Universidade Tecnológica Federal do Paraná
Campus Apucarana
Engenharia de Computação

Iago Macarini Brito, RA: 2320665
Luis Henrique Ferracciu Pagotto Mendes, RA: 2272016

Relatório 8

Semáforos

LRCO7A - Lógica Reconfigurável
Professor: Marcelo de Oliveira

Apucarana - PR
2025

1 Introdução

1.1 Máquina de Estados

As máquinas de estados são conceitos computacionais utilizados para elaboração de técnicas e procedimentos em sistemas digitais que seguem um fluxo sequencial. Tal máquina funciona a partir da transição de um estado para outro com base na entrada e em condições especiais, gerando uma saída correspondente a cada estado.

Uma máquina de estados completa é definida por cinco elementos principais:

- Estados: são as diferentes situações em que o sistema pode se encontrar em determinados momentos;
- Transições: são regras que definem como o sistema transitará de um estado para outro, geralmente são sinais de entrada ou contadores;
- Estado Atual: o estado atual é um registro que armazena como o sistema está no momento, ou seja, qual estado está ativo;
- Lógica de Próximo Estado: é a lógica combinacional que define qual estado será o próximo a ser atingido a partir da entrada do sistema;
- Lógica de Saída: circuito que define qual saída será produzida em cada estado.

2 Metodologia

2.1 Resolução

A partir da explicação realizada há pouco, é possível entender o funcionamento do código presente nas figuras 1 e 2. Em que, temos o funcionamento de uma máquina de estados, sendo cada semáforo um sistema e as cores Verde, Amarelo e Vermelho um estado diferente, a transição dos estados para esse projeto foi definida como sendo a frequência de clock da própria placa, mas manipulada, para diminuir o tempo de espera entre as transições. Além disso, temos algumas condicionais que modificam algumas funções dos semáforos, como as ativações dos switches denominados 'mode-select' no código.

O mode-select declarado no código funciona com 2 bits e faz com que os semáforos mudem de seguintes formas:

- Quando definido em "00", os semáforos funcionam normalmente;
- Quando definido em "01", os semáforos entram em modo teste, o que aumenta a velocidade das transições;
- Quando definido em "10", os semáforos entram em modo stand-by, modo que apenas as luzes amarelas piscam e não há transição entre as cores.

Devido o tamanho das imagens e do código para implementação deste projeto, abaixo segue um link para o GitHub que contém o código completo e implementado para melhor visualização:
https://github.com/luismendess/atividade8

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity semaforos is
    port (
        clock       : in std_logic;                     -- Sinal de clock principal (50 MHz na DE10-Lite)
        reset       : in std_logic;                     -- Botão de reset (ativo em baixo na placa)
        mode_select : in std_logic_vector(1 downto 0); -- Seleção de modo: 00-Normal, 01-Teste, 10-Standby
        semaforo1   : out std_logic_vector(2 downto 0); -- Saídas para o primeiro semáforo (G,Y,R)
        semaforo2   : out std_logic_vector(2 downto 0)  -- Saídas para o segundo semáforo (G,Y,R)
    );
end entity semaforos;

architecture semaforos of semaforos is
    -- Definição dos estados das luzes dos semáforos
    -- Atenção: Na DE10-Lite, os LEDs acendem com '0', então invertemos a lógica
    constant GREEN  : std_logic_vector(2 downto 0) := "100"; -- Verde (bit 2 ativo baixo)
    constant YELLOW : std_logic_vector(2 downto 0) := "010"; -- Amarelo (bit 1 ativo baixo)
    constant RED    : std_logic_vector(2 downto 0) := "001"; -- Vermelho (bit 0 ativo baixo)
    constant OFF    : std_logic_vector(2 downto 0) := "000"; -- Todas as luzes apagadas

    -- Estados da máquina de estados principal
    type estado_t is (S1_GREEN_S2_RED, S1_YELLOW_S2_RED, S1_RED_S2_GREEN, S1_RED_S2_YELLOW, STANDBY_ON, STANDBY_OFF);
    signal estado_atual, proximo_estado : estado_t;

