IMPLEMENTANDO MÁQUINAS DE ESTADOS SÍNCRONAS DO TIPO MOORE EM VHDL

1 REVISANDO A ESTRUTURA "*PROCESS*"

No experimento anterior vimos o elemento do VHDL chamado *processo*. A sintaxe do processo é:

```
label : process (sensitivity_list)
begin
   (sequential statements)
end process label;
```

Um processo é concorrente (isto é, ocorre ao mesmo tempo) de outras declarações (inclusive outros processos). Porém, dentro de um processo, as declarações são executadas *sequencialmente*. Um processo é executado apenas quando ocorre uma **mudança** em um dos elementos de sua lista de sensitividade (*sensitivity list*).

Podemos utilizar processos para implementar circuitos síncronos em VHDL. Por exemplo, um flip-flop tipo D, como o ilustrado na Figura 1, pode ser implementado utilizando a abordagem comportamental da maneira mostrada na Figura 2.

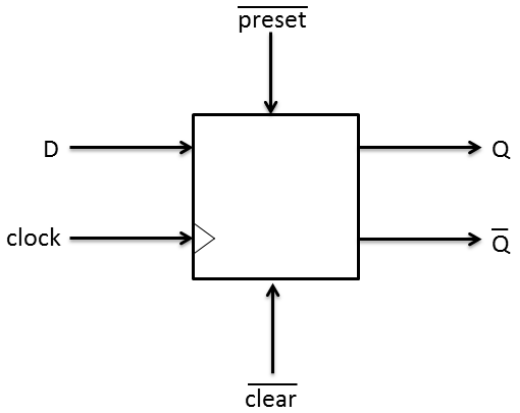

Figura 1 – Diagrama de blocos de um flip-flop D.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity flipflopD is
5     port (D, clock, preset, clear : in  STD_LOGIC;
6           Q, QN : out STD_LOGIC);
7   end flipflopD;
8
9   architecture flipflopD_arch of flipflopD is
10  begin
11    dff: process (clock,preset,clear)
12    begin
13      if (clear = '0') then
14        Q <= '0';
15        QN <= '1';
16      elsif (preset = '0') then
17        Q <= '1';
18        QN <= '0';
19      elsif rising_edge(clock) then
20        Q  <= D;
21        QN <= not(D);
22      end if;
23    end process dff;
24  end flipflopD_arch;
```

Figura 2 – Código VHDL de um flip-flop D.

No código acima, a lista de sensitividade do processo *dff* (linha 11) tem três elementos: *clock*, *preset* e *clear*. Qualquer mudança em qualquer um desses elementos causará a execução do processo e, uma vez em execução, o processo é executado sempre sequencialmente. Ele primeiro verifica se a entrada *clear* está ativa (ou seja, em nível 0) e, em caso positivo, ele muda as saídas de acordo (linhas 13 a 15). Caso contrário, ele verifica se a entrada *preset* está ativa (em nível 0) e, em caso positivo, ele muda as saídas de acordo (linhas 16 a 18). Finalmente, caso a mudança ocorrida foi uma mudança de 0 para 1 na entrada *clock* (descrita pela função rising_edge( )), ele copia o valor de *D* para a saída *Q* (linhas 19 a 21).

A arquitetura mostrada acima ilustra bem como utilizar processos para implementar circuitos sequenciais. A forma usual de se escrever uma máquina de estados em VHDL é mostrada a seguir.

2 IMPLEMENTANDO MÁQUINAS DE ESTADOS SÍNCRONAS DO TIPO MOORE EM VHDL

Considere a máquina de estados síncrona do tipo Moore descrita pelo diagrama de estados mostrado na Figura 3.

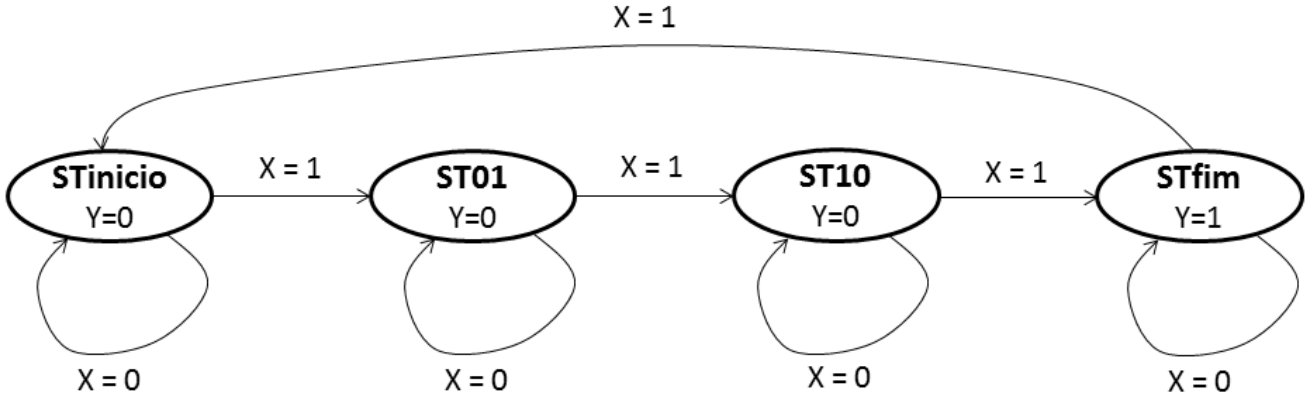

Figura 3 – Diagrama de estados de uma máquina de estados síncrona do tipo Moore.

Note que a máquina acima tem como entrada a variável *X* e, como saída, a variável *Y*. Com base no diagrama de estados, podemos montar a tabela de estados e saída do circuito (Tabela 1).

