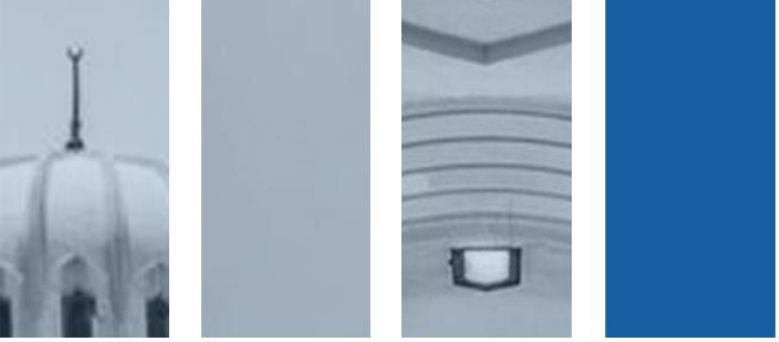

Group 20 PROJECT 2 – Requirement 3

Program: Computer Engineering and Software Systems

Course Code: CSE312

Course Name: Electronic Design Automation

Examination Committee

Dr. Eman Mohamed

Eng. Adham Hazem

Ain Shams University

Faculty of Engineering

Fall Semester – 2022

December 12, 2022

Students Personal Information

Student Name: Engy ahmed mostafa
Student Code: 20P4230

Student Name: Youssef Emad Eldin
Student Code: 20P3844

Student Name: Mark Bassem Heshmat
Student Code: 21P0363

Student Name: Nour Ayman
Student Code: 20P3076

Student Name: Abdelrahman hassan fouad
Student Code: 20P3822

Student Name: Mohamed Salah Ibrahim
Student Code: 20P7886

Student Name: Amr Ayman Salaheldin Hamouda
Student Code: 20P5694

Student Name: Ahmed mohamed ahmed amin
Student Code: 20P2629

1 Table of Contents

1. Verilog Design

We have used 3 different design with different difficulty levels starting with an and gate only to a Moore code using clock and rst. we made the test bench generator as generic as possible.

1.1. And gate(easiest)

the code takes 2 bits from the user in 2 separate variables and ands them together and puts the result in a variable called out according to the following truth table

A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

```
1  module and_gate (
2
3  input   wire    in1,
4  input   wire    in2,
5  output  wire    out
6
7  );
8
9
10 assign out = in1 & in2 ;
11
12 endmodule
```

Figure 1and gate design

1.2. Shifter

Shifter shifts the input either to the right or to the left one bit as required by the user through flagging either the left or right variables by 1. For the input to be shifted the load must be 1 and the clock's edge must be positive. In this case the input value is stored in an internal register then shifted either 1 bit to the right or to the left.

```
1  module shifter (
2
3      input                   clk,
4      input                   load,
5      input                   right,
6      input                   left,
7      input           [4:0]   in_value,
8      output  reg     [4:0]   value
9
10 );
11
12 reg [4:0] internal_reg ;
13
14 always @ (posedge clk)
15  begin
16      if (load)
17       begin
18          internal_reg <= in_value ;
19       end
20      else if (right)
21       begin
22          internal_reg <= internal_reg >> 1 ;
23          value <= internal_reg ;
24       end
25      else if (left)
26       begin
27          internal_reg <= internal_reg << 1 ;
28          value <= internal_reg ;
29       end
30
31  end
32
33
34 endmodule
```

