Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Engenharia de Computação e Automação
DCA0212.2 - Circuitos Digitais
Docente: Tiago Barros

Laboratório 5 - Máquinas de estados finitos (FSM)

Objetivos

1. Reforçar os conceitos e experimentar a descrição em VHDL de máquinas de estados finitos;
2. Pôr em prática conceitos aprendidos na disciplina Circuitos Digitais - Teoria.

Introdução

Nesta aula iremos implementar e simular máquinas de estados finitos, a partir de descrições em VHDL por portas lógicas e comportamentais.

Máquina de estados finitos

Uma máquina de estados finitos, ou FSM (do inglês, *finite state machine*), é um formalismo matemático que consiste em:

- Um conjunto de estados.
- Um conjunto de entradas e um conjunto de saídas.
- Um estado inicial, isto é, um estado para se começar quando o sistema é energizado. O estado inicial de um sistema pode ser indicado por meio de uma linha com seta, chamada de aresta, que não tem estado de origem e aponta para o estado inicial. Uma FSM pode ter apenas um estado inicial.
- Uma descrição que indique para qual estado devemos ir a seguir, com base no estado atual e nos valores das entradas. Por convenção, utiliza-se arestas orientadas ou dirigidas juntamente com as condições de entrada associadas, as quais nos dizem qual é o próximo estado. Essas arestas são conhecidas como *transições*.
- Uma descrição de quais são os valores de saída que devem ser gerados em cada estado. Em uma FSM, a atribuição de uma saída é conhecida como *ação.*

Usamos uma representação gráfica, conhecida como *diagrama de estados*, para representar uma FSM. Os diagramas de estados são bastante utilizados por permitirem uma visualização ampla do comportamento de uma máquina de estados finitos.

O projeto de um sistema digital representado por uma FSM gera um circuito digital conhecido como *bloco de controle* ou *controlador*. Este tipo de processo se aplica quando desejamos projetar um circuito digital que possui comportamento sequencial.

Vamos utilizar uma FSM para representar o projeto de um sistema digital que deve ser usado como parte de um sistema de cirurgia laser. Tal sistema funciona acionando um laser durante um intervalo de tempo preciso. Uma arquitetura genérica para este sistema está mostrada na Figura 1.

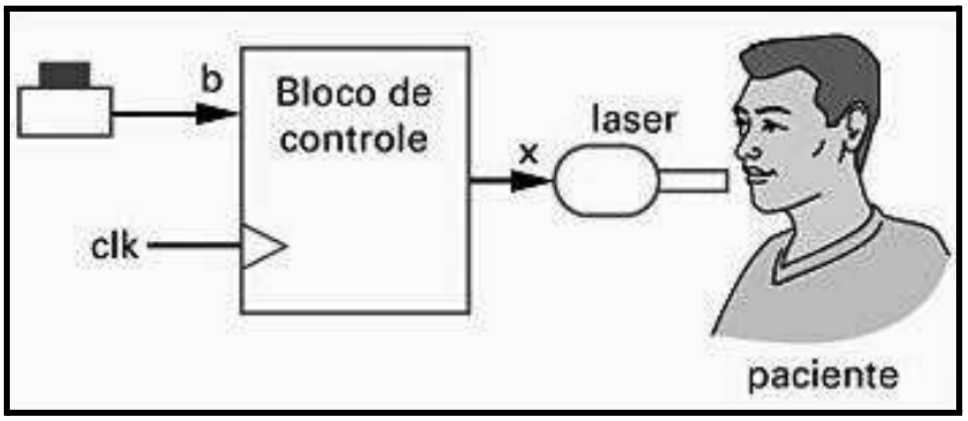

Figura 1 - Sistema temporizador de laser.

Um cirurgião ativa o laser pressionando um botão. Assuma, então, que o laser deve permanecer ativado por exatamente 30 ns. Assumindo que o período do nosso relógio é 10 ns, então 30 ns significa 3 ciclos de relógio (clock). Precisamos projetar um componente de bloco de controle que, tendo detectado que b=1, mantém x em nível alto por exatamente 3 ciclos de relógio, ativando o laser por exatamente 30 ns.

