KUZEY KIBRIS KAMPÜSÜ
ODTÜ METU
NORTHERN CYPRUS CAMPUS

Lab 6 Report

Name: Muhammad Somaan
ID: 2528404

2 to 4 Line Decoder)

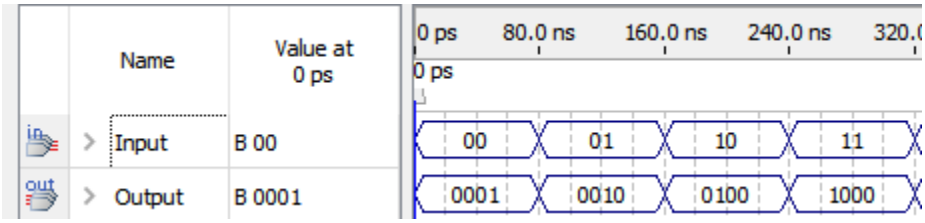
Verilog code:

```
1   module Decoder_2_To_4(Input, Output);
2
3   parameter size = 4;
4
5   input [size-3: 0] Input;
6   output reg [size-1:0] Output;
7
8
9
10  always @(Input) begin
11  |
12  |   if(Input == 2'b00) begin
13  |      Output = 4'b0001;
14  |      end
15  |   else if(Input == 2'b01) begin
16  |      Output = 4'b0010;
17  |      end
18  |   else if(Input == 2'b10) begin
19  |      Output = 4'b0100;
20  |      end
21  |   else if(Input == 2'b11) begin
22  |      Output = 4'b1000;
23  |      end
24  |  end
25  |
26  endmodule
27
```

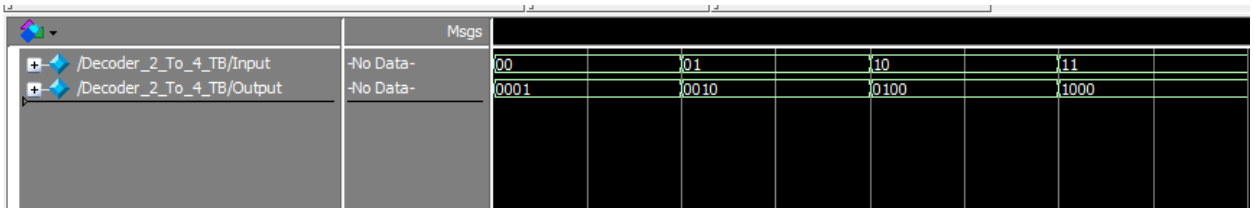
ModelSim Testbench:

```
1   module Decoder_2_To_4_TB();
2
3     parameter size = 4;
4
5     reg [size-3:0] Input;
6     wire [size-1:0] Output;
7
8     Decoder_2_To_4 DUT(Input, Output);
9
10    initial begin
11      Input = 2'b00; #100;
12      Input = 2'b01; #100;
13      Input = 2'b10; #100;
14      Input = 2'b11; #100;
15
16    end
17
18  endmodule
19
```

Functional Simulation:



ModelSim Simulation:



Comments: By using the truth table provided we designed a simple 2 to 4 Line Decoder. The functional simulation and modelsim simulation confirms the functionality of the decoder.

4 bit 2 to 1 Mux)

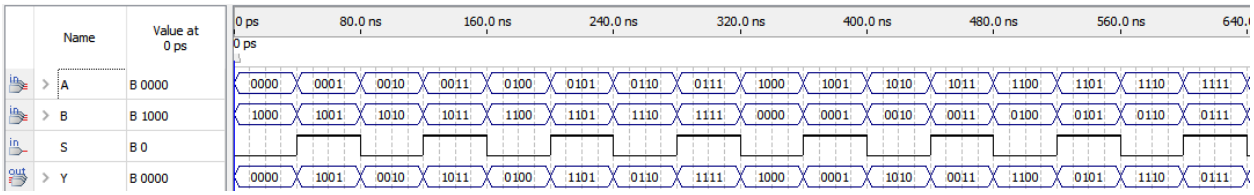
Verilog code:

```
1   module Four_Bit_2_To_1_Mux(A, B, S, Y);
2
3   parameter size = 4;
4
5   input [size-1:0] A, B;
6   input S;
7
8   output reg [size-1:0] Y;
9
10  integer i;
11
12  always @(A or B or S)
13
14     for(i=0; i<size; i=i+1) begin: Four_Bit_2_To_1_Mux
15        Y[i] = A[i] & ~S | B[i] & S;
16     end
17
18  endmodule
19
20
```

