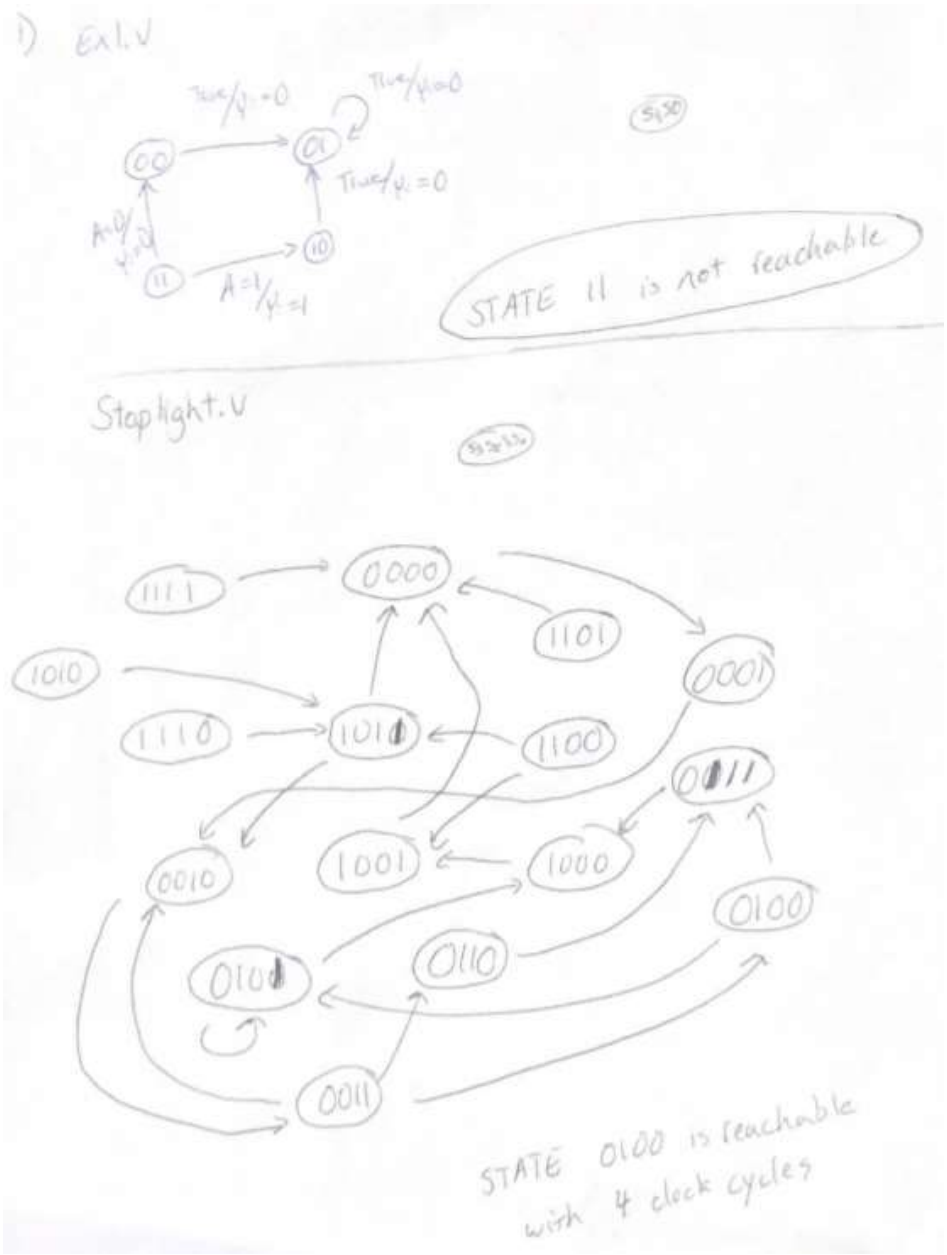
Alan Chen

ECE597 Lab 1

2/21/2020

Part A

1. Draw legibly the state transition graph for ex1.v and stoplight1.v. Include all states, even if unreachable. You can use any method that you prefer for figuring out which state transitions are possible (pen and paper, simulation, etc), but you must include all transitions.



2. Using these two state transition graphs, indicate whether the target states are reachable. If so, what is the minimum number of execution steps (clock cycles) needed to get from the initial state to the target state?

For Ex1.v the target state 11 is not reachable

For Stoplight1.v the target state 0100 is reachable with 4 clock cycles

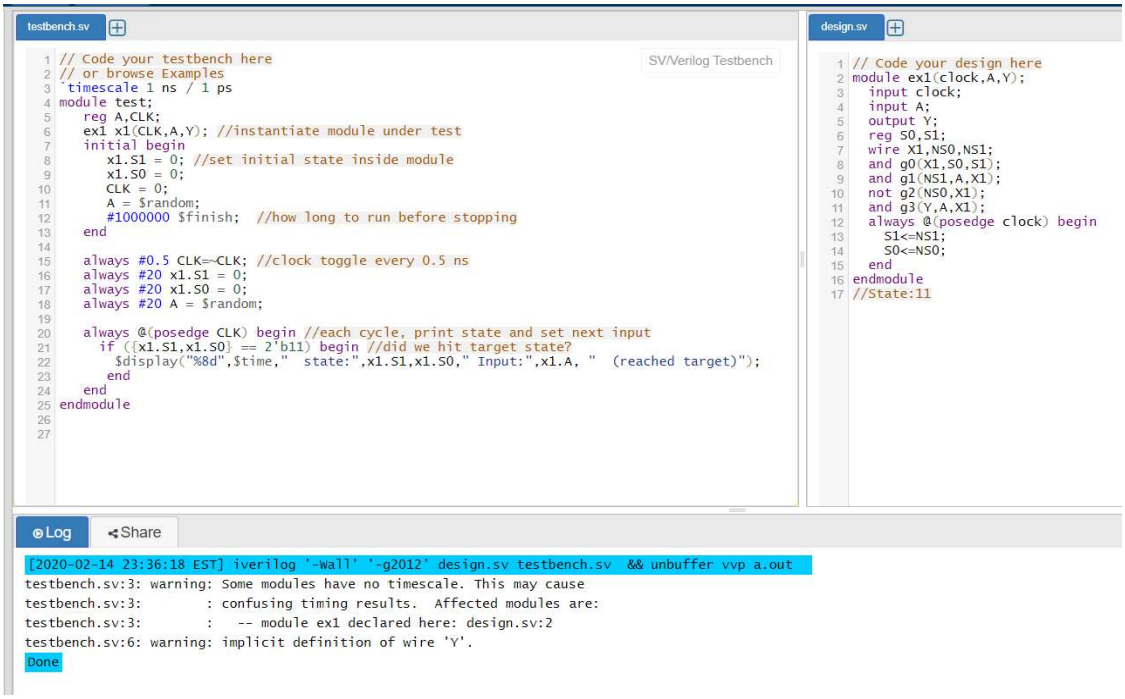
3. If you are given an arbitrary transition relation that maps states to sets of states, describe in pseudo-code the most efficient algorithm you can think of to check whether a target state is reachable from initial state. Your algorithm must terminate upon (1) reaching the target, or (2) finding that the target can never be reached.

Pseudo-Code

```
Given:   State 1: {next state 1, … }, State 3: {next state 1, ... }, …

# General Depth First Search

arrVisited = []

myStack = Stack()

myStack.push (initial State)

while myStack not Empty:

        currentState = myStack.pop()

        arrVisited.append (currentState)

        for transitions in currentState:

                if transition == finalState:

                        return True #reachable

                if transition not in arrVisited:

                        myStack.push (transition)

return False      # can never be reached
```

4. For the benchmarks given (ex1, ex2, ex3, ex4, stoplight1, stoplight2), write a Verilog testbench to initialize the state to 0, apply random inputs in each cycle, and notify you if/when it reaches the target state. If the target state is not reached within 1M cycles, the execution can terminate, and the result can be reported as "timed out". If the target state is reached, report how many cycles of random simulation were performed before reaching it. Include your testbench code with the report.

Ex1.v (the target state is not reachable)



testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1 ns / 1 ps
module test;
    reg A,CLK;
    ex1 x1(CLK,A,Y); //instantiate module under test
    initial begin
       x1.S1 = 0; //set initial state inside module
       x1.S0 = 0;
       CLK = 0;
       A = $random;
       #1000000 $finish;  //how long to run before stopping
    end

    always #0.5 CLK=~CLK; //clock toggle every 0.5 ns
    always #20 x1.S1 = 0;
    always #20 x1.S0 = 0;
    always #20 A = $random;

    always @(posedge CLK) begin //each cycle, print state and set next input
      if ({x1.S1,x1.S0} == 2'b11) begin //did we hit target state?
        $display("%8d",$time,"  state:",x1.S1,x1.S0," Input:",x1.A, "  (reached target)");
       end
    end
endmodule
```

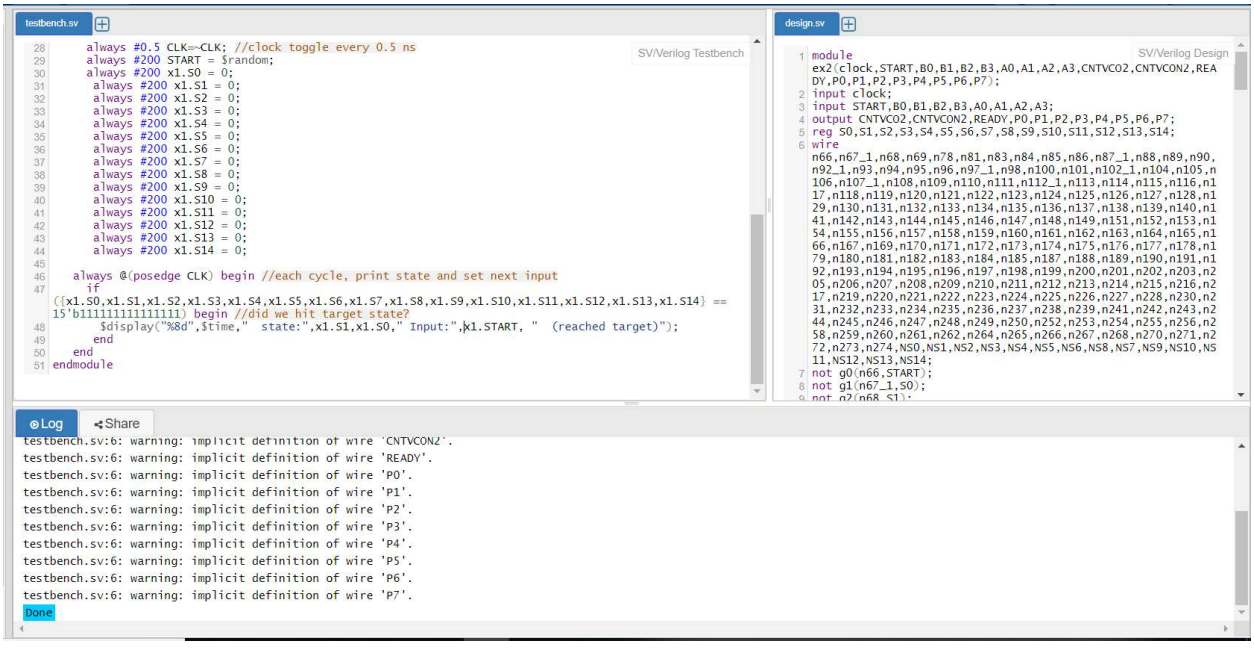
design.sv

```
// Code your design here
module ex1(clock,A,Y);
  input clock;
  input A;
  output Y;
  reg S0,S1;
  wire X1,NS0,NS1;
  and g0(X1,S0,S1);
  and g1(NS1,A,X1);
  not g2(NS0,X1);
  and g3(Y,A,X1);
  always @(posedge clock) begin
    S1<=NS1;
    S0<=NS0;
  end
endmodule
//State:11
```

Log / Share

```
[2020-02-14 23:36:18 EST] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:3: warning: Some modules have no timescale. This may cause
testbench.sv:3:        : confusing timing results.  Affected modules are:
testbench.sv:3:        :    -- module ex1 declared here: design.sv:2
testbench.sv:6: warning: implicit definition of wire 'Y'.
Done
```

Ex2.v (the target state is not reachable)



testbench.sv

```
    always #0.5 CLK=~CLK; //clock toggle every 0.5 ns
    always #200 START = $random;
    always #200 x1.S0 = 0;
     always #200 x1.S1 = 0;
     always #200 x1.S2 = 0;
     always #200 x1.S3 = 0;
     always #200 x1.S4 = 0;
     always #200 x1.S5 = 0;
     always #200 x1.S6 = 0;
     always #200 x1.S7 = 0;
     always #200 x1.S8 = 0;
     always #200 x1.S9 = 0;
     always #200 x1.S10 = 0;
     always #200 x1.S11 = 0;
     always #200 x1.S12 = 0;
     always #200 x1.S13 = 0;
     always #200 x1.S14 = 0;

   always @(posedge CLK) begin //each cycle, print state and set next input
      if
({x1.S0,x1.S1,x1.S2,x1.S3,x1.S4,x1.S5,x1.S6,x1.S7,x1.S8,x1.S9,x1.S10,x1.S11,x1.S12,x1.S13,x1.S14} ==
15'b111111111111111) begin //did we hit target state?
        $display("%8d",$time,"  state:",x1.S1,x1.S0," Input:",x1.START, "  (reached target)");
       end
   end
endmodule
```

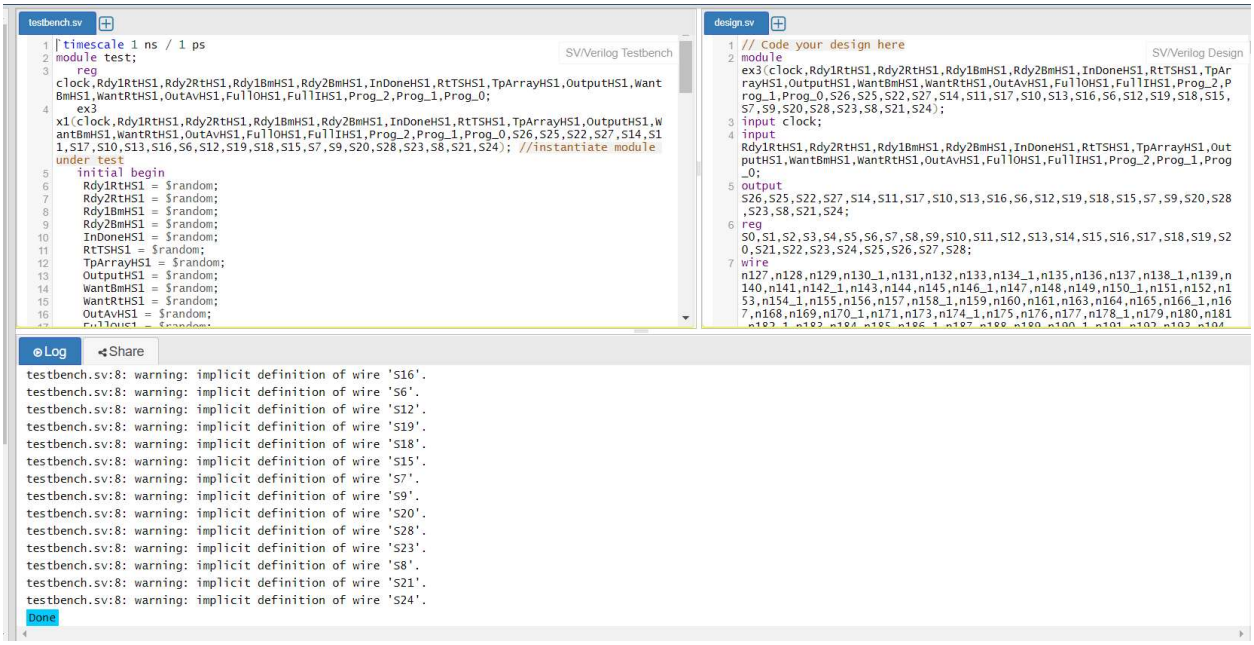
design.sv

