EE446 - Experiment 1

Preliminary Work

1.2.1) Constant Value Generator

```
module constantValueGenerator
    # (parameter W = 3, C = 9) (bus_out);

    output wire [W-1:0] bus_out;

    assign bus_out = C;
endmodule
```

1.2.2) Decoder

.v file:

```
module Decoder2to4 (input0, input1, o0, o1, o2, o3);

    input wire input0, input1;
    output wire o0, o1, o2, o3;

    assign o0 = ~input0 & ~input1;
    assign o1 = input0 & ~input1;
    assign o2 = ~input0 & input1;
    assign o3 = input0 & input1;

endmodule

```

testbench:

```
module Decoder2to4_tb ();
    wire ou0, ou1, ou2, ou3;
    reg in0, in1;
    Decoder2to4 DUT(in0, in1, ou0, ou1, ou2, ou3);
    reg [5:0] mem [3:0];
    integer i;

    initial begin
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/Decoder_tb_vector.txt" , mem);
        #1000;
    end
    always @(*) begin
    for(i=0;i<4;i=i+1) begin
        in0 = mem[i][4];
        in1 = mem[i][5];
        #10;
        if({ou3,ou2,ou1,ou0} == mem[i][3:0])
            $display ("No Error in %1d th row", i+1);
        else
            $display ("Error in %1d th row", i+1);
    end
    end
endmodule
```

testvector:

```
00_0001
01_0010
10_0100
11_1000
```

1.2.3)Multiplexers

2x1 .v file:

```
module Mux2to1
    #(parameter W = 3) (i0, i1, select, out);

    input [W-1:0] i0, i1;
    input select;
    output [W-1:0] out;

    assign out = (select) ? i1 : i0;

endmodule
```

2x1 testbench:

```
module Mux2to1_tb
    #(parameter W = 2) ();

    reg [W-1:0] i0, i1;
    reg select;
    wire [W-1:0] out;

    reg [6:0] mem [1:0];

    Mux2to1 DUT(i0,i1,select,out);
    integer i;

    initial begin
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/Mux2to1_tb_vector.txt" , mem);
        #1000;
    end
    always @(*) begin
    for(i=0;i<2;i=i+1) begin
        i0 = mem[i][4:3];
        i1 = mem[i][6:5];
        select = mem[i][2];
        #10;
        if(out == mem[i][1:0])
            $display ("No Error in %1d th row", i+1);
        else
            $display ("Error in %1d th row", i+1);
    end
    end
endmodule
```

2x1 test vector:

```
1. //in1_in0_select_out
2. 10_11_0_11
3. 10_11_1_10
```

4x1 .v file:

```
1. module Mux4to1
2.     #(parameter W = 2) (in0,in1,in2,in3,select,out);
3.
4.     input wire [W-1:0] in0,in1,in2,in3;
5.     input wire [1:0] select;
6.     output reg [W-1:0] out;
7.
8.     always @* begin
9.         case(select)
10.            2'b00: out=in0;
11.            2'b01: out=in1;
12.            2'b10: out=in2;
13.            2'b11: out=in3;
14.        endcase
15.    end
16.
17. endmodule
```

4x1 testbench:

```
1. module Mux4to1_tb
2.     #(parameter W = 2) ();
3.
4.     reg [W-1:0] i0, i1, i2, i3;
5.     reg [1:0] select;
6.     wire [W-1:0] out;
7.
8.     reg [11:0] mem [3:0];
9.
10.    Mux4to1 DUT(i0,i1,i2,i3,select,out);
11.    integer i;
12.
13.    initial begin
14.        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/Mux4to1_tb
    _vector.txt" , mem);
15.        #1000;
16.    end
17.    always @(*) begin
18.    for(i=0;i<4;i=i+1) begin
19.        i0 = mem[i][11:10];
20.        i1 = mem[i][9:8];
21.        i2 = mem[i][7:6];
22.        i3 = mem[i][5:4];
23.        select = mem[i][3:2];
24.        #10;
25.        if(out == mem[i][1:0])
26.            $display ("No Error in %1d th row", i+1);
27.        else
28.            $display ("Error in %1d th row", i+1);
29.    end
30.    end
31. endmodule
```

4x1 test vector:

```
1. //in0_in1_in2_in3_select_out
2. 00_01_10_11_00_00
3. 00_01_10_11_01_01
4. 00_01_10_11_10_10
5. 00_01_10_11_11_11
```

