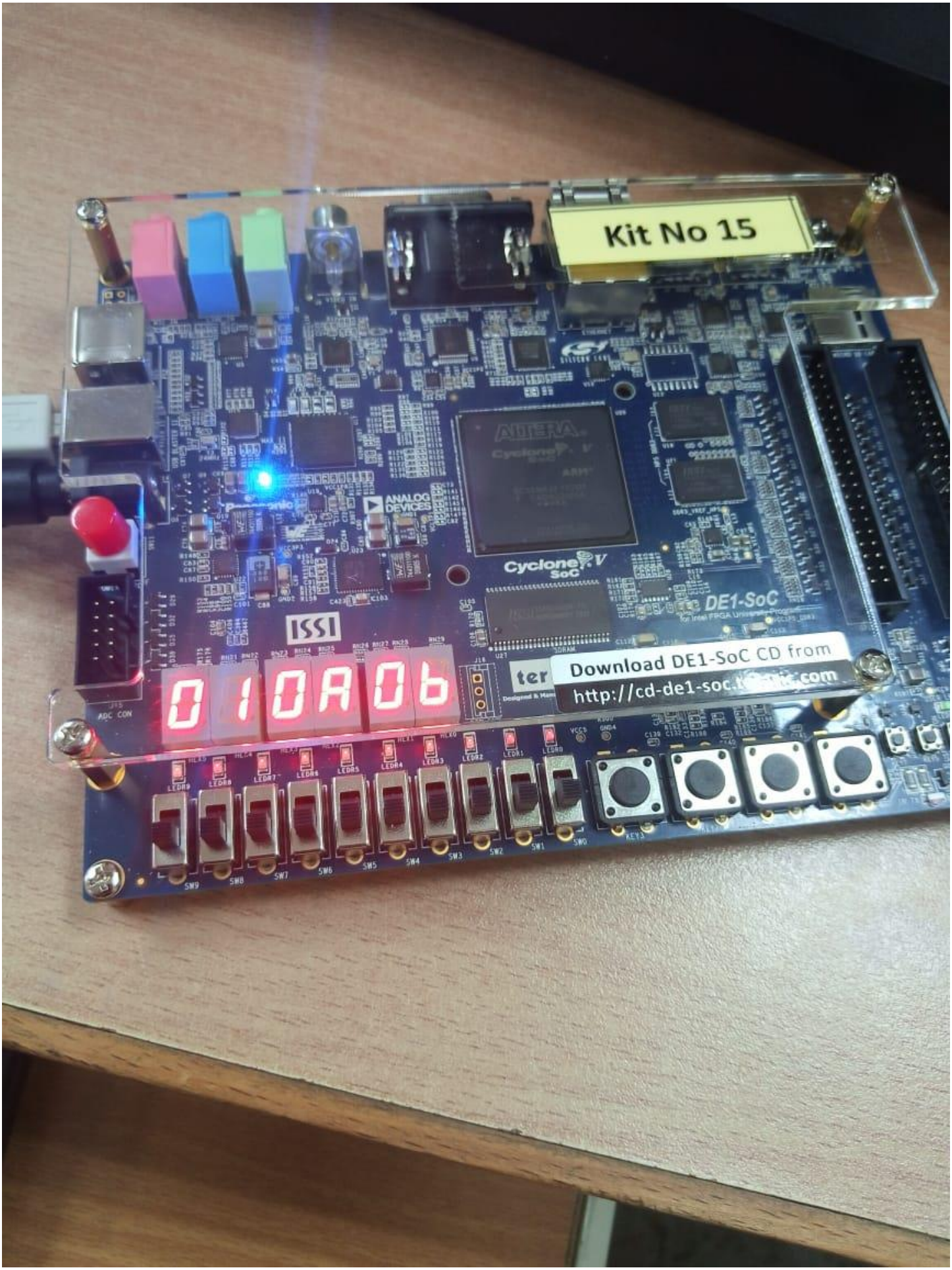
Test Bench Simulation Waveform:





FPGAs OUTPUTS:


Kit No 15
Cyclone V SoC
DE1-SoC
Download DE1-SoC CD from
http://cd-de1-soc.terasic.com
0 1 0 A F 7


Kit No 15
Cyclone V SoC
DE1-SoC
Download DE1-SoC CD from
http://cd-de1-soc.terasic.com
0 1 0A 0b

BLOCK LEVEL DIAGRAM: