Relatório

Tópico: Aula 4 & 5 - Contadores
Grupo: Maicon Chaves Marques 14593530
Pedro Calciolari Jardim 11233668
João Vitor Pereira Candido 13751131

Parte 1: 8-bits counter

O contador de 8 bits foi implementado utilizando Flip-Flops do tipo T, organizados para alternar seus estados de forma sequencial com base no sinal de *clock*. O funcionamento do contador é estruturado conforme descrito abaixo:

- O sinal **`enable1`** habilita o funcionamento do contador. Ele permite que os Flip-Flops alternem seus estados conforme o *clock*.
- O sinal **`clock`** é responsável por coordenar a alternância dos Flip-Flops, sendo aplicado simultaneamente a todos.
- O sinal **`clear`** é assíncrono e força todos os Flip-Flops a assumirem o estado lógico 0, independente de seu estado atual.
- A contagem é realizada com base em uma lógica de *toggle*: cada Flip-Flop T alterna seu estado dependendo do valor do Flip-Flop imediatamente anterior e do sinal de habilitação (`enable`).

Código VHDL:

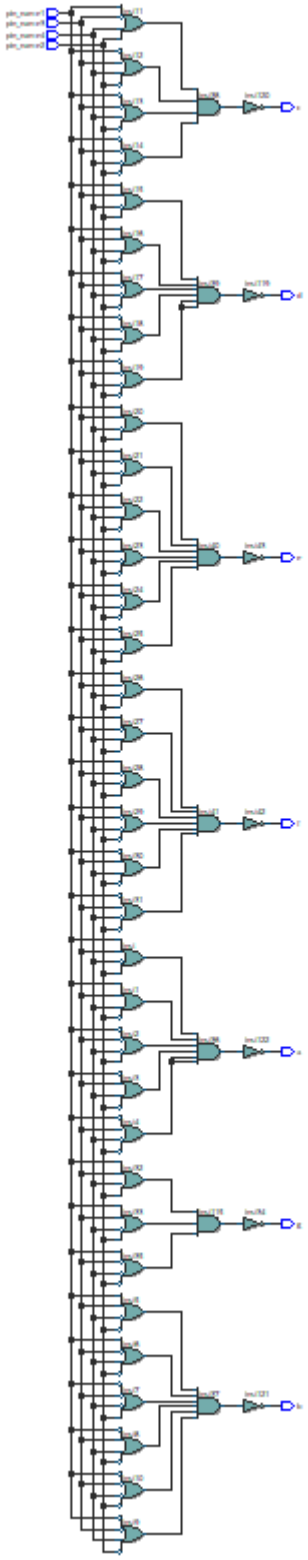
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity contador_8bits is
    Port ( enable1 : in STD_LOGIC;       -- Sinal de clock
                              clock : in STD_LOGIC;       -- Sinal de clock
           clear : in STD_LOGIC;     -- Sinal de clear assíncrono
                              Q0 : out STD_LOGIC;
                              Q1 : out STD_LOGIC;
                              Q2 : out STD_LOGIC;
                              Q3 : out STD_LOGIC;
                              Q4 : out STD_LOGIC;
                              Q5 : out STD_LOGIC;
                              Q6 : out STD_LOGIC;
                              Q7 : out STD_LOGIC
         );
```

```
end contador_8bits;

architecture Behavioral of contador_8bits is
   -- Sinais intermediários para conectar os flip-flops
          signal enable_signals : STD_LOGIC_VECTOR (7 downto 0); -- Estados internos dos
flip-flops
   signal Q_internal : STD_LOGIC_VECTOR (7 downto 0); -- Estados internos dos flip-flops
          signal and_signals : STD_LOGIC_VECTOR (7 downto 0); -- Estados internos dos
flip-flops

                  component t_flip_flop is
                  port (
                           clock : in STD_LOGIC;
         clear : in STD_LOGIC;
         enable : in STD_LOGIC;
         Q : out STD_LOGIC;
                           nQ : out STD_LOGIC
       );
         end component;

begin
   -- Instanciando o primeiro flip-flop T (bit menos significativo)
   tff0: t_flip_flop
      port map (clock => clock,
               clear => clear,
               enable => enable1,          -- Flip-flop T sempre habilitado
               Q => Q_internal(0)  -- Saída do primeiro flip-flop
               );