1.2.4) ALU

.v file:

```
1. module ALU
2.     #(parameter W = 5) (SrcA, SrcB, Control, Result, CO, OVF, N, Z);
3.
4.     //W-bit Inputs
5.     input wire [W-1:0] SrcA, SrcB;
6.     //3 bit Control Signal
7.     input [2:0] Control;
8.     //N-bit Result
9.     output reg [W-1:0] Result;
10.    //Status Flags
11.    output reg CO, OVF, N, Z;
12.
13.    reg [W-1:0] temp_add;
14.
15.
16.    always @(*) begin
17.        case(Control)
18.            3'b000: begin //Addition
19.                        {CO,Result} = {1'b0,SrcA} + {1'b0,SrcB};
20.                        if((SrcA[W-1] == SrcB[W-1]) && (Result[W-1]!=SrcA[W-
   1])) OVF=1'b1;
21.                        else OVF=1'b0;
22.                        if(Result[W-1] == 1'b1) N=1'b1;
23.                        else N=1'b0; //Negative flag
24.                        N = N & (~OVF); //When OVF=1 N should be 0
25.                        if(Result == 0) Z=1'b1;
26.                        else Z=1'b0;
27.            end//End of Addition case
28.            3'b001: begin //SubtractionAB
29.                        {CO,Result} = {1'b0,SrcA} - {1'b0,SrcB};
30.                        if((SrcA[W-1] == ~SrcB[W-1]) && (Result[W-1]!=SrcA[W-
   1])) OVF=1'b1;
31.                        else OVF=1'b0;
32.                        //If Result is a different sign then the inputs then overflo
   w.
33.                        if(Result[W-1] == 1'b1) N=1'b1;
34.                        else N=1'b0; //Negative flag
35.                        N = N & (~OVF); //When OVF=1 N should be 0
36.                        if(Result == 0) Z=1'b1;
37.                        else Z=1'b0;
38.            end // End of SubtractionAB
39.            3'b010: begin //SubtractionBA
40.                        {CO,Result} = {1'b0,SrcB} - {1'b0,SrcA};
41.                        if((~SrcA[W-1] == SrcB[W-1]) && (Result[W-1]!=SrcB[W-
   1])) OVF=1'b1;
42.                        else OVF=1'b0;
43.                        if(Result[W-1] == 1'b1) N=1'b1;
44.                        else N=1'b0; //Negative flag
45.                        N = N & (~OVF); //When OVF=1 N should be 0
46.                        if(Result == 0) Z=1'b1;
```

```
                        else Z=1'b0;
            end // End of SubtractionBA
            3'b011: begin //BitClear
                        Result = SrcA & (~SrcB);
                        if(Result == 0) Z=1'b1;
                        else Z=1'b0;
                        if(Result[W-1] == 1'b1) N=1'b1;
                        else N=1'b0; //Negative flag
                        CO=1'b0;
                        OVF=1'b0;
            end//End of BitClear
            3'b100: begin //AND
                        Result = SrcA & SrcB;
                        if(Result == 0) Z=1'b1;
                        else Z=1'b0;
                        if(Result[W-1] == 1'b1) N=1'b1;
                        else N=1'b0; //Negative flag
                        CO=1'b0;
                        OVF=1'b0;
            end // End of AND
            3'b101: begin //OR
                        Result = SrcA | SrcB;
                        if(Result == 0) Z=1'b1;
                        else Z=1'b0;
                        if(Result[W-1] == 1'b1) N=1'b1;
                        else N=1'b0; //Negative flag
                        CO=1'b0;
                        OVF=1'b0;
            end // End of OR
            3'b110: begin //EXOR
                        Result = SrcA ^ SrcB;
                        if(Result == 0) Z=1'b1;
                        else Z=1'b0;
                        if(Result[W-1] == 1'b1) N=1'b1;
                        else N=1'b0; //Negative flag
                        CO=1'b0;
                        OVF=1'b0;
            end //End of EXOR
            3'b111: begin //EXNOR
                        Result = ~(SrcA ^ SrcB);
                        if(Result == 0) Z=1'b1;
                        else Z=1'b0;
                        if(Result[W-1] == 1'b1) N=1'b1;
                        else N=1'b0; //Negative flag
                        CO=1'b0;
                        OVF=1'b0;
            end //End of EXNOR
            endcase
    end

endmodule
```

testbench:

```
module ALU_tb
    #(parameter W=5) ();

    //W-bit Inputs
    reg [W-1:0] SrcA, SrcB;
    //3 bit Control Signal
    reg [2:0] Control;
    //N-bit Result
    wire [W-1:0] Result;
    //Status Flags
```

```
    wire CO, OVF, N, Z;

    reg [21:0] mem [12:0];

    ALU DUT (SrcA, SrcB, Control, Result, CO, OVF, N, Z);

    integer i;

    initial begin
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/ALU_tb_vector.txt" , mem);
        #1000;
    end
    always @*
    begin
            for(i=0;i<13;i=i+1) begin
            SrcA = mem[i][21:17];
            SrcB = mem[i][16:12];
            Control = mem[i][11:9];
            #10;
            if({Result,CO,OVF,N,Z} == mem[i][8:0])
                $display ("No Error in %1d th row", i+1);
            else
                $display ("Error in %1d th row", i+1);
            end
    end
endmodule
```

