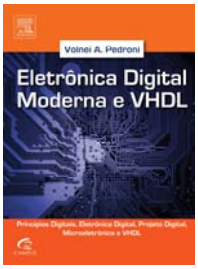


Eletrônica Digital Moderna e VHDL
Volnei A. Pedroni, Elsevier, 2010

Tradução (com revisão, atualização e ampliação) de
Digital Electronics and Design with VHDL
Elsevier / Morgan Kaufmann, USA, 2008

ELSEVIER

Soluções dos Exercícios Ímpares dos Capítulos 19-23

Nota 1: Em todas as soluções neste capítulo foram adotados nas portas os tipos-padrão da indústria, quais sejam, STD_LOGIC e STD_LOGIC_VECTOR.

Nota 2: Detalhes adicionais relativos aos exercícios com VHDL podem ser vistos na referência [1] abaixo.
[1] V. A. Pedroni, *Circuit Design and Simulation with VHDL*, 2nd Edition, MIT Press, 2010.

Capítulo 19: Resumo de VHDL

Exercício 19.1. Pacote *standard* em VHDL 2002

a) Na pasta libraries/vhdl/std que acompanha seu software para síntese com VHDL (Quartus II ou ISE, por exemplo), abra o arquivo *standard.vhd*, o qual contem o pacote *standard*, e observe que ele contem os seguintes tipos: BIT, BIT_VECTOR, BOOLEAN, INTEGER, NATURAL (subtipo), POSITIVE (subtipo), CHARACTER, STRING, REAL, TIME, DELAY_LENGTH (subtipo), SEVERITY LEVEL, FILE_OPEN_KIND e FILE_OPEN_STATUS.
b) Sintetizáveis sem restrições: BIT, BIT_VECTOR, BOOLEAN, INTEGER, NATURAL (subtipo), POSITIVE (subtipo), CHARACTER e STRING.
Somente para simulação: TIME e DELAY_LENGTH.
c) Veja tabela na figura 4.1 da referência da Nota 2 acima.

Exercício 19.3. Pacote *std_logic_1164*

a) Na pasta libraries/vhdl/ieee que acompanha seu software para síntese com VHDL (Quartus II ou ISE, por exemplo), abra o arquivo *std_1164.vhd*, o qual contem o pacote *std_logic_1164*, e veja qual é sua versão (com base na orientação dada no enunciado do exercício).
b) Se a versão for 1993:
Tipos: STD_ULOGIC, STD_ULOGIC_VECTOR e STD_LOGIC_VECTOR
Subtipos: STD_LOGIC, X01, X01Z, UX01 e UX01Z
Se a versão for 2008:
Tipos: STD_ULOGIC, STD_ULOGIC_VECTOR
Subtipos: STD_LOGIC, STD_LOGIC_VECTOR, X01, X01Z, UX01 e UX01Z
c) Veja tabela na figura 4.1 da referência da Nota 2 acima.

Exercício 19.5. Assinalamentos legais versus ilegais #2

Os tipos mais comuns de erros de assinalamento constam na página 441. Aqui, tem-se:
`x1(0) <= x2(0);` → Erro 4 (uso incorreto de parênteses em x1)
`x2(7 DOWNTO 5) <= y1(3 DOWNTO 0);` → Erro 2 (tamanhos diferentes)
`y2 <= -35;` → Erro 5 (operador de assinalamento incorreto; é "<:" para variável)
`x2(0 TO 2) <= x4(5 DOWNTO 3);` → Erro 1 (tipos diferentes) e erro 4 (indexação na ordem incorreta em x2)

Exercício 19.7. Operadores lógicos

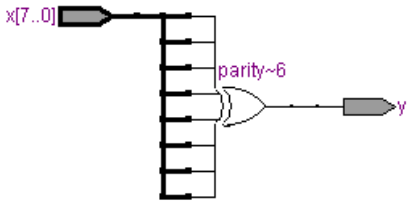
a) "11001110"
b) Idem acima
c) "00110011"

Exercício 19.9. Detector de paridade com código sequencial

```
1   ------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   ------------------------------------------------
```

```
5   ENTITY parity_detector IS
6      GENERIC (N: INTEGER := 8); --number of bits
7      PORT (x: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
8            y: OUT STD_LOGIC);
9   END ENTITY;
10  -------------------------------------------
11  ARCHITECTURE sequential OF parity_detector IS
12  BEGIN
13     PROCESS(x)
14        VARIABLE parity: STD_LOGIC;
15     BEGIN
16        parity := x(0);
17        FOR i IN 1 TO N-1 LOOP
18           parity := parity XOR x(i);
19        END LOOP;
20        y <= parity;
21     END PROCESS;
22  END ARCHITECTURE;
23  -------------------------------------------
```

O resultado apresentado pelo RTL Viewer após a compilação desse código é mostrado abaixo.



