Máquinas de Estados

Laboratório 3

2020.1

PUC
RIO

Fernando Homem da Costa - 1211971

Departamento de Engenharia Elétrica
Pontifícia Universidade Católica do Rio de Janeiro

ENG1448 - Computação Digital

Professor: Wouter Caarls

21 de Setembro de 2020

Lista de Figuras

Conteúdo

1 Introdução

Este projeto é a terceira tarefa da disciplina de Computação Digital (ENG1448) do segundo semestre de 2020, ministrada pelo professor Wouter Caarls.

Aprendemos escrever código sequencial, e como funcionam máquinas de estados. Nesse laboratório, vamos usar esse conhecimento para implementar o controlador de um elevador de dois andares. O elevador tem dois botões de fora (um em cada andar), e dois de dentro. Para os dois conjuntos, o botão de cima manda o elevador para cima, e o botão de baixo o manda para baixo. Existem LEDs dentro dos botões para mostrar se o elevador foi chamado. Para efetuar o movimento, tem motor para subir e descer, e sensores que indicam a presença do elevador no primeiro ou segundo andar. Se o elevador é chamado para uma direção enquanto já está andando para a outra direção, primeiramente finaliza o movimento atual e só depois volta. Então, é um elevador comum, só que não tem porta e não espera para as pessoas entrarem ou saírem.

2 Desenvolvimento

2.1 Exercício 1:

Desenhe a máquina de estados do sistema:

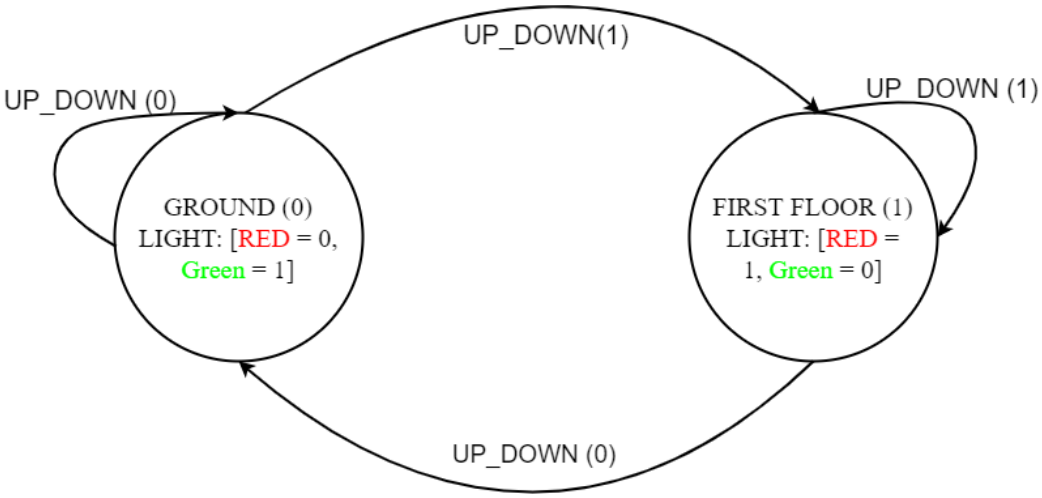

Figura 1: Máquina de Estados

2.2 Exercício 2:

Implementa a máquina de estados em VHDL. Pode usar 1, 2 ou 3 processos, e código concorrente ou sequencial de forma que acha melhor. Verifique que o código é compilável pelo simulador e sintetizável pelo sintetizador.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity elevator is
    port (
            CLK, RST, LIGHT, UP_DOWN : in  std_logic;
            FLOOR : out  std_logic
        );
    end elevator;

architecture rtl of elevator is

type floor_state_type is (ground, first_floor);

signal floor_state : floor_state_type;

begin
    floor_state_machine: process(CLK)
    begin
        if CLK'event and CLK = '1' then
            if RST = '1' then
                floor_state <= ground;
            else
                case floor_state is
                    when ground =>
                        if(UP_DOWN = '1' and LIGHT = '0') then
                            floor_state <= first_floor;
                        else
                            floor_state <= ground;
                        end if;

                    when first_floor =>
                        if(UP_DOWN = '1' and LIGHT = '0') then
                            floor_state <= ground;
                        else
```

```
                        floor_state <= first_floor;
                    end if;

                when others =>
                    floor_state <= first_floor;
            end case;
        end if;
    end if;
end process;

floor <= "0" when (floor_state = ground) else
         "1" when (floor_state = first_floor);
end rtl;
```

2.3 Exercício 3:

Implementa um testbench para testar o funcionamento. O testbench deve criar todas as situações extraordinárias que podem acontecer.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY elevator_tb IS
END elevator_tb;

ARCHITECTURE behavior OF elevator_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT elevator
    PORT(
         CLK : IN  std_logic;
         RST : IN  std_logic;
         LIGHT : IN  std_logic;
         UP_DOWN : IN  std_logic;
         FLOOR : OUT  std_logic_vector(1 downto 0)
        );
    END COMPONENT;


   --Inputs
```

```
    signal CLK : std_logic := '0';
    signal RST : std_logic := '0';
    signal LIGHT : std_logic := '0';
    signal UP_DOWN : std_logic := '0';

          --Outputs
    signal FLOOR : std_logic_vector(1 downto 0);

    -- Clock period definitions
    constant CLK_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
    uut: elevator PORT MAP (
           CLK => CLK,
           RST => RST,
           LIGHT => LIGHT,
           UP_DOWN => UP_DOWN,
           FLOOR => FLOOR
         );

    -- Clock process definitions
    CLK_process :process
    begin
                CLK <= '0';
                wait for CLK_period/2;
                CLK <= '1';
                wait for CLK_period/2;
    end process;


        -- Stimulus process
        stim_proc: process
        begin
                wait for 100 ns;
                RST <= '1';
                LIGHT <= '0';
                UP_DOWN <= '0';

                wait for CLK_PERIOD;
```

```
                RST <= '0';
                LIGHT <= '0';
                UP_DOWN <='1';

                wait for CLK_PERIOD;

                RST <= '0';
                LIGHT <= '1';
                UP_DOWN <='1';

                wait for CLK_PERIOD*2;

                RST <= '0';
                LIGHT <= '0';
                UP_DOWN <='0';
                wait for CLK_PERIOD;

                wait;
end process;

END;
```

2.4 Exercício 4:

Roda o simulador para verificar o funcionamento do código. Cola uma captura do waveform graph no relatório, e explica o que acontece. Existe algum atraso?

Resposta: Não foi possível gerar o waveform graph devido ao erro gerado no ISim. Entretanto, analogamente com o que visto em sala de aula, pudemos perceber que há atraso na resposta do UP_DOWN em relação ao clock.

2.5 Exercício 5:

Use o manual do starter kit para criar um User Constraints File que liga as entradas e saídas do controlador aos botões e LEDs da placa (veja páginas 17 e 20). Também liga o clock ao CLK_50MHZ (página 23).

Resposta: Não consegui fazer, pois não entendi como relacionar o USER Constraints File com o projeto.