EEE 248/ CNG 232
Logic Design

Project 4 - week 14

Yıldız Alara Köymen

2453389

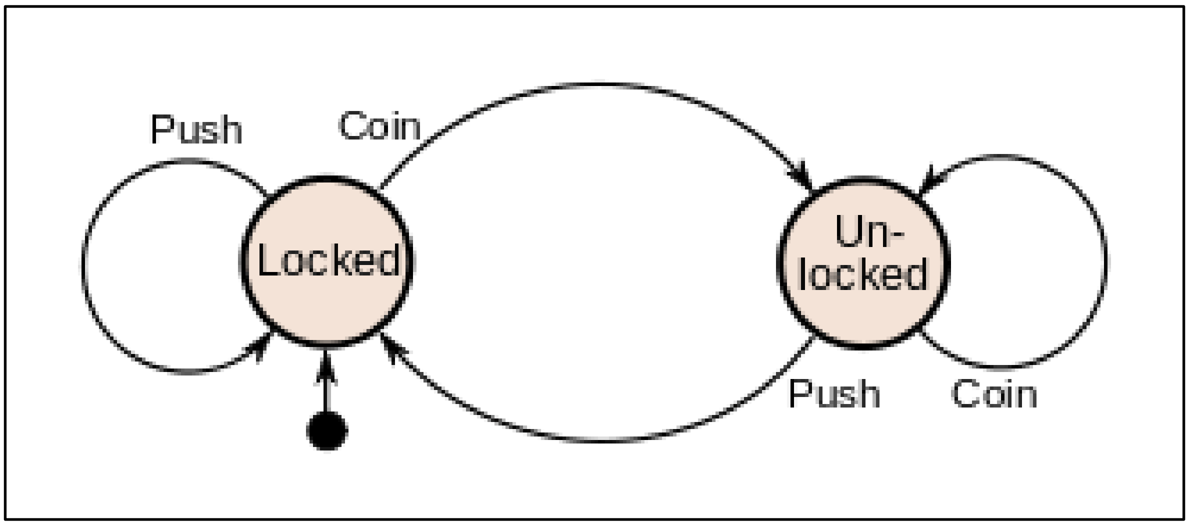
TABLE OF CONTENTS

5.1: Introduction:

We will be connecting the FSM to the datapath we made last week.

What is an FSM?

A finite-state machine or finite-state automaton, finite automaton, or simply a state machine, is a mathematical model of computation.



5.3: FSM Design:

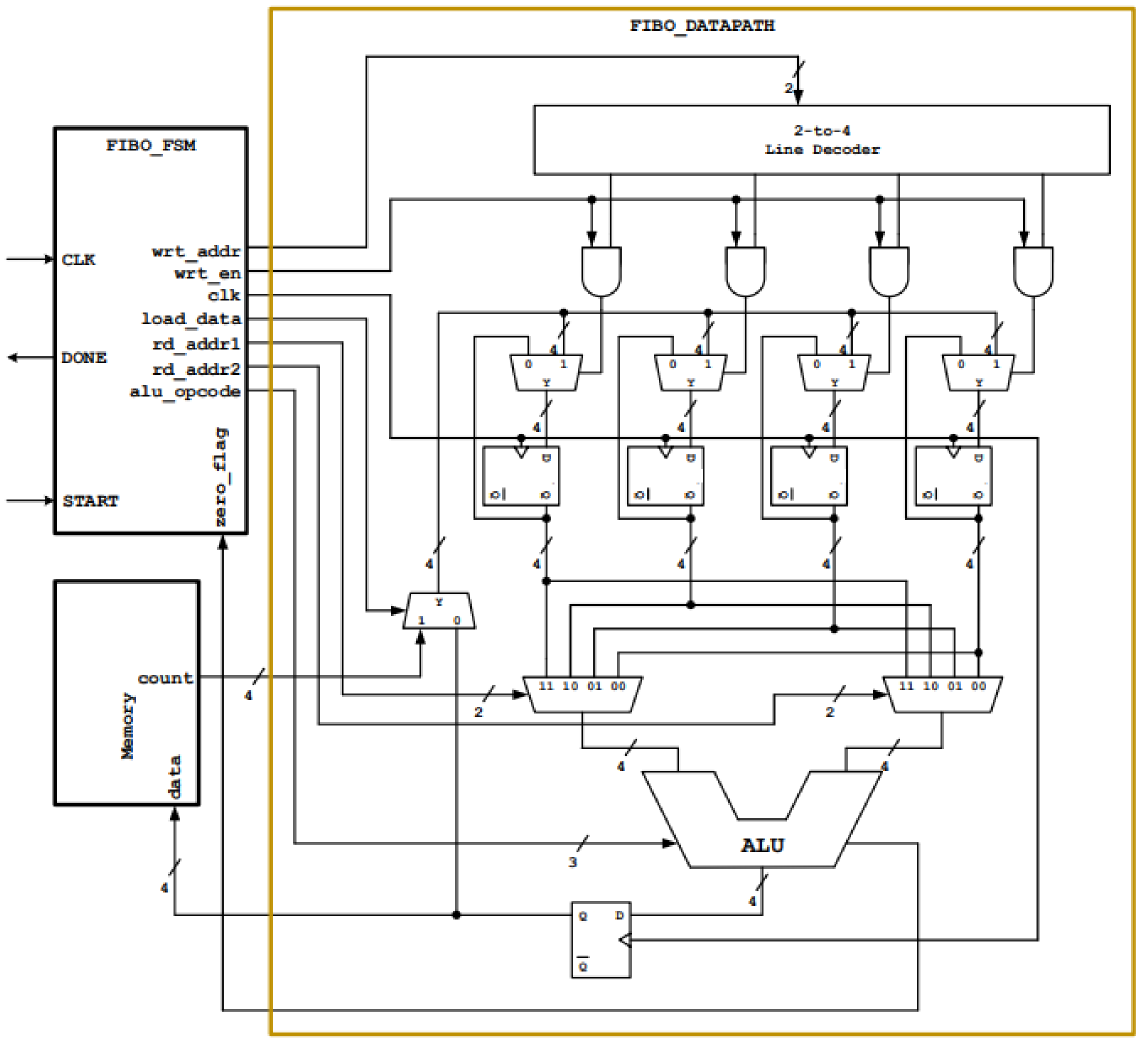


Figure 1: Fibonacci Series Calculator FSM & Datapath

The control unit of this Datapath will be implemented by the FIBO_FSM. FIBO_FSM is divided into two main subcomponents as FSM and FSM_DECO.

The FSM will have four external inputs START, ZERO_FLAG, CLK and RST and one output DONE to indicate that the calculation is completed. This component performs all the required control operations providing three outputs as: opcode, operand1 and operand2.

The FSM_DECO will use these three outputs as an input and decode the signals as stated in Table 3. The output of the FSM_DECO will be used to control the Fibonacci Series calculation on the Datapath as described in Figure 2.

Table 3: FSM_DECO Operations.

Op	op_code	Opr1	Opr2	ALU	Rd_addr1	Rd_addr2	wrt_addr	wrt_en	load_data
noop	000	--	--	000	--	--	--	0	-
set	001	xx	--	001	--	--	xx	1	0
inc	010	xx	--	010	xx	--	xx	1	0
dec	011	xx	--	011	xx	--	xx	1	0
load	100	xx	--	100	--	--	xx	1	1
store	101	xx	--	101	xx	--	--	0	-
add	110	xx	yy	110	xx	yy	xx	1	0
copy	111	xx	yy	111	yy	--	Xx	1	0

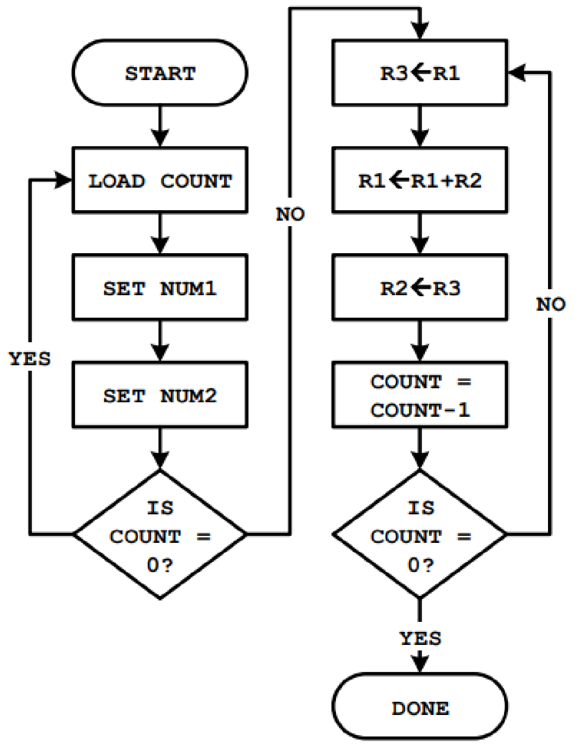


Figure 2: Fibonacci Series Hardware Algorithm.

Predefined COUNT will be stored in a memory. Once a START signal is asserted, the Fibonacci Series calculation process will start and continue until the down counter = 0.

Draw the state diagram for the control FSM showing all the inputs, outputs, and the state transitions clearly. Indicate the opcode required for each state.