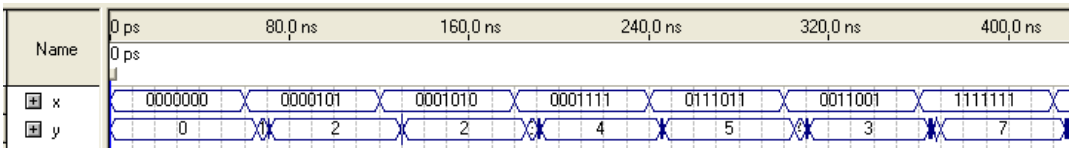
Exercício 19.11. Peso de Hamming com código concorrente

Como em praticamente todos os projetos, há várias maneiras de escrever o código VHDL correspondente. No presente caso, *temp* é um inteiro, de modo que, para passar seu valor a *y* (na linha 21), que é STD_LOGIC_VECTOR, uma função de conversão (*conv_std_logic_vector*) é utilizada, a qual provem do package *std_logic_arith* (linha 4).

Nota: Como não é permitido assinalar valor a um sinal mais do que uma vez, nas linhas 14-15 criou-se um sinal com dimensão uma unidade maior do que a dimensão do sinal a monitorar (*temp* é 1Dx1D, *x* é 1D). Esse importantíssimo artifício para código concorrente foi introduzido na seção 7.7 da referência citada na Nota 2 no início desse documento.

```
1   ---------------------------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   USE ieee.std_logic_arith.all; --conv. integer to slv
5   ---------------------------------------------------------------------------
6   ENTITY hamming_weight IS
7      GENERIC (bits_in: POSITIVE := 7;    --number of input bits
8               bits_out: POSITIVE := 3);  --number of output bits
9      PORT (x: IN STD_LOGIC_VECTOR(bits_in-1 DOWNTO 0);
10           y: OUT STD_LOGIC_VECTOR(bits_out-1 DOWNTO 0));
11  END ENTITY;
12  ---------------------------------------------------------------------------
13  ARCHITECTURE concurrent OF hamming_weight IS
14     TYPE natural_array IS ARRAY (0 TO bits_in) OF NATURAL RANGE 0 TO bits_in;
15     SIGNAL temp: natural_array;
16  BEGIN
17     temp(0) <= 0;
18     gen: FOR i IN 1 TO bits_in GENERATE
19        temp(i) <= temp(i-1) + 1 WHEN x(i-1)='1' ELSE temp(i-1);
20     END GENERATE;
21     y <= conv_std_logic_vector(temp(bits_in), bits_out);
22  END ARCHITECTURE;
23  ---------------------------------------------------------------------------
```

Resultados de simulação do código acima constam na figura abaixo.



Exercício 19.13. Inferência de flip-flops

a) As diferenças entre um sinal e uma variável estão resumidas na tabela da figura 19.7. Flip-flops são inferidos de acordo com a regra 6.

b) De maneira resumida, a regra 6 diz que um sinal é "registrado" (isto é, armazenado em flip-flops) quando assinala-se um valor a ele na borda de um outro sinal. Por exemplo, no segmento de código abaixo,

```
...
IF clk'EVENT AND clk= '1' THEN
   x <= ...
...
```

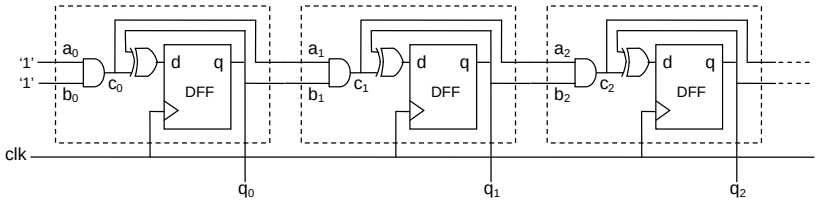
o sinal *x* será registrado (flip-flops serão inferidos) pois está-se assinalando um valor a ele na borda de outro sinal (*clk*).

c) Como há 1000 estados, $\lceil \log_2 1000 \rceil = 10$ DFFs são necessários para implementar esse contador.

d) Apenas substitua 9 e 10 por 999 e 1000, respectivamente, depois compile o código e verifique no relatório de compilação a quantidade de DFFs inferidos.

Exercício 19.15. Contador síncrono com componente

Para facilitar a análise, a figura 14.5(b) foi repetida abaixo, com os sinais usados no projeto marcados explicitamente. Somente a solução genérica é apresentada; para a solução pedida no item (a), simplesmente repita a linha 17 três vezes (com os índices apropriados nos sinais, obviamente, e com as linhas 18-20 atuais removidas). A solução genérica é obviamente superior.

