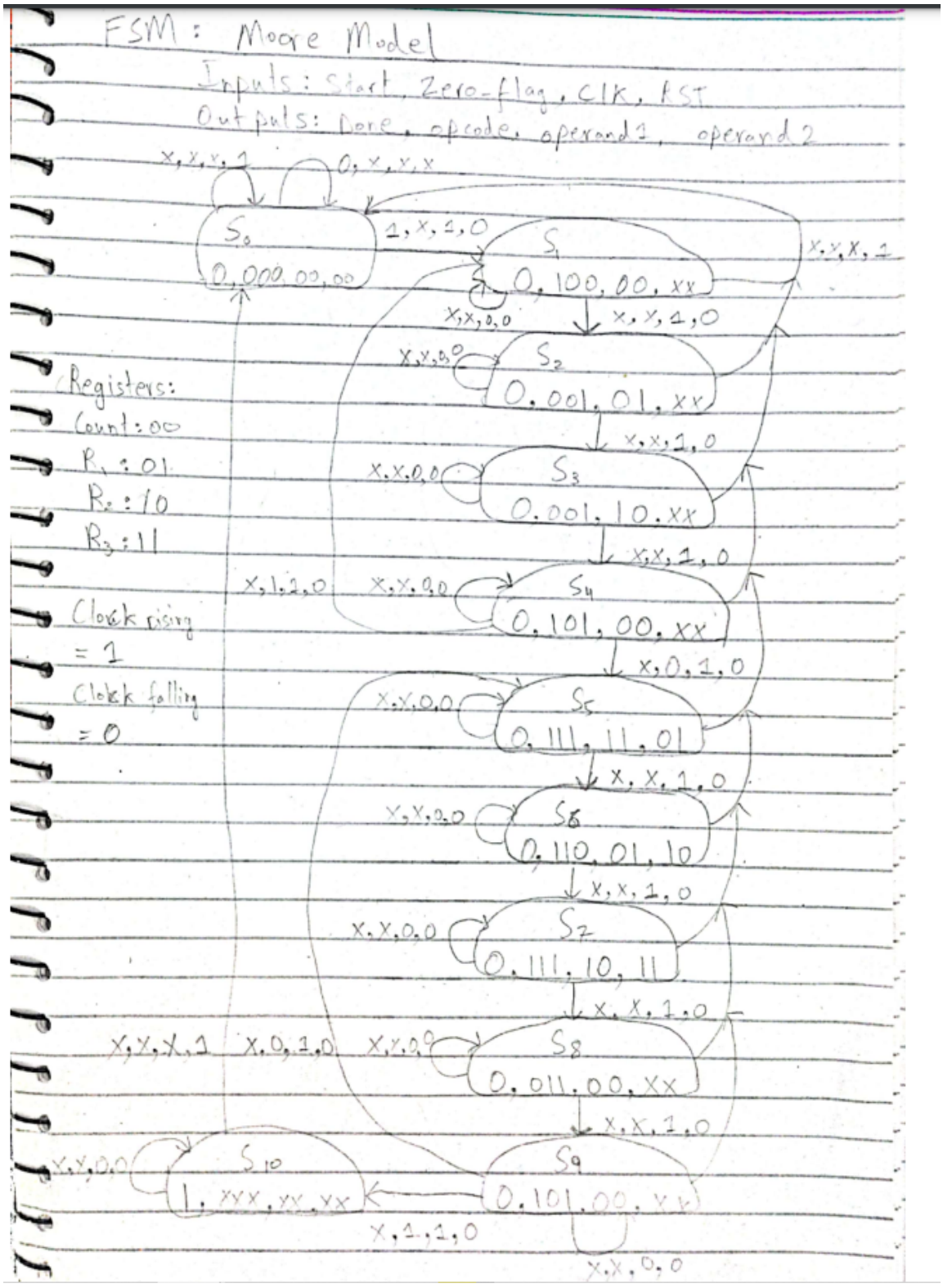
KUZEY KIBRIS KAMPÜSÜ
ODTÜ METU
NORTHERN CYPRUS CAMPUS

Lab 7 Report

Name: Muhammad Somaan
ID: 2528404

FSM)

State diagram:



State Table:

State	Inputs				Outputs			
	0,x,x,1	1,1,1,0	1,0,1,0	x,x,1,1	Done	opcode	op1	op2
S_0	S_0	S_1	S_1	S_0	0	000	00	00
S_1	S_0	S_2	S_2	S_0	0	100	00	xx
S_2	S_0	S_3	S_3	S_0	0	001	01	xx
S_3	S_0	S_4	S_4	S_0	0	001	10	xx
S_4	S_0	S_1	S_5	S_0	0	101	00	xx
S_5	S_0	S_6	S_6	S_0	0	111	11	01
S_6	S_0	S_7	S_7	S_0	0	110	01	10
S_7	S_0	S_8	S_8	S_0	0	111	10	11
S_8	S_0	S_9	S_9	S_0	0	011	00	xx
S_9	S_0	S_{10}	S_5	S_0	0	101	00	xx
S_{10}	S_0	S_{10}	S_{10}	S_0	1	xxx	xx	xx

Verilog code:

```
1   module FSM(START, ZERO_FLAG, CLK, RST, DONE, opcode, operand1, operand2);
2
3   parameter size = 4;
4
5   input START, ZERO_FLAG, CLK, RST;
6   output reg [size-2:0] opcode;
7   output reg [size-3:0] operand1, operand2;
8   output reg DONE;
9
10  reg [size-1:0] state, nextState;
11  parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6, S7=7, S8=8, S9=9, S10=10;
12
13  initial begin
14      state <= S0;
15      nextState <= S0;
16  end
17
18  always @(posedge CLK or posedge RST) begin
19
20      if(RST)
21          state <= S0;
22      else
23          state <= nextState;
24  end
25
26  always @(state) begin
27
28      DONE = 0;
29      opcode = 3'b000;
30      operand1 = 2'b00;
31      operand2 = 2'b00;
32
33      case(state)
34          S0: begin
35              opcode = 3'b000;
36          end
37          S1: begin
38              opcode = 3'b100;
39              operand1 = 2'b00;
40          end
41          S2: begin
42              opcode = 3'b001;
43              operand1 = 2'b01;
44          end
```

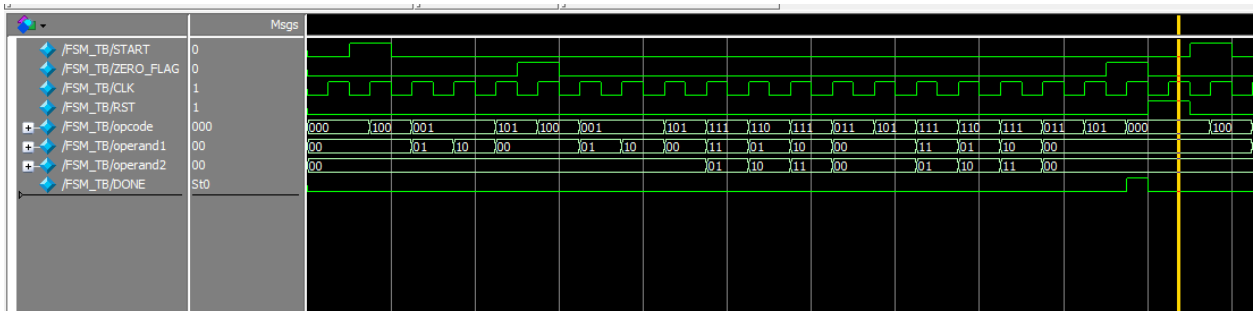
```
44          end
45          S3: begin
46              opcode = 3'b001;
47              operand1 = 2'b10;
48          end
49          S4: begin
50              opcode = 3'b101;
51              operand1 = 2'b00;
52          end
53          S5: begin
54              opcode = 3'b111;
55              operand1 = 2'b11;
56              operand2 = 2'b01;
57          end
58          S6: begin
59              opcode = 3'b110;
60              operand1 = 2'b01;
61              operand2 = 2'b10;
62          end
63          S7: begin
64              opcode = 3'b111;
65              operand1 = 2'b10;
66              operand2 = 2'b11;
67          end
68          S8: begin
69              opcode = 3'b011;
70              operand1 = 2'b00;
71          end
72          S9: begin
73              opcode = 3'b101;
74              operand1 = 2'b00;
75          end
76          S10: begin
77              DONE = 1;
78          end
79      endcase
80  end
81
82  always @(state or START or ZERO_FLAG) begin
83
84      nextState = S0;
85
```

```
86      case(state)
87          S0: begin
88              if(START)
89                  nextState = S1;
90          end
91          S1: begin
92              nextState = S2;
93          end
94          S2: begin
95              nextState = S3;
96          end
97          S3: begin
98              nextState = S4;
99          end
100         S4: begin
101             if(ZERO_FLAG)
102                 nextState = S1;
103             else
104                 nextState = S5;
105         end
106         S5: begin
107             nextState = S6;
108         end
109         S6: begin
110             nextState = S7;
111         end
112         S7: begin
113             nextState = S8;
114         end
115         S8: begin
116             nextState = S9;
117         end
118         S9: begin
119             if(ZERO_FLAG)
120                 nextState = S10;
121             else
122                 nextState = S5;
123         end
124         S10: begin
125             nextState = S10;
126         end
127     endcase
128 end
129 endmodule
```

