ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY
EXPERIMENT REPORT 8

EXPERIMENT NO	:	7
EXPERIMENT DATE	:	04.06.2020
LAB SESSION	:	FRIDAY - 13.30
GROUP NO	:	G11

GROUP MEMBERS:

150180064 : MELİS & GÜNŞEBER

150170043 : EBRAR & ÖMER

SPRING 2020

Contents

1 INTRODUCTION [10 points]

In this experiment we implemented a toy single cycle CPU with Verilog. First we designed a decoder a multiplexer and a register for using them in other parts. Then we combined the decoder and registers to design a register file. After that we implemented ALU with the given operations for each value. At last we combined the register file and the ALU.

2 MATERIALS AND METHODS [40 points]

2.1 PART 1

In the first part we implemented a 3:8 decoder a 8:1 multiplexer and an 8 bit register. In decoder has an enable input which works with positive logic. The 8-bit register has an enable and a negedge reset input.

Design Code

```
module decoder(
    input [2:0] in,
    input En,
    output reg [7:0] out
   );
always@(in or En)
    begin
        if (En) begin
         out=8'd0;
          case (in)
              3'b000: out[0]=1'b1;
              3'b001: out[1]=1'b1;
              3'b010: out[2]=1'b1;
              3'b011: out[3]=1'b1;
              3'b100: out[4]=1'b1;
              3'b101: out[5]=1'b1;
              3'b110: out[6]=1'b1;
              3'b111: out[7]=1'b1;
              default: out=8'd0;
          endcase
      end
else
out=8'd0;
end
endmodule

odule mux(
    input [7:0] In1,
    input [7:0] In2,
    input [7:0] In3,
    input [7:0] In4,
```

```
32      input [7:0] In5,
33      input [7:0] In6,
34      input [7:0] In7,
35      input [7:0] In8,
36      input [2:0] sel,
37      output reg [7:0] Out
38      );
39  always @ (In1 or In2 or In3 or In4 or In5 or In6 or In7 or In8 or sel)
40  begin
41   case (sel)
42    3'b000 : Out = In1;
43    3'b001 : Out = In2;
44    3'b010 : Out = In3;
45    3'b011 : Out = In4;
46    3'b100 : Out = In5;
47    3'b101 : Out = In6;
48    3'b110 : Out = In7;
49    3'b111 : Out = In8;
50    default : Out = 8'bx;
51   endcase
52
53  end
54
55  endmodule
56
57  module register(
58      input En,
59      input Reset,
60      input CLK,
61      input [7:0] Rin,
62      output reg [7:0] Rout
63      );
64
65  always @(posedge CLK or negedge Reset )
66  if (En & Reset)
67  Rout = Rin;
68  else if(~Reset)
69  Rout = 8'd0;
70  endmodule
```

Testbench Code

```
1   module decoder_tb;
2   wire [7:0] out;
3   reg en;
4   reg [2:0] in;
5   integer i;
6
7   decoder dut(in,en,out);
8
9   initial begin
10     for ( i=0; i<16; i=i+1)
11          begin
12             {en,in}  = i;
13              #1;
14          end
15  end
```

```
16  endmodule
17
18  module mux8to1_tb;
19   reg [2:0] Sel;
20   reg [7:0] In1;
21   reg [7:0] In2;
22   reg [7:0] In3;
23   reg [7:0] In4;
24   reg [7:0] In5;
25   reg [7:0] In6;
26   reg [7:0] In7;
27   reg [7:0] In8;
28   wire [7:0] Out;
29   reg [2:0] count = 3'd0;
30
31
32   mux uut(In1,In2,In3,In4,In5, In6,In7,In8,Sel,Out);
33
34   initial begin
35    Sel = 0;
36    In1 = 0;
37    In2 = 0;
38    In3 = 0;
39    In4 = 0;
40    In5 = 0;
41    In6 = 0;
42    In7 = 0;
43    In8 = 0;
44    #100;
45    Sel = 3'd0;
46    In1 = 8'd0;
47    In2 = 8'd1;
48    In3 = 8'd2;
49    In4 = 8'd3;
50    In5 = 8'd4;
51    In6 = 8'd5;
52    In7 = 8'd6;
53    In8 = 8'd7;
54    for (count = 0; count < 8; count = count + 1'b1)
55    begin
56     Sel = count;
57     #20;
58    end
59   end
60  endmodule
61
62  module register_Test();
63      reg clk;
64      reg en,reset;
65      reg [7:0] Rin;
66      wire [7:0] Rout;
67      register uut(en,reset,clk,Rin,Rout);
68      initial begin
69      reset=1; clk=0; #50;
70      reset=0; clk=1; #50;
71      clk=0;#50;
```

```
72      clk=1; en=1; reset =1;Rin=8'd0; #50;
73      clk=0; #50;
74      clk=1;  Rin=8'd1; #50;
75      clk=0; #50;
76      clk=1;  Rin=8'd2; #50;
77      clk=0; #50;
78      clk=1; Rin=8'd3; #50;
79      clk=0; #50;
80      reset=0; clk=1; #50;
81      clk=0; #50;
82      clk=1; reset=1; Rin=8'd4; #50;
83      clk=0; #50;
84      clk=1; Rin=8'd5; #50;
85      clk=0; #50;
86      end
87  endmodule
```

