Ain Shams University

Faculty of Engineering

CHEP- CESS

Mohammed Ehab Elsaeed

16P8160

m.elsaeed1998@gmail.com

Electronic Design Automation

CSE 215

Digital Access Control Finite State Machine Project 2

Introduction

The purpose of this document is to illustrate the usage of Alliance tools to synthesise the digital access control finite state machine created in project one.

Table of Contents

1. Code Modification for Alliance
2. BOOM Outputs Comparison
3. BOOG Outputs Comparison
4. LOON Outputs Comparison
5. LOON Netlist of Chosen State Encoding
6. Delay Simulation After FLATBEH and PROOF.
7. SCAPIN Netlist
8. Simulation of FSM Using the Behavioural Component, and LOON’s and SCAPIN’s structural Components.
9. Test of scan-path
10. Appendix
    1. Makefile
    2. Paramfile
    3. .path file
    4. FLATBEH log
    5. PROOF log
    6. LOON’s and SCAPIN’s Structural Components and FSM’s Behavioural Component Comparison-Based-Testbench

1- Code Modification for Alliance

For Alliance tools to work, I had to make a slight modification to my FSM’s VHDL Code.

For Alliance:

                when others =>

                    assert ('1')

                    report "Invalid state";

For Symphony EDA (ModelSim Equivalent):

                when others =>

                    assert false

                    report "Invalid state"

                        severity failure;

SYF never accepted the “assert false” statement.

2- BOOM Outputs Comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Encoding | A | J | M | O | R |
| Final Literals | 92 | 95 | 78 | 100 | 90 |

3- BOOG Outputs Comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Encoding | A | J | M | O | R |
| Critical Path Delays | 2302 | 2032 | 2112 | 2508 | 2213 |
| Areas | 79500 | 84000 | 70000 | 98000 | 78750 |

4- LOON Outputs Comparison

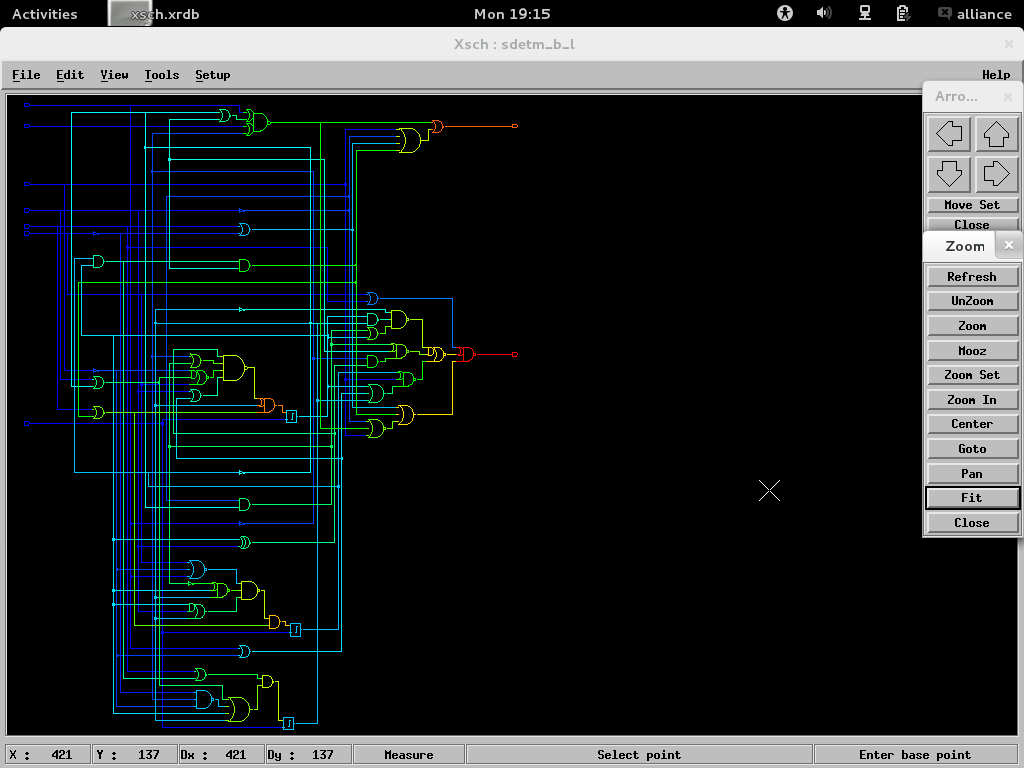
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Encoding | A | J | M | O | R |
| Critical Path Delays | 2363 | 3156 | 2584 | 3022 | 3015 |
| Areas | 80750 | 86250 | 70250 | 103000 | 82000 |

