Rudra Goel
Lab 05 Report
ECE 2031 L10
01 October 2024

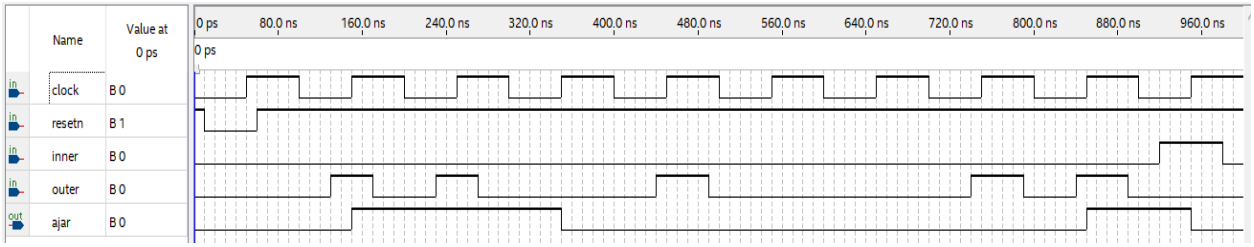

Figure 1. Simulation of three state Door Alert State Machine with "clock" signal oscillating at 10GHz. Reset signal "resetn" active low. Inputs "inner" & "outer" to state machine provide 100% coverage between state transitions to prove correct behavior. Includes assertion of output "ajar" when "inner"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

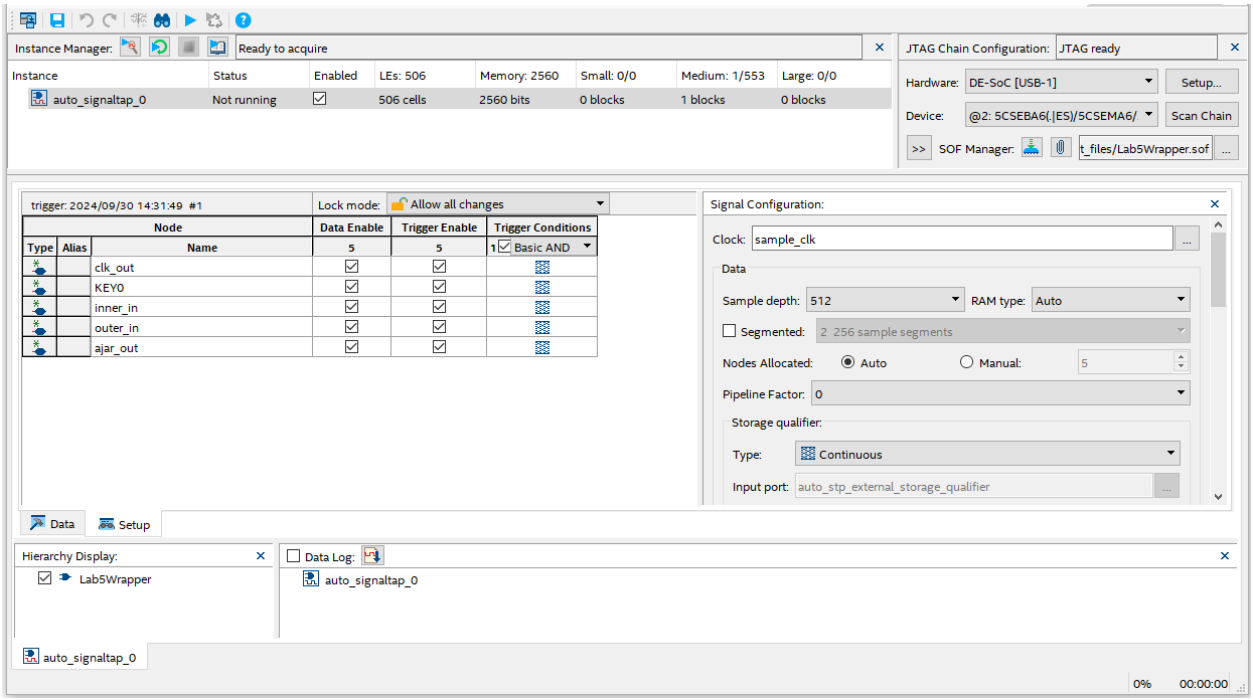

Figure 2. Capture of configured Signal Tap window to monitor signals "clk_out" as state machine's clock, "KEY0" as reset signal, "inner_in" & "outer_in" as inputs "inner" & "outer," and "ajar_out" as output signal. Sampling at 50Hz based on signal "sample_clk" for a total of 512 samples. No trigger conditions set.

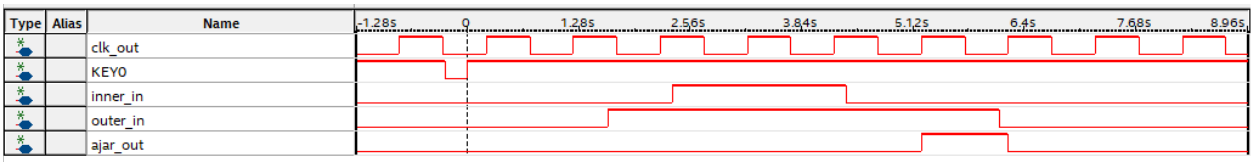

Figure 3. Completed acquisition of signals with Signal Tap on Door Alert state machine set to trigger on rising edge of "KEY0." Acquisition illustrates sequence of leaving doors closed, then opening the outer door for one rising clock edge, opening the inner door for two rising clock edges, then closing 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;
```