Sistemas Digitais 2

Projeto estrutural
Declaração e instanciação de componentes

Prof. Daniel M. Muñoz Arboleda

FGA - UnB

Agenda

- Definição de componente
- Vantagens do uso de componentes
- Declaração de componentes
- Instanciação de componentes PORT MAP
- Instanciação de componentes GENERIC MAP
- Instanciação de componentes PORT MAP e GENERATE
- Exemplos
  - Circuito detector de paridade de 3 bits
  - Circuito gerador de paridade com GENERIC MAP
  - Implementação de uma ULA
  - Registrador de deslocamento usando PORT MAP e GENERATE
  - Video aula: contador up/down de 4 bits com divisor de clock de 1 segundo e decodificador binário para 7 segmentos (componentes)

- Circuitos digitais implementados em VHDL podem ser descritos de forma estrutural e/ou comportamental
- A descrição estrutural define as estruturas de interconexão entre os módulos (como um esquemático ou um *netlist*). De forma geral, a descrição estrutural está baseada em **componentes**.
- A descrição comportamental está baseada em processos (conjunto de instruções sequencias) e/ou atribuições concorrentes.

Descrição estrutural
Declaração de entidade
Declaração de arquitetura
Declaração de sinais
Declaração de componentes
Instanciação de componentes

Descrição comportamental
Atribuição de sinais
Atribuição concorrente de sinais
Processos
Procedures e *Funções*

Componente

- Componentes são uma porção de código (*Library+Entity+Architecture*)
- Componentes são entidades descritas separadamente e que podem ser organizadas em bibliotecas.
- Uma vez testados e validados, os componentes podem ser usados em diversos projetos por diferentes pessoas.

Vantagens do uso de componentes

- O uso de componentes permite a construção de projetos hierárquicos.
- Permite particionar o código de forma que as equipes de trabalho compartilhem e reusem código.
- Podem ser utilizados em diversos circuitos e projetos sem precisar reescrever o código de forma explícita.

Declaração de componentes

- Para usar (instanciar) um componente o mesmo deve ser declarado.
- Os componentes devem ser declarados entre a declaração da *arquitetura* e o *begin* da *arquitetura* **ou** em *packages*.
- A sintaxe para declaração de componentes é mostrada abaixo

```
COMPONENT component_name IS
    PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END COMPONENT;
```

- Observe-se que a declaração de componentes é similar à declaração de entidades. Deve especificar o nomes das portas, os modos (*in*, *out*, *inout*, *buffer*, etc) e tipo de dados (*std_logic_vector*, *bitvector*, *integer*, *boolean*, etc).

Instanciação de componentes: diretiva PORT MAP

- A seguir mostra-se a sintaxe da diretiva PORT MAP.
- A lista de portas especifica como são conectadas (mapeadas) as entradas e saídas do componente.

```
label: component_name PORT MAP (port_list);
```

- Existem duas formas para mapear as portas de um componente durante a sua instanciação: *positional mapping* e *nominal mapping*.

Instanciação de componentes: diretiva PORT MAP

- Considere o componente da porta inversora:

```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
```

- *Positional mapping*: `U1: inverter PORT MAP (x, y);`
- *Nominal mapping*: `U1: inverter PORT MAP (x=>a, y=>b);`

No mapeamento posicional, as portas *x* e *y* correspondem com *a* e *b*, respectivamente. O mapeamento posicional é mais fácil de escrever, porém o mapeamento nominal é menos propenso a erros.

- As portas também podem se deixar desconectadas (OPEN), por exemplo:

```
U2: my_circuit PORT MAP (x=>a, y=>b, w=>OPEN, z=>d);
```

Instanciação de componentes: GENERIC MAP e PORT MAP

- Quando o componentes possui parâmetros genéricos, deve-se usar a sequinte sintaxe:

```
parameter_rom: read_only_memory
   generic map (data_bits => 16, addr_bits => 8);
   port map (en => rom_sel, data => param, addr => a(7 downto 0);
```

Instanciação de componentes: PORT MAP e GENERATE

- A diretiva GENERATE serve para criar múltiplas instancias do mesmo código concorrente. GENERATE aceita condicionais IF-THEN e FOR LOOP
- A seguir mostra-se a sintaxe da diretiva FOR – GENERATE:

```
label: FOR identifier IN range GENERATE
  (concurrent assignments)
END GENERATE;
```

- Quando a atribuição concorrente é um PORT MAP, o GENERATE serve para replicar estruturas (útil para representar arquiteturas **regulares**).

```
g1 : FOR i IN 0 TO 3 GENERATE
 dffx : dff PORT MAP( z(i), clk, z(i + 1));
END GENERATE;
```

Exemplo1: circuito de detecção de paridade de 3 bits

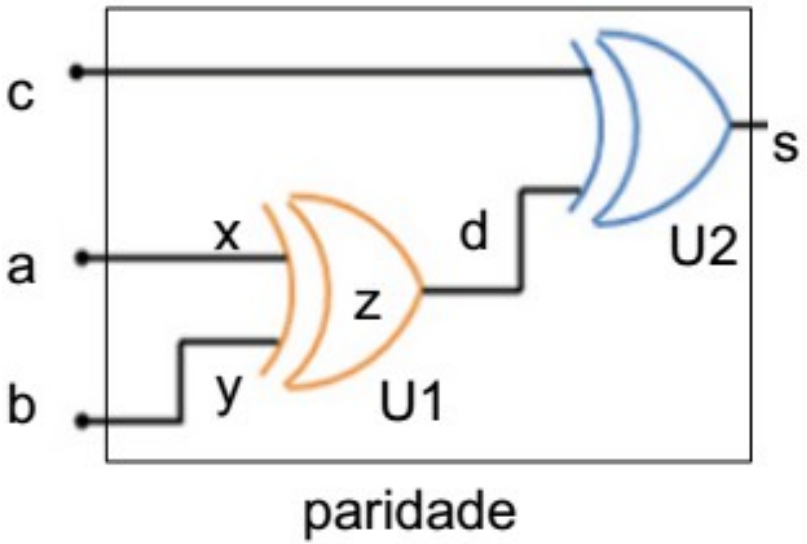
•A saída é '1' se houver um número impar de entradas iguais a '1'. O circuito será descrito instanciando componentes de portas ou-exclusivo.
• Porta ou-exclusivo de 2 entradas:

```
entity xor2 is
Port( x:  in std_logic;
      y:  in std_logic;
      z: out std_logic);
end xor2;

architecture Behavioral of xor2 is
begin
     z <= ((not x) and y) or (x and not(y));
end Behavioral;
```