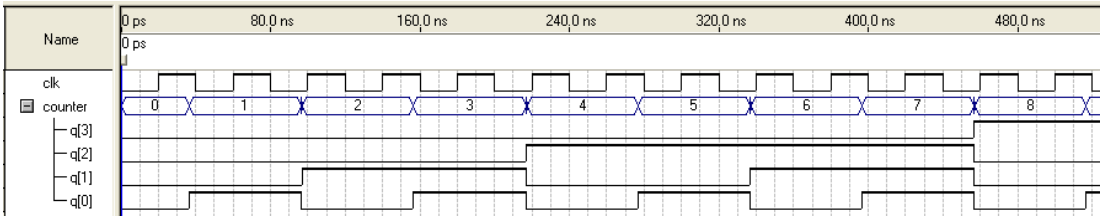



```
1   ---The component (cell of fig. 14.5b):------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   -------------------------------------------
5   ENTITY counter_cell IS
6   PORT (a, b, clk: IN STD_LOGIC;
7         c, q: BUFFER STD_LOGIC);
8   END ENTITY;
9   -------------------------------------------
10  ARCHITECTURE counter_cell OF counter_cell IS
11  BEGIN
12     PROCESS (a, b, clk)
13     BEGIN
14        c <= a AND b;
15        IF (clk'EVENT AND clk='1') THEN
16           q <= q XOR c;
17        END IF;
18     END PROCESS;
19  END ARCHITECTURE;
20  -------------------------------------------
```

```
1   ---Main code:-----------------------------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   -------------------------------------------
5   ENTITY counter IS
6      GENERIC (N: NATURAL := 4);
7      PORT (clk: IN STD_LOGIC;
8            q: BUFFER STD_LOGIC_VECTOR(0 TO N-1));
9   END ENTITY;
10  -----------------------------------------------------------------------------------------
11  ARCHITECTURE structural OF counter IS
12     SIGNAL a, b, c: STD_LOGIC_VECTOR(0 TO N-1);
13     ------------------------
14     COMPONENT counter_cell IS
15        PORT (a, b, clk: IN STD_LOGIC;
16              c, q: BUFFER STD_LOGIC);
17     END COMPONENT;
18     ------------------------
19  BEGIN
20     cell0: counter_cell PORT MAP (clk=>clk, a=>'1', b=>'1', c=>c(0), q=>q(0));
21     other_cells: FOR i IN 1 TO N-1 GENERATE
22        gen_cells: counter_cell PORT MAP (clk=>clk, a=>c(i-1), b=>q(i-1), c=>c(i), q=>q(i));
23     END GENERATE;
24  END ARCHITECTURE;
```

```
25  ------------------------------------------------------------------------------
```

Resultados de simulação do código acima constam na figura abaixo.



Capítulo 20: Projetos de Circuitos Combinacionais Lógicos com VHDL

Nota: Conforme mencionado, em todas as soluções adotaremos nas portas os tipos-padrão da indústria, quais sejam, STD_LOGIC e STD_LOGIC_VECTOR.

Exercício 20.1. Decodificador de endereço #1

Conforme requerido, a solução abaixo é para *N* fixo (=3). Uma solução para *N* genérico foi apresentada na página 474. Outra solução genérica, com apenas uma linha, consta no exercício 20.2, a seguir.

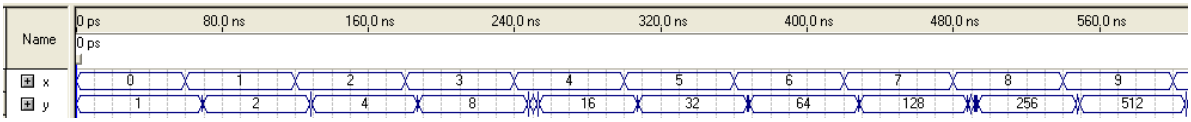
```
 1  ------------------------------------------------
 2  LIBRARY ieee;
 3  USE ieee.std_logic_1164.all;
 4  ------------------------------------------------
 5  ENTITY address_decoder IS
 6     PORT (x: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
 7           y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
 8  END address_decoder;
 9  ------------------------------------------------
10  ARCHITECTURE with_SELECT OF address_decoder IS
11  BEGIN
12     WITH x SELECT
13        y <= "00000001" WHEN "000",
14             "00000010" WHEN "001",
15             "00000100" WHEN "010",
16             "00001000" WHEN "011",
17             "00010000" WHEN "100",
18             "00100000" WHEN "101",
19             "01000000" WHEN "110",
20             "10000000" WHEN OTHERS;
21  END with_SELECT;
22  ------------------------------------------------
```

Exercício 20.2. Decodificador de endereço #2

Nesse caso, apenas uma linha de código é necessária na arquitetura (linha 15). Porem, packages adicionais precisaram ser declarados (linhas 4-5) para permitir a conversão de dados. Isso é necessário porque o operador aritmético de exponenciação (**) só existe para o tipo INTEGER (e seus subtipos, NATURAL e POSITIVE, obviamente).

```
 1  ------------------------------------------------------
 2  LIBRARY ieee;
 3  USE ieee.std_logic_1164.all;
 4  USE ieee.std_logic_unsigned.all; --conv. slv to integer
 5  USE ieee.std_logic_arith.all;    --conv. integer to slv
 6  ------------------------------------------------------
 7  ENTITY address_decoder IS
 8     GENERIC (N: NATURAL := 4);
 9     PORT (x: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
10           y: OUT STD_LOGIC_VECTOR(2**N-1 DOWNTO 0));
11  END address_decoder;
12  ------------------------------------------------------
13  ARCHITECTURE single_line OF address_decoder IS
14  BEGIN
15     y <= conv_std_logic_vector(2**conv_integer(x), 2**N);
16  END single_line;
17  ------------------------------------------------------
```

Resultados de simulação do código acima constam na figura abaixo.



Desafio: O leitor é convidado a apresentar um outro código para o problema acima com ainda menos linhas (removendo as linhas 1-5 e fazendo as modificações necessárias nas outras). É possível fazer isso enquanto ainda utilizando nas portas os tipos-padrão da indústria?