ALU testvector:

```
//input1_input2_Control_Result_CO,OVF,N,Z
00111_00101_000_01100_0000
00111_00101_001_00010_0000
00111_00101_010_11110_1010
00111_00101_011_00010_0000
00111_00101_100_00101_0000
00111_00101_101_00111_0000
00111_00101_110_00010_0000
00111_00101_111_11101_0010
//Zero detection test
00111_00111_001_00000_0001
//Overflow Detection Test in Addition
01101_01001_000_10110_0100
10011_10101_000_01000_1100
01110_11000_000_00110_1000
//Overflow Detection Test in Subtraction
01101_10011_001_11010_1100
```

1.2.5) Registers:

Simple Register .v file:

```
module Reg_Simple
    #(parameter W=5) (clk, reset, data, out);

    input wire clk,reset;
    input wire [W-1:0] data;
    output reg [W-1:0] out;

    always @(posedge clk) begin
        if(reset==1'b1) out <= 0;
```

```
        else out <= data;
    end

endmodule
```

Simple Register testbench:

```
module Reg_Simple_tb
    #(parameter W=5)();

    reg clk,reset;
    reg [W-1:0] data;
    wire [W-1:0] out;

    reg [10:0] mem [3:0];

    Reg_Simple DUT (clk, reset, data, out);

    integer i,j;

    initial begin
        clk=0;
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/Reg_Simple_tb_vector.txt" , mem);
        #1000;
    end
    always @* begin
            for(j=0;j<100;j=j+1) begin
                clk = clk ^ 1'b1;
                #5;
                //For now the frequency is too high but it's just for simulation
            end
    end
    always @*
    begin
            for(i=0;i<4;i=i+1) begin
            reset = mem[i][10];
            data = mem[i][9:5];
            #13;
            if(out == mem[i][4:0])
                $display ("No Error in %1d th row", i+1);
            else
                $display ("Error in %1d th row", i+1);
            end
    end
endmodule
```

Simple Register testvector:

```
//reset_data_out
0_11010_11010
0_10011_10011
0_00001_00001
1_11010_00000
```

Register with WE .v file:

```
module Reg_WE
    #(parameter W=5) (clk, reset, we, data, out);

    input wire clk,reset,we;
```

```
    input wire [W-1:0] data;
    output reg [W-1:0] out;

    always @(posedge clk) begin
        if(reset==1'b1) out <= 0;
        else begin
            if(we == 1'b1) out <= data;
        end
    end

endmodule
```

Reg_WE testbench:

```
module Reg_WE_tb
    #(parameter W=5)();

    reg clk,reset,we;
    reg [W-1:0] data;
    wire [W-1:0] out;

    reg [11:0] mem [5:0];

    Reg_WE DUT (clk, reset, we, data, out);

    integer i,j;

    initial begin
        clk=0;
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/Reg_WE_tb_vector.txt" , mem);
        #1000;
    end
    always @* begin
            for(j=0;j<100;j=j+1) begin
                clk = clk ^ 1'b1;
                #5;
                //For now the frequency is too high but it's just for simulation
            end
    end
    always @*
    begin
            for(i=0;i<6;i=i+1) begin
            reset = mem[i][11];
            we = mem[i][10];
            data = mem[i][9:5];
            #10;
            if(out == mem[i][4:0])
                $display ("No Error in %1d th row", i+1);
            else
                $display ("Error in %1d th row", i+1);
            end
    end
endmodule
```

Reg_WE testvector:

```
//reset_we_data_out
0_1_11010_11010
0_0_10011_11010
0_1_00001_00001
1_0_11010_00000
```

```
0_1_11011_11011
1_1_10011_00000
```

Shift Register .v file:

```
module Reg_Shift
    #(parameter W = 5) (clk, reset, ps_select, lr_select, data, input_left, input_right, out);

    input wire clk,reset,ps_select, lr_select;
    input wire [W-1:0] data;
    input wire input_left, input_right;

    output reg [W-1:0] out;

    always @(posedge clk) begin
        if(reset==1) out<=0; //Synchronous reset
        else begin
            if(ps_select==1) begin
                out <= data; //If Parallel/Serial select=1 then load parallel data
            end
            else begin //Otherwise check for left/right shif signal
                if(lr_select == 1) begin //if lr=1 shift right
                    out <= {input_left, out[W-1:1]};
                end
                else begin //lr=0 so shift left
                    out <= {out[W-2:0],input_right};
                end
            end
        end
    end
endmodule
```

Shift Register testbench:

```
module Reg_Shift_tb
    #(parameter W=5)();

    reg clk,reset,ps_select,lr_select,input_left,input_right;
    reg [W-1:0] data;
    wire [W-1:0] out;

    reg [14:0] mem [6:0];

    Reg_Shift DUT (clk, reset, ps_select, lr_select, data, input_left, input_right, out);

    integer i,j;

    initial begin
        clk=0;
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/lab1_registerfile_tb_vector.txt" , mem);
        #1000;
    end
    always @* begin
            for(j=0;j<100;j=j+1) begin
                clk = clk ^ 1'b1;
                #5;
                //For now the frequency is too high but it's just for simulation
            end
    end
```