```
module
ex2(clock,START,B0,B1,B2,B3,A0,A1,A2,A3,CNTVCO2,CNTVCON2,REA
DY,P0,P1,P2,P3,P4,P5,P6,P7);
input clock;
input START,B0,B1,B2,B3,A0,A1,A2,A3;
output CNTVCO2,CNTVCON2,READY,P0,P1,P2,P3,P4,P5,P6,P7;
reg S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14;
wire
n66,n67_1,n68,n69,n78,n81,n83,n84,n85,n86,n87_1,n88,n89,n90,
n92_1,n93,n94,n95,n96,n97_1,n98,n100,n101,n102_1,n104,n105,n
106,n107_1,n108,n109,n110,n111,n112_1,n113,n114,n115,n116,n1
17,n118,n119,n120,n121,n122,n123,n124,n125,n126,n127,n128,n1
29,n130,n131,n132,n133,n134,n135,n136,n137,n138,n139,n140,n1
41,n142,n143,n144,n145,n146,n147,n148,n149,n151,n152,n153,n1
54,n155,n156,n157,n158,n159,n160,n161,n162,n163,n164,n165,n1
66,n167,n169,n170,n171,n172,n173,n174,n175,n176,n177,n178,n1
79,n180,n181,n182,n183,n184,n185,n187,n188,n189,n190,n191,n1
92,n193,n194,n195,n196,n197,n198,n199,n200,n201,n202,n203,n2
05,n206,n207,n208,n209,n210,n211,n212,n213,n214,n215,n216,n2
17,n219,n220,n221,n222,n223,n224,n225,n226,n227,n228,n230,n2
31,n232,n233,n234,n235,n236,n237,n238,n239,n241,n242,n243,n2
44,n245,n246,n247,n248,n249,n250,n252,n253,n254,n255,n256,n2
58,n259,n260,n261,n262,n264,n265,n266,n267,n268,n270,n271,n2
72,n273,n274,NS0,NS1,NS2,NS3,NS4,NS5,NS6,NS8,NS7,NS9,NS10,NS
11,NS12,NS13,NS14;
not g0(n66,START);
not g1(n67_1,S0);
not g2(n68,S1);
```

Log / Share

```
testbench.sv:6: warning: implicit definition of wire 'CNTVCON2'.
testbench.sv:6: warning: implicit definition of wire 'READY'.
testbench.sv:6: warning: implicit definition of wire 'P0'.
testbench.sv:6: warning: implicit definition of wire 'P1'.
testbench.sv:6: warning: implicit definition of wire 'P2'.
testbench.sv:6: warning: implicit definition of wire 'P3'.
testbench.sv:6: warning: implicit definition of wire 'P4'.
testbench.sv:6: warning: implicit definition of wire 'P5'.
testbench.sv:6: warning: implicit definition of wire 'P6'.
testbench.sv:6: warning: implicit definition of wire 'P7'.
Done
```

Ex3.v (the target state is not reachable)



testbench.sv

```
`timescale 1 ns / 1 ps
module test;
   reg clock,Rdy1RtHS1,Rdy2RtHS1,Rdy1BmHS1,Rdy2BmHS1,InDoneHS1,RtTSHS1,TpArrayHS1,OutputHS1,WantBmHS1,WantRtHS1,OutAvHS1,FullOHS1,FullIHS1,Prog_2,Prog_1,Prog_0;
   ex3 x1(clock,Rdy1RtHS1,Rdy2RtHS1,Rdy1BmHS1,Rdy2BmHS1,InDoneHS1,RtTSHS1,TpArrayHS1,OutputHS1,WantBmHS1,WantRtHS1,OutAvHS1,FullOHS1,FullIHS1,Prog_2,Prog_1,Prog_0,S26,S25,S22,S27,S14,S11,S17,S10,S13,S16,S6,S12,S19,S18,S15,S7,S9,S20,S28,S23,S8,S21,S24); //instantiate module under test
   initial begin
    Rdy1RtHS1 = $random;
    Rdy2RtHS1 = $random;
    Rdy1BmHS1 = $random;
    Rdy2BmHS1 = $random;
    InDoneHS1 = $random;
    RtTSHS1 = $random;
    TpArrayHS1 = $random;
    OutputHS1 = $random;
    WantBmHS1 = $random;
    WantRtHS1 = $random;
    OutAvHS1 = $random;
```

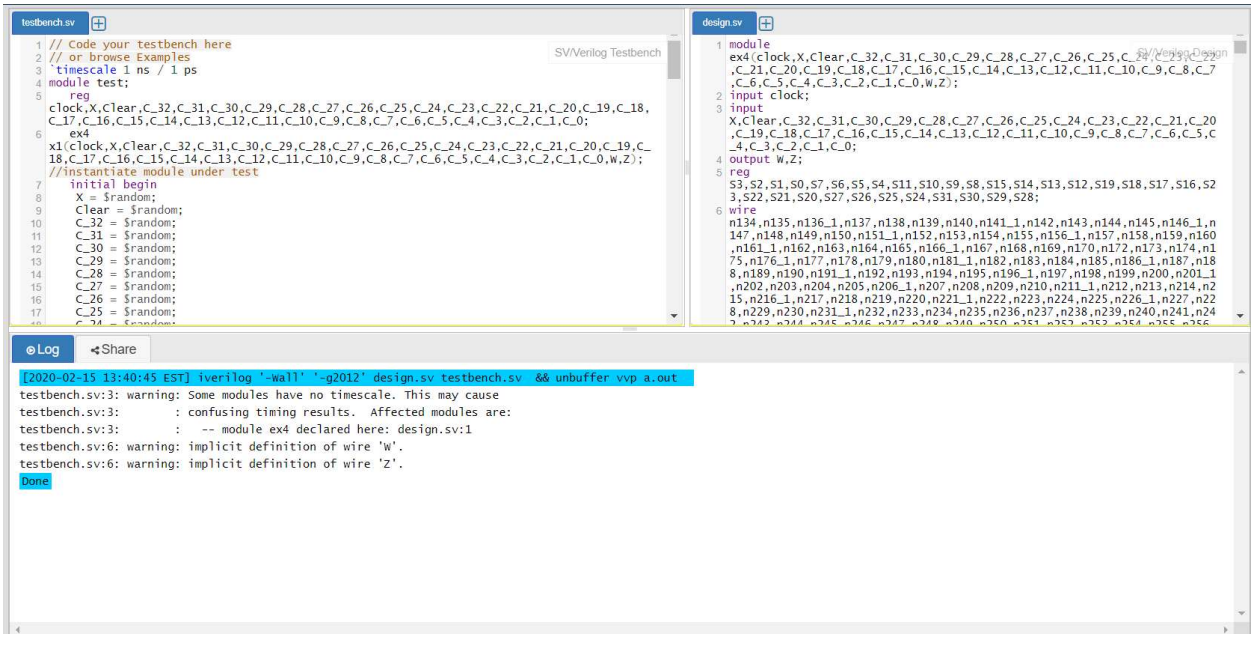
design.sv

