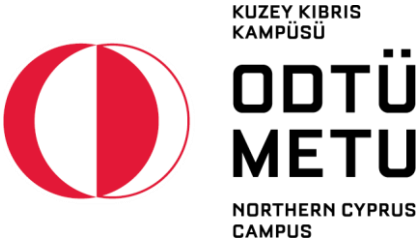


Lab7 Report

Talal Shafei

2542371

Notes:

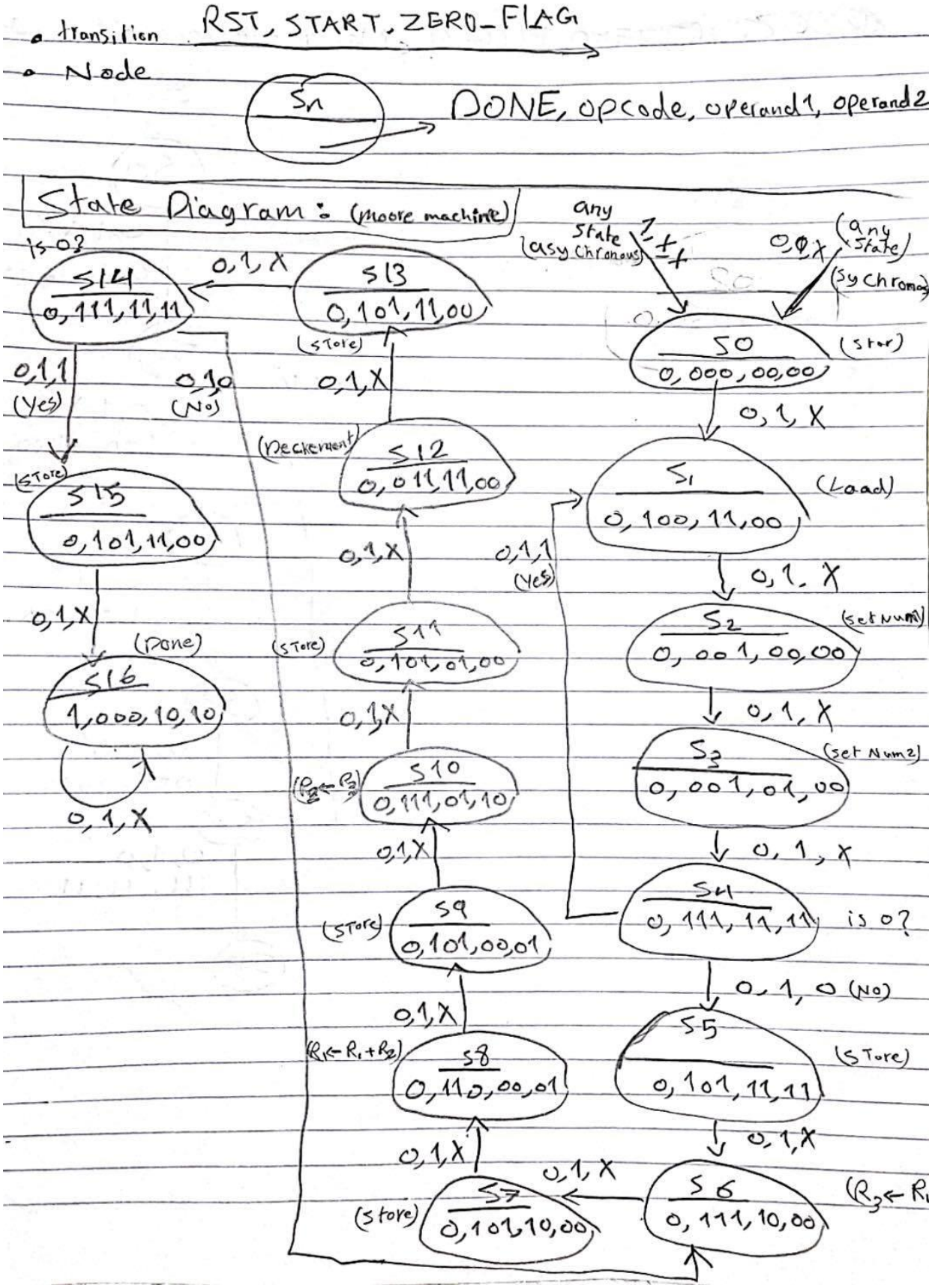
For some reason in Modelsim when I use always block the data is not appearing in the waveform until I use assign keyword even though if I use the assign keyword inside an always block in Quartus it gives me errors.

