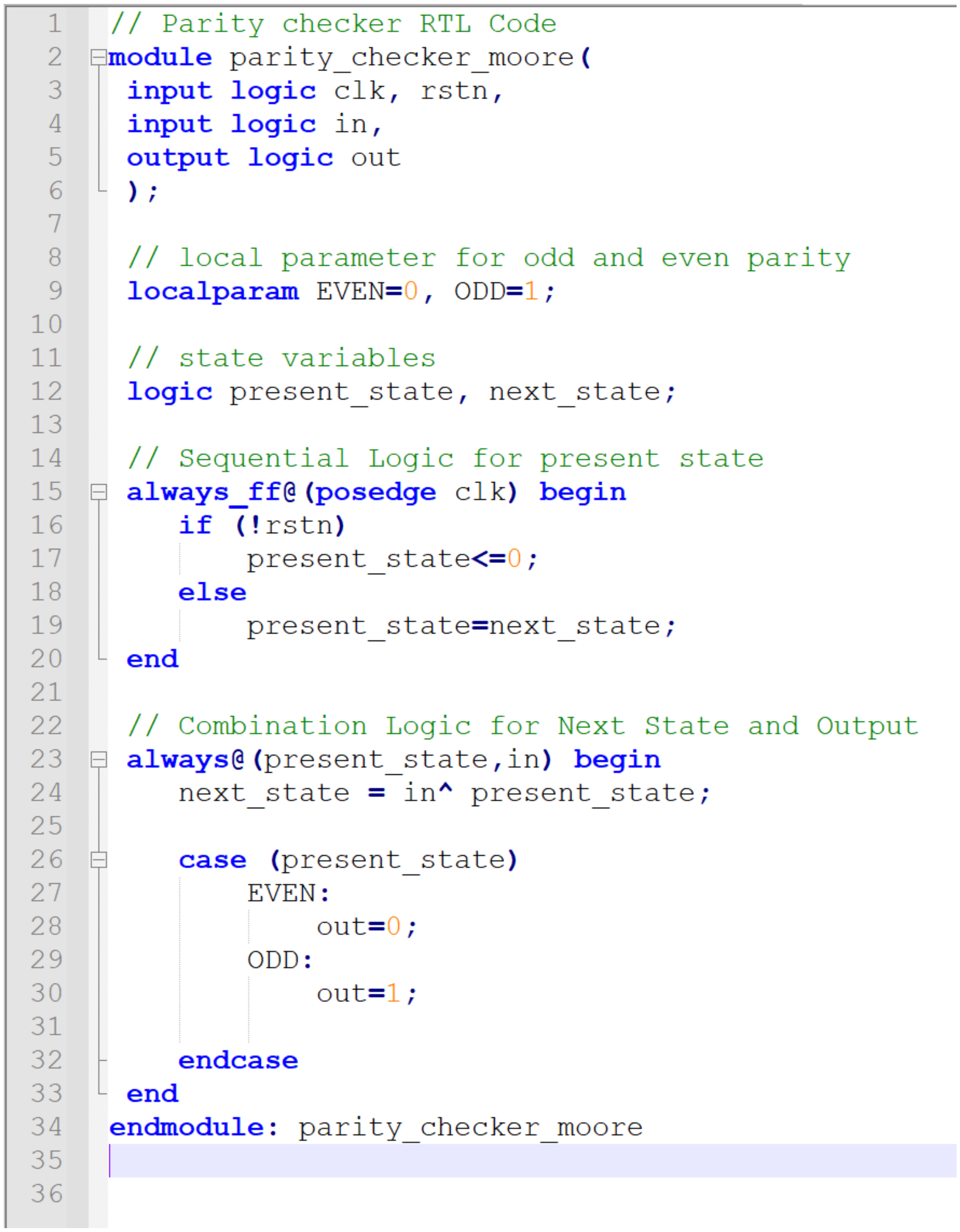
Giao Tran

ECE 111 HW 7 report

<u>Homework 7a: Moore Mealy SystemVerilog FSM code for parity checker</u>

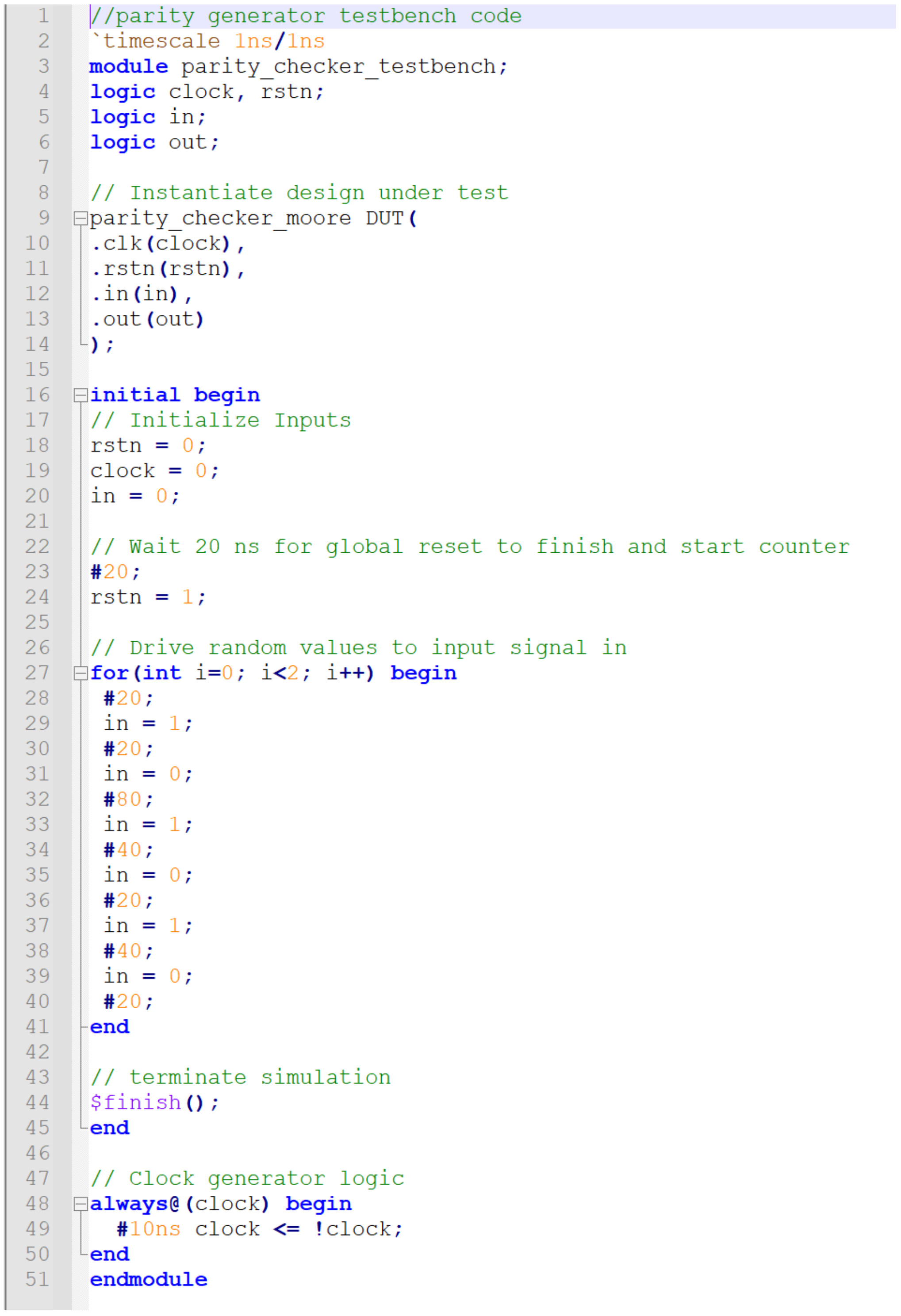
a. **Moore parity checker FSM code**

   1. SystemVerilog FSM code



```
// Parity checker RTL Code
module parity_checker_moore(
 input logic clk, rstn,
 input logic in,
 output logic out
 );

 // local parameter for odd and even parity
 localparam EVEN=0, ODD=1;

 // state variables
 logic present_state, next_state;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
    if (!rstn)
        present_state<=0;
    else
        present_state=next_state;
 end

 // Combination Logic for Next State and Output
 always@(present_state,in) begin
    next_state = in^ present_state;

    case (present_state)
        EVEN:
            out=0;
        ODD:
            out=1;

    endcase
 end
endmodule: parity_checker_moore
```

2. SystemVerilog FSM testbench code (Moore)



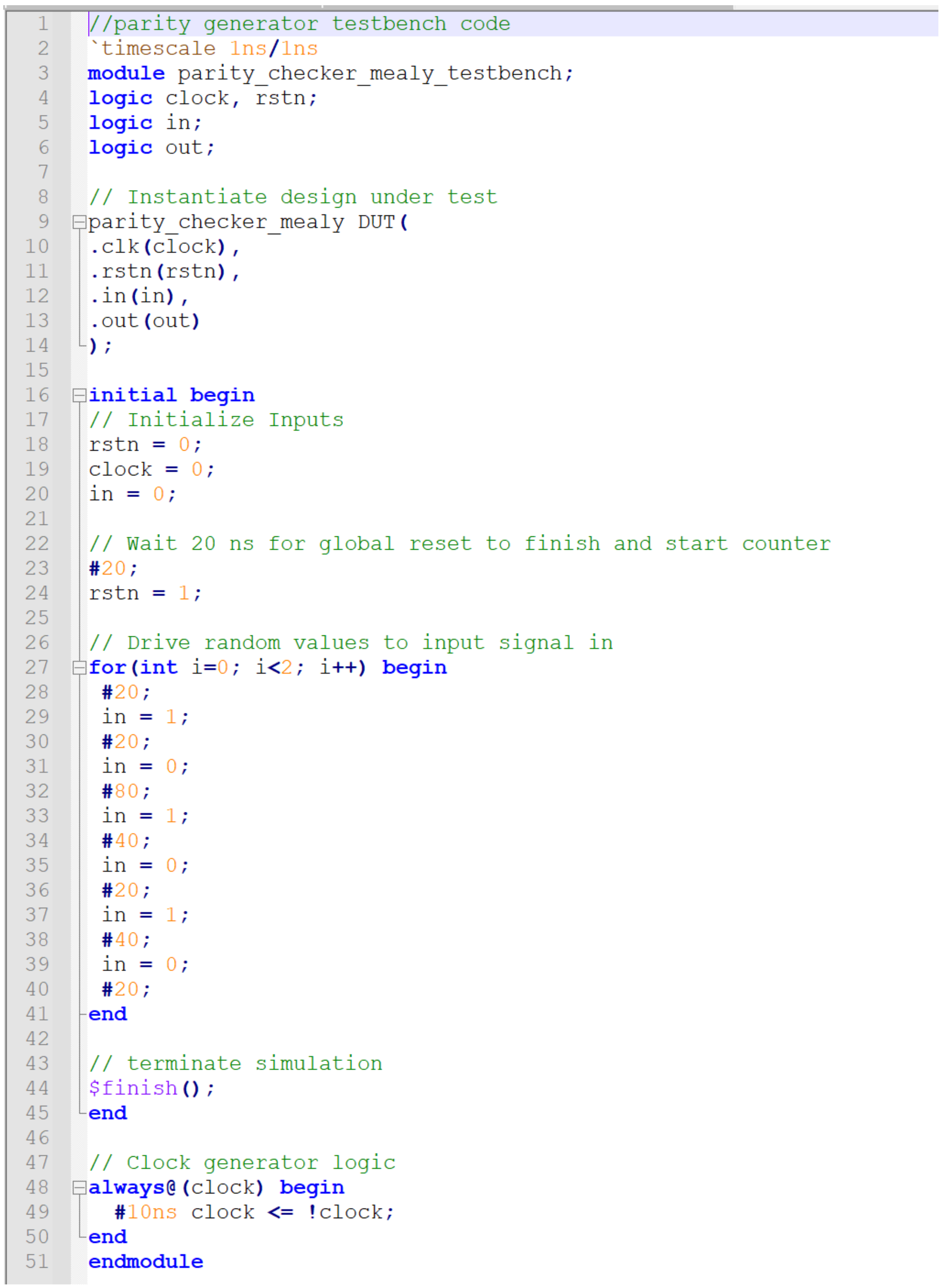
```
//parity generator testbench code
`timescale 1ns/1ns
module parity_checker_testbench;
logic clock, rstn;
logic in;
logic out;

// Instantiate design under test
parity_checker_moore DUT(
.clk(clock),
.rstn(rstn),
.in(in),
.out(out)
);

initial begin
// Initialize Inputs
rstn = 0;
clock = 0;
in = 0;

// Wait 20 ns for global reset to finish and start counter
#20;
rstn = 1;

// Drive random values to input signal in
for(int i=0; i<2; i++) begin
 #20;
 in = 1;
 #20;
 in = 0;
 #80;
 in = 1;
 #40;
 in = 0;
 #20;
 in = 1;
 #40;
 in = 0;
 #20;
end

// terminate simulation
$finish();
end

// Clock generator logic
always@(clock) begin
  #10ns clock <= !clock;
end
endmodule
```