Para projetar este sistema, vamos criar uma FSM com quatro estados: *Desligado* (*Des*), *Ligado1* (*Lig1*), *Ligado2* (*Lig2*) e *Ligado3* (*Lig3*). A saída x deverá ser 0 durante um ciclo, no estado *Des*, e 1 durante três ciclos, nos estados *Lig1*, *Lig2* e *Lig3*. Iremos impor condições de entradas às transições. Para que ocorra uma transição do estado *Des* para o estado *Lig1*, devemos ter uma borda de subida no relógio e b=1. Acrescentaremos também uma transição do estado *Des* retornando ao próprio estado *Des*, com a condição de que ocorra uma borda de subida e b=0. As transições dos estados *Lig1* para *Lig2*, *Lig2* para *Lig3* e *Lig3* para *Des* ocorrem sempre nas próximas bordas de subida do relógio. O diagrama de tempo da Figura 2 mostra o comportamento de estado e de saída do sistema para os dados valores de b.

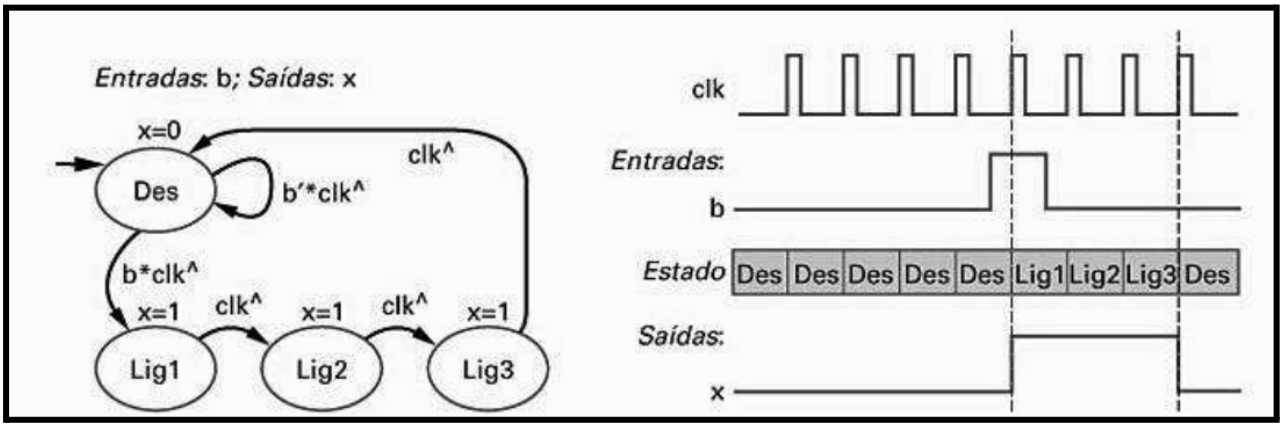

Figura 2 - Sistema para três ciclos em nível alto: diagrama de estados (à esquerda) e diagrama de tempo (à direita).

A FSM deve ser interpretada como segue. Começamos com nosso estado inicial de *Desligado*. Permaneceremos nesse estado até que uma das duas transições que saem dele esteja na condição de verdadeira. Uma dessas transições tem por condição b' AND uma borda de subida de relógio (b'*clk^); nesse caso, a transição irá se dar retornando ao estado *Desligado*. A outra transição tem por condição b AND uma borda de subida de relógio (b*clk^); nesse caso, a transição ocorrerá indo para o estado *Lig1*. Permaneceremos no estado *Lig1* até que se torne verdadeira a condição necessária à transição de saída do estado, ou seja, uma borda de subida do relógio - nesse caso, a transição irá se dar passando-se ao estado *Lig2*. De modo semelhante, permaneceremos no estado *Lig2* até a próxima borda de subida do relógio, quando ocorrerá uma transição para o estado *Lig3*. Permaneceremos em *Lig3* até a próxima borda de subida do relógio, quando será feita uma transição de volta ao estado *Desligado*. No estado *Desligado*, temos a ação associada a saída x=0, ao passo que, nos estados *Lig1, Lig2,* e *Lig3*, temos a ação associada à saída x=1.

Desse modo, usando uma FSM, descrevemos precisamente o comportamento sequenciado no tempo que é esperado do sistema temporizador de laser. Podemos simplificar a notação da FSM, tornando implícita a borda de subida do relógio, como ilustrado na Figura 3.

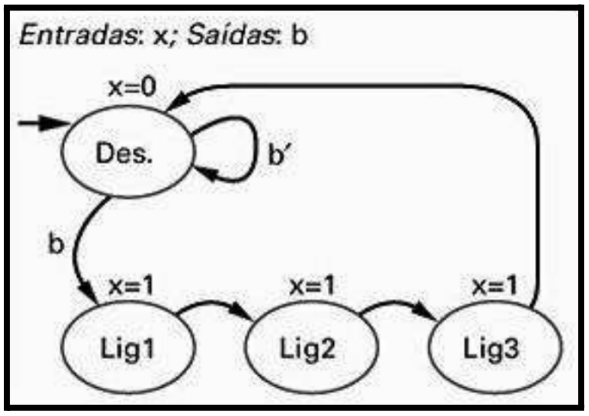

Figura 3 - Diagrama de estados do temporizador laser, no qual se assume que qualquer transição está sendo submetida a uma operação AND juntamente com a borda de subida do relógio.