FSM:

Inputs: Start

Reset

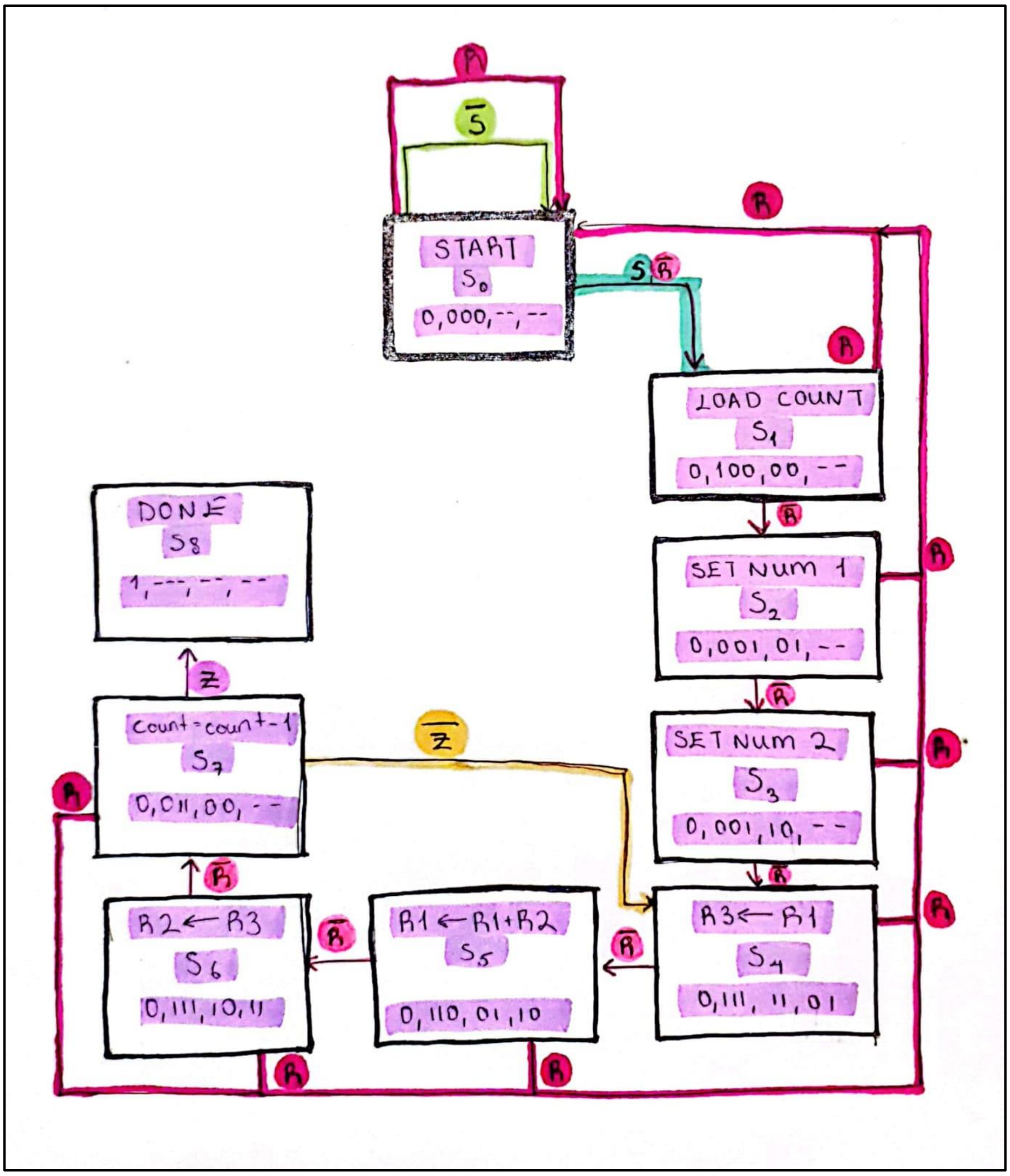
Zero_flag

Outputs: Done

Opcode

Operand1

Operand2

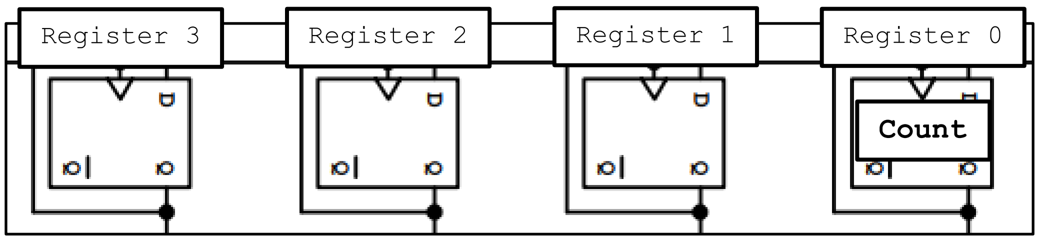


S0: Start:

At state S0 the state will stay at S0 as the input Start is 0 or when Reset is 1. It will only move forward to S1 at Start = 1 and Reset = 0. We can’t talk about input Zero_flag as it will only be 1 when the count value becomes 0. The opcode would be 000 as no operation is done. Operand1 and Operand2 outputs would be don’t care values. Done would be 0.

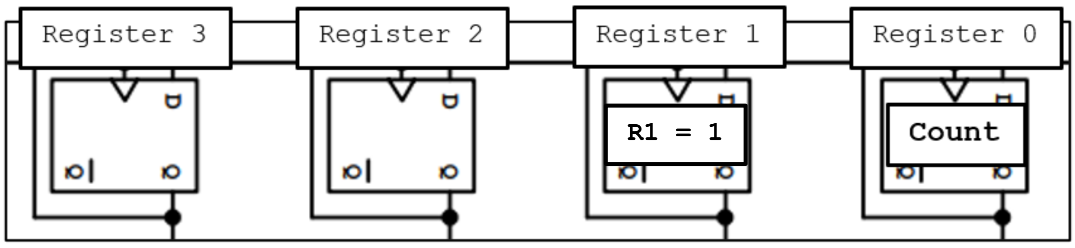
S1: Load count:

At this point, it doesn’t matter what Start is. Since Start would have been given the FSM will continue the operation. When Reset is 1 the next state will go back to S0. The opcode is 100 because we are loading. Operand1 is 00 because we are loading to register 0. Operand2 is a don’t care value. Done is 0.



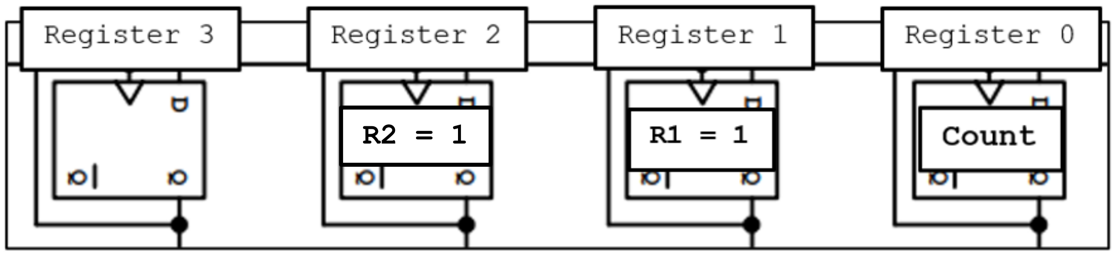
S2: Set Num 1:

When Reset is 1 the next state will go back to S0. The opcode is 001 because we are setting. Operand1 is 01 because we are setting register 1’s value. Operand2 is a don’t care value. Done is 0.



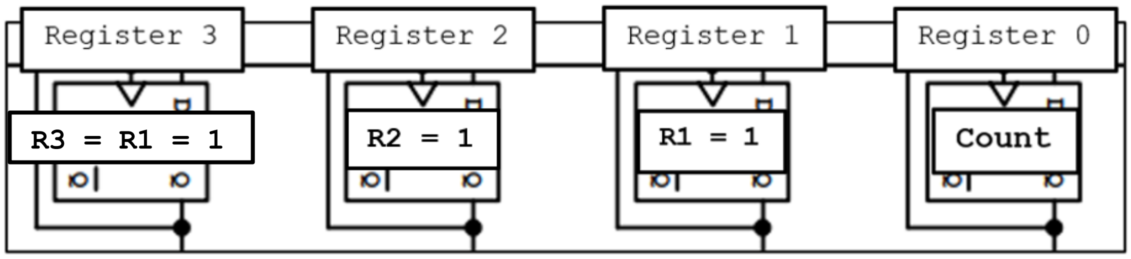
S3: Set Num 2:

When Reset is 1 the next state will go back to S0. The opcode is 001 because we are setting. Operand1 is 10 because we are setting register 2’s value. Operand2 is a don’t care value. Done is 0.



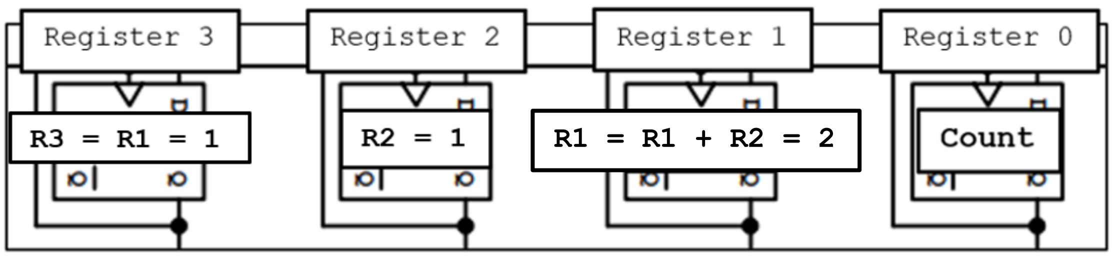
S4: R3 <- R1

When Reset is 1 the next state will go back to S0. The opcode is 111 because we are copying. Operand1 is 11 and Operand2 is 01 because we are copying R1’s value on R3. Done is 0.