Exercício 20.3. Decodificador de endereço #3

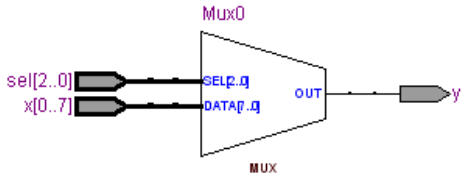
Outra solução genérica para o decodificador de endereço é apresentada abaixo. Conforme requerido, código sequencial foi utilizado. Comparando com as demais soluções genéricas apresentadas (página 474 e exercício 20.2), vê-se que essa é a mais longa.

```
1   ----------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   USE ieee.std_logic_arith.all; --conv. integer to slv
5   ----------------------------------------------------
6   ENTITY address_decoder IS
7      GENERIC (N: NATURAL := 4);
8      PORT (x: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
9            y: OUT STD_LOGIC_VECTOR(2**N-1 DOWNTO 0));
10  END address_decoder;
11  ----------------------------------------------------
12  ARCHITECTURE sequential OF address_decoder IS
13  BEGIN
14     PROCESS(x)
15     BEGIN
16        FOR i IN y'RANGE LOOP
17           IF conv_std_logic_vector(i,N)=x THEN
18              y(i) <= '1';
19           ELSE
20              y(i) <= '0';
21           END IF;
22        END LOOP;
23     END PROCESS;
24  END sequential;
25  ----------------------------------------------------
```

Exercício 20.5. Multiplexador #1

```
1   ------------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   USE ieee.std_logic_unsigned.all; --conv. slv to integer
5   ------------------------------------------------------------
6   ENTITY mux IS
7      GENERIC (inputs: POSITIVE := 8;     --number of inputs
8               sel_bits: POSITIVE := 3);  --number of bits in sel
9      PORT (x: IN STD_LOGIC_VECTOR (0 TO inputs-1);
10           sel: IN STD_LOGIC_VECTOR(sel_bits-1 DOWNTO 0);
11           y: OUT STD_LOGIC);
12  END ENTITY;
13  ------------------------------------------------------------
14  ARCHITECTURE mux OF mux IS
15  BEGIN
16     y <= x(conv_integer(sel));
17  END ARCHITECTURE;
18  ------------------------------------------------------------
```

O resultado apresentado pelo RTL Viewer após a compilação desse código é mostrado abaixo.

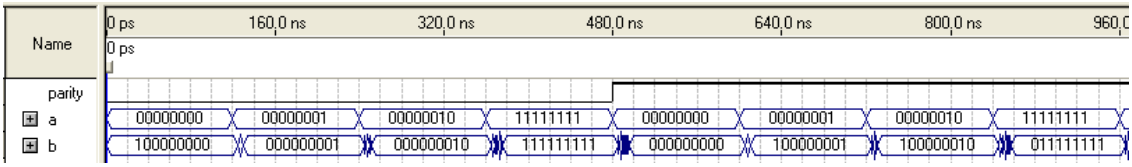


Exercício 20.7. Gerador de paridade

Nota: Como não é permitido assinalar valor a um sinal mais do que uma vez, na linha 13 criou-se um sinal com dimensão uma unidade maior do que a dimensão do sinal a monitorar/calcular (*temp* é 1D, enquanto o sinal a ser calculado tem apenas um bit, ou seja, é um escalar). Esse importantíssimo artifício para código concorrente foi introduzido na seção 7.7 da referência citada na Nota 2 no início desse documento.

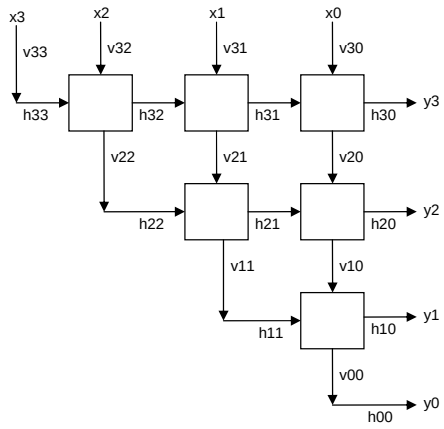
```
1   ------------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   ------------------------------------------------------------
5   ENTITY parity_generator IS
6      GENERIC (N: NATURAL := 8); --number of bits
7      PORT (a: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
8            parity: IN STD_LOGIC;
9            b: OUT STD_LOGIC_VECTOR(N DOWNTO 0));
10  END ENTITY;
11  ------------------------------------------------------------
12  ARCHITECTURE concurrent OF parity_generator IS
13     SIGNAL temp: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
14  BEGIN
15     temp(0) <= a(0);
16     gen: FOR i IN 1 TO N-1 GENERATE
17        temp(i) <= temp(i-1) XOR a(i);
18     END GENERATE;
19     b <= temp(N-1) & a WHEN parity='1' ELSE NOT temp(N-1) & a;
20  END ARCHITECTURE;
21  ------------------------------------------------------------
```

Resultados de simulação do código acima constam na figura abaixo.