2.2 PART 2

In this part we implemented a register file with using 7 registers and a decoder. This way for the CPU this implementation chooses the given register with the help of the decoder and load the data input in the register with a clock signal. When reset goes from 1 to 0 the output is zero.

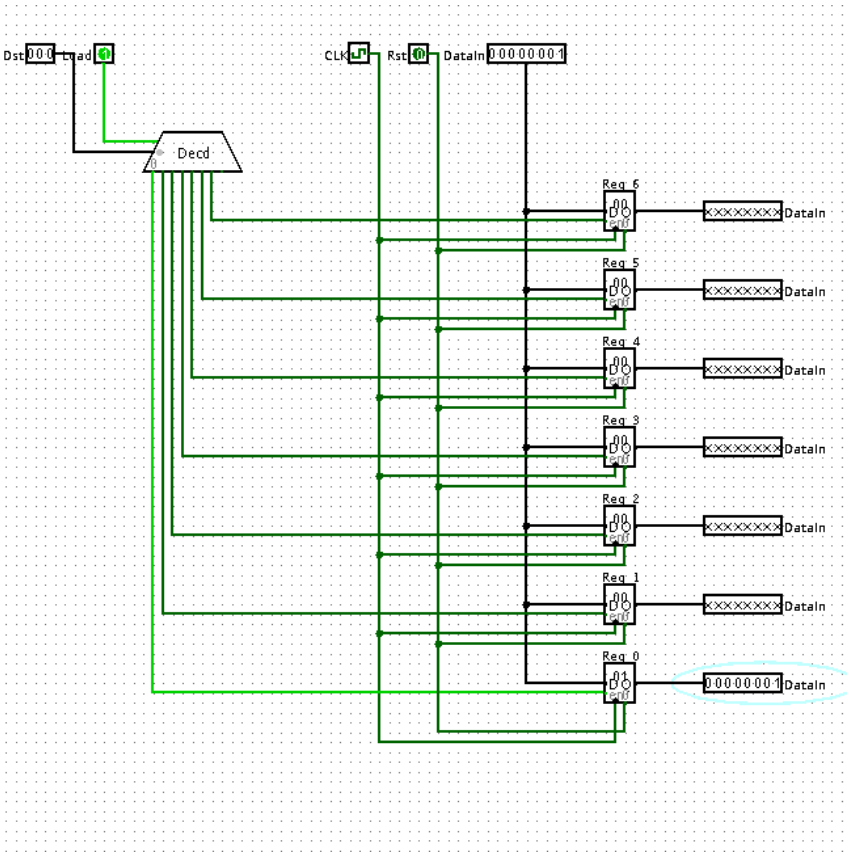

Figure 1: Logism Design for the Register File

Design Code

```
module part2(
    input [2:0] dst,
    input ld,
    input clk,
    input rst,
    input [7:0] dataIn,
    output  [7:0] regOut_0,
    output  [7:0] regOut_1,
    output  [7:0] regOut_2,
    output  [7:0] regOut_3,
    output  [7:0] regOut_4,
    output  [7:0] regOut_5,
    output  [7:0] regOut_6
    );
    wire [7:0] doutput;
    decoder dec(dst,ld,doutput);

    register r0(doutput[0],rst,clk,dataIn,regOut_0);
    register r1(doutput[1],rst,clk,dataIn,regOut_1);
    register r2(doutput[2],rst,clk,dataIn,regOut_2);
    register r3(doutput[3],rst,clk,dataIn,regOut_3);
    register r4(doutput[4],rst,clk,dataIn,regOut_4);
    register r5(doutput[5],rst,clk,dataIn,regOut_5);
    register r6(doutput[6],rst,clk,dataIn,regOut_6);
endmodule
```

Testbench Code

```
module part2_test();
    reg [2:0] dst;
    reg ld, clk, rst;
    reg [7:0] dataIn;
    wire [7:0] regOut_0,regOut_1,regOut_2,regOut_3, regOut_4, regOut_5, regOut_6;

    part2 uut(dst,ld,clk,rst,dataIn,regOut_0,regOut_1,regOut_2,regOut_3, regOut_4, regOut_5, regOut_6);
    initial begin
    rst=1; clk=0; #50;
    rst=0; clk=1; #50;
    clk=0; ld=1;#50;
    clk=1; rst =1;dataIn=8'b00000000; dst=3'b000; #50;
    clk=0; #50;
    clk=1; dataIn=8'b00000001;dst=3'b001; #50;
    clk=0; #50;
    clk=1; dataIn=8'b00000010;dst=3'b010; #50;
    clk=0; #50;
    rst=0; clk=1; #50;
    clk=0; #50;
    clk=1; rst=1; dataIn=8'b00000011;dst=3'b011; #50;
    clk=0; #50;
    clk=1; dataIn=8'b00000100;dst=3'b100; #50;
    clk=0; #50;
    clk=1; dataIn=8'b00000101;dst=3'b101; #50;
    clk=0; #50;
    clk=1; dataIn=8'b000000110;dst=3'b110; #50;
    clk=0; #50;
```

```
    end
endmodule
```

2.3 PART 3