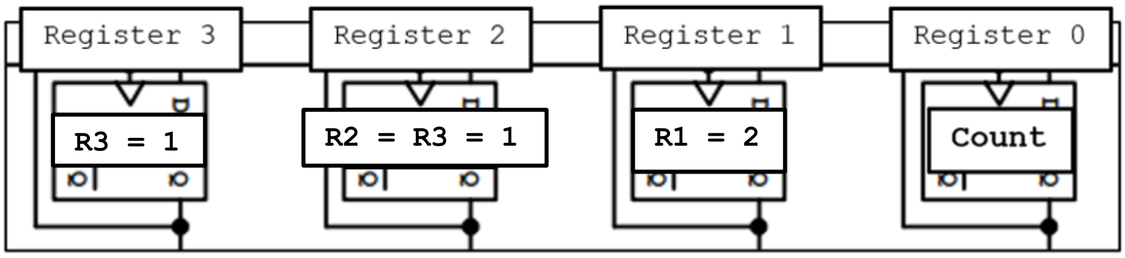
S5: R1 <- R1 + R2

When Reset is 1 the next state will go back to S0. The opcode is 110 because we are adding. Operand1 is 01 and Operand2 is 10 because we are adding R1 and R2 and then copying that value to R1.



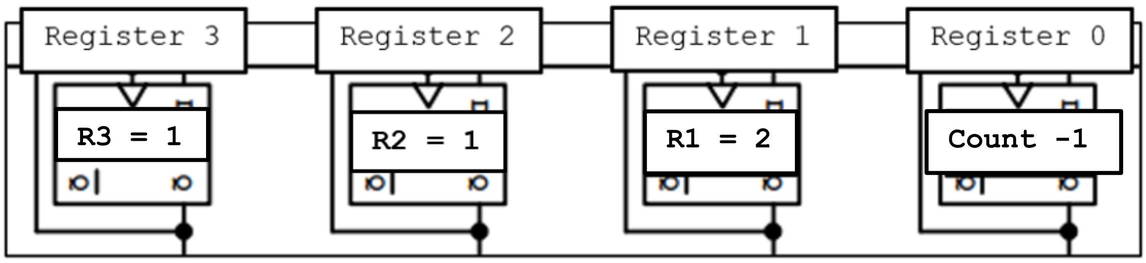
S6: R2 <- R3

When Reset is 1 the next state will go back to S0. The opcode is 111 because we are copying. Operand1 is 10 and Operand2 is 11 because we are copying R3’s value on R2. Done is 0.



S7: Count = count - 1

When Reset is 1 the next state will go back to S0. The opcode is 011 because we are decrementing. Operand1 is 00 because count value is stored on R0. If count becomes 0 after decrementing, zero-flag will become 1. When zero_flag is 1, Done will be 1 and the operaton is done. If count is not 0 and zero_flag is never given, the operation will loop back to S4.



Let’s see if how this algorithm would work for count = 5:

Count = 5:

S0:

R3, R2, R1, R0

S1:

R3, R2, R1, R0 = 5

S2:

R3, R2, R1 = 1, R0 = 5

S3:

R3, R2 = 1, R1 = 1, R0 = 5

S4:

R3 = R1 = 1, R2 = 1, R1 = 1, R0 = 5

S5:

R3 = 1, R2 = 1, R1 = R1 + R2 = 1 + 1 = 2, R0 = 5

S6:

R3 = 1, R2 = R3 = 1, R1 = 2, R0 = 5

S7:

R3 = 1, R2 = 1, R1 = 2, R0 = 5 - 1 = 4

S4:

R3 = R1 = 2, R2 = 1, R1 = 2, R0 = 4

S5:

R3 = 2, R2 = 1, R1 = R1 + R2 = 2 + 1 = 3, R0 = 4

S6:

R3 = 2, R2 = R3 = 2, R1 = 3, R0 = 4

S7:

R3 = 2, R2 = 2, R1 = 3, R0 = 4 - 1 = 3

S4:

R3 = R1 = 3, R2 = 2, R1 = 3, R0 = 3

S5:

R3 = 3, R2 = 2, R1 = R1 + R2 = 3 + 2 = 5, R0 = 3

S6:

R3 = 3, R2 = R3 = 3, R1 = 5, R0 = 3

S7:

R3 = 3, R2 = 3, R1 = 5, R0 = 3 - 1 = 2

S4:

R3 = R1 = 5, R2 = 3, R1 = 5, R0 = 2

S5:

R3 = 5, R2 = 3, R1 = R1 + R2 = 5 + 3 = 8, R0 = 2

S6:

R3 = 5, R2 = R3 = 5, R1 = 8, R0 = 2

S7:

R3 = 5, R2 = 5, R1 = 8, R0 = 2 - 1 = 1

S4:

R3 = R1 = 8, R2 = 5, R1 = 8, R0 = 1

S5:

R3 = 8, R2 = 5, R1 = R1 + R2 = 8 + 5 = 13, R0 = 1

S6:

R3 = 8, R2 = R3 = 8, R1 = 13, R0 = 1

S7:

R3 = 8, R2 = 8, R1 = 13, R0 = 0

S8:

DONE = 1

Fibonnaci series is calculated with F(n) = F(n-2) + F(n-1).

F(1) = 1.

F(2) = 1.

F(3) = F(1) + F(2) = 1 + 1 = 2.

F(4) = F(2) + F(3) = 1 + 2 = 3.

F(5) = F(3) + F(4) = 2 + 3 = 5.

F(6) = F(4) + F(5) = 3 + 5 = 8.

F(7) = F(5) + F(6) = 5 + 8 = 13.

However we can see that we achieved The 7th fibonacci number rather than the 5th number while calculating for count = 5. This means we can write the new formula for fibonnaci series (for this FSM) should be:

F(n+2) = F(n) + F(n+1)

5.3.1: FSM_DECO:

FIBO_FSM is divided into two main subcomponents FSM and FSM_DECO.

For FSM_DECO will use op_code, operand1 and operand2 as an output and decode the signals as stated in Table 3.

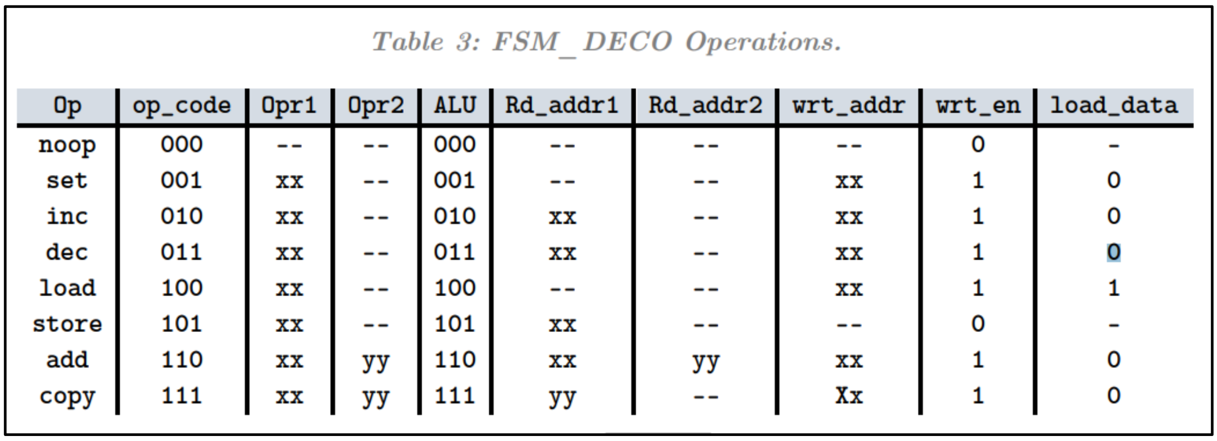
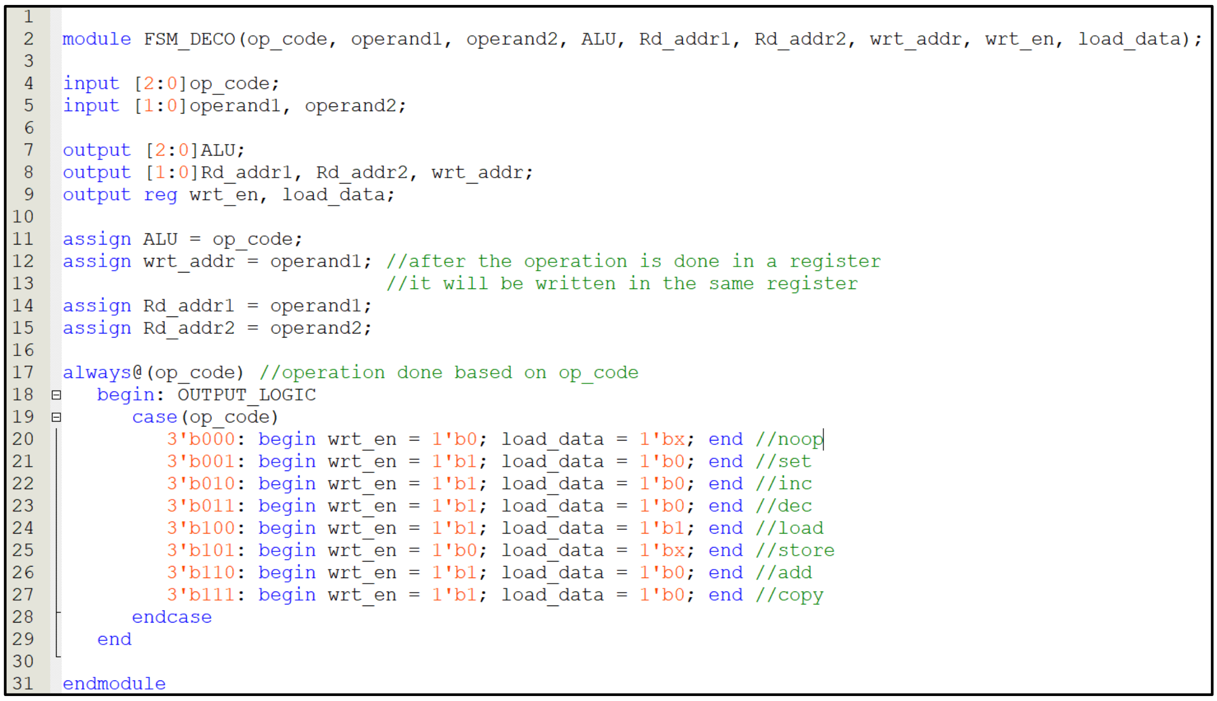


Table 3: FSM_DECO Operations.