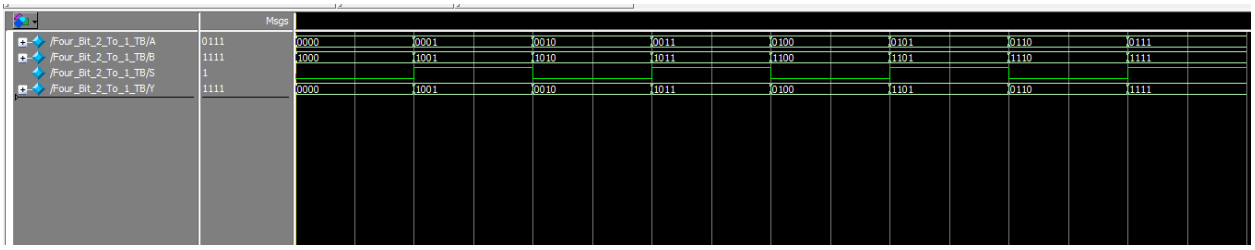
ModelSim Testbench:

```
1   module Four_Bit_2_To_1_TB();
2
3     parameter size = 4;
4
5     reg [size-1:0] A, B;
6     reg S;
7     wire [size-1:0] Y;
8
9     Four_Bit_2_To_1_Mux DUT(A, B, S, Y);
10
11    initial begin
12      A = 4'b0000; B = 4'b1000; S = 1'b0; #100;
13      A = 4'b0001; B = 4'b1001; S = 1'b1; #100;
14      A = 4'b0010; B = 4'b1010; S = 1'b0; #100;
15      A = 4'b0011; B = 4'b1011; S = 1'b1; #100;
16      A = 4'b0100; B = 4'b1100; S = 1'b0; #100;
17      A = 4'b0101; B = 4'b1101; S = 1'b1; #100;
18      A = 4'b0110; B = 4'b1110; S = 1'b0; #100;
19      A = 4'b0111; B = 4'b1111; S = 1'b1; #100;
20    end
21
22  endmodule
```

Functional Simulation:



ModelSim Simulation:



Comments: We designed a simple 4 bit 2 to 1 Mux whose functionality is confirmed by the functional and modelsim simulation. Whenever S = 0, Input A is chosen as output and whenever S = 1, Input B is chosen as the output.

4 bit 4 to 1 Mux)

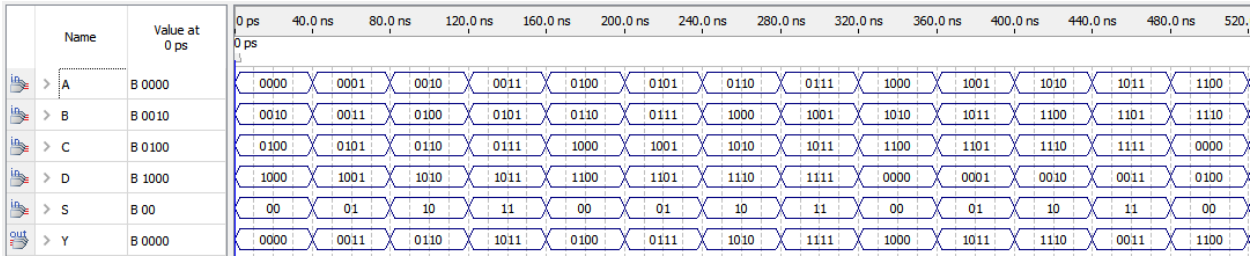
Verilog code:

```
1   module Four_Bit_4_To_1_Mux(A, B, C, D, S, Y);
2
3   parameter size = 4;
4
5   input [size-1:0] A, B, C, D;
6   input [size-3:0] S;
7
8   output reg [size-1:0] Y;
9
10
11  always @(A or B or C or D or S)
12
13     if(S == 2'b00) begin
14        Y = A;
15        end
16     else if(S == 2'b01) begin
17        Y = B;
18        end
19     else if(S == 2'b10) begin
20        Y = C;
21        end
22     else begin
23        Y = D;
24        end
25
26  endmodule
```