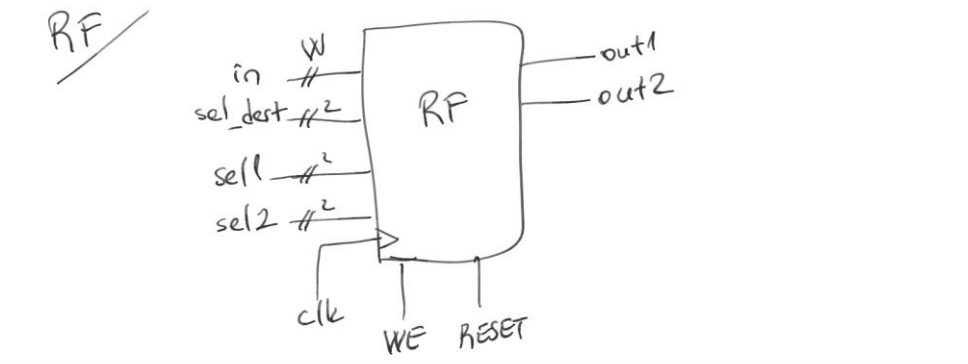
```
    always @*
    begin
            for(i=0;i<7;i=i+1) begin
            reset = mem[i][14];
            ps_select = mem[i][13];
            lr_select = mem[i][12];
            input_left = mem[i][11];
            input_right = mem[i][10];
            data = mem[i][9:5];
            #10;
            if(out == mem[i][4:0])
                $display ("No Error in %1d th row", i+1);
            else
                $display ("Error in %1d th row", i+1);
            end
    end
endmodule
```

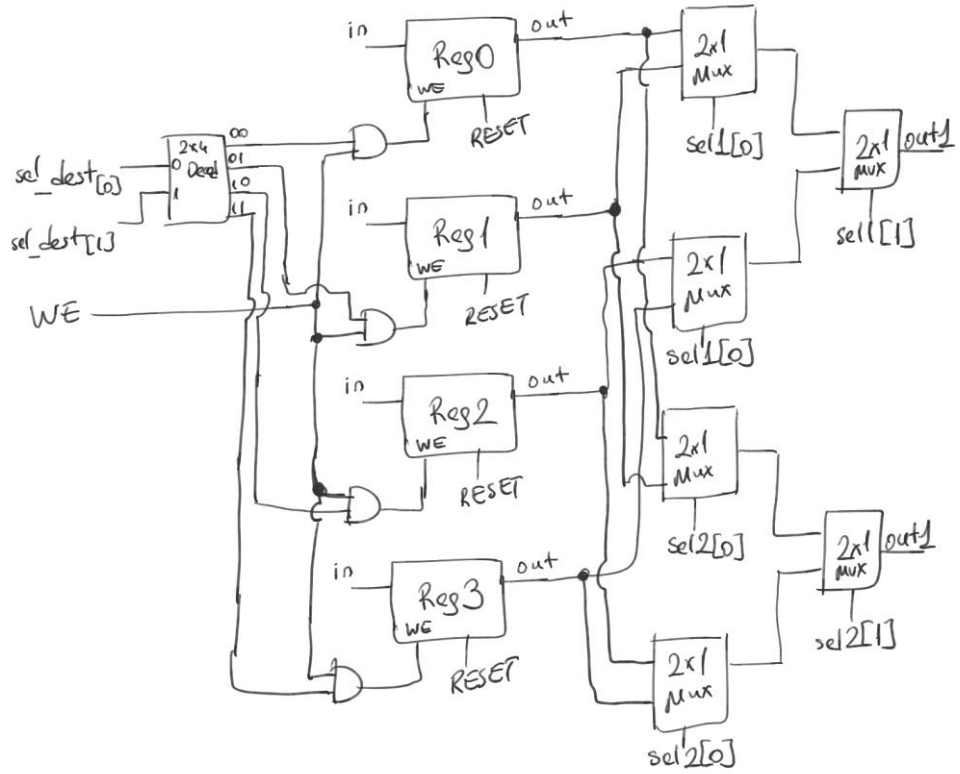
Reg_Shift testvector:

```
//reset_psselect,lrselect_inputleft,inputright_data_out
0_10_11_11010_11010
0_01_10_00011_11101
0_01_00_00101_01110
1_11_01_11010_00000
0_11_10_11011_11011
0_00_00_10011_10110
0_00_01_11111_01101
```

1.3)Register File

Hand drawn schematics:





.v file:

```
1.  // Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
2.  // Your use of Altera Corporation's design tools, logic functions
3.  // and other software and tools, and its AMPP partner logic
4.  // functions, and any output files from any of the foregoing
5.  // (including device programming or simulation files), and any
6.  // associated documentation or information are expressly subject
7.  // to the terms and conditions of the Altera Program License
8.  // Subscription Agreement, the Altera Quartus Prime License Agreement,
9.  // the Altera MegaCore Function License Agreement, or other
10. // applicable license agreement, including, without limitation,
11. // that your use is for the sole purpose of programming logic
12. // devices manufactured by Altera and sold by Altera or its
13. // authorized distributors.  Please refer to the applicable
14. // agreement for further details.
15.
16. // PROGRAM      "Quartus Prime"
17. // VERSION      "Version 16.0.0 Build 211 04/27/2016 SJ Lite Edition"
18. // CREATED      "Tue Mar 27 03:01:29 2018"
19.
20. module lab1_registerfile
21. #(parameter W = 8)
22. (
23.     we,
24.     clk,
25.     reset,
26.     in,
27.     sel1,
28.     sel2,
29.     sel_dest,
30.     out1,
31.     out2
32. );
33.
34.
35. input wire  we;
36. input wire  clk;
37. input wire  reset;
38. input wire  [W-1:0] in;
39. input wire  [W-1:0] sel1;
40. input wire  [1:0] sel2;
41. input wire  [1:0] sel_dest;
42. output wire [W-1:0] out1;
43. output wire [W-1:0] out2;
44.
45. wire    SYNTHESIZED_WIRE_0;
46. wire    SYNTHESIZED_WIRE_1;
47. wire    SYNTHESIZED_WIRE_2;
48. wire    SYNTHESIZED_WIRE_3;
49. wire    SYNTHESIZED_WIRE_4;
50. wire    SYNTHESIZED_WIRE_5;
51. wire    [W-1:0] SYNTHESIZED_WIRE_20;
52. wire    [W-1:0] SYNTHESIZED_WIRE_21;
53. wire    [W-1:0] SYNTHESIZED_WIRE_22;
54. wire    [W-1:0] SYNTHESIZED_WIRE_23;
55. wire    [W-1:0] SYNTHESIZED_WIRE_10;
56. wire    [W-1:0] SYNTHESIZED_WIRE_11;
57. wire    [W-1:0] SYNTHESIZED_WIRE_14;
58. wire    [W-1:0] SYNTHESIZED_WIRE_15;
59. wire    SYNTHESIZED_WIRE_18;
60. wire    SYNTHESIZED_WIRE_19;
61.
62.
```

```
63.
64.
65.
66. Decoder2to4 b2v_inst(
67.     .input0(sel_dest[0]),
68.     .input1(sel_dest[1]),
69.     .o0(SYNTHESIZED_WIRE_2),
70.     .o1(SYNTHESIZED_WIRE_3),
71.     .o2(SYNTHESIZED_WIRE_4),
72.     .o3(SYNTHESIZED_WIRE_5));
73.
74.
75. Reg_WE  b2v_inst10(
76.     .clk(clk),
77.     .reset(reset),
78.     .we(SYNTHESIZED_WIRE_0),
79.     .data(in),
80.     .out(SYNTHESIZED_WIRE_22));
81.     defparam    b2v_inst10.W = W;
82.
83.
84. Reg_WE  b2v_inst11(
85.     .clk(clk),
86.     .reset(reset),
87.     .we(SYNTHESIZED_WIRE_1),
88.     .data(in),
89.     .out(SYNTHESIZED_WIRE_23));
90.     defparam    b2v_inst11.W = W;
91.
92. assign  SYNTHESIZED_WIRE_18 = we & SYNTHESIZED_WIRE_2;
93.
94. assign  SYNTHESIZED_WIRE_19 = we & SYNTHESIZED_WIRE_3;
95.
96. assign  SYNTHESIZED_WIRE_0 = we & SYNTHESIZED_WIRE_4;
97.
98. assign  SYNTHESIZED_WIRE_1 = we & SYNTHESIZED_WIRE_5;
99.
100.
101.       Mux2to1 b2v_inst2(
102.           .select(sel1[0]),
103.           .i0(SYNTHESIZED_WIRE_20),
104.           .i1(SYNTHESIZED_WIRE_21),
105.           .out(SYNTHESIZED_WIRE_10));
106.           defparam    b2v_inst2.W = W;
107.
108.
109.       Mux2to1 b2v_inst3(
110.           .select(sel1[0]),
111.           .i0(SYNTHESIZED_WIRE_22),
112.           .i1(SYNTHESIZED_WIRE_23),
113.           .out(SYNTHESIZED_WIRE_11));
114.           defparam    b2v_inst3.W = W;
115.
116.
117.       Mux2to1 b2v_inst4(
118.           .select(sel1[1]),
119.           .i0(SYNTHESIZED_WIRE_10),
120.           .i1(SYNTHESIZED_WIRE_11),
121.           .out(out1));
122.           defparam    b2v_inst4.W = W;
123.
124.
125.       Mux2to1 b2v_inst5(
126.           .select(sel2[0]),
127.           .i0(SYNTHESIZED_WIRE_20),
128.           .i1(SYNTHESIZED_WIRE_21),
```

```
129.            .out(SYNTHESIZED_WIRE_14));
130.            defparam    b2v_inst5.W = W;
131.
132.
133.        Mux2to1 b2v_inst6(
134.            .select(sel2[1]),
135.            .i0(SYNTHESIZED_WIRE_14),
136.            .i1(SYNTHESIZED_WIRE_15),
137.            .out(out2));
138.            defparam    b2v_inst6.W = W;
139.
140.
141.        Mux2to1 b2v_inst7(
142.            .select(sel2[0]),
143.            .i0(SYNTHESIZED_WIRE_22),
144.            .i1(SYNTHESIZED_WIRE_23),
145.            .out(SYNTHESIZED_WIRE_15));
146.            defparam    b2v_inst7.W = W;
147.
148.
149.        Reg_WE  b2v_inst8(
150.            .clk(clk),
151.            .reset(reset),
152.            .we(SYNTHESIZED_WIRE_18),
153.            .data(in),
154.            .out(SYNTHESIZED_WIRE_20));
155.            defparam    b2v_inst8.W = W;
156.
157.
158.        Reg_WE  b2v_inst9(
159.            .clk(clk),
160.            .reset(reset),
161.            .we(SYNTHESIZED_WIRE_19),
162.            .data(in),
163.            .out(SYNTHESIZED_WIRE_21));
164.            defparam    b2v_inst9.W = W;
165.
166.
167.        // Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
168.        // Your use of Altera Corporation's design tools, logic functions
169.        // and other software and tools, and its AMPP partner logic
170.        // functions, and any output files from any of the foregoing
171.        // (including device programming or simulation files), and any
172.        // associated documentation or information are expressly subject
173.        // to the terms and conditions of the Altera Program License
174.        // Subscription Agreement, the Altera Quartus Prime License Agreement,
175.        // the Altera MegaCore Function License Agreement, or other
176.        // applicable license agreement, including, without limitation,
177.        // that your use is for the sole purpose of programming logic
178.        // devices manufactured by Altera and sold by Altera or its
179.        // authorized distributors.  Please refer to the applicable
180.        // agreement for further details.
181.
182.        // PROGRAM      "Quartus Prime"
183.        // VERSION      "Version 16.0.0 Build 211 04/27/2016 SJ Lite Edition"
184.        // CREATED      "Tue Mar 27 03:01:29 2018"
185.
186.        module lab1_registerfile
187.        #(parameter W = 8)
188.        (
189.            we,
190.            clk,
191.            reset,
192.            in,
193.            sel1,
194.            sel2,
```

```
195.            sel_dest,
196.            out1,
197.            out2
198.        );
199.
200.
201.        input wire  we;
202.        input wire  clk;
203.        input wire  reset;
204.        input wire  [W-1:0] in;
205.        input wire  [1:0] sel1;
206.        input wire  [1:0] sel2;
207.        input wire  [1:0] sel_dest;
208.        output wire [W-1:0] out1;
209.        output wire [W-1:0] out2;
210.
211.        wire    SYNTHESIZED_WIRE_0;
212.        wire    SYNTHESIZED_WIRE_1;
213.        wire    SYNTHESIZED_WIRE_2;
214.        wire    SYNTHESIZED_WIRE_3;
215.        wire    SYNTHESIZED_WIRE_4;
216.        wire    SYNTHESIZED_WIRE_5;
217.        wire    [W-1:0] SYNTHESIZED_WIRE_20;
218.        wire    [W-1:0] SYNTHESIZED_WIRE_21;
219.        wire    [W-1:0] SYNTHESIZED_WIRE_22;
220.        wire    [W-1:0] SYNTHESIZED_WIRE_23;
221.        wire    [W-1:0] SYNTHESIZED_WIRE_10;
222.        wire    [W-1:0] SYNTHESIZED_WIRE_11;
223.        wire    [W-1:0] SYNTHESIZED_WIRE_14;
224.        wire    [W-1:0] SYNTHESIZED_WIRE_15;
225.        wire    SYNTHESIZED_WIRE_18;
226.        wire    SYNTHESIZED_WIRE_19;
227.
228.
229.
230.
231.
232.        Decoder2to4 b2v_inst(
233.            .input0(sel_dest[0]),
234.            .input1(sel_dest[1]),
235.            .o0(SYNTHESIZED_WIRE_2),
236.            .o1(SYNTHESIZED_WIRE_3),
237.            .o2(SYNTHESIZED_WIRE_4),
238.            .o3(SYNTHESIZED_WIRE_5));
239.
240.
241.        Reg_WE  b2v_inst10(
242.            .clk(clk),
243.            .reset(reset),
244.            .we(SYNTHESIZED_WIRE_0),
245.            .data(in),
246.            .out(SYNTHESIZED_WIRE_22));
247.            defparam    b2v_inst10.W = W;
248.
249.
250.        Reg_WE  b2v_inst11(
251.            .clk(clk),
252.            .reset(reset),
253.            .we(SYNTHESIZED_WIRE_1),
254.            .data(in),
255.            .out(SYNTHESIZED_WIRE_23));
256.            defparam    b2v_inst11.W = W;
257.
258.        assign  SYNTHESIZED_WIRE_18 = we & SYNTHESIZED_WIRE_2;
259.
260.        assign  SYNTHESIZED_WIRE_19 = we & SYNTHESIZED_WIRE_3;
```

```
261.
262.        assign  SYNTHESIZED_WIRE_0 = we & SYNTHESIZED_WIRE_4;
263.
264.        assign  SYNTHESIZED_WIRE_1 = we & SYNTHESIZED_WIRE_5;
265.
266.
267.        Mux2to1 b2v_inst2(
268.            .select(sel1[0]),
269.            .i0(SYNTHESIZED_WIRE_20),
270.            .i1(SYNTHESIZED_WIRE_21),
271.            .out(SYNTHESIZED_WIRE_10));
272.            defparam    b2v_inst2.W = W;
273.
274.
275.        Mux2to1 b2v_inst3(
276.            .select(sel1[0]),
277.            .i0(SYNTHESIZED_WIRE_22),
278.            .i1(SYNTHESIZED_WIRE_23),
279.            .out(SYNTHESIZED_WIRE_11));
280.            defparam    b2v_inst3.W = W;
281.
282.
283.        Mux2to1 b2v_inst4(
284.            .select(sel1[1]),
285.            .i0(SYNTHESIZED_WIRE_10),
286.            .i1(SYNTHESIZED_WIRE_11),
287.            .out(out1));
288.            defparam    b2v_inst4.W = W;
289.
290.
291.        Mux2to1 b2v_inst5(
292.            .select(sel2[0]),
293.            .i0(SYNTHESIZED_WIRE_20),
294.            .i1(SYNTHESIZED_WIRE_21),
295.            .out(SYNTHESIZED_WIRE_14));
296.            defparam    b2v_inst5.W = W;
297.
298.
299.        Mux2to1 b2v_inst6(
300.            .select(sel2[1]),
301.            .i0(SYNTHESIZED_WIRE_14),
302.            .i1(SYNTHESIZED_WIRE_15),
303.            .out(out2));
304.            defparam    b2v_inst6.W = W;
305.
306.
307.        Mux2to1 b2v_inst7(
308.            .select(sel2[0]),
309.            .i0(SYNTHESIZED_WIRE_22),
310.            .i1(SYNTHESIZED_WIRE_23),
311.            .out(SYNTHESIZED_WIRE_15));
312.            defparam    b2v_inst7.W = W;
313.
314.
315.        Reg_WE  b2v_inst8(
316.            .clk(clk),
317.            .reset(reset),
318.            .we(SYNTHESIZED_WIRE_18),
319.            .data(in),
320.            .out(SYNTHESIZED_WIRE_20));
321.            defparam    b2v_inst8.W = W;
322.
323.
324.        Reg_WE  b2v_inst9(
325.            .clk(clk),
326.            .reset(reset),
```

```
            .we(SYNTHESIZED_WIRE_19),
            .data(in),
            .out(SYNTHESIZED_WIRE_21));
            defparam    b2v_inst9.W = W;


        endmodule

endmodule
```