So my codes in Modelsim has assign kewords inside the always block
My codes in Quartus doesn't

In this report I will show the Modelsim codes

FIBO_FSM

State Diagram



State Table

Present State	Inputs	Next States	Outputs
A	RST, START, ZERO_FLAG	A+	DONE, opcode, operand1, operand2
S0 = 00000	0, 1, X	S1	0, 000, 00, 00
S0	0, 0, X	S0	0, 000, 00, 00
S0	1, X, X	S0	0, 000, 00, 00
S1 = 00001	0, 1, X	S2	0, 100, 11, 00
S1	0, 0, X	S0	0, 100, 11, 00
S1	1, X, X	S0	0, 100, 11, 00
S2 = 00010	0, 1, X	S3	0, 001, 00, 00
S2	0, 0, X	S0	0, 001, 00, 00
S2	1, X, X	S0	0, 001, 00, 00
S3 = 00011	0, 1, X	S4	0, 001, 01, 00
S3	0, 0, X	S0	0, 001, 01, 00
S3	1, X, X	S0	0, 001, 01, 00
S4 = 00100	0, 1, 0	S5	0, 111, 11, 11
S4	0, 1, 1	S1	0, 111, 11, 11
S4	0, 0, X	S0	0, 111, 11, 11
S4	1, X, X	S0	0, 111, 11, 11
S5 = 00101	0, 1, X	S6	0, 101, 11, 11
S5	0, 0, X	S0	0, 101, 11, 11
S5	1, X, X	S0	0, 101, 11, 11
S6 = 00110	0, 1, X	S7	0, 111, 10, 00
S6	0, 0, X	S0	0, 111, 10, 00
S6	1, X, X	S0	0, 111, 10, 00
S7 = 00111	0, 1, X	S8	0, 101, 10, 00
S7	0, 0, X	S0	0, 101, 10, 00
S7	1, X, X	S0	0, 101, 10, 00
S8 = 01000	0, 1, X	S9	0, 110, 00, 01
S8	0, 0, X	S0	0, 110, 00, 01
S8	1, X, X	S0	0, 110, 00, 01
S9 = 01001	0, 1, X	S10	0, 101, 00, 01
S9	0, 0, X	S0	0, 101, 00, 01
S9	1, X, X	S0	0, 101, 00, 01
S10 = 01010	0, 1, X	S11	0, 111, 01, 10
S10	0, 0, X	S0	0, 111, 01, 10
S10	1, X, X	S0	0, 111, 01, 10
S11 = 01011	0, 1, X	S12	0, 101, 01, 00
S11	0, 0, X	S0	0, 101, 01, 00
S11	1, X, X	S0	0, 101, 01, 00
S12 = 01100	0, 1, X	S13	0, 011, 11, 00
S12	0, 0, X	S0	0, 011, 11, 00
S12	1, X, X	S0	0, 011, 11, 00
S13 = 01101	0, 1, X	S14	0, 101, 11, 00
S13	0, 0, X	S0	0, 101, 11, 00
S13	1, X, X	S0	0, 101, 11, 00
S14 = 01110	0, 1, 0	S6	0, 111, 11, 11
S14	0, 1, 1	S15	0, 111, 11, 11
S14	0, 0, X	S0	0, 111, 11, 11

S14	1, X, X	S0	0, 111, 11, 11
S15 = 01111	0, 1, X	S16	0, 101, 11, 00
S15	0, 0, X	S0	0, 101, 11, 00
S15	1, X, X	S0	0, 101, 11, 00
S16 = 10000	0, 1, X	S16	1, 000, 10, 10
S16	0, 0, X	S0	1, 000, 10, 10
S16	1, X, X	S0	1, 000, 10, 10

Comments: State diagram based on the flow chart in the manual
We can see that every box in the flow chart has a similar node to it in the state diagram plus the states that are used to store the values after transferring data operation

Also when RST is 1 the next state is S0 for all states (Asynchronous reset)
But when START is 0 and RST is 0 the next state will be S0 for all states (Synchronous reset)

We don't care about the value of ZERO_FLAG unless we are at S4 or S14 (the states where we check if the count is 0)

Also, there is a lot of states where we don't care about the value of operand2 but I assigned 00 to use it in Verilog instead of writing xx