In the third part we designed an Arithmetic logic Unit(ALU) that does the operations in the given table. We implemented the operations with using the operators dependent on the operation codes.When the output is zero there is a zero flag which becomes high. We implemented the zero flag like a D flip flop for storing the value. ALU have 2 inputs and an output with also an enable input.

Design Code

```
module Zero_flag(
    input D,
    output Q
    );
    assign Q=D;
endmodule

module part3(
    input [7:0] src1,
    input [7:0] src2,
    input en,
    input [2:0]Op,
    output reg zero,
    output reg [7:0] dst  );
    wire z;
     always@(*)
     begin
     if(en) begin
     case(Op)
        3'b000:
           dst = src1 + src2 ;
        3'b001:
           dst = src1 - src2 ;
        3'b010:
           dst = src1 << 1;
        3'b011:
           dst = src1;
        3'b100:
           dst = src1 & src2;
        3'b101:
           dst = src1 | src2;
        3'b110:
           dst = src1^src2;
        3'b111:
           dst = ~src1;
         default: dst =8'b0;
        endcase
    if(dst==8'b0)  zero=1;
    else zero=0;
    end
```

```
    end
    Zero_flag ze(zero,z);
endmodule
```

Testbench Code

```
module part3test();
    reg [7:0] src1, src2;
    reg en;
    reg [2:0]Op;
    wire zero;
    wire  [7:0] dst;
    part3 uut(src1, src2,en,Op,zero,dst);

    initial begin
    en=0; src1=8'd4;src2=8'd5; Op=3'd0;#50;
    en=1;src1=8'd4;src2=8'd5; Op=3'd0;#50; //add

    en=1;src1=8'd10;src2=8'd10; Op=3'd1;#50;

    en=1;src1=8'd46; Op=3'd2;#50;

    en=1;src1=8'd10; Op=3'd3;#50;

    en=1;src1=8'b01010101;src2=8'b11110000; Op=3'd4;#50;

    en=1;src1=8'b01010101;src2=8'b11110000; Op=3'd5;#50;

    en=1;src1=8'b01010101;src2=8'b11110000; Op=3'd6;#50;

    en=1;src1=8'b11111111;Op=3'd7;#50;
    end
```

2.4 PART 4

In the fourth part we combined the ALU and register file. There is an instruction code for giving the inputs and the destination register the operation and the Z field. The Z field and the Z flag are the inputs of a NAND gate and when the output is 1 the instruction will be executed. With these steps we combined them all together and made a single cycle CPU which reads the instruction code and make the necessary operations with ALU and give the result to the right register.