Decision: Encoding **M** was chosen.

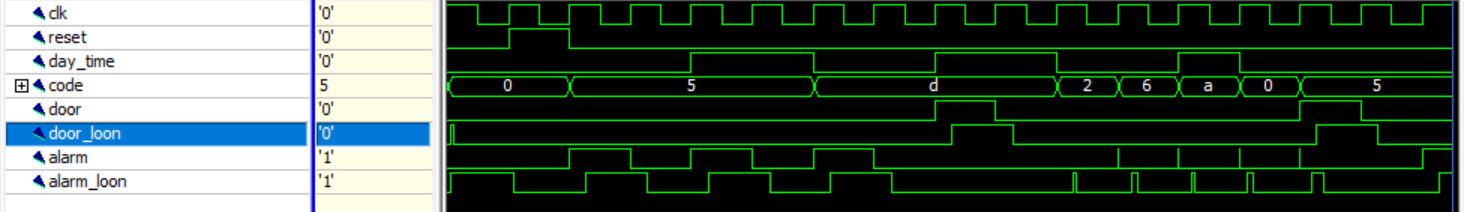
Justification: It has the lowest area and the 2nd lowest delay among other encodings.

Paramfile is in the appendix.

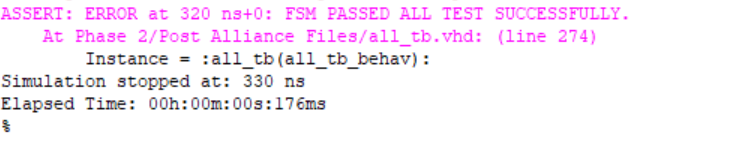
5- LOON Netlist of Chosen Encoding



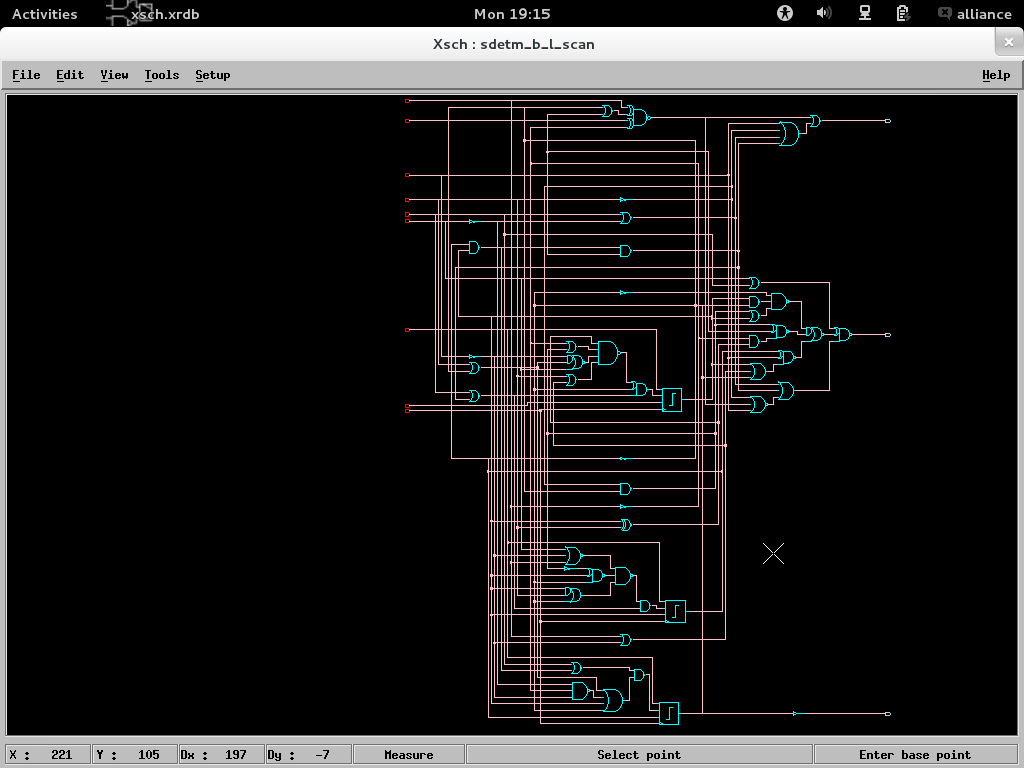
6- Delay Simulation After FLATBEH and PROOF.