Figure 2 shifter code

1.3. MOORE Code

1.3.1. Finite state diagram

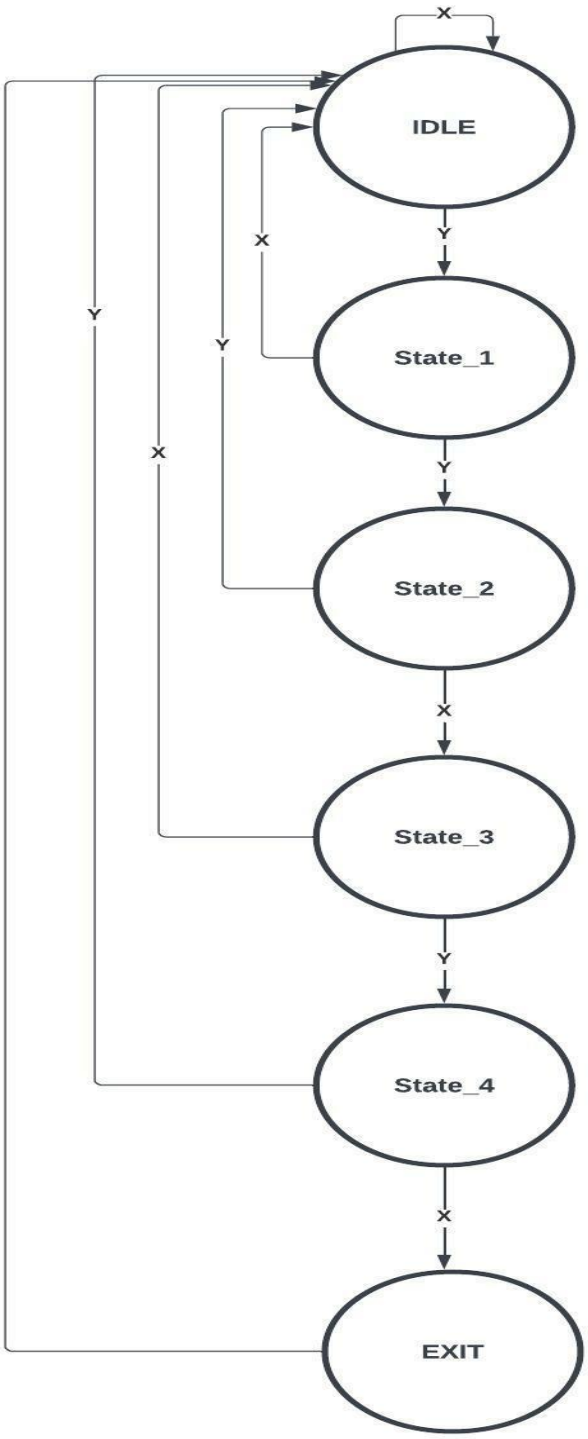

Figure 3 MOORE STATE DIGRAM

1.3.2. FSM plan and code

The design is used to transmit between states according to giving input.in the beginning if the rst was 0 the current state will be idle when the rst change to 0 the current state will be equal to the next state Going to the cases step. It will be IDLE if the input was x if the input was y it will go to the next state state_1 else will stay at idle .when it enter case state_1 if the user input was X the next state will be idle and if the user enter Y the next state will be state_2.in case state_2 if the user enter y the next state will be idle else if the user enter x the next state will be stae_3.in State_3 case if the user enter x the next state will be IDLE else if the user enter x the next state will State_4.in case state_4 if the user enter x it will be exit if the user enter y the next state will be idle. The output will be only form exit as the unlock key is only 1 at exit

```
module FSM (
input  wire        X,
input  wire        Y,
input  wire        rst,
input  wire        clk,
output reg         unlock
);



localparam  [2:0]    IDLE = 3'b000,
                     state_1 = 3'b001,
                     state_2 = 3'b011,
                     state_3 = 3'b010,
                     state_4 = 3'b110,
                     Exit = 3'b111 ;

reg    [2:0]         current_state,
                     next_state ;

// state transition
always @(posedge clk or negedge rst)
 begin
  if(!rst)
   begin
    current_state <= IDLE ;
   end
  else
   begin
    current_state <= next_state ;
   end
 end

// next_state logic
always @(*)
 begin
  case(current_state)
  IDLE     : begin
               if(X)
                next_state = IDLE ;
               else if (Y)
                next_state = state_1 ;
               else
                next_state = IDLE ;
              end
  state_1       : begin
               if(X)
```

```
47    state_1        : begin
48                     if(X)
49                      next_state = IDLE ;
50                     else if (Y)
51                      next_state = state_2 ;
52                     else
53                      next_state = state_1 ;
54                   end
55    state_2        : begin
56                     if(X)
57                      next_state = state_3 ;
58                     else if (Y)
59                      next_state = IDLE ;
60                     else
61                      next_state = state_2 ;
62                   end
63    state_3        : begin
64                     if(X)
65                      next_state = IDLE ;
66                     else if (Y)
67                      next_state = state_4 ;
68                     else
69                      next_state = state_3 ;
70                   end
71    state_4        : begin
72                     if(X)
73                      next_state = Exit ;
74                     else if (Y)
75                      next_state = IDLE ;
76                     else
77                      next_state = state_4 ;
78                   end
79    Exit    : begin
80                      next_state = IDLE ;
81                   end
82    default :    next_state = IDLE ;
83
84    endcase
85  end
86
87
88  // next_state logic
89  always @(*)
90   begin
91    case(current_state)
92    IDLE       : begin
93                      unlock    =  1'b0 ;
```

```
87
88  // next_state logic
89  always @(*)
90   begin
91    case(current_state)
92    IDLE       : begin
93                   unlock    =  1'b0 ;
94                 end
95    state_1        : begin
96                   unlock    =  1'b0 ;
97                 end
98    state_2        : begin
99                   unlock    =  1'b0 ;
00                 end
01    state_3        : begin
02                   unlock    =  1'b0 ;
03                 end
04    state_4        : begin
05                   unlock    =  1'b0 ;
06                 end
07    Exit    : begin
08                   unlock    =  1'b1 ;
09                 end
10    default  : begin
11                   unlock    =  1'b0 ;
12                 end
13    endcase
14  end
15
16
17  endmodule
```