Arquitetura padrão do bloco de controle para implementar uma FSM na forma de circuito sequencial

A arquitetura de um bloco de controle padrão para uma FSM consiste em um registrador de estado e uma lógica combinacional, conforme ilustrado na Figura 4. O registrador de estado consiste em um registrador que contém um número binário que representa o estado atual. As entradas da lógica combinacional são as entradas da FSM e também as saídas do registrador de estado. As saídas da lógica combinacional são as saídas da FSM e também os bits do próximo estado que serão carregados no registrador de estado. Os detalhes da lógica combinacional determinam o comportamento do circuito. Na Figura 4, assumimos que o registrador de estado possui *m* bits de largura. A máquina de estados finitos, representada pela arquitetura da Figura 4, é conhecida como máquina de Moore.

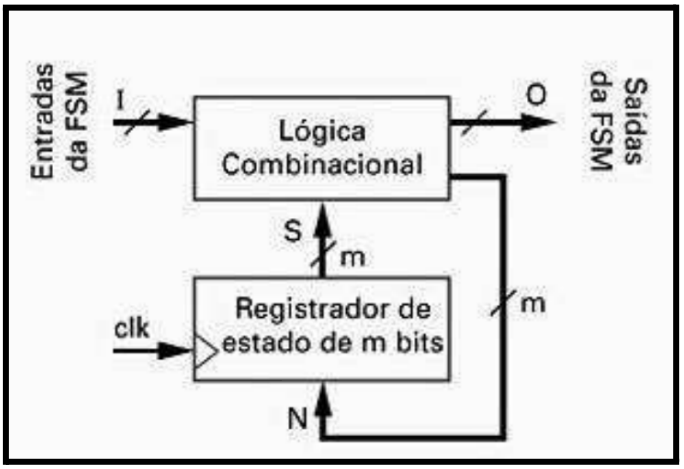

Figura 4 - Arquitetura de um bloco de controle padrão - visão geral.

Projeto de bloco de controle

Podemos projetar um bloco de controle usando um processo de cinco passos, descrito na Figura 5. Iremos ilustrar esse processo usando como exemplo o sistema temporizador de laser.

	Passo	Descrição
Passo 1	Capture a FSM	Crie uma FSM que descreva o comportamento desejado do bloco de controle.
Passo 2	Crie a arquitetura	Crie a arquitetura padrão usando um registrador de estado com largura apropriada e uma lógica combinacional, cujas entradas são os bits do registrador de estado e as entradas da FSM e cujas saídas são os bits de próximo estado e as saídas da FSM.
Passo 3	Codifique os estados	Atribua um número binário único a cada um dos estados. Cada número binário que representa um estado é conhecido como uma *codificação*. Qualquer codificação poderá ser usada desde que cada estado tenha uma codificação única.
Passo 4	Crie a tabela de estados	Crie uma tabela-verdade para a lógica combinacional de modo tal que a lógica irá gerar as saídas e os sinais de próximo estado corretos para a FSM. Ordenando as entradas primeiro com os bits de estado faz com que a tabela-verdade descreva o comportamento dos estados. Assim, a tabela é uma tabela-verdade.
Passo 5	Implemente a lógica combinacional	Implemente a lógica combinacional usando qualquer método.

Figura 5 - Processo de projeto de um bloco de controle.

Passo 1 - Capture a FSM. A FSM já foi capturada e seu comportamento está ilustrado na Figura 3.

Passo 2 - Crie a arquitetura. A arquitetura do bloco de controle padrão para a FSM do temporizador de laser está ilustrada na Figura 6. O registrador de estado tem dois bits de largura para representar cada um dos quatro estados. A lógica combinacional tem a entrada externa b e as entradas s1 e s0 que vêm do registrador de estado, e tem a saída externa x e as saídas n1 e n0 que se dirigem ao registrador de estado.