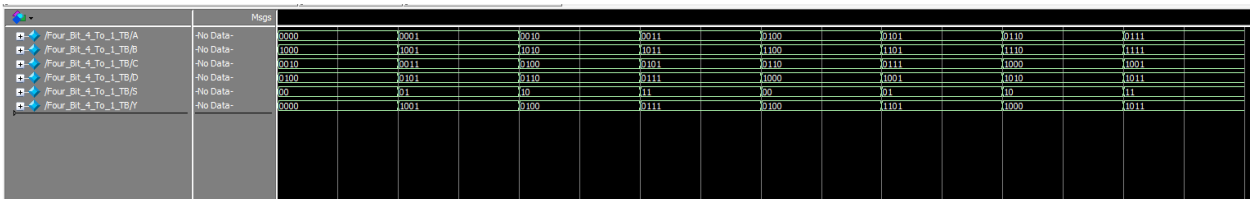
ModelSim Testbench:

```
1   module Four_Bit_4_To_1_TB();
2
3     parameter size = 4;
4
5     reg [size-1:0] A, B, C, D;
6     reg [size-3:0] S;
7     wire [size-1:0] Y;
8
9     Four_Bit_4_To_1_Mux DUT(A, B, C, D, S, Y);
10
11    initial begin
12      A = 4'b0000; B = 4'b1000; C = 4'b0010; D = 4'b0100; S = 2'b00; #100;
13      A = 4'b0001; B = 4'b1001; C = 4'b0011; D = 4'b0101; S = 2'b01; #100;
14      A = 4'b0010; B = 4'b1010; C = 4'b0100; D = 4'b0110; S = 2'b10; #100;
15      A = 4'b0011; B = 4'b1011; C = 4'b0101; D = 4'b0111; S = 2'b11; #100;
16      A = 4'b0100; B = 4'b1100; C = 4'b0110; D = 4'b1000; S = 2'b00; #100;
17      A = 4'b0101; B = 4'b1101; C = 4'b0111; D = 4'b1001; S = 2'b01; #100;
18      A = 4'b0110; B = 4'b1110; C = 4'b1000; D = 4'b1010; S = 2'b10; #100;
19      A = 4'b0111; B = 4'b1111; C = 4'b1001; D = 4'b1011; S = 2'b11; #100;
20    end
21
22  endmodule
```

Functional Simulation:



ModelSim Simulation:



Comments: We designed a simple 4 bit 4 to 1 Mux whose functionality is confirmed by the functional and modelsim simulation. Whenever S = 00, Input A is chosen as output, whenever S = 01, Input B is chosen as the output, whenever S = 10, Input C is chosen as the output, whenever S = 11, Input D is chosen as the output.

D Flip Flop)

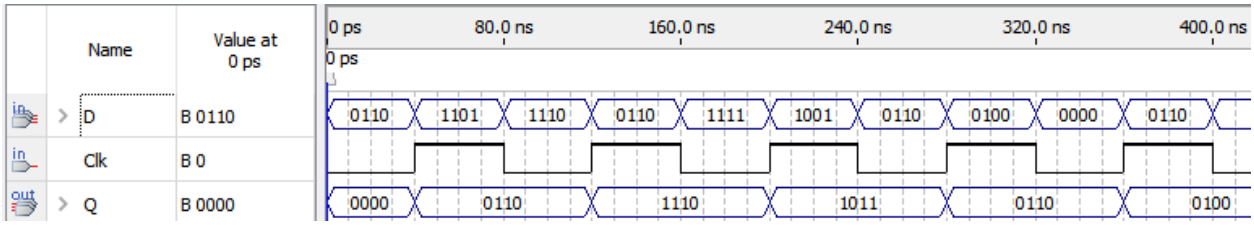
Verilog code:

```
1   module D_Flip_Flop(D, Clk, Q);
2
3   parameter size = 4;
4
5   input [size-1:0] D;
6   input Clk;
7   output reg [size-1:0] Q;
8
9
10  always @(posedge Clk)
11     Q <= D;
12
13  endmodule
```

