IMPLEMENTANDO UM CONTADOR COM RESET E LOAD SÍNCRONOS EM VHDL

Neste tutorial, apresentaremos a implementação em VHDL de um contador de módulo 4, projetado como uma máquina de estados do tipo Moore. Essa máquina terá 4 estados, cada um associado a uma saída Q de 2 bits: de “00” (decimal 0) até “11” (decimal 3). As transições de estado serão controladas pelas variáveis de entrada:

- reset (que, de forma síncrona, leva a máquina de volta ao estado inicial se reset = ‘1’);
- enable e RCI (que são ativas em nível baixo, ou seja, a contagem está ativada quando enable = ‘0’ e RCI = ‘0’) (RCI vem da sigla em inglês *ripple carry-in*); e
- load e D (de forma síncrona, a máquina deve ser levada ao estado indicado por D se load = ‘1’).

A saída RCO (sigla do inglês *ripple carry-out*) é ativa em nível baixo, e deverá ser ‘0’ se e somente se Q = “11”, caso contrário será ‘1’. Essa saída é usada para cascatear contadores de módulo 4, de modo a construir contadores de módulo 16, módulo 64, etc. A ação de ‘reset’ deve ter prioridade sobre a ação de ‘load’, que por sua vez deve ter prioridade sobre a contagem.

A caixa preta desse contador é mostrada na Figura 1. A Tabela 1 mostra a tabela de transição de estados (em função das variáveis de entrada) e de saídas (em função somente do estado atual, já que se trata de uma máquina de estados do tipo Moore). Note que, quando reset = ‘0’ e load = ‘1’, a ação de ‘load’ é selecionada; nesse caso, o estado seguinte será determinado pela entrada D: se D = “00”, então STload = ST0; se D = ”01”, então STload = ST1; se D = “10”, então STload = ST2; e se D = ”11”, então STload = ST3.

O diagrama de estados é mostrado na Figura 2. Note que o diagrama de estados foi simplificado, pois não mostra as transições referentes à condição quando load = ‘1’. Incluir essas transições no diagrama tornaria sua visualização um tanto difícil.

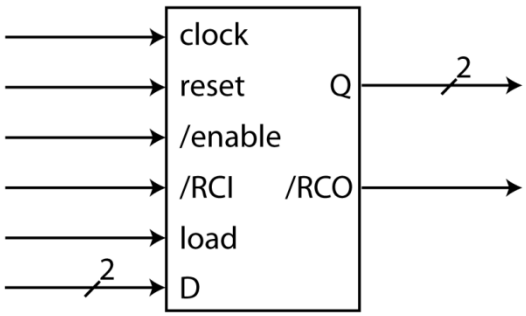

Figura 1 – Caixa preta do contador módulo 4.

Tabela 1 – Tabela de transição de estados e saídas do contador módulo 4.

entradas	ação					
	reseta	faz load	incrementa	espera		
reset	1	0	0	0	0	
load	x	1	0	0	0	
/enable	x	x	0	1	x	
/RCI	x	x	0	x	1	
estado atual	estado seguinte					saídas
						Q /RCO
ST0	ST0	STload*	ST1	ST0		00 1
ST1	ST0	STload*	ST2	ST1		01 1
ST2	ST0	STload*	ST3	ST2		10 1
ST3	ST0	STload*	ST0	ST3		11 0

*STload é o estado indicado pela entrada D: Se D = 00, então STload = ST0; se D = 01, então STload = ST1, etc.