Exemplo1: circuito de detecção de paridade de 3 bits

- O circuito será descrito instanciando **componentes** de portas ou-exclusivo.




```
entity paridade is
port(a,b,c :  in std_logic;
       s : out std_logic);
end paridade;

architecture arch_paridade of paridade is
      component xor2 is
            port(x,y: in std_logic;
                  z: out std_logic);
      end component;
      signal d: std_logic;
begin
      U1: xor2 PORT MAP(a,b,d);
      U2: xor2 PORT MAP(c,d,s);
end arch_paridade;
```

Exemplo2: circuito gerador de paridade com GENERIC MAP

• O circuito gerador de paridade adiciona um bit no vetor de entrada. Esse bit é '0' se o número de '1's no vetor de entrada é par, ou '1' se o número de '1's é impar. Dessa forma o vetor resultante sempre terá um número para de '1's.

• Primeiro será apresentada a arquitetura genérica (GENERIC) do circuito gerador de paridade.

• Em seguida essa arquitetura será instanciada como um **componente.**

Exemplo2: circuito gerador de paridade com GENERIC MAP

```
5  ENTITY parity_gen IS
6     GENERIC (n : INTEGER := 7);  -- default is 7
7     PORT ( input: IN BIT_VECTOR (n DOWNTO 0);
8            output: OUT BIT_VECTOR (n+1 DOWNTO 0));
9  END parity_gen;
10 ---------------------------------
11 ARCHITECTURE parity OF parity_gen IS
12 BEGIN
13    PROCESS (input)
14       VARIABLE temp1: BIT;
15       VARIABLE temp2: BIT_VECTOR (output'RANGE);
16    BEGIN
17       temp1 := '0';
18       FOR i IN input'RANGE LOOP
19          temp1 := temp1 XOR input(i);
20          temp2(i) := input(i);
21       END LOOP;
22       temp2(output'HIGH) := temp1;
23       output <= temp2;
24    END PROCESS;
25 END parity;
```

Arquitetura genérica chamada *parity_gen* do circuito gerador de paridade.

Mudando o valor de *n* modifica-se o tamanho do vetor de entrada e saída do gerador de paridade.

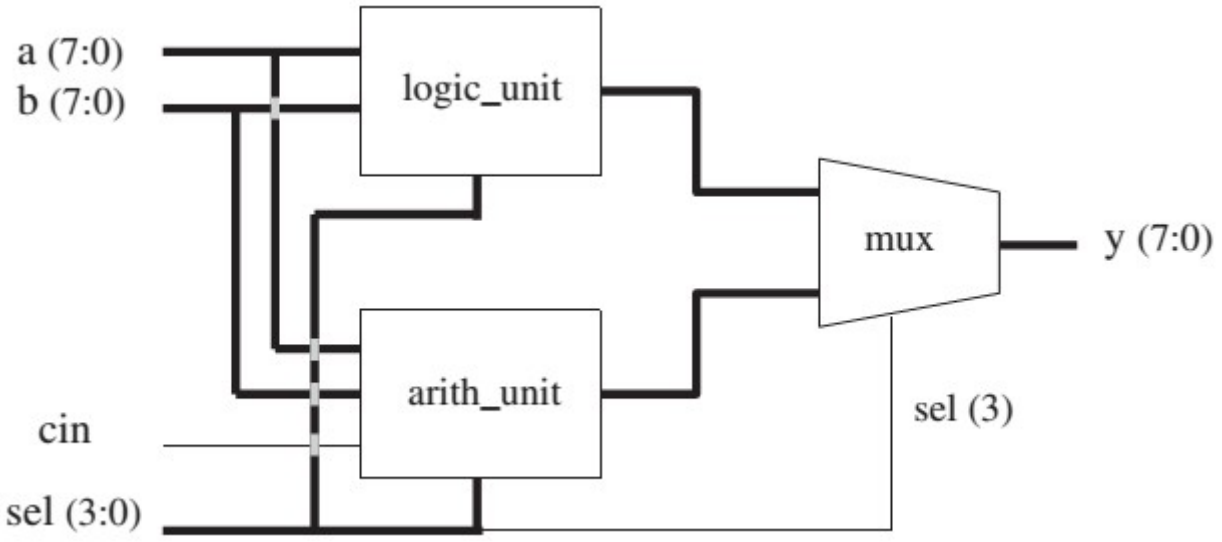
Exemplo2: circuito gerador de paridade com GENERIC MAP

```
1  ------ File my_code.vhd (actual project): ------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------
5  ENTITY my_code IS
6     GENERIC (n : POSITIVE := 2); -- 2 will overwrite 7
7     PORT ( inp: IN BIT_VECTOR (n DOWNTO 0);
8            outp: OUT BIT_VECTOR (n+1 DOWNTO 0));
9  END my_code;
10 ---------------------------------
11 ARCHITECTURE my_arch OF my_code IS
12 ------------------------
13    COMPONENT parity_gen IS
14       GENERIC (n : POSITIVE);
15       PORT (input: IN BIT_VECTOR (n DOWNTO 0);
16             output: OUT BIT_VECTOR (n+1 DOWNTO 0));
17    END COMPONENT;
18 ------------------------
19 BEGIN
20    C1: parity_gen GENERIC MAP(n) PORT MAP(inp, outp);
21 END my_arch;
22 -------------------------------------------------
```

Instanciação do componente *parity_gen*. Mudando o valor de *n* pode-se configurar o tamanho do circuito.