Design Code

```
module part4(
    input clk,
    input rst,
    input ld,
    input [20:0] instruction  ,
    output [7:0] out
    );

```

```
wire [7:0]dst;
wire [7:0] immediate; wire [7:0]src1; wire [7:0]src2; wire [2:0] op; wire z;
wire [7:0] regOut_0,regOut_1, regOut_2, regOut_3, regOut_4, regOut_5, regOut_6;

assign dst[0]=instruction[0];
assign dst[1]=instruction[1];
assign dst[2]=instruction[2];

assign src2[0]=instruction[3];
assign src2[1]=instruction[4];
assign src2[2]=instruction[5];

assign src1[0]=instruction[6];
assign src1[1]=instruction[7];
assign src1[2]=instruction[8];

assign immediate[0]=instruction[9];
assign immediate[1]=instruction[10];
assign immediate[2]=instruction[11];
assign immediate[3]=instruction[12];
assign immediate[4]=instruction[13];
assign immediate[5]=instruction[14];
assign immediate[6]=instruction[15];
assign immediate[7]=instruction[16];

assign op[0]=instruction[17];
assign op[1]=instruction[18];
assign op[2]=instruction[19];

assign z=instruction[20];
wire en,zero;
assign en=~(z&zero);

part2 regfile(dst,ld,clk,rst,out,regOut_0, regOut_1, regOut_2, regOut_3, regOut_4, regOut_5, regOut_6);

wire [7:0] mux1out;
mux m1(regOut_6,regOut_5,regOut_4,regOut_3,regOut_2,regOut_1,regOut_0,immediate,src1,mux1out);
wire [7:0] mux2out;
mux m2(regOut_6,regOut_5,regOut_4,regOut_3,regOut_2,regOut_1,regOut_0,immediate,src2,mux2out);

part3 alu(mux1out,mux2out,en,op,zero,out);
endmodule
```

Testbench Code

```
module part4test();
    reg clk,rst,ld;
    reg [20:0] instruction  ;
    wire [7:0] outAlu;
    part4 uut(clk,rst,ld,instruction,outAlu);

    initial begin
    clk=1; rst=1; #20;clk=1; rst=0; #20;clk=1; rst=1; #20;
    clk=0; ld=0; #20;
    clk=1; ld=1; instruction=21'b010100000000010101000; #50;
    clk=0; ld=0; #20;
    clk=1; ld=1; instruction=21'b001100101011111xxx110; #50;
```

```
13      clk=0; ld=0; #20;
14      clk=1; ld=1; instruction=21'b010010000000111010000; #50;
15      clk=0; ld=0; #20;
16      clk=1; ld=1; instruction=21'b101100000000111xxx010; #50;
17      clk=0; ld=0; #20;
18      clk=1; ld=1; instruction=21'b0011xxxxxxxx010xxx011; #50;
19      clk=0; ld=0; #20;
20      clk=1; ld=1; instruction=21'b000010000000111010000; #50;
21      clk=0; ld=0; #20;
22      clk=1; ld=1; instruction=21'b101100000000111xxx010; #50;
23      clk=0; ld=0; #20;
24      clk=1; ld=1; instruction=21'b0011xxxxxxxx010xxx011; #50;
25      end
26
27  endmodule
```