Figure 4 Fsm code

2. Python code

2.1 Steps to run code

1- install python 3.6+

2- run main.py from the same directory as the desired design file (so you won't have to enter the fil's absolute path)

3- the generated test bench file will be in the same directory as the directory you ran python from, and its name will be the module_name + "_tb. v"

2.2 Implementation

first, we extract the inputs and outputs from functions get_inputs/ get_outputs

then we check if there is a clk exists and a rst

after that we parse the inputs and outputs so they can be regs and wires

after that we initialize the variables based on the inputs gathered and monitor all the outputs

lastly, we generate the tests and the randoms based on the inputs

C: > Users > Engy_ > Downloads > main.py

```python
import random

def get_inputs_outputs(design):
        # split all statments ending in ;
        statements = design.split(';')
        # get first statement which is the module initialization
        moduleInitStatement = statements[0]
        # remove any redudent spaces
        moduleInitStatement = ' '.join(moduleInitStatement.split())
        # get ports area by getting the loc of the two parentheses
        portsArea = moduleInitStatement[moduleInitStatement.find('(') + 1: moduleInitStatement.find(')')].strip("
        # array of ports
        ports = portsArea.split(",")

        inputs = []
        outputs = []

        # iterate over each port
        for port in ports:
            # split to port type(input, output) and any other type including reg or wire..
            portTypeAndName = port.strip(" ").split(" ")
            # port type(input, output)
            portType = portTypeAndName[0]
            # other data including port name and bit count if vector
            portName = portTypeAndName[1:]
            if portType == "input":
                inputs.append(' '.join(portTypeAndName[1:]))
            else:
                outputs.append(' '.join(portTypeAndName[1:]))
```

```
                outputs.append(' '.join(portTypeAndName[1:]))
        return inputs, outputs


def init_inputs(inputs):
    s = ""
    for inp in inputs:
        inp = inp.replace("reg ", "").replace("wire ", "")
        if "clk" in inp:
            s+=f"{inp} = 1;\n"
            continue
        if "rst" in inp:
            s+=f"{inp} = 0;\n"
            continue
        if len(inp.split()) == 1:
            s+= f"{inp} = 0;\n"
        else:
            s+= f"{inp.split()[1]} = 0;\n"


    return s

def generate_tests(inputs, outputs):
    s=""
    for l in range(6):
        for inp in inputs:
            if 'clk' in inp:
```

```
def generate_tests(inputs, outputs):
    s=""
    for l in range(6):
        for inp in inputs:
            if 'clk' in inp:
                continue
            if 'rst' in inp:
                continue
            inp = inp.replace("reg", "").replace("wire", "")
            # print(inp)
            if len(inp.split()) == 1:
                bit = random.randrange(0,2)
                s+= f"#2 {inp} = 1'b{bit};\n"
            else:
                n = int(inp.split()[0][1])
                bits = ''
                for i in range(n):
                    bits += str(random.randrange(0,2))
                s+= f"#2 {inp.split()[1]} = {n+1}'b{bits};\n"

        if(l==0):
            for out in outputs:
                if len(out.split()) > 1:
                    out = out.split()[-1]
                s+= f'$mointor("monitor value =%b" , {out});\n'
        s+= f'$display("Test case {l+1}"); \n'
        s+="#40;\n\n"