    -- Divisor de frequência para tornar a contagem visível
    constant DIVISOR : integer := 25000000; -- Divide por 25M para ter períodos de 0,5 segundo

    -- Tempos para o modo normal (em períodos de 0,5 segundo)
    constant TEMPO_VERDE_S1   : integer := 10;  -- 5 segundos no verde para semáforo 1
    constant TEMPO_AMARELO    : integer := 4;   -- 2 segundos no amarelo (ambos semáforos)
    constant TEMPO_VERDE_S2   : integer := 8;   -- 4 segundos no verde para semáforo 2 (tempo diferente)

    -- Tempos para o modo teste (5x mais rápido)
    constant TEMPO_VERDE_S1_TESTE   : integer := 2;  -- 1 segundo
    constant TEMPO_AMARELO_TESTE    : integer := 1;  -- 0,5 segundo
    constant TEMPO_VERDE_S2_TESTE   : integer := 2;  -- 1 segundo

    -- Tempo para o standby (piscar a cada 0.5 segundos)
    constant TEMPO_STANDBY    : integer := 1;   -- 0,5 segundo

    -- Sinais internos
    signal contador : integer range 0 to TEMPO_VERDE_S1 := 0;
    signal tempo_atual : integer range 0 to TEMPO_VERDE_S1 := 0;
    signal clock_div : std_logic := '0';
    signal contador_div : integer range 0 to DIVISOR := 0;

begin
    -- Processo para divisão do clock (torna a operação visível)
    process(clock, reset)
    begin
        if reset = '0' then  -- Reset ativo em baixo na DE10-Lite
            contador_div <= 0;
            clock_div <= '0';
        elsif rising_edge(clock) then
            if contador_div = DIVISOR-1 then
                contador_div <= 0;
                clock_div <= not clock_div;
            else
                contador_div <= contador_div + 1;
            end if;
        end if;
    end process;

    -- Processo para atualizar o estado atual e o contador
    process(clock_div, reset)
    begin
        if reset = '0' then -- Reset ativo em baixo na DE10-Lite
            -- Reset: inicializa estado e contador
            estado_atual <= S1_GREEN_S2_RED;
            contador <= 0;
        elsif rising_edge(clock_div) then
            -- Incrementa contador e verifica transições de estado
            if contador >= tempo_atual - 1 then
                estado_atual <= proximo_estado;
                contador <= 0;
            else
                contador <= contador + 1;
            end if;
        end if;
    end process;

    -- Processo para determinar o próximo estado e o tempo necessário
    process(estado_atual, mode_select, contador)
    begin
        -- Valores padrão (serão sobrescritos conforme necessário)
        proximo_estado <= estado_atual;
        tempo_atual <= TEMPO_VERDE_S1;

        case mode_select is
            when "00" =>  -- Modo Normal
                case estado_atual is
                    when S1_GREEN_S2_RED =>
                        tempo_atual <= TEMPO_VERDE_S1;
                        if contador >= TEMPO_VERDE_S1 - 1 then
                            proximo_estado <= S1_YELLOW_S2_RED;
                        end if;

                    when S1_YELLOW_S2_RED =>
                        tempo_atual <= TEMPO_AMARELO;
                        if contador >= TEMPO_AMARELO - 1 then
                            proximo_estado <= S1_RED_S2_GREEN;
                        end if;

                    when S1_RED_S2_GREEN =>
                        tempo_atual <= TEMPO_VERDE_S2;
                        if contador >= TEMPO_VERDE_S2 - 1 then
                            proximo_estado <= S1_RED_S2_YELLOW;
                        end if;

                    when S1_RED_S2_YELLOW =>
                        tempo_atual <= TEMPO_AMARELO;
                        if contador >= TEMPO_AMARELO - 1 then
                            proximo_estado <= S1_GREEN_S2_RED;
                        end if;

