Rudra Goel

Lab 05 Report

ECE 2031 L10

01 October 2024

A screenshot of a graph

Description automatically generated

**Figure 1.** Simulation of three state Door Alert State Machine with “clock” signal oscillating at 10GHz. Reset signal “resetn” active low. Inputs “inner” & “outer” to state machine provide 100% coverage between state transitions to prove correct behavior. Includes assertion of output “ajar” when “inner” is low and “outer” is high for two clock cycles.

A screenshot of a computer

Description automatically generated

**Figure 2.** Capture of configured Signal Tap window to monitor signals “clk\_out” as state machine’s clock, “KEY0” as reset signal, “inner\_in” & “outer\_in” as inputs “inner” & “outer,” and “ajar\_out” as output signal. Sampling at 50Hz based on signal “sample\_clk” for a total of 512 samples. No trigger conditions set.

A screenshot of a computer

Description automatically generated

**Figure 3.** Completed acquisition of signals with Signal Tap on Door Alert state machine set to trigger on rising edge of “KEY0.” Acquisition illustrates sequence of leaving doors closed, then opening the outer door for one rising clock edge, opening the inner door for two rising clock edges, then closing inner door for one more clock edge to assert “ajar\_out” high, and finally closing outer door to assert “ajar\_out” low.

Appendix A

VHDL Code Implementing Door Alert Moore State Machine

-- Door\_state-Machine.vhd (VHDL)

-- Author: Rudra Goel

-- This code implements a state machine for:

-- Door Alert Mechanism

-- 09/30/2024

library IEEE;

use IEEE.std\_logic\_1164.all;

entity door\_state\_machine is

Port (

clock : in std\_logic;

resetn : in std\_logic;

inner : in std\_logic;

outer : in std\_logic;

ajar : out std\_logic

);

end door\_state\_machine;

architecture behaviour of door\_state\_machine is

-- make new types for state

type state\_type is (nothing\_bad, undesireable\_once, undesireable\_twice);

signal state, next\_state : state\_type;

begin

-- following process assigns next state to current state

-- also resets state if resetn is low

-- state only changes on rising edges of clock

-- resetn being low

process(clock, resetn)

begin

-- following logic only changes on rising clock edges

if resetn = '0' then

state <= undesireable\_once;

elsif rising\_edge(clock) then

state <= next\_state;

end if;

end process;

--process to define combinational logic

process(state, inner, outer)

begin

case state is

when nothing\_bad =>

if inner = '0' and outer = '1' then

next\_state <= undesireable\_once;

else

next\_state <= nothing\_bad;

end if;

when undesireable\_once =>

if inner = '0' and outer = '1' then

next\_state <= undesireable\_twice;

elsif inner = '0' and outer = '0' then

next\_state <= nothing\_bad;

else

next\_state <= undesireable\_once;

end if;

when undesireable\_twice =>

if inner = '0' and outer = '1' then

next\_state <= undesireable\_twice;

else

next\_state <= nothing\_bad;

end if;

end case;

end process;

process(state)

begin

case state is

when undesireable\_twice =>

ajar <= '1';

when others =>

ajar <= '0';

end case;

end process;

end behaviour;