Exercício 20.10. Ordenador binário #2

Essa é a solução genérica para o circuito dos exercícios 20.9 e 20.10. Para facilitar a análise do código, o circuito do ordenador binário foi repetido abaixo com todos os sinais usados no código explicitamente marcados.




```
1   ------------------------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   ------------------------------------------------------------------------
5   ENTITY binary_sorter IS
6      GENERIC (N: INTEGER := 5);
7      PORT (x: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
8            y: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
9   END binary_sorter;
10  ------------------------------------------------------------------------
11  ARCHITECTURE sorter OF binary_sorter IS
12  BEGIN
13     PROCESS(x)
14        TYPE matrix IS ARRAY (N-1 DOWNTO 0) OF STD_LOGIC_VECTOR (N-1 DOWNTO 0);
15        VARIABLE h, v: matrix; --internal horizontal and vertical signals
16     BEGIN
17        --Connect input x to top row cells:
```

```
18         FOR i IN N-1 DOWNTO 0 LOOP
19            v(N-1)(i) := x(i);
20         END LOOP;
21         --Get left diagonal:
22         FOR i IN N-1 DOWNTO 0 LOOP
23            h(i)(i) := v(i)(i);
24         END LOOP;
25         --Get internal horizontal & vertical signals:
26         FOR j IN N-2 DOWNTO 0 LOOP
27            FOR i IN j DOWNTO 0 LOOP
28               h(j+1)(i) := h(j+1)(i+1) OR v(j+1)(i);
29               v(j)(i) := h(j+1)(i+1) AND v(j+1)(i);
30            END LOOP;
31         END LOOP;
32         --Get output:
33         FOR i IN N-1 DOWNTO 0 LOOP
34            y(i) <= h(i)(0);
35         END LOOP;
36      END PROCESS;
37   END sorter;
38   ---------------------------------------------------------------------------
```

Capítulo 21: Projetos de Circuitos Combinacionais Aritméticos com VHDL

Nota: Conforme mencionado, em todas as soluções adotaremos nas portas os tipos-padrão da indústria, quais sejam, STD_LOGIC e STD_LOGIC_VECTOR.

Exercício 21.1. Incrementador

Observe que a estrutura do circuito da figura 12.14(b) é bem parecida com aquela do somador carry-ripple, mostrado na página 487, cujo código consta na página 488. Portanto, com base nesse código, um código para o incrementador pode facilmente ser escrito.

Exercício 21.3. Complementador de dois

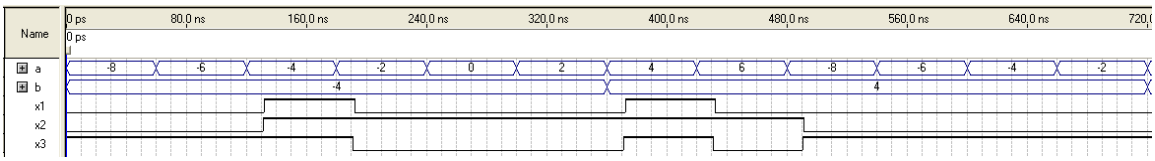
Simplesmente faça os ajustes necessários no código do exercício 21.1

Exercício 21.5. Comparador com sinal

Na solução abaixo, foi adotado o procedimento recomendado para quando trabalha-se com sinais *com signal* (conversão a SIGNED antes de iniciar qualquer operação).

```
1    -------------------------------------------------
2    LIBRARY ieee;
3    USE ieee.std_logic_1164.all;
4    USE ieee.numeric_std.all; --for type SIGNED
5    -------------------------------------------------
6    ENTITY sig_comparator IS
7       GENERIC (N: INTEGER := 4);
8       PORT (a, b: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
9             x1, x2, x3: OUT STD_LOGIC);
10   END sig_comparator;
11   -------------------------------------------------
12   ARCHITECTURE comparator OF sig_comparator IS
13      SIGNAL a_sig, b_sig: SIGNED(N-1 DOWNTO 0);
14   BEGIN
15      a_sig <= signed(a);
16      b_sig <= signed(b);
17      x1 <= '1' WHEN a_sig = b_sig ELSE '0';
18      x2 <= '1' WHEN a_sig >= b_sig ELSE '0';
19      x3 <= '1' WHEN a_sig <= b_sig ELSE '0';
20   END comparator;
21   -------------------------------------------------
```

Resultados de simulação do código acima constam na figura abaixo.



Exercício 21.7. Somadores/subtratores com sinal e sem sinal #1

A solução recomendada, isto é, (i) com I/Os de tamanho genérico, (ii) com I/Os todos do tipo STD_LOGIC_VECTOR e (iii) com conversão a SIGNED antes de iniciar as operações, já foi apresentada na seção 21.3 (código na página 492). A partir dela, a implementação do código pedido neste exercício é simples e direta.

Exercício 21.9. Multiplicadores/divisores com sinal e sem sinal #1

A solução recomendada, isto é, (i) com I/Os de tamanho genérico, (ii) com I/Os todos do tipo STD_LOGIC_VECTOR e (iii) com conversão a SIGNED antes de iniciar as operações, já foi apresentada na seção 21.4 (código na página 494). A partir dela, a implementação do código pedido neste exercício é simples e direta.

Exercício 21.11. ALU

a) Nas linhas 17-32, os vetores binários "0000", "0001", "0010", etc. deveriam ser substituídos pelos valores decimais 0, 1, 2, etc.
b) A operações aritméticas nas linhas 26-33 não seriam permitidas.
c) O pacote *std_logic_signed* produziria os mesmos resultados.