   -- Geração dos sinais AND entre os flip-flops
   and_signals(1) <= Q_internal(0) AND enable1;  -- O primeiro flip-flop não precisa de AND

   tff1: t_flip_flop
      port map (clock => clock,
               clear => clear,
               enable => and_signals(1), -- O segundo flip-flop alterna quando o primeiro
estiver em '1'
               Q => Q_internal(1)
               );

   -- AND para o próximo flip-flop
   and_signals(2) <= and_signals(1) AND Q_internal(1);

   tff2: t_flip_flop
      port map (clock => clock,
               clear => clear,
               enable => and_signals(2), -- O terceiro flip-flop alterna quando os dois anteriores
estiverem em '1'
```

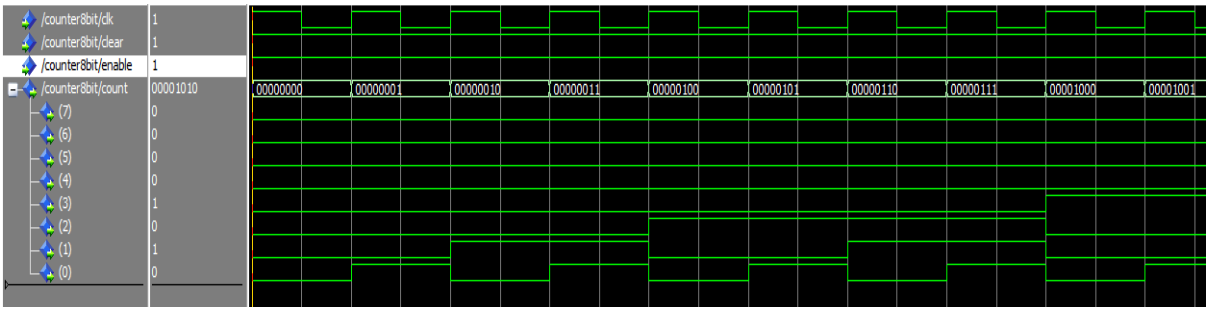
```
            Q => Q_internal(2)
            );

-- AND para o próximo flip-flop
and_signals(3) <= and_signals(2) AND Q_internal(2);

tff3: t_flip_flop
   port map (clock => clock,
            clear => clear,
            enable => and_signals(3),
            Q => Q_internal(3)
            );

and_signals(4) <= and_signals(3) AND Q_internal(3);

tff4: t_flip_flop
   port map (clock => clock,
            clear => clear,
            enable => and_signals(4),
            Q => Q_internal(4)
            );

and_signals(5) <= and_signals(4) AND Q_internal(4);

tff5: t_flip_flop
   port map (clock => clock,
            clear => clear,
            enable => and_signals(5),
            Q => Q_internal(5)
            );

and_signals(6) <= and_signals(5) AND Q_internal(5);

tff6: t_flip_flop
   port map (clock => clock,
            clear => clear,
            enable => and_signals(6),
            Q => Q_internal(6)
            );

and_signals(7) <= and_signals(6) AND Q_internal(6);

tff7: t_flip_flop
   port map (clock => clock,
            clear => clear,
            enable => and_signals(7),
            Q => Q_internal(7)
            );
```

```
    -- Atribuindo a saída dos flip-flops à saída do contador
    Q0 <= Q_internal(0);
            Q1 <= Q_internal(1);
            Q2 <= Q_internal(2);
            Q3 <= Q_internal(3);
            Q4 <= Q_internal(4);
            Q5 <= Q_internal(5);
            Q6 <= Q_internal(6);
            Q7 <= Q_internal(7);

