**EE446 - Experiment 1**

**Preliminary Work**

1.2.1) Constant Value Generator

1. module constantValueGenerator
2. # (parameter W = 3, C = 9) (bus\_out);
4. output wire [W-1:0] bus\_out;
6. assign bus\_out = C;
7. endmodule

1.2.2) Decoder

.v file:

1. module Decoder2to4 (input0, input1, o0, o1, o2, o3);
3. input wire input0, input1;
4. output wire o0, o1, o2, o3;
6. assign o0 = ~input0 & ~input1;
7. assign o1 = input0 & ~input1;
8. assign o2 = ~input0 & input1;
9. assign o3 = input0 & input1;
11. endmodule

testbench:

1. module Decoder2to4\_tb ();
2. wire ou0, ou1, ou2, ou3;
3. reg in0, in1;
4. Decoder2to4 DUT(in0, in1, ou0, ou1, ou2, ou3);
5. reg [5:0] mem [3:0];
6. integer i;
8. initial begin
9. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/Decoder\_tb\_vector.txt" , mem);
10. #1000;
11. end
12. always @(\*) begin
13. **for**(i=0;i<4;i=i+1) begin
14. in0 = mem[i][4];
15. in1 = mem[i][5];
16. #10;
17. **if**({ou3,ou2,ou1,ou0} == mem[i][3:0])
18. $display ("No Error in %1d th row", i+1);
19. **else**
20. $display ("Error in %1d th row", i+1);
21. end
22. end
23. endmodule

testvector:

1. 00\_0001
2. 01\_0010
3. 10\_0100
4. 11\_1000

1.2.3)Multiplexers

2x1 .v file:

1. module Mux2to1
2. #(parameter W = 3) (i0, i1, select, out);
4. input [W-1:0] i0, i1;
5. input select;
6. output [W-1:0] out;
8. assign out = (select) ? i1 : i0;
10. endmodule

2x1 testbench:

1. module Mux2to1\_tb
2. #(parameter W = 2) ();
4. reg [W-1:0] i0, i1;
5. reg select;
6. wire [W-1:0] out;
8. reg [6:0] mem [1:0];
10. Mux2to1 DUT(i0,i1,select,out);
11. integer i;
13. initial begin
14. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/Mux2to1\_tb\_vector.txt" , mem);
15. #1000;
16. end
17. always @(\*) begin
18. **for**(i=0;i<2;i=i+1) begin
19. i0 = mem[i][4:3];
20. i1 = mem[i][6:5];
21. select = mem[i][2];
22. #10;
23. **if**(out == mem[i][1:0])
24. $display ("No Error in %1d th row", i+1);
25. **else**
26. $display ("Error in %1d th row", i+1);
27. end
28. end
29. 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);
4. input wire [W-1:0] in0,in1,in2,in3;
5. input wire [1:0] select;
6. output reg [W-1:0] out;
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
17. endmodule

4x1 testbench:

1. module Mux4to1\_tb
2. #(parameter W = 2) ();
4. reg [W-1:0] i0, i1, i2, i3;
5. reg [1:0] select;
6. wire [W-1:0] out;
8. reg [11:0] mem [3:0];
10. Mux4to1 DUT(i0,i1,i2,i3,select,out);
11. integer i;
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);
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;
13. reg [W-1:0] temp\_add;