Tabela 1- Tabela de estados e saída

Estado atual	Próximo estado se X = 0	Próximo estado se X = 1	saída (Y)
STinicio	STinicio	ST01	0
ST01	ST01	ST10	0
ST10	ST10	STfim	0
STfim	STfim	STinicio	1

A implementação usual de máquinas de estados tem duas áreas bem definidas: lógica combinacional e registradores. Os registradores são responsáveis pela memória do circuito e a lógica combinacional decide a transição da memória e as saídas do circuito. Esse modelo é chamado de *register transfer logic*, ou RTL.

A máquina de estados descrita pelo diagrama da Figura 3 é implementada pelo código VHDL da Figura 4. Para entender essa arquitetura, precisamos de vários novos conceitos. Vamos discutir cada um deles a seguir.

Primeiramente, na linha 12, criamos um tipo especial de variável chamada *estado* (através do comando *type*) para guardar os estados da nossa máquina. Essa variável tem apenas quatro valores possíveis: (*STinicio*, *ST01*, *ST10* e *STfim*). Note que, a princípio, não se sabe como esses valores serão representados pelo sintetizador; apenas sabemos que são quatro valores distintos.

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

entity maquinaEstados is
  port (X, clock : in STD_LOGIC;
        Y : out STD_LOGIC);
end maquinaEstados;

architecture maquinaEstados_arch of maquinaEstados is

  -- define um novo tipo chamado 'estado' com os quatro valores possíveis listados
  type estado is (STinicio,ST01,ST10,STfim);

  -- cria dois sinais do tipo 'estado': currentState (estado atual) e nextState (próximo estado)
  signal currentState, nextState : estado;

begin

  -- processo síncrono (memória da máquina de estados)
  sync_proc: process(clock)
  begin
    if rising_edge(clock) then
      currentState <= nextState;
    end if;
  end process sync_proc;

  -- processo combinacional (lógica de saida e lógica de estado seguinte)
  comb_proc: process(currentState,X)
  begin
    case currentState is
      when STinicio =>
        Y <= '0';
        if (X = '1') then
          nextState <= ST01;
        else
          nextState <= STinicio;
        end if;
      when ST01 =>
        Y <= '0';
        if (X = '1') then
          nextState <= ST10;
        else
          nextState <= ST01;
        end if;
      when ST10 =>
        Y <= '0';
        if (X = '1') then
          nextState <= STfim;
        else
          nextState <= ST10;
        end if;
      when STfim =>
        Y <= '1';
        if (X = '1') then
          nextState <= STinicio;
        else
          nextState <= STfim;
        end if;
      when others =>
        Y <= '0';
        nextState <= STinicio;
    end case;
  end process comb_proc;

end maquinaEstados_arch;
```

Figura 4 – Implementação em VHDL, usando o modelo RTL, da máquina de estados síncrona do tipo Moore descrita pelo diagrama de estados mostrado na Figura 3.