Exemplo3: implementação de uma ULA

- A seguir mostra-se a arquitetura de uma ULA (unidade lógica aritmética) de 8 bits.
- A ULA está constituída de uma unidade lógica e uma unidade aritmética.
- A operação é feita a partir da entrada de seleção sel de 4 bits.
- Um multiplexador é usado na saída para direcionar a operação selecionada.



Exemplo3: implementação de uma ULA

- As operações da ULA são as seguintes:

sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

Observe que o bit mais significativo de *sel* escolhe se será feita uma operação aritmética ou uma operação lógica.

Se sel(3)='0' então é feita uma operação aritmética

Se sel(3)='1' então é feita uma operação lógica

```
1  -------- COMPONENT arith_unit: -------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_unsigned.all;
5  ---------------------------------------
6  ENTITY arith_unit IS
7     PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8            sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
9            cin: IN STD_LOGIC;
10           x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END arith_unit;
12 ---------------------------------------
13 ARCHITECTURE arith_unit OF arith_unit IS
14    SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16    WITH sel SELECT
17       x <=  a WHEN "000",
18             a+1 WHEN "001",
19             a-1 WHEN "010",
20             b WHEN "011",
21             b+1 WHEN "100",
22             b-1 WHEN "101",
23             a+b WHEN "110",
24             a+b+cin WHEN OTHERS;
25 END arith_unit;
26 -----------------------------------------------
```

Implementação da unidade aritmética *arith_unit*.

Operações:

- *a*
- incremento em *a*
- decremento em *a*
- *b*
- incremento em *b*
- decremento em *b*
- *a*+*b*
- *a*+*b*+*carry_in*

```
1  -------- COMPONENT logic_unit: -------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY logic_unit IS
6     PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8            x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END logic_unit;
10 ----------------------------------------
11 ARCHITECTURE logic_unit OF logic_unit IS
12 BEGIN
13    WITH sel SELECT
14       x <=  NOT a WHEN "000",
15             NOT b WHEN "001",
16             a AND b WHEN "010",
17             a OR b WHEN "011",
18             a NAND b WHEN "100",
19             a NOR b WHEN "101",
20             a XOR b WHEN "110",
21             NOT (a XOR b) WHEN OTHERS;
22 END logic_unit;
23 ----------------------------------------------
```

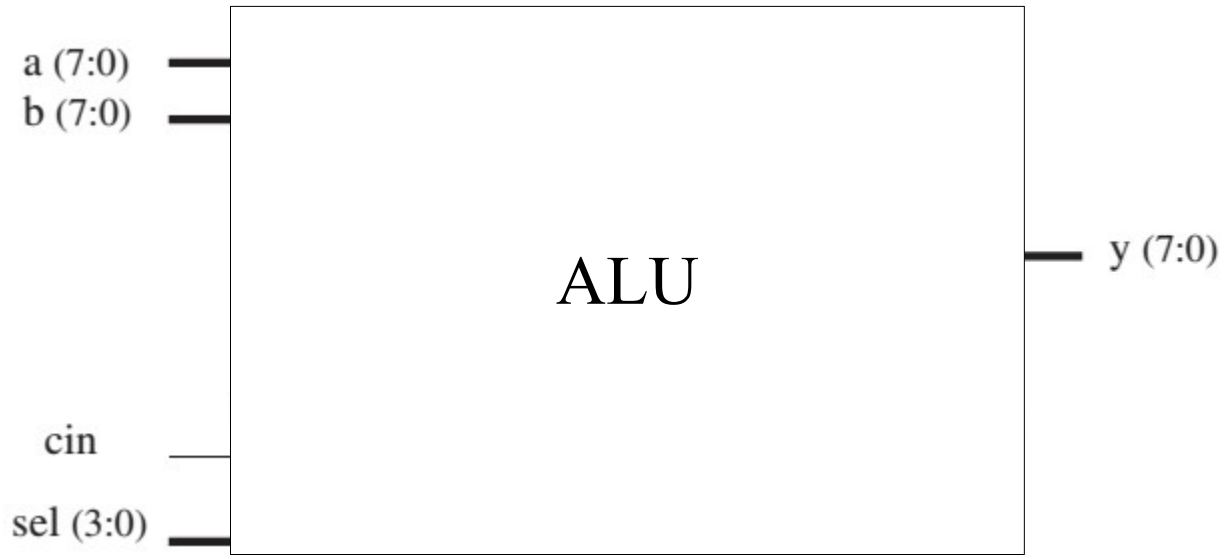
Implementação da unidade lógica *logic_unit*.