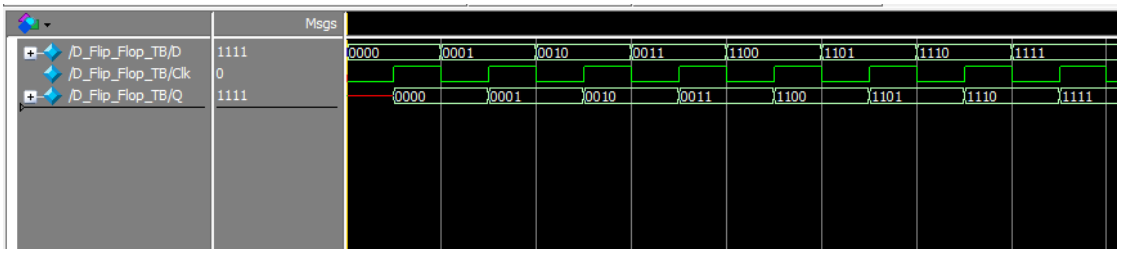
ModelSim Testbench:

```
1   module D_Flip_Flop_TB();
2
3     parameter size = 4;
4
5     reg [size-1:0] D;
6     reg Clk;
7     wire [size-1:0] Q;
8
9     D_Flip_Flop DUT(D, Clk, Q);
10
11    always begin
12        Clk = 0; #50;
13        Clk = 1; #50;
14    end
15    initial begin
16      D = 4'b0000; #100;
17      D = 4'b0001; #100;
18      D = 4'b0010; #100;
19      D = 4'b0011; #100;
20      D = 4'b1100; #100;
21      D = 4'b1101; #100;
22      D = 4'b1110; #100;
23      D = 4'b1111; #100;
24    end
25
26  endmodule
```

Functional Simulation:



ModelSim Simulation:



Comments: We designed a simple D flip flop. On the rising edge of the clock the input is assigned to the ouput and then the output value is held until the next rising edge of clock. The functionality is confirmed by the simulation.

ALU)

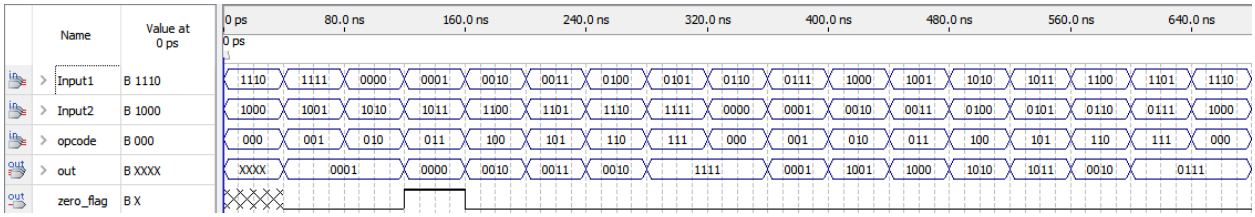
Verilog code:

```
1   module ALU(Input1, Input2, opcode, zero_flag, out);
2
3   parameter size = 4;
4
5   input [size-1:0] Input1, Input2;
6   input [size-2:0] opcode;
7
8   output reg zero_flag;
9   output reg [size-1:0] out;
10
11  always @(Input1, Input2, opcode) begin
12
13     if(opcode == 3'b001) begin
14        out = 1;
15        end
16     else if(opcode == 3'b010) begin
17        out = Input1 + 1;
18        end
19     else if(opcode == 3'b011) begin
20        out = Input1 - 1;
21        end
22     else if(opcode == 3'b100) begin
23        out = Input1;
24        end
25     else if(opcode == 3'b101) begin
26        out = Input1;
27        end
28     else if(opcode == 3'b110) begin
29        out = Input1 + Input2;
30        end
31     else if(opcode == 3'b111) begin
32        out = Input2;
33        end
34
35     zero_flag = (~out[0] & ~out[1] & ~out[2] & ~out[3]);
36  end
37  endmodule
```