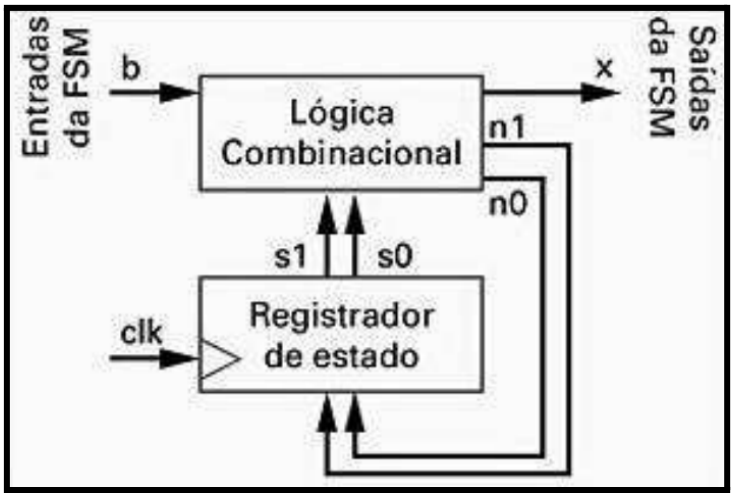

Figura 6 - Arquitetura de bloco de controle padrão para o temporizador laser.

Passo 3 - Codifique os estados. Podemos codificar os estados como segue. *Des:* 00; *Lig1*: 01; *Lig2*: 10 e *Lig3*: 11. Lembre-se, qualquer codificação não repetida é aceitável. O diagrama de estados com as codificações está mostrado na Figura 7.

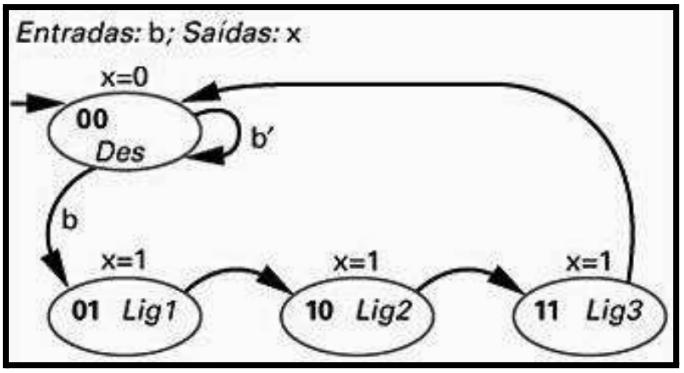

Figura 7 - Diagrama de estados do temporizador de laser com os estados codificados.

Passo 4 - Crie a tabela de estados. Dadas a arquitetura da implementação e a codificação binária de cada estado, podemos criar a tabela de estados para a lógica combinacional, como mostrado na Figura 8. Nas colunas de entrada, se listarmos primeiro

as entradas do registrador de estado, então poderemos ver facilmente quais linhas correspondem a quais estados. Na esquerda, preenchemos todas as combinações de entradas como se faz com uma tabela-verdade. Em cada linha, olhamos no diagrama de estados na Figura 3 para determinar as saídas apropriadas. Para as duas linhas que começam com s1s0=00 (estado *Desligado*), x deve ser 0. Se b=0, o bloco de controle deverá permanecer no estado *Desligado*, de modo que n1n0 deverá ser 00. Se b=1, o bloco de controle deverá ir para o estado *Lig1*, de modo que n1n0 deverá ser 01.

De modo semelhante, para as duas linhas que começam com s1s0=01 (estado *Lig1*), x deve ser 1 e o próximo estado deve ser *Lig2* (independente do valor de b) de modo que n1n0 deverá ser 10. Completamos as últimas quatro linhas de modo semelhante.

	Entradas			Saídas		
	s1	s0	b	x	n1	n0
Des	0	0	0	0	0	0
	0	0	1	0	0	1
Lig1	0	1	0	1	1	0
	0	1	1	1	1	0
Lig2	1	0	0	1	1	1
	1	0	1	1	1	1
Lig3	1	1	0	1	0	0
	1	1	1	1	0	0

Figura 8 - Tabela de estados para o bloco de controle do temporizador de laser.

Tenha cuidado e observe a diferença entre as entradas e saídas da FSM da Figura 3 e as entradas e saídas da lógica combinacional da Figura 9; esta última inclui os bits que se originam no registrador de estado e os que se destinam a este mesmo registrador.

Passo 5 - Implemente a lógica combinacional. Podemos terminar o projeto usando o processo de projeto de lógica combinacional, visto na disciplina de teoria de Circuitos Digitiais. Da tabela verdade, obtemos as seguintes equações para as três saídas de lógica combinacional, já simplificadas.

$$x = s1 + s0$$
$$n1 = s1' s0 + s1 s0'$$
$$n0 = s1' s0' b + s1 s0'$$

Obtemos, então, o circuito sequencial da Figura 9, que implementa a FSM.