Operações:
- not *a*
- not *b*
- *a* and *b*
- *a* or *b*
- *a* nand *b*
- *a* nor *b*
- *a* xor *b*
- *a* xnor *b*

```
1  -------- COMPONENT mux: ---------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY mux IS
6     PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            sel: IN STD_LOGIC;
8            x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END mux;
10 ----------------------------------------
11 ARCHITECTURE mux OF mux IS
12 BEGIN
13    WITH sel SELECT
14       x <=   a WHEN '0',
15              b WHEN OTHERS;
16 END mux;
17 ------------------------------------------------
```

Implementação do multiplexador *mux*.

```
1  -------- Project ALU (main code): -----------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY alu IS
6     PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7            cin: IN STD_LOGIC;
8            sel: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9            y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
10 END alu;
```



Implementação da ULA:

declaração da entidade
Entradas:
- *a* e *b* de 8 bits
- *cin* (carry in) de 1 bit
- *sel* de 4 bits (seleção)

Saída:
- *y* de 8 bits

```
11 ----------------------------------------
12 ARCHITECTURE alu OF alu IS
13 ----------------------
14    COMPONENT arith_unit IS
15    PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
16           cin: IN STD_LOGIC;
17           sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
18           x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
19    END COMPONENT;
20    ----------------------
21    COMPONENT logic_unit IS
22    PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
23           sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
24           x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
25    END COMPONENT;
26    ----------------------
27    COMPONENT mux IS
28    PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
29           sel: IN STD_LOGIC;
30           x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
31    END COMPONENT;
32    ----------------------
33    SIGNAL x1, x2: STD_LOGIC_VECTOR(7 DOWNTO 0);
34 ----------------------
```

Implementação da ULA:

declaração de componentes
- *arith_unit*
- *logic_unit*
- *mux*

Declaração de sinais intermediários:
- *x*1 : vetor de 8 bits
- *x*2 : vetor de 8 bits

```
35 BEGIN
36    U1: arith_unit PORT MAP (a, b, cin, sel(2 DOWNTO 0), x1);
37    U2: logic_unit PORT MAP (a, b, sel(2 DOWNTO 0), x2);
38    U3: mux PORT MAP (x1, x2, sel(3), y);
39 END alu;
40 -------------------------------------------------
```

Implementação da ULA: **instanciação dos componentes**

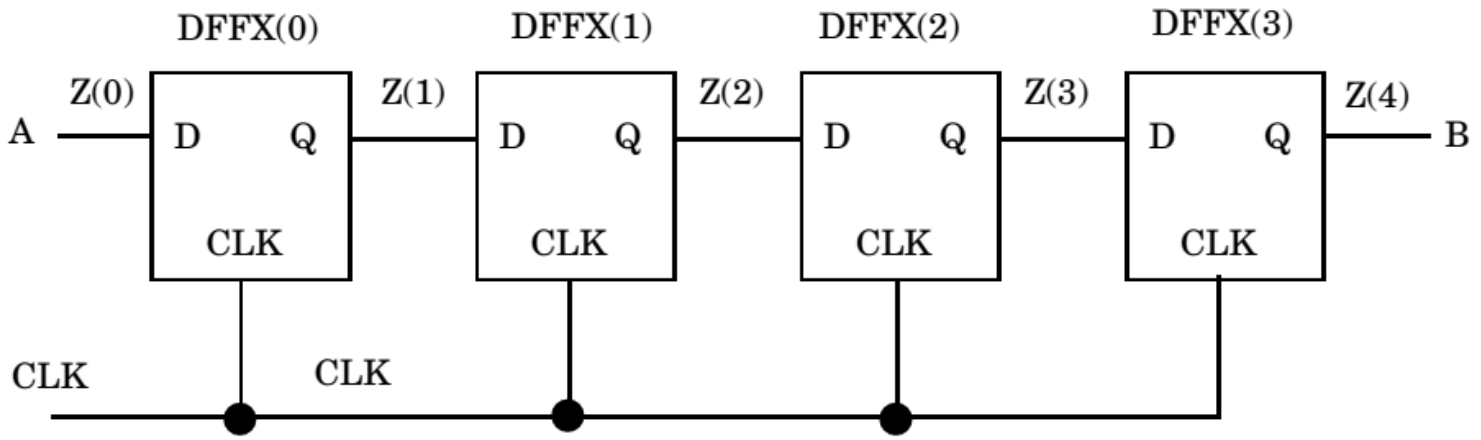
- *arith_unit*
- *logic_unit*
- *mux*

Observe que a saída do componente arith_unit vai para o sinal *x*1 e que a saída do componente *logic_unit* vai para o sinal *x*2.

Observe que as entradas do componente *mux* são *x*1, *x*2 e o bit mais significativo do vetor de seleção *sel*(3). A saída do *mux* vai para a saída geral *y*.

Exemplo 4: shift register usando PORT MAP e GENERATE

A seguir mostra-se a arquitetura de um registrador de deslocamento SISO de 4 bits baseado na réplica do componente DFF, o qual é um flip-flop tipo D.



Exemplo 4: shift register usando PORT MAP e GENERATE

- Registrador de deslocamento de 4 bits usando PORT MAP e GENERATE

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY shift IS
 PORT( a, clk : IN std_logic;
       b :  OUT std_logic);
END shift;

ARCHITECTURE gen_shift OF shift IS
 COMPONENT dff
  PORT( d, clk : IN std_logic;
        q : OUT std_logic);
 END COMPONENT;

 SIGNAL z : std_logic_vector( 0 TO 4 );
BEGIN
 z(0) <= a;

 g1 : FOR i IN 0 TO 3 GENERATE
  dffx : dff PORT MAP( z(i), clk, z(i + 1));
 END GENERATE;

 b <= z(4);
END gen_shift;
```

Exemplo 4: shift register usando PORT MAP e GENERATE

Registrador de deslocamento de 4 bits usando PORT MAP e GENERATE

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY shift IS
 PORT( a, clk : IN std_logic;
       b :  OUT std_logic);
END shift;

ARCHITECTURE gen_shift OF shift IS
 COMPONENT dff
  PORT( d, clk : IN std_logic;
        q : OUT std_logic);
 END COMPONENT;

 SIGNAL z : std_logic_vector( 0 TO 4 );
BEGIN
 z(0) <= a;

 g1 : FOR i IN 0 TO 3 GENERATE
  dffx : dff PORT MAP( z(i), clk, z(i + 1));
 END GENERATE;

 b <= z(4);
END gen_shift;
```