b. Mealy parity checker FSM code

1. SystemVerilog FSM code

```
// Parity checker RTL Code
module parity_checker_mealy(
 input logic clk, rstn,
 input logic in,
 output logic out
 );

 // state variables
 enum logic[1:0] {EVEN=2'b00, ODD=2'b01} present_state, next_state;

 // local variable to store output in always_comb block
 logic out_t;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
    if (!rstn) begin
        present_state<= EVEN;
        out<=0;
    end

    else begin
        present_state<=next_state;
        out<=out_t;
    end
 end

 always_comb begin
    case (present_state)
        EVEN: begin
            if (in==1)begin
                next_state=ODD;
                out_t=1;
            end
            else begin
                next_state=EVEN;
                out_t=0;
            end
        end

        ODD: begin
        if (in==1)begin
                next_state=EVEN;
                out_t=0;
            end
        else begin
                next_state=ODD;
                out_t=1;
            end
        end

        default: begin
            out_t=0;
            next_state=EVEN;
        end
    endcase
  end
endmodule: parity_checker_mealy
```

2. SystemVerilog FSM testbench code (Mealy)

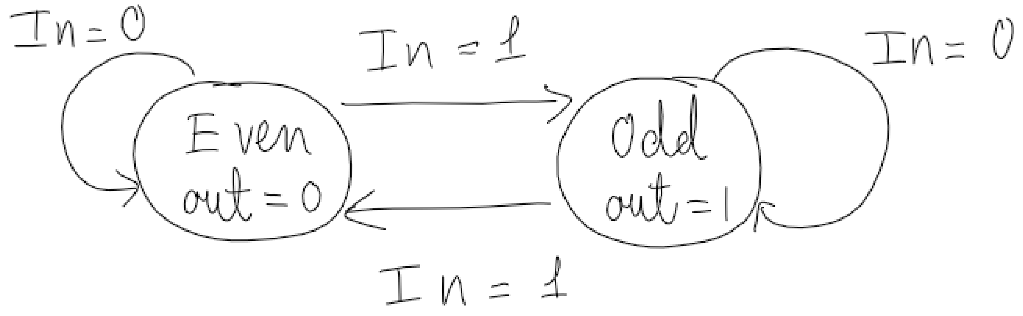


```
//parity generator testbench code
`timescale 1ns/1ns
module parity_checker_mealy_testbench;
logic clock, rstn;
logic in;
logic out;

// Instantiate design under test
parity_checker_mealy DUT(
.clk(clock),
.rstn(rstn),
.in(in),
.out(out)
);

initial begin
// Initialize Inputs
rstn = 0;
clock = 0;
in = 0;

// Wait 20 ns for global reset to finish and start counter
#20;
rstn = 1;

// Drive random values to input signal in
for(int i=0; i<2; i++) begin
 #20;
 in = 1;
 #20;
 in = 0;
 #80;
 in = 1;
 #40;
 in = 0;
 #20;
 in = 1;
 #40;
 in = 0;
 #20;
end

// terminate simulation
$finish();
end

// Clock generator logic
always@(clock) begin
  #10ns clock <= !clock;
end
endmodule
```

c. **Moore parity checker FSM state transition diagram and table**

1. State transition diagram

Moore



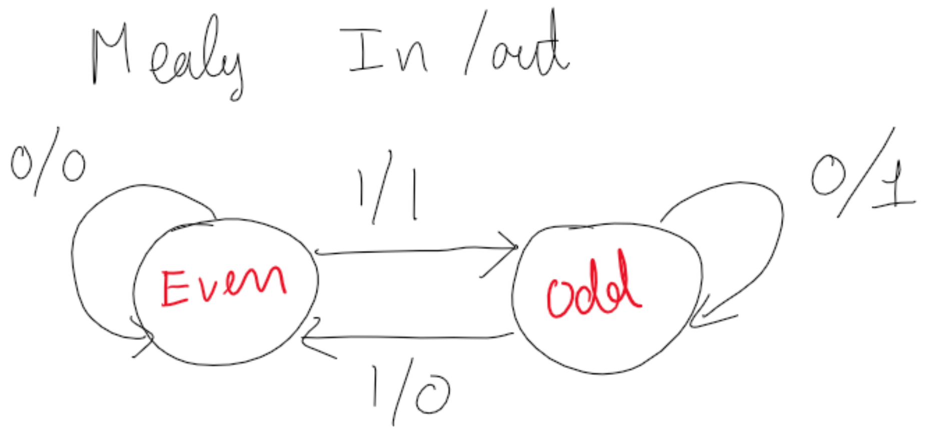
2. State transition table

Present state		input	next state		Out
Even	0	0	even	0	0
Even	0	1	odd	1	0
Odd	1	0	odd	1	1
Odd	1	1	even	0	1

out = present_state
next_state = In ^ present_state

d. Mealy parity checker FSM state transition diagram and table

1. State transition diagram

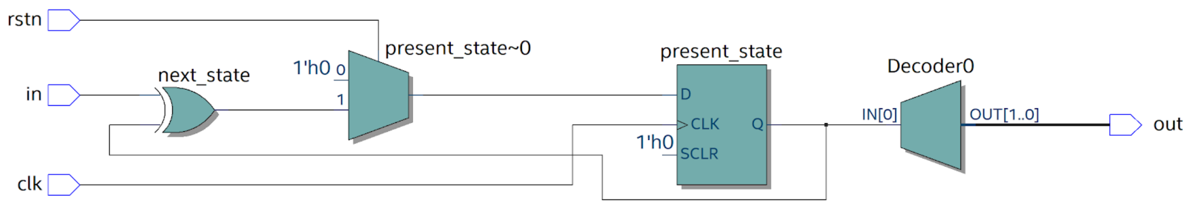


2. State transition table

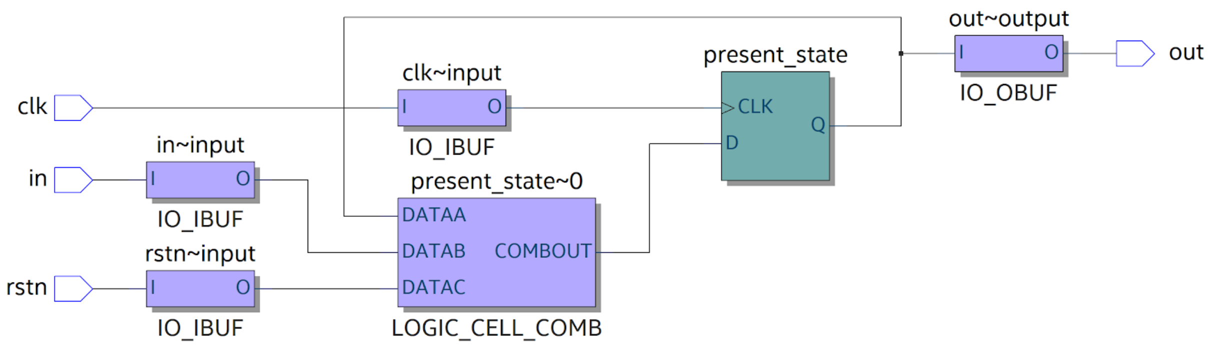
Present state	In	Next state	Out
Even	0	Even	0
Even	1	odd	1
odd	0	odd	1
odd	1	Even	0