```
// Code your design here
module ex3(clock,Rdy1RtHS1,Rdy2RtHS1,Rdy1BmHS1,Rdy2BmHS1,InDoneHS1,RtTSHS1,TpArrayHS1,OutputHS1,WantBmHS1,WantRtHS1,OutAvHS1,FullOHS1,FullIHS1,Prog_2,Prog_1,Prog_0,S26,S25,S22,S27,S14,S11,S17,S10,S13,S16,S6,S12,S19,S18,S15,S7,S9,S20,S28,S23,S8,S21,S24);
input clock;
input Rdy1RtHS1,Rdy2RtHS1,Rdy1BmHS1,Rdy2BmHS1,InDoneHS1,RtTSHS1,TpArrayHS1,OutputHS1,WantBmHS1,WantRtHS1,OutAvHS1,FullOHS1,FullIHS1,Prog_2,Prog_1,Prog_0;
output S26,S25,S22,S27,S14,S11,S17,S10,S13,S16,S6,S12,S19,S18,S15,S7,S9,S20,S28,S23,S8,S21,S24;
reg S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16,S17,S18,S19,S20,S21,S22,S23,S24,S25,S26,S27,S28;
wire n127,n128,n129,n130_1,n131,n132,n133,n134_1,n135,n136,n137,n138_1,n139,n140,n141,n142_1,n143,n144,n145,n146_1,n147,n148,n149,n150_1,n151,n152,n153,n154_1,n155,n156,n157,n158_1,n159,n160,n161,n163,n164,n165,n166_1,n167,n168,n169,n170_1,n171,n173,n174_1,n175,n176,n177,n178_1,n179,n180,n181
```

Log | Share

```
testbench.sv:8: warning: implicit definition of wire 'S16'.
testbench.sv:8: warning: implicit definition of wire 'S6'.
testbench.sv:8: warning: implicit definition of wire 'S12'.
testbench.sv:8: warning: implicit definition of wire 'S19'.
testbench.sv:8: warning: implicit definition of wire 'S18'.
testbench.sv:8: warning: implicit definition of wire 'S15'.
testbench.sv:8: warning: implicit definition of wire 'S7'.
testbench.sv:8: warning: implicit definition of wire 'S9'.
testbench.sv:8: warning: implicit definition of wire 'S20'.
testbench.sv:8: warning: implicit definition of wire 'S28'.
testbench.sv:8: warning: implicit definition of wire 'S23'.
testbench.sv:8: warning: implicit definition of wire 'S8'.
testbench.sv:8: warning: implicit definition of wire 'S21'.
testbench.sv:8: warning: implicit definition of wire 'S24'.
Done
```

Ex4.v (the target state is not reachable)



testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1 ns / 1 ps
module test;
   reg clock,X,Clear,C_32,C_31,C_30,C_29,C_28,C_27,C_26,C_25,C_24,C_23,C_22,C_21,C_20,C_19,C_18,C_17,C_16,C_15,C_14,C_13,C_12,C_11,C_10,C_9,C_8,C_7,C_6,C_5,C_4,C_3,C_2,C_1,C_0;
   ex4 x1(clock,X,Clear,C_32,C_31,C_30,C_29,C_28,C_27,C_26,C_25,C_24,C_23,C_22,C_21,C_20,C_19,C_18,C_17,C_16,C_15,C_14,C_13,C_12,C_11,C_10,C_9,C_8,C_7,C_6,C_5,C_4,C_3,C_2,C_1,C_0,W,Z);
//instantiate module under test
   initial begin
    X = $random;
    Clear = $random;
    C_32 = $random;
    C_31 = $random;
    C_30 = $random;
    C_29 = $random;
    C_28 = $random;
    C_27 = $random;
    C_26 = $random;
    C_25 = $random;
```