Op	op_code	Opr1	Opr2	ALU	Rd_addr1	Rd_addr2	wrt_addr	wrt_en	load_data
noop	000	--	--	000	--	--	--	0	-
set	001	xx	--	001	--	--	xx	1	0
inc	010	xx	--	010	xx	--	xx	1	0
dec	011	xx	--	011	xx	--	xx	1	0
load	100	xx	--	100	--	--	xx	1	1
store	101	xx	--	101	xx	--	--	0	-
add	110	xx	yy	110	xx	yy	xx	1	0
copy	111	xx	yy	111	yy	--	Xx	1	0

So I implemented the table for the FSM_DECO code.

The outputs are ALU, Rd_addr1, Rd_addr2, wrt_addr, wrt_en and load_data.


```

module FSM_DECO(op_code, operand1, operand2, ALU, Rd_addr1, Rd_addr2, wrt_addr, wrt_en, load_data);

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

output [2:0]ALU;
output [1:0]Rd_addr1, Rd_addr2, wrt_addr;
output reg wrt_en, load_data;

assign ALU = op_code;
assign wrt_addr = operand1; //after the operation is done in a register
                            //it will be written in the same register
assign Rd_addr1 = operand1;
assign Rd_addr2 = operand2;

always@(op_code) //operation done based on op_code
   begin: OUTPUT_LOGIC
      case(op_code)
         3'b000: begin wrt_en = 1'b0; load_data = 1'bx; end //noop
         3'b001: begin wrt_en = 1'b1; load_data = 1'b0; end //set
         3'b010: begin wrt_en = 1'b1; load_data = 1'b0; end //inc
         3'b011: begin wrt_en = 1'b1; load_data = 1'b0; end //dec
         3'b100: begin wrt_en = 1'b1; load_data = 1'b1; end //load
         3'b101: begin wrt_en = 1'b0; load_data = 1'bx; end //store
         3'b110: begin wrt_en = 1'b1; load_data = 1'b0; end //add
         3'b111: begin wrt_en = 1'b1; load_data = 1'b0; end //copy
      endcase
   end

endmodule
```

ALU is assigned op_code as shown in table.

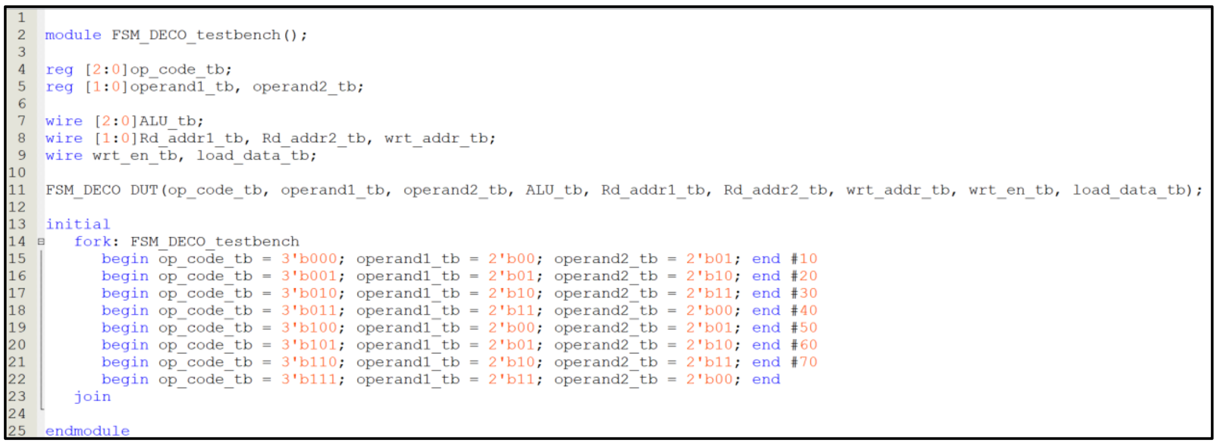
Wr_addr is assigned op_code because the operations made in the ALU will be written back to the same register.

Rd_addr1 is assigned operand1 as shown in table.

Rd_addr2 is assigned operand2 as shown in table.

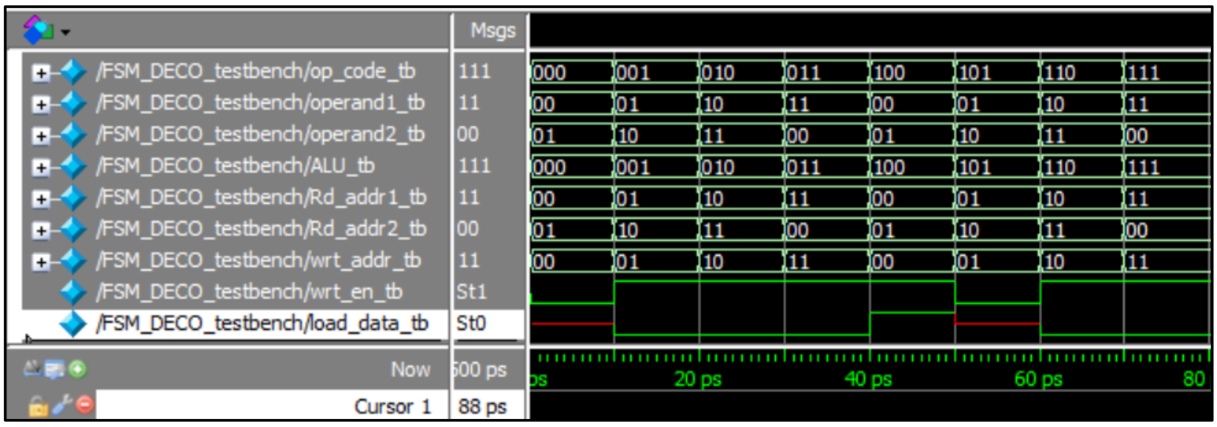
Operation on wrt_en and load_data will be done based on op_code so op_code was written in an always block. The statements are as the table has shown.

Testbench code:



```

module FSM_DECO_testbench();

reg [2:0]op_code_tb;
reg [1:0]operand1_tb, operand2_tb;

wire [2:0]ALU_tb;
wire [1:0]Rd_addr1_tb, Rd_addr2_tb, wrt_addr_tb;
wire wrt_en_tb, load_data_tb;

FSM_DECO DUT(op_code_tb, operand1_tb, operand2_tb, ALU_tb, Rd_addr1_tb, Rd_addr2_tb, wrt_addr_tb, wrt_en_tb, load_data_tb);

initial
   fork: FSM_DECO_testbench
      begin op_code_tb = 3'b000; operand1_tb = 2'b00; operand2_tb = 2'b01; end #10
      begin op_code_tb = 3'b001; operand1_tb = 2'b01; operand2_tb = 2'b10; end #20
      begin op_code_tb = 3'b010; operand1_tb = 2'b10; operand2_tb = 2'b11; end #30
      begin op_code_tb = 3'b011; operand1_tb = 2'b11; operand2_tb = 2'b00; end #40
      begin op_code_tb = 3'b100; operand1_tb = 2'b00; operand2_tb = 2'b01; end #50
      begin op_code_tb = 3'b101; operand1_tb = 2'b01; operand2_tb = 2'b10; end #60
      begin op_code_tb = 3'b110; operand1_tb = 2'b10; operand2_tb = 2'b11; end #70
      begin op_code_tb = 3'b111; operand1_tb = 2'b11; operand2_tb = 2'b00; end
   join

endmodule
```

In the testbench, the workings of the op_code was shown. So operand1 and operand2 were written in random.


Msgs
/FSM_DECO_testbench/op_code_tb 111
/FSM_DECO_testbench/operand1_tb 11
/FSM_DECO_testbench/operand2_tb 00
/FSM_DECO_testbench/ALU_tb 111
/FSM_DECO_testbench/Rd_addr1_tb 11
/FSM_DECO_testbench/Rd_addr2_tb 00
/FSM_DECO_testbench/wrt_addr_tb 11
/FSM_DECO_testbench/wrt_en_tb St1
/FSM_DECO_testbench/load_data_tb St0
000 001 010 011 100 101 110 111
00 01 10 11 00 01 10 11
01 10 11 00 01 10 11 00
000 001 010 011 100 101 110 111
00 01 10 11 00 01 10 11
01 10 11 00 01 10 11 00
00 01 10 11 00 01 10 11
Now 500 ps
Cursor 1 88 ps
ps
20 ps
40 ps
60 ps
80

<u>Opcode = 000:</u>

Wrt_en is 0.