```

```
            if(l==0):
                for out in outputs:
                    if len(out.split()) > 1:
                        out = out.split()[-1]
                    s+= f'$monitor("monitor value =%b" , {out});\n'
            s+= f'$display("Test case {l+1}"); \n'
            s+="#40;\n\n"

        return s


    def generate_randoms(inputs):
        rst = extract_reset(extracted_inputs)
        s=''
        if rst:
            s+="rst=0;\nrst=1;\n"
        s+="for(i=0;i<1000000;i=i+1)\nbegin\n#40\n"
        for l in range(6):
            seed = random.choice(["seed1", "seed2", "seed3"])
            for inp in inputs:
                if 'clk' in inp:
                    continue
                if 'rst' in inp:
                    continue
                inp = inp.replace("reg ", "").replace("wire", "")
                if len(inp.split()) == 1:
                    bit = random.randrange(0,2)
                    s+= f"#2 {inp} = $random({seed});\n"
                else:
                    n = int(inp.split()[0][1])
                    bits = ''
                    for i in range(n):
                        bits += str(random.randrange(0,2))
                    s+= f"#2 {inp.split()[1]} = $random({seed});\n"
                # s+= f"#2 {inp} = $random({seed});\n"

            s+="#40;\n\n"

        s+="end\n"

        return s
    def generate_list(string: str, item):
        for line in string.split('\n'):
            if item in line:
                yield line

    def parsed_instantiations(inputs):
        s = ""
        for inp in inputs:
            list1 = inp.split(" ")
            inputName = list1[len(list1)-1]
            s+= f"        .{inputName}({inputName}),\n"
        s = s[0:len(s)-2]
        return s


    def extract_clock(inputs):
```

```
            if "clk" in inp:
                #return "always #5 clk = !clck;"
                return """
initial
    begin
    clk = 0;
    forever begin
    #5;
    clk = ~clk;

    end
end
                """
    return ""

def extract_reset(inputs):
    for inp in inputs:
        if "rst" in inp:
            return "#10 rst = 1;\n"
    return ""

def parsed_inputs(inputs):
    s = ""
    for inp in inputs:
        replacedStr = inp.replace("reg ", "").replace("wire", "")
        s+= f"reg {replacedStr};\n"

    return s
def parsed_outputs(inputs):
    s = ""
    for inp in inputs:
        replacedStr = inp.replace("reg ", "").replace("wire", "")
        s+= f"wire {replacedStr};\n"

    return s

# design = open("../2- Shift_Register/Shifter.v").read()
# design = open("./AND_GATE_assign.v").read()

x = input("Enter file name: ")
design = open(x).read()

module_name = design.split()[1]

extracted_inputs, extracted_outputs = get_inputs_outputs(design)
clk = extract_clock(extracted_inputs)
rst = extract_reset(extracted_inputs)
test_inputs = parsed_inputs(extracted_inputs)

test_outputs = parsed_outputs(extracted_outputs)

parsed_instantiations = parsed_instantiations(extracted_inputs+extracted_outputs)

inital = init_inputs(extracted_inputs)
tests = generate_tests(extracted_inputs, extracted_outputs)
randoms = generate_randoms(extracted_inputs)

```

```
178   test_outputs = parsed_outputs(extracted_outputs)
179
180   parsed_instantiations = parsed_instantiations(extracted_inputs+extracted_outputs)
181
182   inital = init_inputs(extracted_inputs)
183   tests = generate_tests(extracted_inputs, extracted_outputs)
184   randoms = generate_randoms(extracted_inputs)
185
186
187   out = f"""module {module_name}_tb ();
188   {test_inputs}
189   {test_outputs}
190
191   {module_name} DUT (
192   {parsed_instantiations}
193       );
194
195   integer seed1=10;
196   integer seed2=20;
197   integer seed3=30;
198   integer i=0;
199
200   {clk}
201
202   initial
203       begin
204       $dumpfile("{module_name}.vcd");
205       $dumpvars ;
206
207   //initial values
```

```
198    integer i=0;
199
200    {clk}
201
202    initial
203        begin
204        $dumpfile("{module_name}.vcd");
205        $dumpvars ;
206
207    //initial values
208    {inital}
209
210        #40;
211    {tests}
212    {randoms}
213    {"rst = 0;" if rst else ""}
214        #100;
215        $finish();
216        end
217
218
219
220    endmodule
221    """
222    # print(out)
223    print(extracted_outputs)
224    output_file = open(module_name+"_tb.v", "w+").write(out)
```