3 RESULTS [15 points]

3.1 Simulation Codes for Part 1

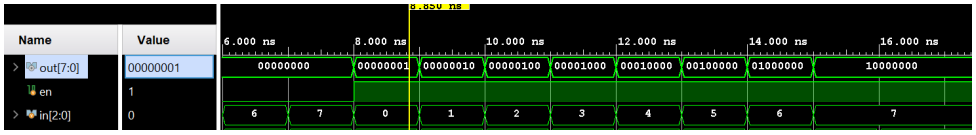

Figure 2: Simulation Code for 3:8 Decoder

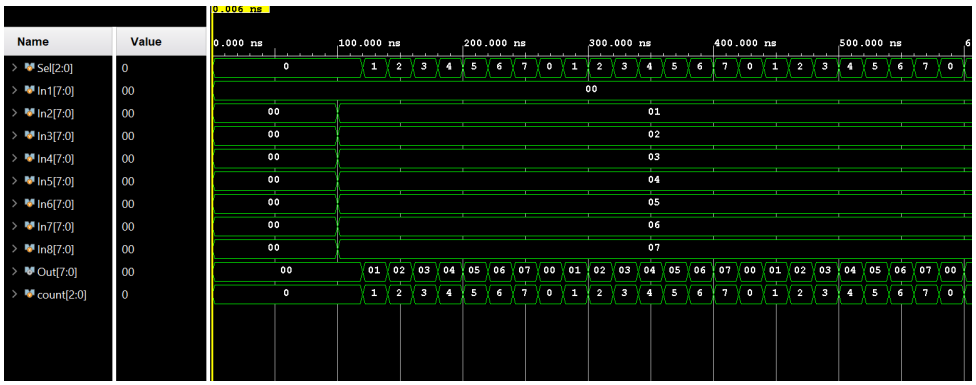

Figure 3: Simulation Code for 8:1 Multiplexer

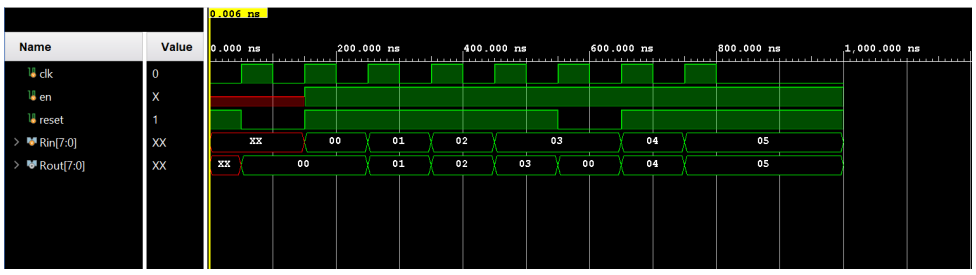

Figure 4: Simulation Code for 8-bit Register

In this part we observed the outputs of a decoder, multiplexer and a register as basic designs.

3.2 Simulation Codes for Part 2

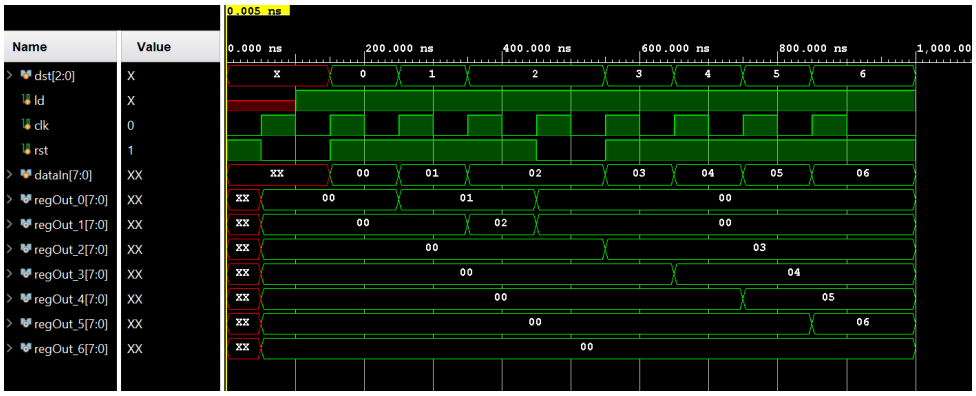

Figure 5: Simulation Code for Register File

In this part we observed the given inputs in the selected registers.

