Universidade Tecnológica Federal do Paraná – Toledo
Engenharia da Computação – COENC

# Lógica Reconfigurável

# GENERATE

Tiago Piovesan Vendruscolo




UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

# GENERATE

- Permite repetições em códigos concorrentes;

- A forma regular é FOR/GENERATE, sempre usando label e seguindo a sintaxe:

```
Label: FOR identificador IN range GENERATE
        (atribuições concorrentes)
END GENERATE label;
```

- Gerador de paridade PAR

Fonte: (TOCCI; WIDMER; MOSS, 2011)

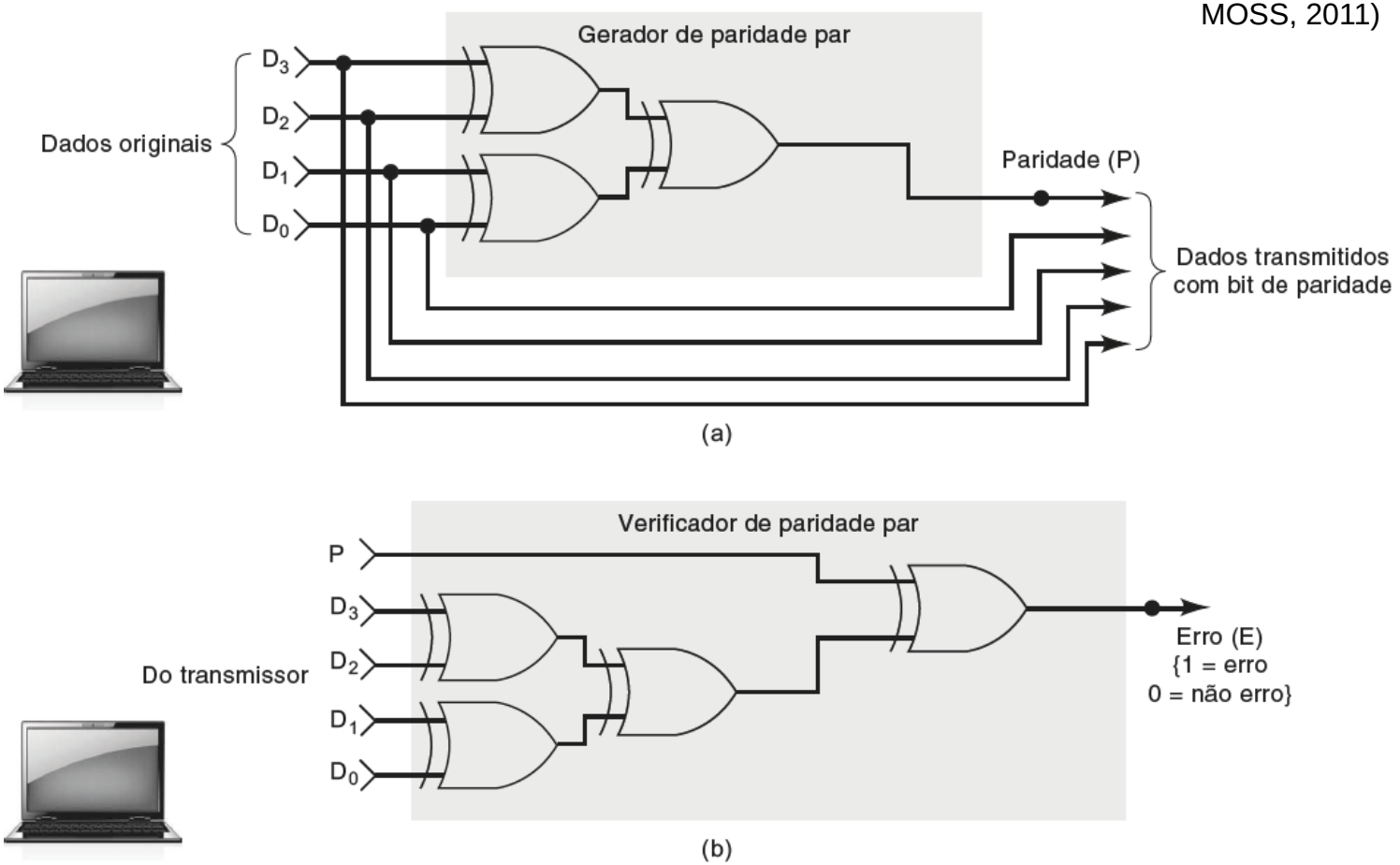
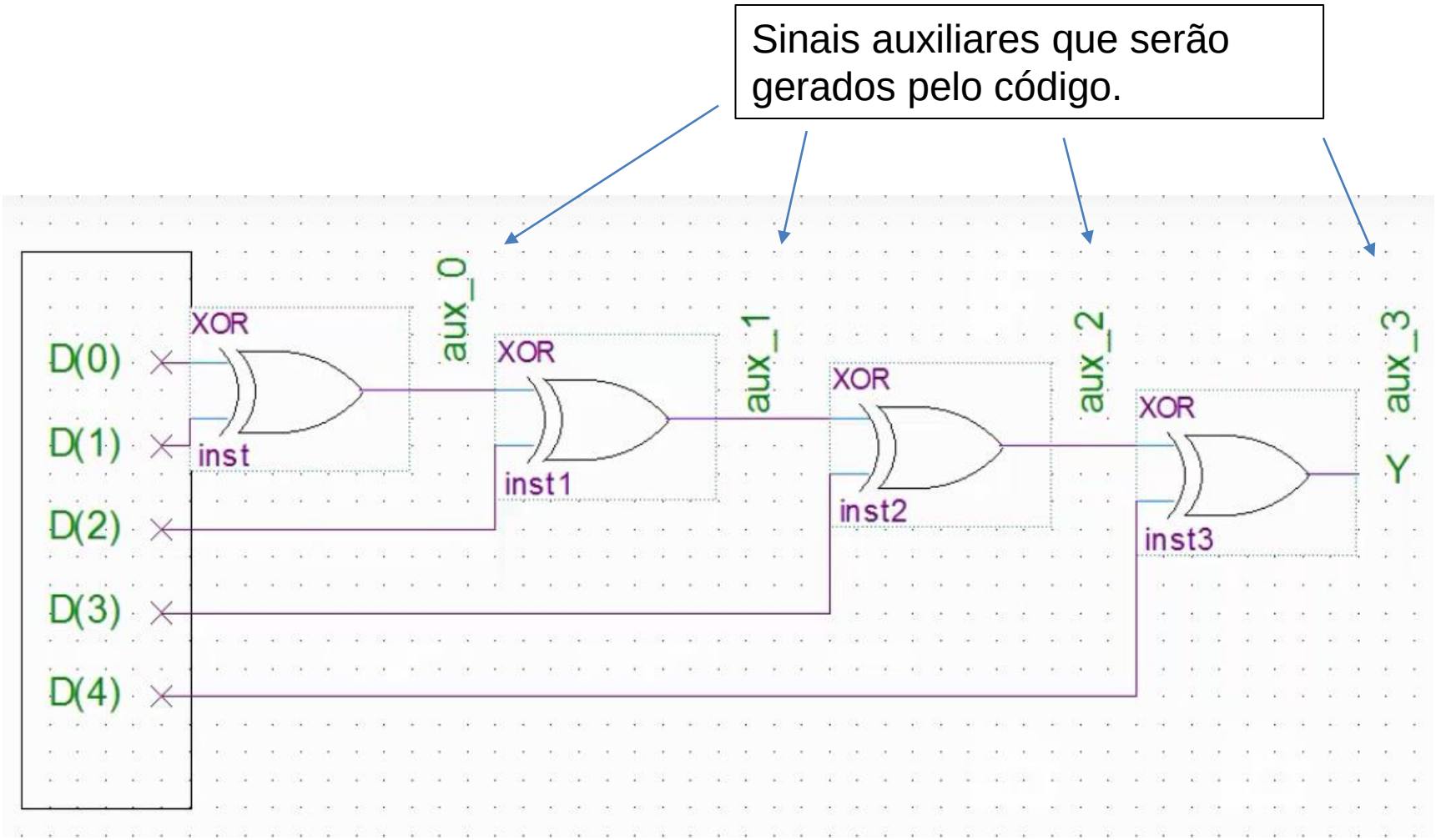
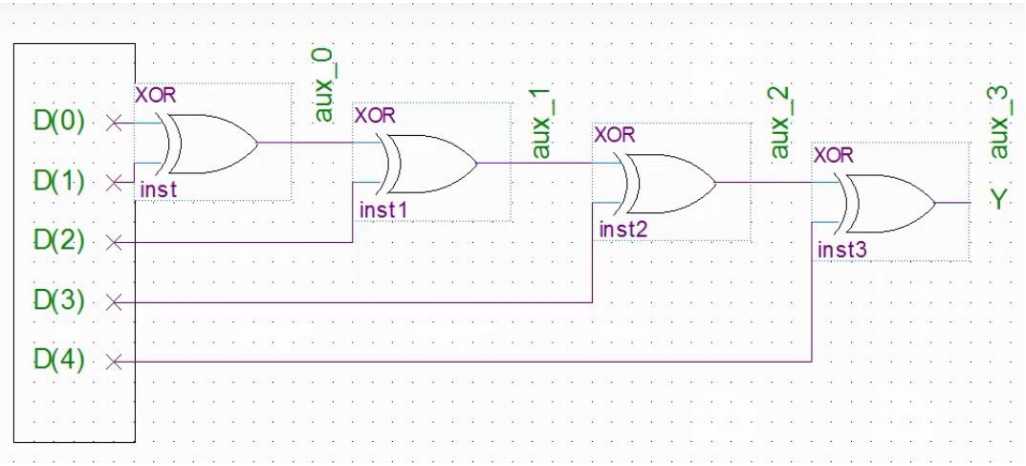