ModelSim Testbench:

```
module FSM_TB();

  parameter size = 4;

  reg START, ZERO_FLAG, CLK, RST;
  wire [size-2:0] opcode;
  wire [size-3:0] operand1, operand2;
  wire DONE;

  FSM DUT(START, ZERO_FLAG, CLK, RST, DONE, opcode, operand1, operand2);

  always begin
      CLK = 0; #50;
      CLK = 1; #50;
  end

  initial begin
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 1; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 1; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 1; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 1; #100;

    START = 1; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
  end


endmodule;
```

ModelSim Simulation:



Comments: By using the flowchart provided in the lab manual, I first drew a state table followed by a state table. Then using the state diagram and table, I used the Moore Model provided in the manual and wrote all the verilog code according to it. For the modelsim testbench I followed the flowchart making sure to making sure to include both decisions yes and no for the decision box “count == 0”. The modesim simulation completely verifies my state diagram and agree with the flowchart as well.

FSM_DECO)

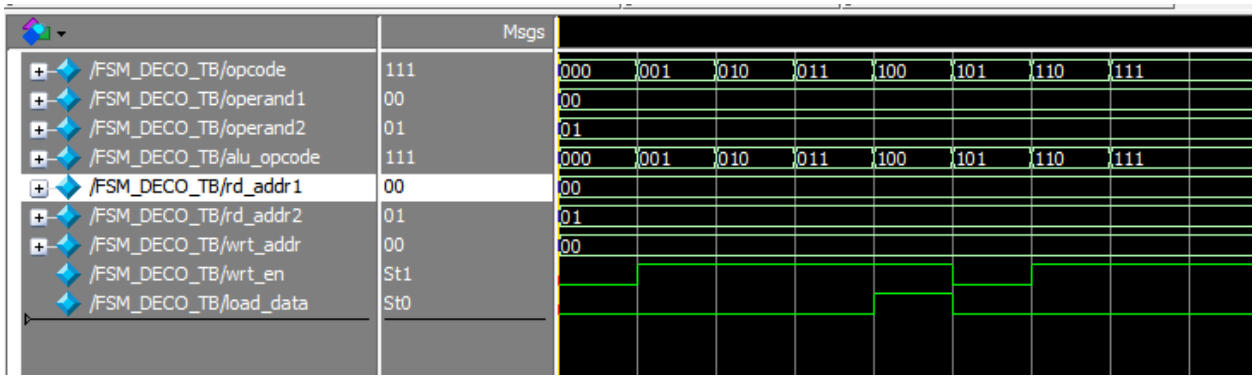
Verilog code:

```
module FSM_DECO(opcode, operand1, operand2, alu_opcode, rd_addr1, rd_addr2, wrt_addr, wrt_en, load_data);

parameter size = 4;

input [size-2:0] opcode;
input [size-3:0] operand1, operand2;

output reg [size-2:0] alu_opcode;
output reg [size-3:0] rd_addr1, rd_addr2, wrt_addr;
output reg wrt_en, load_data;

always @(opcode or operand1 or operand2) begin

   alu_opcode = opcode;
   wrt_addr = operand1;
   rd_addr1 = operand1;
   rd_addr2 = operand2;
   wrt_en = 1;
   load_data = 0;

   if(opcode == 3'b000 || opcode == 3'b101)
      wrt_en = 0;
   if(opcode == 3'b100)
      load_data = 1;

end
endmodule
```

ModelSim Testbench:

```
module FSM_DECO_TB();

  parameter size = 4;

  reg [size-2:0] opcode;
  reg [size-3:0] operand1, operand2;

  wire [size-2:0] alu_opcode;
  wire [size-3:0] rd_addr1, rd_addr2, wrt_addr;
  wire wrt_en, load_data;

  FSM_DECO DUT(opcode, operand1, operand2, alu_opcode, rd_addr1, rd_addr2, wrt_addr, wrt_en, load_data);

  initial begin
    opcode = 3'b000; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b001; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b010; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b011; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b100; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b101; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b110; operand1 = 2'b00; operand2 = 2'b01; #100;
    opcode = 3'b111; operand1 = 2'b00; operand2 = 2'b01; #100;

  end
endmodule
```

ModelSim Simulation:



Comments: FSM_DECO is designed according to the table provided in the lab manual. The verilog code is first written and then the modelsim testbench to simulate all combinations of the table. The simulation shows that the FSM_DECO works according to the table.