Capítulo 22: Projetos de Circuitos Sequenciais com VHDL

Nota: Conforme mencionado, em todas as soluções adotaremos nas portas os tipos-padrão da indústria, quais sejam, STD_LOGIC e STD_LOGIC_VECTOR.

Exercício 22.1. Linha de retardo programável

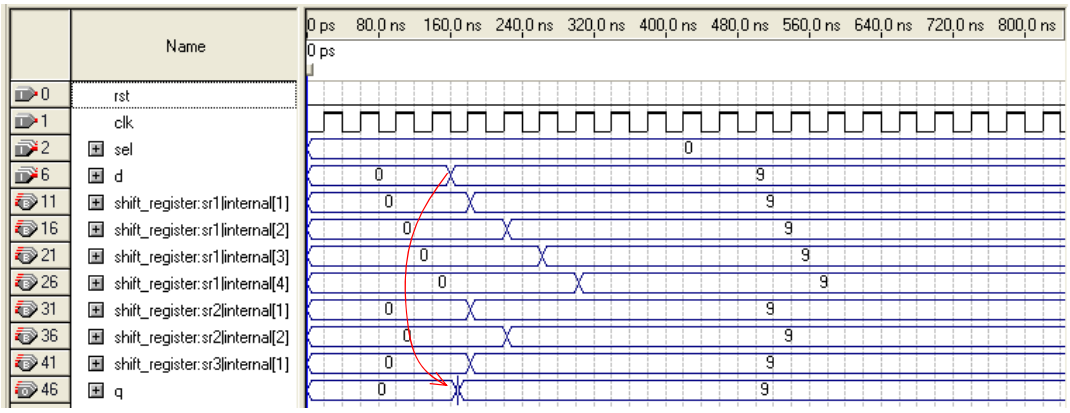
Estruturalmente, essa solução é semelhante àquela apresentada para o shift register na seção 22.1.

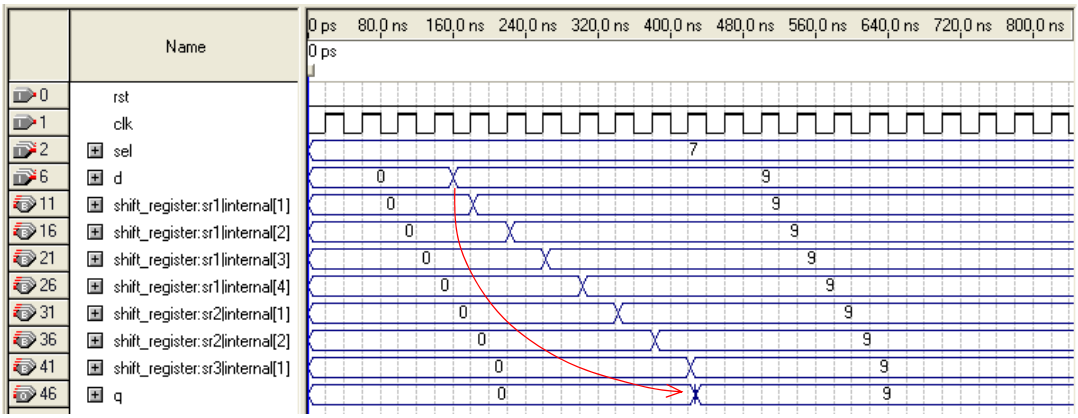
```
1   ----1st component: shift register-----------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   -------------------------------------------------------
5   ENTITY shift_register IS
6      GENERIC (bits: POSITIVE;      --number of bits
7               stages: POSITIVE);   --number of stages
8      PORT (d: IN STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
9            clk, rst: IN STD_LOGIC;
10           q: OUT STD_LOGIC_VECTOR(bits-1 DOWNTO 0));
11  END ENTITY;
12  -------------------------------------------------------
13  ARCHITECTURE shift_register OF shift_register IS
14     TYPE matrix IS ARRAY (1 TO stages) OF
15        STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
16     SIGNAL internal: matrix;
17  BEGIN
18     PROCESS (clk, rst)
19     BEGIN
20        IF (rst='1') THEN
21           FOR i IN 1 TO stages LOOP
22              internal(i) <= (OTHERS => '0');
23           END LOOP;
24        ELSIF (clk'EVENT AND clk='1') THEN
25           internal <= d & internal(1 TO stages-1);
26        END IF;
27     END PROCESS;
28     q <= internal(stages);
29  END ARCHITECTURE;
30  -------------------------------------------------------
```

```
1   -----2nd component: multiplexer-------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   -------------------------------------------------------
5   ENTITY multiplexer IS
6      GENERIC (bits: POSITIVE);  --number of bits
7      PORT (inp1, inp2: IN STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
8            sel: IN STD_LOGIC;
9            outp: OUT STD_LOGIC_VECTOR(bits-1 DOWNTO 0));
10  END ENTITY;
11  -------------------------------------------------------
12  ARCHITECTURE multiplexer OF multiplexer IS
13  BEGIN
14     outp <= inp1 WHEN sel='0' ELSE inp2;
15  END ARCHITECTURE;
16  -------------------------------------------------------
```

```
1   ----Main code---------------------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   ------------------------------------------------------------------------------------
5   ENTITY tapped_delay_line IS
6      GENERIC (N: POSITIVE := 4);  --number of bits
7      PORT (clk, rst: IN STD_LOGIC;
8            d: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
9            sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
10           q: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
11  END tapped_delay_line;
12  ------------------------------------------------------------------------------------
13  ARCHITECTURE structural OF tapped_delay_line IS
14     ----Signal declarations:-----------
15     SIGNAL sr1_out, mux1_out: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
16     SIGNAL sr2_out, mux2_out: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
17     SIGNAL sr3_out, mux3_out: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
18     ----1st component declaration:-----
19     COMPONENT shift_register IS
20        GENERIC (bits: POSITIVE;
21                 stages: POSITIVE);
22        PORT (d: IN STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
23              clk, rst: IN STD_LOGIC;
24              q: OUT STD_LOGIC_VECTOR(bits-1 DOWNTO 0));
25     END COMPONENT;
26     ----2nd component declaration:-----
27     COMPONENT multiplexer IS
28        GENERIC (bits: POSITIVE);
29        PORT (inp1, inp2: IN STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
30              sel: IN STD_LOGIC;
31              outp: OUT STD_LOGIC_VECTOR(bits-1 DOWNTO 0));
32     END COMPONENT;
33     ----------------------------------
34  BEGIN
35     sr1: shift_register GENERIC MAP (N, 4) PORT MAP (d, clk, rst, sr1_out);
36     mux1: multiplexer GENERIC MAP (N) PORT MAP (d, sr1_out, sel(2), mux1_out);
37     sr2: shift_register GENERIC MAP (N, 2) PORT MAP (mux1_out, clk, rst, sr2_out);
38     mux2: multiplexer GENERIC MAP (N) PORT MAP (mux1_out, sr2_out, sel(1), mux2_out);
39     sr3: shift_register GENERIC MAP (N, 1) PORT MAP (mux2_out, clk, rst, sr3_out);
40     mux3: multiplexer GENERIC MAP (N) PORT MAP (mux2_out, sr3_out, sel(0), mux3_out);
41     q <= mux3_out;
42  END structural;
43  ------------------------------------------------------------------------------------
```