Verilog

```
module FIBO_FSM(START,ZERO_FLAG,RST,CLK,DONE,opcode,operand1,operand2,Clk_out);

input START,ZERO_FLAG,RST,CLK;

output reg[2:0]opcode;
output reg[1:0]operand1,operand2;
output reg DONE;
output Clk_out;

parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6, S7 = 7,
                          S8 = 8, S9 = 9, S10 = 10, S11 = 11, S12 = 12, S13 = 13, S14 = 14,
                          S15 = 15, S16 = 16;

reg [5:0]state,nextState; // size of the register 5 bits because of the 16 states

initial begin // initiallizing
        state = S0;
        nextState = S0;
end

always@(posedge CLK or posedge RST)begin // choose whether reset or proceed
        if(RST)//asychronous reset
                state <= S0;

        else
                state <= nextState;

end

always@(state)begin
        case(state)
                S0: assign {DONE,opcode,operand1,operand2} = 0;
                S1: assign {DONE,opcode,operand1,operand2} = 8'b01001100;
```

```
                S2: assign {DONE,opcode,operand1,operand2} = 8'b00010000;
                S3: assign {DONE,opcode,operand1,operand2} = 8'b00010100;
                S4: assign {DONE,opcode,operand1,operand2} = 8'b01111111;
                S5: assign {DONE,opcode,operand1,operand2} = 8'b01011111;
                S6: assign {DONE,opcode,operand1,operand2} = 8'b01111000;
                S7: assign {DONE,opcode,operand1,operand2} = 8'b01011000;
                S8: assign {DONE,opcode,operand1,operand2} = 8'b01100001;
                S9: assign {DONE,opcode,operand1,operand2} = 8'b01010001;
                S10:assign {DONE,opcode,operand1,operand2} = 8'b01110110;
                S11:assign {DONE,opcode,operand1,operand2} = 8'b01010100;
                S12:assign {DONE,opcode,operand1,operand2} = 8'b00111100;
                S13:assign {DONE,opcode,operand1,operand2} = 8'b01011100;
                S14:assign {DONE,opcode,operand1,operand2} = 8'b01111111;
                S15:assign {DONE,opcode,operand1,operand2} = 8'b01011100;
                S16:assign {DONE,opcode,operand1,operand2} = 8'b10001010;

        endcase

end

assign Clk_out = CLK; // to send the clock signals to the Datapath

always@(state or START or ZERO_FLAG)begin // computing next state
        nextState = S0;

        if(START == 0)//sychronous reset
                nextState <= S0;

        else begin
                case(state)
                        S0: nextState = S1;
                        S1: nextState = S2;
                        S2: nextState = S3;
                        S3: nextState = S4;
                        S4:begin
                                        if(!ZERO_FLAG)
                                                nextState = S5;
                                        else
                                                nextState = S1;
                                end
                        S5: nextState = S6;
                        S6: nextState = S7;
                        S7: nextState = S8;
                  S8: nextState = S9;
                        S9: nextState = S10;
                        S10: nextState = S11;
                        S11: nextState = S12;
                        S12: nextState = S13;
                        S13: nextState = S14;
                        S14:begin
                                        if(ZERO_FLAG)
                                                nextState = S15;
                                        else
                                                nextState = S6;
                                end
```

```
                        S15: nextState = S16;
                        S16: nextState = S16;

                endcase
        end
end


endmodule
```

Comments: the code was written based on the table above.
Clk_out is hust a wire to connect the CLK to the Datapath.

First initializing current state and the next state

Second, we check if we should reset (RST=1) or proceed (RST=0) to the next state

Third, we assign the outputs based on the current state

finally, the next state when RST=0 and START = 1 for Sn is Sn+1
except for S4 which is the S1 if ZERO_FLAG = 1 and S5 if ZERO_FLAG = 0
and for S14 which is S6 if ZERO_FLAG = 0 and S15 if ZERO_FLAG = 1
and S16 is 16

also, all the states reset if RST = 0 and START = 0 to S0

FIBO_DECO

Verilog

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

input [2:0]opcode;
input [1:0] operand1,operand2;

output [2:0]alu_opcode;
output [1:0] rd_addr1,rd_addr2,wrt_addr;
output wrt_en,load_data;

assign alu_opcode = opcode;
assign rd_addr1 = operand1;
assign rd_addr2  = operand2;