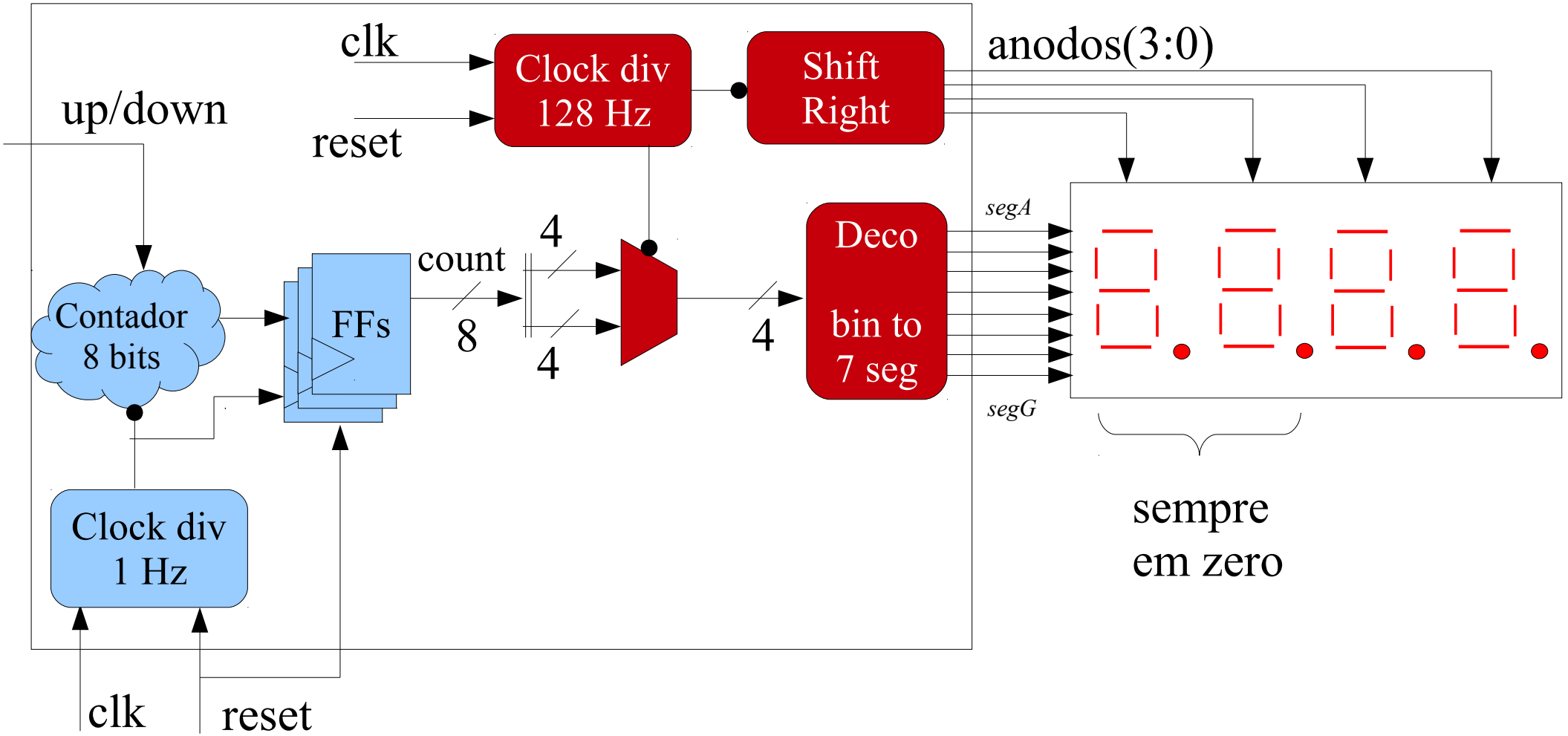
```
ARCHITECTURE long_way_shift OF shift IS
 COMPONENT dff
  PORT( d, clk : IN std_logic;
        q : OUT std_logic);
 END COMPONENT;

 SIGNAL z : std_logic_vector( 0 TO 4 );
BEGIN
 z(0) <= a;

 dff1: dff PORT MAP( z(0), clk, z(1) );
 dff2: dff PORT MAP( z(1), clk, z(2) );
 dff3: dff PORT MAP( z(2), clk, z(3) );
 dff4: dff PORT MAP( z(3), clk, z(4) );
```

Representações equivalentes!

Exemplo 5: contador up/down a 1seg com decodificador 7 segmentos



```
entity divisor_clk is
    generic (preset : std_logic_vector(24 downto 0):= (others=>'0'));
    port ( reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           outclk : out  STD_LOGIC);
end divisor_clk;

architecture Behavioral of divisor_clk is
signal count : std_logic_vector(24 downto 0) := (others=>'0');
signal clkaux : std_logic := '0';
begin

outclk <= clkaux;
process(clk,reset)
begin
      if rising_edge(clk) then
            if reset='1' then
                  count <= (others=>'0');
                  clkaux <= '0';
            elsif count=preset then
                  clkaux <= not clkaux;
                  count <= (others=>'0');
            else
                  count <= count + '1';
            end if;
       end if;
end process;
end Behavioral;
```