Load_data is x.

<u>Opcode = 001:</u>

Wrt_en is 1.

Load_data is 0.

Opcode = 010:

Wrt_en is 1.

Load_data is 0.

Opcode = 011:

Wrt_en is 1.

Load_data is 0.

Opcode = 100:

Wrt_en is 1.

Load_data is 1.

Opcode = 101:

Wrt_en is 0.

Load_data is x.

Opcode = 110:

Wrt_en is 1.

Load_data is 0.

Opcode = 111:

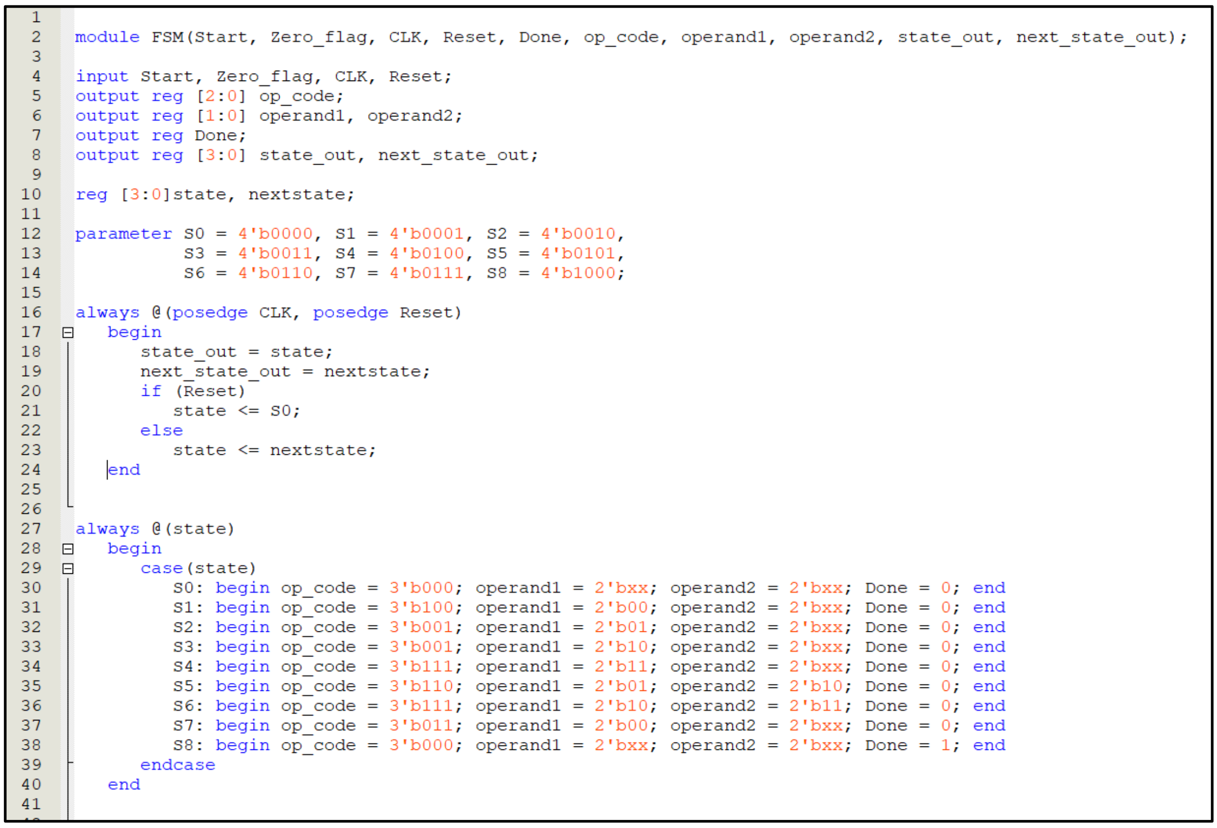
Wrt_en is 1.

Load_data is 0.

5.3.2: FSM:

FSM will have four external inputs Start, Zero_flag, CLK and Reset. It will have outputs op_code, operand1, operand2 and Done.

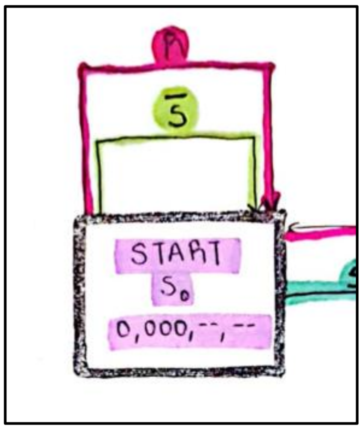
FSM will later be connected to FSM_DECO at top level and create FIBO_FSM.



```
 1
 2  module FSM(Start, Zero_flag, CLK, Reset, Done, op_code, operand1, operand2, state_out, next_state_out);
 3
 4  input Start, Zero_flag, CLK, Reset;
 5  output reg [2:0] op_code;
 6  output reg [1:0] operand1, operand2;
 7  output reg Done;
 8  output reg [3:0] state_out, next_state_out;
 9
10  reg [3:0]state, nextstate;
11
12  parameter S0 = 4'b0000, S1 = 4'b0001, S2 = 4'b0010,
13            S3 = 4'b0011, S4 = 4'b0100, S5 = 4'b0101,
14            S6 = 4'b0110, S7 = 4'b0111, S8 = 4'b1000;
15
16  always @(posedge CLK, posedge Reset)
17      begin
18          state_out = state;
19          next_state_out = nextstate;
20          if (Reset)
21              state <= S0;
22          else
23              state <= nextstate;
24      end
25
26
27  always @(state)
28      begin
29          case(state)
30              S0: begin op_code = 3'b000; operand1 = 2'bxx; operand2 = 2'bxx; Done = 0; end
31              S1: begin op_code = 3'b100; operand1 = 2'b00; operand2 = 2'bxx; Done = 0; end
32              S2: begin op_code = 3'b001; operand1 = 2'b01; operand2 = 2'bxx; Done = 0; end
33              S3: begin op_code = 3'b001; operand1 = 2'b10; operand2 = 2'bxx; Done = 0; end
34              S4: begin op_code = 3'b111; operand1 = 2'b11; operand2 = 2'bxx; Done = 0; end
35              S5: begin op_code = 3'b110; operand1 = 2'b01; operand2 = 2'b10; Done = 0; end
36              S6: begin op_code = 3'b111; operand1 = 2'b10; operand2 = 2'b11; Done = 0; end
37              S7: begin op_code = 3'b011; operand1 = 2'b00; operand2 = 2'bxx; Done = 0; end
38              S8: begin op_code = 3'b000; operand1 = 2'bxx; operand2 = 2'bxx; Done = 1; end
39          endcase
40      end
41
```

In the code there are state_out and next_state_out to help me see if the output is correct or not.

For line 16-25 there is an always block that is working with CLK and Reset. For each CLK posedge, the state is assigned S0 if Reset is 1. For other cases state is assigned nextstate.



In the state diagram it was shown that if Reset is 1 the operation restarts from S0.

Start = 0 didn't need to be stated since the operation won't start until Start is 1.

For lines 27-40 What was written in the state graph was itterated.

For S0: op_code = 000 which is no operation.

For S1: op_code = 100 which is loading register 00 or R0.

For S2: op_code = 001 which is setting register 01 or R1.

For S3: op_code = 001 which is setting register 10 or R2.

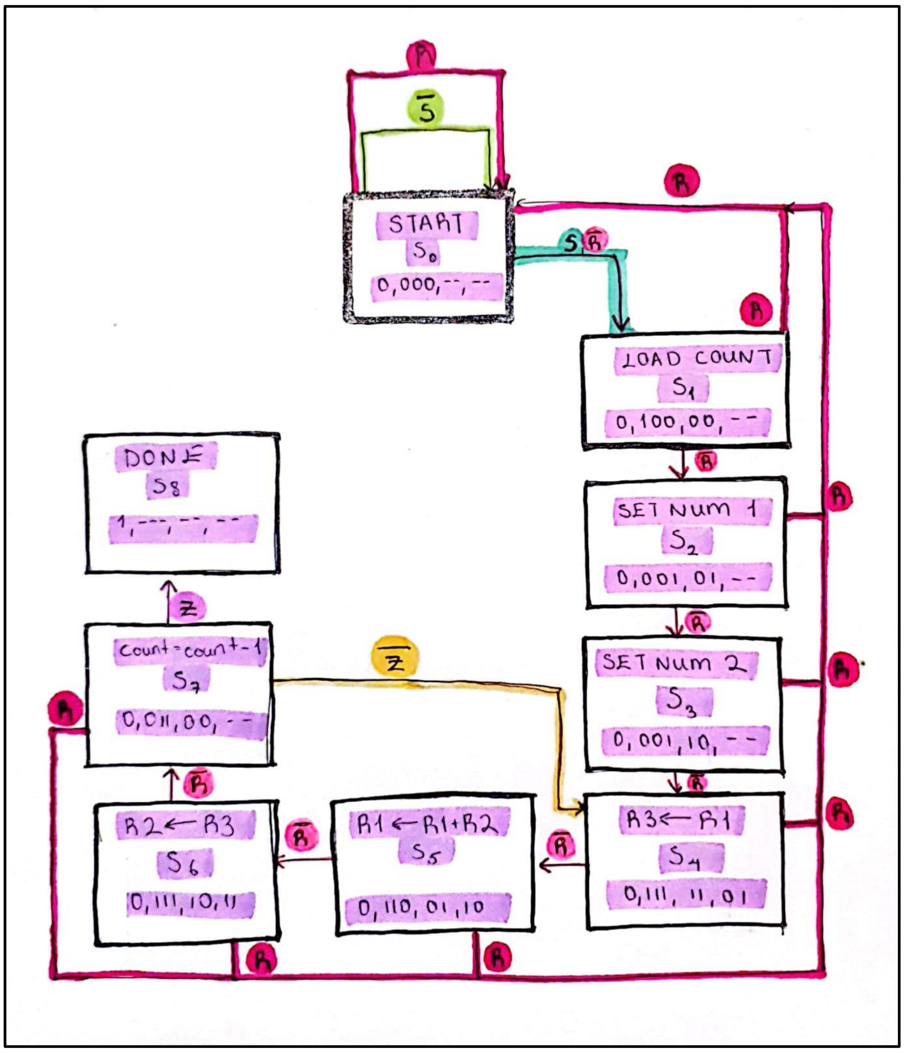
For S4: op_code = 111 which is copying register 01 to 11 or R1 to R3.