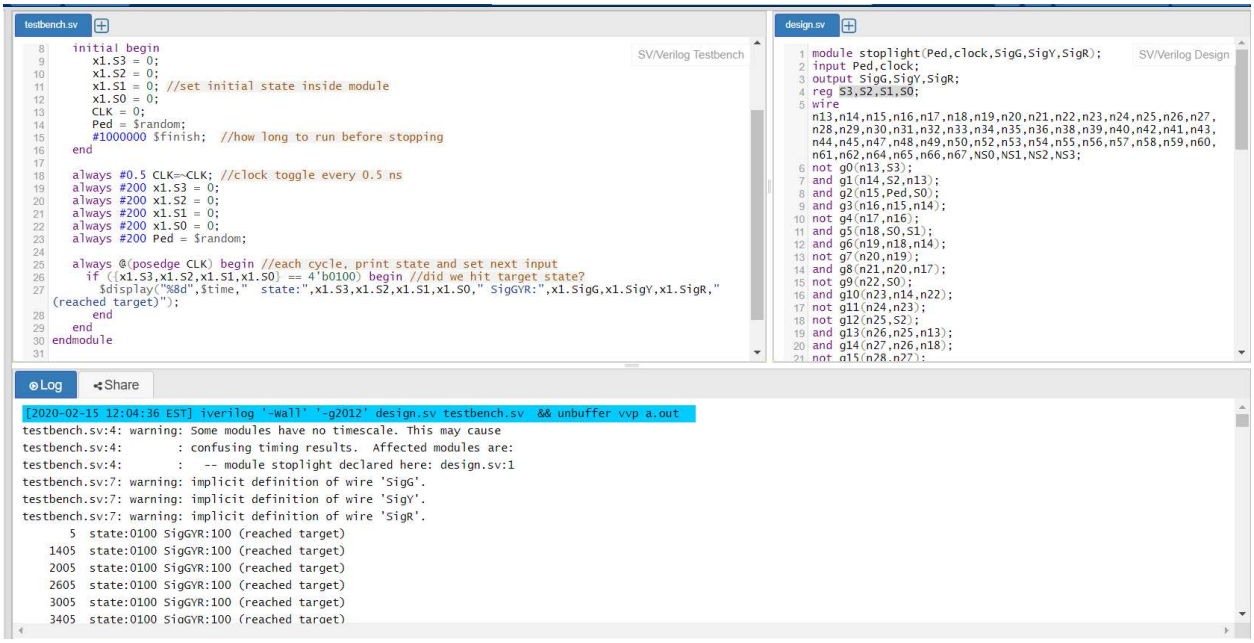
design.sv

```
module ex4(clock,X,Clear,C_32,C_31,C_30,C_29,C_28,C_27,C_26,C_25,C_24,C_23,C_22,C_21,C_20,C_19,C_18,C_17,C_16,C_15,C_14,C_13,C_12,C_11,C_10,C_9,C_8,C_7,C_6,C_5,C_4,C_3,C_2,C_1,C_0,W,Z);
input clock;
input X,Clear,C_32,C_31,C_30,C_29,C_28,C_27,C_26,C_25,C_24,C_23,C_22,C_21,C_20,C_19,C_18,C_17,C_16,C_15,C_14,C_13,C_12,C_11,C_10,C_9,C_8,C_7,C_6,C_5,C_4,C_3,C_2,C_1,C_0;
output W,Z;
reg S3,S2,S1,S0,S7,S6,S5,S4,S11,S10,S9,S8,S15,S14,S13,S12,S19,S18,S17,S16,S23,S22,S21,S20,S27,S26,S25,S24,S31,S30,S29,S28;
wire n134,n135,n136_1,n137,n138,n139,n140,n141_1,n142,n143,n144,n145,n146_1,n147,n148,n149,n150,n151_1,n152,n153,n154,n155,n156_1,n157,n158,n159,n160,n161_1,n162,n163,n164,n165,n166_1,n167,n168,n169,n170,n172,n173,n174,n175,n176_1,n177,n178,n179,n180,n181_1,n182,n183,n184,n185,n186_1,n187,n188,n189,n190,n191_1,n192,n193,n194,n195,n196_1,n197,n198,n199,n200,n201_1,n202,n203,n204,n205,n206_1,n207,n208,n209,n210,n211_1,n212,n213,n214,n215,n216_1,n217,n218,n219,n220,n221_1,n222,n223,n224,n225,n226_1,n227,n228,n229,n230,n231_1,n232,n233,n234,n235,n236,n237,n238,n239,n240,n241,n242,n243,n244,n245,n246,n247,n248,n249,n250,n251,n252,n253,n254,n255,n256
```