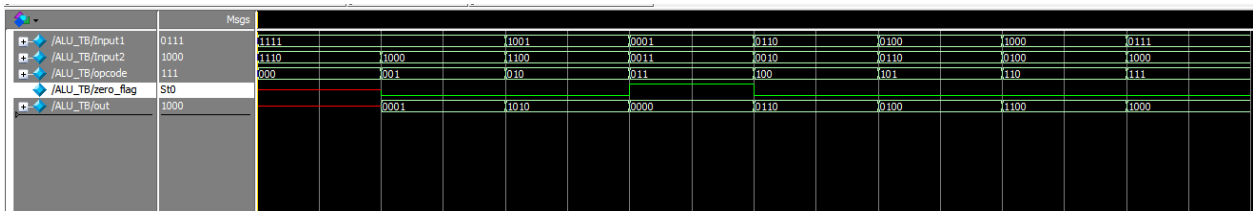
ModelSim Testbench:

```
1   module ALU_TB();
2
3     parameter size = 4;
4
5     reg [size-1:0] Input1, Input2;
6     reg [size-2:0] opcode;
7     wire zero_flag;
8     wire [size-1:0] out;
9
10    ALU DUT(Input1, Input2, opcode, zero_flag, out);
11
12    initial begin
13      Input1 = 4'b1111; Input2 = 4'b1110; opcode = 3'b000; #100;
14      Input1 = 4'b1111; Input2 = 4'b1000; opcode = 3'b001; #100;
15      Input1 = 4'b1001; Input2 = 4'b1100; opcode = 3'b010; #100;
16      Input1 = 4'b0001; Input2 = 4'b0011; opcode = 3'b011; #100;
17      Input1 = 4'b0110; Input2 = 4'b0010; opcode = 3'b100; #100;
18      Input1 = 4'b0100; Input2 = 4'b0110; opcode = 3'b101; #100;
19      Input1 = 4'b1000; Input2 = 4'b0100; opcode = 3'b110; #100;
20      Input1 = 4'b0111; Input2 = 4'b1000; opcode = 3'b111; #100;
21
22    end
23
24  endmodule
```

Functional Simulation:



ModelSim Simulation:



Comments: We designed the ALU as stated in the lab manual. All operations work according to the table provided in the manual and the simulations confirms the functionality. When opcode = 000, output is not defined so we get the red line in modelsim simulation.

FIBO Datapath)

Verilog code:

```
module FIBO_DATAPATH(wrt_addr, wrt_en, clk, load_data, rd_addr1, rd_addr2, alu_opcode, count, zero_flag, data);

parameter size = 4;

input [size-3:0] wrt_addr, rd_addr1, rd_addr2;
input [size-2:0] alu_opcode;
input [size-1:0] count;
input wrt_en, clk, load_data;

output [size-1:0] data;
output zero_flag;

wire [size-1:0] decoder_out, And_out, Mux_Enable_out, Mux0, Mux1, Mux2, Mux3, Mux4, FF1, FF2, FF3, FF4, Mux5, Mux6, Alu_out;

Decoder_2_To_4 D1(wrt_addr, decoder_out);

and A1(And_out[0], decoder_out[0], wrt_en);
and A2(And_out[1], decoder_out[1], wrt_en);
and A3(And_out[2], decoder_out[2], wrt_en);
and A4(And_out[3], decoder_out[3], wrt_en);

Four_Bit_2_To_1_Mux M0(data, count, load_data, Mux0);

//muxes
Four_Bit_2_To_1_Mux M1(FF1, Mux0, And_out[0], Mux1);
Four_Bit_2_To_1_Mux M2(FF2, Mux0, And_out[1], Mux2);
Four_Bit_2_To_1_Mux M3(FF3, Mux0, And_out[2], Mux3);
Four_Bit_2_To_1_Mux M4(FF4, Mux0, And_out[3], Mux4);

D_Flip_Flop F1(Mux1, clk, FF1);
D_Flip_Flop F2(Mux2, clk, FF2);
D_Flip_Flop F3(Mux3, clk, FF3);
D_Flip_Flop F4(Mux4, clk, FF4);

Four_Bit_4_To_1_Mux M01(FF1, FF2, FF3, FF4, rd_addr1, Mux5);
Four_Bit_4_To_1_Mux M02(FF1, FF2, FF3, FF4, rd_addr2, Mux6);

ALU AL1(Mux5, Mux6, alu_opcode, zero_flag, Alu_out);

D_Flip_Flop F5(Alu_out, clk, data);

endmodule
```

ModelSim Testbench:

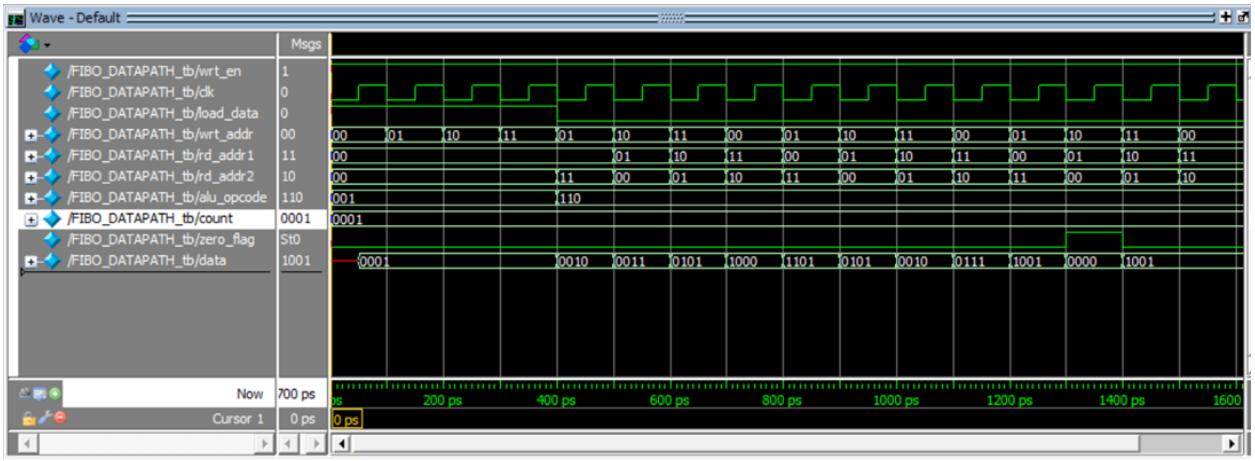
```
module FIBO_DATAPATH_tb();
parameter   size=4;
reg         wrt_en,clk,load_data;
reg         [1:0]wrt_addr,rd_addr1,rd_addr2;
reg         [2:0]alu_opcode;
reg         [size-1:0]count;
wire        zero_flag;
wire        [size-1:0]data;

FIBO_DATAPATH DUT(wrt_addr,wrt_en,clk,load_data,rd_addr1,rd_addr2,alu_opcode,count,zero_flag,data);
  always begin
  clk=0; #50;
  clk=1; #50;
  end

initial
 begin
  wrt_addr=2'b00; wrt_en=1; load_data=1; rd_addr1=2'b00; rd_addr2=2'b00; alu_opcode=3'b001; count=4'b0001; #100;
  wrt_addr=2'b01; wrt_en=1; load_data=1; rd_addr1=2'b00; rd_addr2=2'b00; alu_opcode=3'b001; count=4'b0001; #100;
  wrt_addr=2'b10; wrt_en=1; load_data=1; rd_addr1=2'b00; rd_addr2=2'b00; alu_opcode=3'b001; count=4'b0001; #100;
  wrt_addr=2'b11; wrt_en=1; load_data=1; rd_addr1=2'b00; rd_addr2=2'b00; alu_opcode=3'b001; count=4'b0001; #100;
  wrt_addr=2'b01; wrt_en=1; load_data=0; rd_addr1=2'b00; rd_addr2=2'b11; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b10; wrt_en=1; load_data=0; rd_addr1=2'b01; rd_addr2=2'b00; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b11; wrt_en=1; load_data=0; rd_addr1=2'b10; rd_addr2=2'b01; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b00; wrt_en=1; load_data=0; rd_addr1=2'b11; rd_addr2=2'b10; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b01; wrt_en=1; load_data=0; rd_addr1=2'b00; rd_addr2=2'b11; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b10; wrt_en=1; load_data=0; rd_addr1=2'b01; rd_addr2=2'b00; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b11; wrt_en=1; load_data=0; rd_addr1=2'b10; rd_addr2=2'b01; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b00; wrt_en=1; load_data=0; rd_addr1=2'b11; rd_addr2=2'b10; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b01; wrt_en=1; load_data=0; rd_addr1=2'b00; rd_addr2=2'b11; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b10; wrt_en=1; load_data=0; rd_addr1=2'b01; rd_addr2=2'b00; alu_opcode=3'b110; count=4'b0001; #100;
  wrt_addr=2'b11; wrt_en=1; load_data=0; rd_addr1=2'b10; rd_addr2=2'b01; alu_opcode=3'b110; count=4'b0001; #100;

  wrt_addr=2'b00; wrt_en=1; load_data=0; rd_addr1=2'b11; rd_addr2=2'b10; alu_opcode=3'b110; count=4'b0001; #100;
end
endmodule
```

ModelSim Simulation:



Comments: We designed the datapath according to the schematic provided in the lab manual and then confirmed the fibonacci sequence using modelsim simulation. The simulation is as expected and the datapath works accordingly. Data is undefined for the first 50 ns as the clock hasn't yet reached a rising edge. After 90 ns we can see that data should have been 21 in denary, but since 21 cannot be stored in 4 bits the first 4 bits are correct but it doesn't show the 5th bit which becomes 1. So where it shows 0101 it is actually supposed to be 10101, which is 21.