**FIGURA 4.25** Portas XOR utilizadas para implementar (a) um gerador de paridade e (b) um verificador de paridade para um sistema que usa paridade par.

- Gerador de paridade PAR

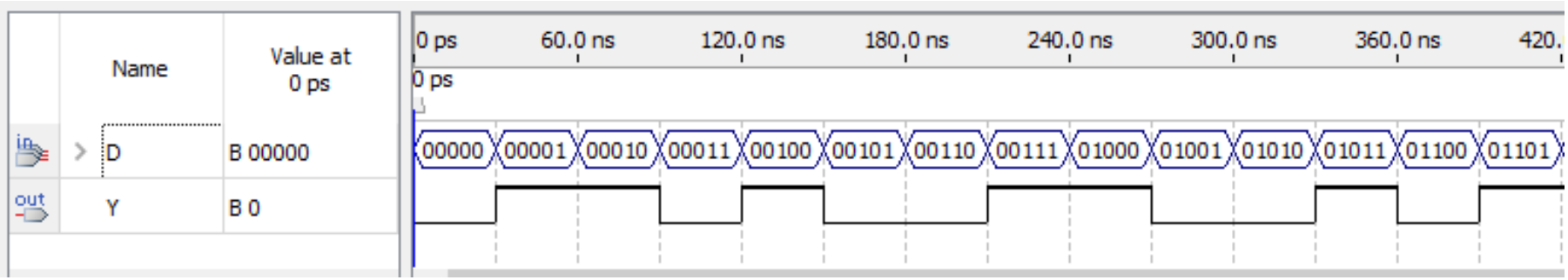


Façam o exemplo e analisem o RTL VIEWER

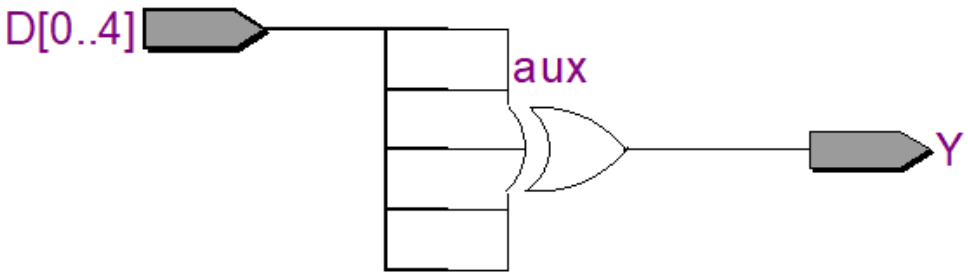



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY paridade IS
PORT (D: IN BIT_VECTOR (0 TO 4);
      Y: OUT BIT);
END ENTITY;