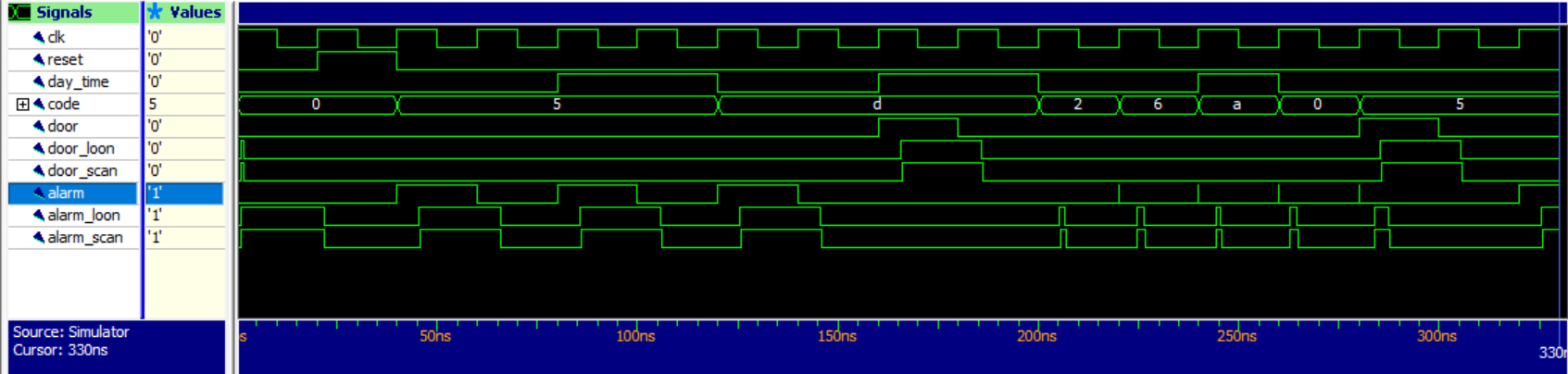
Terminal Output:

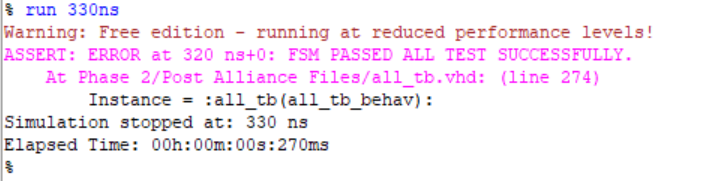


7- SCAPIN Netlist

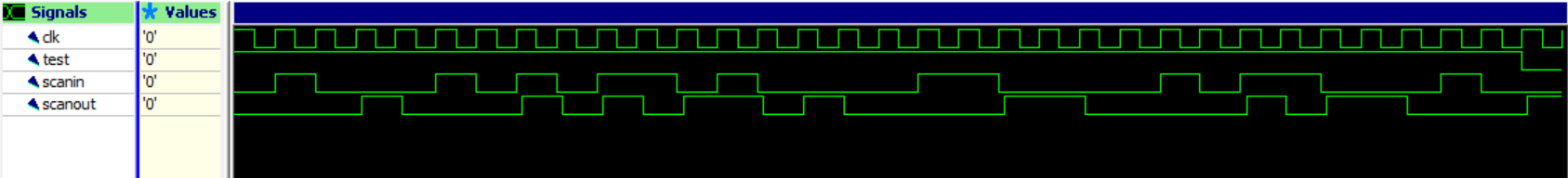


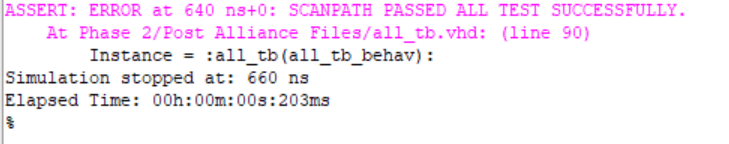
8- Simulation of FSM Using the Behavioural Component, and LOON’s and SCAPIN’s structural Components.





Comments: The Structural Components produced by SCAPIN and LOON have delays unlike the initial behavioural component. These delays, however, cause no problems as they are within half a clock period.

9- Test of scan-path



10- Appendix

## Makefile

all\_syf: sdeta.vbe \

    sdetj.vbe \

    sdetm.vbe \

    sdeto.vbe \

    sdetr.vbe

        @echo "<-- SYF DONE"

all\_boom: sdeta\_b.vbe \

    sdetj\_b.vbe \

    sdetm\_b.vbe \

    sdeto\_b.vbe \

    sdetr\_b.vbe

        @echo "<-- BOOM DONE"

sdet\_boog : sdeta\_b.vst \

    sdetj\_b.vst \

    sdetm\_b.vst \

    sdeto\_b.vst \

    sdetr\_b.vst

        @echo "<-- BOOG DONE"

sdet\_loon : sdeta\_b\_l.vst \

    sdetj\_b\_l.vst \

    sdetm\_b\_l.vst \

    sdeto\_b\_l.vst \

    sdetr\_b\_l.vst

        @echo "<-- LOON DONE"

#-------Finite State Machine Synthesis-----------------#

vhd\_to\_fsm:

    rename .vhd .fsm \*.vhd

%\_scan.vst : %.vst scan.path

    @echo " scan-path insertion -> $@ "

    scapin -VRB $\* scan $\*\_scan > scapin.out

%\_b\_l\_net.vbe : %\_b\_l.vst %.vbe

    @echo " Formal checking -> $@ "

    flatbeh $\*\_b\_l $\*\_b\_l\_net > $\*\_flatbeh.out

    proof -d $\* $\*\_b\_l\_net > $\*\_proof.out

%.vst : %.vbe paramfile.lax

    @echo " Logical Synthesis -> $@ "

    boog -x 1 -l paramfile $\* > $\*\_boog.out

%\_l.vst : %.vst paramfile.lax

    @echo " Netlist Optimization -> $@ "

    loon -x 1 $\* $\*\_l paramfile > $\*\_loon.out

%\_b.vbe: %.vbe

    @echo "     Boolean Optimization -> $@"

    boom -V -d 50 $\* $\*\_b > $\*\_boom.out

sdeta.vbe: sdet.fsm

    @echo " Encoding Synthesis -> sdeta.vbe"

    syf -CEV -a sdet

sdetj.vbe: sdet.fsm

    @echo " Encoding Synthesis -> sdetj.vbe"