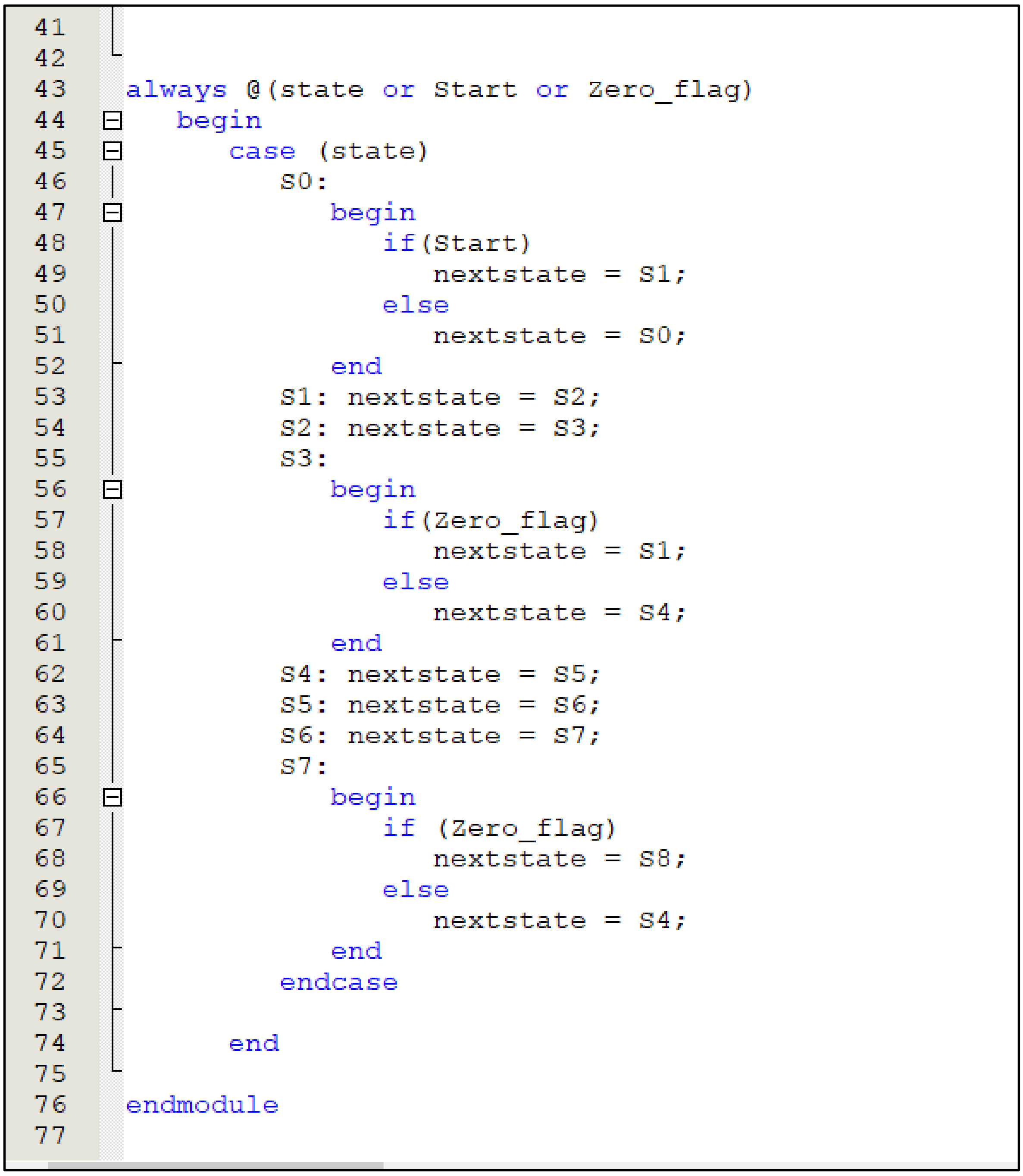
For S5: op_code = 110 which is adding register 10 to 01 or R2 to R1.

For S6: op_code = 111 which is copying register 11 to 10 or R3 to R2.

For S7: op_code = 011 which is decrementing register 00 or R0.

For S8: op_code = 000 which is no operation.





```
41
42
43  always @(state or Start or Zero_flag)
44     begin
45        case (state)
46           S0:
47              begin
48                 if(Start)
49                    nextstate = S1;
50                 else
51                    nextstate = S0;
52              end
53           S1: nextstate = S2;
54           S2: nextstate = S3;
55           S3:
56              begin
57                 if(Zero_flag)
58                    nextstate = S1;
59                 else
60                    nextstate = S4;
61              end
62           S4: nextstate = S5;
63           S5: nextstate = S6;
64           S6: nextstate = S7;
65           S7:
66              begin
67                 if (Zero_flag)
68                    nextstate = S8;
69                 else
70                    nextstate = S4;
71              end
72           endcase
73
74        end
75
76  endmodule
77
```

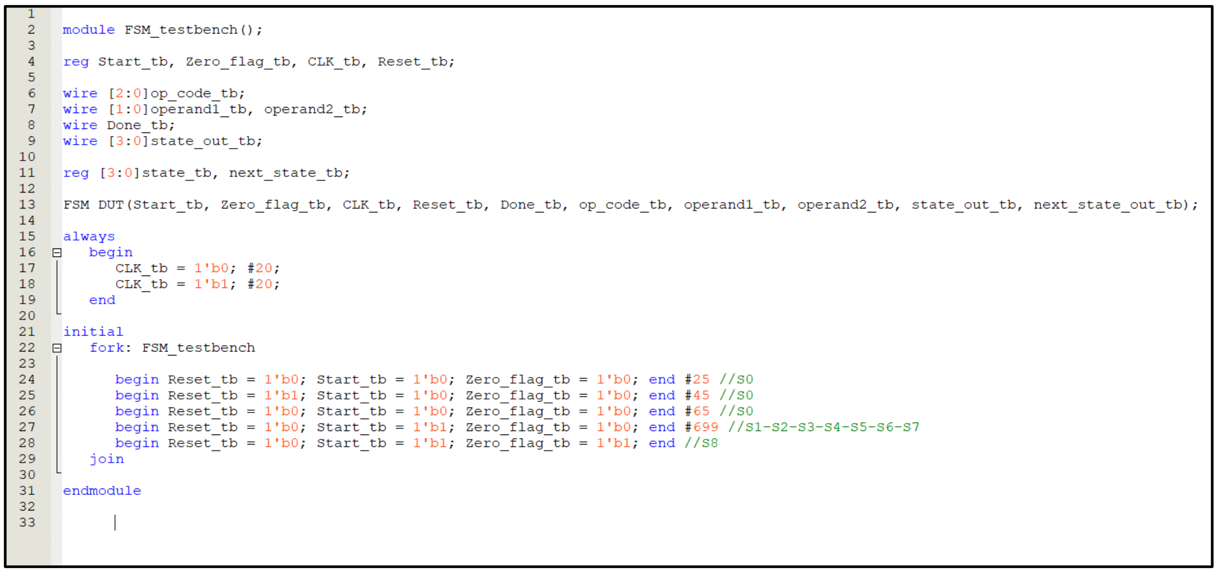
For lines 43-74 the next state will be assigned.

S0 will only move on to S1 if start is given.

if S3 is zero_flag it will loop back to S1 else it will continue onwards to S4.

if at S7 count is decremented to 0, nextstate will be S8 and the operation will be finished. Else it will loop back to S4.