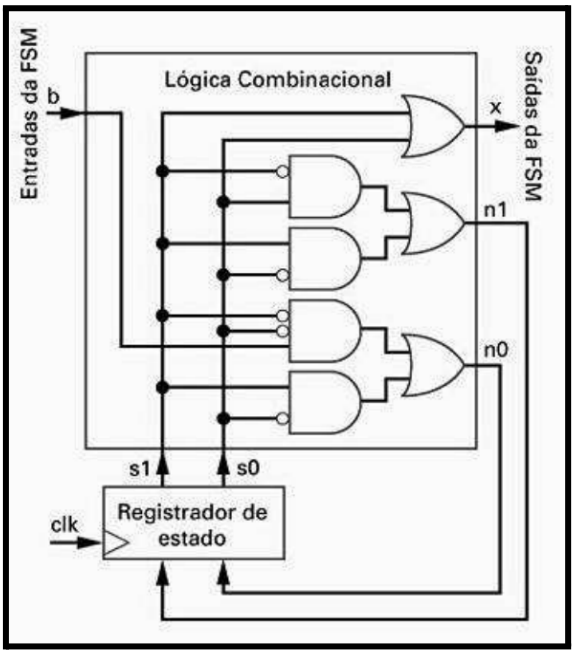

Figura 9 - Implementação final do bloco de controle para o temporizador de laser de três ciclos em nível alto.

Descrição por portas lógicas em VHDL de uma FSM

No Quadro 1, pode-se observar o código com a descrição por portas lógicas em VHDL do bloco de controle para o temporizador de laser de três ciclos em nível alto, ilustrado na Figura 9. O Quadro 2 apresenta os componentes que contém o registrador de estado e o circuito combinacional do bloco de controle do Quadro 1.

A Figura 10 ilustra a forma de onda obtida com a simulação do código presente no Quadro 1. Note que a partir do instante em que a entrada *b* assume valor igual a 1, a partir da próxima borda de subida do relógio, a saída *x* permanece ativada por, exatamente, três ciclos de relógio.

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

entity LaserT is
port(b, clk : in bit;
       x      : out bit);
end LaserT;

architecture behavior of circuito is
  signal n1, n0: bit; -- Proximo estado
```

```
  signal s1, s0: bit; -- Estado atual
  component reg2 is
    port(c, i1, i0 : in bit;
           q1, q0  : out bit);
  end component;
  component comb_circ is
    port(buttom, c, current1, current0  : in bit;
           next1, next0, laser_out         : out bit);
  end component;
begin
         u1 : reg2 port map(c => clk, i1 => n1, i0 => n0, q1 => s1, q0 => s0);
         u2 : circ_comb port map(buttom => b, c => clk,
                                 current1 => s1, current0 => s0,
                                 next1 => n1, next0 => n0, laser_out => x);
end architecture behavior;
```

Quadro 1: Código da FSM do temporizador de laser com descrição por portas lógicas.

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

entity reg2 is
port(c, i1, i0 : in bit;
       q1, q0   : out bit);
end reg2;

architecture behav of reg2 is
begin
  process(c)
    begin
    if (clk ' event AND clk = '1') then
      q1 <= i1;
      q0 <= i0;
    end if;
  end process;
end architecture behav;
```

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

entity comb_circ is
port(buttom, c, current1, current0 : in bit;
       next1, next0, laser_out         : out bit);
end comb_circ;