e. Moore schematic and usage table

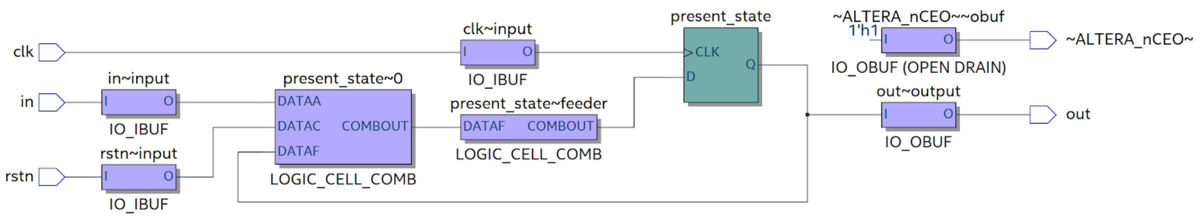
1. RTL schematic



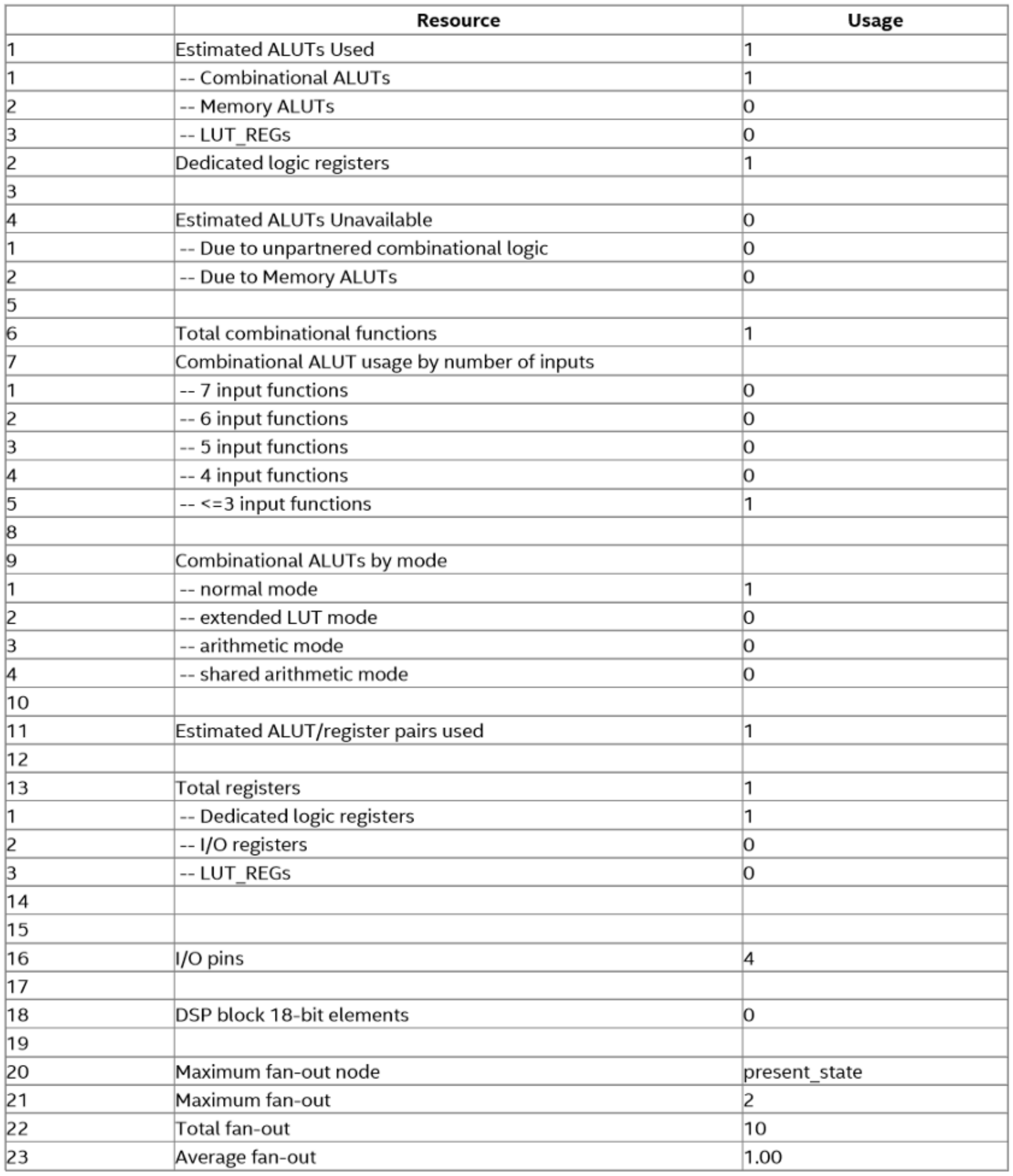
2. Post mapping schematic



3. Post fitting schematic



4. Resource usage table (Moore)

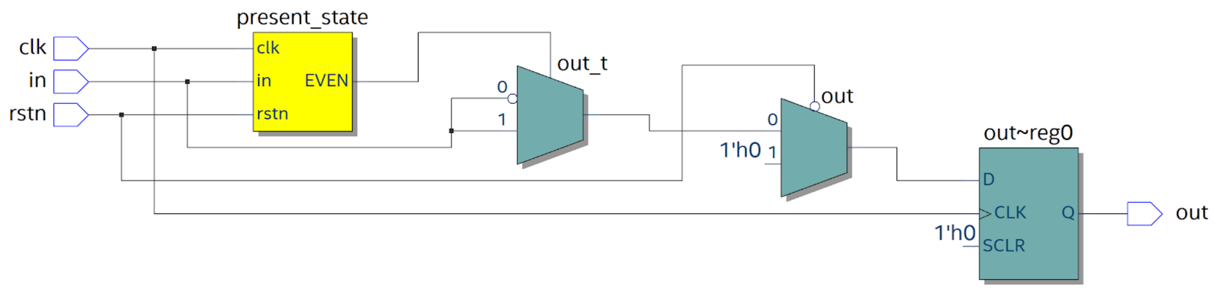


	Resource	Usage
1	Estimated ALUTs Used	1
1	-- Combinational ALUTs	1
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	1
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	1
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	0
5	-- <=3 input functions	1
8		
9	Combinational ALUTs by mode	
1	-- normal mode	1
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	1
12		
13	Total registers	1
1	-- Dedicated logic registers	1
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	4
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	present_state
21	Maximum fan-out	2
22	Total fan-out	10
23	Average fan-out	1.00

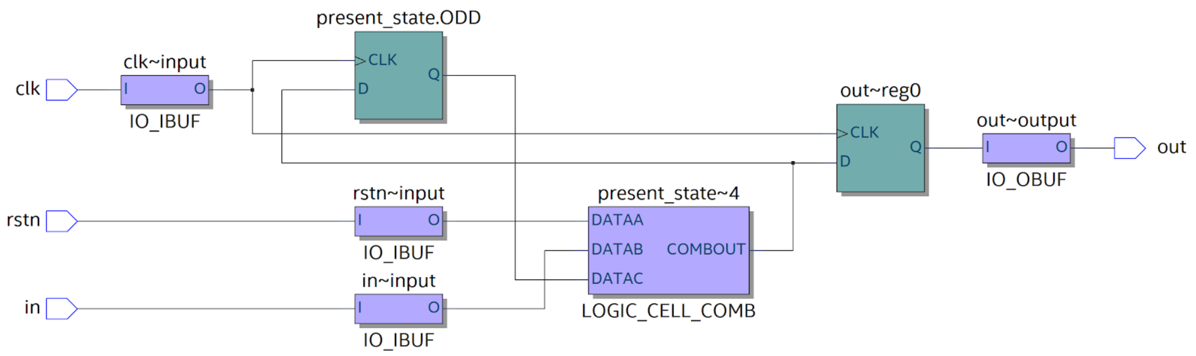
Number of ALUT: 1 (4 I/O pins)
Number of Functions: 1 (3 input functions)

f. Mealy schematic and usage table

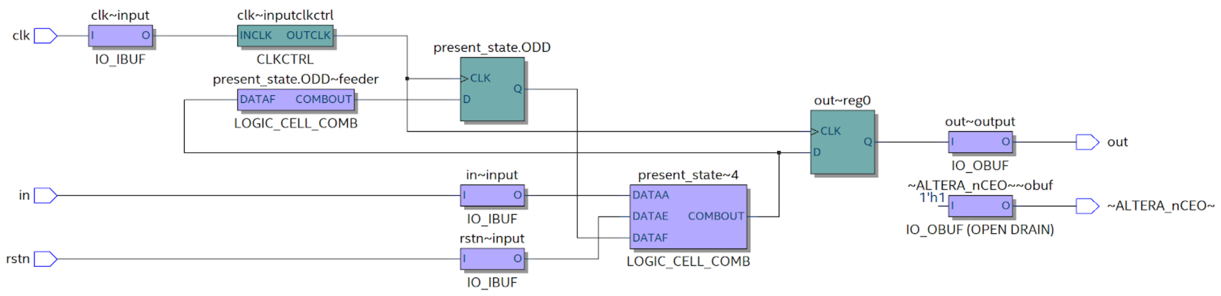
1. RTL schematic



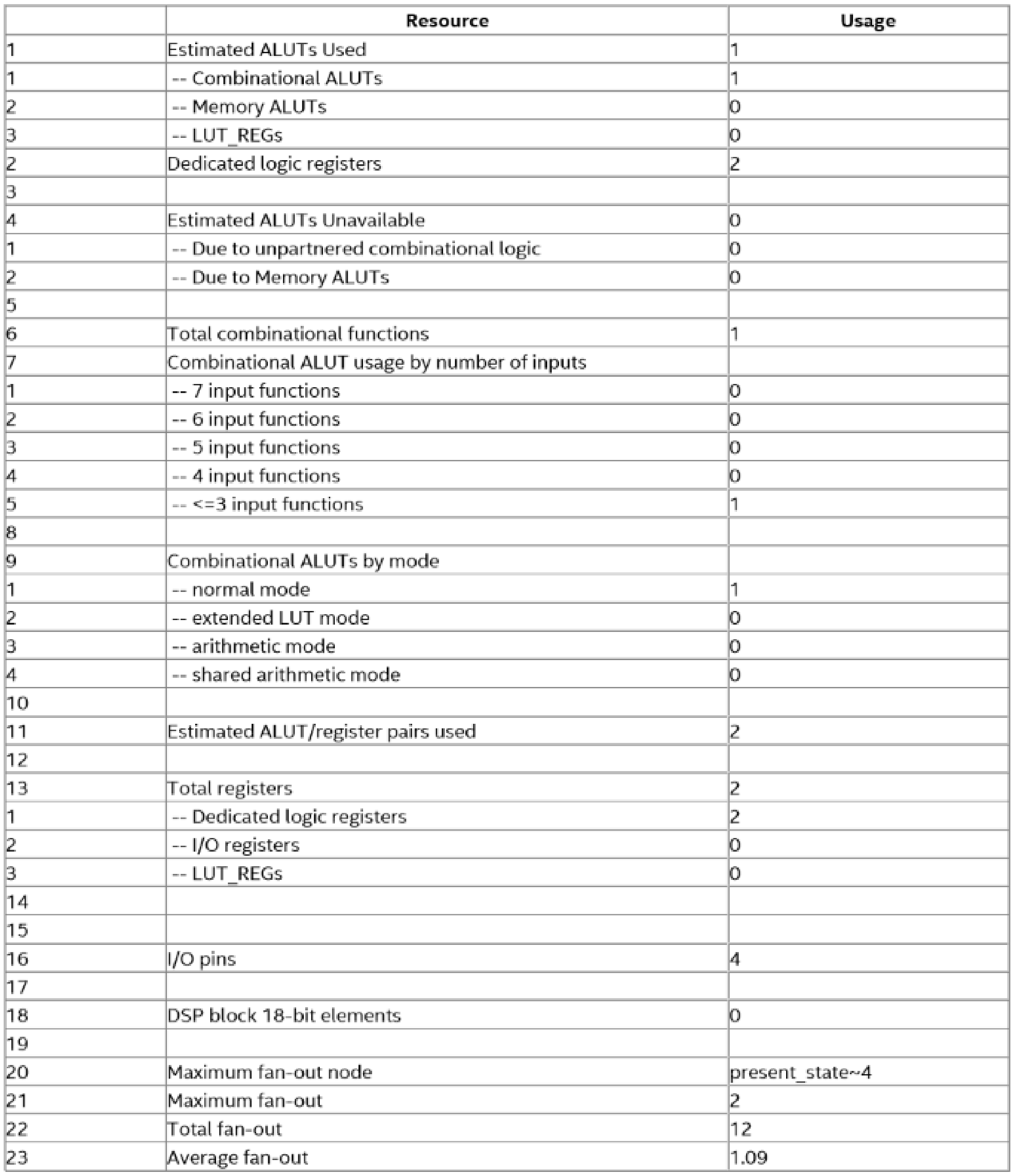
2. Post mapping schematic



3. Post fitting schematic



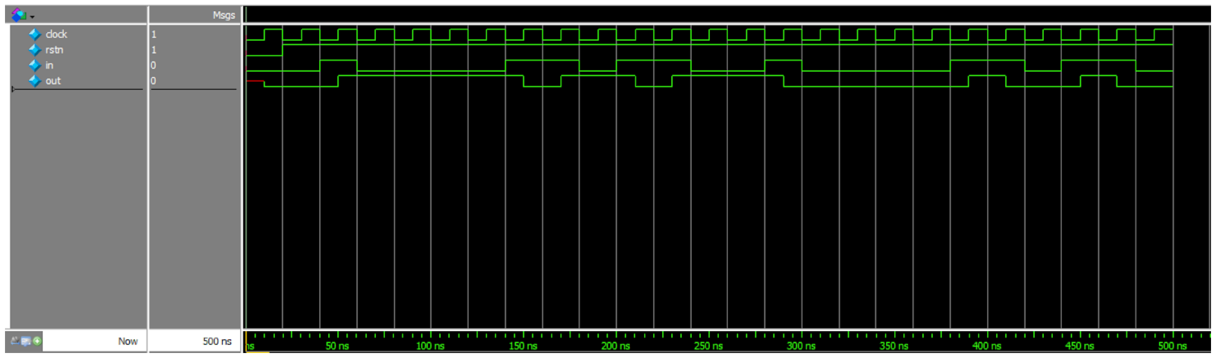
4. Resource usage table (Mealy)



	Resource	Usage
1	Estimated ALUTs Used	1
1	-- Combinational ALUTs	1
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	2
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	1
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	0
5	-- <=3 input functions	1
8		
9	Combinational ALUTs by mode	
1	-- normal mode	1
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	2
12		
13	Total registers	2
1	-- Dedicated logic registers	2
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	4
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	present_state~4
21	Maximum fan-out	2
22	Total fan-out	12
23	Average fan-out	1.09