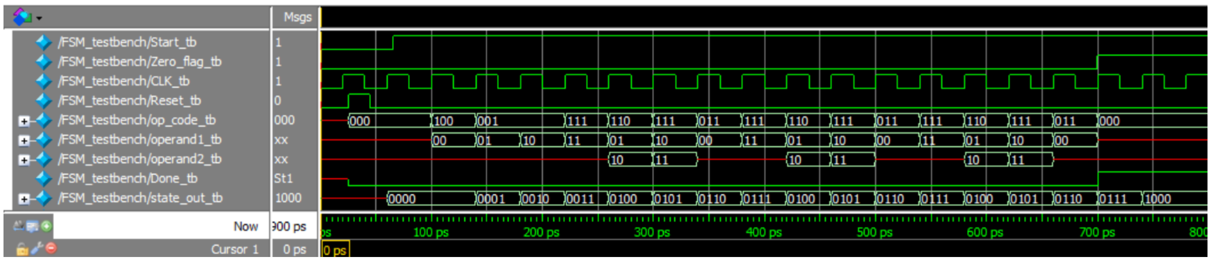
Testbench code:



```
module FSM_testbench();

reg Start_tb, Zero_flag_tb, CLK_tb, Reset_tb;

wire [2:0]op_code_tb;
wire [1:0]operand1_tb, operand2_tb;
wire Done_tb;
wire [3:0]state_out_tb;

reg [3:0]state_tb, next_state_tb;

FSM DUT(Start_tb, Zero_flag_tb, CLK_tb, Reset_tb, Done_tb, op_code_tb, operand1_tb, operand2_tb, state_out_tb, next_state_out_tb);

always
   begin
      CLK_tb = 1'b0; #20;
      CLK_tb = 1'b1; #20;
   end

initial
   fork: FSM_testbench

      begin Reset_tb = 1'b0; Start_tb = 1'b0; Zero_flag_tb = 1'b0; end #25 //S0
      begin Reset_tb = 1'b1; Start_tb = 1'b0; Zero_flag_tb = 1'b0; end #45 //S0
      begin Reset_tb = 1'b0; Start_tb = 1'b0; Zero_flag_tb = 1'b0; end #65 //S0
      begin Reset_tb = 1'b0; Start_tb = 1'b1; Zero_flag_tb = 1'b0; end #699 //S1-S2-S3-S4-S5-S6-S7
      begin Reset_tb = 1'b0; Start_tb = 1'b1; Zero_flag_tb = 1'b1; end //S8
   join

endmodule
```

CLK will change every 20 picoseconds.

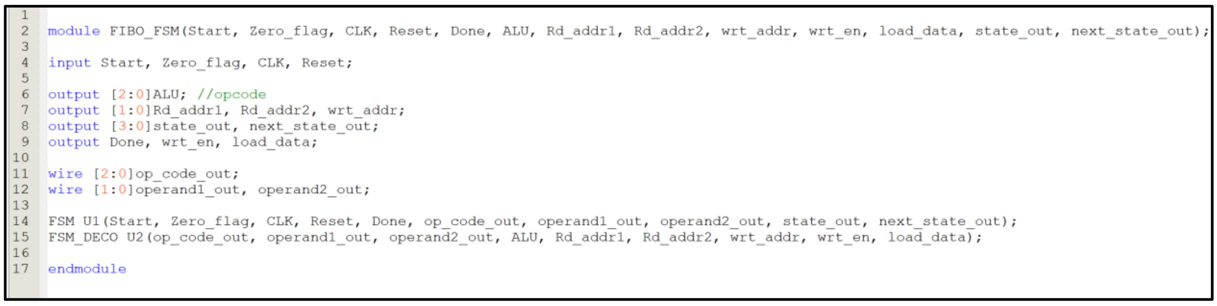
Reset, Start and Zero_flag will change at 25, 45, 65 and 699.



Start first becomes 1 at 65 ps. Since CLK is not posedge yet, the operation waits for the CLK posedge to change. After the CLK becomes posedge at 100 ps the operation starts. The opcode is changed at posedge CLK as it was coded. We can also follow the state_out and see that the states are executed correctly. In order: S0, S1, S2, S3, S4, S5, S6, S7, S4, S5, S6, S7, S4, S5, S6, S7, S8. The operation stops and goes to S8 when Zero_flag is 1.

5.3.3: Fibo_FSM:

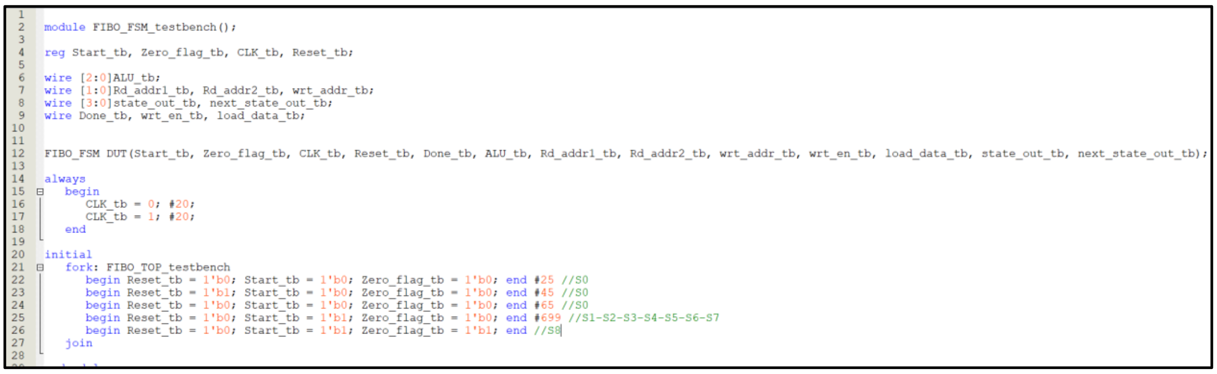
To create FSM_toplevel Fibo_FSM needs to be connected with FSM and FSM_DECO.



```

module FIBO_FSM(Start, Zero_flag, CLK, Reset, Done, ALU, Rd_addr1, Rd_addr2, wrt_addr, wrt_en, load_data, state_out, next_state_out);

input Start, Zero_flag, CLK, Reset;

output [2:0]ALU; //opcode
output [1:0]Rd_addr1, Rd_addr2, wrt_addr;
output [3:0]state_out, next_state_out;
output Done, wrt_en, load_data;

wire [2:0]op_code_out;
wire [1:0]operand1_out, operand2_out;

FSM U1(Start, Zero_flag, CLK, Reset, Done, op_code_out, operand1_out, operand2_out, state_out, next_state_out);
FSM_DECO U2(op_code_out, operand1_out, operand2_out, ALU, Rd_addr1, Rd_addr2, wrt_addr, wrt_en, load_data);

endmodule
```

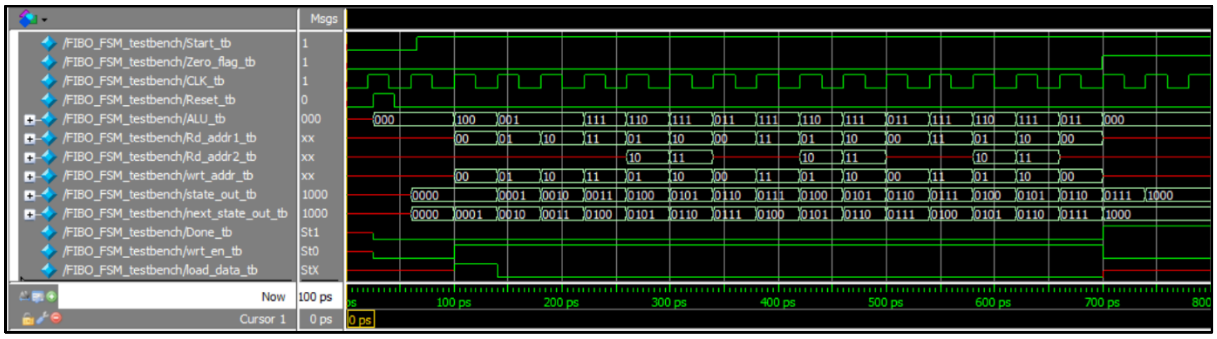
FSM and FSM_DECO is connected with the wires op_code_out, operand1_out, operand2_out.

Testbench code:



```

module FIBO_FSM_testbench();

reg Start_tb, Zero_flag_tb, CLK_tb, Reset_tb;

wire [2:0]ALU_tb;
wire [1:0]Rd_addr1_tb, Rd_addr2_tb, wrt_addr_tb;
wire [3:0]state_out_tb, next_state_out_tb;
wire Done_tb, wrt_en_tb, load_data_tb;


FIBO_FSM DUT(Start_tb, Zero_flag_tb, CLK_tb, Reset_tb, Done_tb, ALU_tb, Rd_addr1_tb, Rd_addr2_tb, wrt_addr_tb, wrt_en_tb, load_data_tb, state_out_tb, next_state_out_tb);

always
  begin
    CLK_tb = 0; #20;
    CLK_tb = 1; #20;
  end

initial
  fork: FIBO_TOP_testbench
    begin Reset_tb = 1'b0; Start_tb = 1'b0; Zero_flag_tb = 1'b0; end #25 //S0
    begin Reset_tb = 1'b1; Start_tb = 1'b0; Zero_flag_tb = 1'b0; end #45 //S0
    begin Reset_tb = 1'b0; Start_tb = 1'b0; Zero_flag_tb = 1'b0; end #65 //S0
    begin Reset_tb = 1'b0; Start_tb = 1'b1; Zero_flag_tb = 1'b0; end #699 //S1-S2-S3-S4-S5-S6-S7
    begin Reset_tb = 1'b0; Start_tb = 1'b1; Zero_flag_tb = 1'b1; end //S8
  join

```

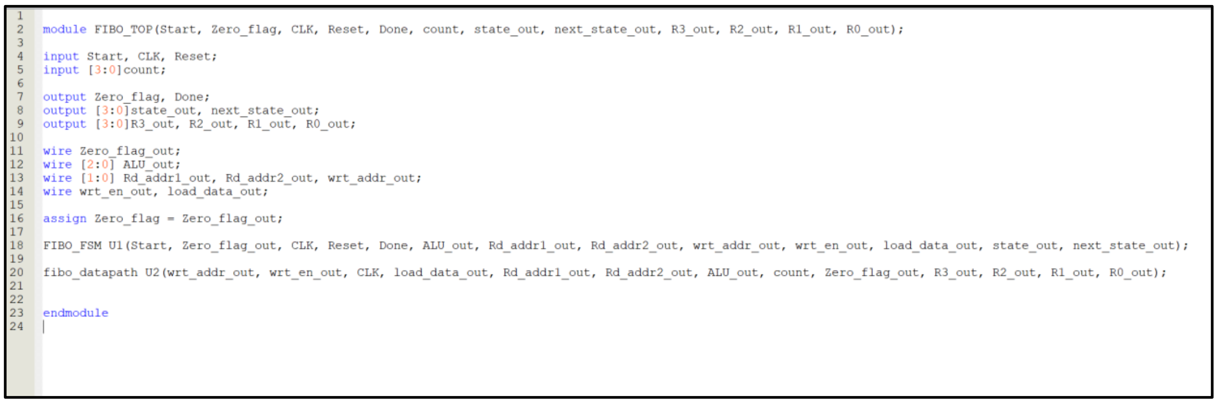
CLK changes every 20 pico seconds and the same inputs as FSM_testbench were given.