Figure 5 python code

3 Test benches

3.1 And gate test bench

```
module and_gate_tb ();
reg  in1;
reg  in2;

wire  out;


and_gate DUT (
        .in1(in1),
        .in2(in2),
        .out(out)
    );

integer seed1=10;
integer seed2=20;
integer seed3=30;
integer i=0;


initial
    begin
    $dumpfile("and_gate.vcd");
    $dumpvars ;

//initial values
in1 = 0;
in2 = 0;


    #40;
#2  in1 = 1'b1;
#2  in2 = 1'b0;
$monitor("monitor value =%b" , out);
$display("Test case 1");
#40;

#2  in1 = 1'b0;
#2  in2 = 1'b0;
$display("Test case 2");
#40;

#2  in1 = 1'b1;
#2  in2 = 1'b0;
$display("Test case 3");
#40;

#2  in1 = 1'b0;
#2  in2 = 1'b1;
$display("Test case 4");
#40;

#2  in1 = 1'b1;
#2  in2 = 1'b1;
```

```
54    #2  in2 = 1'b1;
55    $display("Test case 5");
56    #40;
57
58    #2  in1 = 1'b0;
59    #2  in2 = 1'b0;
60    $display("Test case 6");
61    #40;
62
63
64    for(i=0;i<1000000;i=i+1)
65    begin
66    #40
67    #2  in1 = $random(seed3);
68    #2  in2 = $random(seed3);
69    #40;
70
71    #2  in1 = $random(seed3);
72    #2  in2 = $random(seed3);
73    #40;
74
75    #2  in1 = $random(seed2);
76    #2  in2 = $random(seed2);
77    #40;
78
79    #2  in1 = $random(seed1);
80    #2  in2 = $random(seed1);
81    #40;
82
83    #2  in1 = $random(seed3);
84    #2  in2 = $random(seed3);
85    #40;
86
87    #2  in1 = $random(seed1);
88    #2  in2 = $random(seed1);
89    #40;
90
91    end
92
93
94        #100;
95        $finish();
96        end
97
98
99
```

Figure 6 and test bench

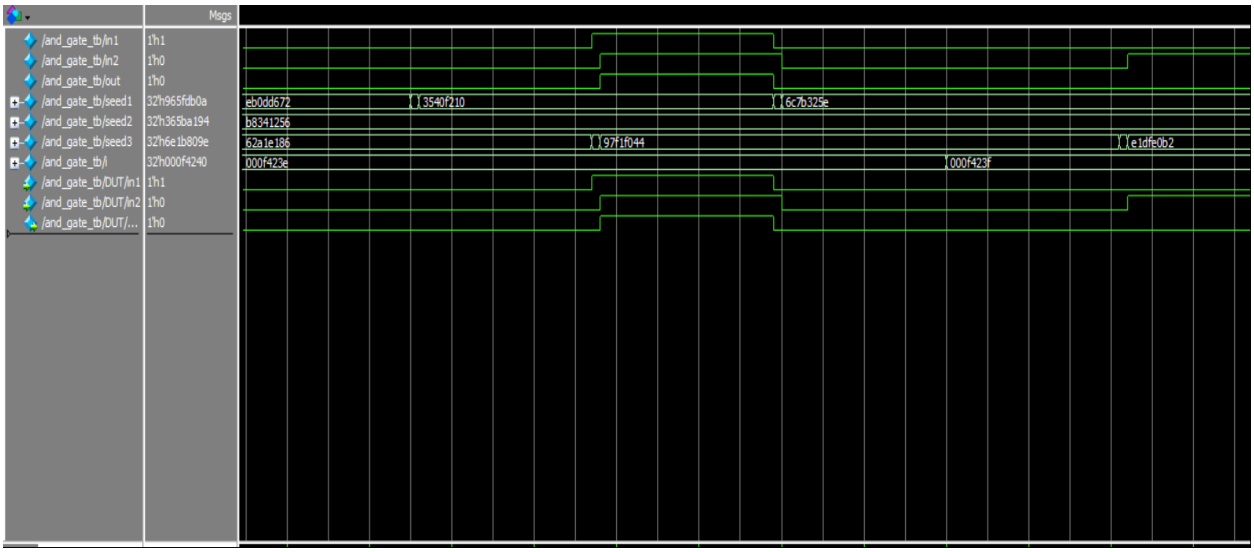

Figure 7 and wavelength

As we made the loop one million for randomization the display is in between almost one million line so this picture is when one million loop

```
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# monitor value =1
# monitor value =0
# ** Note: $finish    : C:/questasim64_10.7c/examples/shifter_tb.v(98)
#    Time: 304000404 ns  Iteration: 0  Instance: /and_gate_tb
# 1
# Break in Module and_gate_tb at C:/questasim64_10.7c/examples/shifter_tb.v line 98
```

Figure 8 monitor of and

However, to make display visible we just decreased the loop for the testing only so it can be easier to detect but the main code will be one million

```
VSIM 2> run -all
# Test case 1
# monitor value =0
# Test case 2
# Test case 3
# Test case 4
# monitor value =1
# Test case 5
# monitor value =0
```