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 overflow.
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;
47. **else** Z=1'b0;
48. end // End of SubtractionBA
49. 3'b011: begin //BitClear
50. Result = SrcA & (~SrcB);
51. **if**(Result == 0) Z=1'b1;
52. **else** Z=1'b0;
53. **if**(Result[W-1] == 1'b1) N=1'b1;
54. **else** N=1'b0; //Negative flag
55. CO=1'b0;
56. OVF=1'b0;
57. end//End of BitClear
58. 3'b100: begin //AND
59. Result = SrcA & SrcB;
60. **if**(Result == 0) Z=1'b1;
61. **else** Z=1'b0;
62. **if**(Result[W-1] == 1'b1) N=1'b1;
63. **else** N=1'b0; //Negative flag
64. CO=1'b0;
65. OVF=1'b0;
66. end // End of AND
67. 3'b101: begin //OR
68. Result = SrcA | SrcB;
69. **if**(Result == 0) Z=1'b1;
70. **else** Z=1'b0;
71. **if**(Result[W-1] == 1'b1) N=1'b1;
72. **else** N=1'b0; //Negative flag
73. CO=1'b0;
74. OVF=1'b0;
75. end // End of OR
76. 3'b110: begin //EXOR
77. Result = SrcA ^ SrcB;
78. **if**(Result == 0) Z=1'b1;
79. **else** Z=1'b0;
80. **if**(Result[W-1] == 1'b1) N=1'b1;
81. **else** N=1'b0; //Negative flag
82. CO=1'b0;
83. OVF=1'b0;
84. end //End of EXOR
85. 3'b111: begin //EXNOR
86. Result = ~(SrcA ^ SrcB);
87. **if**(Result == 0) Z=1'b1;
88. **else** Z=1'b0;
89. **if**(Result[W-1] == 1'b1) N=1'b1;
90. **else** N=1'b0; //Negative flag
91. CO=1'b0;
92. OVF=1'b0;
93. end //End of EXNOR
94. endcase
95. end
97. endmodule

testbench:

1. module ALU\_tb
2. #(parameter W=5) ();
4. //W-bit Inputs
5. reg [W-1:0] SrcA, SrcB;
6. //3 bit Control Signal
7. reg [2:0] Control;
8. //N-bit Result
9. wire [W-1:0] Result;
10. //Status Flags
11. wire CO, OVF, N, Z;
13. reg [21:0] mem [12:0];
15. ALU DUT (SrcA, SrcB, Control, Result, CO, OVF, N, Z);
17. integer i;
19. initial begin
20. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/ALU\_tb\_vector.txt" , mem);
21. #1000;
22. end
23. always @\*
24. begin
25. **for**(i=0;i<13;i=i+1) begin
26. SrcA = mem[i][21:17];
27. SrcB = mem[i][16:12];
28. Control = mem[i][11:9];
29. #10;
30. **if**({Result,CO,OVF,N,Z} == mem[i][8:0])
31. $display ("No Error in %1d th row", i+1);
32. **else**
33. $display ("Error in %1d th row", i+1);
34. end
35. end
36. endmodule

ALU testvector:

1. //input1\_input2\_Control\_Result\_CO,OVF,N,Z
2. 00111\_00101\_000\_01100\_0000
3. 00111\_00101\_001\_00010\_0000
4. 00111\_00101\_010\_11110\_1010
5. 00111\_00101\_011\_00010\_0000
6. 00111\_00101\_100\_00101\_0000
7. 00111\_00101\_101\_00111\_0000
8. 00111\_00101\_110\_00010\_0000
9. 00111\_00101\_111\_11101\_0010
10. //Zero detection test
11. 00111\_00111\_001\_00000\_0001
12. //Overflow Detection Test in Addition
13. 01101\_01001\_000\_10110\_0100
14. 10011\_10101\_000\_01000\_1100
15. 01110\_11000\_000\_00110\_1000
16. //Overflow Detection Test in Subtraction
17. 01101\_10011\_001\_11010\_1100

1.2.5) Registers:

Simple Register .v file:

1. module Reg\_Simple
2. #(parameter W=5) (clk, reset, data, out);
4. input wire clk,reset;
5. input wire [W-1:0] data;
6. output reg [W-1:0] out;
8. always @(posedge clk) begin
9. **if**(reset==1'b1) out <= 0;
10. **else** out <= data;
11. end
13. endmodule

Simple Register testbench:

1. module Reg\_Simple\_tb
2. #(parameter W=5)();
4. reg clk,reset;
5. reg [W-1:0] data;
6. wire [W-1:0] out;
8. reg [10:0] mem [3:0];
10. Reg\_Simple DUT (clk, reset, data, out);
12. integer i,j;
14. initial begin
15. clk=0;
16. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/Reg\_Simple\_tb\_vector.txt" , mem);
17. #1000;
18. end
19. always @\* begin
20. **for**(j=0;j<100;j=j+1) begin
21. clk = clk ^ 1'b1;
22. #5;
23. //For now the frequency is too high but it's just for simulation
24. end
25. end
26. always @\*
27. begin
28. **for**(i=0;i<4;i=i+1) begin
29. reset = mem[i][10];
30. data = mem[i][9:5];
31. #13;
32. **if**(out == mem[i][4:0])
33. $display ("No Error in %1d th row", i+1);
34. **else**
35. $display ("Error in %1d th row", i+1);
36. end
37. end
38. endmodule

Simple Register testvector:

1. //reset\_data\_out
2. 0\_11010\_11010
3. 0\_10011\_10011
4. 0\_00001\_00001
5. 1\_11010\_00000

Register with WE .v file:

1. module Reg\_WE
2. #(parameter W=5) (clk, reset, we, data, out);
4. input wire clk,reset,we;
5. input wire [W-1:0] data;
6. output reg [W-1:0] out;
8. always @(posedge clk) begin
9. **if**(reset==1'b1) out <= 0;
10. **else** begin
11. **if**(we == 1'b1) out <= data;
12. end
13. end
15. endmodule

Reg\_WE testbench:

1. module Reg\_WE\_tb
2. #(parameter W=5)();
4. reg clk,reset,we;
5. reg [W-1:0] data;
6. wire [W-1:0] out;
8. reg [11:0] mem [5:0];
10. Reg\_WE DUT (clk, reset, we, data, out);
12. integer i,j;
14. initial begin
15. clk=0;
16. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/Reg\_WE\_tb\_vector.txt" , mem);
17. #1000;
18. end
19. always @\* begin
20. **for**(j=0;j<100;j=j+1) begin
21. clk = clk ^ 1'b1;
22. #5;
23. //For now the frequency is too high but it's just for simulation
24. end
25. end
26. always @\*
27. begin
28. **for**(i=0;i<6;i=i+1) begin
29. reset = mem[i][11];
30. we = mem[i][10];
31. data = mem[i][9:5];
32. #10;
33. **if**(out == mem[i][4:0])
34. $display ("No Error in %1d th row", i+1);
35. **else**
36. $display ("Error in %1d th row", i+1);
37. end
38. end
39. endmodule

Reg\_WE testvector:

1. //reset\_we\_data\_out
2. 0\_1\_11010\_11010
3. 0\_0\_10011\_11010
4. 0\_1\_00001\_00001
5. 1\_0\_11010\_00000
6. 0\_1\_11011\_11011
7. 1\_1\_10011\_00000

Shift Register .v file:

1. module Reg\_Shift
2. #(parameter W = 5) (clk, reset, ps\_select, lr\_select, data, input\_left, input\_right, out);
4. input wire clk,reset,ps\_select, lr\_select;
5. input wire [W-1:0] data;
6. input wire input\_left, input\_right;
8. output reg [W-1:0] out;
10. always @(posedge clk) begin
11. **if**(reset==1) out<=0; //Synchronous reset
12. **else** begin
13. **if**(ps\_select==1) begin
14. out <= data; //If Parallel/Serial select=1 then load parallel data
15. end
16. **else** begin //Otherwise check for left/right shif signal
17. **if**(lr\_select == 1) begin //if lr=1 shift right
18. out <= {input\_left, out[W-1:1]};
19. end
20. **else** begin //lr=0 so shift left
21. out <= {out[W-2:0],input\_right};
22. end
23. end
24. end
25. end
26. endmodule

Shift Register testbench:

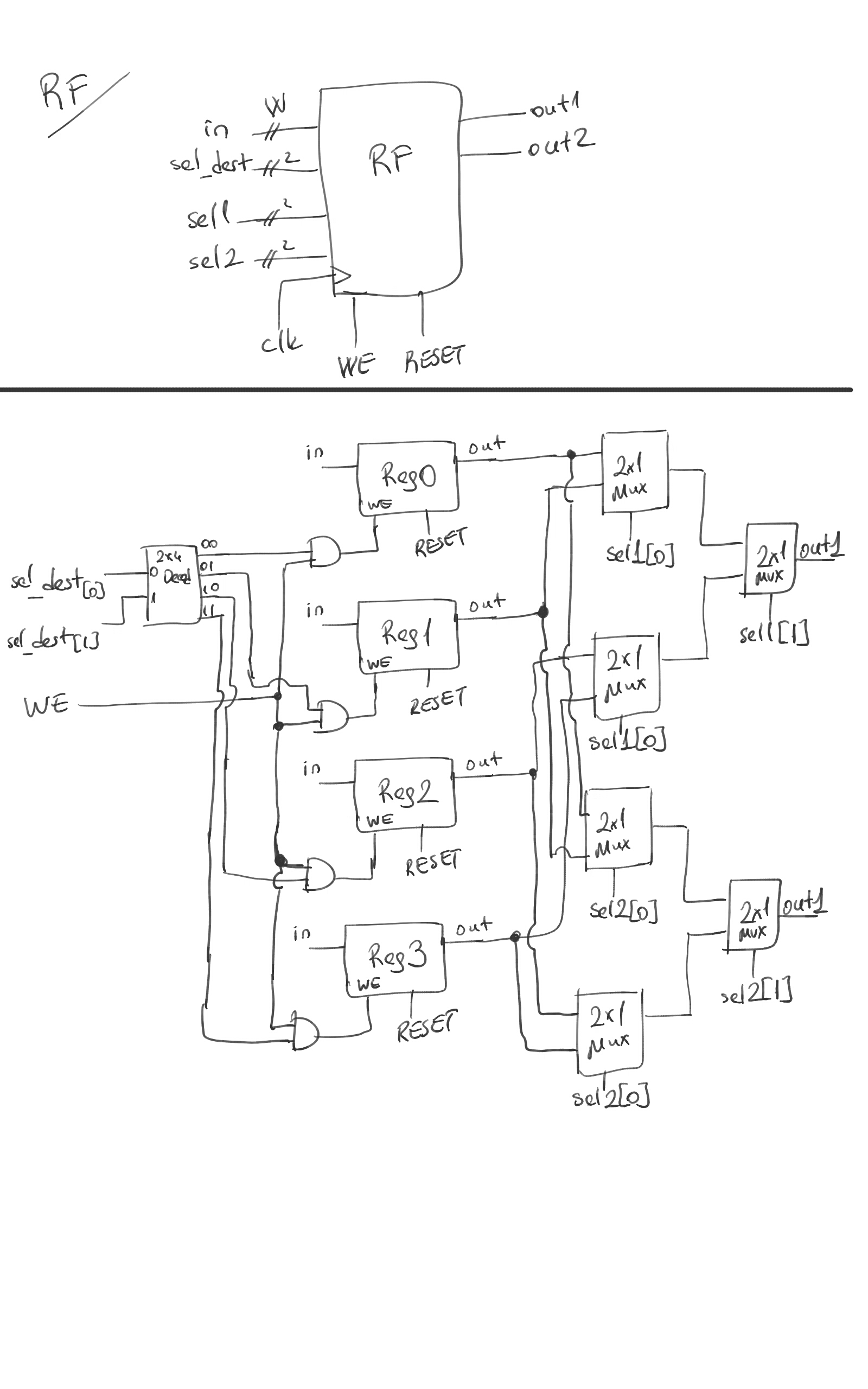
1. module Reg\_Shift\_tb
2. #(parameter W=5)();
4. reg clk,reset,ps\_select,lr\_select,input\_left,input\_right;
5. reg [W-1:0] data;
6. wire [W-1:0] out;
8. reg [14:0] mem [6:0];
10. Reg\_Shift DUT (clk, reset, ps\_select, lr\_select, data, input\_left, input\_right, out);
12. integer i,j;
14. initial begin
15. clk=0;
16. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/lab1\_registerfile\_tb\_vector.txt" , mem);
17. #1000;
18. end
19. always @\* begin
20. **for**(j=0;j<100;j=j+1) begin
21. clk = clk ^ 1'b1;
22. #5;
23. //For now the frequency is too high but it's just for simulation
24. end
25. end
26. always @\*
27. begin
28. **for**(i=0;i<7;i=i+1) begin
29. reset = mem[i][14];
30. ps\_select = mem[i][13];
31. lr\_select = mem[i][12];
32. input\_left = mem[i][11];
33. input\_right = mem[i][10];
34. data = mem[i][9:5];
35. #10;
36. **if**(out == mem[i][4:0])
37. $display ("No Error in %1d th row", i+1);
38. **else**
39. $display ("Error in %1d th row", i+1);
40. end
41. end
42. endmodule