end Behavioral;
```

2 - Representação das portas lógicas geradas pelo Quartus RTL Viewer a partir do código VHDL - Parte 1



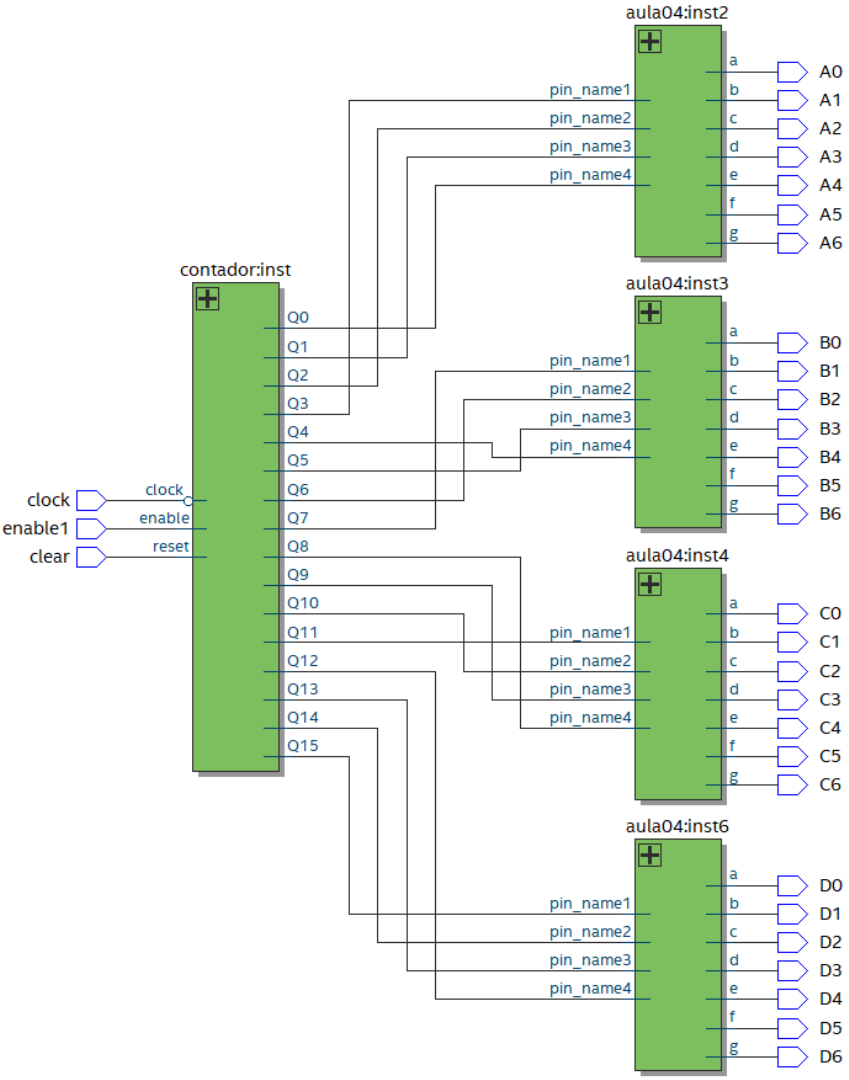
4- Simulação ModelSim - Parte 1



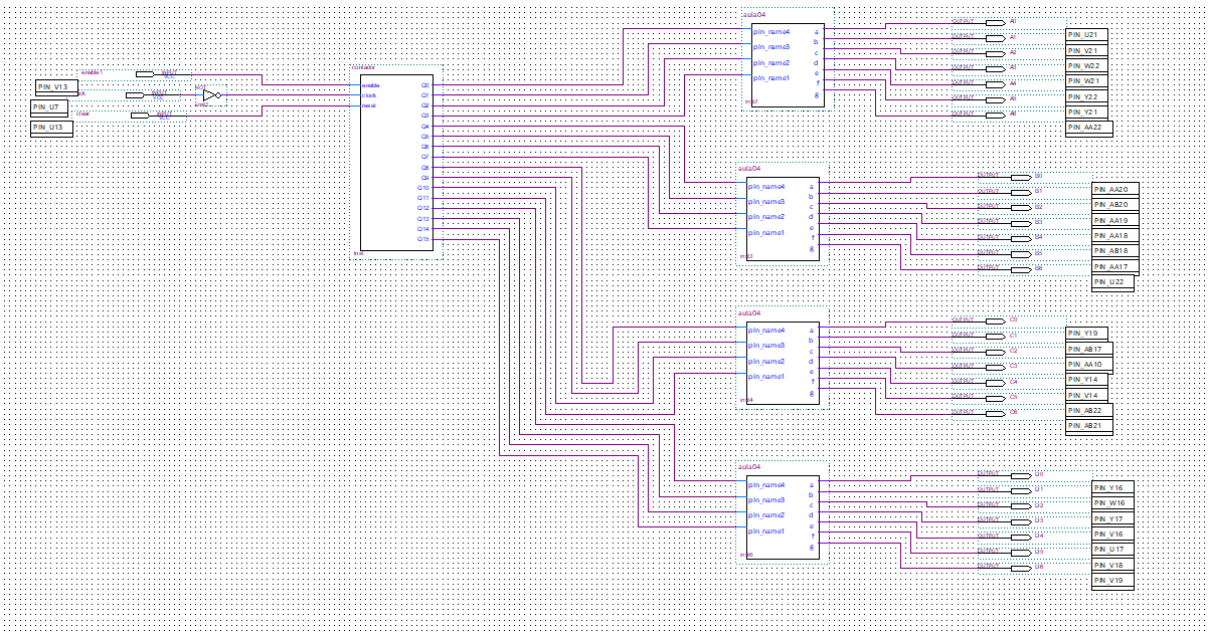
Parte 2: 16-bits counter

O contador de 16 bits descrito no código VHDL é uma unidade lógica sequencial que incrementa seu estado a cada ciclo de clock, desde que habilitado. Ele utiliza entradas como `clock`, `clear` e `enable`1 para controlar suas operações básicas, como contagem, reset e habilitação.

5 - Representação das portas lógicas geradas pelo Quartus RTL Viewer a partir do código VHDL - Parte 2



BDF da parte 2 -



7- Simulação ModelSim - Parte 2

Parte 3:

O circuito funciona como um divisor de frequência que conta até 50 milhões (50 MHz) para gerar um pulso de 1 segundo. Ao atingir 50 milhões, ele reseta e emite o sinal, que habilita um contador secundário de dígitos. Esse contador incrementa de 0 a 9, avançando a cada pulso de 1 segundo recebido do contador principal, e reseta para 0 ao atingir 9.

Um decodificador de 7 segmentos é usado para converter o valor do contador de dígitos no formato adequado para acionar um display de 7 segmentos, exibindo os números de 0 a 9.

Segue o código VHDL da parte 3:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY parte3 IS
	PORT
	(
		clock :  IN  STD_LOGIC;
		clear :  IN  STD_LOGIC;