    syf -CEV -j sdet

sdetm.vbe: sdet.fsm

    @echo " Encoding Synthesis -> sdetm.vbe"

    syf -CEV -m sdet

sdeto.vbe: sdet.fsm

    @echo " Encoding Synthesis -> sdeto.vbe"

    syf -CEV -o sdet

sdetr.vbe: sdet.fsm

    @echo " Encoding Synthesis -> sdetr.vbe"

    syf -CEV -r sdet

#-------Clean Up---------------------------------------#

clean :

    rm -f \*.out \*.vbe \*.enc \*~

    @echo "Erase all the files generated by the makefile"

## Paramfile

#M{2}

#L{2}

#C{

door:100;

alarm:100;

}

## .Path File

BEGIN\_PATH\_REG

dac\_current\_state\_0\_ins

dac\_current\_state\_1\_ins

dac\_current\_state\_2\_ins

END\_PATH\_REG

BEGIN\_CONNECTOR

SCAN\_IN scanin

SCAN\_OUT scanout

SCAN\_TEST test

END\_CONNECTOR

## FLATBEH Log

@@@@@@@@@ @@@@ @@@@@@@ @@@

@@ @ @@ @ @@ @@ @@

@@ @ @@ @@ @@ @@ @@

@@ @@ @@@@ @@ @@ @@ @@@@@ @@ @@@

@@ @ @@ @@ @ @@@@@@@@ @@ @@ @ @ @@@ @@

@@@@@@ @@ @@ @@ @@ @@@@@@ @@ @@ @@ @@

@@ @ @@ @@@@@ @@ @@ @@ @@@@@@@@@ @@ @@

@@ @@ @@ @@ @@ @@ @@ @@ @@ @@

@@ @@ @@ @@ @@ @@ @@ @@ @ @@ @@

@@ @@ @@ @@@ @@ @ @@ @@ @@ @@ @@ @@

@@@@@@ @@@@@@ @@@@ @@ @@@@ @@@@@@@@ @@@@ @@@@ @@@@

a netlist abstractor

Alliance CAD System 5.0 20090901, flatbeh 5.0 [2000/11/01]

Copyright (c) 1993-2019, ASIM/LIP6/UPMC

Author(s): Fran�ois DONNET, Huu Nghia VUONG

E-mail : alliance-users@asim.lip6.fr

========================= Environnement =========================

MBK\_WORK\_LIB = .

MBK\_CATA\_LIB = .:/usr/lib64/alliance/cells/sxlib:/usr/lib64/alliance/cells/dp\_sxlib:/usr/lib64/alliance/cells/rflib:/usr/lib64/alliance/cells/rf2lib:/usr/lib64/alliance/cells/ramlib:/usr/lib64/alliance/cells/romlib:/usr/lib64/alliance/cells/pxlib:/usr/lib64/alliance/cells/padlib

MBK\_CATAL\_NAME = CATAL

============================== Files ==============================

Netlist file = sdetm\_b\_l.vst

Output file = sdetm\_b\_l\_net.vbe

===================================================================

Loading './sdetm\_b\_l.vst'

flattening figure sdetm\_b\_l

loading nao2o22\_x1

loading nxr2\_x1

loading inv\_x4

loading na4\_x1

loading ao22\_x2

loading oa22\_x2

loading inv\_x2

loading sff1\_x4

loading no3\_x1

loading o3\_x2

loading o2\_x2

loading na2\_x1

loading na3\_x1

loading a2\_x2

loading noa22\_x1

loading nao22\_x1

loading o4\_x2

loading no2\_x1

Restoring array's orders

BEH : Saving 'sdetm\_b\_l\_net' in a vhdl file (vbe)