ARCHITECTURE funcao OF paridade IS
   SIGNAL aux: BIT_VECTOR (0 TO 3);
BEGIN
   aux(0)<=D(0) XOR  D(1);
   GEN: FOR i IN 1 TO 3 GENERATE
      aux(i)<=D(i+1) XOR aux(i-1);
   END GENERATE GEN;
   y<= aux(3);
END ARCHITECTURE;
```

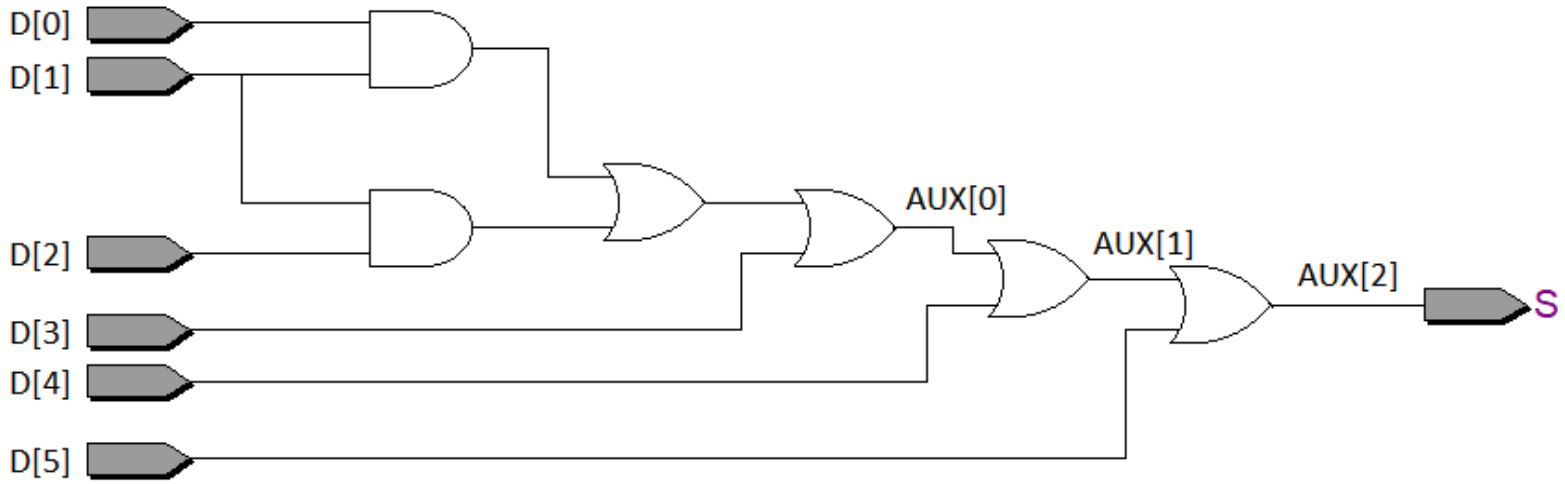
Simulação:




RTL viewer:



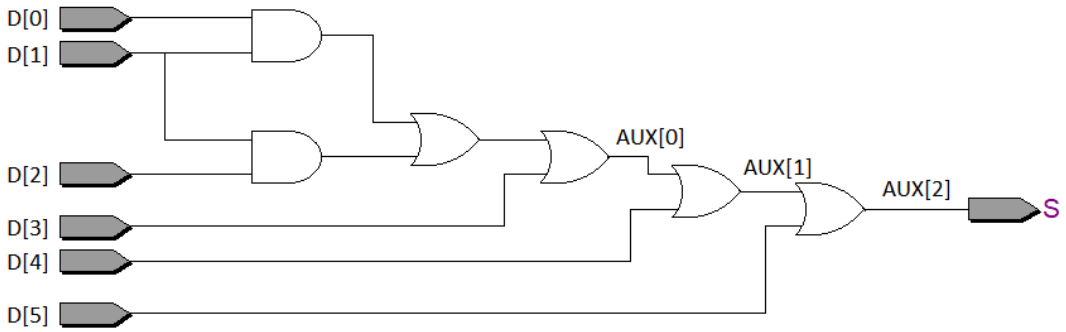
Exercício 1: Faça o circuito abaixo:




Exemplo Estrutura:

```
ENTITY paridade IS
PORT (D: IN BIT_VECTOR (0 TO 4);
      Y: OUT BIT);
END ENTITY;

ARCHITECTURE funcao OF paridade IS
   SIGNAL aux: BIT_VECTOR (0 TO 3);
BEGIN
   aux(0)<=D(0) XOR  D(1);
   GEN: FOR i IN 1 TO 3 GENERATE
      aux(i)<=D(i+1) XOR aux(i-1);
   END GENERATE GEN;
   y<= aux(3);
END ARCHITECTURE;
```

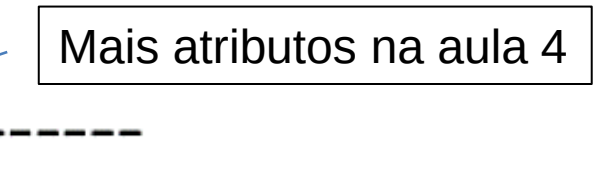
# GENERATE




```
ENTITY ex2 IS
PORT (D: IN STD_LOGIC_VECTOR(0 TO 5);
      S: OUT STD_LOGIC);
END ex2;
---------------------------
ARCHITECTURE funcao OF ex2 IS
Signal aux: std_logic_vector (0 to 2);
BEGIN
   aux(0) <= ((D(0) AND D(1)) OR (D(1) AND D(2))) OR D(3);

   GEN1: FOR i IN 1 TO 2 GENERATE

         aux(i) <= aux(i-1) OR D(i+3);

   END   GENERATE GEN1;
   S <= aux(2);
END funcao;
```

# GENERATE INCONDICIONAL

```
-------------------------------------
SIGNAL a, b, x: BIT_VECTOR(7 DOWNTO 0);
-------------------------------------
gen: FOR i IN 0 TO 7 GENERATE
   x(i) <= a(i) XOR b(7-i);
END GENERATE;
-------------------------------------
gen: FOR i IN a'RANGE GENERATE
   x(i) <= a(i) XOR b(7-i);
END GENERATE;
-------------------------------------
gen: FOR i IN a'REVERSE_RANGE GENERATE
   x(i) <= a(i) XOR b(7-i);
END GENERATE;
-------------------------------------
```



# GENERATE

```
-------------------------------------------
SIGNAL a, b, x, y: BIT_VECTOR(3 DOWNTO 0);
SIGNAL z: INTEGER RANGE 0 TO 7;
-------------------------------------------
OK: FOR i IN x'RANGE GENERATE
   x(i)<='1' WHEN (a(i) AND b(i))='1' ELSE '0';
END GENERATE;
-------------------------------------------
NotOK: FOR i IN y'LOW TO y'HIGH GENERATE
   y <="1111" WHEN (a(i) AND b(i))='1' ELSE
        "0000";
END GENERATE;
-------------------------------------------
```

ERRO! Falta o ínidice

# GENERATE CONDICIONAL

Deve obedecer uma condição para ser executado:

```
Label: IF condition GENERATE
       [Atribuições concorrentes]
END GENERATE label;
```

Exemplo:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY exemplo IS
   GENERIC (TESTE : INTEGER := 0);
   -- 0 = Não executa teste do codigo
   -- 1 = Executa teste do codigo
   PORT(D: IN INTEGER RANGE 0 to 7;
        Y: OUT STD_LOGIC);
END ENTITY;

ARCHITECTURE funcao OF exemplo IS

   BEGIN
      label_if : IF TESTE = 1 GENERATE

      -- Código concorrente

      END GENERATE label_if;
END;
```

O objeto “teste” deve ser estático

# GENERATE CONDICIONAL

- É possível usar IF/GENERATE dentro de FOR/GENERATE e vice-versa:

```
Label1: FOR identificador IN range GENERATE
...
      Label2: IF condicao GENERATE
            [atribuições concorrentes]
      END GENERATE label2;
END GENERATE label1;
```

- Na forma IF/GENERATE, não é permitido usar ELSE.