FIBO_FSM)

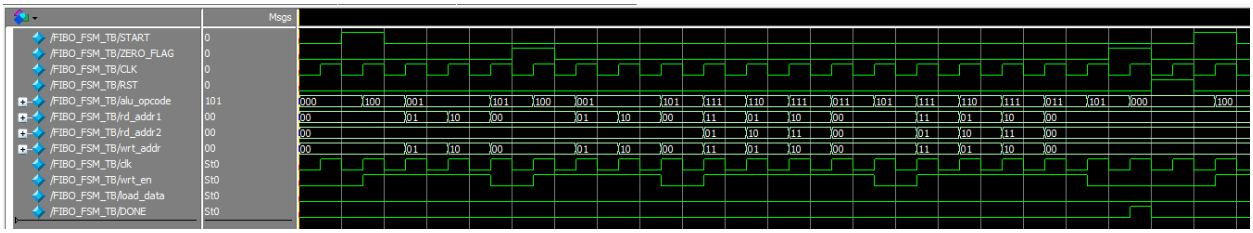
Verilog code:

```
module FIBO_FSM(START, CLK, RST, ZERO_FLAG, wrt_addr, wrt_en, clk, load_data, rd_addr1, rd_addr2, alu_opcode, DONE);

parameter size = 4;

input START, ZERO_FLAG, CLK, RST;

output [size-2:0] alu_opcode;
output [size-3:0] rd_addr1, rd_addr2, wrt_addr;
output clk, wrt_en, load_data, DONE;

wire [size-2:0] opcode;
wire [size-3:0] operand1, operand2;

FSM F1(START, ZERO_FLAG, CLK, RST, DONE, opcode, operand1, operand2);
FSM_DECO F2(opcode, operand1, operand2, alu_opcode, rd_addr1, rd_addr2, wrt_addr, wrt_en, load_data);

assign clk = CLK;

endmodule
```

ModelSim Testbench:

```
module FIBO_FSM_TB();

  parameter size = 4;

  reg START, ZERO_FLAG, CLK, RST;
  wire [size-2:0] alu_opcode;
  wire [size-3:0] rd_addr1, rd_addr2, wrt_addr;
  wire clk, wrt_en, load_data, DONE;

  FIBO_FSM DUT(START, CLK, RST, ZERO_FLAG, wrt_addr, wrt_en, clk, load_data, rd_addr1, rd_addr2, alu_opcode, DONE);


  always begin
      CLK = 0; #50;
      CLK = 1; #50;
  end

  initial begin
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 1; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 1; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 1; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 1; #100;

    START = 1; ZERO_FLAG = 0; RST = 0; #100;
    START = 0; ZERO_FLAG = 0; RST = 0; #100;
  end
endmodule
```

ModelSim Simulation:



Comments: FIBO_FSM is the combination of FSM and FSM_DECO. It is written using top level design and the modelsim simulations verifies that the outputs are correct.

FIBO_CALCULATOR)