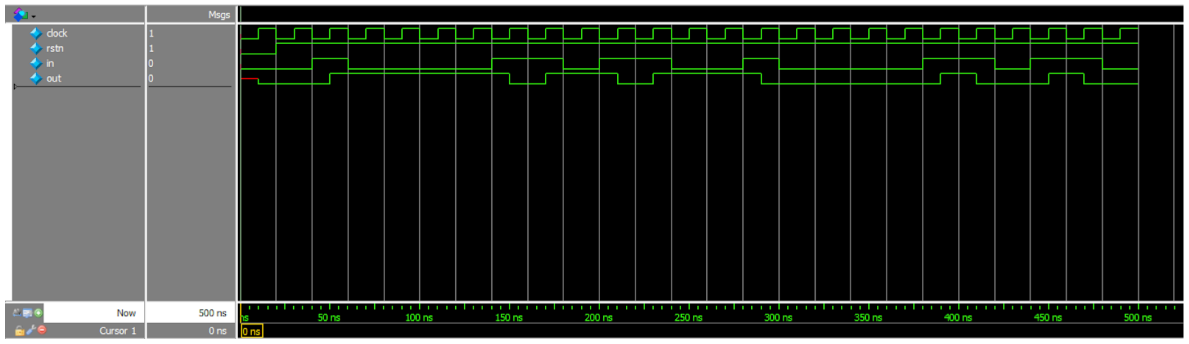
Number of ALUT: 1 (4 I/O pins)
Number of Functions: 1 (3 input functions)

g. Moore simulation waveform and explanation



If the number of 1 bits is even, the output “out” will give out 1 bit, but if the number if 1 bits is odd, then the output would be 0; the parity checker only has 1 input per clock cycle. Notice that the output wave is synchronous, this means the output responds to the changes in the input only at positive clock signal (in this case).

h. Mealy simulation waveform and explanation



The output waveform is the same for both Mealy and Moore machines because they describe the same process. However, to achieve this the Mealy machine must register its output so that the output of Mealy can be synchronous, otherwise the output as soon as the input changes.

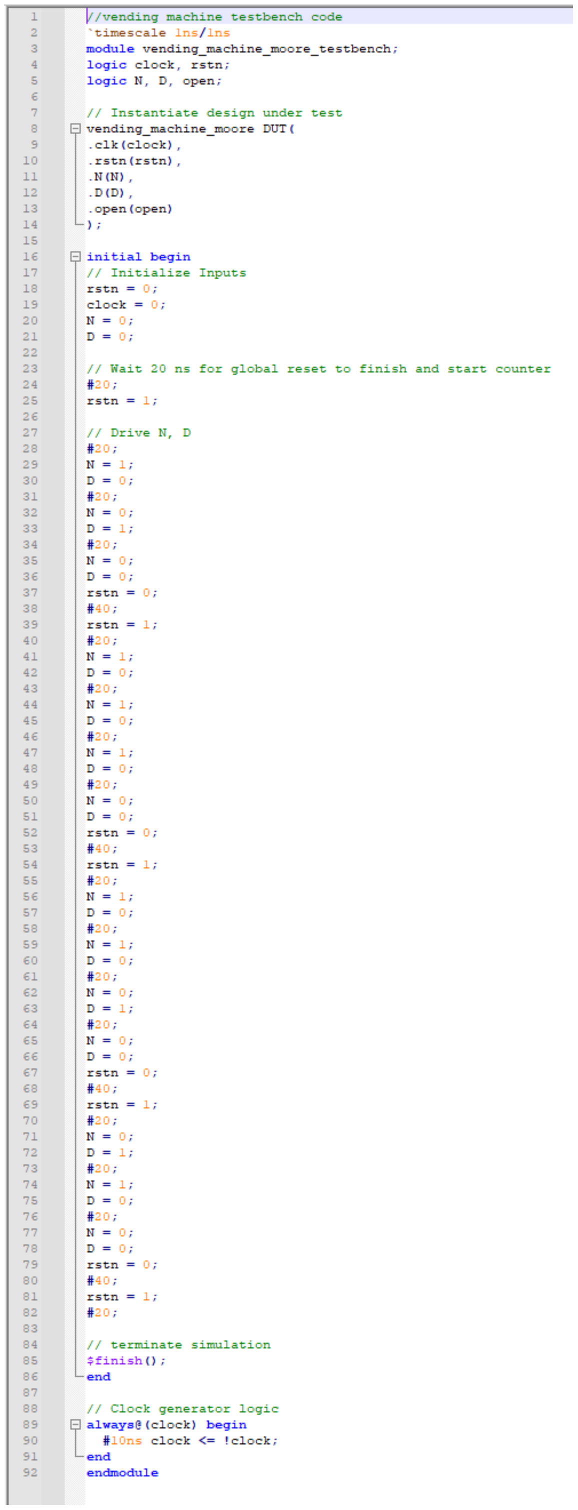
Homework 7b: Moore Mealy SystemVerilog FSM code for Vending Machine

a. Moore parity checker FSM code

1. SystemVerilog FSM code

```
// Vending Machine RTL Code
module vending_machine_moore(
 input logic clk, rstn,
 input logic N, D,
 output logic open);

 // state variables and state encoding parameters
 parameter[3:0] CENTS_0=0, CENTS_5=1, CENTS_10=2, CENTS_15=3;
 logic[3:0] present_state, next_state;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if (!rstn) begin
    present_state <= 0;
    present_state[CENTS_0] <= 1'b1;
  end
  else
    present_state <= next_state;
 end

 // Combination Logic for Next State and Output
 always_comb begin
  // default values to avoid latches for next_state since partial bits of next_state
  are assigned inside case if statements
  next_state = 4'b0;
  open = 1'b0;

  // use of priority before case, since case(1'b1) has multiple possible case item
  matching, priority enforces priority encoding of case items in order it is specified
  priority case(1'b1)
   present_state[CENTS_0]: begin
      open = 0;
      if(N==1) next_state[CENTS_5] = 1'b1;
      else if(D==1) next_state[CENTS_10] = 1'b1;
      else next_state[CENTS_0] = 1'b1;
   end
   present_state[CENTS_5]: begin
      open = 0;
      if (N==1) next_state[CENTS_10]=1'b1;
      else if (D==1) next_state[CENTS_15]=1'b1;
      else next_state[CENTS_5] = 1'b1;

   end
   present_state[CENTS_10]: begin
        open=0;
        if (N==1) next_state[CENTS_15]=1'b1;
        else if (D==1) next_state[CENTS_15]=1'b1;
        else next_state[CENTS_10] = 1'b1;
   end
   present_state[CENTS_15]: begin
        open =1;
        next_state[CENTS_0]=1'b1;
    end

   default: begin
   next_state[CENTS_0]=1'b1;
   end
  endcase
 end
endmodule: vending_machine_moore
```

2. SystemVerilog FSM testbench code (Moore)



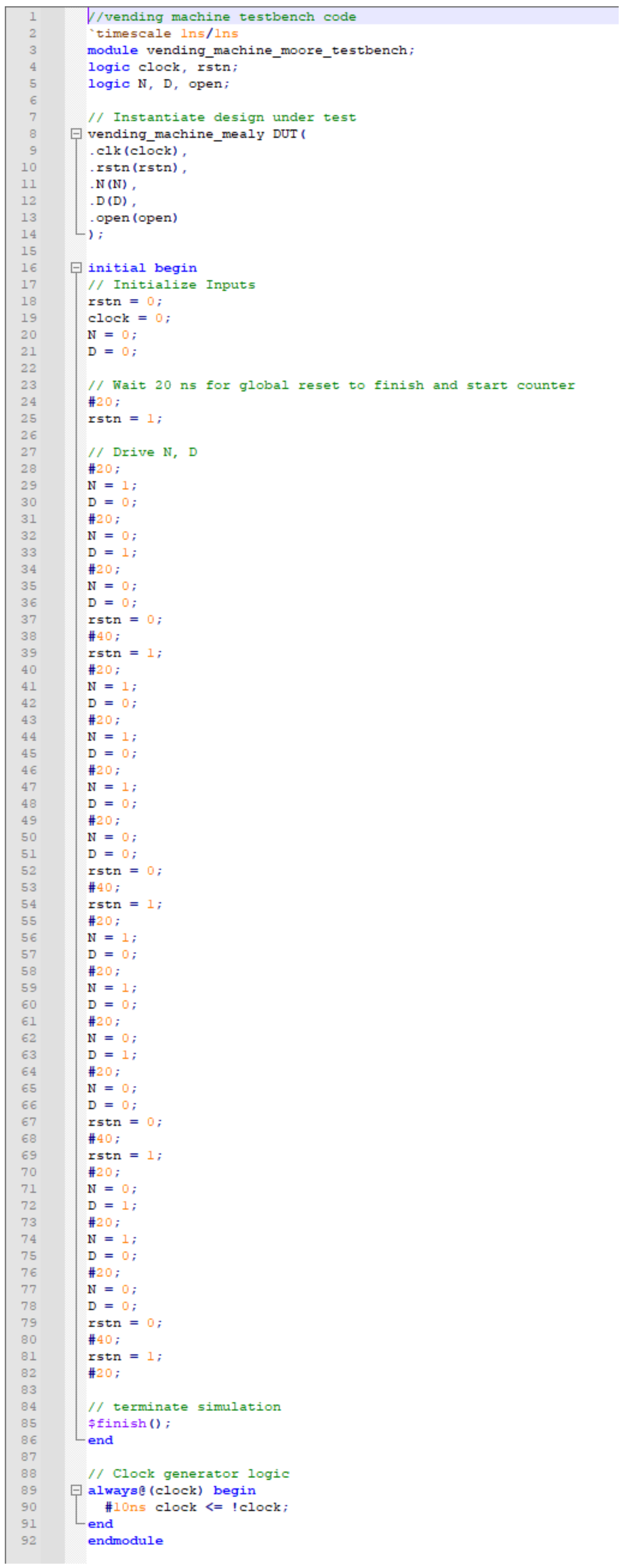
```
//vending machine testbench code
`timescale 1ns/1ns
module vending_machine_moore_testbench;
logic clock, rstn;
logic N, D, open;

// Instantiate design under test
vending_machine_moore DUT(
.clk(clock),
.rstn(rstn),
.N(N),
.D(D),
.open(open)
);

initial begin
// Initialize Inputs
rstn = 0;
clock = 0;
N = 0;
D = 0;

// Wait 20 ns for global reset to finish and start counter
#20;
rstn = 1;

// Drive N, D
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 1;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;
N = 1;
D = 0;
#20;
N = 1;
D = 0;
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;
N = 1;
D = 0;
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 1;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;
N = 0;
D = 1;
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;

// terminate simulation
$finish();
end

// Clock generator logic
always@(clock) begin
  #10ns clock <= !clock;
end
endmodule
```

b. Mealy parity checker FSM code

1. SystemVerilog FSM code

```
// Vending Machine RTL Code
module vending_machine_mealy(
 input logic clk, rstn,
 input logic N, D,
 output logic open);

 // state encoding and state variables
 parameter[3:0] CENTS_0=0, CENTS_5=1, CENTS_10=2, CENTS_15=3;
 logic[3:0] present_state, next_state;


 // Note : output open is not registered in this example for students to compare moore
 and mealy machine waveform and see what is the different between mealy and moore
 // remember we learnt in class that mealy reacts immediately to change in input !!

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if(!rstn) begin
    present_state <= 0;
    present_state[CENTS_0] <= 1'b1;
  end

  else begin
    present_state <= next_state;
  end
 end

 // Note : output open is not registered in this example for students to compare moore
 and mealy machine waveform and see what is the different between mealy and moore
 // remember we learnt in class that mealy reacts immediately to change in input !!
 // Combination Logic for Next State and Output
 always_comb begin
    next_state = 4'b0;

priority case(1'b1)
    present_state[CENTS_0]:
        begin
            if(N==1) begin
            next_state[CENTS_5] = 1'b1;
            open=0;
            end

            else if(D==1) begin
            next_state[CENTS_10] = 1'b1;
            open=0;
            end

            else begin
            next_state[CENTS_0] = 1'b1;
            open=0;
            end
        end

    present_state[CENTS_5]:
        begin
            if (N==1) begin
            next_state[CENTS_10]=1'b1;
            open=0;
            end

            else if (D==1) begin
            next_state[CENTS_15]=1'b1;
            open=1;
            end

            else begin
            next_state[CENTS_5] = 1'b1;
            open=0;
            end
        end

    present_state[CENTS_10]:
        begin
            if (N==1) begin
            next_state[CENTS_15]=1'b1;
            open=1;
            end

            else if (D==1) begin
            next_state[CENTS_15]=1'b1;
            open=1;
            end

            else begin
            next_state[CENTS_10] = 1'b1;
            open=0;
            end

        end

    present_state[CENTS_15]:
        begin
            open=1;
            next_state[CENTS_0]=1'b1;
        end

    default: begin
        open=0;
        next_state[CENTS_0]=1'b1;
        end

endcase
 end
endmodule: vending_machine_mealy
```