Resultados de simulação do código acima constam nas figuras abaixo, para *sel*=0 (mínimo retardo, isto é, zero) e *sel*=7 (máximo retardo, ou seja, 7 bordas positivas do clock).





Exercício 22.3. Gerador de sequência pseudorandômica

```
1   ------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   ------------------------------------------------
5   ENTITY pseudo_random IS
6      PORT (clk, rst: IN STD_LOGIC;
7            d: BUFFER STD_LOGIC);
8   END ENTITY;
9   ------------------------------------------------
10  ARCHITECTURE shift_register OF pseudo_random IS
11     SIGNAL q: STD_LOGIC_VECTOR(1 TO 4);
12  BEGIN
13     PROCESS (clk, rst)
14     BEGIN
15        IF (rst='1') THEN
16           q <= "1111";
17        ELSIF (clk'EVENT AND clk='1') THEN
18           q <= d & q(1 TO 3);
19        END IF;
20     END PROCESS;
21     d <= q(3) XOR q(4);
22  END ARCHITECTURE;
23  ------------------------------------------------
```

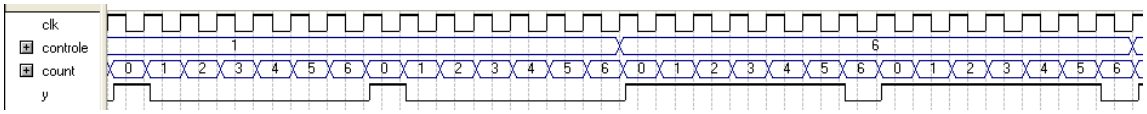
Exercício 22.5. PWM digital

Abaixo consta um código *genérico* para esse circuito (basta trocar o valor de *N* na linha 7 para obter um PWM de qualquer tamanho). Na simulação mostrada após o código foi utilizado *N*=3. O número total de pulsos do circuito é sempre 2^N-1, podendo o sinal de controle ter qualquer valor desde zero até esse limite.

```
1   ------------------------------------------------------
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   USE ieee.std_logic_unsigned.all; --conv. slv to int
5   ------------------------------------------------------
6   ENTITY pwm IS
7      GENERIC (N: NATURAL := 3); -- # of control bits
8      PORT (clk: IN STD_LOGIC;
9            controle: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
10           y: OUT STD_LOGIC);
11  END pwm;
12  ------------------------------------------------------
13  ARCHITECTURE generic_pwm OF pwm IS
14  BEGIN
15     PROCESS (clk)
16        VARIABLE count: NATURAL RANGE 0 TO 2**N-1;
17     BEGIN
18        IF (clk'EVENT AND clk='1') THEN
19           --Independent counter:
20           count := count + 1;
21           IF (count=2**N-1) THEN
22              count := 0;
23           END IF;
24           --PWM cell:
25           IF (count < conv_integer(controle)) THEN
26              y <= '1';
27           ELSE
28              y <= '0';
```

```
29          END IF;
30       END IF;
31    END PROCESS;
32 END generic_pwm;
33 ---------------------------------------------------
```

Resultados de simulação deste código constam na figura abaixo, para *controle*=1 e *controle*=6.