Reg\_Shift testvector:

1. //reset\_psselect,lrselect\_inputleft,inputright\_data\_out
2. 0\_10\_11\_11010\_11010
3. 0\_01\_10\_00011\_11101
4. 0\_01\_00\_00101\_01110
5. 1\_11\_01\_11010\_00000
6. 0\_11\_10\_11011\_11011
7. 0\_00\_00\_10011\_10110
8. 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.
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"
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. );

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;
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;




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));

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;

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;
92. assign  SYNTHESIZED\_WIRE\_18 = we & SYNTHESIZED\_WIRE\_2;
94. assign  SYNTHESIZED\_WIRE\_19 = we & SYNTHESIZED\_WIRE\_3;
96. assign  SYNTHESIZED\_WIRE\_0 = we & SYNTHESIZED\_WIRE\_4;
98. assign  SYNTHESIZED\_WIRE\_1 = we & SYNTHESIZED\_WIRE\_5;

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;

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;

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;

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;

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;

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;

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;

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;

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.
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"
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. );

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;
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;




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));

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;

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;
258. assign  SYNTHESIZED\_WIRE\_18 = we & SYNTHESIZED\_WIRE\_2;
260. assign  SYNTHESIZED\_WIRE\_19 = we & SYNTHESIZED\_WIRE\_3;
262. assign  SYNTHESIZED\_WIRE\_0 = we & SYNTHESIZED\_WIRE\_4;
264. assign  SYNTHESIZED\_WIRE\_1 = we & SYNTHESIZED\_WIRE\_5;

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;

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;

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;

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;

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;

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;

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;

324. Reg\_WE  b2v\_inst9(
325. .clk(clk),
326. .reset(reset),
327. .we(SYNTHESIZED\_WIRE\_19),
328. .data(in),
329. .out(SYNTHESIZED\_WIRE\_21));
330. defparam    b2v\_inst9.W = W;

333. endmodule

endmodule

testbench:

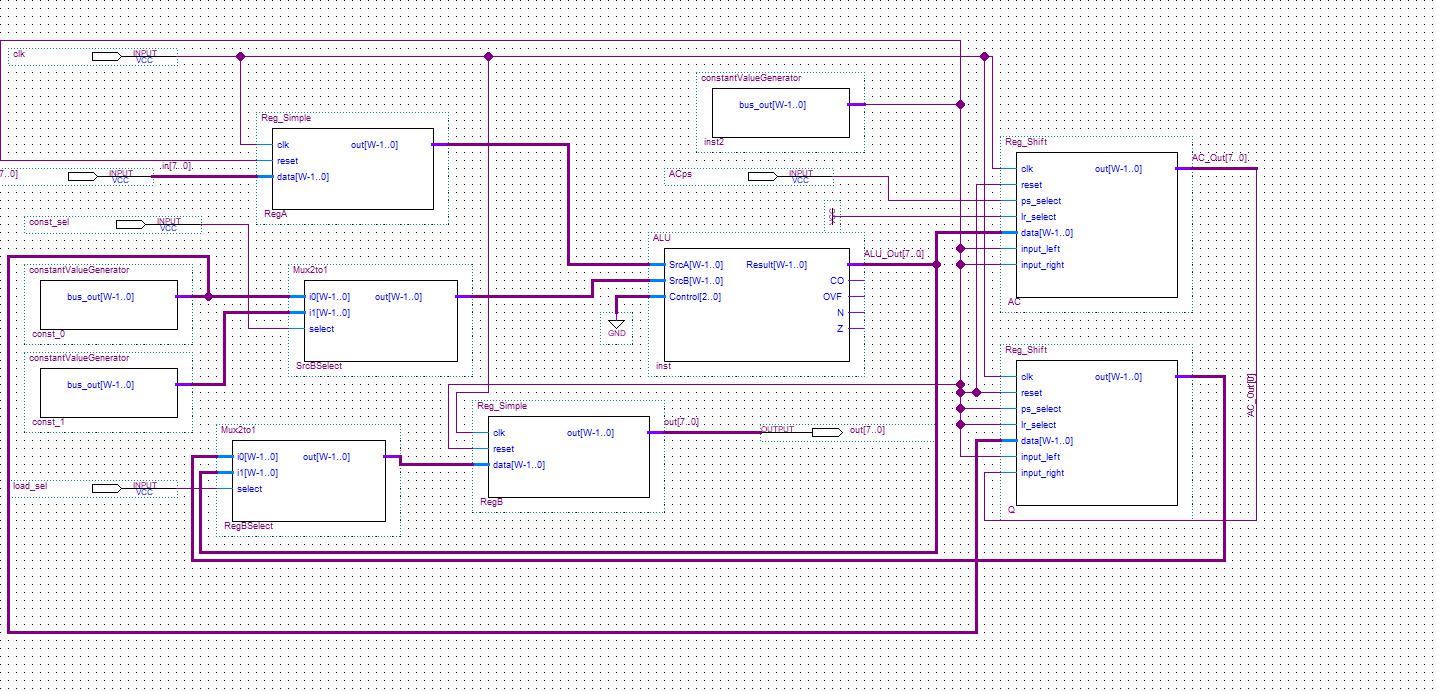
1. module lab1\_registerfile\_tb
2. #(parameter W = 3) ();
4. reg we,clk,reset;
5. reg [1:0] sel1,sel2,sel\_dest;
6. reg [W-1:0] in;
7. wire [W-1:0] out1,out2;
9. reg [16:0] mem [6:0];
11. lab1\_registerfile DUT (we, clk, reset,in,sel1,sel2,sel\_dest,out1,out2);
13. integer i,j;
15. initial begin
16. clk=0;
17. $readmemb("C:/Users/AhmetSalih/Desktop/EE446\_Desktop/test\_vectors/lab1\_registerfile\_tb\_vector.txt" , mem);
18. #1000;
19. end
20. always @\* begin
21. **for**(j=0;j<100;j=j+1) begin
22. clk = clk ^ 1'b1;
23. #5;
24. //For now the frequency is too high but it's just for simulation
25. end
26. end
27. always @\*
28. begin
29. **for**(i=0;i<7;i=i+1) begin
30. we = mem[i][16];
31. reset = mem[i][15];
32. in = mem[i][14:12];
33. sel1 = mem[i][11:10];
34. sel2 = mem[i][9:8];
35. sel\_dest = mem[i][7:6];
36. #10;
37. **if**({out1,out2} == mem[i][5:0])
38. $display ("No Error in %1d th row", i+1);
39. **else**
40. $display ("Error in %1d th row", i+1);
41. end
42. end
43. endmodule

testvector:

1. //we\_reset\_in\_sel1\_sel2\_sel\_dest\_out1\_out2);
2. 1\_1\_101\_10\_11\_10\_000\_000
3. 1\_0\_101\_10\_00\_00\_000\_101
4. 1\_0\_111\_11\_01\_01\_000\_111
5. 1\_0\_001\_11\_10\_10\_000\_001
6. 1\_0\_011\_11\_11\_11\_011\_011
7. 1\_0\_010\_01\_10\_11\_111\_001
8. 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.