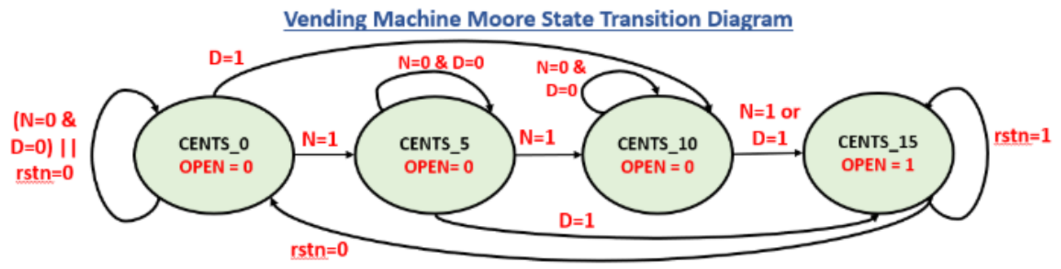
2. SystemVerilog FSM testbench code (Moore)



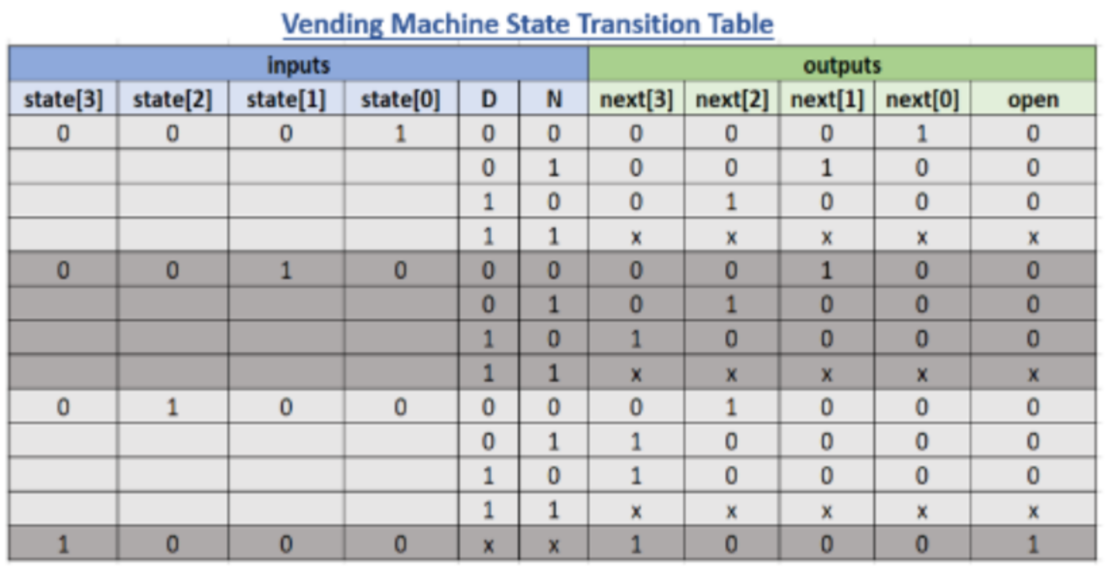
```
//vending machine testbench code
`timescale 1ns/1ns
module vending_machine_moore_testbench;
logic clock, rstn;
logic N, D, open;

// Instantiate design under test
vending_machine_mealy DUT(
.clk(clock),
.rstn(rstn),
.N(N),
.D(D),
.open(open)
);

initial begin
// Initialize Inputs
rstn = 0;
clock = 0;
N = 0;
D = 0;

// Wait 20 ns for global reset to finish and start counter
#20;
rstn = 1;

// Drive N, D
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 1;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;
N = 1;
D = 0;
#20;
N = 1;
D = 0;
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;
N = 1;
D = 0;
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 1;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;
N = 0;
D = 1;
#20;
N = 1;
D = 0;
#20;
N = 0;
D = 0;
rstn = 0;
#40;
rstn = 1;
#20;

// terminate simulation
$finish();
end

// Clock generator logic
always@(clock) begin
  #10ns clock <= !clock;
end
endmodule
```

c. Moore Vending MachineFSM state transition diagram and table

1. FSM state transition diagram



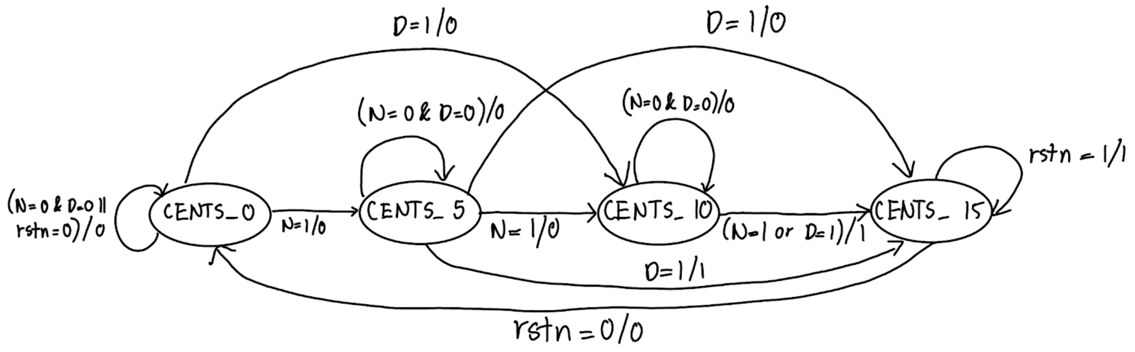
2. FSM state transition table


Vending Machine State Transition Table

inputs						outputs				
state[3]	state[2]	state[1]	state[0]	D	N	next[3]	next[2]	next[1]	next[0]	open
0	0	0	1	0	0	0	0	0	1	0
				0	1	0	0	1	0	0
				1	0	0	1	0	0	0
				1	1	x	x	x	x	x
0	0	1	0	0	0	0	0	1	0	0
				0	1	0	1	0	0	0
				1	0	1	0	0	0	0
				1	1	x	x	x	x	x
0	1	0	0	0	0	0	1	0	0	0
				0	1	1	0	0	0	0
				1	0	1	0	0	0	0
				1	1	x	x	x	x	x
1	0	0	0	x	x	1	0	0	0	1

d. Mealy Vending MachineFSM state transition diagram and table

1. FSM state transition diagram

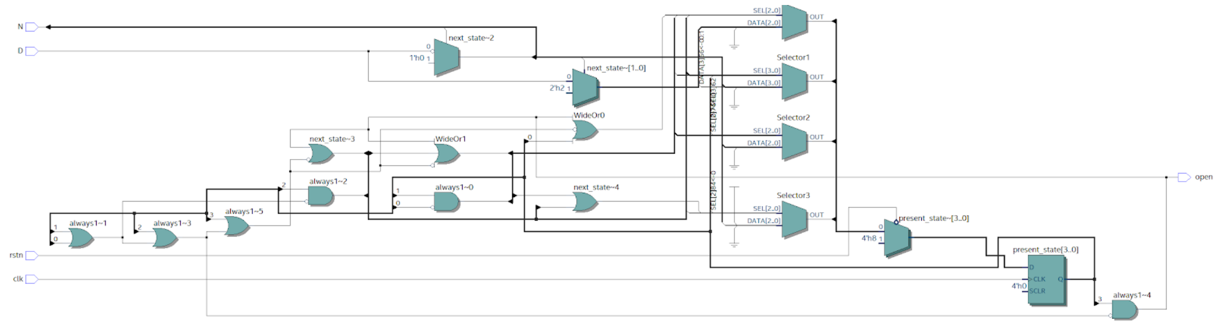


2. FSM state transition table

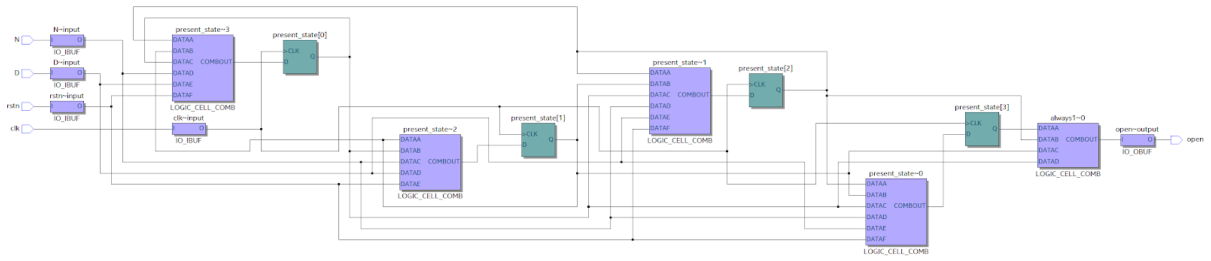
state[3]	state[2]	state[1]	state[0]	D	N	next[3]	next[2]	next[1]	next[0]	open
0	0	0	1	0	0	0	0	0	1	0
				0	1	0	0	1	0	0
				1	0	0	1	0	0	0
				1	1	x	x	x	x	x
0	0	1	0	0	0	0	0	1	0	0
				0	1	0	1	0	0	0
				1	0	1	0	0	0	1
				1	1	x	x	x	x	x
0	1	0	0	0	0	0	1	0	0	0
				0	1	1	0	0	0	1
				1	0	1	0	0	0	1
				1	1	x	x	x	x	x
1	0	0	0	x	x	1	0	0	0	1