testbench:

```
module lab1_registerfile_tb
    #(parameter W = 3) ();

    reg we,clk,reset;
    reg [1:0] sel1,sel2,sel_dest;
    reg [W-1:0] in;
    wire [W-1:0] out1,out2;

    reg [16:0] mem [6:0];

    lab1_registerfile DUT (we, clk, reset,in,sel1,sel2,sel_dest,out1,out2);

    integer i,j;

    initial begin
        clk=0;
        $readmemb("C:/Users/AhmetSalih/Desktop/EE446_Desktop/test_vectors/lab1_registerfile_tb_vector.txt" , mem);
        #1000;
    end
    always @* begin
            for(j=0;j<100;j=j+1) begin
                clk = clk ^ 1'b1;
                #5;
                //For now the frequency is too high but it's just for simulation
            end
    end
    always @*
    begin
            for(i=0;i<7;i=i+1) begin
            we = mem[i][16];
            reset = mem[i][15];
            in = mem[i][14:12];
            sel1 = mem[i][11:10];
            sel2 = mem[i][9:8];
            sel_dest = mem[i][7:6];
            #10;
            if({out1,out2} == mem[i][5:0])
                $display ("No Error in %1d th row", i+1);
            else
                $display ("Error in %1d th row", i+1);
            end
    end
endmodule
```