Figure 9 and gate display and monitor

3.2 Shifter test bench

```
module shifter_tb ();
reg clk;
reg load;
reg right;
reg left;
reg [4:0] in_value;

wire [4:0] value;


shifter DUT (
        .clk(clk),
        .load(load),
        .right(right),
        .left(left),
        .in_value(in_value),
        .value(value)
    );

integer seed1=10;
integer seed2=20;
integer seed3=30;
integer i=0;


initial
    begin
    clk = 0;
    forever begin
    #5;
    clk = ~clk;

    end
end


initial
    begin
    $dumpfile("shifter.vcd");
    $dumpvars ;

//initial values
clk = 1;
load = 0;
right = 0;
left = 0;
in_value = 0;

```

```
    #40;
#2 load = 1'b0;
#2 right = 1'b0;
#2 left = 1'b1;
#2 in_value = 5'b0000;
$monitor("monitor value =%b" , value);
$display("Test case 1");
#40;

#2 load = 1'b0;
#2 right = 1'b0;
#2 left = 1'b1;
#2 in_value = 5'b0000;
$display("Test case 2");
#40;

#2 load = 1'b0;
#2 right = 1'b1;
#2 left = 1'b1;
#2 in_value = 5'b1001;
$display("Test case 3");
#40;

#2 load = 1'b0;
#2 right = 1'b1;
#2 left = 1'b0;
#2 in_value = 5'b1111;
$display("Test case 4");
#40;

#2 load = 1'b0;
#2 right = 1'b1;
#2 left = 1'b1;
#2 in_value = 5'b0110;
$display("Test case 5");
#40;

#2 load = 1'b0;
#2 right = 1'b0;
#2 left = 1'b0;
#2 in_value = 5'b0010;
$display("Test case 6");
#40;


for(i=0;i<1000000;i=i+1)
begin
#40
```

```
94
95  for(i=0;i<1000000;i=i+1)
96  begin
97  #40
98  #2 load = $random(seed1);
99  #2 right = $random(seed1);
100 #2 left = $random(seed1);
101 #2 in_value = $random(seed1);
102 #40;
103
104 #2 load = $random(seed2);
105 #2 right = $random(seed2);
106 #2 left = $random(seed2);
107 #2 in_value = $random(seed2);
108 #40;
109
110 #2 load = $random(seed1);
111 #2 right = $random(seed1);
112 #2 left = $random(seed1);
113 #2 in_value = $random(seed1);
114 #40;
115
116 #2 load = $random(seed2);
117 #2 right = $random(seed2);
118 #2 left = $random(seed2);
119 #2 in_value = $random(seed2);
120 #40;
121
122 #2 load = $random(seed3);
123 #2 right = $random(seed3);
124 #2 left = $random(seed3);
125 #2 in_value = $random(seed3);
126 #40;
127
128 #2 load = $random(seed3);
129 #2 right = $random(seed3);
130 #2 left = $random(seed3);
131 #2 in_value = $random(seed3);
132 #40;
133
134 end
135
136
137     #100;
138     $finish();
139     end
140
141
142
143 endmodule
144
```