e. Moore schematic and usage table

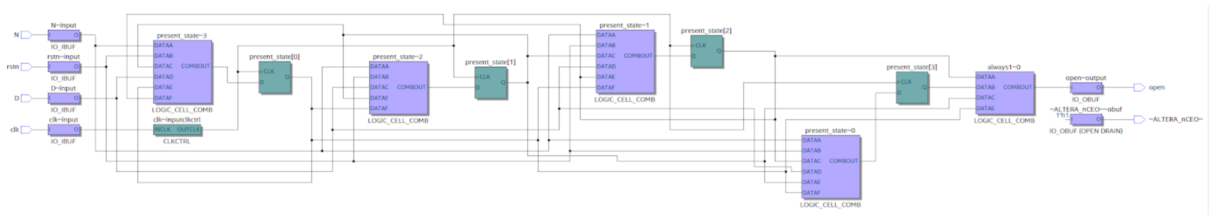
1. RTL schematic



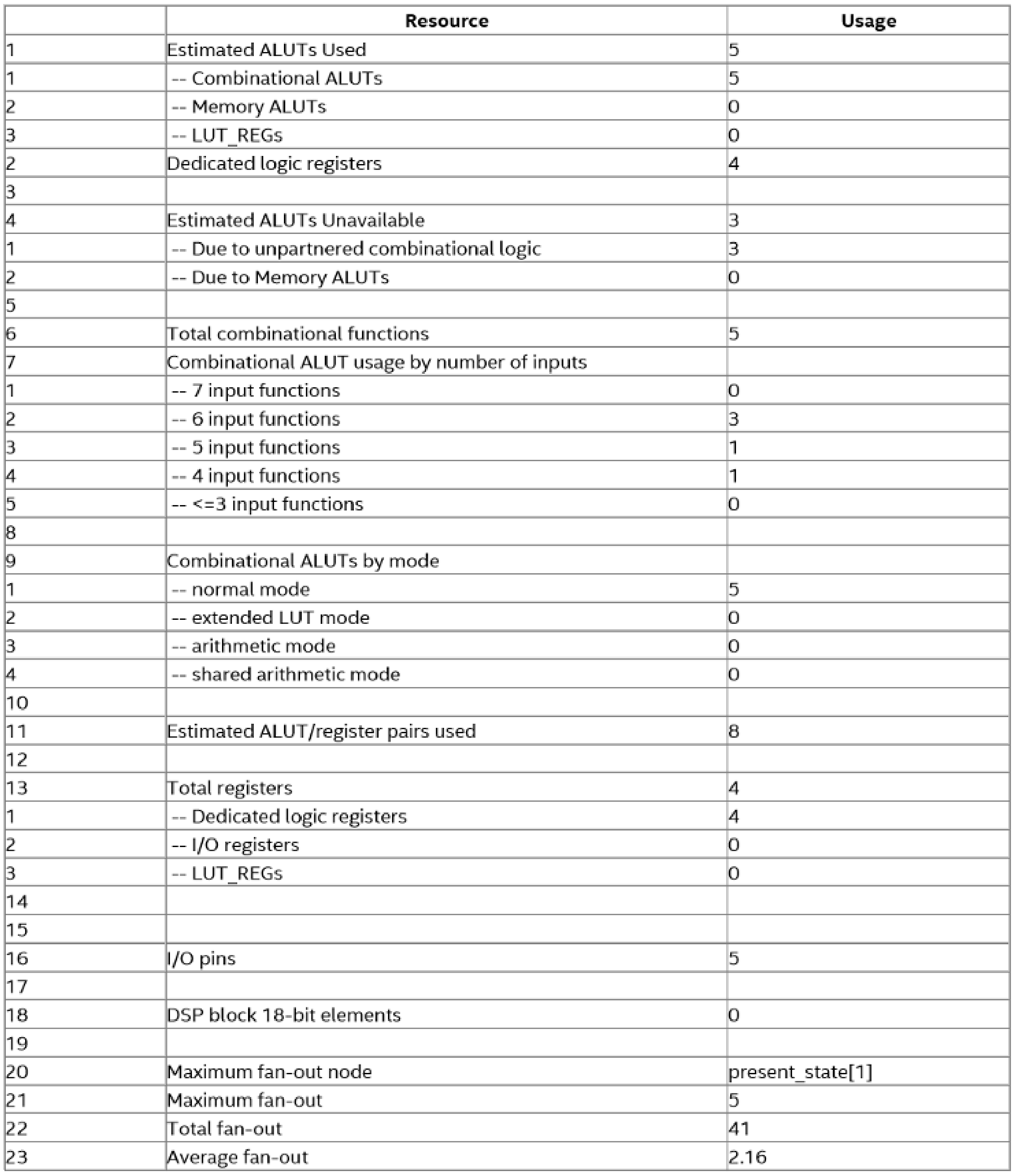
2. Post mapping schematic



3. Post fitting schematic



4. Resource Usage table (Moore)



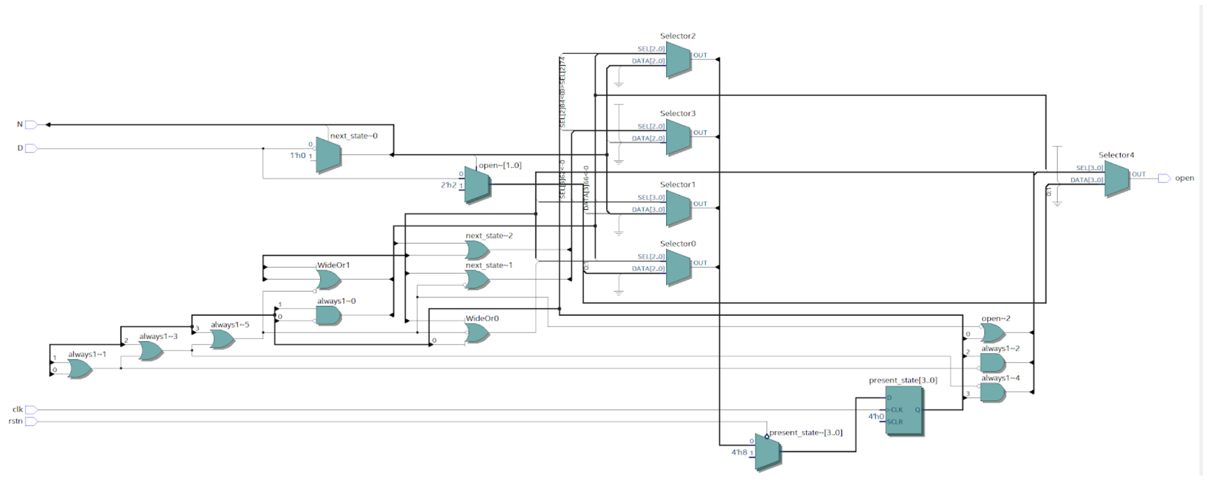
	Resource	Usage
1	Estimated ALUTs Used	5
1	-- Combinational ALUTs	5
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	4
3		
4	Estimated ALUTs Unavailable	3
1	-- Due to unpartnered combinational logic	3
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	5
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	3
3	-- 5 input functions	1
4	-- 4 input functions	1
5	-- <=3 input functions	0
8		
9	Combinational ALUTs by mode	
1	-- normal mode	5
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	8
12		
13	Total registers	4
1	-- Dedicated logic registers	4
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	5
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	present_state[1]
21	Maximum fan-out	5
22	Total fan-out	41
23	Average fan-out	2.16

Number of ALUT: 5 (5 I/O pins)
Number of Functions: 5

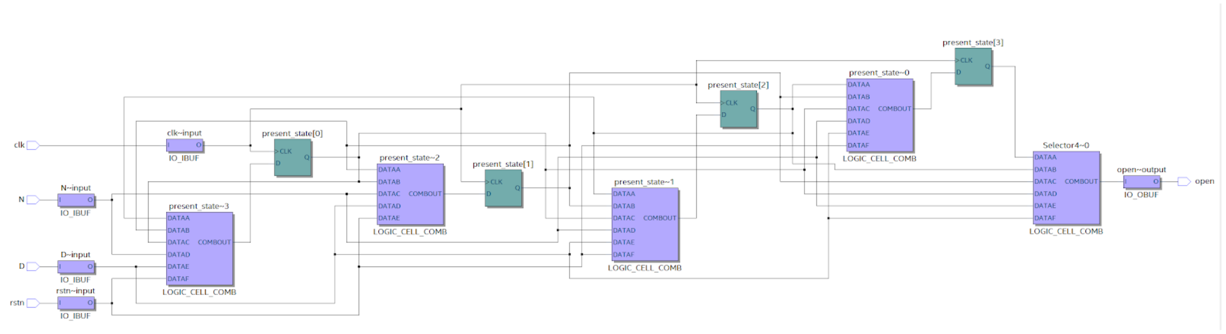
3 (6 input functions)
1 (5 input functions)
1 (4 input functions)