		enable1 :  IN  STD_LOGIC;
		A0 :  OUT  STD_LOGIC;
		A1 :  OUT  STD_LOGIC;
		A2 :  OUT  STD_LOGIC;
		A3 :  OUT  STD_LOGIC;
		A4 :  OUT  STD_LOGIC;
		A5 :  OUT  STD_LOGIC;
		A6 :  OUT  STD_LOGIC
	);
END parte3;

ARCHITECTURE bdf_type OF parte3 IS

COMPONENT contador
	PORT(enable : IN STD_LOGIC;
		clock : IN STD_LOGIC;
		reset : IN STD_LOGIC;
		Q0 : OUT STD_LOGIC;
		Q1 : OUT STD_LOGIC;
		Q2 : OUT STD_LOGIC;
		Q3 : OUT STD_LOGIC
	);
```

```
END COMPONENT;

COMPONENT aula04
	PORT(pin_name1 : IN STD_LOGIC;
		pin_name4 : IN STD_LOGIC;
		pin_name2 : IN STD_LOGIC;
		pin_name3 : IN STD_LOGIC;
		a : OUT STD_LOGIC;
		b : OUT STD_LOGIC;
		c : OUT STD_LOGIC;
		d : OUT STD_LOGIC;
		e : OUT STD_LOGIC;
		f : OUT STD_LOGIC;
		g : OUT STD_LOGIC
	);
END COMPONENT;

SIGNAL	SYNTHESIZED_WIRE_0 :  STD_LOGIC;
SIGNAL	SYNTHESIZED_WIRE_1 :  STD_LOGIC;
SIGNAL	SYNTHESIZED_WIRE_2 :  STD_LOGIC;
SIGNAL	SYNTHESIZED_WIRE_3 :  STD_LOGIC;


BEGIN



b2v_inst : contador
PORT MAP(enable => enable1,
		clock => clock,
		reset => clear,
		Q0 => SYNTHESIZED_WIRE_1,
		Q1 => SYNTHESIZED_WIRE_3,
		Q2 => SYNTHESIZED_WIRE_2,
		Q3 => SYNTHESIZED_WIRE_0);


b2v_inst2 : aula04
PORT MAP(pin_name1 => SYNTHESIZED_WIRE_0,
		pin_name4 => SYNTHESIZED_WIRE_1,
		pin_name2 => SYNTHESIZED_WIRE_2,
		pin_name3 => SYNTHESIZED_WIRE_3,
		a => A0,
		b => A1,
		c => A2,
		d => A3,
		e => A4,
		f => A5,
```

```
                g => A6);


END bdf_type;
```