## PROOF LOG

@@@@@@@ @@@

@@ @@ @ @@

@@ @@ @@ @@

@@ @@ @@@ @@@ @@@ @@@ @@

@@ @@ @@@ @@ @@ @@ @@ @@ @@@@@@@@

@@@@@ @@ @@ @@ @@ @@ @@ @@

@@ @@ @@ @@ @@ @@ @@

@@ @@ @@ @@ @@ @@ @@

@@ @@ @@ @@ @@ @@ @@

@@ @@ @@ @@ @@ @@ @@

@@@@@@ @@@@ @@@ @@@ @@@@@@

Formal Proof

Alliance CAD System 5.0 20090901, proof 5.0

Copyright (c) 1990-2019, ASIM/LIP6/UPMC

E-mail : alliance-users@asim.lip6.fr

================================ Environment ================================

MBK\_WORK\_LIB    = .

MBK\_CATA\_LIB    = .:/usr/lib64/alliance/cells/sxlib:/usr/lib64/alliance/cells/dp\_sxlib:/usr/lib64/alliance/cells/rflib:/usr/lib64/alliance/cells/rf2lib:/usr/lib64/alliance/cells/ramlib:/usr/lib64/alliance/cells/romlib:/usr/lib64/alliance/cells/pxlib:/usr/lib64/alliance/cells/padlib

======================= Files, Options and Parameters =======================

First VHDL file   = sdetm.vbe

Second VHDL file  = sdetm\_b\_l\_net.vbe

The auxiliary signals are erased

Errors are displayed

===============================================================================

Compiling 'sdetm' ...

Compiling 'sdetm\_b\_l\_net' ...

---> final number of nodes = 343(127)

Running Abl2Bdd on `sdetm\_b\_l\_net`

--------------------------------------------------------------------------------

Formal proof with Ordered Binary Decision Diagrams between

'./sdetm' and './sdetm\_b\_l\_net'

--------------------------------------------------------------------------------

============================== PRIMARY OUTPUT ===============================

============================= AUXILIARY SIGNAL ==============================

============================== REGISTER SIGNAL ==============================

=============================== EXTERNAL BUS =================================

================================ INTERNAL BUS =================================

Formal Proof : OK

pppppppppppppppppppppppprrrrrrrrrrrrooooooooooooooooooooooooooooofffffffffffffff

--------------------------------------------------------------------------------

## LOON’s and SCAPIN’s Structural Components and FSM’s Behavioural Component Comparison-Based-Testbench

entity all\_tb is

end all\_tb;

architecture all\_tb\_behav of all\_tb is

component dac is

port (

reset : in bit;

day\_time : in bit;

code : in bit\_vector(3 downto 0);

door : out bit;

alarm : out bit;

clk : in bit;

vdd : in bit;

vss : in bit

);

end component dac;

component dac\_loon is

port (

reset : in bit;

day\_time : in bit;

code : in bit\_vector(3 downto 0);

door : out bit;

alarm : out bit;

clk : in bit;

vdd : in bit;

vss : in bit

);

end component dac\_loon;

component dac\_scan is

port (

reset : in bit;

day\_time : in bit;

code : in bit\_vector(3 downto 0);

door : out bit;

alarm : out bit;

clk : in bit;

vdd : in bit;

vss : in bit;

scanin : in bit;

test : in bit;

scanout : out bit

);

end component dac\_scan;

signal reset : bit;

signal day\_time : bit;

signal code : bit\_vector(3 downto 0);

signal door : bit;

signal alarm : bit;

signal door\_loon : bit;

signal alarm\_loon : bit;

signal door\_scan : bit;

signal alarm\_scan : bit;

signal clk : bit;

signal vdd : bit := '1';

signal vss : bit := '0';

signal scanin : bit := '0';

signal test : bit := '0';

signal scanout : bit := '0';

for all : dac use entity work.dac(dac\_behav);

for all : dac\_loon use entity work.sdetm\_b\_l(structural);

for all : dac\_scan use entity work.sdetm\_b\_l\_scan(structural);

constant clk\_period : time := 20 ns;