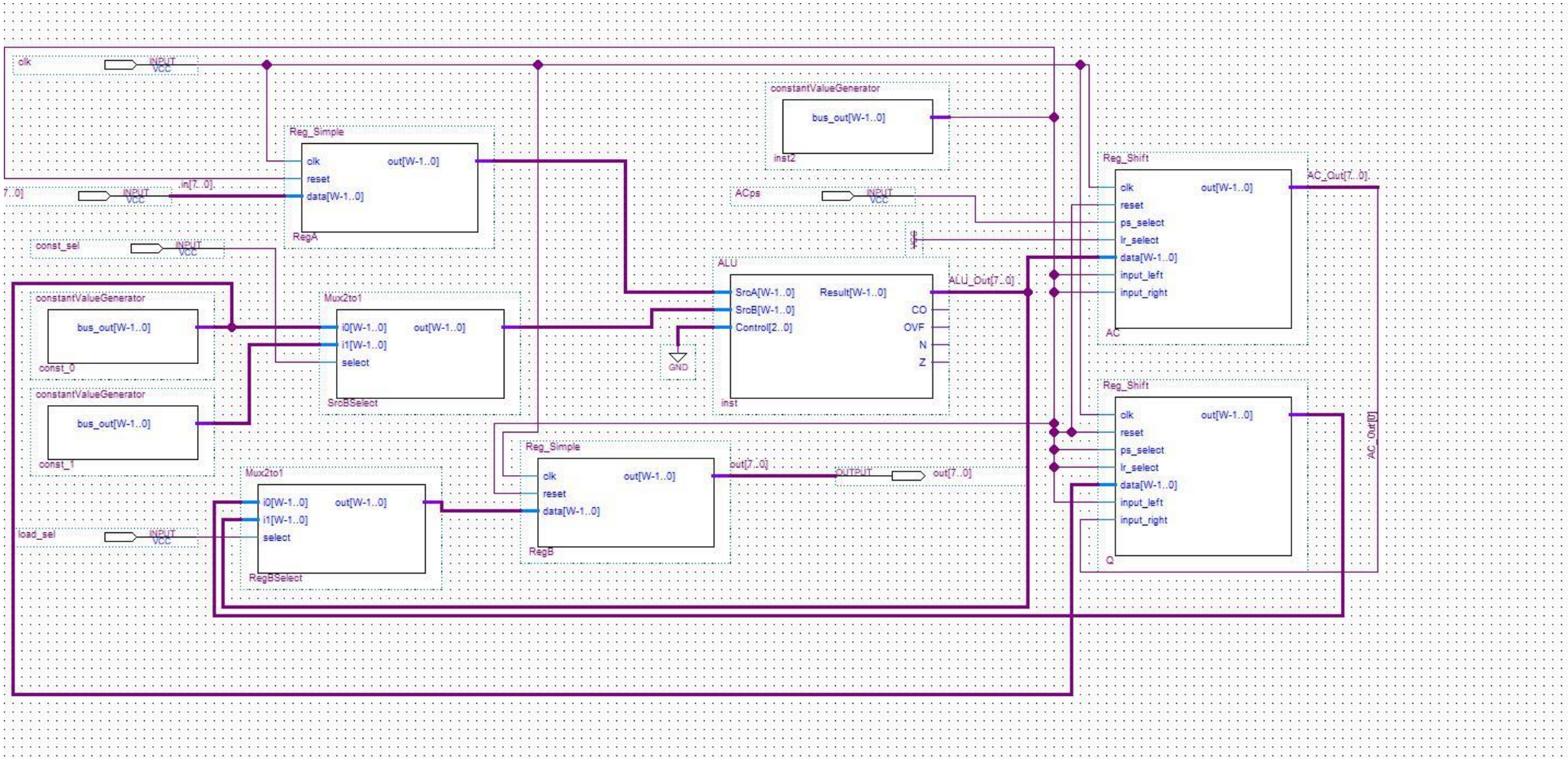
testvector:

```
//we_reset_in_sel1_sel2_sel_dest_out1_out2);
1_1_101_10_11_10_000_000
1_0_101_10_00_00_000_101
```

```
1_0_111_11_01_01_000_111
1_0_001_11_10_10_000_001
1_0_011_11_11_11_011_011
1_0_010_01_10_11_111_001
1_1_001_10_11_01_000_000
```