Figure 10 shifter test bench

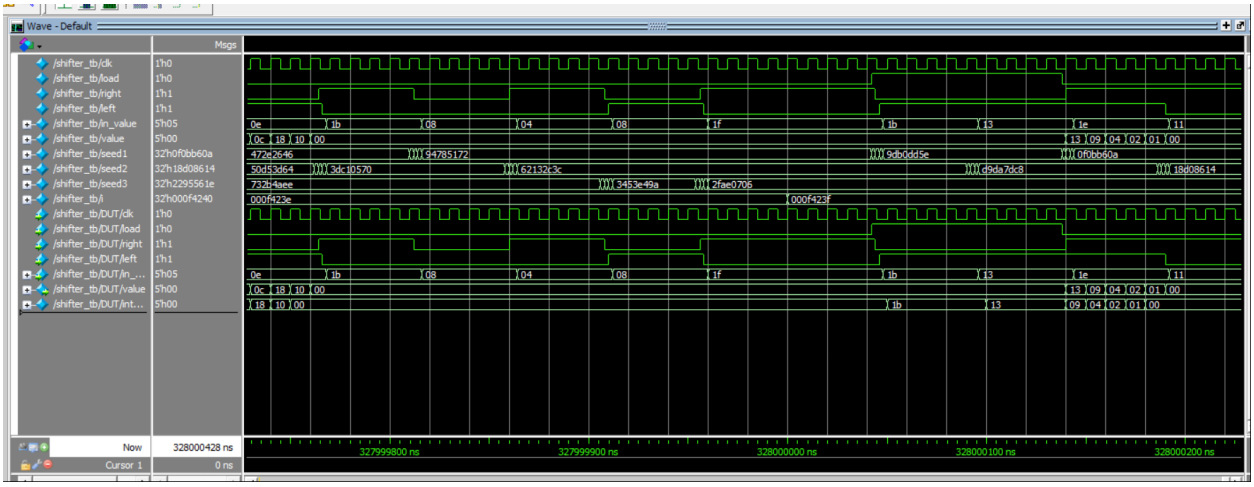

Figure 11 shifter wavelength

As we made the loop one million for randomization the display is in between almost one million line so this picture is when one million loo

```
# monitor value =01100
# monitor value =11000
# monitor value =10000
# monitor value =00000
# monitor value =10011
# monitor value =01001
# monitor value =00100
# monitor value =00010
# monitor value =00001
# monitor value =00000
# monitor value =01100
# monitor value =00110
# monitor value =00011
# monitor value =00001
# monitor value =00000
# ** Note: $finish    : C:/questasim64_10.7c/examples/shifter_tb.v(138)
#    Time: 328000428 ns  Iteration: 0  Instance: /shifter_tb
# 1
# Break in Module shifter_tb at C:/questasim64_10.7c/examples/shifter_tb.v line 138

VSIM 4>
```

Figure 12 monitor of shifter

However, to make display visible we just decreased the loop for the testing only so it can be easier to detect but the main code will be one million

```
# Test case 2
# monitor value =00000
# Test case 3
# Test case 4
# Test case 5
# Test case 6
# monitor value =11110
# monitor value =11100
# monitor value =11000
```

Figure 13 monitor and display shifter

3.3 MOORE test bench

```
module FSM_tb ();
reg  X;
reg  Y;
reg  rst;
reg  clk;

wire unlock;


FSM DUT (
        .X(X),
        .Y(Y),
        .rst(rst),
        .clk(clk),
        .unlock(unlock)
    );

integer seed1=10;
integer seed2=20;
integer seed3=30;
integer i=0;


initial
    begin
    clk = 0;
    forever begin
    #5;
    clk = ~clk;

    end
end


initial
    begin
    $dumpfile("FSM.vcd");
    $dumpvars ;

//initial values
X = 0;
Y = 0;
rst = 0;
clk = 1;


    #40;
#2  X = 1'b0;
#2  Y = 1'b1;
$monitor("monitor value =%b" , unlock);
$display("Test case 1");
#40;

#2  X = 1'b0;
```

```
rst = 0;
clk = 1;


    #40;
#2  X = 1'b0;
#2  Y = 1'b1;
$monitor("monitor value =%b" , unlock);
$display("Test case 1");
#40;

#2  X = 1'b0;
#2  Y = 1'b1;
$display("Test case 2");
#40;

#2  X = 1'b0;
#2  Y = 1'b0;
$display("Test case 3");
#40;

#2  X = 1'b0;
#2  Y = 1'b0;
$display("Test case 4");
#40;

#2  X = 1'b1;
#2  Y = 1'b1;
$display("Test case 5");
#40;

#2  X = 1'b0;
#2  Y = 1'b0;
$display("Test case 6");
#40;


rst=0;
rst=1;
for(i=0;i<1000000;i=i+1)
begin
#40
#2  X = $random(seed3);
#2  Y = $random(seed3);
#40;

#2  X = $random(seed1);
#2  Y = $random(seed1);
#40;

#2  X = $random(seed2);
#2  Y = $random(seed2);
#40;
```

```
rst=1;
for(i=0;i<1000000;i=i+1)
begin
#40
#2  X = $random(seed3);
#2  Y = $random(seed3);
#40;

#2  X = $random(seed1);
#2  Y = $random(seed1);
#40;

#2  X = $random(seed2);
#2  Y = $random(seed2);
#40;

#2  X = $random(seed1);
#2  Y = $random(seed1);
#40;

#2  X = $random(seed1);
#2  Y = $random(seed1);
#40;

#2  X = $random(seed1);
#2  Y = $random(seed1);
#40;

end

rst = 0;
    #100;
    $finish();
    end


endmodule
```