Em seguida, criamos dois processos: *comb_proc* e *sync_proc* (respectivamente, combinacional e sequencial). O primeiro (linhas 20 a 25) gera a lógica de estado seguinte (*nextState*, no código), enquanto o segundo (linhas 28 a 63) trata de sincronizar esse estado com o estado atual (*currentState*, no código). Note que a lógica combinacional dentro de *comb_proc* é realizada utilizando uma estrutura *case*: dependendo do estado atual, assinalamos valores ao próximo estado e às saídas. Note que, embora não seja possível "enxergar" o circuito pelo exemplo do código RTL, conseguimos ver claramente a separação em camadas, isto é, a lógica combinacional e a lógica dos registradores.

O circuito sintetizado tem a topologia mostrada na Figura 5. A lógica de estado seguinte é implementada pelo processo combinacional **comb_proc** (linhas 28 a 63), o qual decide, a partir das entradas (*X*) e do estado atual (*currentState*) qual é o estado seguinte (*nextState*). Note que tanto o sinal *currentState* quanto o sinal *X* estão na lista de sensitividade do processo, o que indica que o processo é executado assim que uma mudança em um desses sinais ocorre. A lógica de saída também é implementada pelo processo combinacional **comb_proc**, o qual decide, exclusivamente a partir do estado atual (*currentState*) qual é a saída do circuito (*Y*).

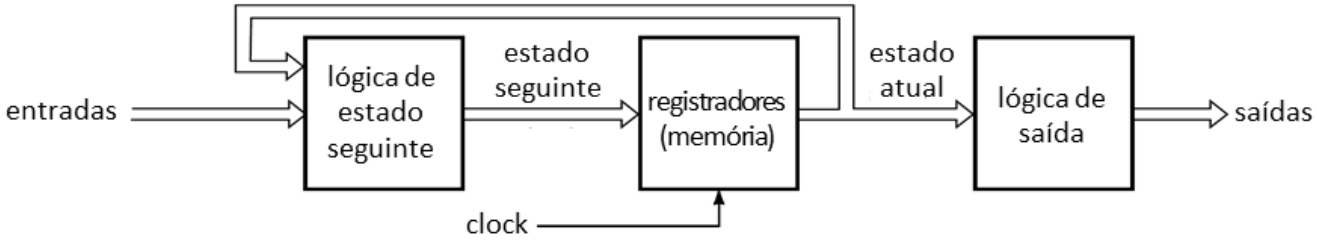

Figura 5 – Topologia genérica das máquinas de estados síncronas do tipo Moore.

Já a memória do circuito (um arranjo de registradores) é implementada pelo processo síncrono *sync_proc* (linhas 20 a 25), o qual registra a mudança de estados, a partir dos sinais de clock (*clock*) e de estado seguinte (*nextState*). Porém, apenas o sinal *clock* está na lista de sensitividade do processo, isto é, uma mudança em *nextState* não acarreta uma mudança de estado. Quando *clock* muda de 0 para 1, o processo atualiza o estado atual (*currentState*) com o valor que estava no sinal que indicava o estado seguinte (*nextState*). Neste momento, se o valor de *currentState* for alterado, isso acarretará uma execução do processo *comb_proc*, que pode mudar o valor de *nextState*. Porém, como *nextState* não está na lista de sensitividade do processo *sync_proc*, isto não acarretará em uma mudança de estados.

O processo sequencial (*sync_proc*) muda muito pouco entre diferentes máquinas de estados. A principal variação é a possível presença de entradas assíncronas (isto é, independentes do sinal de *clock*) de *clear* e *preset*, que podem ser implementadas como exemplificado na Figura 6. A principal mudança entre os códigos VHDL de uma máquina de estados para outra está mesmo no processo combinacional (*comb_proc*), que pode ser facilmente descrito a partir do diagrama de estados ou da tabela de transição de estados da máquina.

```
sync_proc: process(clock,reset)
begin
  if (reset='1') then
    currentState <= STinicio;
  elsif rising_edge(clock) then
    currentState <= nextState;
  end if;
end process sync_proc;
```

Figura 6 – Implementação alternativa da lógica dos registradores (processo síncrono *sync_proc*), desta vez incluindo uma condição para reinicialização assíncrona da máquina de estados. Note que a variável *reset* foi incluída na lista de sensitividade do processo, de modo a garantir que a máquina de estados seja reinicializada mesmo que não ocorra uma transição no sinal de *clock*.