As it was the case before in FSM modelsim simulation, when zero_flag i given the state would be S8 and the operation would stop.

5.3.4: Fibo_toplevel:

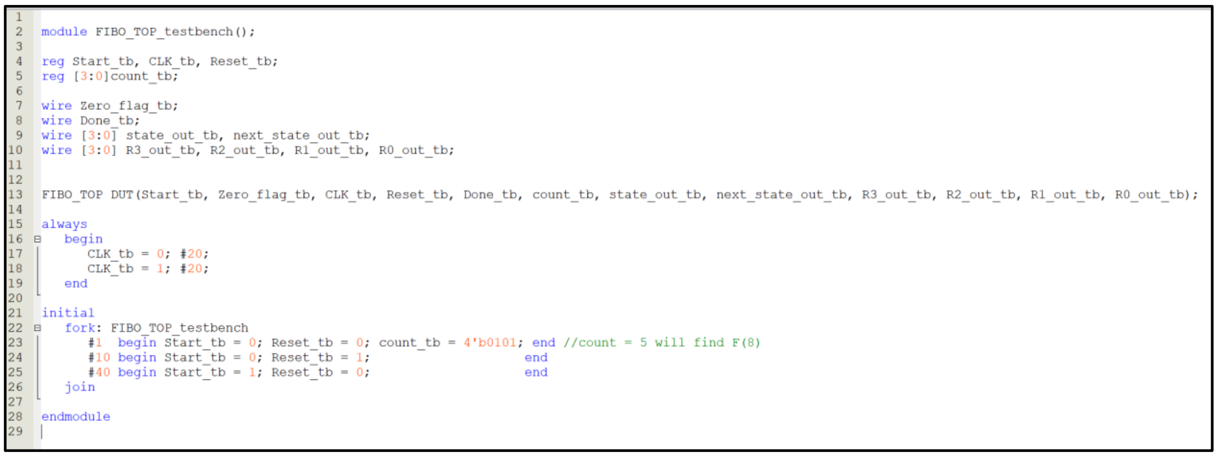
To create the FIBO_TOP we need to connect FIBO_FSM and fibo_datapath. Datapath was made last week.



```verilog

module FIBO_TOP(Start, Zero_flag, CLK, Reset, Done, count, state_out, next_state_out, R3_out, R2_out, R1_out, R0_out);

input Start, CLK, Reset;
input [3:0]count;

output Zero_flag, Done;
output [3:0]state_out, next_state_out;
output [3:0]R3_out, R2_out, R1_out, R0_out;

wire Zero_flag_out;
wire [2:0] ALU_out;
wire [1:0] Rd_addr1_out, Rd_addr2_out, wrt_addr_out;
wire wrt_en_out, load_data_out;

assign Zero_flag = Zero_flag_out;

FIBO_FSM U1(Start, Zero_flag_out, CLK, Reset, Done, ALU_out, Rd_addr1_out, Rd_addr2_out, wrt_addr_out, wrt_en_out, load_data_out, state_out, next_state_out);

fibo_datapath U2(wrt_addr_out, wrt_en_out, CLK, load_data_out, Rd_addr1_out, Rd_addr2_out, ALU_out, count, Zero_flag_out, R3_out, R2_out, R1_out, R0_out);


endmodule

```

FIBO_FSM and fibo_datapath is connected with the wires Zero_flag_out, ALU_out, Rd_addr1_out, Rd_addr2_out, wrt_addr_out, wrt_en_out and load_data_out.

Testbench code:



```verilog

module FIBO_TOP_testbench();

reg Start_tb, CLK_tb, Reset_tb;
reg [3:0]count_tb;

wire Zero_flag_tb;
wire Done_tb;
wire [3:0] state_out_tb, next_state_out_tb;
wire [3:0] R3_out_tb, R2_out_tb, R1_out_tb, R0_out_tb;


FIBO_TOP DUT(Start_tb, Zero_flag_tb, CLK_tb, Reset_tb, Done_tb, count_tb, state_out_tb, next_state_out_tb, R3_out_tb, R2_out_tb, R1_out_tb, R0_out_tb);

always
   begin
      CLK_tb = 0; #20;
      CLK_tb = 1; #20;
   end

initial
   fork: FIBO_TOP_testbench
      #1  begin Start_tb = 0; Reset_tb = 0; count_tb = 4'b0101; end //count = 5 will find F(8)
      #10 begin Start_tb = 0; Reset_tb = 1;                     end
      #40 begin Start_tb = 1; Reset_tb = 0;                     end
   join

endmodule

```

CLK changes every 20 pico seconds. Count was chosen to be 5.

From the formula F(n + 2) = F(n) + F(n + 1):

F(7) = F(5) + F(6);

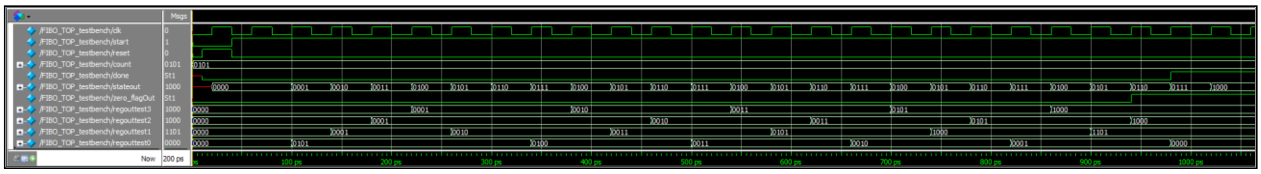
F(6) = F(4) + F(5);

F(5) = 5 (as calculated above)

F(4) = 3 (as calculated above)

F(6) = 5 + 3 = 8.

F(7) = 8 + 5 = 13 will be found.

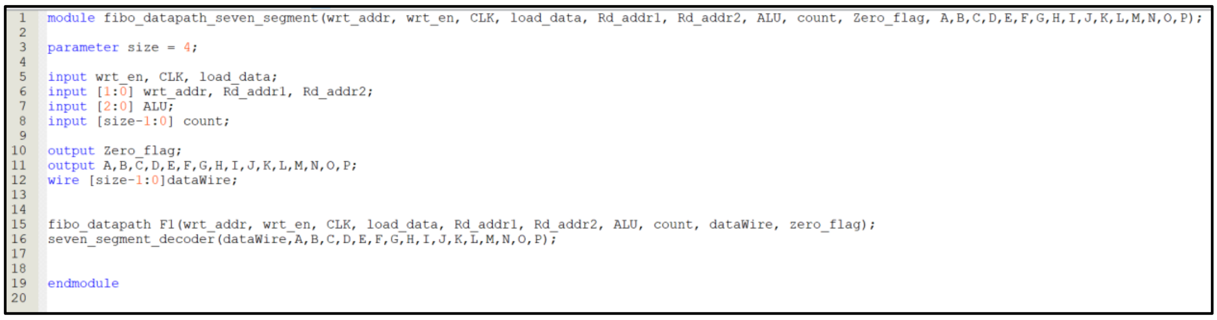


	Msgs
/FIBO_TOP_testbench/clk	0
/FIBO_TOP_testbench/start	1
/FIBO_TOP_testbench/reset	0
/FIBO_TOP_testbench/count	0101
/FIBO_TOP_testbench/done	St1
/FIBO_TOP_testbench/stateout	1000
/FIBO_TOP_testbench/zero_flagOut	St1
/FIBO_TOP_testbench/regouttest3	1000
/FIBO_TOP_testbench/regouttest2	1000
/FIBO_TOP_testbench/regouttest1	1101
/FIBO_TOP_testbench/regouttest0	0000
Now	200 ps

As it can be viewed R1 becomes 1101 at the end. 1101 is 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13. R1 gives the correct output.

5.3.5: Seven_segment_display:

Seven Segment display has been written exactly like the ones that were written in previous labs. The outputs will be displayed with FPGA in lab hours.



```verilog
module fibo_datapath_seven_segment(wrt_addr, wrt_en, CLK, load_data, Rd_addr1, Rd_addr2, ALU, count, Zero_flag, A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P);

parameter size = 4;

input wrt_en, CLK, load_data;
input [1:0] wrt_addr, Rd_addr1, Rd_addr2;
input [2:0] ALU;
input [size-1:0] count;

output Zero_flag;
output A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P;
wire [size-1:0]dataWire;


fibo_datapath F1(wrt_addr, wrt_en, CLK, load_data, Rd_addr1, Rd_addr2, ALU, count, dataWire, zero_flag);
seven_segment_decoder(dataWire,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P);


endmodule

```