                    when others =>
                        proximo_estado <= S1_GREEN_S2_RED;
                end case;

            when "01" =>  -- Modo Teste (mais rápido)
                case estado_atual is
                    when S1_GREEN_S2_RED =>
                        tempo_atual <= TEMPO_VERDE_S1_TESTE;
                        if contador >= TEMPO_VERDE_S1_TESTE - 1 then
                            proximo_estado <= S1_YELLOW_S2_RED;
                        end if;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity semaforos is
    port (
        clock       : in std_logic;                     -- Sinal de clock principal (50 MHz na DE10-Lite)
        reset       : in std_logic;                     -- Botão de reset (ativo em baixo na placa)
        mode_select : in std_logic_vector(1 downto 0); -- Seleção de modo: 00-Normal, 01-Teste, 10-Standby
        semaforo1   : out std_logic_vector(2 downto 0); -- Saídas para o primeiro semáforo (G,Y,R)
        semaforo2   : out std_logic_vector(2 downto 0)  -- Saídas para o segundo semáforo (G,Y,R)
    );
end entity semaforos;

architecture semaforos of semaforos is
    -- Definição dos estados das luzes dos semáforos
    -- Atenção: Na DE10-Lite, os LEDs acendem com '0', então invertemos a lógica
    constant GREEN  : std_logic_vector(2 downto 0) := "100"; -- Verde (bit 2 ativo baixo)
    constant YELLOW : std_logic_vector(2 downto 0) := "010"; -- Amarelo (bit 1 ativo baixo)
    constant RED    : std_logic_vector(2 downto 0) := "001"; -- Vermelho (bit 0 ativo baixo)
    constant OFF    : std_logic_vector(2 downto 0) := "000"; -- Todas as luzes apagadas

    -- Estados da máquina de estados principal
    type estado_t is (S1_GREEN_S2_RED, S1_YELLOW_S2_RED, S1_RED_S2_GREEN, S1_RED_S2_YELLOW, STANDBY_ON, STANDBY_OFF);
    signal estado_atual, proximo_estado : estado_t;

    -- Divisor de frequência para tornar a contagem visível
    constant DIVISOR : integer := 25000000; -- Divide por 25M para ter períodos de 0,5 segundo

    -- Tempos para o modo normal (em períodos de 0,5 segundo)
    constant TEMPO_VERDE_S1   : integer := 10;  -- 5 segundos no verde para semáforo 1
    constant TEMPO_AMARELO    : integer := 4;   -- 2 segundos no amarelo (ambos semáforos)
    constant TEMPO_VERDE_S2   : integer := 8;   -- 4 segundos no verde para semáforo 2 (tempo diferente)

    -- Tempos para o modo teste (5x mais rápido)
    constant TEMPO_VERDE_S1_TESTE   : integer := 2;  -- 1 segundo
    constant TEMPO_AMARELO_TESTE    : integer := 1;  -- 0,5 segundo
    constant TEMPO_VERDE_S2_TESTE   : integer := 2;  -- 1 segundo

    -- Tempo para o standby (piscar a cada 0.5 segundos)
    constant TEMPO_STANDBY    : integer := 1;   -- 0,5 segundo

    -- Sinais internos
    signal contador : integer range 0 to TEMPO_VERDE_S1 := 0;
    signal tempo_atual : integer range 0 to TEMPO_VERDE_S1 := 0;
    signal clock_div : std_logic := '0';
    signal contador_div : integer range 0 to DIVISOR := 0;

begin
    -- Processo para divisão do clock (torna a operação visível)
    process(clock, reset)
    begin
        if reset = '0' then  -- Reset ativo em baixo na DE10-Lite
            contador_div <= 0;
            clock_div <= '0';
        elsif rising_edge(clock) then
            if contador_div = DIVISOR-1 then
                contador_div <= 0;
                clock_div <= not clock_div;
            else
                contador_div <= contador_div + 1;
            end if;
        end if;
    end process;