f. Mealy schematic and usage table

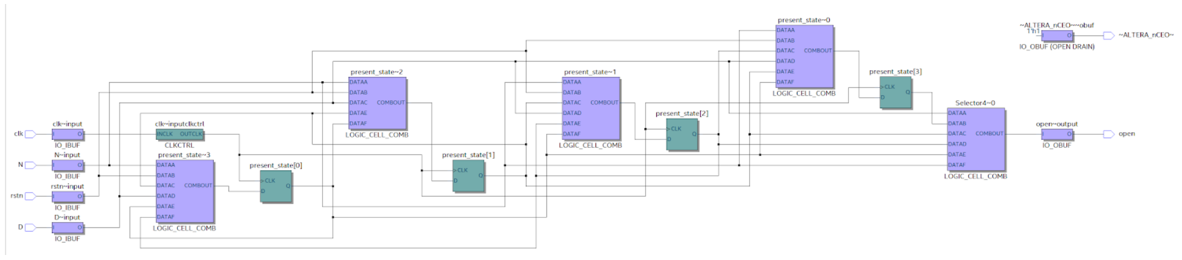
1. RTL schematic



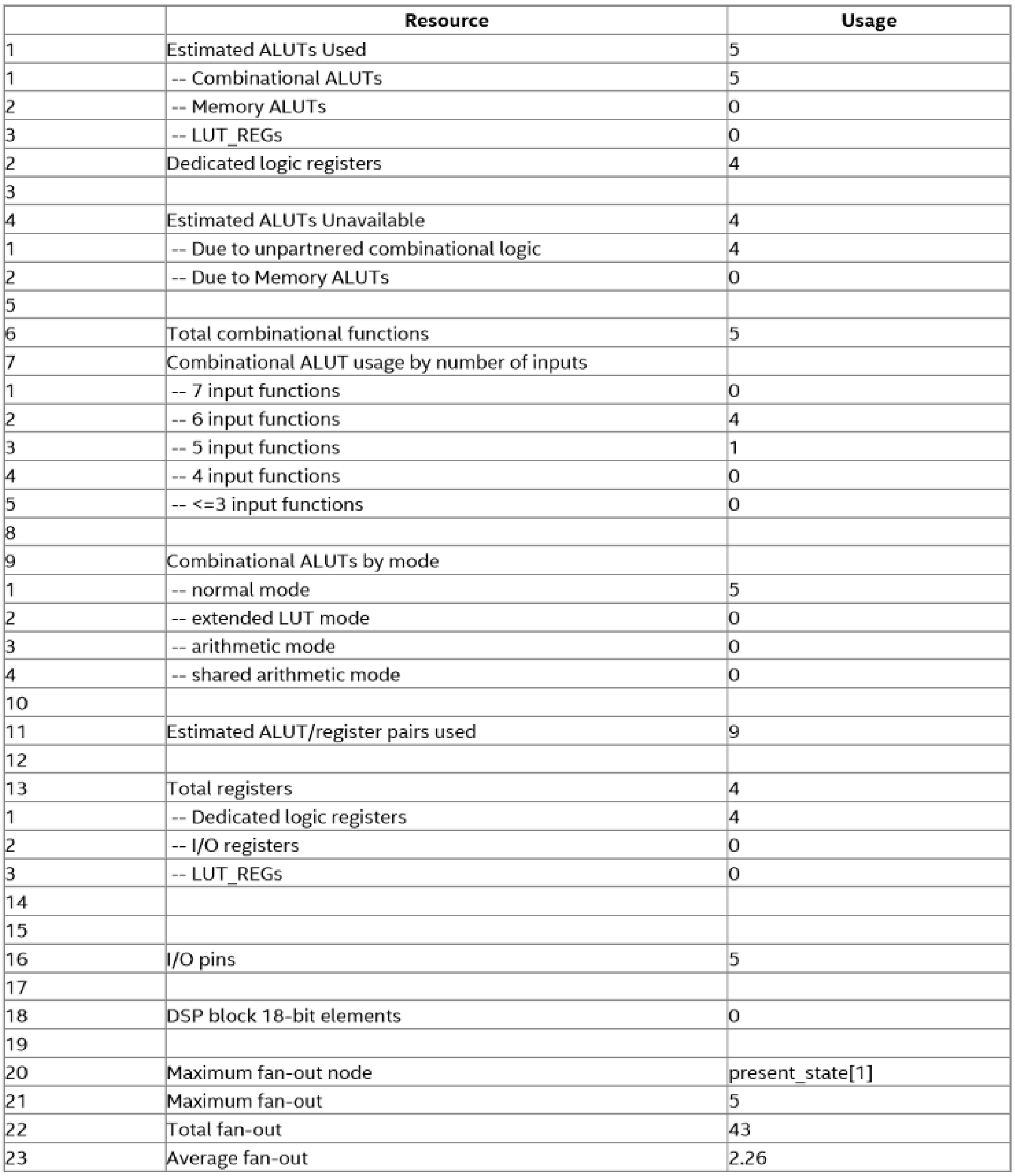
2. Post mapping schematic



3. Post fitting schematic



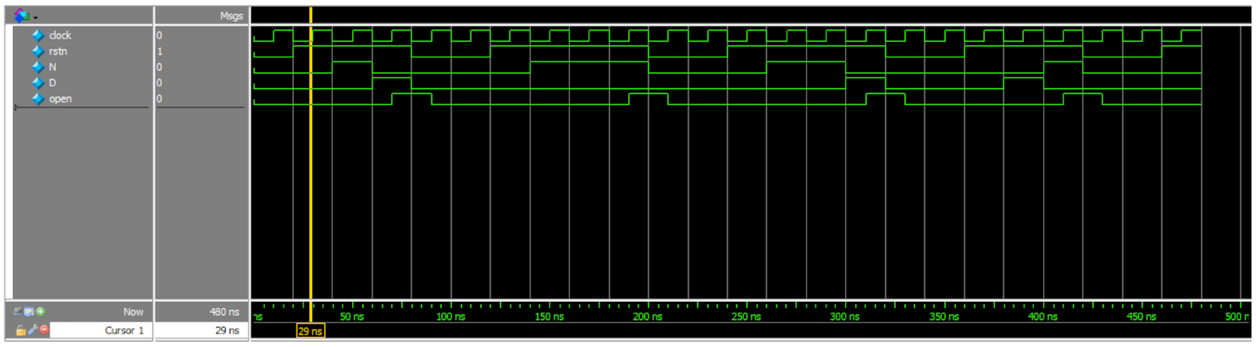
4. Resource Usage table (Mealy)



	Resource	Usage
1	Estimated ALUTs Used	5
1	-- Combinational ALUTs	5
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	4
3		
4	Estimated ALUTs Unavailable	4
1	-- Due to unpartnered combinational logic	4
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	5
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	4
3	-- 5 input functions	1
4	-- 4 input functions	0
5	-- <=3 input functions	0
8		
9	Combinational ALUTs by mode	
1	-- normal mode	5
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	9
12		
13	Total registers	4
1	-- Dedicated logic registers	4
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	5
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	present_state[1]
21	Maximum fan-out	5
22	Total fan-out	43
23	Average fan-out	2.26

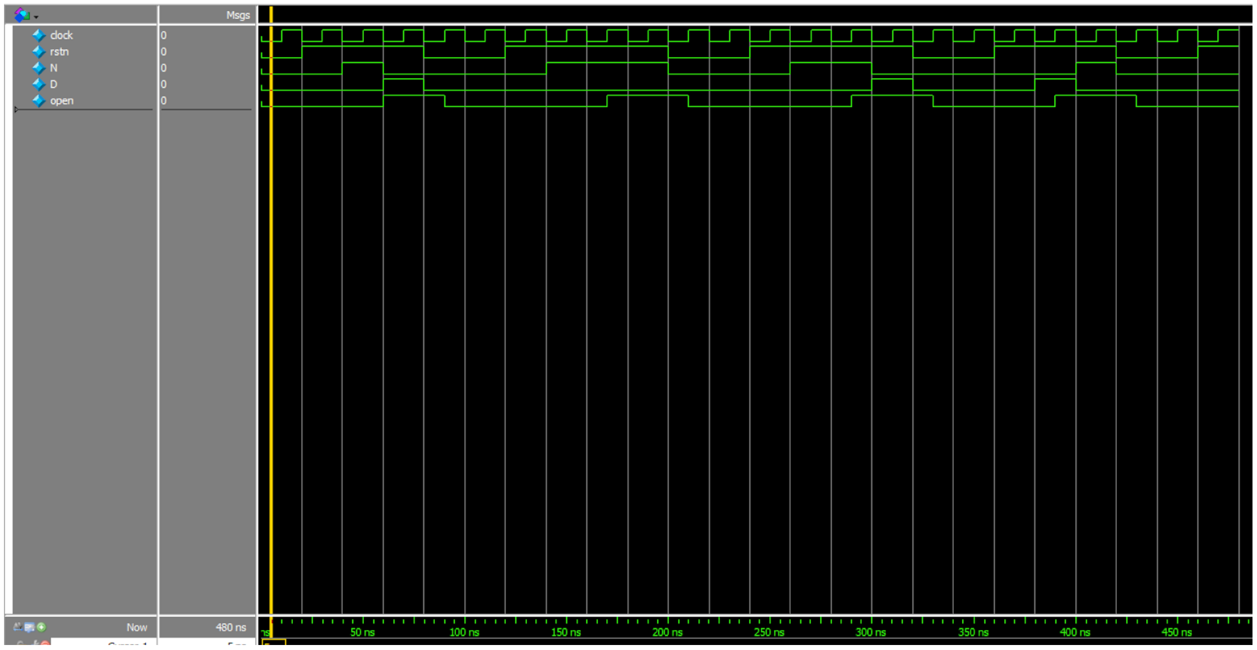
Number of ALUT: 5 (5 I/O pins)
Number of Functions: 5
    4 (6 input functions)
    1 (5 input functions)

g. Moore simulation waveform and explanation



The vending machine only takes nickels and dimes, and only opens when there is a total of 15 cents, then the machine resets. In the Moore machine we can see that the output is synchronous since it only changes at the positive clock signal.

h. Mealy schematic and usage table



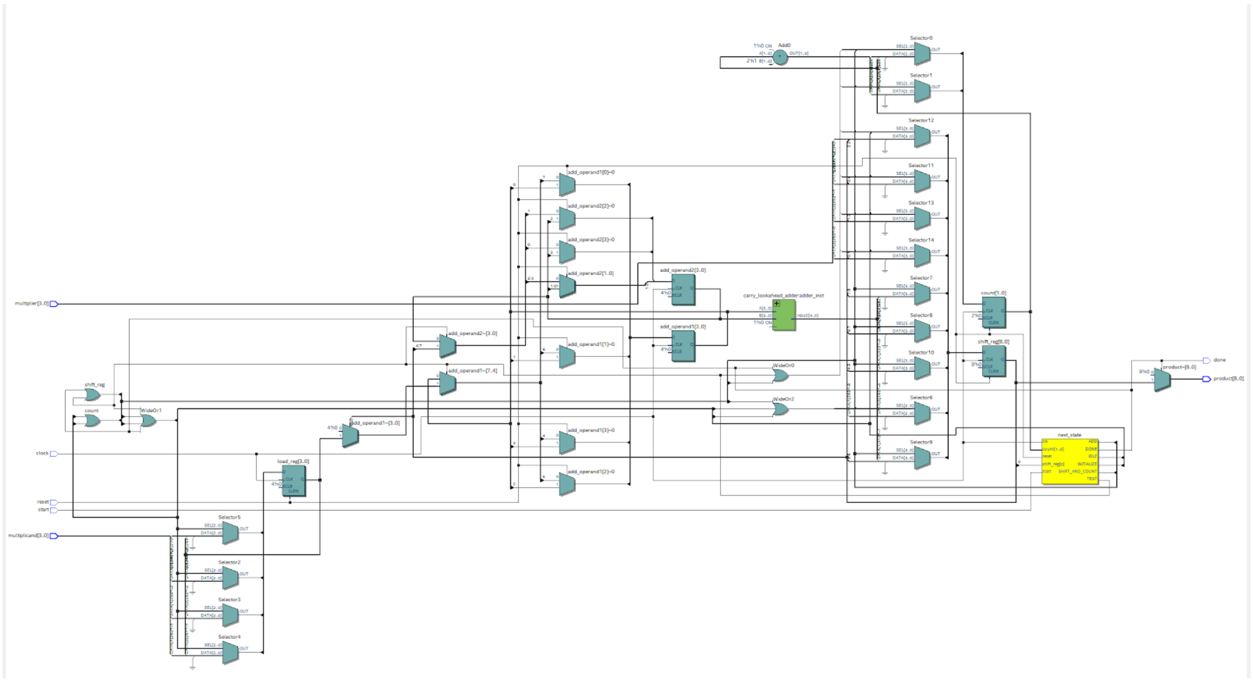
Notice that since we did not register the output in the Mealy machine, the output changes as soon as the input changes. Other than that, the output should be the same as the Moore machine because both describe the same process.

Homework 7c: SystemVerilogRTL model for N-bit Integer Multiplier

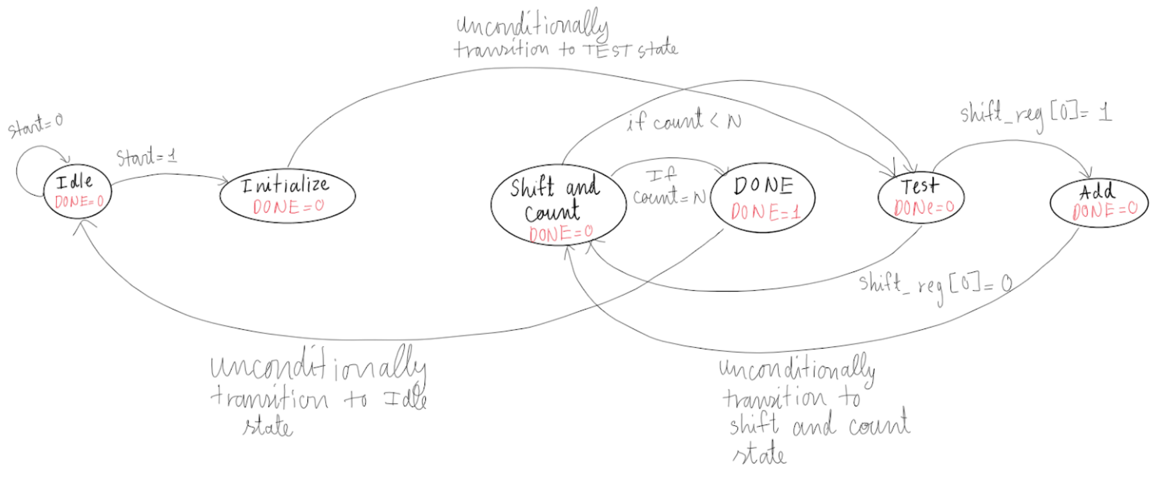
a. Design code

```
`timescale 1ns/1ps
//`include "carry_lookahead_adder.sv"
module integer_multiplier
#(parameter N=4)
(
  input clock, reset, start,
  input logic[N-1:0] multiplicand, multiplier,
  output logic[(2*N):0] product,
  output logic done
);

logic [$clog2(N)-1:0] count;
logic[N-1:0] load_reg;
logic[(2*N):0] shift_reg;

logic[N-1:0] add_operand1, add_operand2;
logic[N:0] sum;

enum logic[2:0]{
  IDLE             = 3'b000,
  INITIALIZE       = 3'b001,
  TEST             = 3'b010,
  ADD              = 3'b011,
  SHIFT_AND_COUNT  = 3'b100,
  DONE             = 3'b101
} next_state;



carry_lookahead_adder #(.N(N)) adder_inst(
.A (add_operand1),
.B (add_operand2),
.result(sum),
.CIN(0)
);

always_ff@(posedge clock, posedge reset) begin
 if(reset) begin
   count <= 0;
   next_state <= IDLE;
   load_reg <= 0;
   shift_reg <= 0;
 end
 else begin
    case(next_state)
       IDLE: begin
        count<=0;
        load_reg<=0;
        shift_reg<=0;
        if(start<=1) begin
            next_state<=INITIALIZE;
        end

        else begin
            next_state<=IDLE;
        end

       end

       INITIALIZE: begin
         shift_reg <= {1'b0, {N{1'b0}}, multiplier};
         count<=0;
         next_state<=TEST;
         load_reg<=multiplicand;
        end

        TEST: begin
           if(shift_reg[0] == 1'b1) begin
            next_state<=ADD;
            add_operand1<=load_reg;
            add_operand2<=shift_reg[(2*N)-1:N];

           end
           else begin
            add_operand1<=0;
            next_state<=SHIFT_AND_COUNT;
            add_operand2<=shift_reg[(2*N)-1:N];
           end
     end

     ADD: begin
          shift_reg <= {sum, shift_reg[N-1:0]};
          next_state<=SHIFT_AND_COUNT;
         end

         SHIFT_AND_COUNT: begin
            shift_reg<=shift_reg>>1;
            count<=count+1;

           if(count == N-1) begin
               next_state<=DONE;

           end
       else begin
              next_state<=TEST;

           end
         end

         DONE: begin
            next_state <= IDLE;
     end
    endcase
 end
end

assign done = (next_state == DONE) ? 1 : 0;

assign product = (next_state == DONE) ? shift_reg : 0;

endmodule: integer_multiplier
```