1.4) Datapath

Schematics:



Q1) There are 3 control pins namely:

-const_sel (For the 2x1 MUX that selects the constant that goes to ALU)

-load_sel (For the 2x1 MUX that decides the input of RegB

-ACps (AC register's parallel/serial select. First it has to load parallel then do serial computation)

Q2) There are 13 control signals in my architecture, including the reset signals. However 8 of those signals does not change over time. They also do not have to be changed for different instructions. For example ALU always does add operation so ALU's control signal is always 00, we also do not do any resetting so resets are always zero etc.

Q3) The 3 control pins is the minimum you can get. I have already reduced the number by inserting constant zeros and ones to unchanging pins. However,

- const_sel has to change for different instructions. LOAD and REVERSE LOAD requires 0 as constant, whereas INCREMENTED LOAD requires 1.
- load_sel has to change for different instructions. For LOAD and INCREMENTED LOAD operations it should select ALU's output as source. For REVERSE LOAD, it should select register Q's output as source.
- ACps has to change over time when the REVERSE LOAD instruction is executed. At first AC has to parallel load the data from ALU's output, but then it has to shift it one by one towards reg Q.

Q4) It takes 10 cycles:

Through all cycles const_sel=0 and load_sel=0(Q's output as source).

1st cycle: ACps=1 => AC Loads the data parallel

2-9 cycles: ACps=0 => AC shifts the data to right. Q shifts the to left. AC_Out[0] is connected as input_right to Q. This way the data is reversed in 8 cycles.

10th cycle ACps=X => RegB Loads the correct reversed data.