assign wrt_addr  = operand1;
assign wrt_en    = (opcode[2] || opcode[1] || opcode[0]) & ((~opcode[2]) || opcode[1] || (~opcode[0])); //
if opcode  == 000 or 101
assign load_data = opcode[2] & (~opcode[1]) & (~opcode[0]); // if opcode == 100

endmodule
```

Comments: the FSM decoder is simple assigning opcode to alu_opcode and assigning
Rd_addr1 and wrt_addr to operand 1
And assigning rd_addr2 to operand 2

And writing the logic for load_data and wrt_en from the table using and, or, inverter gates

FSM

Verilog

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

input START,ZERO_FLAG,RST,CLK;

output DONE,Clk_out;

output [2:0]alu_opcode;
output [1:0] rd_addr1,rd_addr2,wrt_addr;
output wrt_en,load_data;

wire [2:0]opcode;
wire [1:0]operand1,operand2;


FIBO_FSM f(START,ZERO_FLAG,RST,CLK,DONE,opcode,operand1,operand2,Clk_out);

FIBO_DECO d(opcode,operand1,operand2,alu_opcode,rd_addr1,rd_addr2,wrt_addr,wrt_en,load_data);

endmodule
```

Comments: this block connect the FSM code to the decoder

TOP_DESIGN

Verilog

```
module Calculator_top(START,RST,CLK,count,DONE,data_out);

parameter size = 4;

input START,RST,CLK;
input [size-1:0]count;
output DONE;
output [size-1:0]data_out;


wire ZERO_FLAG,Clk_out,wrt_en,load_data;
wire [1:0]wrt_addr,rd_addr1,rd_addr2;
wire [2:0]alu_opcode;

FSM
F(START,ZERO_FLAG,RST,CLK,DONE,Clk_out,alu_opcode,rd_addr1,rd_addr2,wrt_addr,wrt_en,loa
d_data);

FIBO_DATAPATH D
```

```
(wrt_addr,wrt_en,Clk_out,load_data,rd_addr1,rd_addr2,alu_opcode,count,data_out,ZERO_FLAG);


endmodule
```

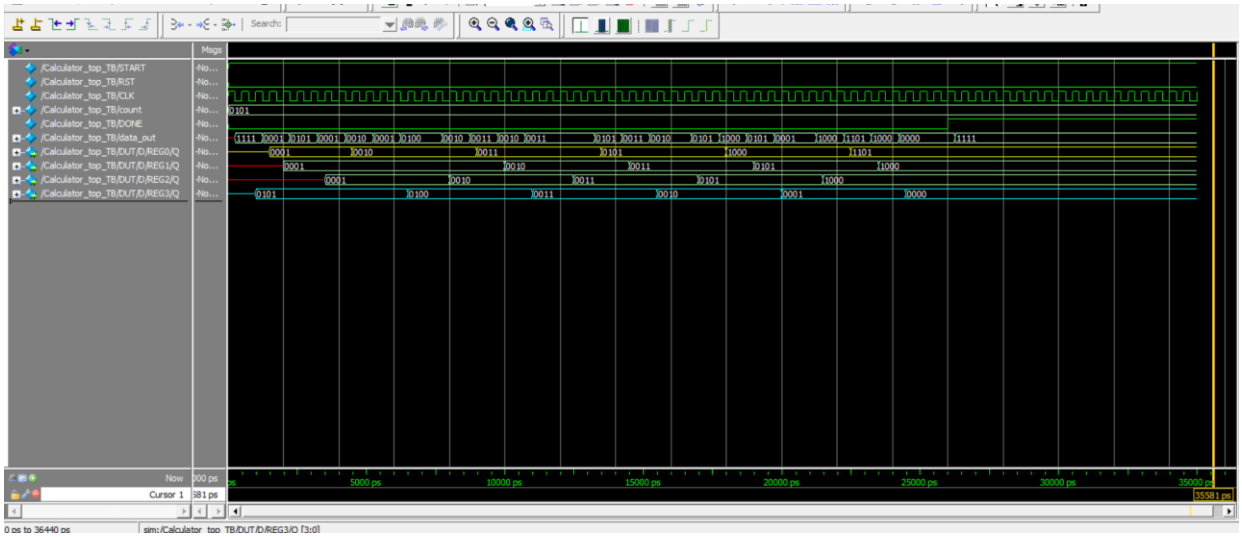
Comments: this block connect the FSM with the Datapath to have a Fibonacci series calculator