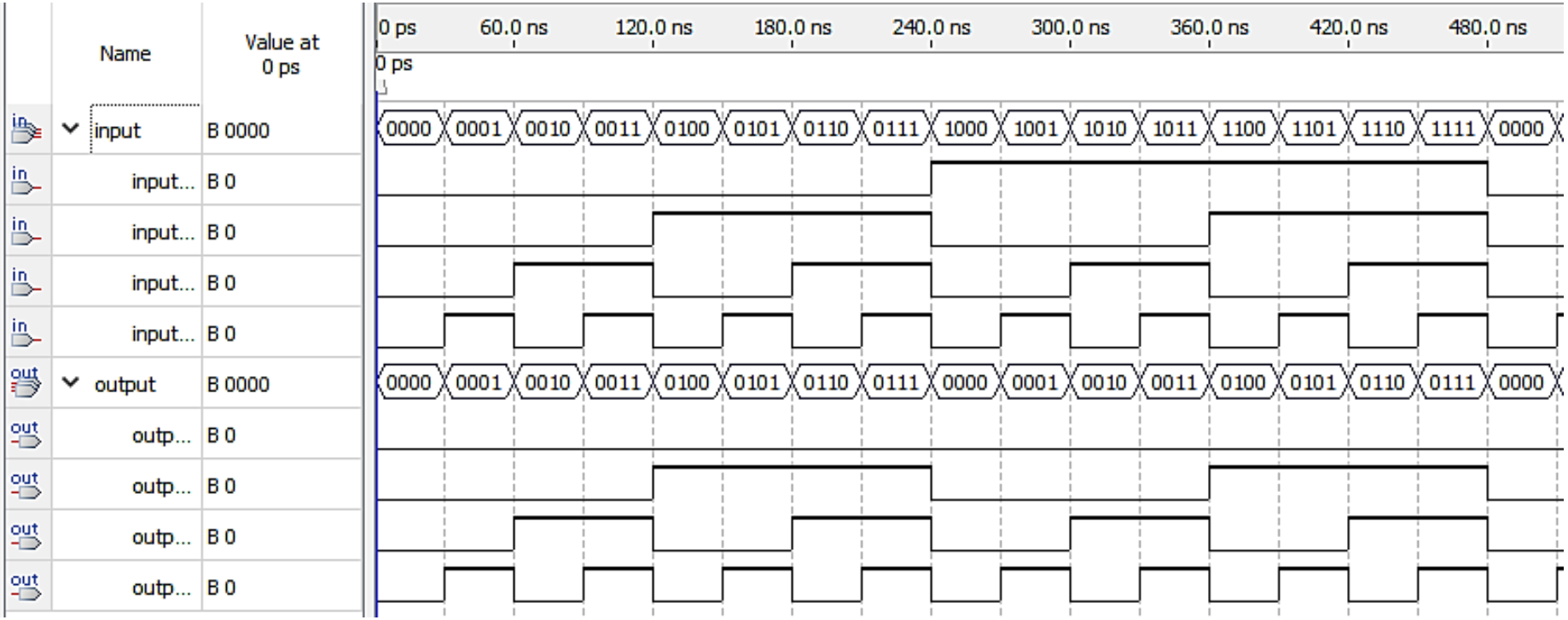
- Nota: GENERATE é muito utilizado com COMPONENT.
  - *Registradores.*

# GENERATE CONDICIONAL

Exercício: Faça um código que tenha uma entrada e uma saída de 4 bits. A saída deve ser igual à entrada, com exceção do bit mais significativo que deve ser sempre ‘0’.

```
ENTITY condicional IS
PORT (input: IN BIT_VECTOR (3 downto 0);
      output: OUT BIT_VECTOR (3 downto 0));
END ENTITY;

ARCHITECTURE funcao OF condicional IS
   BEGIN
   GEN1: FOR i IN 3 DOWNTO 0 GENERATE
      GEN2: IF i = 3 GENERATE
      output(i)<='0';
      END GENERATE GEN2;

      GEN3: IF i /= 3 GENERATE
      output(i)<=input(i);
      END GENERATE GEN3;
   END GENERATE GEN1;
END ARCHITECTURE;
```