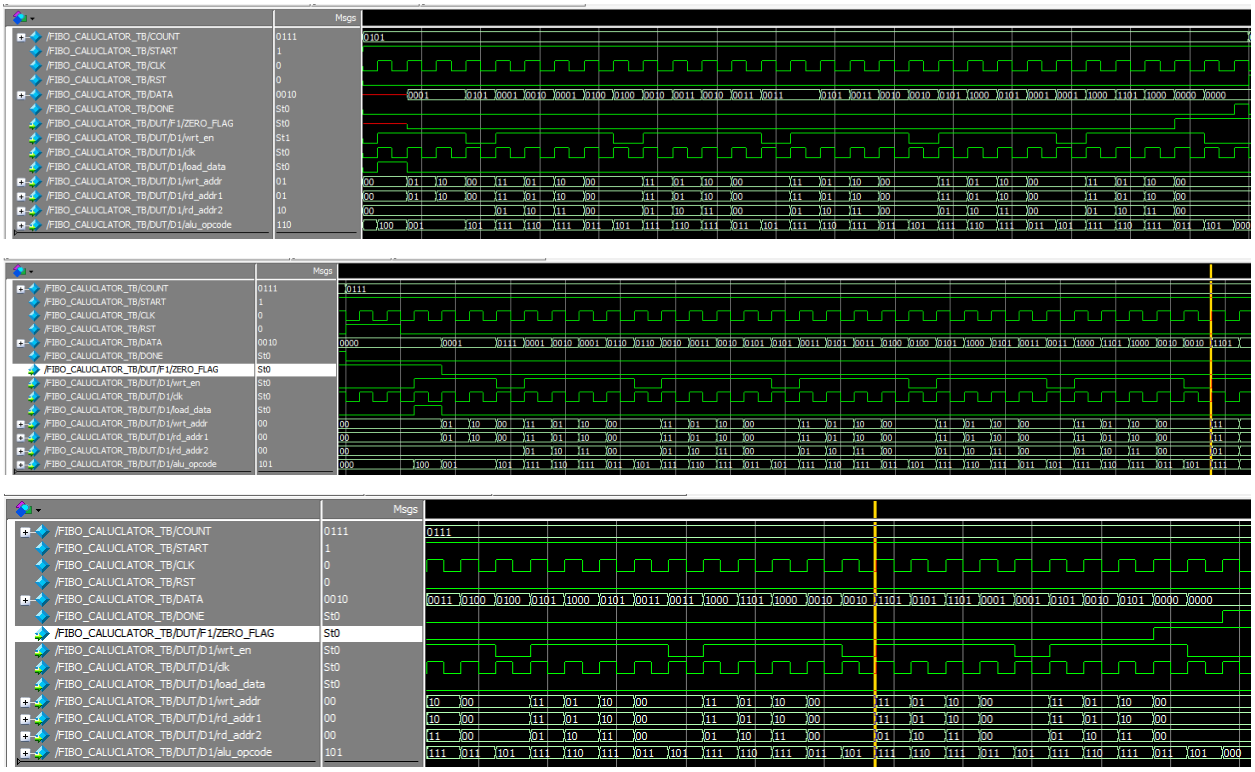
Verilog code:

```
1   module FIBO_CALCULATOR(START, CLK, RST, COUNT, DONE, DATA);
2
3   parameter size = 4;
4
5   input [size-1:0] COUNT;
6   input START, CLK, RST;
7
8   output [size-1:0] DATA;
9   output DONE;
10
11  wire [size-2:0] alu_opcode;
12  wire [size-3:0] rd_addr1, rd_addr2, wrt_addr;
13  wire clk, wrt_en, load_data;
14  wire zero_flag;
15
16  FIBO_FSM F1(START, CLK, RST, zero_flag, wrt_addr, wrt_en, clk, load_data, rd_addr1, rd_addr2, alu_opcode, DONE);
17  FIBO_DATAPATH D1(wrt_addr, wrt_en, clk, load_data, rd_addr1, rd_addr2, alu_opcode, COUNT, zero_flag, DATA);
18
19  endmodule
20
```

ModelSim Testbench:

```
1   module FIBO_CALUCLATOR_TB();
2     parameter size = 4;
3
4     reg [size-1:0] COUNT;
5     reg START, CLK, RST;
6
7     wire [size-1:0] DATA;
8     wire DONE;
9
10    FIBO_CALCULATOR DUT(START, CLK, RST, COUNT, DONE, DATA);
11
12    always begin
13        CLK = 0; #50;
14        CLK = 1; #50;
15    end
16
17    initial begin
18      START = 1; RST = 0; COUNT = 4'b0101; #3000;
19      START = 1; RST = 1; COUNT = 4'b0111; #200;
20      START = 1; RST = 0; COUNT = 4'b0111; #100;
21    end
22
23  endmodule
```

ModelSim Simulation:



Comments: This is the top level design for the whole implementation of fibonacci calculator. This includes the FIBO_FSM and the DATAPATH. The FIBO_FSM controls the outputs and uses the datapath to find the fibonacci. The modelsim simulation is used to first find the fibonacci for 5 sequence then after a reset it calculates for 7 sequences. The simulation shows that the output is correct for the 5 sequences but for 7 sequences since the number gets larger then what 4 bit can store, the output is not as expected. Since we have used a parameterized approach we can easily increase the bit size and overcome this issue. Overall the simulation is successful in showing a correct fibonacci sequence.

FMAX)

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	401.12 MHz	250.0 MHz	CLK	limit due to minimum period restriction (max I/O toggle rate)
2	531.35 MHz	257.2 MHz	FIBO_FSM:F1|FSM:F1|state.S0	limit due to hold check

Comments: Overall, the fibonacci fsm calculator is written in a scalable way to ensure that number of bits can be increased, furthermore every verilog code written is tested using the modelsim testbench to look for any anomaly. I can safely say that my modelsim simulations act accordingly and that the requirements of the lab have been met.