3.3 Simulation Codes for Part 3

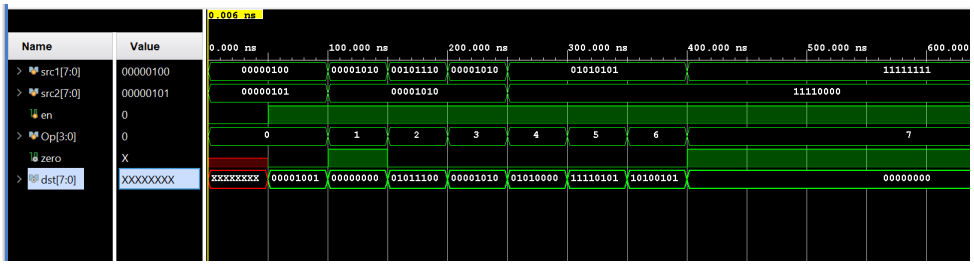

Figure 6: Simulation Code for ALU

In this part we observed an ALU with given operations. With 2 inputs we obtain an output with AND, OR, addition shift etc. operations.

3.4 Simulation Codes for Part 4

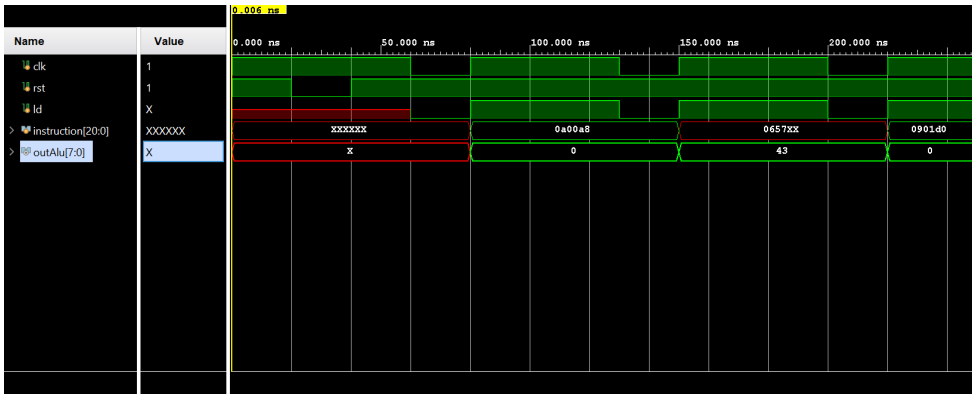

Figure 7: Simulation Code for the Single Cycle CPU

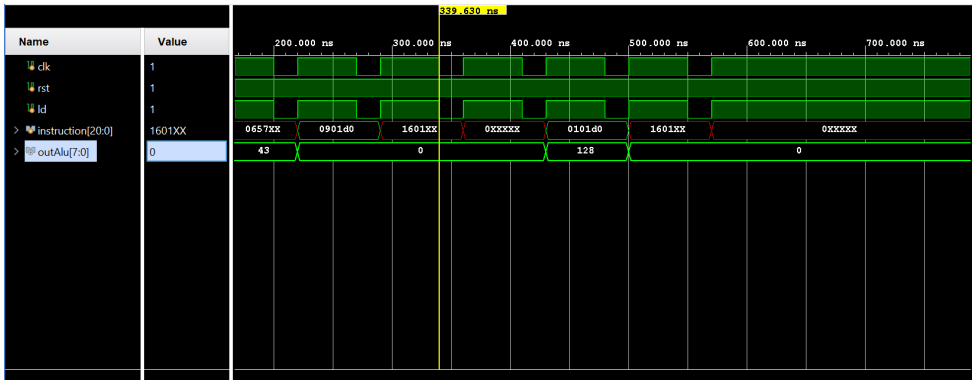

Figure 8: Simulation Code for the Single Cycle CPU

In this part we gave the given instruction codes and observed the outputs as it do the right operations with the immediate values or the values in the right registers.

4 DISCUSSION [25 points]

In this experiment we have implemented a single cycle CPU step by step. We used multiplexers and decoders to give the selected output and registers for storing the output. We use the arithmetic logic unit for making the arithmetical operations with the reg type parameters and when we combined the parts all together we obtained a CPU.

5 CONCLUSION [10 points]

In this experiment we learned how to make a basic CPU using Verilog and observed the results in the simulations. It was a bit hard for us to combine them all together without errors but at the end we had the correct results.