architecture behav of comb_circ is
begin
  laser_out <= current1 OR current0;
  next1 <= (NOT(current1) AND current0)
OR (current1 AND NOT(current0));
  next0 <= (NOT(current1) AND
NOT(current0) AND buttom) OR (current1
AND NOT(current0));
end architecture behav;
```

Quadro 2: Componentes da descrição por portas lógicas do registrador de estado e circuito combinacional.

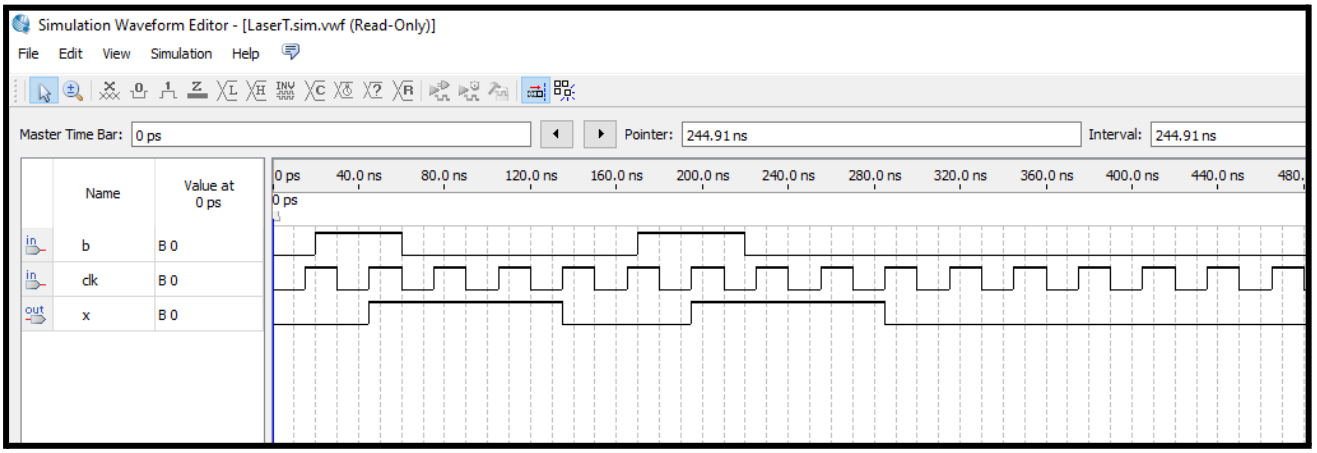

Figura 10 - Diagrama de tempo do código do Quadro 1.

A estrutura CASE-WHEN

A estrtura CASE-WHEN é executada de forma sequencial, podendo somente ser utilizada em regiões de código sequencial (determinada pela estrutura PROCESS). A estrtura CASE-WHEN faz com que o programa siga um dentre vários caminhos diferentes, dependendo do valor de um sinal, variável ou expressão. É uma alternativa mais elegante a uma declaração IF-THEN-ELSIF-ELSE com múltiplos ELSIF. A sintaxe básica para a estrutura CASE-WHEN é mostrada no Quadro 3.

```
case <expressão> is
        when <condição> =>
                -- código para esta condição

        when <condição> =>
                -- código para esta condição
        …
end case;
```

Quadro 3 – Sintaxe da estrutura CASE-WHEN da linguagem VHDL.

A <expressão> é geralmente uma variável ou um sinal. A declaração CASE pode conter várias condições para o mesmo “when”, mas apenas uma delas será selecionada.

A <condição> pode ser da seguinte forma:

- Um valor único como "11":
    - when "11" =>
- Um intervalo, como 5 a 10:
    - when 5 to 10 =>
- Pode conter vários valores como 1 ou 3 ou 5:
    - when 1 | 3 | 5 =>

- Quando nenhuma outra condição for correspondida, usamos a condição "others", a qual é equivalente ao ramo "else" na estrutura IF-THEN-ELSIF-ELSE.
    - when others =>

Descrição comportamental em VHDL de uma FSM

O Quadro 4 mostra um modo de se modelar um bloco de controle em VHDL. O bloco de controle modelado é descrito pela FSM mostrada nas figuras 2 e 3. A entidade VHDL denominada *LaserTimer* define as entradas e saídas do bloco de controle.

A arquitetura VHDL descreve o comportamento da entidade. Ela consiste em dois processos, um que modela o registrador de estado e outro que modela a lógica combinacional. Os dois formam a arquitetura padrão do bloco de controle da Figura 6.

O primeiro processo descreve o registrador de estado do bloco de controle. Esse processo denominado *statereg* é sensível às entradas *clk* e *rst*. Se a entrada *rst* estiver habilitada, então o processo atribuirá assincronamente o estado *S_Off* da FSM ao sinal *currentstate*. Em caso contrário, se o relógio estiver subindo, o processo atualizará o registrador de estado com o próximo estado.

Os sinais *currentstate* e *nextstate* são definidos como sendo de um tipo definido pelo usuário, de nome *statetype*. Esse tipo é definido pelo comando ***type*** (*tipo*) e especifica os valores possíveis que um sinal desse tipo pode assumir. Ao se especificar *statetype* para representar os estados de uma FSM, a declaração ***type*** listará os nomes de todos os estados do bloco de controle, especificamente *S_Off*, *S_On1*, *S_On2* e *S_On3*.

O segundo processo descreve a lógica combinacional do bloco de controle. Esse processo denominado *comblogic* é sensível às entradas da lógica combinacional da Figura 6, especificamente, às entradas externas (nesse caso, *b*) e as saídas do registrador de estado (*currentstate*). Quando qualquer um desses sinais da lista de sensibilidade sofre uma alteração, o processo coloca o valor apropriado do estado atual na saída *x*, neste caso, da FSM. O processo também determina qual deve ser o estado seguinte, baseado no estado atual e nos valores das entradas (isto é, nas condições das transições da FSM). Na próxima borda de subida do relógio, o próximo estado será carregado no registrador de estado pelo processo do registrador de estado.

Observe que a arquitetura declara os dois sinais *currentstate* e *nextstate*. Sinais são visíveis em todos os processos de uma arquitetura. O sinal *currentstate* representa o valor atual que está armazenado no registrador de estado. O sinal *nextstate* representa o valor que vem da lógica combinacional e que se dirige ao registrador de estado. Observe também que a arquitetura declara esses sinais como sendo do tipo *statetype*, o qual foi definido na arquitetura como um tipo cujo valor pode ser *S_Off*, *S_On1*, *S_On2* e *S_On3*.

A Figura 11 ilustra a forma de onda obtida com a simulação do código presente no Quadro 4. Note que a partir do instante em que a entrada *b* assume valor igual a 1, a partir da próxima borda de subida do relógio, a saída *x* permanece ativada por, exatamente, três ciclos de relógio.

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

entity LaserTimer is
        port (b         : in std_logic;
              x         : out std_logic;
              clk, rst : in std_logic
              );
end LaserTimer;

architecture behavior of LaserTimer is
        type statetype is
            (S_Off, S_On1, S_On2, S_on3);
        signal currentstate, nextstate: statetype;

begin
        statereg: process(clk, rst)
        begin
                if (rst='1') then -- estado inicial
                        currentstate <= S_Off;
                elsif (clk='1' and clk ' event) then
                        currentstate <= nextstate;
                end if;
        end process;
        comblogic: process (currentstate, b)
        begin
                case currentstate is
                        when S_Off =>
                                x <='0'; -- laser desligado
                                if (b='0') then
                                        nextstate <= S_Off;
                                else
                                        nextstate <= S_On1;
                                end if;
                        when S_On1 =>
                                x <= '1'; -- laser on
                                nextstate <=S_On2;
                        when S_On2 =>
                                x <= '1'; -- laser ainda ligado
                                nextstate <= S_On3;
                        when S_On3 =>
                                x <= '1'; -- laser ainda ligado
                                nextstate <= S_Off;
                end case;
        end process;
end behavior;
```