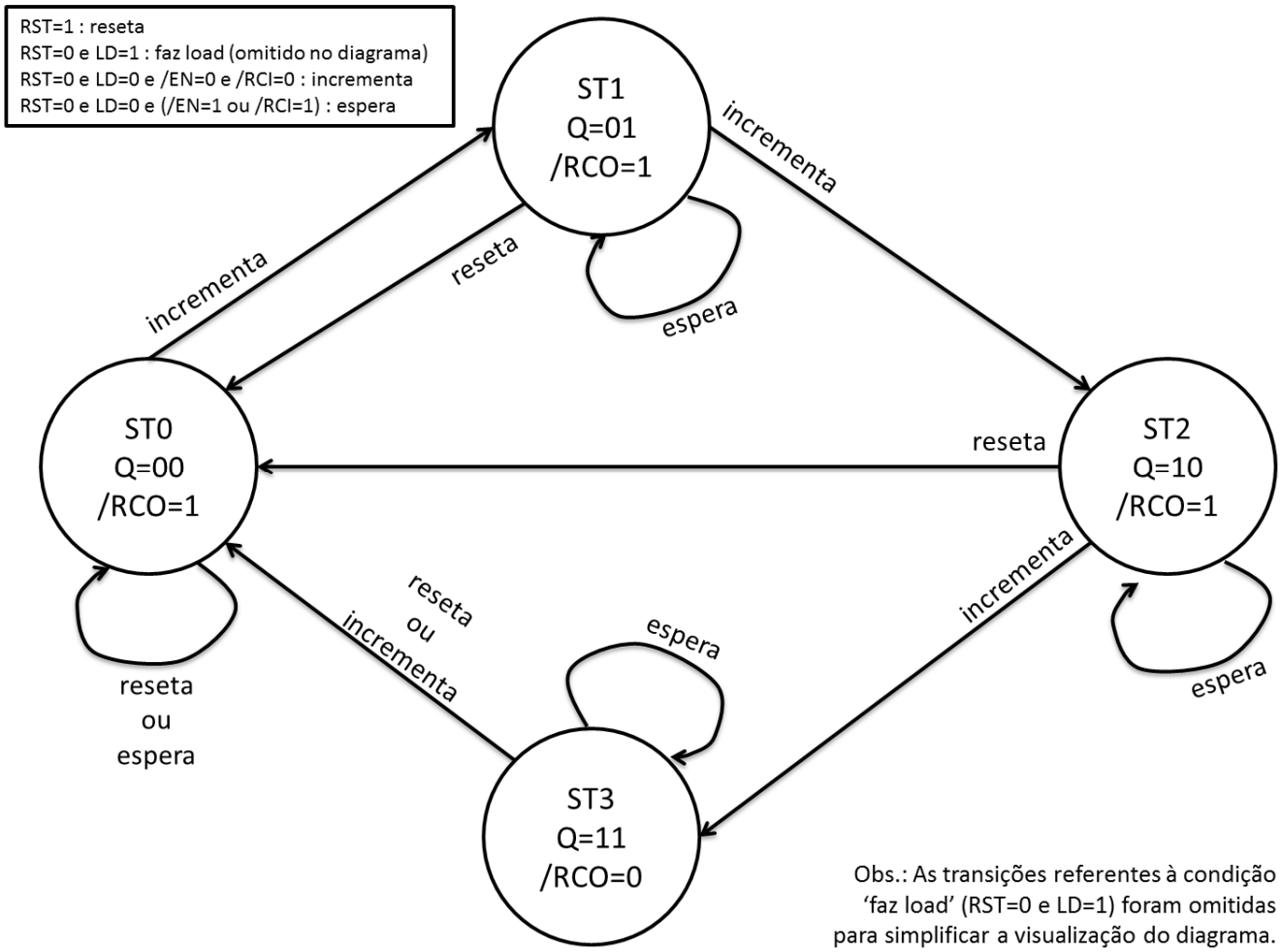

Figura 2 – Diagrama de estados de um contador de módulo 4 com entradas de 'reset' e 'load' síncronas.

O código VHDL de uma entidade que implementa esse contador é mostrado na Figura 3. Embora fosse mais econômico (do ponto de vista de linhas de código) implementar as ações de 'reset' e 'load', e até mesmo as condições para habilitar contagem (/enable = '0' e /rci = '0'), dentro da estrutura *sync_proc* (linhas 27 a 32), essa não é a forma correta de descrever esse sistema, pois pode confundir o sintetizador. A forma mais eficaz é colocar todas as condições de transição síncrona dentro da estrutura *comb_proc* (linhas 34 a 84), conforme exemplificado na Figura 3. Recomenda-se colocar dentro da estrutura *sync_proc* somente as condições assíncronas e a atribuição síncrona "`currentState <= nextState;`". Como regra geral, as condições assíncronas (na estrutura *sync_proc*) serão implementadas pelo sintetizador como lógicas conectadas a entradas de "preset" e "clear" assíncronas de flip-flops tipo D e as condições síncronas (na estrutura *comb_proc*) serão implementadas como lógicas conectadas à entrada D desses flip-flops (que é síncrona), ou a entradas de "preset" e "clear" síncronas (note que o FPGA utilizado na placa de desenvolvimento Basys2 dispõe de blocos lógicos que implementam flip-flops D tanto com preset e clear síncronos quanto assíncronos).

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

entity contador2bits is
  port( clock : in STD_LOGIC;
        reset : in STD_LOGIC;
        enable : in STD_LOGIC;
        rci : in STD_LOGIC;
        load : in STD_LOGIC;
        D : in STD_LOGIC_VECTOR(1 downto 0);
        Q : out STD_LOGIC_VECTOR(1 downto 0);
        rco : out STD_LOGIC );
end contador2bits;

architecture contador2bits_arch of contador2bits is
  type estado is (ST0,ST1,ST2,ST3);
  signal currentState, nextState, loadState : estado;
begin

  with D select
    loadState <= ST0 when "00",
                 ST1 when "01",
                 ST2 when "10",
                 ST3 when "11",
                 ST0 when others;

  sync_proc: process(clock)
  begin
    if rising_edge(clock) then
      currentState <= nextState;
    end if;
  end process sync_proc;

  comb_proc: process(currentState,reset,enable,rci,load,loadState)
  begin
    case currentState is

      when ST0 =>
        Q <= "00";
        rco <= '1';
        if (reset = '1') then nextState <= ST0;
        elsif (load = '1') then nextState <= loadState;
        elsif ((enable = '0') and (rci = '0')) then nextState <= ST1;
        else nextState <= ST0;
        end if;

      when ST1 =>
        Q <= "01";
        rco <= '1';
        if (reset = '1') then nextState <= ST0;
        elsif (load = '1') then nextState <= loadState;
        elsif ((enable = '0') and (rci = '0')) then nextState <= ST2;
        else nextState <= ST1;
        end if;

      when ST2 =>
        Q <= "10";
        rco <= '1';
        if (reset = '1') then nextState <= ST0;
        elsif (load = '1') then nextState <= loadState;
        elsif ((enable = '0') and (rci = '0')) then nextState <= ST3;
        else nextState <= ST2;
        end if;

      when ST3 =>
        Q <= "11";
        rco <= '0';
        if (reset = '1') then nextState <= ST0;
        elsif (load = '1') then nextState <= loadState;
        elsif ((enable = '0') and (rci = '0')) then nextState <= ST0;
        else nextState <= ST3;
        end if;

      when others =>
        Q <= "00";
        rco <= '1';
        if (reset = '1') then nextState <= ST0;
        elsif (load = '1') then nextState <= loadState;
        elsif ((enable = '0') and (rci = '0')) then nextState <= ST1;
        else nextState <= ST0;
        end if;

    end case;
  end process comb_proc;
end contador2bits_arch;
```

Figura 3 - Código VHDL de uma entidade que implementa um contador de módulo 4 com entradas de ‘reset’ e ‘load’ síncronas.