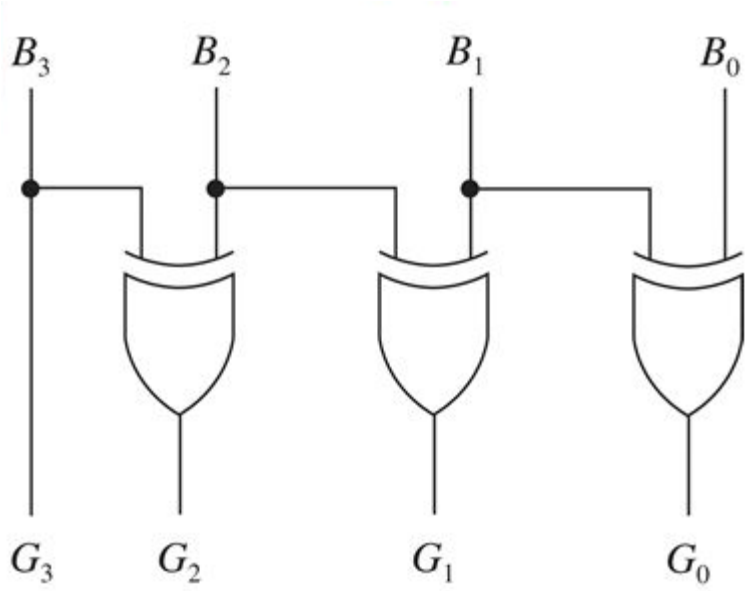
# GENERATE CONDICIONAL



# Exercício

- Escreva dois códigos em VHDL para um conversor de código binário para código Gray de 4 bits:
  - Usando GENERATE condicional;
  - Usando GENERATE incondicional.

Entrada binária




Saída em código Gray

| Binary | | | | Gray Code | | | |
|---|---|---|---|---|---|---|---|
| $b_3$ | $b_2$ | $b_1$ | $b_0$ | $g_3$ | $g_2$ | $g_1$ | $g_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

- Grave na FPGA usando SW[0-3] como entrada e LED[0-3] como saída.

# Exercício

- GENERATE Condicional

```
ENTITY codigo_gray IS
PORT (input: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      output: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END codigo_gray;
---------------------------
ARCHITECTURE funcao OF codigo_gray IS
BEGIN
-- GENERATE CONDICIONAL --

   GEN1: FOR i IN 3 DOWNTO 0 GENERATE

         GEN2: IF i = 3 GENERATE
            output(i) <= input(i);
         END GENERATE GEN2;

         GEN3: IF i /=3 GENERATE
            output(i) <= input(i) XOR input(i+1);
         END GENERATE GEN3;
   END GENERATE GEN1;

END funcao;
```

# Exercício

- GENERATE Incondicional

```
ENTITY codigo_gray IS
PORT (input: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      output: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END codigo_gray;
--------------------------
ARCHITECTURE funcao OF codigo_gray IS
BEGIN
-- GENERATE INCONDICIONAL --
   output(3) <= input(3);

   GEN1: FOR i IN 2 DOWNTO 0 GENERATE

         output(i) <= input(i) XOR input(i+1);

   END   GENERATE GEN1;

END funcao;
```

# GENERATE com PROCESS

É possível gerar um código sequencial dentro do GENERATE com o uso do PROCESS.

```
GEN: for i in d'range generate
   PROCESS (X)
      BEGIN

      --Código sequencial

   END PROCESS;

   END generate GEN;
```

# Próxima aula

- Trabalho 1

# Bibliografia


- PEDRONI, Volnei A. Eletrônica Digital Moderna e VHDL. 1. ed. Campus. 2010, 648 p. ISBN 8535234659

- TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. Sistemas digitais : princípios e aplicações. 11. ed. São Paulo: Pearson Prentice Hall, 2011. ISBN 978-85-7605-922-6