constant a : bit\_vector(3 downto 0) := "1010";

constant b : bit\_vector(3 downto 0) := "1011";

constant o : bit\_vector(3 downto 0) := "1101";

constant scantest : bit\_vector := "01000101011010000110000101100010";

begin

dut : dac port map(reset, day\_time, code, door, alarm, clk, vdd, vss);

dut\_loon : dac\_loon port map(reset, day\_time, code, door\_loon, alarm\_loon, clk, vdd, vss);

dut\_scan : dac\_scan port map(reset, day\_time, code, door\_scan, alarm\_scan, clk, vdd, vss, scanin, test, scanout);

process begin

test <= '1';

for i In 0 to scantest'length-1 loop

scanin <= scantest(i);

wait for clk\_period;

if i>=2 then

Assert scanout=scantest(i-2)

Report "scanout does not follow scan in"

Severity error;

end if;

end loop;

assert false

report "SCANPATH PASSED ALL TEST SUCCESSFULLY."

severity error;

test<= '0';

wait for clk\_period;

-- 1

reset <= '1';

-- day\_time<='0';

-- code <= x"";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Setting reset = 1 should reset the circuit to its initial state"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Setting reset = 1 should reset the circuit to its initial state"

severity failure;

-- 2

reset <= '0';

day\_time <= '0';

code <= x"5";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering a wrong code, should trigger the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering a wrong code, should trigger the alarm"

severity failure;

-- 3

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

-- 4

reset <= '0';

day\_time <= '1';

code <= x"5";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering a wrong code, even during daytime should trigger the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering a wrong code, even during daytime should trigger the alarm"

severity failure;

-- 5

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

-- 6

reset <= '0';

day\_time <= '0';

code <= o;

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 'O' during night time should trigger the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 'O' during night time should trigger the alarm"

severity failure;

-- 7

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

-- 8

reset <= '0';

day\_time <= '1';

code <= o;

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 'O' during daytime should open the door"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 'O' during daytime should open the door"

severity failure;

-- 9

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

-- 10

reset <= '0';

day\_time <= '0';

code <= x"2";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 2 neither opens the door, nor triggers the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 2 neither opens the door, nor triggers the alarm"

severity failure;

-- 11

reset <= '0';

day\_time <= '0';

code <= x"6";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 2, 6 neither opens the door, nor triggers the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 2, 6 neither opens the door, nor triggers the alarm"

severity failure;

-- 12

reset <= '0';

day\_time <= '1';

code <= a;

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 2, 6, A neither opens the door, nor triggers the alarm, switching daytime to 1 and not pressing 'O' , will not open the door nor trigger the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 2, 6, A neither opens the door, nor triggers the alarm, switching daytime to 1 and not pressing 'O' , will not open the door nor trigger the alarm"

severity failure;

-- 13

reset <= '0';

day\_time <= '0';

code <= x"0";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 2, 6, A, 0 neither opens the door, nor triggers the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 2, 6, A, 0 neither opens the door, nor triggers the alarm"

severity failure;

-- 14

reset <= '0';

day\_time <= '0';

code <= x"5";

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Entering 2, 6, A, 0, 5 opens the door, but doesn't trigger the alarm"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Entering 2, 6, A, 0, 5 opens the door, but doesn't trigger the alarm"

severity failure;

-- 15

wait for clk\_period;

assert door = door\_loon and alarm = alarm\_loon

report "door\_loon != door OR alarm\_loon != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

assert door = door\_scan and alarm = alarm\_scan

report "door\_scan != door OR alarm\_scan != alarm. Waiting for a clock cycle, should reset both door and alarm, to zero"

severity failure;

assert false

report "FSM PASSED ALL TEST SUCCESSFULLY."

severity error;

wait;

end process;

process begin

clk <= '1', '0' after (clk\_period/2);

wait for clk\_period;

end process;

end architecture all\_tb\_behav;