Exemplo 5 contador updown com deco7seg

Implementação do divisor de clock *divisor_clk*

- *preset* genérico
- mudando o valor do *preset* muda-se a frequência da saída do clock dividido.

```
entity contador_8bits is
    Port ( reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           updown : in  STD_LOGIC;
           saida : out  STD_LOGIC_VECTOR (7 downto 0));
end contador_8bits;

architecture Behavioral of contador_8bits is
signal count : std_logic_vector(7 downto 0) := "00000000";
begin

      process(clk,reset)
      begin
            if rising_edge(clk) then
                  if reset='1' then
                        count<="00000000";
                  elsif updown='0' then
                        count<=count+'1';
                  elsif updown='1' then
                        count<=count-'1';
                  end if;
            end if;
      end process;
      saida <= count;

end Behavioral;
```

Exemplo 5 contador updown com deco7seg

Implementação do contador de 8 bits *contador_8bits*

- se *updown*='0' incrementa
- se *updown*='1' decrementa
- o sinal de clk irá receber o divisor de clock de 1 Hz.

```
entity mux4_1 is
    Port ( reset : in std_logic;
           clk  : in std_logic;
           dig1 : in  STD_LOGIC_VECTOR (3 downto 0);
           dig2 : in  STD_LOGIC_VECTOR (3 downto 0);
           dig3 : in  STD_LOGIC_VECTOR (3 downto 0);
           dig4 : in  STD_LOGIC_VECTOR (3 downto 0);
           saida : out  STD_LOGIC_VECTOR (3 downto 0));
end mux4_1;

architecture Behavioral of mux4_1 is
signal sel : STD_LOGIC_VECTOR (1 downto 0) := "00";
begin
        with sel select
                saida <= dig1 when "00",
                         dig2 when "01",
                         dig3 when "10",
                         dig4 when others;

        process(clk,reset)
        begin
                if rising_edge(clk) then
                        if reset='1' then
                                sel<="00";
                        else
                                sel <= sel+"01";
                        end if;
                end if;
        end process;
end Behavioral;
```

Exemplo 5 contador updown com deco7seg

Implementação do multiplexador *mux4_1*

- O multiplexador é implementado combinacionalmente com as diretivas ***with select.***
- O processo sequencial implementa um contador de 2 bits que controla o mux.
- E entrada de *clk* irá receber a saída do divisor de clock de 128 Hz que controla a multiplexação dos displays de 7 segmentos.

```
entity registrador_deslocamento is
    Port ( reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           anodos : out  STD_LOGIC_VECTOR (3 downto 0));
end registrador_deslocamento;

architecture Behavioral of registrador_deslocamento is

begin

        process(clk,reset)
        variable aux_anodo : std_logic_vector(4 downto 0) := "11110";
        begin
                if rising_edge(clk) then
                        if reset='1' then
                                aux_anodo := "11110";
                        else
                                aux_anodo := aux_anodo(3 downto 0)&'1';
                                if aux_anodo = "01111" then
                                        aux_anodo := "11110";
                                end if;
                        end if;
                        anodos <= aux_anodo(3 downto 0);
                end if;
        end process;

end Behavioral;
```

Exemplo 5 contador updown com deco7seg

Implementação do registrador de deslocamento

- Na condição de *reset* a saída dos anodos é “1110” ligando o primeiro display de 7 segmentos.
- O processo sequencial realiza um *shift left* a cada borda de subida do clock.
- E entrada de *clk* irá receber a saída do divisor de clock de 128 Hz que controla a multiplexação dos displays de 7 segmentos.

```
entity deco_7seg is
   Port ( entrada : in  STD_LOGIC_VECTOR (3 downto 0);
          segmentos : out  STD_LOGIC_VECTOR (7 downto 0));
end deco_7seg;

architecture Behavioral of deco_7seg is
begin
      process(entrada)
      begin
            case entrada is
                  when "0000" => segmentos <= "11000000"; --0
                  when "0001" => segmentos <= "11111001"; --1
                  when "0010" => segmentos <= "10100100"; --2
                  when "0011" => segmentos <= "10110000"; --3
                  when "0100" => segmentos <= "10011001"; --4
                  when "0101" => segmentos <= "10010010"; --5
                  when "0110" => segmentos <= "10000010"; --6
                  when "0111" => segmentos <= "11111000"; --7
                  when "1000" => segmentos <= "10000000"; --8
                  when "1001" => segmentos <= "10011000"; --9
                  when "1010" => segmentos <= "10001000"; --A
                  when "1011" => segmentos <= "10000011"; --b
                  when "1100" => segmentos <= "11000110"; --c
                  when "1101" => segmentos <= "10100001"; --d
                  when "1110" => segmentos <= "10000110"; --E
                  when others => segmentos <= "10001110"; --F
            end case;
      end process;
end Behavioral;
```

Exemplo 5 contador updown com deco7seg

Implementação do decodificador de 7 segmentos

- implementação combinacional (processo não depende do clock).
- Decodificação binário para 7 segmentos.