Parte 4:

O circuito implementado utiliza um divisor de clock para criar um atraso aproximado de 1 segundo. Isso é alcançado com a contagem de ciclos de clock até atingir um valor limite (50 milhões, considerando um clock de 50 MHz), momento em que o contador é resetado.

Um contador adicional é utilizado para rastrear a posição de exibição no display, alternando entre diferentes estados. Este contador incrementa seu valor a cada segundo e reseta após atingir 3, resultando em um ciclo de 4 estados.

A palavra "dE10" é representada em um vetor de 28 bits, onde cada caractere é codificado em 7 bits, correspondendo aos segmentos de um display de 7 segmentos. De acordo com o valor do contador, os segmentos apropriados são atribuídos aos displays HEX0 a HEX3. Assim, é criado um efeito de rotação da palavra de "direita para esquerda", exibindo os caracteres em sequência nos displays conectados.

Segue o código VHDL da parte 4:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY parte4 IS
        PORT
        (
                clock :  IN  STD_LOGIC;
                clear :  IN  STD_LOGIC;
                enable1 :  IN  STD_LOGIC;
                A0 :  OUT  STD_LOGIC;
                A1 :  OUT  STD_LOGIC;
                A2 :  OUT  STD_LOGIC;
                A3 :  OUT  STD_LOGIC;
                A4 :  OUT  STD_LOGIC;
                A5 :  OUT  STD_LOGIC;
                A6 :  OUT  STD_LOGIC;
                B0 :  OUT  STD_LOGIC;
                B1 :  OUT  STD_LOGIC;
                B2 :  OUT  STD_LOGIC;
                B3 :  OUT  STD_LOGIC;
```

```
		B4 :  OUT  STD_LOGIC;
		B5 :  OUT  STD_LOGIC;
		B6 :  OUT  STD_LOGIC;
		C0 :  OUT  STD_LOGIC;
		C1 :  OUT  STD_LOGIC;
		C2 :  OUT  STD_LOGIC;
		C3 :  OUT  STD_LOGIC;
		C4 :  OUT  STD_LOGIC;
		C5 :  OUT  STD_LOGIC;
		C6 :  OUT  STD_LOGIC;
		D0 :  OUT  STD_LOGIC;
		D1 :  OUT  STD_LOGIC;
		D2 :  OUT  STD_LOGIC;
		D3 :  OUT  STD_LOGIC;
		D4 :  OUT  STD_LOGIC;
		D5 :  OUT  STD_LOGIC;
		D6 :  OUT  STD_LOGIC
	);
END parte4;

ARCHITECTURE bdf_type OF parte4 IS

COMPONENT palavra04
	PORT(enable : IN STD_LOGIC;
		clock : IN STD_LOGIC;
		reset : IN STD_LOGIC;
		Q0 : OUT STD_LOGIC;
		Q1 : OUT STD_LOGIC;
		Q2 : OUT STD_LOGIC;
		Q3 : OUT STD_LOGIC;
		Q4 : OUT STD_LOGIC;
		Q5 : OUT STD_LOGIC;
		Q6 : OUT STD_LOGIC;
		Q7 : OUT STD_LOGIC;
		Q8 : OUT STD_LOGIC;
		Q9 : OUT STD_LOGIC;
		Q10 : OUT STD_LOGIC;
		Q11 : OUT STD_LOGIC;
		Q12 : OUT STD_LOGIC;
		Q13 : OUT STD_LOGIC;
		Q14 : OUT STD_LOGIC;
		Q15 : OUT STD_LOGIC
	);
END COMPONENT;