    -- Processo para atualizar o estado atual e o contador
    process(clock_div, reset)
    begin
        if reset = '0' then -- Reset ativo em baixo na DE10-Lite
            -- Reset: inicializa estado e contador
            estado_atual <= S1_GREEN_S2_RED;
            contador <= 0;
        elsif rising_edge(clock_div) then
            -- Incrementa contador e verifica transições de estado
            if contador >= tempo_atual - 1 then
                estado_atual <= proximo_estado;
                contador <= 0;
            else
                contador <= contador + 1;
            end if;
        end if;
    end process;

    -- Processo para determinar o próximo estado e o tempo necessário
    process(estado_atual, mode_select, contador)
    begin
        -- Valores padrão (serão sobrescritos conforme necessário)
        proximo_estado <= estado_atual;
        tempo_atual <= TEMPO_VERDE_S1;

        case mode_select is
            when "00" =>  -- Modo Normal
                case estado_atual is
                    when S1_GREEN_S2_RED =>
                        tempo_atual <= TEMPO_VERDE_S1;
                        if contador >= TEMPO_VERDE_S1 - 1 then
                            proximo_estado <= S1_YELLOW_S2_RED;
                        end if;

                    when S1_YELLOW_S2_RED =>
                        tempo_atual <= TEMPO_AMARELO;
                        if contador >= TEMPO_AMARELO - 1 then
                            proximo_estado <= S1_RED_S2_GREEN;
                        end if;

                    when S1_RED_S2_GREEN =>
                        tempo_atual <= TEMPO_VERDE_S2;
                        if contador >= TEMPO_VERDE_S2 - 1 then
                            proximo_estado <= S1_RED_S2_YELLOW;
                        end if;

                    when S1_RED_S2_YELLOW =>
                        tempo_atual <= TEMPO_AMARELO;
                        if contador >= TEMPO_AMARELO - 1 then
                            proximo_estado <= S1_GREEN_S2_RED;
                        end if;

                    when others =>
                        proximo_estado <= S1_GREEN_S2_RED;
                end case;