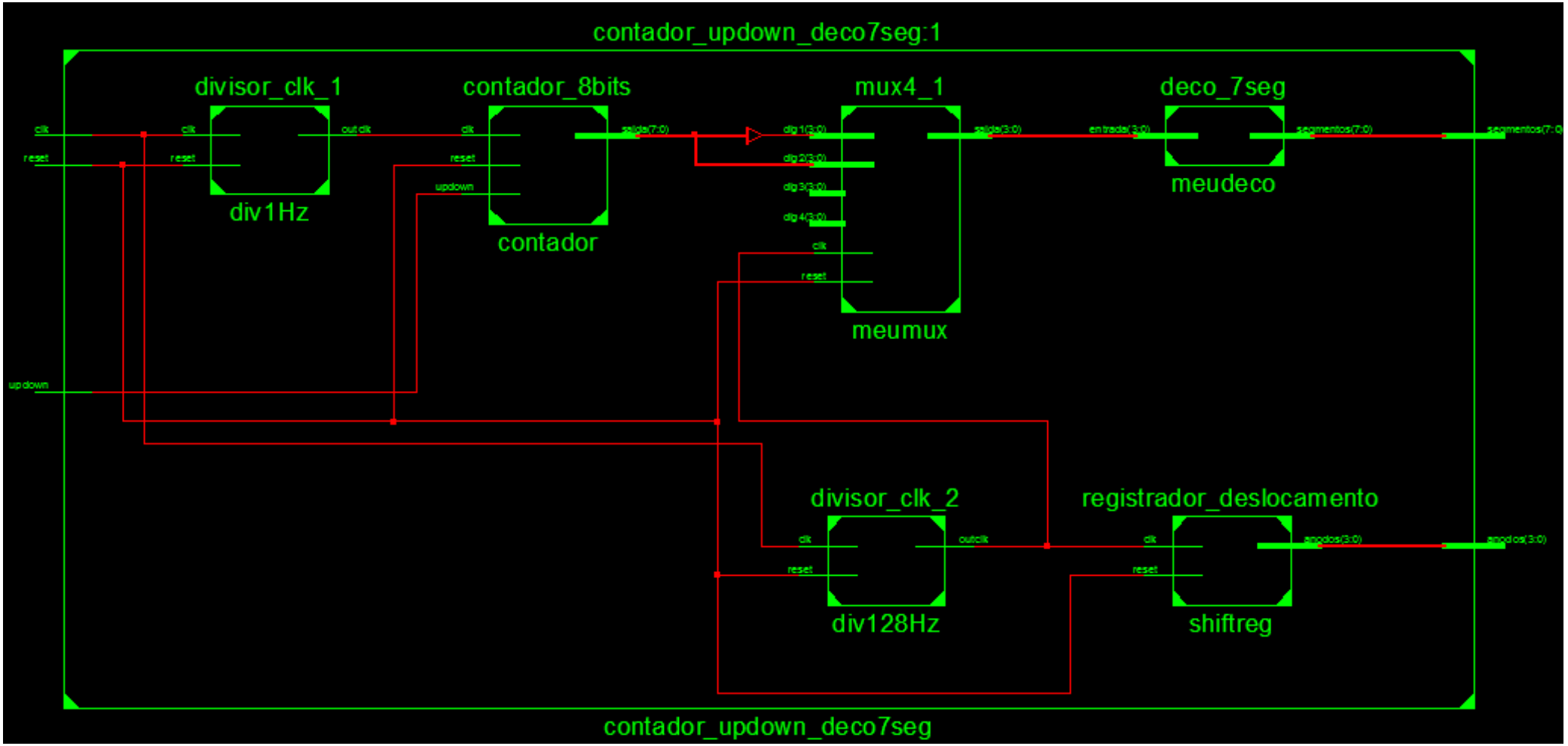
```
architecture Behavioral of contador_updown_deco7seg is
-- declaracao de componentes
-- declaracao de sinais
begin
div1Hz: divisor_clk generic map (preset => "10111110101111000001000000")
                    port map(reset  => reset,
                             clk    => clk, -- 50 MHz
                             outclk => clk1hz);
div128Hz: divisor_clk generic map (preset => "00000010111110101111100001")
                      port map(reset  => reset,
                               clk    => clk,
                               outclk => clk128hz);
contador: contador_8bits port map(reset  => reset,
                                  clk    => clk1hz,
                                  updown => updown,
                                  saida  => conta);
meumux: mux4_1 port map(reset => reset,
                        clk   => clk128hz,
                        dig1  => conta(3 downto 0),
                        dig2  => conta(7 downto 4),
                        dig3  => "0000",
                        dig4  => "0000",
                        saida => deco);
shiftreg: registrador_deslocamento port map(reset   => reset,
                                            clk     => clk128hz,
                                            anodos  => anodos);
meudeco: deco_7seg port map(entrada   => deco,
                            segmentos => segmentos);
end Behavioral;
```

Exemplo 5 contador updown com deco7seg

- Instanciação dos componentes
- o clock de 1 Hz é a entrada de clock do contador de 8 bits
- o clock de 128 Hz é a entrada de clock do multiplexador e do registrador de 7 segmentos.
- a saída *conta* do contador é segmentada nos dígitos do multiplexador.
- a saída *deco* de 4 bits do multiplexador é a entrada do decodificador de 7 segmentos.