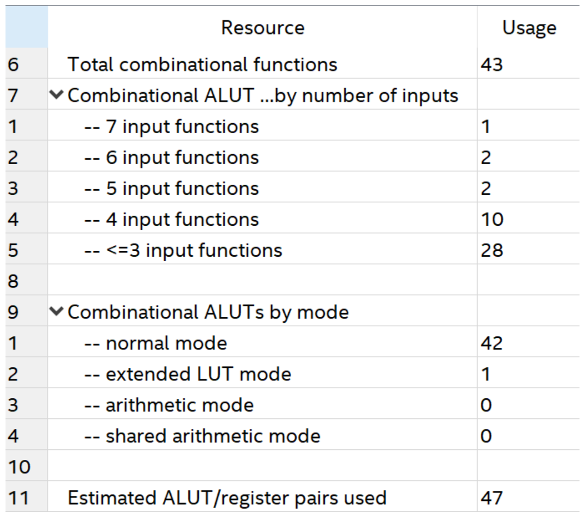
b. RTL schematic



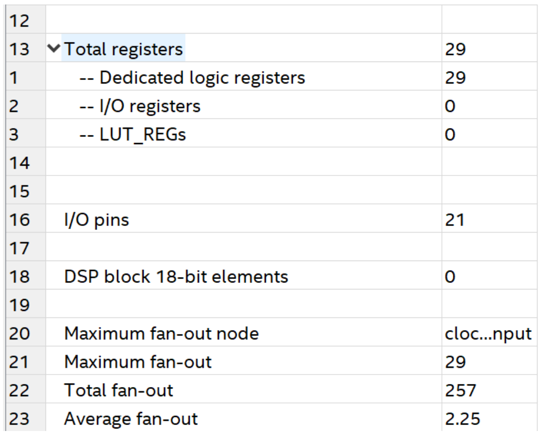
c. FSM diagram



d. Resource usage summary

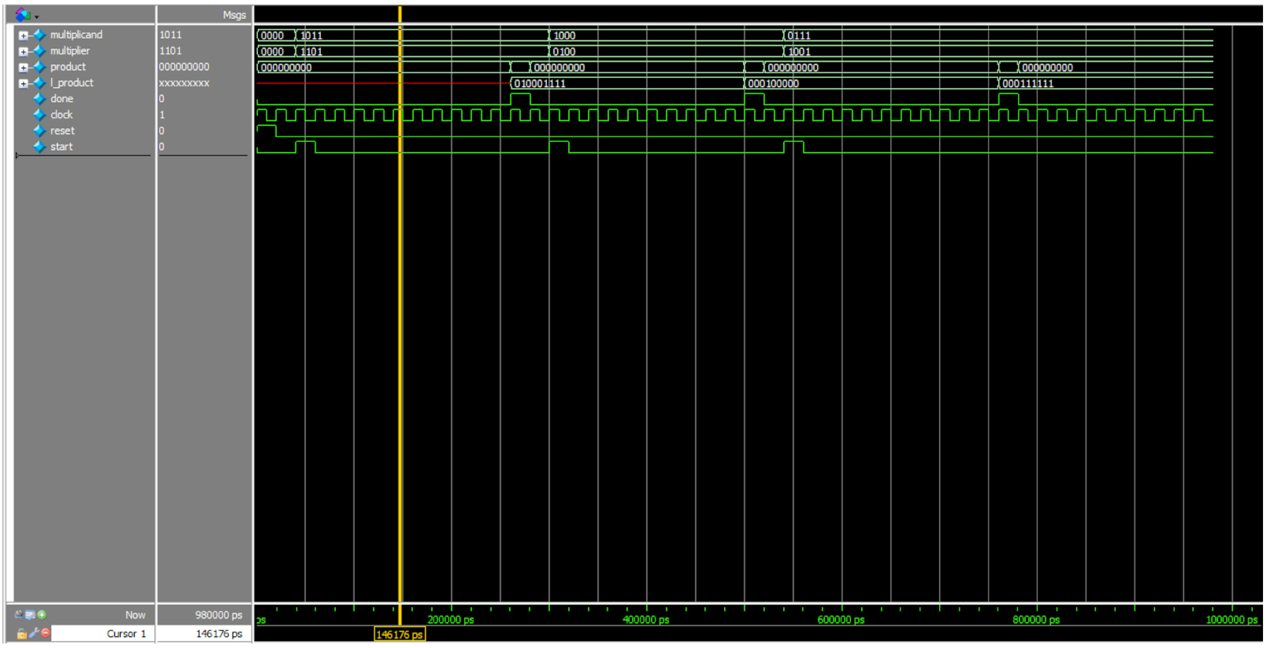


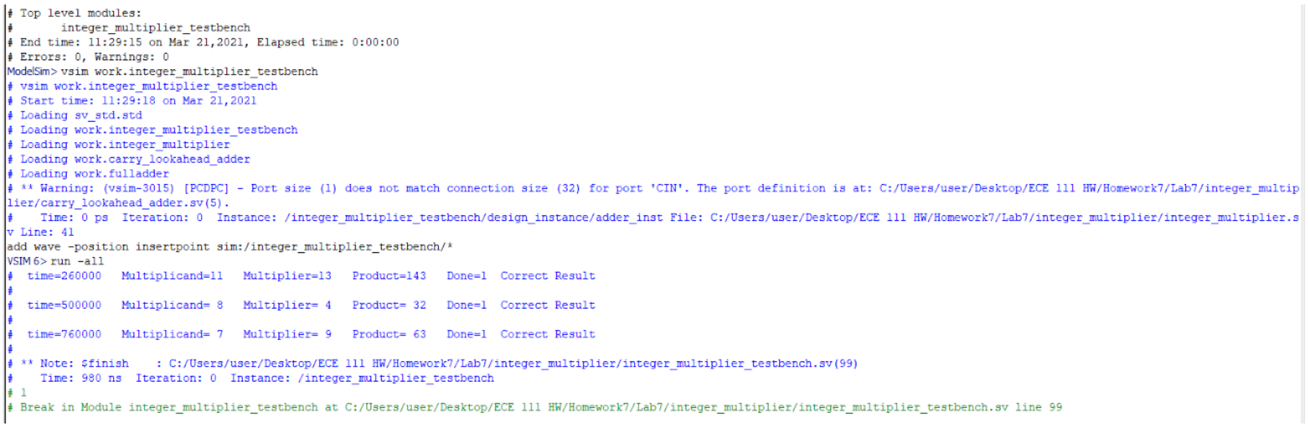
	Resource	Usage
6	Total combinational functions	43
7	⌄ Combinational ALUT ...by number of inputs	
1	-- 7 input functions	1
2	-- 6 input functions	2
3	-- 5 input functions	2
4	-- 4 input functions	10
5	-- <=3 input functions	28
8		
9	⌄ Combinational ALUTs by mode	
1	-- normal mode	42
2	-- extended LUT mode	1
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	47



12		
13	⌄ Total registers	29
1	-- Dedicated logic registers	29
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	21
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	cloc...nput
21	Maximum fan-out	29
22	Total fan-out	257
23	Average fan-out	2.25

e. Simulation waveform, transcript, and explanations





```
# Top level modules:
#       integer_multiplier_testbench
# End time: 11:29:15 on Mar 21,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
ModelSim> vsim work.integer_multiplier_testbench
# vsim work.integer_multiplier_testbench
# Start time: 11:29:18 on Mar 21,2021
# Loading sv_std.std
# Loading work.integer_multiplier_testbench
# Loading work.integer_multiplier
# Loading work.carry_lookahead_adder
# Loading work.fulladder
# ** Warning: (vsim-3015) [PCDPC] - Port size (1) does not match connection size (32) for port 'CIN'. The port definition is at: C:/Users/user/Desktop/ECE 111 HW/Homework7/Lab7/integer_multip
lier/carry_lookahead_adder.sv(5).
#    Time: 0 ps  Iteration: 0  Instance: /integer_multiplier_testbench/design_instance/adder_inst File: C:/Users/user/Desktop/ECE 111 HW/Homework7/Lab7/integer_multiplier/integer_multiplier.s
v Line: 41
add wave -position insertpoint sim:/integer_multiplier_testbench/*
VSIM 6> run -all
#  time=260000   Multiplicand=11   Multiplier=13   Product=143   Done=1  Correct Result
#
#  time=500000   Multiplicand= 8   Multiplier= 4   Product= 32   Done=1  Correct Result
#
#  time=760000   Multiplicand= 7   Multiplier= 9   Product= 63   Done=1  Correct Result
#
# ** Note: $finish    : C:/Users/user/Desktop/ECE 111 HW/Homework7/Lab7/integer_multiplier/integer_multiplier_testbench.sv(99)
#    Time: 980 ns  Iteration: 0  Instance: /integer_multiplier_testbench
# 1
# Break in Module integer_multiplier_testbench at C:/Users/user/Desktop/ECE 111 HW/Homework7/Lab7/integer_multiplier/integer_multiplier_testbench.sv line 99
```

Explanation: The integer multiplier multiplies N bits binary values by utilizing the add and shift algorithm. When we have two binary values, we first multiply the values normally; 0x0=0, 0x1=0, 1x0=0,1x1=1. Then we add all the binary values together using the carry_look_ahead_adder and the shift register because binary values are shifted in addition. Looking at the waveform, we can see that product calculation does not begin until “start” is high, while, “done” is only high when the final product values are calculated. In the waveform we have three examples:

Binary to decimal

1011 => 11

1101=> 13

11x13= 143 (matches transcript)

1000=> 8
0100=> 4
8x4=32 (matches transcript)

0111=> 7
1001=> 9
7x9=63 (matches transcript)