COMPONENT aula04
	PORT(pin_name1 : IN STD_LOGIC;
		pin_name4 : IN STD_LOGIC;
```

```
                pin_name2 : IN STD_LOGIC;
                pin_name3 : IN STD_LOGIC;
                a : OUT STD_LOGIC;
                b : OUT STD_LOGIC;
                c : OUT STD_LOGIC;
                d : OUT STD_LOGIC;
                e : OUT STD_LOGIC;
                f : OUT STD_LOGIC;
                g : OUT STD_LOGIC
        );
END COMPONENT;

SIGNAL      SYNTHESIZED_WIRE_0 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_1 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_2 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_3 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_4 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_5 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_6 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_7 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_8 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_9 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_10 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_11 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_12 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_13 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_14 :  STD_LOGIC;
SIGNAL      SYNTHESIZED_WIRE_15 :  STD_LOGIC;


BEGIN



b2v_inst : palavra04
PORT MAP(enable => enable1,
                clock => clock,
                reset => clear,
                Q0 => SYNTHESIZED_WIRE_1,
                Q1 => SYNTHESIZED_WIRE_3,
                Q2 => SYNTHESIZED_WIRE_2,
                Q3 => SYNTHESIZED_WIRE_0,
                Q4 => SYNTHESIZED_WIRE_5,
                Q5 => SYNTHESIZED_WIRE_7,
                Q6 => SYNTHESIZED_WIRE_6,
                Q7 => SYNTHESIZED_WIRE_4,
                Q8 => SYNTHESIZED_WIRE_9,
                Q9 => SYNTHESIZED_WIRE_11,
```

```
                Q10 => SYNTHESIZED_WIRE_10,
                Q11 => SYNTHESIZED_WIRE_8,
                Q12 => SYNTHESIZED_WIRE_13,
                Q13 => SYNTHESIZED_WIRE_15,
                Q14 => SYNTHESIZED_WIRE_14,
                Q15 => SYNTHESIZED_WIRE_12);


b2v_inst2 : aula04
PORT MAP(pin_name1 => SYNTHESIZED_WIRE_0,
                pin_name4 => SYNTHESIZED_WIRE_1,
                pin_name2 => SYNTHESIZED_WIRE_2,
                pin_name3 => SYNTHESIZED_WIRE_3,
                a => A0,
                b => A1,
                c => A2,
                d => A3,
                e => A4,
                f => A5,
                g => A6);


b2v_inst3 : aula04
PORT MAP(pin_name1 => SYNTHESIZED_WIRE_4,
                pin_name4 => SYNTHESIZED_WIRE_5,
                pin_name2 => SYNTHESIZED_WIRE_6,
                pin_name3 => SYNTHESIZED_WIRE_7,
                a => B0,
                b => B1,
                c => B2,
                d => B3,
                e => B4,
                f => B5,
                g => B6);


b2v_inst4 : aula04
PORT MAP(pin_name1 => SYNTHESIZED_WIRE_8,
                pin_name4 => SYNTHESIZED_WIRE_9,
                pin_name2 => SYNTHESIZED_WIRE_10,
                pin_name3 => SYNTHESIZED_WIRE_11,
                a => C0,
                b => C1,
                c => C2,
                d => C3,
                e => C4,
                f => C5,
                g => C6);
```

```
b2v_inst6 : aula04
PORT MAP(pin_name1 => SYNTHESIZED_WIRE_12,
                pin_name4 => SYNTHESIZED_WIRE_13,
                pin_name2 => SYNTHESIZED_WIRE_14,
                pin_name3 => SYNTHESIZED_WIRE_15,
                a => D0,
                b => D1,
                c => D2,
                d => D3,
                e => D4,
                f => D5,
                g => D6);


END bdf_type;
```