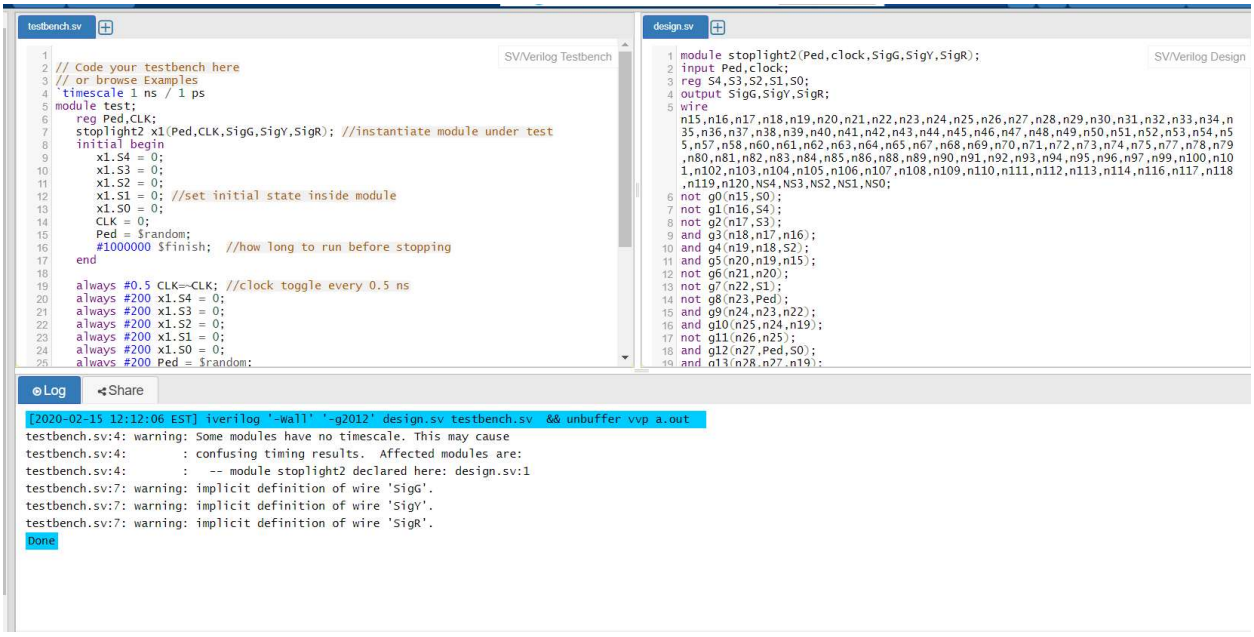
Log | Share

```
[2020-02-15 13:40:45 EST] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:3: warning: Some modules have no timescale. This may cause
testbench.sv:3:        : confusing timing results.  Affected modules are:
testbench.sv:3:        :    -- module ex4 declared here: design.sv:1
testbench.sv:6: warning: implicit definition of wire 'W'.
testbench.sv:6: warning: implicit definition of wire 'Z'.
Done
```

Stoplight1.v (the target state is reachable with about 400 cycles of simulation)



Stoplight2.v (the target state is not reachable)



5. Are your results from part A.4 consistent with your results from A.2? Describe this. If the results are consistent, give a precise explanation of what results would show them to be incompatible with each other.

The results are consistent. An example of an inconsistent result would be if the testbench of Ex.1 saying that state 11 is reachable.

Part B

1. Use your program to generate the CNF files for ex1 with the transition relation unrolled twice. Be sure to add the initial state of 0 and be sure to enforce that the next state of each transition relation is equal to the starting state of the next transition relation. Use the SAT solver to check whether state 11 is reachable as the 2nd state after the initial one. Is your finding consistent with the state transition graph from question A.1? Explain your answer. Now that you've checked whether "11" is reachable in 2 cycles, modify the example and repeat, to check whether the other 3 states (00,01,10) are reachable in 2 cycles, and again explain whether your findings are consistent with question A.1. Be sure to fix any bugs here before trying the larger examples that follow!

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex1.v 2

THIS IS THE 1  UNROLL RESULTS:
FINAL STATE:  11
s UNSATISFIABLE

THIS IS THE 2  UNROLL RESULTS:
FINAL STATE:  11
s UNSATISFIABLE
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex1.v 2

THIS IS THE 1  UNROLL RESULTS:
FINAL STATE:  00
s UNSATISFIABLE

THIS IS THE 2  UNROLL RESULTS:
FINAL STATE:  00
s UNSATISFIABLE
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex1.v 2

THIS IS THE 1  UNROLL RESULTS:
FINAL STATE:  01
s SATISFIABLE
v -1 -2 -3 -4 5 6 -7 0

THIS IS THE 2  UNROLL RESULTS:
FINAL STATE:  01
s SATISFIABLE
v -1 -2 3 -4 -5 -6 -7 8 -9 10 11 12 -13 -14 0
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex1.v 2

THIS IS THE 1  UNROLL RESULTS:
FINAL STATE:  10
s UNSATISFIABLE

THIS IS THE 2  UNROLL RESULTS:
FINAL STATE:  10
s UNSATISFIABLE
```

State 11 is not reachable as the 2nd state which is consistent with the state transition graph from question A.1. State 11 is not the initial state and cannot be transitioned into by any state so therefore it is not reachable at all.

11 -> unsatisfiable

00 -> unsatisfiable

01 -> satisfiable

10 -> unsatisfiable

2. For each of the benchmarks, perform symbolic reachability analysis, with 10 unrollings. Note that you are checking whether the target state is reachable in exactly the 10th state after the initial one. In other words, you are checking whether the next state in the 10th unrolling can be the target state. Describe how you've implemented your program and any data structures used. For each design, indicate whether or not the target state is reachable as the 10th state. How long does the SAT solver take to perform its search for a satisfying assignment?

For my program, I parsed the user inputs (Verilog file and number of unrolling). Then I parsed through the given file name (Verilog file) for different types of gates, registers, transitions, and final state. For every unrolling, I duplicate the gates and registers, and assigning appropriate transitions to them. The main data structure I used, is a dictionary to keep track of the registers to the number representation for the dimacs file. All designs are unsatisfiable at exactly the 10th unrolling. The time for the SAT solver to run ranges from .02 to .69 seconds. The specific timing to each file is listed below.

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex1.v 10

THIS IS THE 10  UNROLL RESULTS:
TESTING FOR FINAL STATE:  11
s UNSATISFIABLE
My program took 0.021463632583618164 seconds to run
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex2.v 10

THIS IS THE 10  UNROLL RESULTS:
TESTING FOR FINAL STATE:  111111111111111
s UNSATISFIABLE
My program took 0.050106048583984375 seconds to run
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex3.v 10

THIS IS THE 10  UNROLL RESULTS:
TESTING FOR FINAL STATE:  000000000101101100000011000001
s SATISFIABLE
v -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20 -21
v -22 -23 -24 -25 -26 -27 -28 -29 -30 -31 32 33 -34 35 -36 -37 -38 39 -40 -41
v -42 -43 -44 45 -46 -47 48 49 -50 -51 -52 -53 -54 -55 -56 -57 -58 -59 -60 61
v 5998 5999 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 -6011 -6012
v -6013 -6014 -6015 -6016 -6017 -6018 -6019 -6020 0
My program took 0.6857435703277588 seconds to run
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/ex4.v 10

THIS IS THE 10  UNROLL RESULTS:
TESTING FOR FINAL STATE:  00000000000000000000000010000000
s UNSATISFIABLE
My program took 0.11552214622497559 seconds to run
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/stoplight1.v 10

THIS IS THE 10  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE
My program took 0.030384302139282227 seconds to run
```