Quadro 4: Descrição comportamental em VHDL do bloco de controle do temporizador de laser.

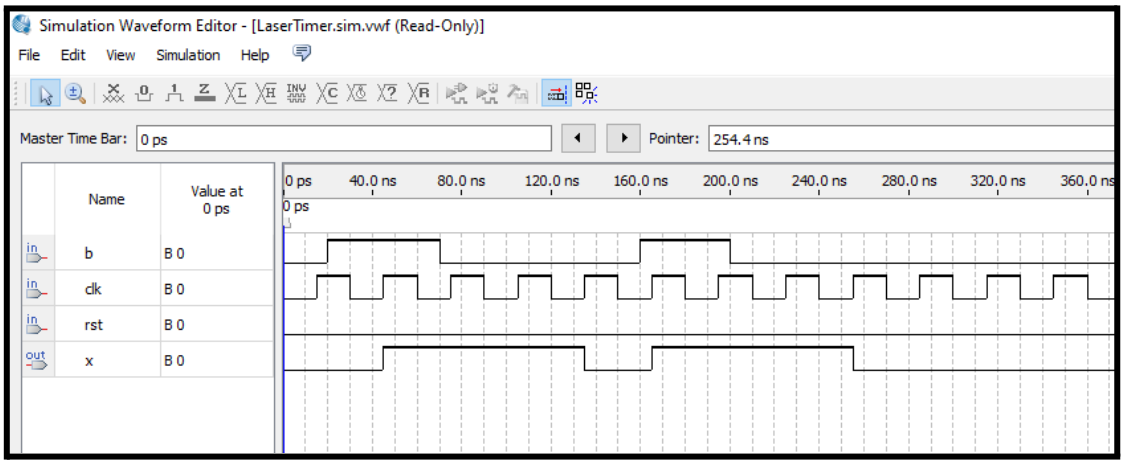

Figura 11 - Diagrama de tempo do código do Quadro 4.

O marca-passo:

Podemos descrever o funcionamento do bloco de controle de um marca-passo simples usando a FSM da Figura 12. O lado esquerdo da figura mostra o marca-passo que consiste em um bloco de controle e um temporizador. O temporizador tem uma entrada t, que aplica um *reset* quando t=1. Depois do *reset*, o temporizador começa uma contagem regressiva a partir de 0,8 segundo. Se o temporizador chegar a 0, ele colocará sua saída z em 1. Pode acontecer do temporizador sofrer um *reset* antes de chegar a 0, caso em que o temporizador não colocará z em 1 e o temporizador começará uma nova contagem regressiva a partir de 0,8 segundo. O bloco de controle tem uma entrada s, que se torna 1 quando uma contração é detectada no ventrículo direito. O bloco de controle tem uma saída p, que é colocada em 1 quando o bloco de controle deve disparar uma contração.