Exercício 22.7. Temporizador #1

O código completo para esse timer consta abaixo. Lembre que *ena* é ativo baixo. Os SSDs foram considerados do tipo anodo comum (veja figura 11.13(c)).

```
1  ---------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------------
5  ENTITY timer IS
6     GENERIC (fclk: POSITIVE := 50_000_000); --clock freq.
7     PORT (clk, ena: IN STD_LOGIC;
8         full_count: OUT STD_LOGIC;
9         dig1, dig2: OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
10 END ENTITY;
11 ---------------------------------------------------
12 ARCHITECTURE timer OF timer IS
13    SIGNAL reset, status: STD_LOGIC;
14 BEGIN
15    ---Process for enable and reset:-----
16    PROCESS(clk, ena)
17       VARIABLE count0: INTEGER RANGE 0 TO 2*fclk;
18       VARIABLE flag: STD_LOGIC;
19    BEGIN
20       IF (clk'EVENT AND clk='1') THEN
21          IF (ena='0') THEN
22             flag := '1';
23             count0 := count0 + 1;
24             IF (count0=2*fclk) THEN
25                reset <= '1';
26             END IF;
27          ELSE
28             count0 := 0;
29             IF (flag='1') THEN
30                status <= NOT status;
31                flag := '0';
32                reset <= '0';
33             END IF;
34          END IF;
35       END IF;
36    END PROCESS;
37    ---Process for the timer:------------
38    PROCESS(clk, reset)
39       VARIABLE count1: INTEGER RANGE 0 TO fclk; --for 1Hz
40       VARIABLE count2: INTEGER RANGE 0 TO 10;   --for dig1
41       VARIABLE count3: INTEGER RANGE 0 TO 7;    --for dig2
42    BEGIN
43       ---Counters:
44       IF (reset='1') THEN
45          count1 := 0;
46          count2 := 0;
47          count3 := 0;
48          full_count <= '0';
49       ELSIF (clk'EVENT AND clk='1') THEN
50          IF (count3=6) THEN
51             full_count <= '1';
52          ELSE
53             full_count <= '0';
54             IF (status='1') THEN
55                count1 := count1 + 1;
56                IF (count1=fclk) THEN
57                   count1 := 0;
58                   count2 := count2 + 1;
59                   IF (count2=10) THEN
60                      count2 := 0;
61                      count3 := count3 + 1;
62                   END IF;
```

```
63                END IF;
64             END IF;
65          END IF;
66       END IF;
67       ---BCD to SSD conversion:
68       CASE count2 IS
69          WHEN 0 => dig1 <= "0000001";
70          WHEN 1 => dig1 <= "1001111";
71          WHEN 2 => dig1 <= "0010010";
72          WHEN 3 => dig1 <= "0000110";
73          WHEN 4 => dig1 <= "1001100";
74          WHEN 5 => dig1 <= "0100100";
75          WHEN 6 => dig1 <= "0100000";
76          WHEN 7 => dig1 <= "0001111";
77          WHEN 8 => dig1 <= "0000000";
78          WHEN 9 => dig1 <= "0000100";
79          WHEN OTHERS => dig1 <= "0110000"; --"E"rror
80       END CASE;
81       CASE count3 IS
82          WHEN 0 => dig2 <= "0000001";
83          WHEN 1 => dig2 <= "1001111";
84          WHEN 2 => dig2 <= "0010010";
85          WHEN 3 => dig2 <= "0000110";
86          WHEN 4 => dig2 <= "1001100";
87          WHEN 5 => dig2 <= "0100100";
88          WHEN 6 => dig <= "0100000";
89          WHEN OTHERS => dig2 <= "0110000"; --"E"rror
90       END CASE;
91    END PROCESS;
92 END ARCHITECTURE;
93 -------------------------------------------------
```

Exercício 22.9. Temporizador #3

Precisamos introduzir outro contador para fazer *full_count* piscar a cada 0,5 s. Isso pode ser feito com uma nova variable (*count4*) no segundo processo da solução acima, conforme ilustrado no segmento de código abaixo.

```
-----------------------------------------
...
VARIABLE count4: NATURAL RANGE 0 TO fclk/4;
...
ELSIF (clk'EVENT AND clk='1') THEN
   IF (count3=6) THEN
      count4 := count4 + 1;
      IF (count4=fclk/4) THEN
         full_count <= NOT full_count;
         count4 := 0;
      END IF;
...
-----------------------------------------
```

Exercício 22.11. Frequencímetro com contador Gray

Basta tomar o código VHDL da seção 22.5 e transformar a sua saída, *fx*, de código binário sequencial para código Gray. Para tal, utilize as expressões de conversão dadas no final da seção 2.3, nas quais são necessários apenas portas XOR (total de $N-1$ portas XOR, onde N é o número de bits de *fx*).

Capítulo 23: Projetos de Máquinas de Estados com VHDL

Nota: Sugere-se utilizar a referência [1] abaixo em substituição ao Capítulo 23 (assim como ao Capítulo 15).

[1] V. A. Pedroni, *Finite State Machines in Hardware: Theory and Design (with VHDL and SystemVerilog)*, MIT Press, Dec. 2013.