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py verilogFiles/stoplight2.v 10

THIS IS THE 10  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE
My program took 0.048485755920410156 seconds to run
```

3. Compare your results using SAT-based reachability to the results from random simulation. Explain any differences in results and what causes them. Are there any limitations in SAT-based reachability due to only checking whether a state is reachable in the last unrolling and not intermediate ones? How might you improve the symbolic search to avoid this limitation?

There are some cases where the SAT-based reachability results differ from the Verilog testbench results like in ex3 Verilog file. This is because the SAT is an exhaustive test whereas my Verilog testbenches use random inputs. There may be times when we get unlucky with the random generator especially if it requires lot of random input (looking for a specific input out of all combinations). The limitation of a SAT solver depends on how you write the your DIMACS file. If there is a problem with only checking whether a state is reachable in the last unrolling and not intermediate ones, then you can write your DIMACS file to be NS0_1 or NS0_2 or NS0_3 depending on what you are search for. Another way is to check your state at unrolling 1, unrolling 2, …, up to the last unrolling.

4. Submit the source code of your program as a separate file with your report and include instructions for how to run it for testing.

Should be attached, if not :

https://github.com/achen173/Modeling_Embedding_Systems

5. For the benchmark stoplight1.v, check whether the target state is reachable as the ith state after the initial state for i=1,2,…,32. This should be done by invoking your program (and the SAT solver) 32 times while changing the input argument that specifies the number of unrollings. You can do this using a script. Report the values of i for which the target state is reachable. How does this compare with your results from A.2?

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py -c verilogFiles/stoplight1.v 32

THIS IS THE 1  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE

THIS IS THE 2  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE

THIS IS THE 3  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE

THIS IS THE 4  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s SATISFIABLE
v -1 -2 -3 -4 -5 -6 -7 8 -9 -10 11 -12 -13 -14 15 16 17 18 19 20 -21 -22 -23
v -24 -25 26 -27 -28 -29 30 -31 -32 -33 -34 -35 -36 37 38 39 40 -41 -42 -43 44
v -45 -46 -47 -48 49 50 51 52 53 54 55 56 57 -58 59 -60 -61 -62 -63 -64 65 66
v 67 68 69 70 71 72 73 74 75 76 -77 -78 -79 80 81 82 83 -84 85 86 -87 -88 -89
v 90 -91 92 -93 94 -95 -96 -97 -98 -99 -100 101 102 103 104 105 106 107 -108
v 109 110 111 -112 113 114 115 -116 -117 -118 -119 120 -121 -122 -123 -124 -125
v -126 -127 -128 -129 130 -131 -132 -133 134 -135 -136 137 -138 139 140 141
v -142 143 -144 -145 146 -147 148 -149 -150 -151 152 153 154 155 -156 -157 -158
v -159 -160 161 162 163 164 165 166 167 168 -169 -170 -171 -172 -173 -174 -175
v 176 -177 -178 179 -180 181 182 -183 184 185 -186 -187 188 -189 -190 191 192
v -193 -194 195 -196 197 198 -199 200 -201 -202 -203 -204 -205 -206 -207 -208
v 209 210 211 212 213 214 -215 216 217 -218 -219 220 -221 222 223 -224 225 226
v -227 228 229 230 -231 232 -233 234 -235 236 -237 238 -239 240 241 -242 243
v -244 0

THIS IS THE 5  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE

THIS IS THE 6  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE

THIS IS THE 7  UNROLL RESULTS:
TESTING FOR FINAL STATE:  0100
s UNSATISFIABLE
```

i-Values = 4, 12 – 32

My answer is consistent with my results from A.2. It is reachable at exactly 4 clock cycles (minimum). However, it is also reachable with anything above 12 clock cycle.

6. Repeat previous question using benchmark stoplight2.v. How does this compare to your finding from random execution of stoplight2.v in A.4?

```
alanchen@DESKTOP-TM62I39:/mnt/c/Users/Alan Chen/Desktop/ECE597/Lab 1/PythonFiles/Part2$ python3.6 main.py -c verilogFiles/stoplight2.v 32

THIS IS THE 1  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 2  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 3  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 4  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 5  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 6  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 7  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 8  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE

THIS IS THE 9  UNROLL RESULTS:
TESTING FOR FINAL STATE:  01000
s UNSATISFIABLE
```

i-Values = 15 – 32

My results are inconsistent with the random execution of stoplight2.v from A.4 because the testbench use random input. With the # of unrolling to succeed (15-32), the odds for a correct random input decrease. Therefore, it is expected to be inconsistent.

7. Given the number of state bits in the stoplight designs, is it correct to conclude that any states not reachable within 32 unrollings will remain unreachable with more unrollings? Justify your answer.

Yes, because there are in stoplight has a maximum of 5-bit state, so 2^5 = 32 possibilities. As a result, the 32 unrolling's covers all possibilities.