Figure 14 Moore test bench

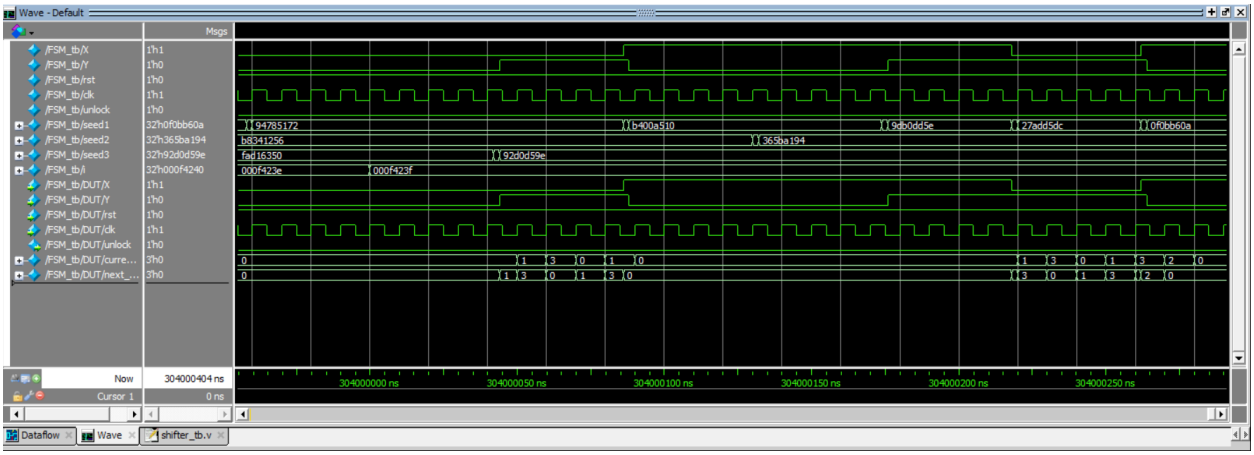

Figure 15 Moore wavelength

As monitor changes was low here, we didn't need to test on different loop values

```
VSIM 5> run -all
# Test case 1
# monitor value =0
# Test case 2
# Test case 3
# Test case 4
```

Figure 16 monitor and display Moore

4 Coverage

4.1 And coverage report

```
Coverage Report Summary Data by file

================================================================================
=== File: and.v
================================================================================
    Enabled Coverage             Active      Hits    Misses % Covered
    ----------------             ------      ----    ------ ---------
    Stmts                             1         1         0    100.00
    FEC Expression Terms              2         2         0    100.00
    Toggle Bins                       6         6         0    100.00

================================================================================
=== File: and_gate_tb.v
================================================================================
    Enabled Coverage             Active      Hits    Misses % Covered
    ----------------             ------      ----    ------ ---------
    Stmts                            81        81         0    100.00
    Toggle Bins                     262       237        25     90.45

Total Coverage By File (code coverage only, filtered view): 96.89%
```

Figure 17 and coverage report

4.2 Shifter coverage report

```
Coverage Report Summary Data by file

================================================================================
=== File: shifter.v
================================================================================
    Enabled Coverage             Active      Hits    Misses % Covered
    ----------------             ------      ----    ------ ---------
    Stmts                             6         6         0    100.00
    Branches                          4         4         0    100.00
    Toggle Bins                      38        38         0    100.00

================================================================================
=== File: shifter_tb.v
================================================================================
    Enabled Coverage             Active      Hits    Misses % Covered
    ----------------             ------      ----    ------ ---------
    Stmts                           136       136         0    100.00
    Toggle Bins                     284       259        25     91.19

Total Coverage By File (code coverage only, filtered view): 97.41%
```

Figure 18 shifter coverage report

4.3 Moore coverage report

```
Coverage Report Summary Data by file

================================================================================
=== File: FSM.v
================================================================================
    Enabled Coverage             Active      Hits    Misses % Covered
    ----------------             ------      ----    ------ ---------
    Stmts                            29        20         9     68.96
    Branches                         31        21        10     67.74
    FSMs                                                      63.33
        States                        6         4         2     66.66
        Transitions                  10         6         4     60.00
    Toggle Bins                      22        16         6     72.72

================================================================================
=== File: FSM_tb.v
================================================================================
    Enabled Coverage             Active      Hits    Misses % Covered
    ----------------             ------      ----    ------ ---------
    Stmts                            87        87         0    100.00
    Toggle Bins                     266       239        27     89.84

Total Coverage By File (code coverage only, filtered view): 77.96%
```

Figure 19 Moore coverage report