Test Bench

```
module Calculator_top_TB();

reg START,RST,CLK;
reg [3:0]count;
wire DONE;
wire [3:0]data_out;



Calculator_top DUT(START,RST,CLK,count,DONE,data_out);

always #250 CLK = ~CLK;

initial begin
CLK=1;
RST=0;
count = 5;
START = 1;
end




endmodule
```

Comments: the test bench will test the codes above and find the first 7 sequence of Fibonacci series Count is set to 5 because in the flow chart we set Reg0 and Reg1 to 1 so already have the values of F(1) and F(2) so to find F(7) count must be 5

Simulation



Comments: The yellow line represent the Reg0 (Reg1 in the manual) and this register will store the value of Fibonacci sequence we can see that F(0) is skipped because as said above we set Reg0 to 1 and

Reg1 to 1 so we start counting from after the first two index

At the end of the waveform Reg0 is 1101 (13) and Reg3 (count) is 0
So we know that the code gives the expected output

Also Reg1 and Reg2 store F(n-1) which we can see from the wave form

Note: Reg0 to Reg3 are not outputs of the Calculator I just included them in the wave form

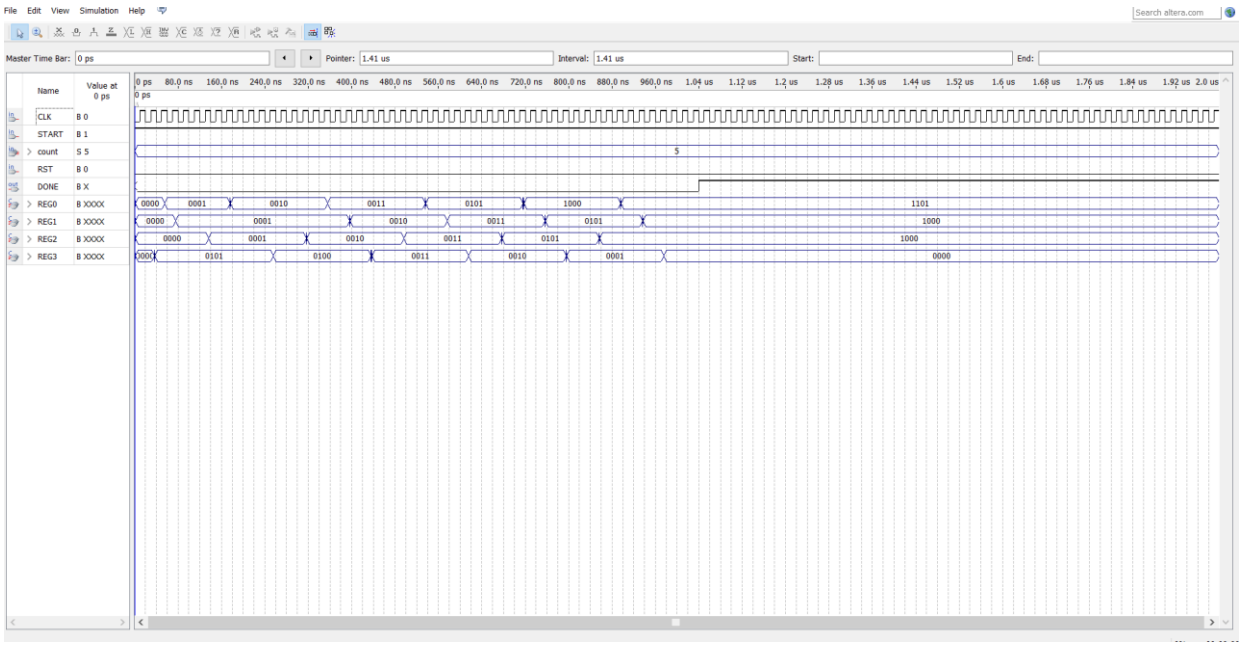
Timing analysis

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	57.23 MHz	57.23 MHz	CLK	

Comments: from the image fmax is 57.23 MHz so the least clock period we can use in the FPGA is 1/fmax which is 17.47 ns to make sure the design is functioning properly.

Time Simulation



Comments: Timing simulation testing using 20 ns Clock period
The output as expected the same as the Modelsim simulation
Note: Reg0 to Reg3 are not outputs of the Calculator I just included them in the wave form