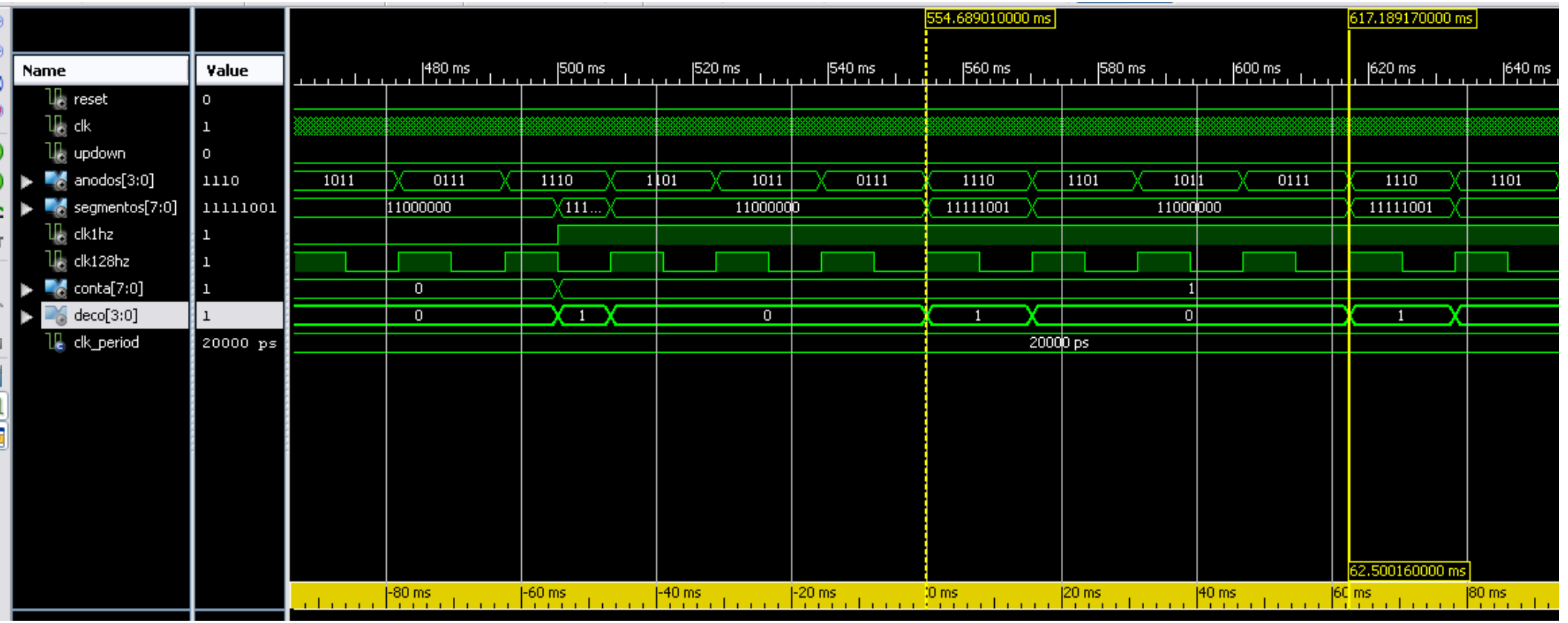
Exemplo 5 contador updown com deco7seg

Arquitetura RTL após a síntese lógica.



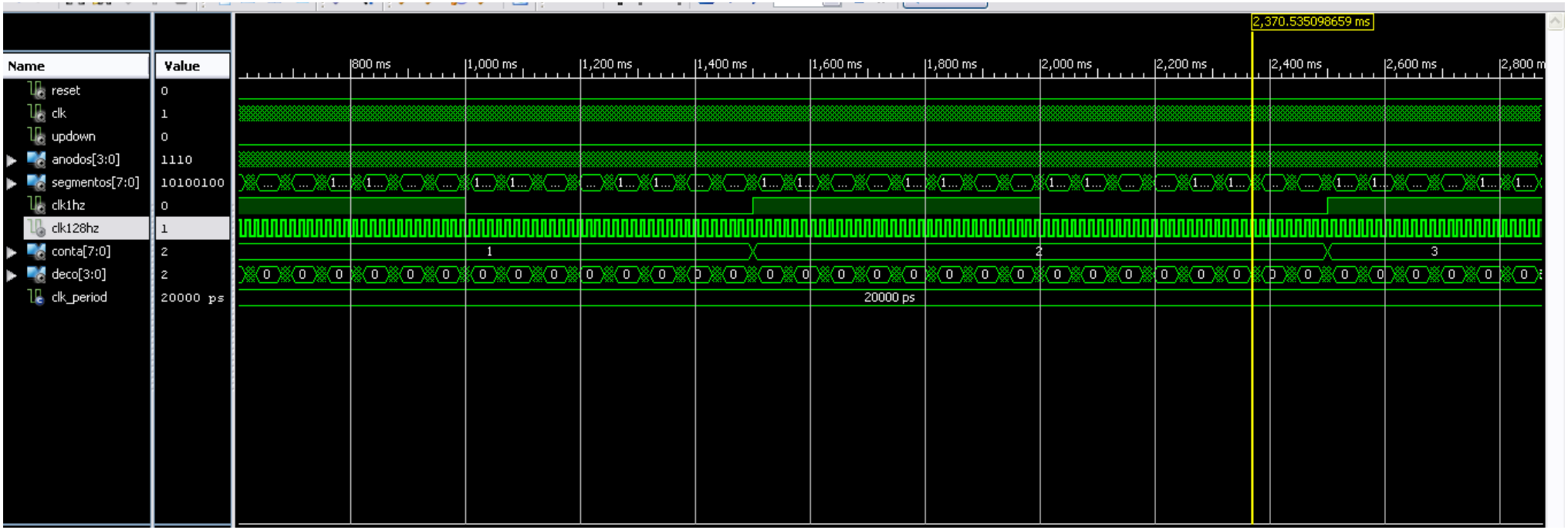
Exemplo 5 contador updown com deco7seg

Testbench e simulação



Exemplo 5 contador updown com deco7seg

Testbench e simulação



Sugestão de leitura:
Capítulo 10, Livro: Pedroni, V.A., “Circuit Design with VHDL”, MIT Press, USA, 2004

Sugestão de exercícios:
1. Usando a metodologia de instanciação por componentes implemente 3 divisores de clock a 2Hz, 1Hz e 0.5Hz
2. Modifique a ULA do exemplo 3 desta aula para trabalhar com 4 bits. Implemente no FPGA usando 4 switches para a entrada a e 4 switches para a entrada b. A saída deve ser mapeada nos leds da placa de desenvolvimento.
3. Acrescente um componente decodificador binário para 7 segmentos na implementação da ULA do exercício anterior.

Sugestão vídeo aula:
Implementação por componentes de um contador updown de 8 bits com divisor de clock e decodificador de 7 segmentos (3 vídeos, aprox. 47 minutos)

https://www.youtube.com/watch?v=rwyBVK0vnDU&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=16
https://www.youtube.com/watch?v=3drbBdLOI5Y&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=17
https://www.youtube.com/watch?v=_BHfpzNOpLc&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=18