            when "01" =>  -- Modo Teste (mais rápido)
                case estado_atual is
                    when S1_GREEN_S2_RED =>
                        tempo_atual <= TEMPO_VERDE_S1_TESTE;
                        if contador >= TEMPO_VERDE_S1_TESTE - 1 then
                            proximo_estado <= S1_YELLOW_S2_RED;
                        end if;
```

Figura 1: Parte 1 código VHDL comentado. Figura 2: Parte 2 código VHDL comentado.

A partir da estruturação e execução do código, foi possível gerar o diagrama RTL representado pela figura 3, o relatório, a simulação (figura 4) e o pin planner (figura 5). Com tudo gerado e funcional, o código foi enviado à placa DE10-Lite e foram realizados os testes para que o projeto seja mostrado ao professor em sala de aula. O RTL também segue em anexo para caso haja dificuldade no entendimento do circuito gerado.

Figura 3: Diagrama RTL.

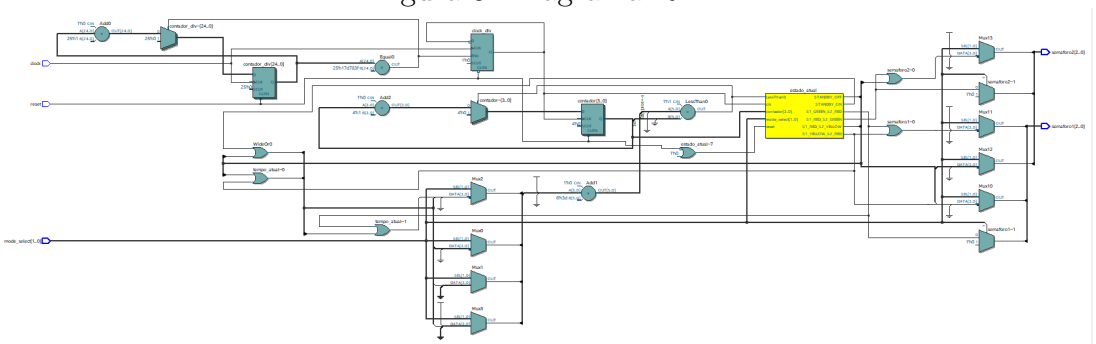


Figura 4: Simulação.

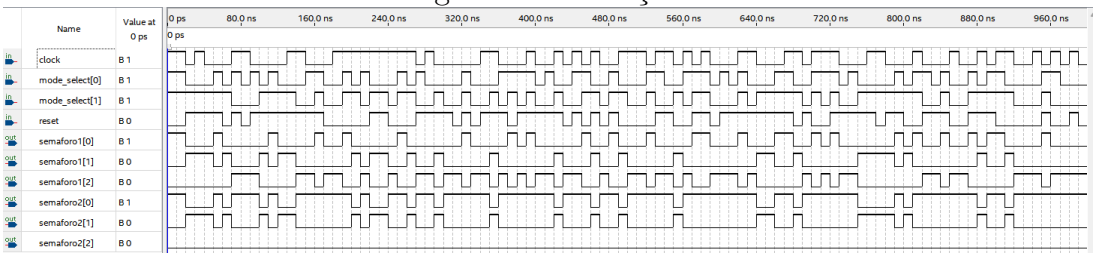
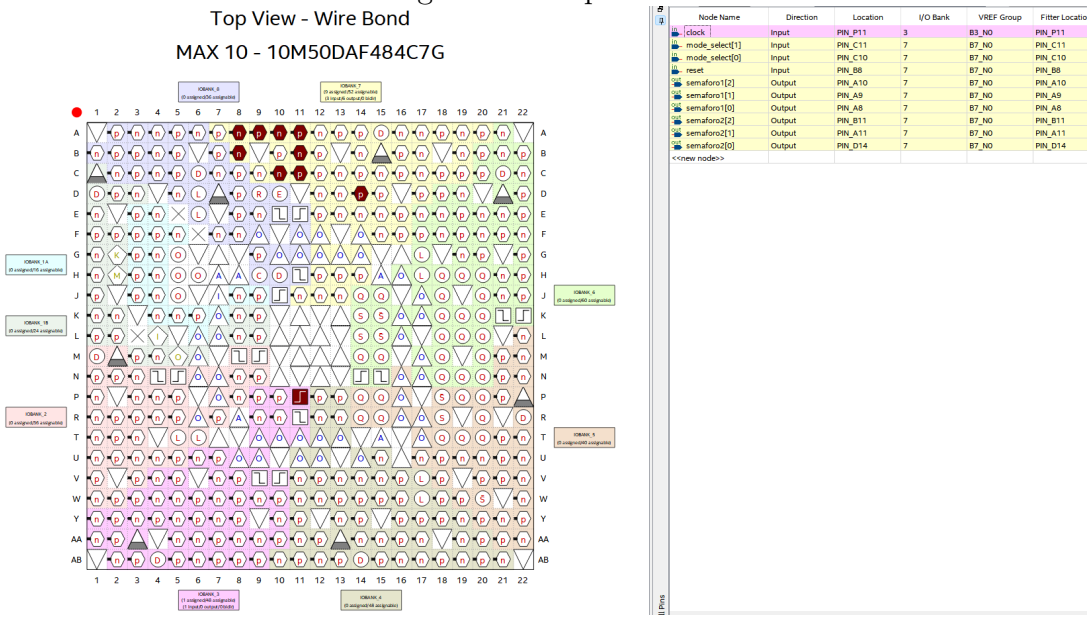


Figura 5: Pin planner.



Referências

[1] PEDRONI, V. A., **Circuit Design and Simulation with VHDL**. 2. ed. Londres, England: MIT Press, 2010.

[2] TOCCI, R.; WIDMER, N.; MODD, G. **Sistemas Digitais - Princípios e Aplicações**. [S.I]: Pearson Education Limited, 2011