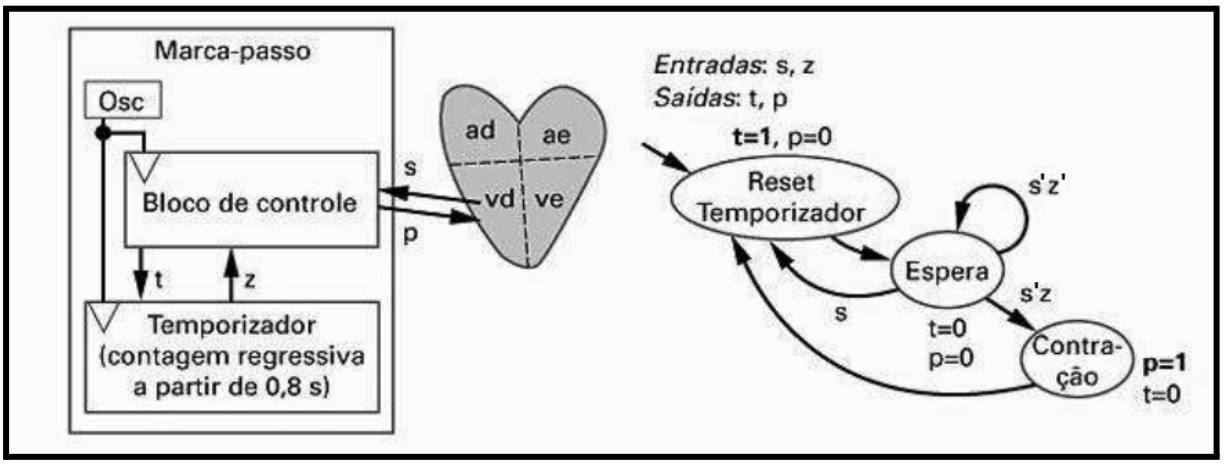

Figura 12 - FSM do bloco de controle de um marca-passo básico.

O lado direito da figura mostra o funcionamento do bloco de controle como uma FSM. Inicialmente, no estado *ResetTemporizador*, o bloco de controle causa um *reset* no temporizador fazendo t=1. Normalmente, o bloco de controle ficará esperando no estado *Espera* e assim permanecerá, enquanto uma contração não for detectada (s') nem o temporizador tiver chegado a 0 (z'). Se o bloco de controle detectar uma contração natural (s), então ele fará novamente um *reset* no temporizador e voltará a ficar esperando. Por outro lado, se o bloco de controle ver que o temporizador chegou a 0 (z=1), então ele seguirá para o estado *Contração*, o qual obrigará o coração a se contrair aplicando p=1. Em seguida, o bloco de controle voltará a esperar novamente. Assim, enquanto o coração estiver se contraindo naturalmente, o marca-passo não aplicará estímulo ao coração. Entretanto, se o coração não se contrair dentro de 0,8 segundo, após a última contração (natural ou forçada), o marca-passo irá forçar uma contração.

Atividades:

1) Projete o bloco de controle do marca-passo da Figura 12, usando o projeto em 5 passos, descrito na Figura 5. Implemente o bloco de controle utilizando uma descrição em VHDL por portas lógicas. Simule o circuito descrito e plote as formas de onda de suas entradas e saídas.
2) Implemente o bloco de controle do marca-passo da Figura 12 utilizando uma descrição comportamental em VHDL. Simule o circuito descrito e plote as formas de onda de suas entradas e saídas.
3) Entregue um relatório descrevendo a execução dos itens 1 e 2, contendo os projetos, todos os códigos VHDL, simulações e análises dos resultados.

Dica: para implementar um temporizador, utilize um contador decrescente com carga em paralelo, em que a saída tc seja 1 a cada 0,8 segundo. Por exemplo, para contar 0,8 segundo, você pode utilizar uma entrada de clock de 1 KHz, em que o contador contaria até 1000 em 1 segundo. Partindo de um valor inicial de 799, o contador decrescente contaria de 799 até 0 em 0,8 segundo, com essa contagem sendo indicada pela saída tc=1, sempre o contador chegasse a 0, a cada 0,8 segundo.